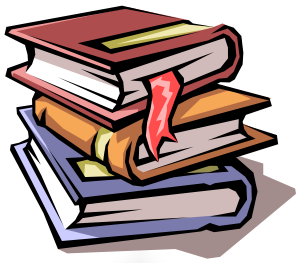
BỘ GIÁO DỤC VÀ ĐÀO TẠO

TRƯỜNG ĐẠI HỌC ...

KHOA ...



Bài giảng
Hệ quản trị cơ sở dữ liệu

GV: Chu Thị Hường

MỤC LỤC

hệ quản trị cơ sở dữ liệu quan hệ, các siêu dữ liệu bao gồm các tên của các quan hệ, tên các thuộc tính của các quan hệ, và các kiểu dữ liệu đối với các thuộc tính này.

- Bộ quản lý lưu trữ: Nhiệm vụ của bộ quản lý lưu trữ là lấy ra các thông tin được yêu cầu từ những thiết bị lưu trữ dữ liệu và thay đổi những thông tin này khi được yêu cầu bởi các mức trên nó của hệ thống.

- Bộ xử lý câu hỏi: Bộ xử lý câu hỏi điều khiển không chỉ các câu hỏi mà cả các yêu cầu thay đổi dữ liệu hay siêu dữ liệu. Nhiệm vụ của nó là tìm ra cách tốt nhất một thao tác được yêu cầu và phát ra lệnh đối với bộ quản lý lưu trữ và thực thi thao tác đó.

- Bộ quản trị giao dịch: Bộ quản trị giao dịch có trách nhiệm đảm bảo tính toàn vẹn của hệ thống. Nó phải đảm bảo rằng một số thao tác thực hiện đồng thời không cản trở mỗi thao tác khác và hệ thống không mất dữ liệu thậm chí cả khi lỗi hệ thống xảy ra.

+ Nó tương tác với bộ xử lý câu hỏi, do vậy nó phải biết dữ liệu nào được thao tác bởi các thao tác hiện thời để tránh sự đụng độ giữa các thao tác và cần thiết nó có thể làm trễ một số truy vấn nhất định hay một số thao tác cập nhật để đụng độ không thể xảy ra.

+ Nó tương tác với bộ quản lý lưu trữ bởi vì các sơ đồ đối với việc bảo vệ dữ liệu thường kéo theo việc lưu trữ một nhật ký các thay đổi đối với dữ liệu. Hơn nữa, việc sắp thứ tự các thao tác một cách thực sự được nhật ký này sẽ chứa trong một bản ghi đối với mỗi thay đổi khi gặp lỗi hệ thống, các thay đổi chưa được ghi vào đĩa có thể được thực hiện lại.

- Các kiểu thao tác đối với hệ quản trị CSDL: Tại đỉnh kiến trúc, ta thấy có 3 kiểu thao tác:

+ *Các truy vấn:* Đây là các thao tác hỏi đáp về dữ liệu được lưu trữ trong CSDL. Chúng được sinh ra theo hai cách sau:

    - Thông qua giao diện truy vấn chung. Ví dụ: Hệ quản trị CSDL quan hệ cho phép người sử dụng nhập các câu

+ Mỗi một thuộc tính nhận tập số các giá trị nhất định được gọi là domain của thuộc tính đó.

- *Một quan hệ (Relation):* Định nghĩa một cách đơn giản, một quan hệ là một bảng dữ liệu có các cột là các thuộc tính và các hàng là các bộ dữ liệu cụ thể của quan hệ.

- *Các liên kết:* Một liên kết là một sự kết hợp giữa một số thực thể (hay quan hệ). Ví dụ: Mối liên kết giữa phòng ban và nhân viên thể hiện: Một nhân viên A sẽ thuộc một phòng ban B nào đó.

+ Các liên kết một – một: đây là dạng liên kết đơn giản, liên kết trên hai thực thể là một – một, có nghĩa là mỗi thực thể trong tập thực thể này có nhiều nhất một thực thể trong tập thực thể kia kết hợp với nó và ngược lại.

+ Các liên kết một – nhiều: Trong một liên kết một – nhiều, một thực thể trong tập thực thể A được kết hợp với không hay nhiều thực thể trong tập thực thể B. Nhưng mỗi thực thể trong tập thực thể B được kết hợp với nhiều nhất một thực thể trong tập thực thể A.

+ Các liên kết nhiều – nhiều: Đây là dạng liên kết mà mỗi thực thể trong tập thực thể này có thể liên kết với không hay nhiều thực thể trong tập thực thể kia và ngược lại.

Ví dụ 1.1. Các mối liên kết giữa các thực thể:

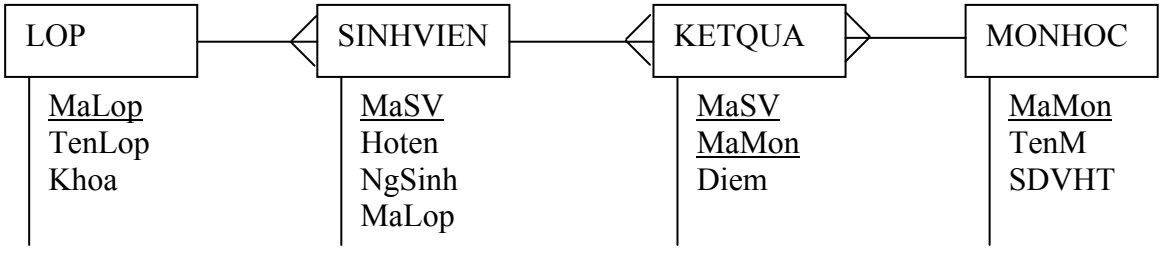
LOP(MaLop, TenLop, Khoa),

SINHVIEN(MaSV, Hoten, NgSinh, MaLop),

MONHOC(MaMon, TenM, SDVHT) và

KETQUA (MaSV, MaMon, Diem)

Ta có mối quan hệ giữa các thực thể đó là:



Quản lý khung nhìn bao gồm việc phiên dịch câu vấn tin người dùng trên dữ liệu ngoài thành dữ liệu khái niệm. Nếu câu vấn tin của người dùng được diễn tả bằng các phép toán quan hệ, câu vấn tin được áp dụng cho dữ liệu khái niệm vẫn giữ nguyên dạng này.

- *Tầng điều khiển (Control Layer):* chịu trách nhiệm điều khiển câu vấn tin bằng cách đưa thêm các vị từ toàn vẹn ngữ nghĩa và các vị từ cấp quyền.

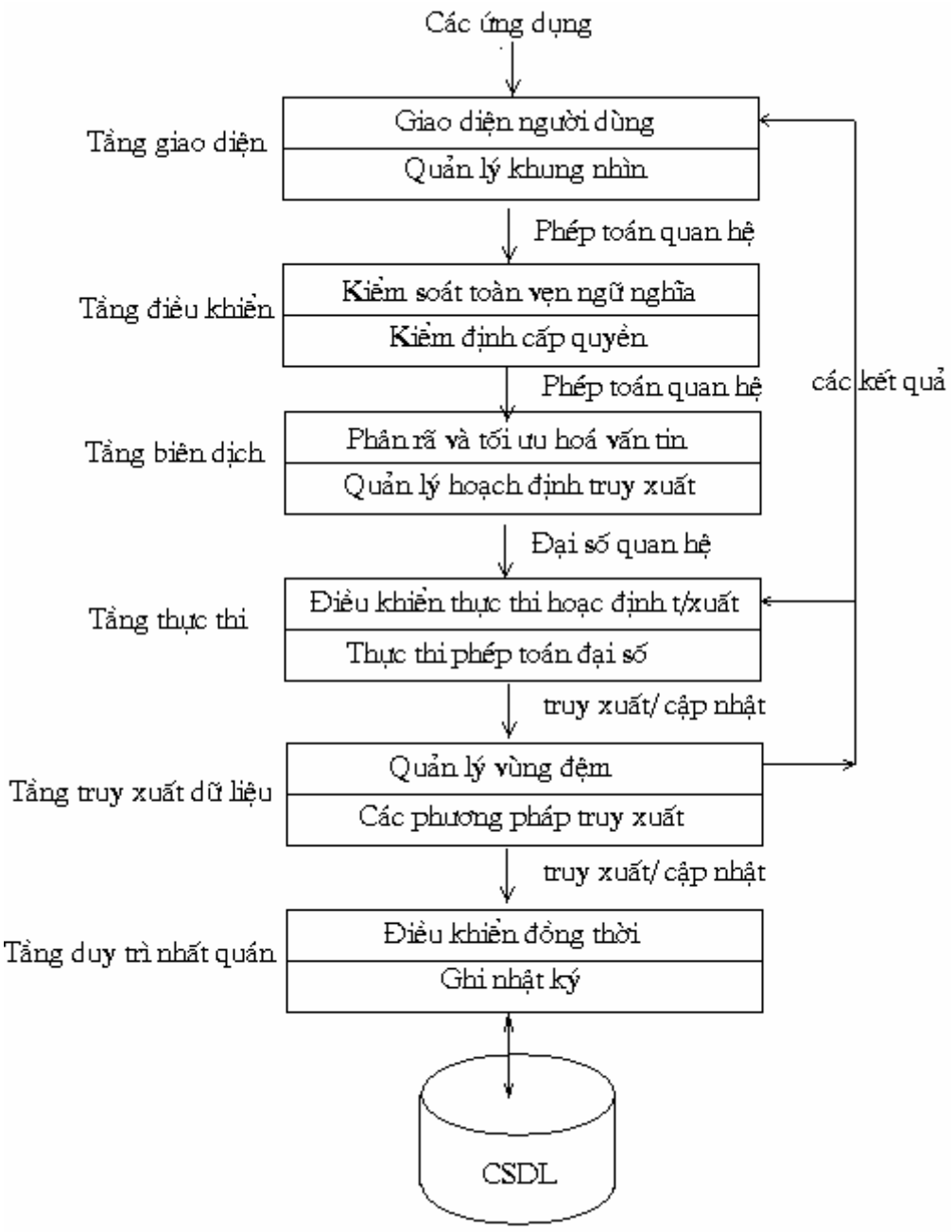
- *Tầng xử lý vấn tin (Query processing layer):* chịu trách nhiệm ánh xạ câu vấn tin thành chuỗi thao tác đã được tối ưu ở mức thấp hơn.

Tầng này liên quan đến vấn đề hiệu năng. Nó phân rã câu vấn tin thành một cây biểu thị các phép toán đại số quan hệ và thử tìm ra một thứ tự "tối ưu" cho các phép toán này. Kết xuất của tầng này là câu vấn tin được diễn tả bằng đại số quan hệ hoặc một dạng mã ở mức thấp.

- *Tầng thực thi (Execution layer):* Có trách nhiệm hướng dẫn việc thực hiện các hoạch định truy xuất, bao gồm việc quản lý giao dịch (uỷ thác, tái khởi động) và động bộ hoá các phép đại số quan hệ. Nó thông dịch các phép toán đại số quan hệ bằng cách gọi tầng truy xuất dữ liệu qua các yêu cầu truy xuất và cập nhật.

- *Tầng truy xuất dữ liệu (data access layer):* Quản lý các cấu trúc dữ liệu dùng để cài đặt các quan hệ (tập tin, chỉ mục). Nó quản lý các vùng đệm bằng cách lưu tạm các dữ liệu thường được truy xuất đến nhiều nhất. Sử dụng tầng này làm giảm thiểu việc truy xuất đến đĩa.

- *Tầng duy trì nhất quán (Consistency layer):* chịu trách nhiệm điều khiển các hoạt động đồng thời và việc ghi vào nhật ký các yêu cầu cật nhật. Tầng này cũng cho phép khôi phục lại giao dịch, hệ thống và thiết bị sau khi bị sự cố.



Hình 1.3. Các chức năng của hệ quản trị CSDL quan hệ

```
Where a.MaMT=b.MaMT;
```

+ SELECT a.SoBD, TenSV, a.Diem
From Tam a, THISINH b
Where a.SoBD=b.SoBD

Ví dụ 2.21. Liên kết nhiều bảng authors, titleauthor và title.

```
SELECT au_lname, au_fname, title, price FROM authors
    JOIN titleauthor ON authors.au_id = titleauthor.au_id
    JOIN titles ON titleauthor.title_id = titles.title_id
    ORDER BY au_lname, au_fname
```

- *Ánh xạ lồng*

Ví dụ 2.22. Cho các quan hệ:

S(S#, SNAME, STATUS, CITY)
SP(S#, P#, QTY)
P(P#, PNAME, COLOR, WEIGH)

Hãy cho biết mã và tên các hãng có bán sản phẩm màu đỏ.

```
SELECT S#, SNAME From S Where S# IN
    (SELECT S# From SP Where P# IN
        (Select P# From P Where COLOR='Red'));
```

- *Sử dụng các lượng từ: EXISTS, ANY, ALL,…*

Ví dụ 2.23. Cho các bảng trong ví dụ 2.22. Tìm các nhà cung cấp đã cung cấp ít nhất một mặt hàng nào đó.

```
SELECT * From S Where EXISTS
    (SELECT * From SP Where SP.S# =S.S#);
```

Ta có thể thay thế bằng câu lệnh:

```
SELECT * From S
    Where 0 < (SELECT Count(*) From SP Where SP.S#=S.S#);
```

Ví dụ 2.24. Tìm tên những mặt hàng có mã số mặt hàng mà mặt hàng nào đó mà hãng S1 đã bán.

```
SELECT PNAME From P Where S# = ANY
```

(SELECT P# From SP Where S#='S1');

Ví dụ 2.25. Tìm những hãng cung cấp số lượng một lần một mặt hàng nào đó > số lượng mỗi lần của các hãng cung cấp.

```
SELECT S From SP
    Where QTY>= ALL (SELECT QTY From SP ); hay
SELECT S From SP Where QTY=
    (SELECT Max(QTY) From SP );
```

trong đó ứng dụng client có thể truy cập dữ liệu thông qua các thành phần server.

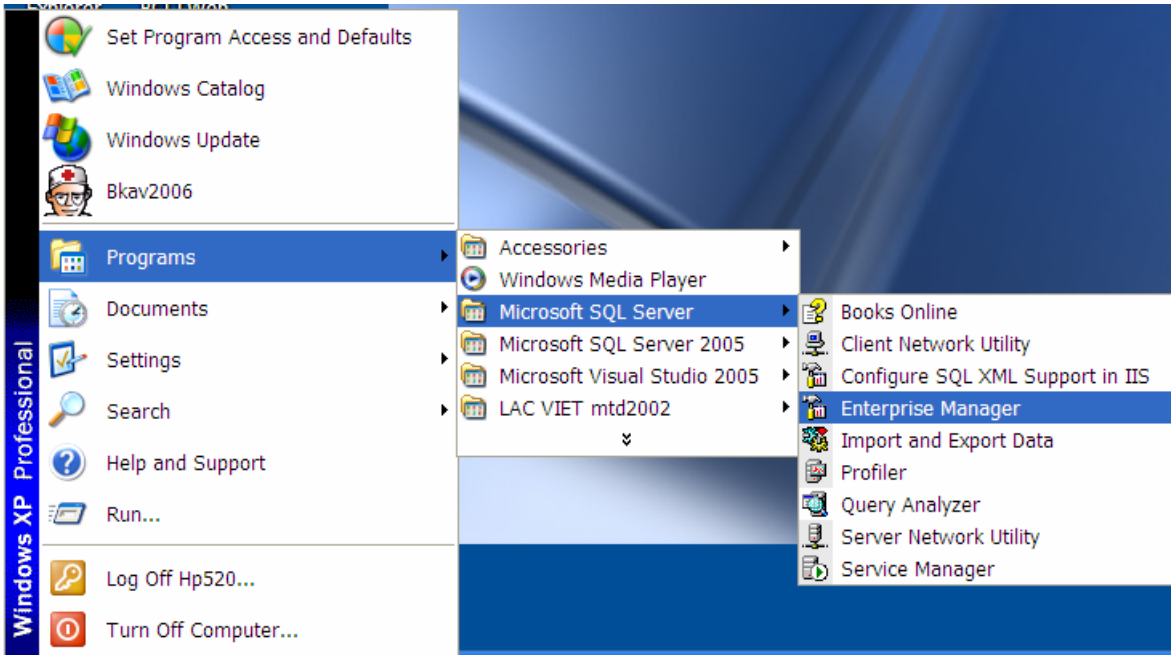
a) Các thành phần server

Các thành phần server của SQL Server 2000 thường được thực hiện như các dịch vụ Windows 32 bit. Do đó các dịch vụ của SQL Server và SQL Server Agent có thể chạy như các ứng dụng độc lập trên bất kỳ nền tảng nào được hỗ trợ hệ điều hành Windows. Các thành phần Server được mô tả trong bảng 3.1:

Bảng 3.1. Các thành phần server của SQL Server 2000

Thành phần server	***Chức năng***
Dịch vụ SQL Server	Dịch vụ MSSQLServer thực thi cỗ máy CSDL SQL Server 2000. Có một dịch vụ này cho mỗi thể hiện của SQL Server 2000.
Dịch vụ về các dịch vụ phân tích của SQL Server 2000	MSSQLServerOLAPService thực thi các dịch vụ phân tích của SQL Server 2000. Chỉ có một dịch vụ không liên quan đến số thể hiện của SQL Server 2000.
Dịch vụ SQL Server Agent	Dịch vụ SQLServerAgent thực thi các tác nhân chạy các tác vụ quản trị đã được định thời gian biểu của SQL Server 2000.
Dịch vụ tìm kiếm	Dịch vụ tìm kiếm Microsoft thực thi cỗ máy tìm kiếm toàn văn bản. Chỉ có một dịch vụ, không liên quan đến số thể hiện của SQL Server 2000.
Dịch vụ MS DTC - Distributed Transaction Coordinator	Thành phần điều phối giao dịch phân tán, quản lý các giao dịch phân tán giữa các thể hiện của SQL Server 2000. Chỉ có một dịch vụ, không liên quan đến số thể hiện của SQL Server 2000.

SQL Server 2000 cung cấp các giao diện đồ họa giúp cho người sử dụng sử dụng các dịch vụ của SQL Server 2000 (Hình 3.1. và Bảng 3.2).



Hình 3.1. Các giao diện đồ họa của SQL Server 2000.

Bảng 3.2 Các công cụ giao diện đồ họa của SQL Server 2000.

Công cụ	***Chức năng***
SQL Server Enterprise Manager	Đây là công cụ quản trị CSDL server chính, nó cung cấp một giao tiếp với người dùng Microsoft Management Console (MMC).
SQL Query Analyzer	Dùng để tạo và quản lý các đối tượng CSDL và kiểm tra các phát biểu Transact-SQL, các bó lệnh và các script một cách tương tác.
SQL Profiler	Giám sát và ghi nhận các sự kiện SQL Server 2000 đã chọn để phân tích và xem lại.
SQL Server Service Manager	Ứng dụng nằm trên thanh task bar của Windows được dùng để chạy, tạm dừng hoặc thay đổi các dịch vụ SQL Server 2000.
Client Network	Được dùng để quản lý Net-Libraries của client và

Utility	định nghĩa các bí danh server.
Server Network Utility	Dùng để quản lý Net-Libraries của server bao gồm thiết lập mã hóa SSL.

b) Các thành phần giao tiếp Client.

Người dùng truy cập SQL Server 2000 thông qua các ứng dụng client, SQL Server 2000 cung cấp hai kiểu ứng dụng client chính:

- Các ứng dụng CSDL quan hệ, là kiểu ứng dụng truyền thống dùng môi trường client/server 2 lớp. Các ứng dụng này gửi các phát biểu T-SQL đến cỗ máy CSDL quan hệ và nhận kết quả trả về như tập kết quả quan hệ.
- Các ứng dụng Internet, chúng là thành phần của nền tảng Microsoft.NET. Chúng gửi các phát biểu T-SQL hoặc các truy vấn Xpath tới cỗ máy CSDL quan hệ và nhận về kết quả dạng XML.

Các tiện ích dòng lệnh thường được sử dụng do SQL Server 2000 cung cấp cho trong bảng 3.3.

Bảng 3.3 Các tiện ích dòng lệnh của SQL Server 2000.

Tiện ích	***Chức năng***
Osql	Tiện ích này cho phép truy vấn tương tác một thể hiện của SQL Server 2000 bằng các phát biểu T-SQL, các thủ tục và các script.
Scm (Server Control Manager)	Dùng để chạy, dừng, tạm dừng, cài đặt, xóa hoặc thay đổi các dịch vụ SQL Server 2000.
Sqldiag	Tiện ích này thu thập và lưu trữ các thông tin chuẩn đoán để xử lý và đơn giản hóa thông tin thu thập bởi dịch vụ hỗ trợ sản phẩm Microsoft.
Bcp	Tiện ích này sao chép dữ liệu giữa một thể hiện của SQL Server 2000 và tập tin dữ liệu theo định dạng của người dùng.

chỉ có các phiên bản Enterprise, Standard, và Workgroup được cài đặt và sử dụng trong môi trường server phục vụ cho hoạt động thực tế.

+ ***SQL Server 2005 Enterprise Edition (32-bit và 64-bit)***

  Enterprise Edition được sử dụng trong các doanh nghiệp, tổ chức có các mức yêu cầu xử lý giao dịch trực tuyến trên diện rộng (online transaction processing - OLTP), khả năng phân tích dữ liệu phức tạp cao, hệ thống kho dữ liệu (data warehousing systems) và web sites. Enterprise Edition phù hợp cho các tổ chức lớn và các yêu cầu phức tạp.

+ ***SQL Server 2005 Standard Edition (32-bit và 64-bit)***

  Standard Edition là phiên bản phục vụ cho việc quản trị và phân tích dữ liệu phù hợp cho các doanh nghiệp, tổ chức vừa và nhỏ. Nó bao gồm các giải pháp cần thiết cho thương mại điện tử (e-commerce), kho dữ liệu (data warehousing) và dòng doanh nghiệp (line-of-business).

+ ***SQL Server 2005 Workgroup Edition (32-bit only)***

  Workgroup Edition là giải pháp quản trị dữ liệu phù hợp cho các doanh nghiệp, tổ chức nhỏ chỉ cần một cơ sở dữ liệu không giới hạn kích thước hoặc số người sử dụng. Workgroup Edition là lý tưởng cho các mức cơ sở dữ liệu tin cậy, mạnh mẽ và dễ quản trị.

+ ***SQL Server 2005 Developer Edition (32-bit và 64-bit)***

  Developer Edition có tất cả các tính năng của phiên bản SQL Server 2005 Enterprise Edition, nhưng nó chỉ là phiên bản sử dụng cho phát triển và kiểm tra ứng dụng. Phiên bản này phù hợp cho các cá nhân, tổ chức xây dựng và kiểm tra ứng dụng.

+ ***SQL Server 2005 Express Edition (32-bit only)***

  SQL Server Express, dễ sử dụng và quản trị cơ sở dữ liệu đơn giản. Được tích hợp với Microsoft Visual Studio 2005, SQL Server Express trở nên dễ dàng để phát triển các ứng dụng dữ liệu giầu khả năng, an toàn trong lưu trữ, và nhanh chóng triển khai.

SQL Server Express là phiên bản miễn phí, có thể dùng như một cơ sở dữ liệu máy khách hoặc cơ sở dữ liệu máy chủ đơn giản. SQL Server Express là lựa chọn tốt cho những người dùng chỉ cần một phiên bản SQL Server 2005 nhỏ gọn, dùng trên máy chủ có cấu hình thấp, những nhà phát triển ứng dụng không chuyên hay những người yêu thích xây dựng các ứng dụng nhỏ.

b) Các thành phần Server của SQL Server 2005:

Các thành phần server của SQL Server 2005 được cho trong bảng 3.4.

Bảng 3.4 Các thành phần server của SQL Server 2005.

Thành phần Server	***Chức năng***
SQL Server Database	Cỗ máy cơ sở dữ liệu bao gồm Database Engine, lõi dịch vụ cho việc lưu trữ, xử lý và bảo mật dữ liệu, sao lưu và đồng bộ (Replication), tìm kiếm toàn văn (Full-Text Search), và các công cụ cho việc quản trị dữ liệu quan hệ và XML.
Analysis Services	Analysis Services bao gồm các công cụ cho việc tạo và quản lý tiến trình phân tích trự tuyến (online analytical processing - OLAP) và các ứng dụng khai thác dữ liệu.
Reporting Services	Reporting Services bao gồm các thành phần server và client cho việc tạo, quản lý và triển khai các báo cáo. Reporting Services cũng là nền tảng cho việc phát triển và xây dựng các ứng dụng báo cáo.
Notification Services	Dịch vụ thông báo Notification Services là nền tảng cho sự phát triển và triển khai các ứng dụng tạo và gửi thông báo. Notification Services có thể gửi thông báo theo địch thời đến hàng ngàn người đăng ký sử dụng nhiều loại thiết bị khác nhau.

Integration Services	Integration Services là một tập hợp các công cụ đồ họa và các đối tượng lập trình cho việc di chuyển, sao chép và chuyển đổi dữ liệu.

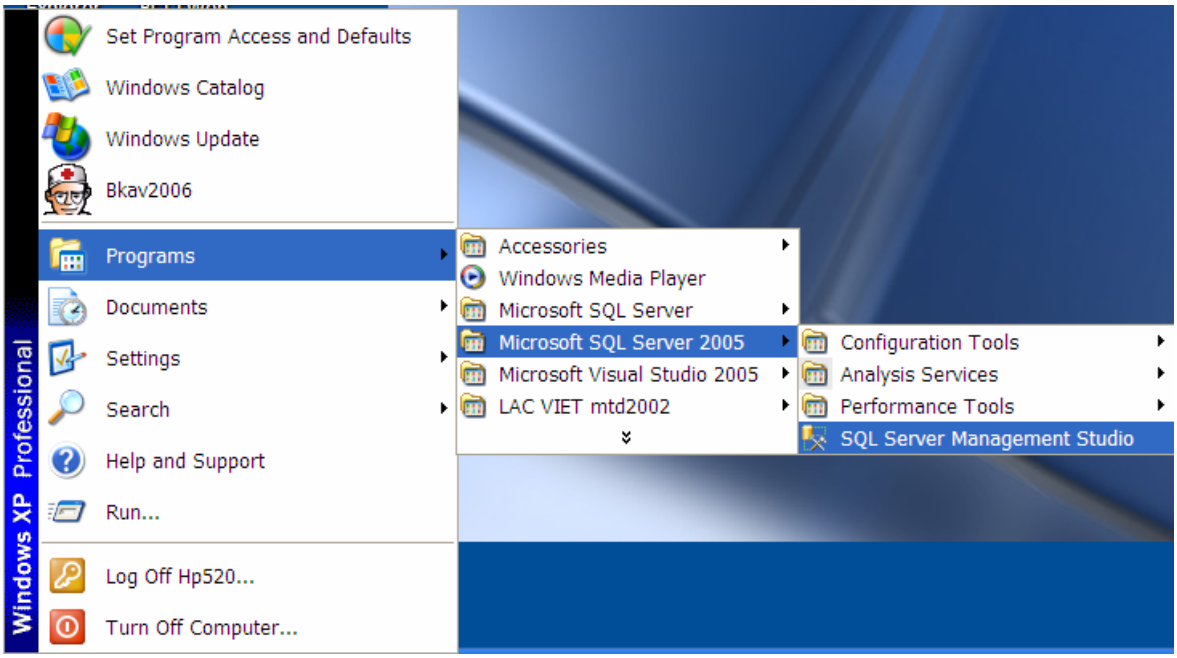
c) Các thành phần Client

Bảng 3.5 Các thành phần client của SQL Server 2005.

Thành phần Client	***Chức năng***
Connectivity Components	Là các thành phần cho việc truyền thông giữa clients và servers, và các thư viện mạng như DB-Library, ODBC, and OLE DB.

c) Các công cụ đồ họa

Các công cụ giao diện đồ họa giúp cho việc truy xuất và quản trị SQL Server được thay đổi khá nhiều so với các phiên bản trước đó, các công cụ quản trị đó được cho trong bảng 3.6. và hình 3.2.



Hình 3.2. Các giao diện đồ họa của SQL Server 2005.

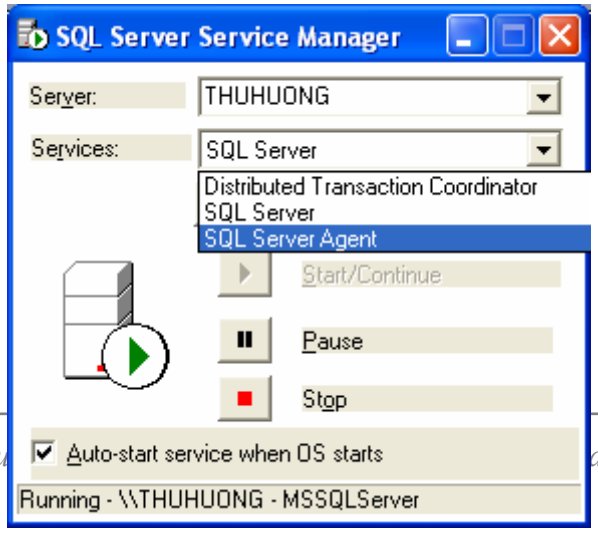
Bảng 3.5 Các công cụ quản trị trên SQL Server 2005.

tập tin CSDL, xử lý các phát biểu Transaction – SQL, cấp phát tài nguyên giữa các kết nối người dùng đồng thời, đảm bảo tính nhất quán dữ liệu,.v.v…

- *Dịch vụ SQL Server Agent:* Dịch vụ này hỗ trợ lập các chương trình, thực thi tác vụ, cảnh báo, thông báo và kế hoạch bảo trì CSDL. Nó cho phép ta thực hiện tự động hóa các tác vụ bảo trì CSDL.
- *Dịch vụ Distributed Transaction Coordinator:* Là trình quản lý giao dịch cung cấp các khả năng bao gồm nhiều nguồn dữ liệu khác nhau kể cả các CSDL từ xa trong các giao dịch ứng dụng.

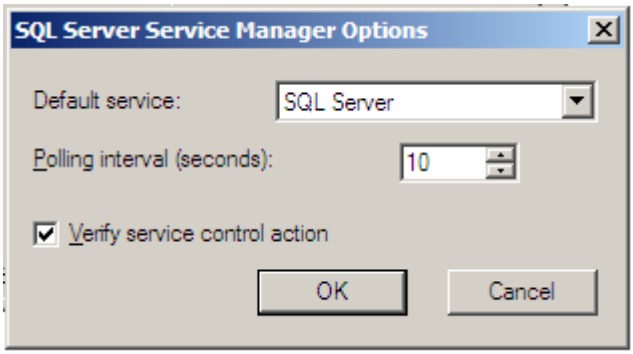
Khởi động hoặc dừng các dịch vụ SQL Server sử dụng trình SQL Server Service Manager thực hiện theo các bước sau:

1. Vào *start/Programs/Microsoft SQL Server/Service Manager*; Hoặc dưới góc phải của màn hình trên thanh task bar hệ thống của windows, double–click vào biểu tượng Service Manager. Khi đó cửa sổ SQL Server Service Manager xuất hiện như hình 3.3.
2. Trong danh sách Server chọn tên server và danh sách Service chọn dịch vụ SQL Server.
3. Click nút *Start/Continue* để khởi chạy dịch vụ; Click nút *Pause* để tạm dừng dịch vụ, tạm dừng dịch vụ để ngăn chặn người dùng đăng nhập vào SQL Server và có thời gian cho người dùng đang kết nối có thời gian hoàn tất các tác vụ và thoát khỏi SQL Server trước khi ta đóng SQL Server; Click vào nút *Stop* để dừng dịch vụ.



Hình 3.3. Ứng dụng SQL Server Service Manager

4. Trong khi chạy *Service Manager*, trạng thái hiển thị của các dịch vụ được mặc định là 5giây, để thay đổi thay đổi thời gian cập nhật, ta click chuột vào biểu tượng ở góc trên bên trái của SQL Server Service Manager, chọn Options xuất hiện hộp thoại SQL Server Service Manager Options nhập vào khoảng thời gian kiểm soát vòng mới cho các dịch vụ, chẳng hạn là 10 (Hình 3.4).



Hình 3.4. SQL Server Service Manager Options

b) Sử dụng Enterprise Manager

Enterprise Manager là thành phần Microsoft Management Console (MMC). MMC là ứng dụng trung tâm dùng để quản lý tất cả các giao tiếp của hệ thống. Nó cho phép thực hiện các tác vụ sau:

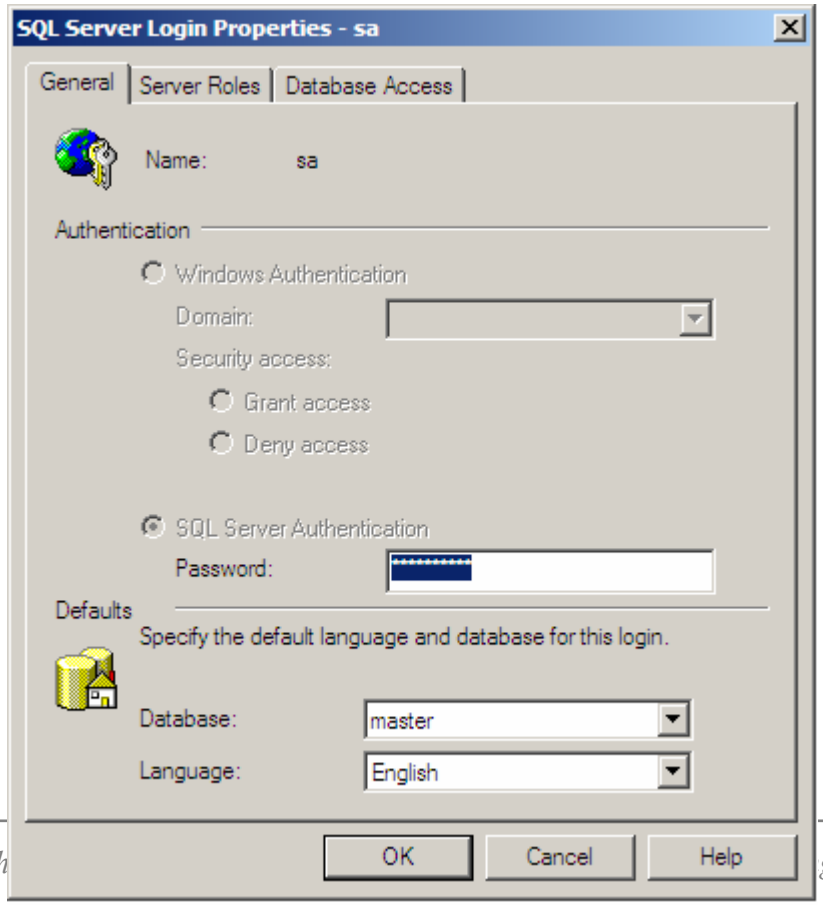
+ Khởi động, dừng, tạm dừng Server.
+ Đăng ký server,
+ Cấu hình server cục bộ và từ xa
+ Cấu hình và quản lý các thể hiện của server,
+ Thiết lập bảo mật đăng nhập, thêm người dùng, người quản trị hệ thống.
+ Gán mật khẩu cho người quản trị hệ thống,

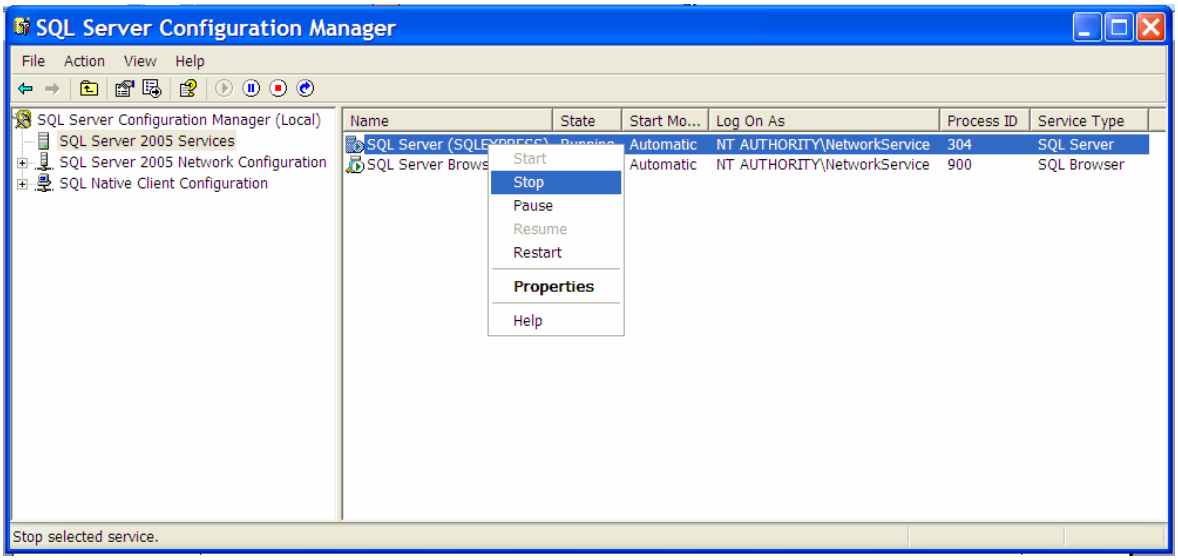
+ Tạo và lập thời gian biểu thực thi công việc,
+ Thiết lập và quản lý CSDL, bảng, chỉ mục, view, stored proceduce, trigger, …
+ Quản lý các dịch vụ SQL server khác,…

** Thay đổi mật khẩu mặc định*

Tất cả các SQL server đều có một tài khoản quản trị mặc định sẵn là sa (system administrator). Lúc mới cài tài khoản này chưa được gán mật khẩu. Để đảm bảo mức bảo mật cao nhất cho SQL server ta phải gán cho tài khoản sa một mật khẩu. Khi gán mật khẩu ta thực hiện theo các bước sau:

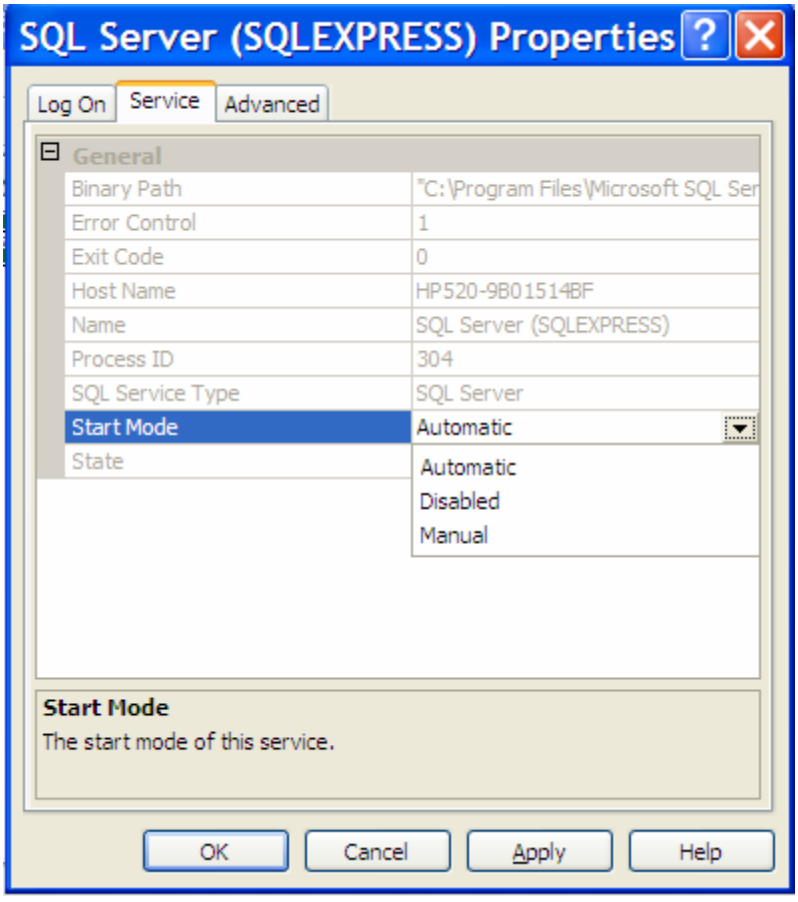
1. Trong *Enterprise Manager* chọn tên server
2. Chọn *Security/Logins* để hiển thị tất cả các tài khoản người dùng.
3. Right click chuột lên tài khoản *sa*, và chọn *Properties*, xuất hiện cửa sổ SQL Server Login Properties như hình 3.5.
4. Nhập mật khẩu mới vào hộp Password sau đó click OK để hiểm hị hộp thoại Confirm Password.
5. Trong hộp thoại Confirm Password nhập lại mật khẩu trên để xác nhận lại mật khẩu và chọn OK.





Hình 3.6. Cửa sổ SQL Server Configuration Manager.

3. Muốn khởi chạy tự động dịch vụ SQL Server, trong cửa sổ trên chọn Properties. Trong hộp thoại SQL Server Properties, chọn tab Service và chọn thuộc tính Start Mode là Automatic.



Hình 3.7. Cửa sổ SQL Server Properties.

b) Sử dụng SQL Server Management Studio

Microsoft SQL Server Management Studio là môi trường tích hợp cho việc truy cập, cấu hình, quản lý, quản trị và phát triển tất cả các thành phần của SQL Server. SQL Server Management Studio kết hợp một nhóm rộng lớn các công cụ đồ họa giầu trình biên tập (script editors) cung cấp các truy xuất đến SQL Server để phát triển và quản trị tất cả các mức kỹ năng. Và có thể dùng nó để quản trị SQL Server 2000.

SQL Server Management Studio kết hợp các tính năng của Enterprise Manager, Query Analyzer, và Analysis Manager trong phiên bản trước. Thêm vào đó, SQL Server Management Studio làm việc với tất cả các thành phần cuả SQL Server như là: Reporting Services, Integration Services, SQL Server Mobile, và Notification Services.

Microsoft SQL Server Management Studio bao gồm các tính năng tổng quát sau:

+ Cung cấp hầu hết các tác vụ quản trị cho SQL Server 2005 và SQL Server 2000.
+ Là môi trường đơn, tích hợp cho việc quản trị và trao quyền SQL Server Database Engine.
+ Các hộp thoại mới cho việc quản lý các đối tượng trong SQL Server Database Engine, Analysis Services, Reporting Services, Notification Services, và SQL Server Mobile, cho phép ta thực thi các hành động ngay lập tức, gửi chúng tới Code Editor, hoặc tạo tập lệnh cho lần thực thi tiếp theo.
+ Các hộp thoại cho phép truy cập đến nhiều điều khiển trong khi hộp thoại đó đang được mở.
+ Lập lịch cho phép ta thực thi các hành động của các hộp thoại quản trị.
+ Export và import đăng ký server SQL Server Management Studio từ một môi trường Management Studio này đến môi trường khác.

+ Save hoặc in file XML Showplan hoặc Deadlock files được sinh bởi SQL Server Profiler, xem lại, hoặc gửi chúng tới administrators để phân tích.v.v…

Để truy cập vào SQL Server Management Studio:

1. Để chạy SQL Server Management Studio, trên thanh taskbar, click *Start/ Programs/Microsoft SQL Server 2005*, và sau đó click SQL Server Management Studio.
2. Khi khởi chạy SQL Server Management Studio, một hộp thoại "Connect to Server" (Hình 3.8) xuất hiện. Ta có thể chọn một thể hiện của Server để kết nối hoặc không chọn một thể hiện nào cả.

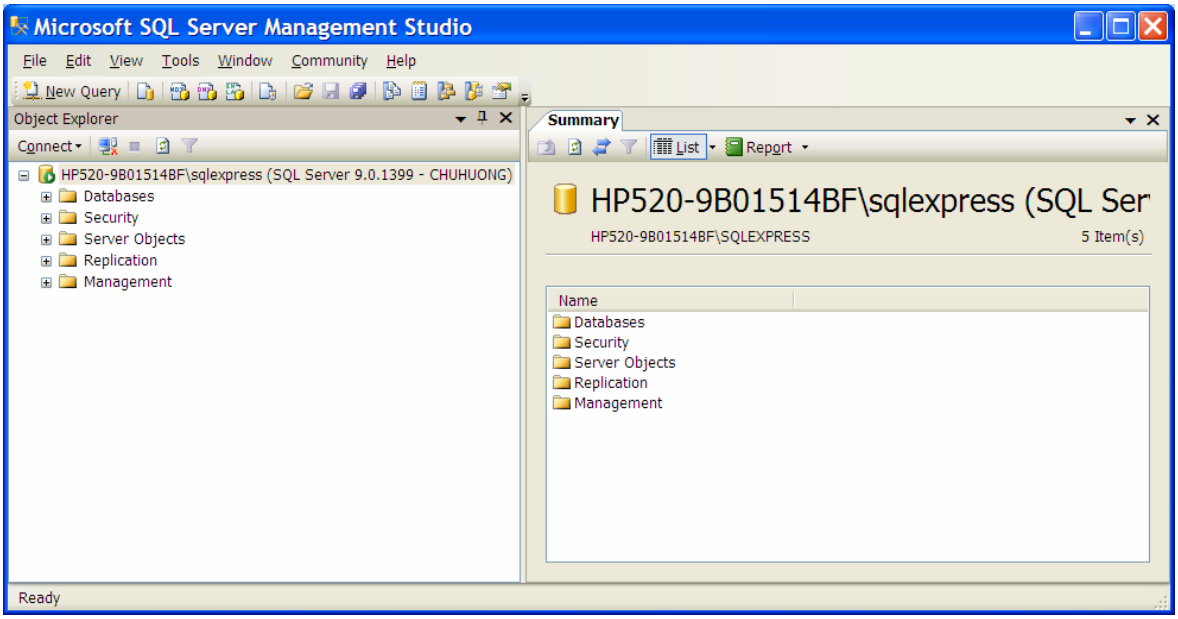
Hình 3.8. Cửa sổ Connect to Server.

+ Server type: Chọn Database Engine để kết nối đến cỗ máy cơ sở dữ liệu.
+ Server name: chọn hoặc nhập tên server
+ Authentication: Chọn chế độ xác thực là Windows Authentication hoặc SQL Server Authentication, nếu chọn SQL Server Authentication thì ta phải cung cấp thông tin cho các mục Login và Password.
+ Login: Nhập tên đăng nhập

+ Password: Mật khẩu của tên đăng nhập

+ Remember password: Tùy chọn được chọn để là đăng nhập sau không phải đánh mật khẩu.

Sau hộp thoại "Connect to Server "cho vào cửa sổ SQL Server Management Studio (Hình 3.9)



Hình 3.9. Cửa sổ SQL Server Management Studio.

** Đăng ký Server*

Cửa sổ Registered Servers trong SQL Server Management Studio chứa danh sách các thể hiện danh sách các server đã đăng ký. Ta sử dụng cửa sổ Registered Servers nhằm mục đích:

+ Lưu thông tin kết nối cho các thể hiện của SQL Server trên mạng.

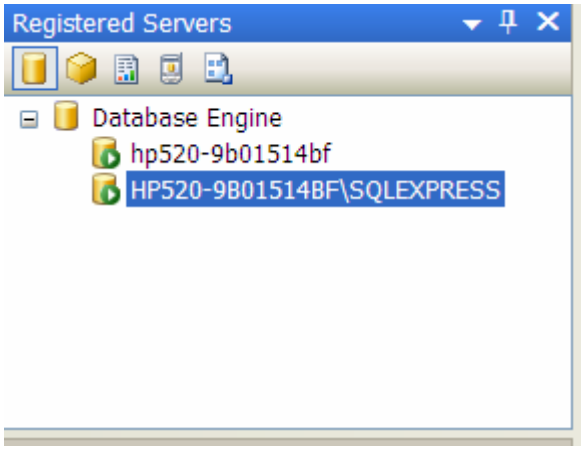
+ Hiển thị một thể hiện đang chạy hay không chạy

+ Kết nối tới một thể hiện trong cửa sổ Object Explorer hoặc Query Editor

+ Nhóm các server

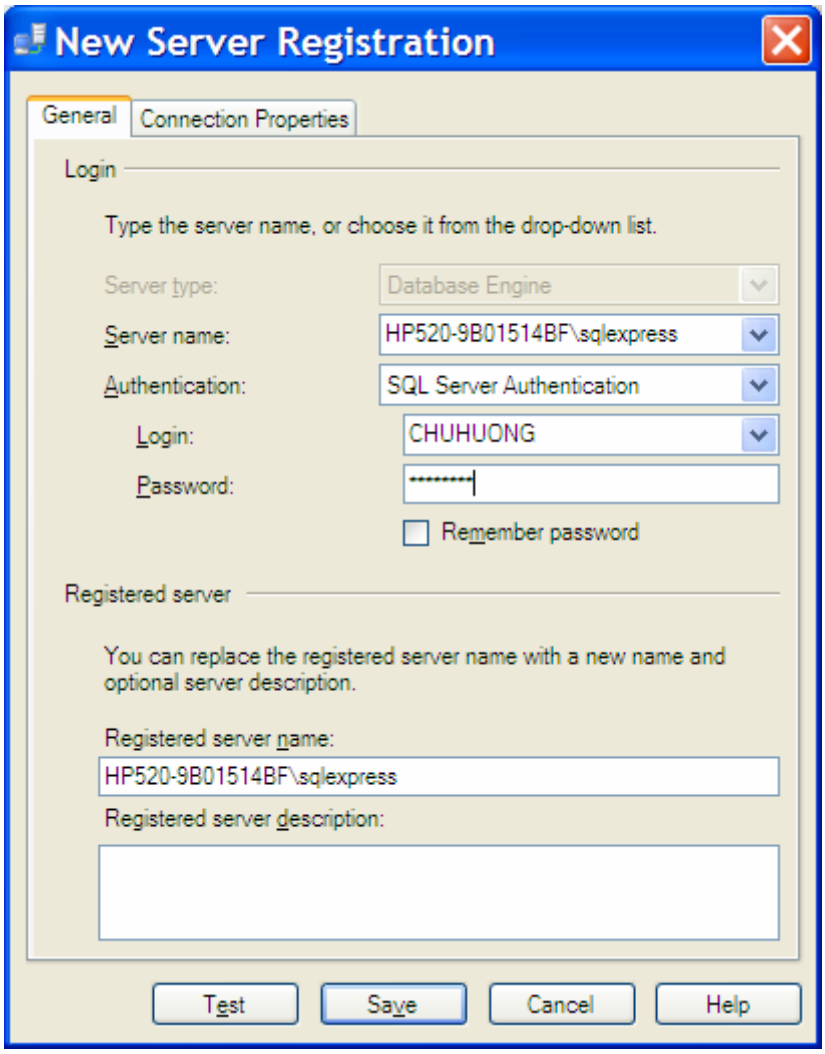
Để đăng ký một thể hiện của SQL Server ta thực hiện các bước sau

1. Vào View\chọn Registered Servers để xuất hiện cửa sổ Registered Servers hình 3.10.
2. Chọn biểu tượng kiểu thể hiện muốn đăng ký, theo thứ tự từ trái sang phải là Database Engine, Analysis Services, Reporting Services, SQL Server Mobile và Integration Services. Ta chọn Database Engine.



Hình 3.10. Cửa sổ Registered Servers.

3. Right click lên biểu tượng hoặc vùng trống chọn New\Server Registration xuất hiện cửa sổ New Server Registration (Hình 3.11)



Hình 3.11. Cửa sổ New Server Registration.

Trong cửa sổ này ta thực hiện các lựa chọn sau:

+ Server Name: Nhập hoặc chọn tên thể hiện của SQL Server từ hộp danh sách Server Name, có thể chọn <Browser for more …> để tự tìm các thể hiện của SQL Server trên mạng.
+ Authentication: Chọn chế độ xác thực, tương tự như trong cửa sổ Connect to Server.
+ Registered server name: Tên của thể hiện trên cửa sổ Registered Servers.
+ Nút Test: Dùng để kiểm tra xem kết nối có thành công hay không?

+ Nút Save: Dùng để lưu thông tin đăng ký.

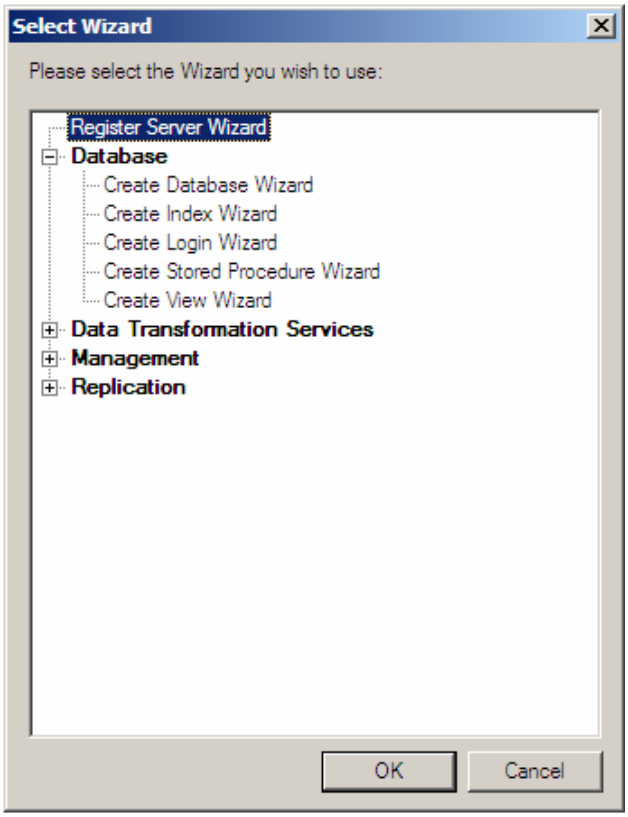
Để kết nối tới thể hiện của SQL Server và hiển thị nó trong cửa sổ Object Explorer hoặc tạo truy vấn mới ta thực hiện các bước sau:

1. Trong cửa sổ Registered Servers chọn biểu tượng kiểu thể hiện muốn kết nối. Ta chọn Database Engine.
2. Right click lên thể hiện muốn kết nối, chọn Connect và chọn:
   + New Query: để tạo một truy vấn đã kết nối tới thể hiện này.
   + Object Explorer: Để hiển thị nó trong cửa sổ Object Explorer.

** Thay đổi mật khẩu mặc định*

Thay đổi mật khẩu cho tài khoản đăng nhập sa ta thực hiện theo các bước như sau:

1. Trong cửa sổ Object Explorer của SQL Server Management Studio chọn tên thể hiện của server
2. Chọn *Security/Logins* để hiển thị tất cả các tài khoản người dùng.
3. Right click chuột lên tài khoản *sa*, và chọn *Properties*, xuất hiện cửa sổ Login Properties như hình 3.12.

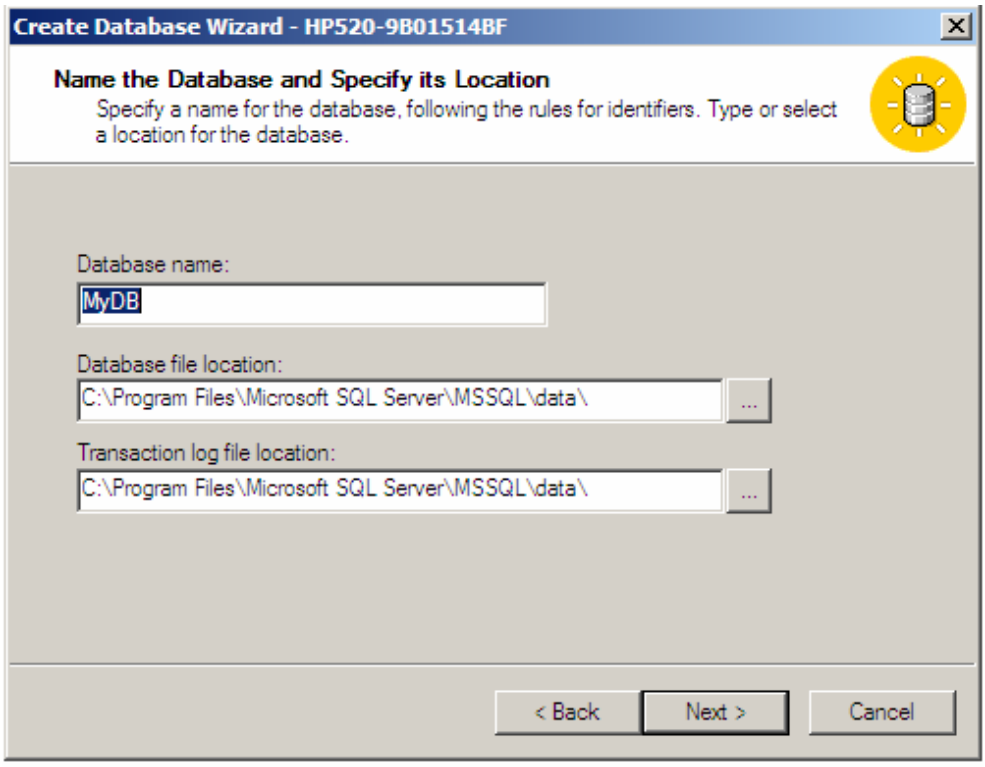


Hình 3.13. Cửa sổ Select Wizard

Hình 3.14. Cửa sổ Welcome to Create Database Wizard

2. Chọn *Create Database Wizard* và chọn *OK* → Xuất hiện cửa sổ Cửa sổ Welcome to Create Database Wizard hình 3.14.

3. Click *Next* → Xuất hiện cửa sổ 3.15. *Name the Database And Specify its Location*. Nhập tên Database và vị trí lưu các file cơ sở dữ liệu và file log → và chọn Next.



Hình 3.15. Cửa sổ Name the Database And Specify its Location

4. Màn hình Name the Database files (hình 3.16), nhập tên tập tin dữ liệu chính (có thể dùng tên mặc) và kích thước khởi tạo cho tập tin đó. Sau đó chọn Next → Xuất hiện cửa sổ 3.17.

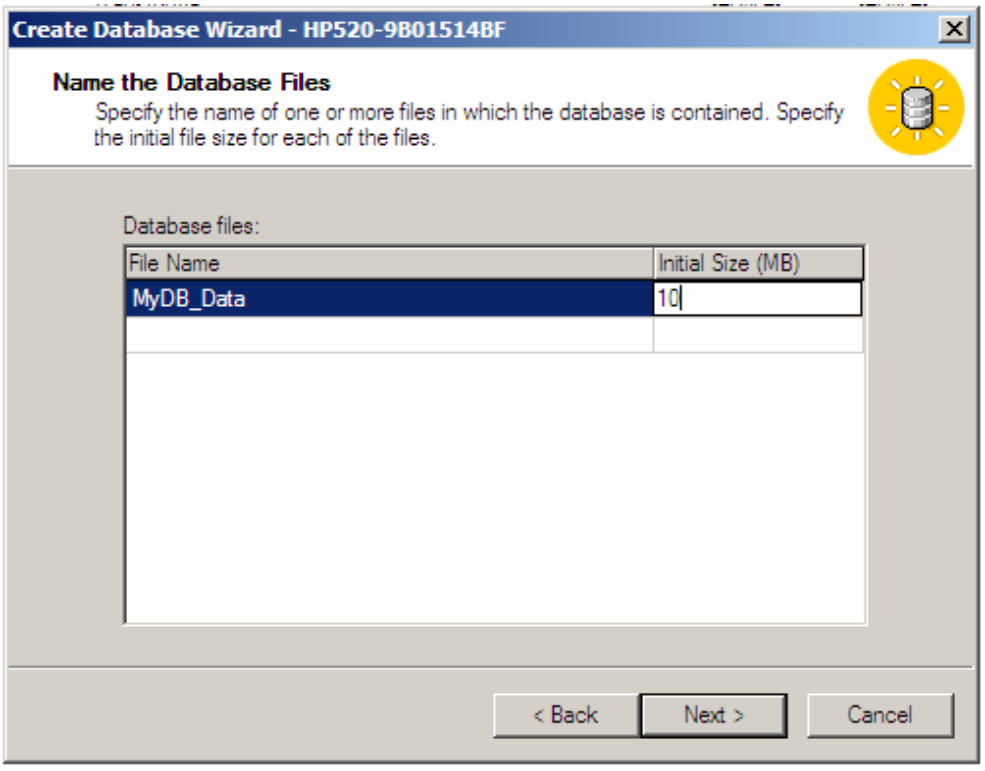
5. Theo tùy chọn mặc định: Tập tin CSDL tự động gia tăng và gia tăng 10%, không giới hạn dung lượng tối đa. Ta có thể thay đổi các tham số đó thông qua các tùy chọn:

   + Do not automatically grow the data file: Không tự động gia tăng kích thước của file dữ liệu.
   + Automatically grow the data file: Tự động gia tăng kích thước của file dữ liệu.

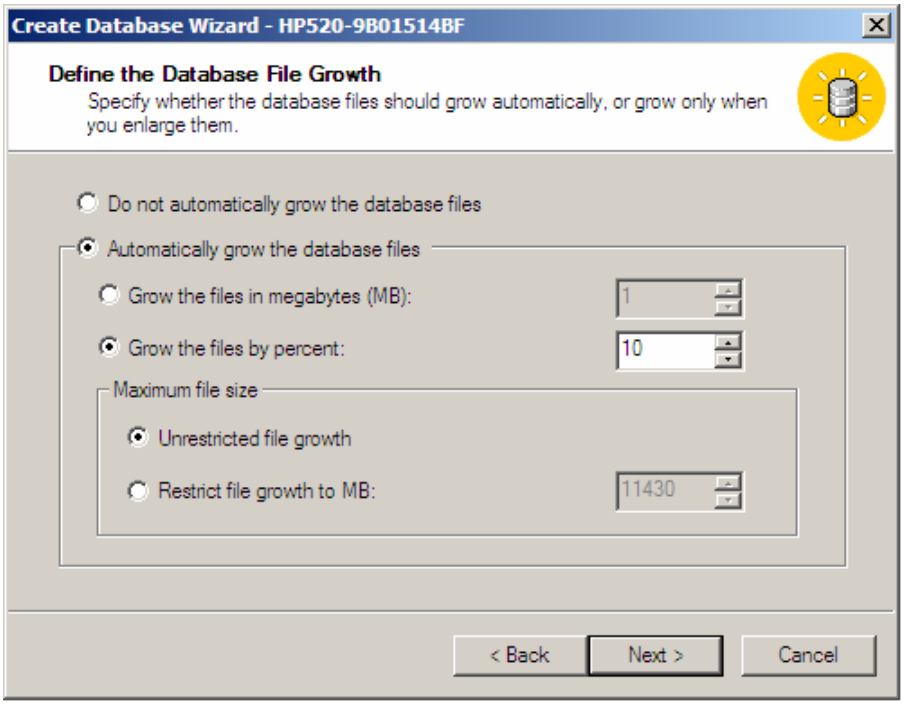
+ Grow the file in megabytes (MB): Gia tăng theo MB.
+ Grow the file by percent: Gia tăng theo phần trăm.
+ Unrestricted file growth: Gia tăng không giới hạn cận trên của file.
+ Restricted file growth to MB: Giới hạn cận trên của file theo MB.

Chọn Next. Xuất hiện của sổ Name the Transaction Log Files hình 3.18.

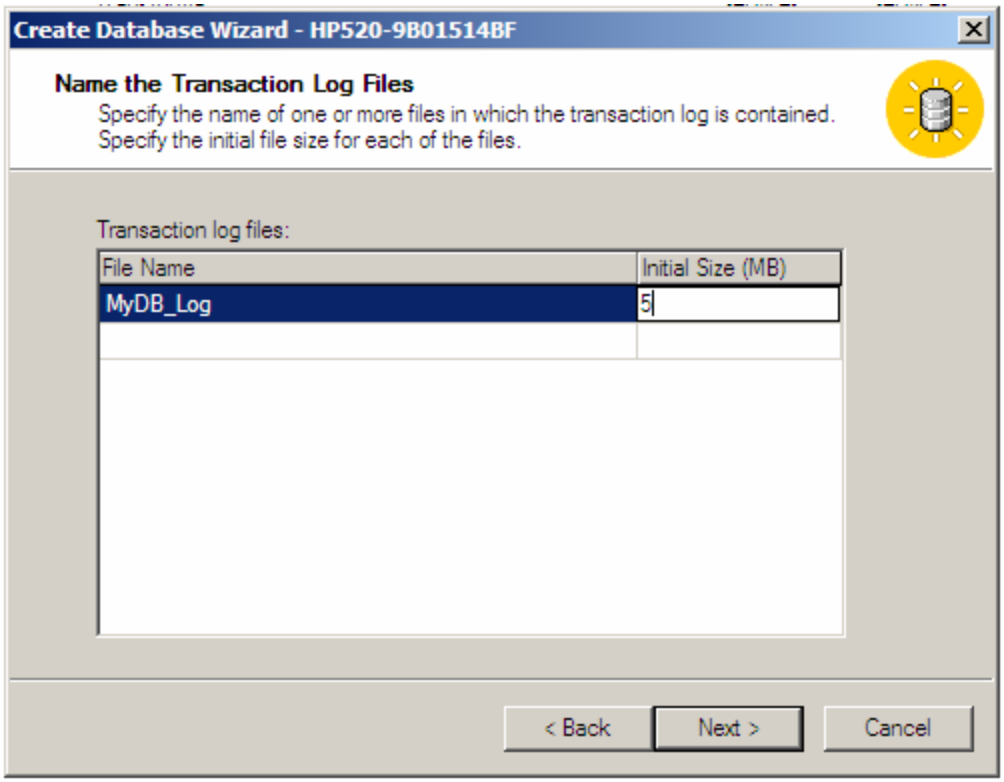
6. Nhập tên tập tin bản ghi giao dịch và kích thước khởi tạo chúng. Chọn Next.
7. Xuất hiện cửa sổ Define the Transaction Log File Growth. Các thông tin điền tương tự như cửa sổ Define the Database file Growth. Chọn Next để tiếp tục.
8. Chọn Finish để kết thúc.



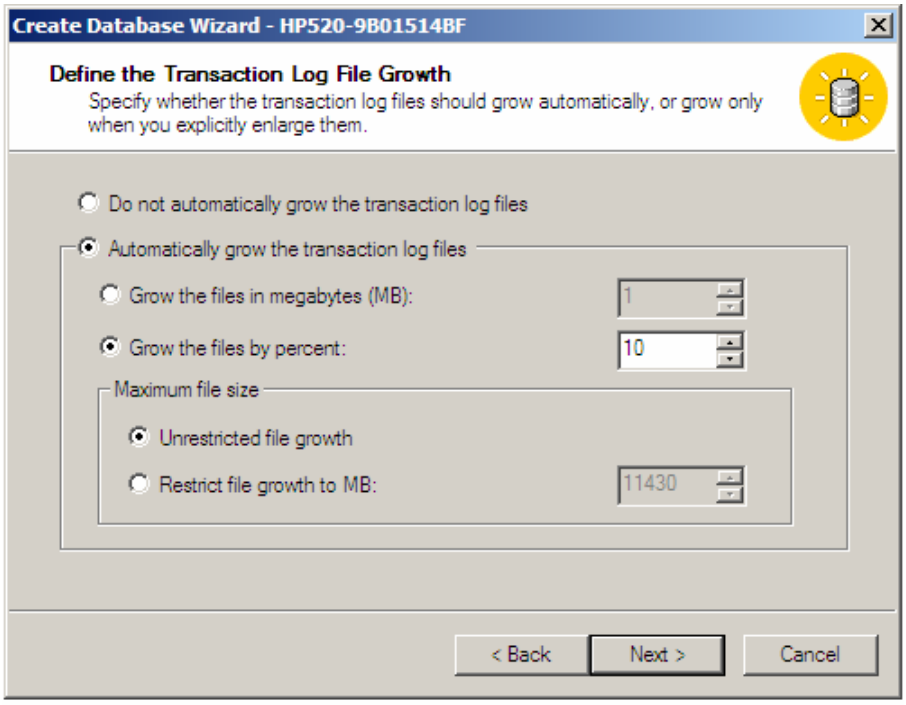
Hình 3.16. Cửa sổ Name the Database files



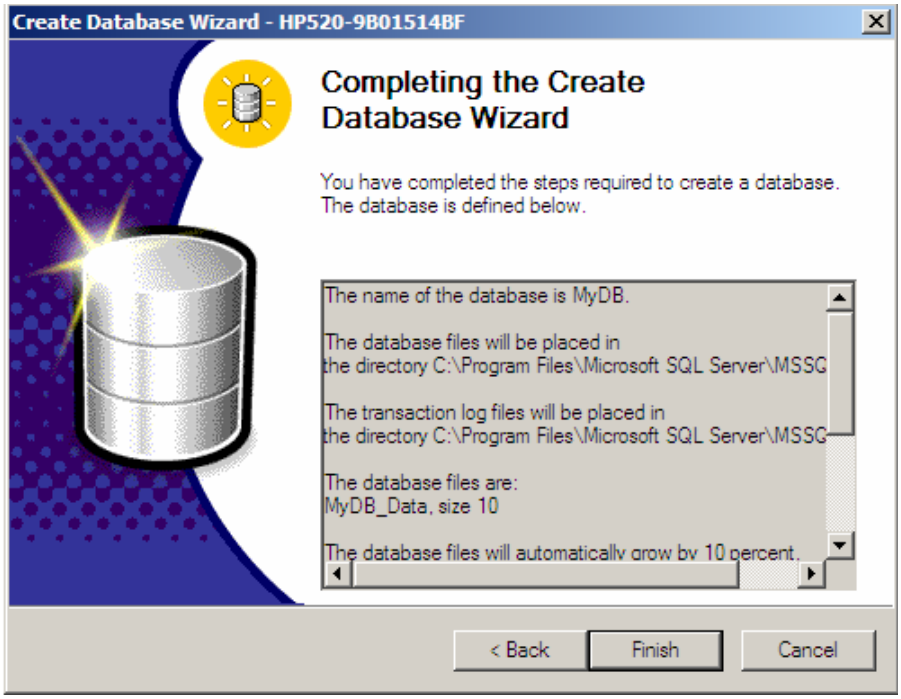
Hình 3.17. Cửa sổ Define the Database file Growth



Hình 3.18. Cửa sổ Name the Transaction Log Files



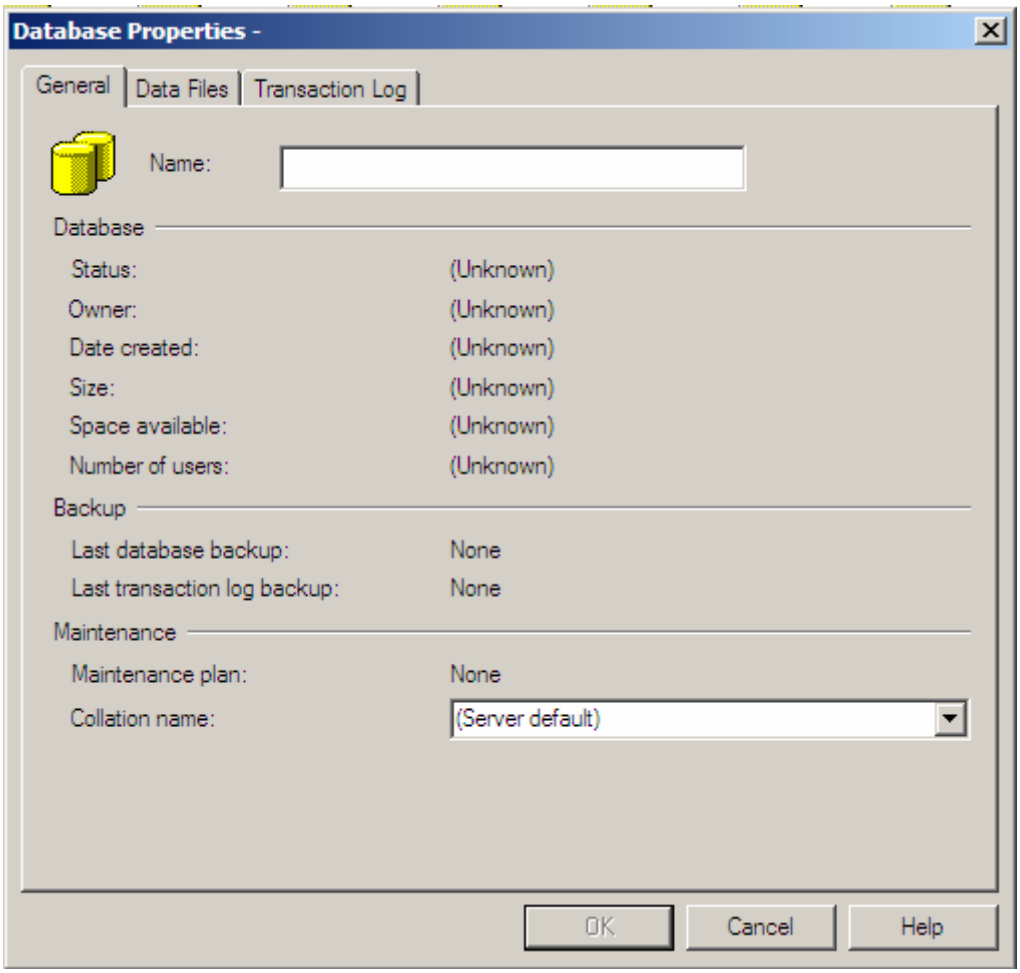
Hình 3.19. Cửa sổ Define the Transaction Log File Growth



Hình 3.20. Cửa sổ hoàn thành tạo Database.

**** Sử dụng SQL Server Enterprise Manager trên SQL Server 2000***

Khởi động SQL Server Enterprise Manager. Chọn tên *Server* và chọn *Database*. Rigth click lên mục Database và chọn New Database. Xuất hiện cửa sổ Database Properties (hình 3.21).



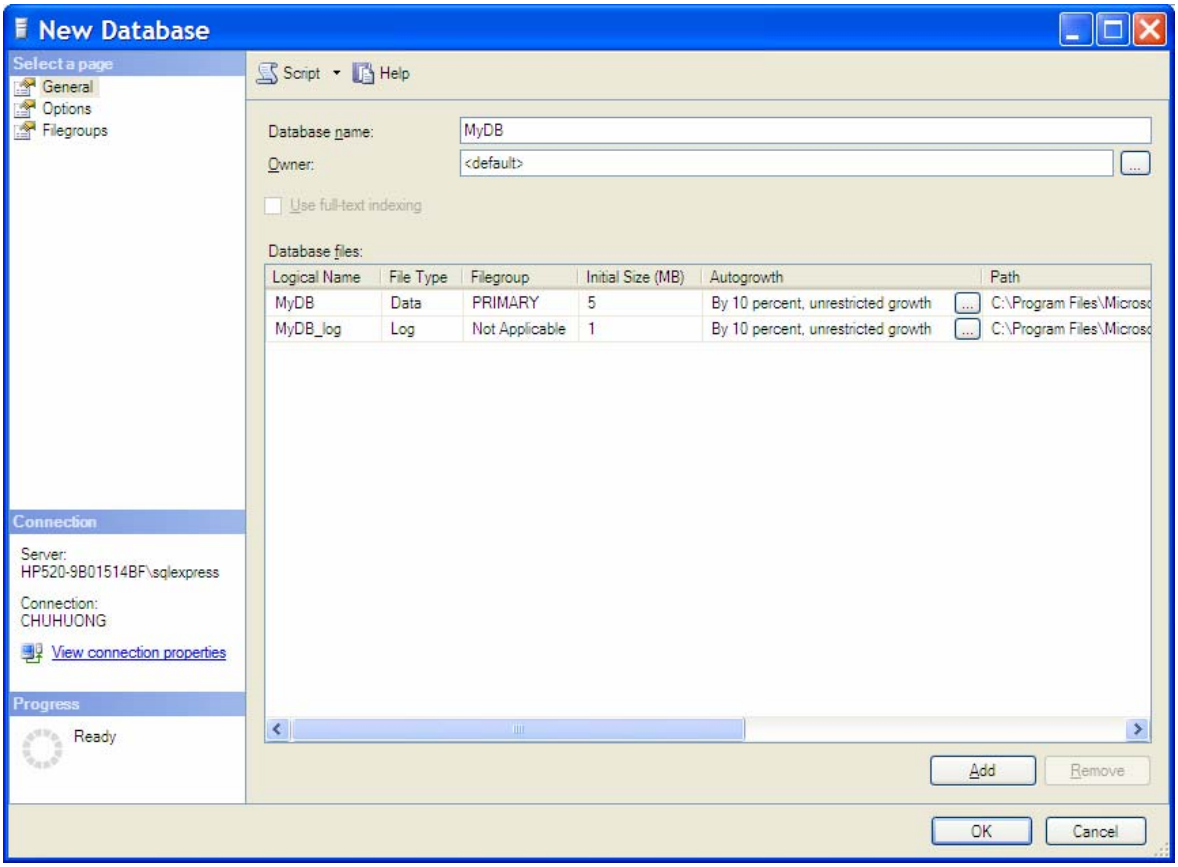
Hình 3.21. Cửa sổ Database Properties.

+ *Tab General:* Nhập tên Database chẳng hạn QLDiemSV
+ *Tab Data files:* Đặt tên các file dữ liệu, vị trí lưu trữ chúng và các tham số growth. Chú ý, SQL Server Enterprise Manager tự động tạo tập tin dữ liệu chính QLDiemSV_Data (có thể thay đổi được tên) thuộc nhóm PRIMARY không thể thay đổi nhóm. Ta có thể tạo các tập tin dữ liệu phụ trên các nhóm khác nhau.
+ *Tab Transaction log:* Tương tự như trên nhưng đối với tập tin ghi các bản ghi giao dịch.

**** Sử dụng SQL Server Management Studio của SQL Server 2005:***

Như đã giới thiệu ở trên, SQL Server Management Studio là công cụ tích hợp SQL Server Enterprise Manager của phiên bản 2000 nên ta có thể sử dụng để tạo database tương tự như phiên bản 2000. Cách tạo như sau:

1. Mở rộng các đối tượng trên thể hiện của SQL trong cửa sổ Object Explorer muốn tạo Database.
2. Right click lên mục Database\ chọn New Database xuất hiện cửa sổ New Database (Hình 3.22).

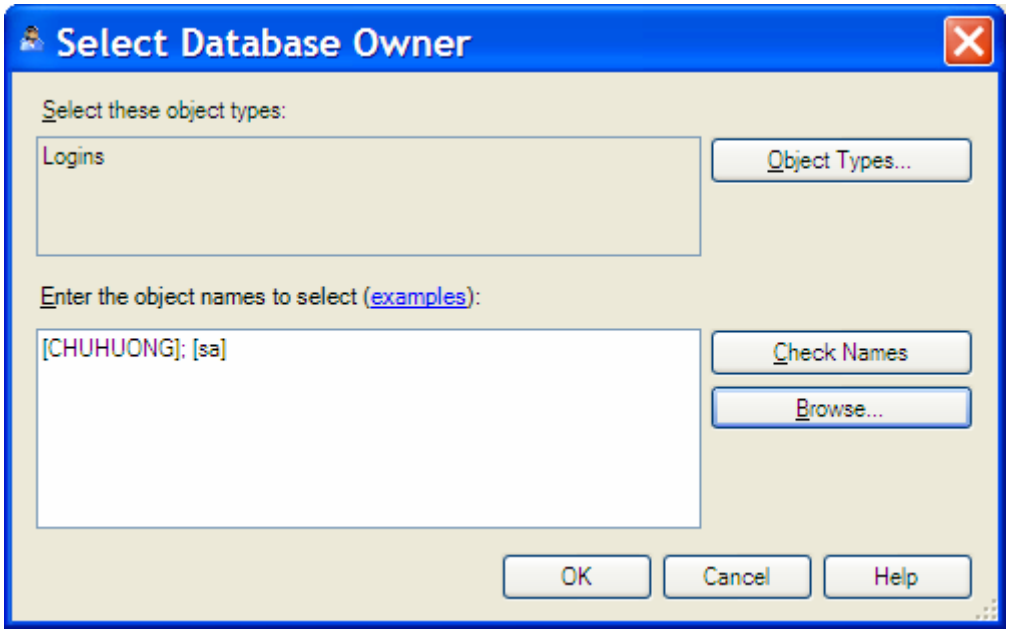


Hình 3.22. Cửa sổ New Database.

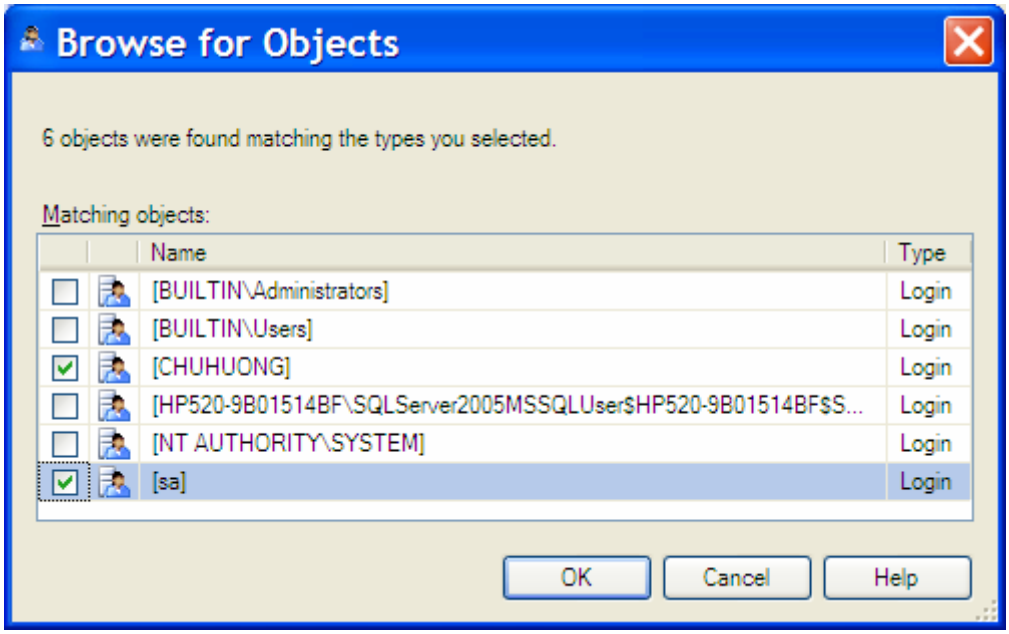
Trong cửa sổ New Database, tại tab General ta thực hiện điền các thông số cho các mục như sau:

+ Database name: Điền tên Database muốn tạo
+ Owner: Chỉ định tên các Logins sở hữu Database đang tạo. Để chọn các logins ta click vào nút … xuất hiện cửa sổ 'Select Database Owner' (Hình 3.23). Trong mục 'Enter the Object names to select' ta nhập tên các logins hoặc chọn nút

Browse để liệt kê và chọn các logins trong cửa sổ 'Browse for Objects' (Hình 3.24)



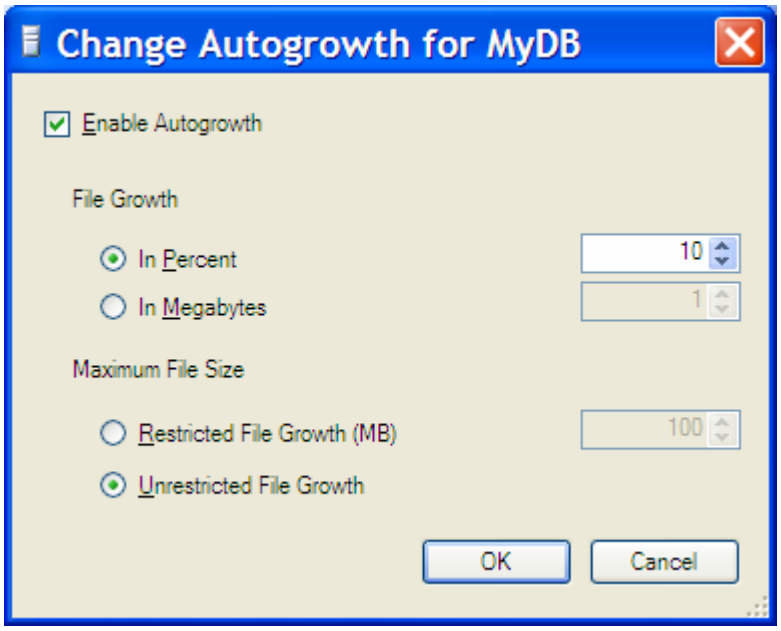
Hình 3.23. Cửa sổ Select Database Owner.



Hình 3.24. Cửa sổ Browse for Objects.

+ Database files: Thực hiện chọn các thuộc tính cho các files database, bao gồm:
  - Logical name: Tên logic của file database.
  - File Type: Kiểu file (Data, Log)

- Filegroup: Chỉ định nhóm file.
- Initial Size (MB): Kích thước khởi tạo dung lượng các file, tính theo MB.
- Autogrowth: Chỉ định các tham số tự động gia tăng dung lượng cho các file. Click vào nút … xuất hiện cửa sổ 'Change Autogrowth for MyDB' (Hình 3.25) cho phép khai báo các tham số Autogrowth:
  - Tùy chọn Enable Autogrowth cho phép tự động gia tăng;
  - File Growth chỉ định độ tự động gia tăng theo phần trăm (In Percent) hoặc theo dung lượng chỉ định (In Megabytes);
  - Maximum File Size định nghĩa độ gia tăng của file đến dung lượng lớn nhất.



Hình 3.25. Cửa sổ Change Autogrowth for MyDB

- Path: Chỉ định đường dẫn vật lý lưu trữ các files database. Click nút … để thay đổi đường dẫn mặc định.

**** Sử dụng T-SQL***

Ta có thể sử dụng T-SQL hay script để tạo Database, với cách này ta cần hiểu rõ về các cú pháp câu lệnh trong SQL Server để tạo Database.

```
CREATE DATABASE database_name
    [ ON
        [ PRIMARY ] [ <filespec> [ ,...n ]
        [ , <filegroup> [ ,...n ] ]
    [ LOG ON { <filespec> [ ,...n ] } ]
    ]
    ]
[;]
```

Để attach một database:

```
CREATE DATABASE database_name
    ON <filespec> [ ,...n ]
    FOR { ATTACH | ATTACH_REBUILD_LOG }
[;]
```

```
<filespec> ::=
{
(
    NAME = logical_file_name ,
    FILENAME = 'os_file_name'
        [ , SIZE = size [ KB | MB | GB | TB ] ]
        [ , MAXSIZE = { max_size [ KB | MB | GB | TB ]
| UNLIMITED } ]
        [ , FILEGROWTH = growth_increment [KB | MB |
GB | TB | % ] ]
) [ ,...n ]
}
```

```
<filegroup> ::=
{
FILEGROUP filegroup_name [ DEFAULT ]
    <filespec> [ ,...n ]
}
```

Các tham số trong đó:

database_name:

Là tên của CSDL. Nếu tên của file dữ liệu (data file) không được chỉ định thì SQL Server sử dụng *database_name* là tên cho *logical_file_name* và *os_file_name*.

ON

Chỉ định định nghĩa các file trên đĩa được sử dụng để lưu trữ các phần dữ liệu của database, *data files*.

PRIMARY

Chỉ định này liên quan đến danh sách định nghĩa primary file <filespec>. File đầu tiên được chỉ định trong <filespec> của nhóm *filegroup primary* filegroup trở thành *file primary*. Một database chỉ có thể duy nhất một *file primary*.

Nếu từ khóa PRIMARY không được chỉ định thì file đầu tiên trong danh sách các fie của câu lệnh CREATE DATABASE sẽ trở thành *file primary*.

LOG ON

Chỉ định định nghĩa file log lưu trữ trên đĩa, log files.

FOR ATTACH

Chỉ định database được tạo băng việc attach tập các file hệ thống đã tồn tại.

FOR ATTACH có các yêu cầu sau:

- Các files data (MDF và NDF) phải đã tồn tại.
- Nếu nhiều files log tồn tại, thì tất cả phải sẵn có.

FOR ATTACH_REBUILD_LOG

Chỉ định database được tạo băng việc attach tập các file hệ thống đã tồn tại. Nếu một hoặc nhiều files log giao dịch bị lỗi thì file log sẽ được xây dựng lại.

<filespec>

Điều khiển các thuộc tính của file.

- NAME logical_file_name: Chỉ định tên logical cho file. NAME được yêu cầu khi FILENAME được chỉ định.
- FILENAME *os_file_name*: Chỉ định tên, đường dẫn file hệ điều hành (file vật lý).
- SIZE size: Chỉ định kích thước file.

- MAXSIZE *max_size:* Chỉ định kích thước lớn nhất mà file có thể phát triển đến. Từ khóa UNLIMITED chỉ định file được phát triển cho đến khi đĩa bị đầy.
- FILEGROWTH *growth_increment*: Chỉ định độ tự đông gia tăng của file.

<filegroup>

Điều khiển các thuộc tính của filegroup.

- FILEGROUP filegroup_name: Chỉ định tên logical của filegroup.
- DEFAULT: Chỉ định tên filegroup là filegroup mặc định trên database.

Ví dụ 3.1. Đưa ra một cách tạo Database MyDB với tập tin dữ liệu chính là MyDB_Data.mdf, dung lượng khởi tạo là 1MB và tối đa là 10MB và độ gia tăng kích thước là 10%. Tập tin bản ghi giao dịch là MyDB_Log.ldf với dung lượng ban đầu là 2MB và kích thước tối đa không giới hạn, độ gia tăng dung lượng là 10MB.

Để tạo script (dùng T- SQL) ta mở cửa sổ query để soạn thảo:

- Đối với SQL Server 2000: Vào *Start/Programs/Microsoft SQL Server/Query Analyzer*, xuất hiện hộp thoại *Connect to SQL Server*, trong danh sách server chọn server cục bộ máy mình và nhập đoạn mã lệnh sau.
- Đối với SQL Server 2005: Trong cửa sổ SQL Server Management Studio, trên thanh Standard chọn nút New Query để tạo cửa sổ truy vấn kết nối đến thể hiện SQL đang được kết nối hoặc nút Database Engine Query để xuất hiện cửa sổ kết nối (Hình 3.8) ta thực hiện kết nối đến thể hiện SQL mà muốn thực hiện trên đó. Xuất hiện cửa sổ truy vấn ta thực hiện nhập đoạn mã lệnh sau:

```
Use master
go
create database MyDBT
On
(
Name= MyDBT_Data,
```

```
FileName='E:\Tempt\MyDBT_Data.mdf',
Size=10MB,
MaxSize=100MB,
FileGrowth=10%
)
Log On
(
Name= MyDBT_log,
FileName='E:\Tempt\MyDBT_Log.ldf',
Size=2MB,
MaxSize=UNLIMITED,
FileGrowth=10%
)
```

- Click nút *Execute* hoặc *F5* để chạy.

c) Xóa Database

** Dùng Enterprise Manager trong SQL Server 2000:*

Trong Enterprisse Manager, chọn nhóm server cục bộ và mở rộng mục Database. Sau đó right click lên CSDL muốn xóa và chọn Delete Database. Xuất hiện hộp thoại xác nhận xóa và chọn Yes.

** Sử dụng SQL Server Management Studio của SQL Server 2005:*

Để xóa một database ta đăng nhập vào SQL Server Management Studio với một login có quyền xóa database đó và thực hiện các bước sau:

1. Mở rộng các đối tượng trên thể hiện của SQL trong cửa sổ Object Explorer muốn xóa Database.
2. Mở rộng mục Database và Right click lên database muốn xóa và chọn Delete xuất hiện cửa sổ 'Delete Object' chọn OK để xóa.

** Dùng T-SQL*

Để xóa Database ta sử dụng cú pháp sau:

```
DROP DATABASE database_name [ ,...n ]
```

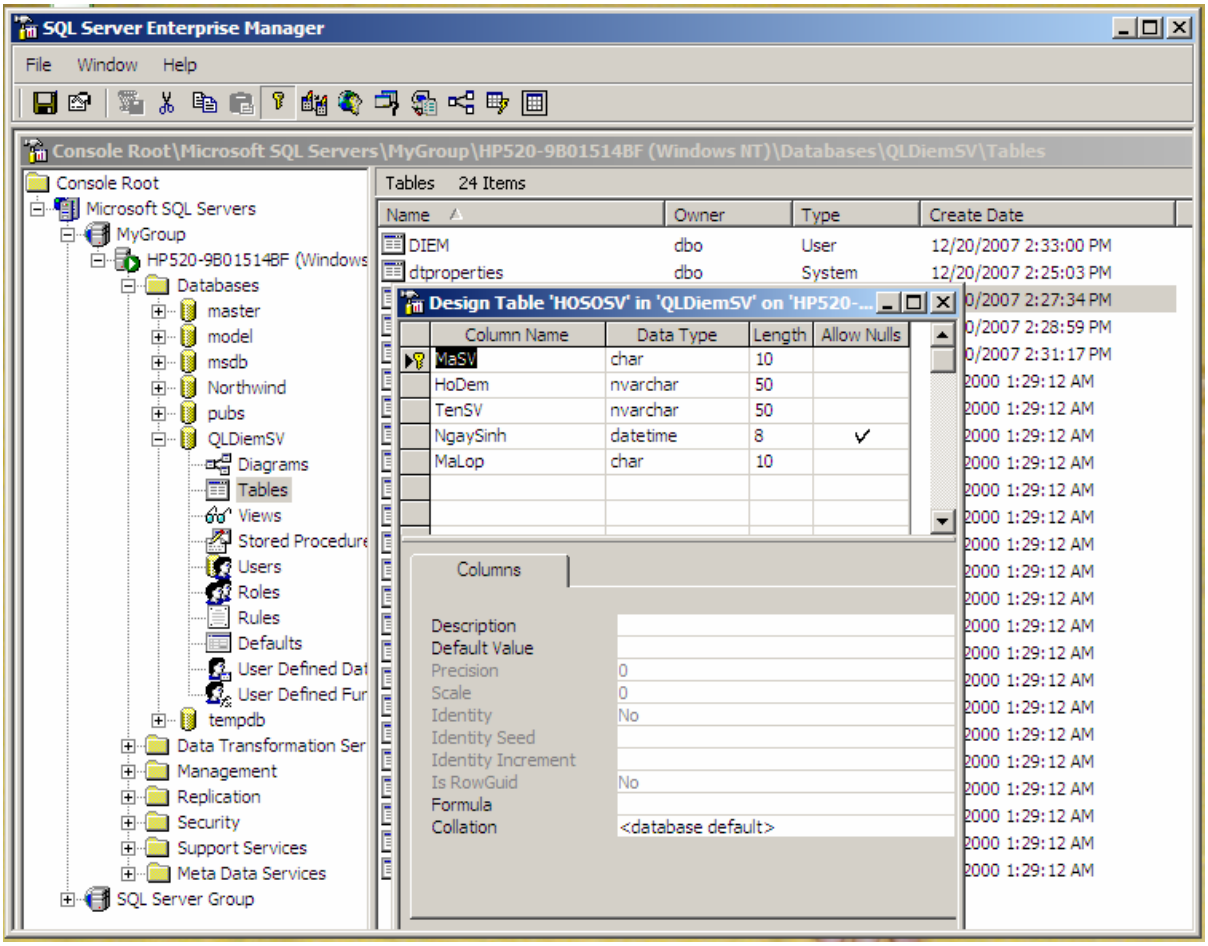
Ví dụ: nhập đoạn T-SQL sau để xóa Database MyDB.

```
Use master
Go
Drop Database MyDB
Go
```

	2^30-1 ký tự. Hỗ trợ unicode.
nvarchar[(n)]	Dữ liệu ký tự có chiều dài thay đổi với n ký tự. Có hỗ trợ Unicode. Với 1<n<4000.
real	Dữ liệu số thực động, phạm vi từ - 3.4E+38 đến 3.4E+38. Chiếm 4 byte.
smalldatetime	Dữ liệu ngày giờ từ 1/1/1900 đến 6/6/2079. Chiếm 4 byte.
smallint	Dữ liệu số nguyên từ -2^15 đến 2^15-1. Chiếm 2 byte.
smallmoney	Dữ liệu kiểu tiền tệ từ -2^31 đến 2^31. Chiếm 4 byte.
sql_variant	Cho phép giữ các giá trị của các kiểu dữ liệu khác nhau (tất cả các kiểu dữ liệu hệ thống khác).
sysname	Đây là kiểu dữ liệu đặc biệt, do SQL Server định nghĩa là kiểu nvarchar(128). Chiếm 256 byte.
table	Tương tự như bảng tạm, khai báo gồm danh sách các cột và các kiểu dữ liệu.
text	Dữ liệu ký tự có chiều dài thay đổi, dài hơn 8000byte. Có thể lưu tới 2^31 ký tự. Không hỗ trợ Unicode.
timestamp	Cột timestamp được cập nhật tự động mỗi khi dòng được thêm hoặc được cập nhật. Mỗi bảng chỉ có thể có 1 cột timestamp. Kích thước lưu trữ là 8 byte.
tinyint	Dữ liệu số nguyên từ 0 đến 255. Chiếm 1 byte.
unique-identifier	Giá trị nhị phân 16 byte, là số định danh duy nhất toàn cục.
varbinary[(n)]	Dữ liệu nhị phân n có chiều dài thay đổi. Với 1<n<8000
varchar[(n)	Dữ liệu ký tự có chiều dài thay đổi với n ký tự. Không hỗ trợ Unicode. Với 1<n<4000.

b) Tảo bảng.

** Dùng SQL Server Enterprise Manager trong SQL Server 2000:*



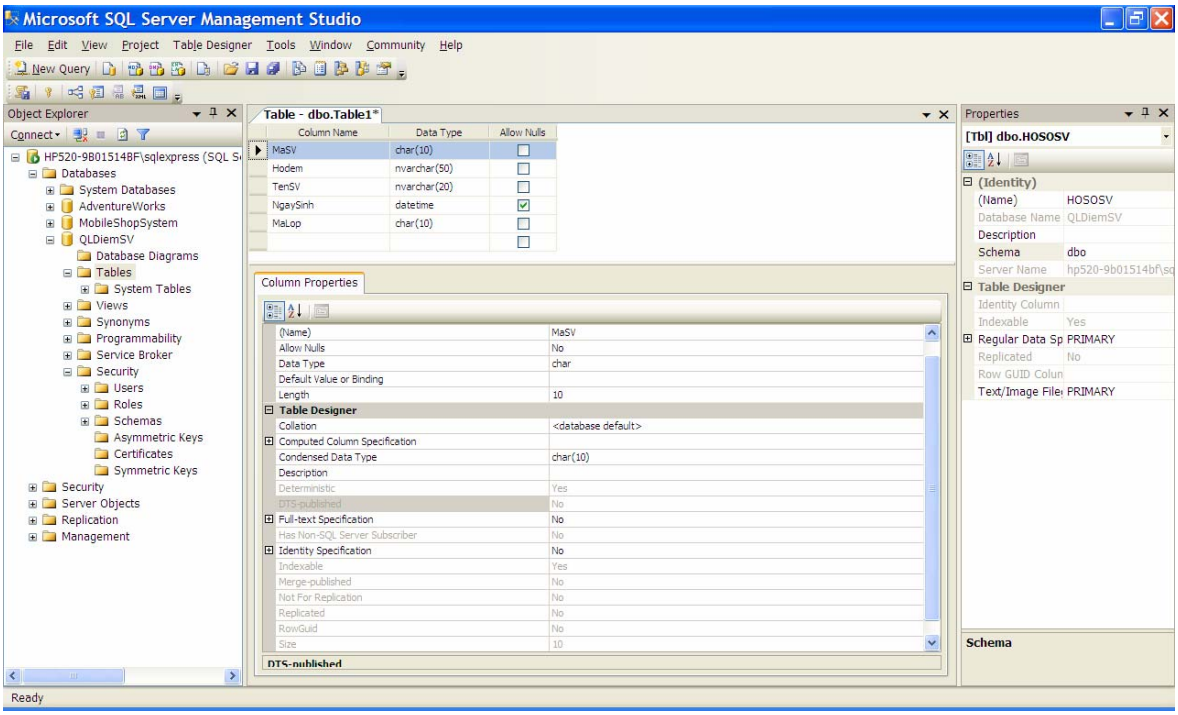
Hình 3.26. Cửa sổ thiết Table.

- Trong Enterprise Manager, mở rộng danh mục Database và mở rộng cơ sở dữ liệu QLDiemSV.
- Right Click lên danh mục Tables, chọn New Table.
- Xuất hiện cửa sổ thiết kế bảng như hình 3.26. Ta thực hiện thiết kế bảng HOSOSV gồm các trường (trong cột Column Name), kiểu trường (Data Type) và độ dài dữ liệu (Length) và cho phép trường đó nhận giá trị NULL hay không (Allow Nulls) hình 3.26.
- Chọn dòng MaSV, sau đó click vào nút Set primary key để thiết lập trường MaSV là khóa.
- Click vào nút Save, xuất hiện cửa sổ *Choose Name* nhập tên bảng là HOSOSV để lưu lại bảng.

** Dùng SQL Server Management Studio trong SQL Server 2005:*

Để tạo bảng trong SQL Server Management Studio ta thực hiện:

1. Mở rộng các đối tượng trên thể hiện của SQL trong cửa sổ Object Explorer muốn tạo bảng.
2. Mở rộng mục Database và chọn cơ sở dữ liệu muốn tạo bảng. Right click lên mục Table và chọn New Table xuất hiện cửa sổ thiết kế table (Hình 3.27):
   + Column Name: Nhập tên các cột trong bảng.
   + Data Type: Chọn kiểu dữ liệu và độ dài cho kiểu.
   + Allow Nulls: Cho phép chấp nhận dữ liệu NULL hay không?
   + Column Properties: Thiết lập các thuộc tính cho cột đang được chọn.



Hình 3.27. Cửa sổ thiết Table trong SQL Server Management Studio .

3. Để thiết lập khóa, chọn các trường khóa của bảng sau đó click vào nút biểu tượng khóa (Set Primary Key) hoặc right click vào các trường đó và chọn Set Primary Key.

4. Dùng tổ hợp phím Ctrl+S hoặc click vào nút Save để lưu cấu trúc bảng vừa tạo trong cơ sở dữ liệu.

** Dùng T-SQL*

Để tạo bảng dùng T – SQL ta sử dụng cú pháp sau:

```
CREATE TABLE table_name
   ( { < column_definition > | < table_constraint > }
[ ,...n ]
   )

< column_definition > ::=
   { column_name data_type }
   [ { DEFAULT constant_expression
      | [ IDENTITY [ ( seed , increment ) ]
      ]
    } ]
   [ < column_constraint > [ ...n ] ]

< column_constraint > ::=
   [ CONSTRAINT constraint_name ]
   { [ NULL | NOT NULL ]
      | [ PRIMARY KEY | UNIQUE ]
      | REFERENCES ref_table [ ( ref_column ) ]
      [ ON DELETE { CASCADE | NO ACTION } ]
      [ ON UPDATE { CASCADE | NO ACTION } ]
   }

< table_constraint > ::=
   [ CONSTRAINT constraint_name ]
   { [ { PRIMARY KEY | UNIQUE }
      { ( column [ ,...n ] ) }
      ]
   | FOREIGN KEY
     ( column [ ,...n ] )
      REFERENCES ref_table [ ( ref_column [ ,...n ] )
]
      [ ON DELETE { CASCADE | NO ACTION } ]
      [ ON UPDATE { CASCADE | NO ACTION } ]
   }
```

Ví dụ 3.2. Tạo bảng

a) Tạo bảng LOP dùng T-SQL.

```
Use QLDiemSV
Go
Create Table DMLop
(MaLop char(10) NOT NULL,
TenLop nvarchar(50) NOT NULL,
Khoa char(10)
CONSTRAINT PK_DMLOP Primary Key (MaLop))
```

b) Tạo hai bảng MyCustomers và MyOrders. Bảng MyCustomers có hai cột, cột CustID vừa là cột nhận dạng `IDENTITY[(seed,increment )]` có giá trị khởi tạo ban đầu là 100 và bước nhảy là 1 vừa là khóa. Bảng MyOrders có khóa là OrderID và khóa ngoại là CustID tham chiếu đến bảng MyCustomers.

```
CREATE TABLE MyCustomers (CustID int IDENTITY (100,1)
PRIMARY KEY, CompanyName nvarchar (50))
CREATE TABLE MyOrders (OrderID int PRIMARY KEY,
CustID int REFERENCES MyCustomers(CustID))
```

c) Thay đổi cấu trúc bảng

** Dùng SQL Server Enterprise Manager trong SQL Server 2000:*

+ Trong Enterprise Manager, mở rộng danh mục Database, mở rộng cơ sở dữ liệu có bảng cần thay đổi cấu trúc, chẳng hạn QLDiemSV và chọn mục Table.
+ Right Click lên bảng thuộc CSDL QLDiemSV cần thay đổi cấu trúc, chọn Design Table. Xuất hiện cửa sổ thiết kế bảng như hình 3.26. Ta thực hiện các thay đổi về cấu trúc: đổi kiểu dữ liệu, thêm trường, xóa trường, thiết lập khóa .v.v…với bảng đã chọn đó.

** Dùng SQL Server Management Studio trong SQL Server 2005:*

Để thay đổi cấu trúc bảng trong SQL Server Management Studio ta thực hiện các bước sau:

+ Mở rộng các đối tượng trên thể hiện của SQL trong cửa sổ Object Explorer muốn thay đổi cấu trúc bảng.

+ Mở rộng mục Database và chọn cơ sở dữ liệu và bảng cần thay đổi cấu trúc. Right click lên bảng đó và chọn Modify xuất hiện cửa sổ thiết kế table (Hình 3.27) và ta thực hiện các thay đổi đối với cấu trúc bảng như cách tạo bảng.

** Dùng T-SQL*

Để thay đổi cấu trúc bảng bằng T-SQL ta có cú pháp câu lệnh sau:

```
ALTER TABLE table_name
{
    ALTER COLUMN column_name
    {
        type_name [ ( { precision [ , scale ]
            | max } ) ]
        [ NULL | NOT NULL ]
    }

    | [ WITH { CHECK | NOCHECK } ] ADD
    {
        <column_definition>
      | <computed_column_definition>
      | <table_constraint>
    } [ ,...n ]

    | DROP
    {
        [ CONSTRAINT ] constraint_name
        | COLUMN column_name
    } [ ,...n ]
    | [ WITH { CHECK | NOCHECK } ] { CHECK | NOCHECK }
CONSTRAINT
        { ALL | constraint_name [ ,...n ] }
    | { ENABLE | DISABLE } TRIGGER
        { ALL | trigger_name [ ,...n ] }
 [ ; ]
```

Ví dụ 3.3. Thay đổi cấu trúc bảng MyCustomers

```
ALTER TABLE MyCustomers ADD CustType CHAR(10) NULL
```

d) Nhập dữ liệu cho bảng

** Dùng Enterprise Manager*

- Trong Enterprise Manager, mở rộng danh mục Database, mở rộng cơ sở dữ liệu QLDiemSV và chọn mục Table.
- Right Click lên bảng thuộc CSDL QLDiem cần nhập dữ liệu, chọn *Open Table\Return all row*. Xuất hiện cửa sổ nhập dữ liệu cho bảng đó.

** Dùng SQL Server Management Studio trong SQL Server 2005:*

+ Mở rộng các đối tượng trên thể hiện của SQL trong cửa sổ Object Explorer muốn nhập dữ liệu cho bảng.
+ Mở rộng mục Database, chọn cơ sở dữ liệu và bảng cần nhập dữ liệu. Right click lên bảng cần nhập dữ liệu và chọn Open Table xuất hiện cửa sổ nhập dữ liệu cho bảng.

** Dùng T-SQL*

Ví dụ 3.3. Chèn dữ liệu vào bảng LOP

- Chèn một bản ghi:

```
Insert Into LOP(MaLop, TenLop, Khoa)
        VALUES ('TH6A',N'Tin học 6A', '6')
```

- Chèn tất cả các bản ghi từ bảng DMLOP vào bảng LOP.

```
Insert Into LOP(MaLop, TenLop, Khoa)
    Select MaLop, TenLop, Khoa From DMLOP
```

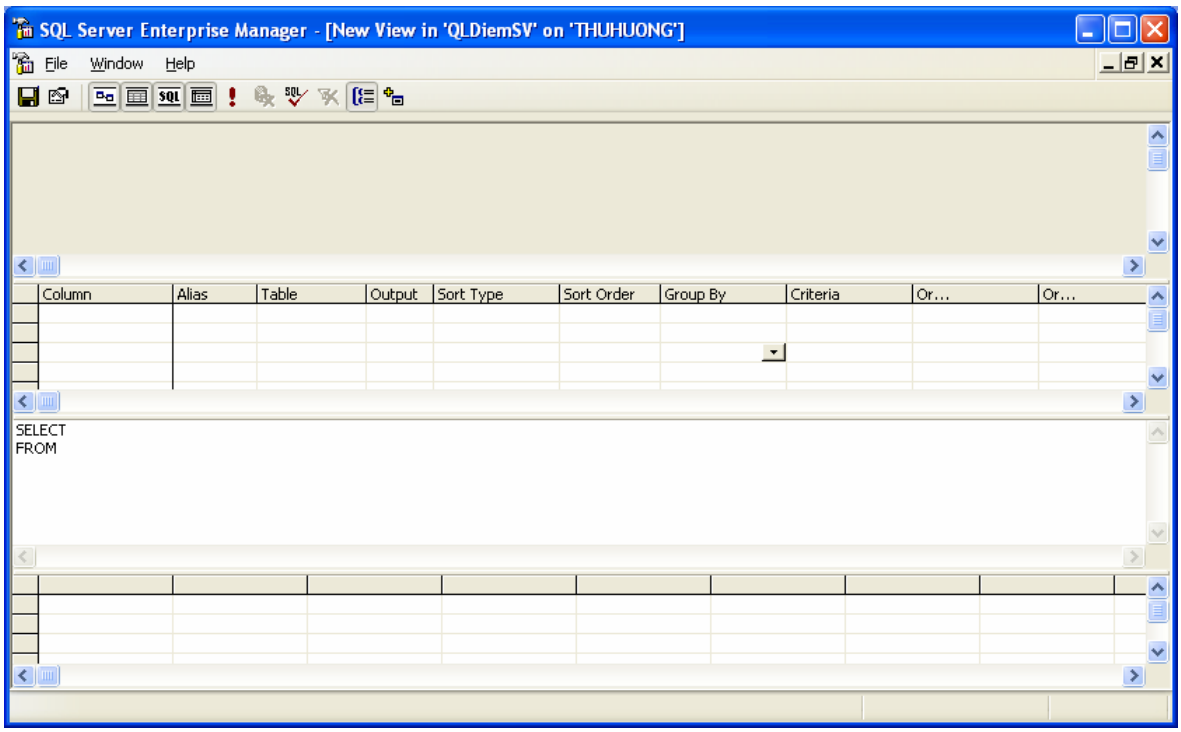
e) Xóa bảng

** Dùng Enterprise Manager*

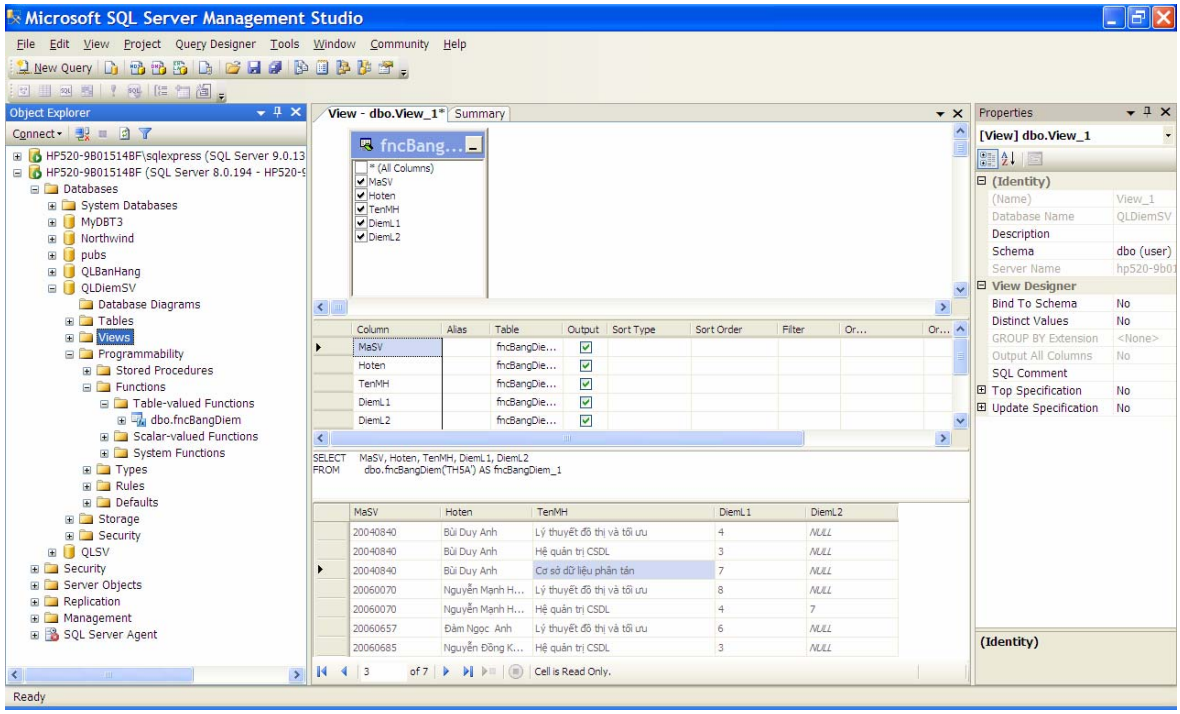
- Trong Enterprise Manager, mở rộng danh mục Database, mở rộng cơ sở dữ liệu QLDiemSV và chọn mục Table.
- Right Click lên bảng cần xóa, chọn Delete.

** Dùng SQL Server Management Studio trong SQL Server 2005:*

+ Mở rộng các đối tượng trên thể hiện của SQL trong cửa sổ Object Explorer muốn xóa bảng.



Hình 3.28. *Cửa sổ thiết kế view trên SQL Server 2000.*



Hình 3.29. *Cửa sổ thiết kế view trên SQL Server 2005.*

+ *Diagram pane*: Hiển thị dữ liệu nguồn có thể là các bảng, các view khác, functions để tạo view. Các cột dữ liệu của view được chọn từ vùng này.

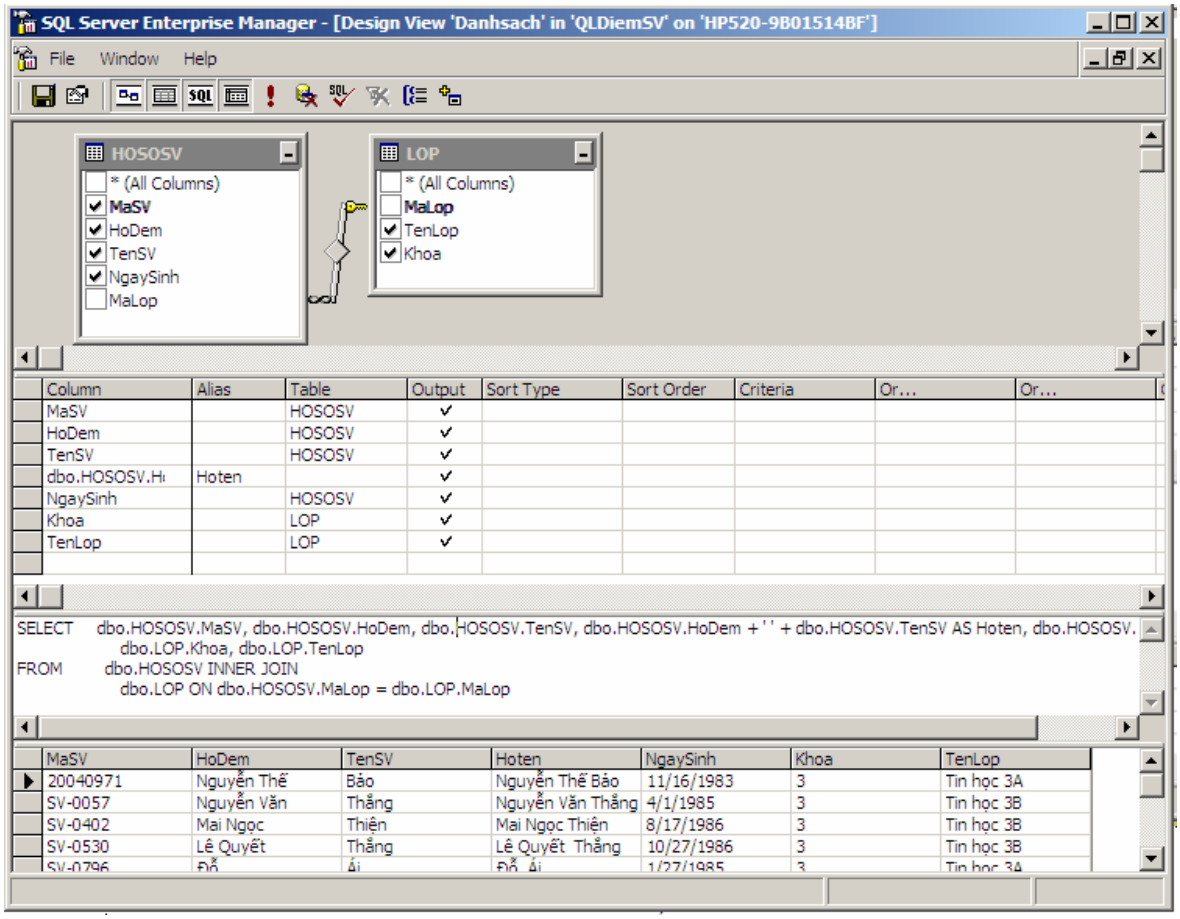
+ *Grid pane (Criteria pane)*: Hiển thị các cột của view đã được chọn từ vùng diagram.
+ *SQL pane*: Hiển thị phát biểu SQL dùng để định nghĩa view.
+ *Results pane*: Hiển thị kết quả nhận được từ view.

Ta có thể ẩn hoặc hiện các cửa sổ này bằng cách click vào các nút tương ứng trên thanh công cụ của cửa sổ thiết kế view. Danh sách sau thể hiện ý nghĩa của các nút trên thanh công cụ tính từ trái sang phải:

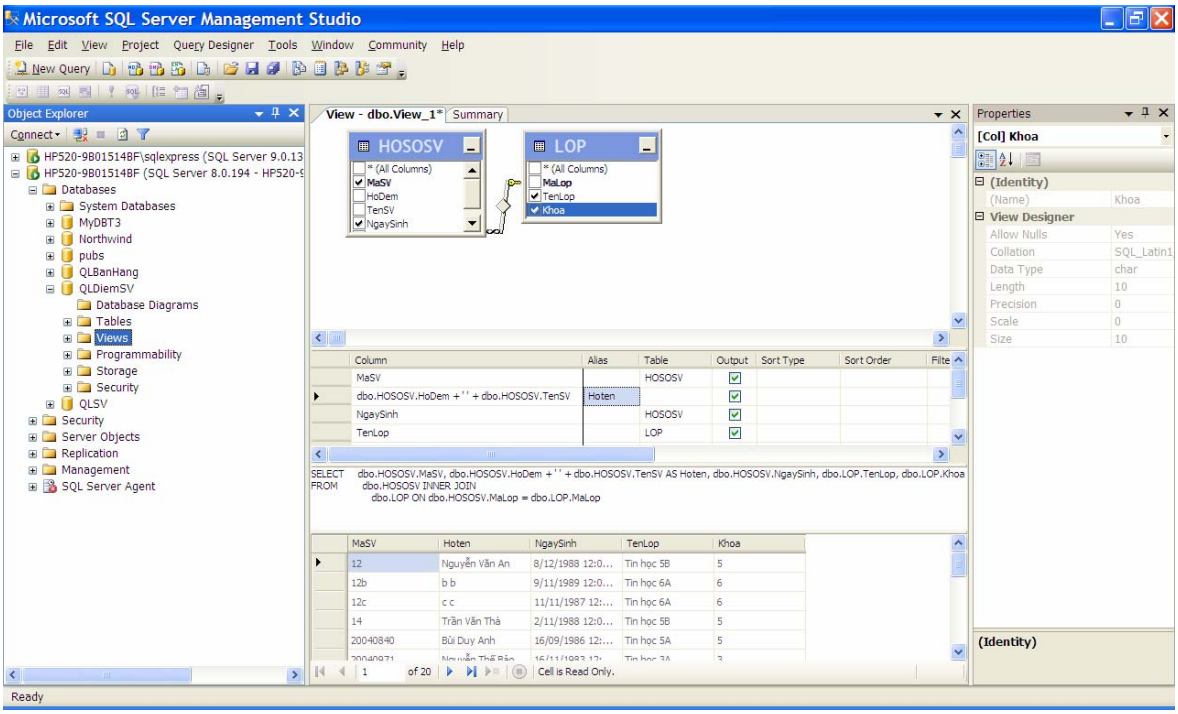
- Đối với SQL Server 2000:
  + *Save*: Lưu view.
  + *Properties*: Cho phép thay đổi thuộc tính của view.
  + *Show hide pane*: các nút ẩn hoặc hiện các cửa sổ trên.
  + *Run*: Thực thi và hiển thị kết quả của view trong Results pane.
  + *Cancel Execution And Clear Results*: Xóa Results pane.
  + *Verify SQL*: Kiểm tra cú pháp của phát biểu SQL.
  + *Remove Filter*: loại bỏ tất cả các lọc dữ liệu đã được định nghĩa.
  + *Use GROUP BY*: Thêm mệnh đề group by trong câu lệnh SQL.
  + *Add Table*: Thêm các bảng, view làm nguồn dữ liệu cho view mới.
- Đối với SQL Server 2005:
  + *Show Diagram pane*: Ẩn hoặc hiện cửa sổ Diagram pane.
  + *Show Criteria pane*: Ẩn hoặc hiện cửa sổ Criteria pane.
  + *Show SQL pane*: Ẩn hoặc hiện cửa sổ SQL pane.
  + *Show Results pane*: Ẩn hoặc hiện cửa sổ kết quả.
  + *Execute SQL*: Thực hiện truy vấn.
  + *Verify SQL Syntax*: Kiểm tra cú pháp câu lệnh SQL.
  + *Add Group By*: Thêm mệnh đề group by trong câu lệnh SQL.

+ *Add Table*: Thêm các bảng, view, hàm làm nguồn dữ liệu cho view mới.

- Click vào nút Add Table, xuất hiện hộp thoại Add Table chọn các bảng các view làm nguồn dữ liệu cho view mới. Trong ví dụ ta chọn bảng HOSOSV và LOP.
- Sau khi chọn nguồn dữ liệu ta thực hiện chọn các trường làm dữ liệu trên view như hình 3.30 (SQL Server 2000) và 3.31 (SQL Server 2005)
- Click vào nút Save, xuất hiện hộp thoại Save lưu với tên Danhsach.



Hình 3.30. *Cửa sổ thiết kế view Danhsach trong SQL Server 2000 .*



Hình 3.31. *Cửa sổ thiết kế view Danhsach trong SQL Server 2005.*

Sử dụng vùng Grid Pane (Criteria) để hỗ trợ cho việc thiết kế View:

+ *Cột Column*: Chứa tên các trường trong bảng\view làm nguồn dữ liệu cho truy vấn này hoặc chứa một biểu thức định nghĩa một trường mới cho view.
+ *Cột Alias*: Chỉ định bí danh cho trường trong view mới.
+ *Cột Table*: Chỉ định nguồn dữ liệu.
+ *Cột Output*: Chỉ định hiển thị hay không hiển thị trường đó.
+ *Cột Sort Type và Sort Order*: Dùng để sắp xếp dữ liệu.
+ *Cột Criteria*: Dùng để đặt điều kiện lọc cho các bản ghi.
+ *Các cột Or*: Dùng kết hợp với cột Criteria để tạo các điều kiện lọc dữ liệu phức tạp.

Tính tổng trong truy vấn: Để tính tổng trong truy vấn dùng vùng Grid Pane ta có thể tiến hành phân nhóm các bản ghi và thực hiện tính toán trên từng phân nhóm đó. Để tính tổng trong truy vấn ta chọn nút Use Group By

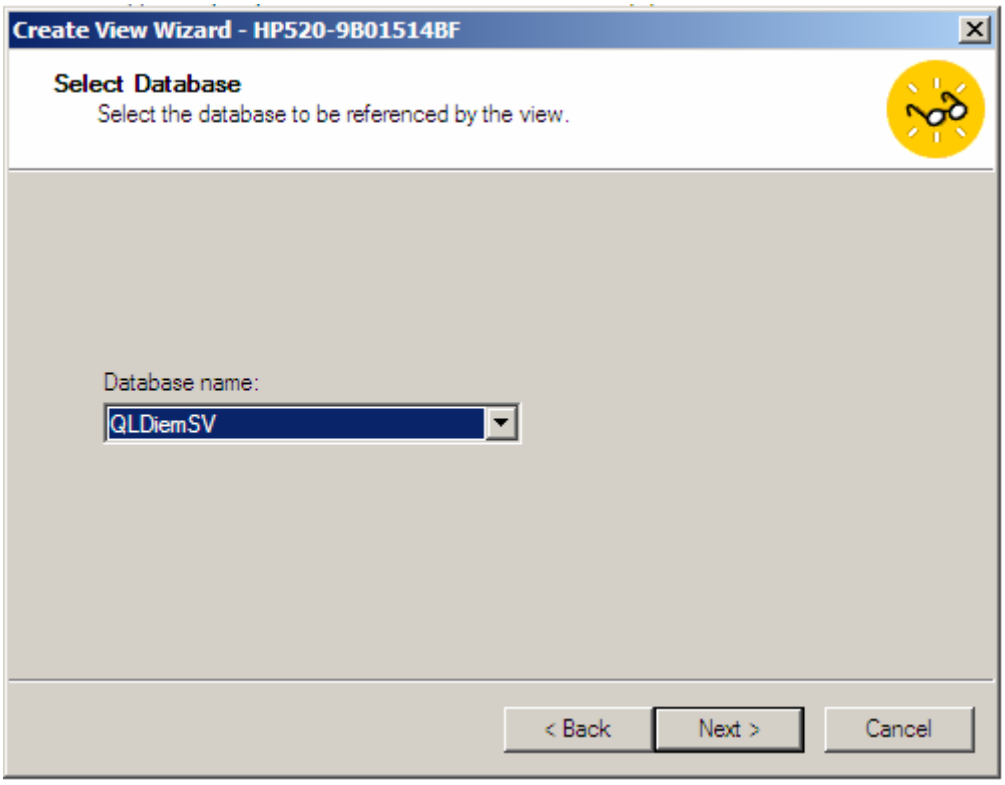
xuất hiện cột Group By trong vùng Grid Pane. Ta thực hiện tiến hành phân các nhóm trường như sau:

+ Trường làm điều kiện, tiêu chuẩn tham gia phân nhóm và tính tổng: Chọn Where trong cột Group By và đặt biểu thức điều kiện trong cột Creteria.
+ Trường phân nhóm: Chọn Group by trong cột Group By.
+ Trường tính toán: Chọn một hàm có sẵn (sum, count, avg, max, min, .v.v...) trong cột Group By hoặc xây dựng một biểu thức tính toán trong cột Column.
+ Định tiêu chuẩn hiển thị kết quả: Đặt điều kiện ở cột Criteria tại các trường phân nhóm và trường tính toán.
+ Chọn thứ tự hiển thị: Dùng cột Sort Type và Sort Order tại các các trường phân nhóm và các trường tính toán.

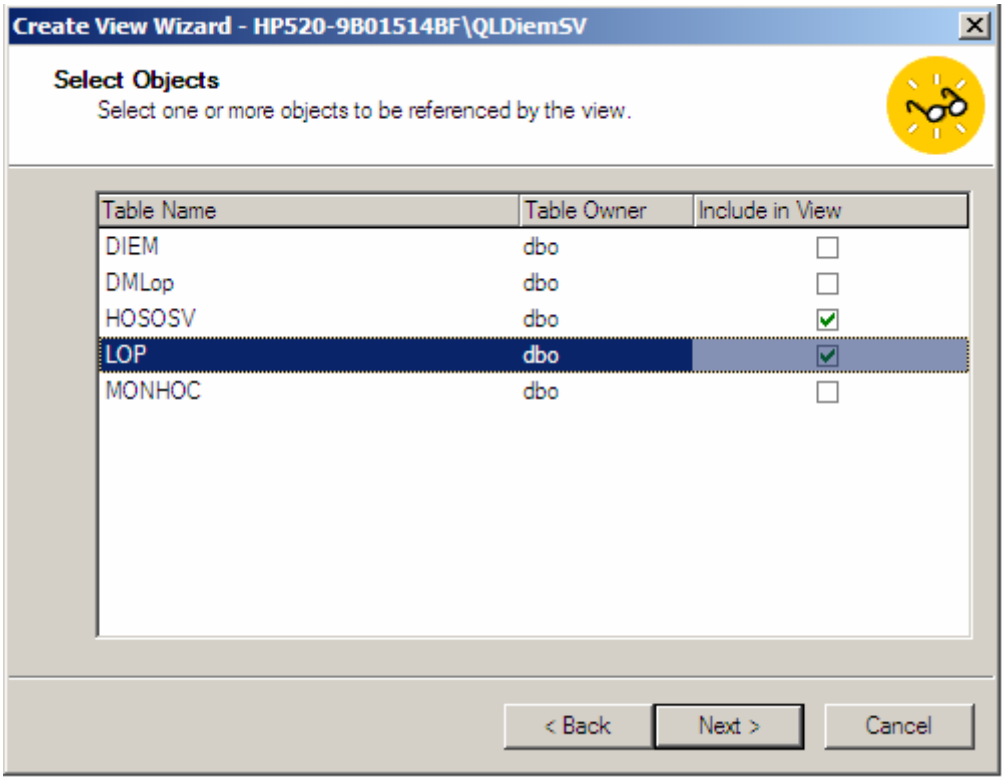
** Dùng Create view wizard*

- Khởi động SQL Server Enterprise Manager, chọn tên server cục bộ và vào menu Tools\Wizards. Xuất hiện cửa sổ Select Wizard hình 3.5.
- Chọn Create View Wizard xuất hiện cửa sổ Welcometo the Create View Wizard, như hình 3.32. Chọn Next.
- Xuất hiện cửa sổ Select Database (Hình 3.33) chọn CSDL mà view được tạo trên đó. Chọn Next.
- Trong cửa sổ Select Objects chọn các bảng làm nguồn dữ liệu cho View mới (Hình 3.34). Chọn Next.
- Chọn các cột dữ liệu của view (hình 3.35).
- Cửa sổ tiếp theo (hình 3.36): Đặt điều kiện lọc tương tự như mệnh đề WHERE trong khối câu lệnh SELECT. Chọn Next.
- Chọn tên view (hình 3.37) và chọn Finish (hình 3.38) để kết thúc.

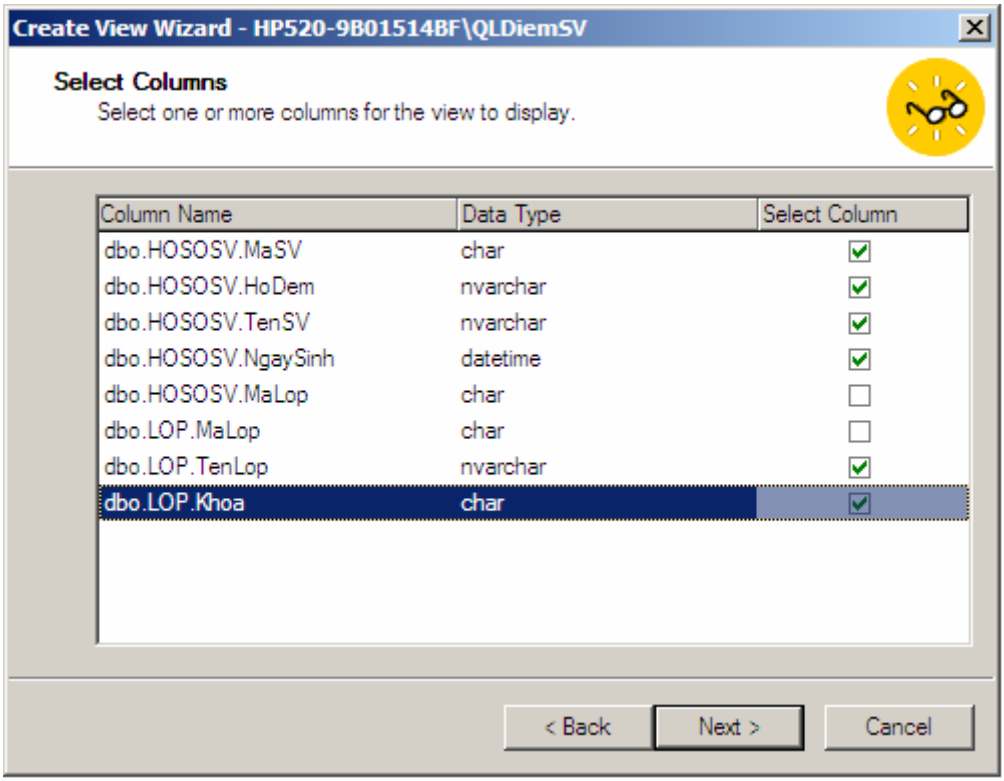
Hình 3.32.*Cửa sổ Welcometo the Create View Wizard.*



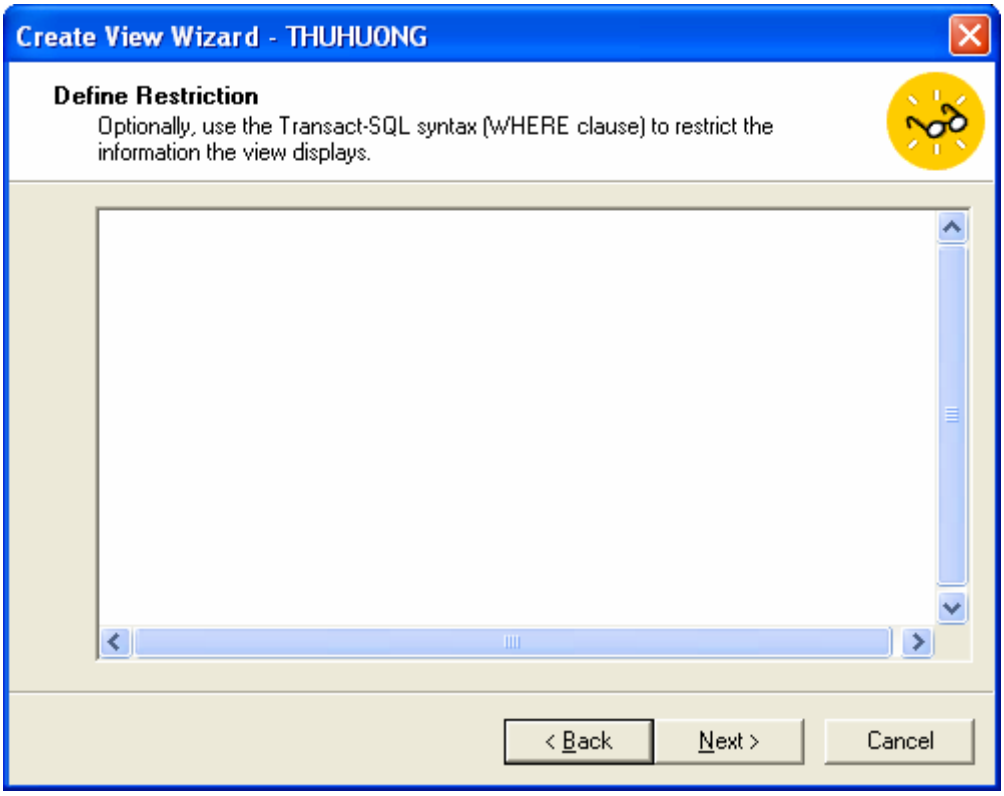
Hình 3.33.*Cửa sổ Select Database.*



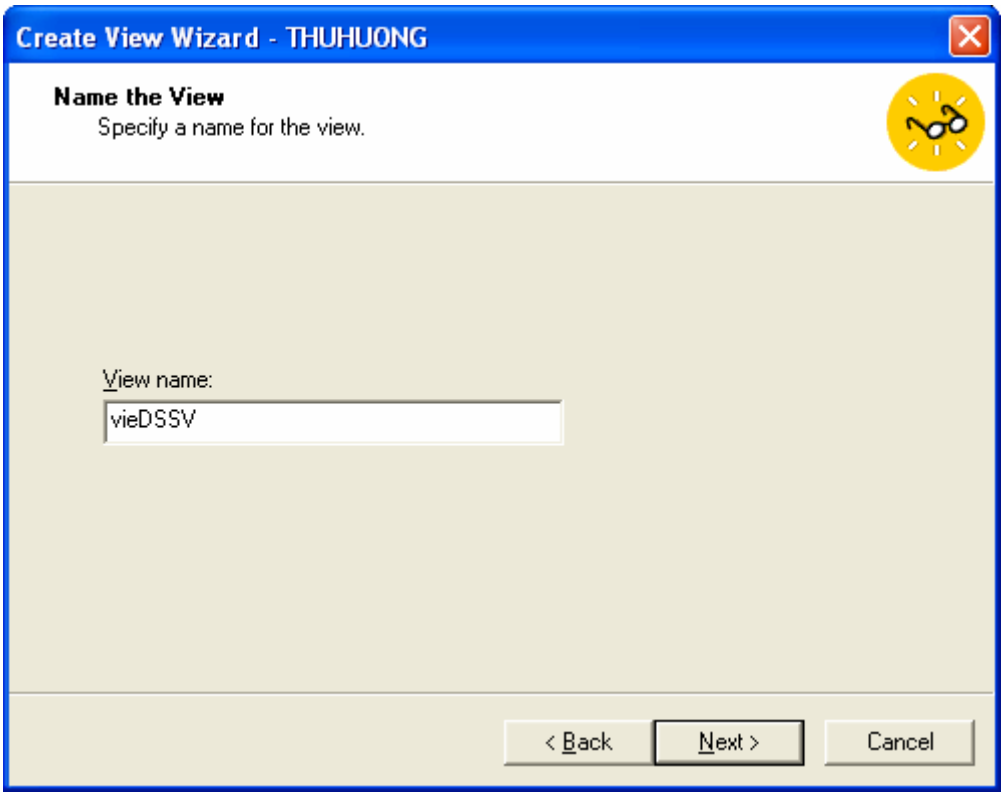
Hình 3.34.*Cửa sổ Select Objects.*



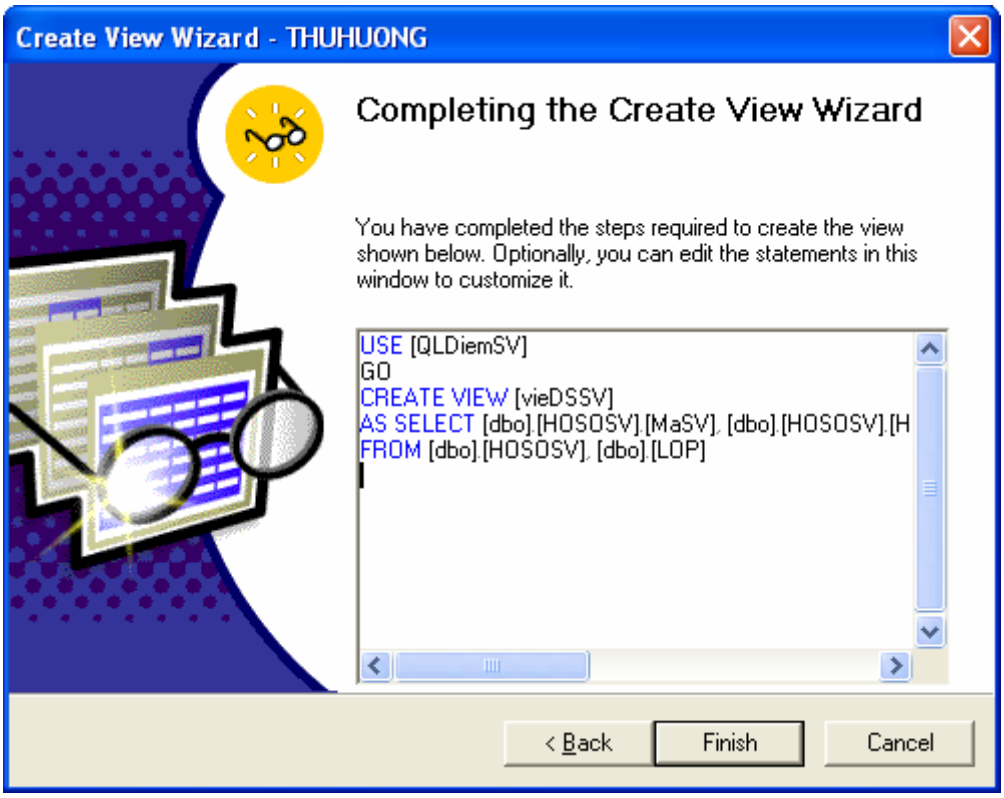
Hình 3.35.*Cửa sổ Select Columns.*



Hình 3.36.*Cửa sổ Define Restriction*



Hình 3.37.*Cửa sổ Define Restriction*



Hình 3.38.*Cửa sổ Define Restriction*

** Dùng T-SQL:* Ta dùng cú pháp câu lệnh sau:

```
CREATE VIEW [schema_name.]view_name[(column[,...n])]
[ WITH <view_attribute> [ ,...n ] ]
AS select_statement [ ; ]
[ WITH CHECK OPTION ]

<view_attribute> ::=
{    [ ENCRYPTION ]
     [ SCHEMABINDING ]
     [ VIEW_METADATA ]      }
```

Trong đó:

schema_name

Là tên của lược đồ mà view thuộc lược đồ đó.

view_name

Là tên của view.

column

Là tên được sử dụng cho một cột trong view. Tên một cột chỉ yêu cầu khi cột đó được sinh ra từ một biểu thức đại số, một hàm, một hằng; khi hai hoặc nhiều cột khác nhau có cùng tên, điển hình là trong liên kết; hoặc khi một cột trong view được chỉ định tên khác với tên từ nguồn dữ liệu. Các tên cột có thể được gán trong câu lệnh SELECT. Nếu cột không được chỉ định trong view thì các cột trong view có tên cùng tên với các cột câu lệnh SELECT.

select_statement

Là các khối câu lệnh SELECT định nghĩa view. Các khối câu lệnh SELECT được phân cách nhau bởi mệnh đề UNION hoặc UNION ALL.

Một Index được định nghĩa trên view đòi hỏi view phải được xây dựng từ một bảng đơn hoặc nhiều bảng liên kết nhau trong tổ hợp tùy chọn.

CHECK OPTION

Bắt buộc tất các câu lệnh sửa đổi dữ liệu thực hiện trên view phải tuân theo các điều kiện trong khối câu lệnh select_statement.

ENCRYPTION

Mã hóa bản text của câu lệnh CREATE VIEW trong sys.syscomments. Việc sử dụng WITH ENCRYPTION ngăn cản view bị công bố như là một phần bản sao SQL Server.

SCHEMABINDING

Buộc view vào một lược đồ dưới một bảng hoặc nhiều bảng. Khi SCHEMABINDING được chỉ định, bảng hoặc các bảng cơ cở không được sửa trong những cách ảnh hưởng đến định nghĩa view.

VIEW_METADATA

Chỉ định các thể hiện SQL Server sẽ trả về cho DB-Library, ODBC, và OLE DB APIs thông tin siêu dữ liệu về view, thay cho bảng và các bảng cơ sở khi trình duyệt siêu dữ liệu được yêu cầu cho một truy vấn tham chiếu đến view.

Ví dụ 3.4. Trong CSDL *Northwind* có hai bảng *Orders* và *Order Details*. Ta xây dựng view tính tổng giá trị cho từng hóa đơn (SQL Server 2000).

```
Create View TongGT
as
SELECT dbo.Orders.OrderID,
       SUM(dbo.[Order Details].UnitPrice *
dbo.[Order Details].Quantity) AS Tolal
FROM   dbo.Orders INNER JOIN
       dbo.[Order Details] ON dbo.Orders.OrderID
= dbo.[Order Details].OrderID
GROUP BY dbo.Orders.OrderID
Go
```

c) Thay đổi view

** Dùng Enterprise Manager*

- Trong Enterprise Manager, mở rộng danh mục Database, mở rộng cơ sở dữ liệu muốn tạo view, chẳng hạn CSDL QLDiemSV và chọn mụcViews.
- Right Click lên View, chọn Design View. Xuất hiện cửa sổ thiết kế view.
- Thực hiện các thay đổi trên view và ghi lại các thay đổi đó.

** Dùng SQL Server Management Studio*

- Trong SQL Server Management Studio, mở rộng danh mục Database, mở rộng cơ sở dữ liệu muốn tạo view, chẳng hạn CSDL QLDiemSV và chọn mụcViews.
- Right Click lên View, chọn Modify. Xuất hiện cửa sổ thiết kế view.
- Thực hiện các thay đổi trên view và ghi lại các thay đổi đó.

** Dùng T-SQL:* Ta dùng cú pháp câu lệnh sau

```
ALTER VIEW [schema_name.]view_name
[(column[,...n])]
[ WITH <view_attribute> [ ,...n ] ]
AS select_statement [ ; ]
[ WITH CHECK OPTION ]

<view_attribute> ::=
{[ENCRYPTION][SCHEMABINDING][VIEW_METADATA]
}
```

Ví dụ 3.5. Sửa đổi view DSSV như sau:

```
ALTER VIEW DSSV
AS
    Select  MaSV, Hodem + ' '+ TenSV AS Hoten, Ngaysinh
    From HosoSV
GO
```

- Chỉ mục liên cung (clustered): Là chỉ mục lưu trữ các dòng dữ liệu thực sự của bảng trong nút lá, theo thứ tự đã được sắp xếp.
- Chỉ mục phi liên cung (Nonclustered): Không chứa dữ liệu trong nút lá, mà nó chứa thông tin về vị trí của dòng dữ liệu: nếu không có chỉ mục liên cung trên bảng thì nó chứa số nhận dạng dòng (Row ID); nếu có chỉ mục liên cung thì trong nút lá này sẽ chứa giá trị khóa chỉ mục liên cung cho dữ liệu đó.

Chú ý: Indexes được tạo tự động khi các ràng buộc PRIMARY KEY và UNIQUE được định nghĩa trên các cột của bảng

b) Tạo chỉ mục

Để tạo Indexes trong SQL Server 2000 , ta có 3 cách khác nhau để tạo:

+ Sử dụng Create View Wizard
+ SQL Server Enterprise Manager
+ Dùng T-SQL

Trong SQL Server 2005, ta có 2 cách khác nhau để tạo:

+ SQL Server Management Studio
+ Dùng T-SQL

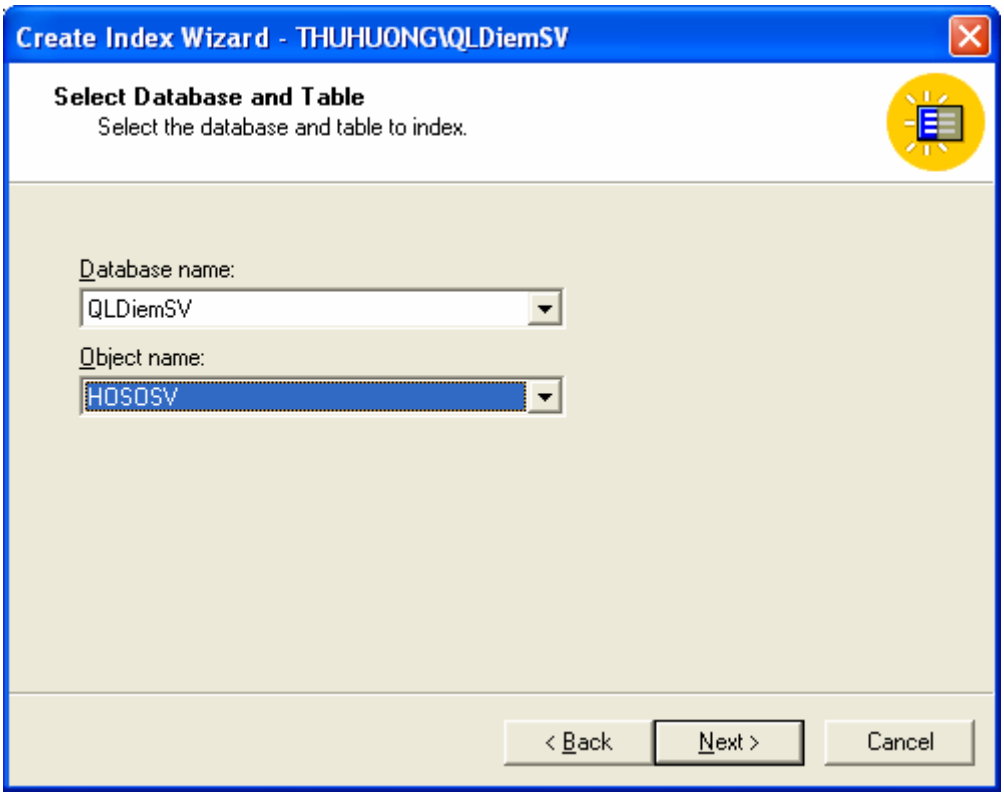
** Dùng wizard*

- Khởi động SQL Server Enterprise Manager, chọn tên server cục bộ và vào menu *Tools\Wizards*. Xuất hiện cửa sổ Select Wizard hình 3.5.
- Chọn Create Index Wizard xuất hiện cửa sổ *Welcometo the Create Index Wizard*, như hình 3.39. Chọn Next.
- Xuất hiện cửa sổ *Select Database and Table (hình 3.40)*. Chọn CSDL và bảng dữ liệu cần tạo index.
    + Mục Database name: Chọn cơ sở dữ liệu
    + Mục Object name: Chọn bảng dữ liệu

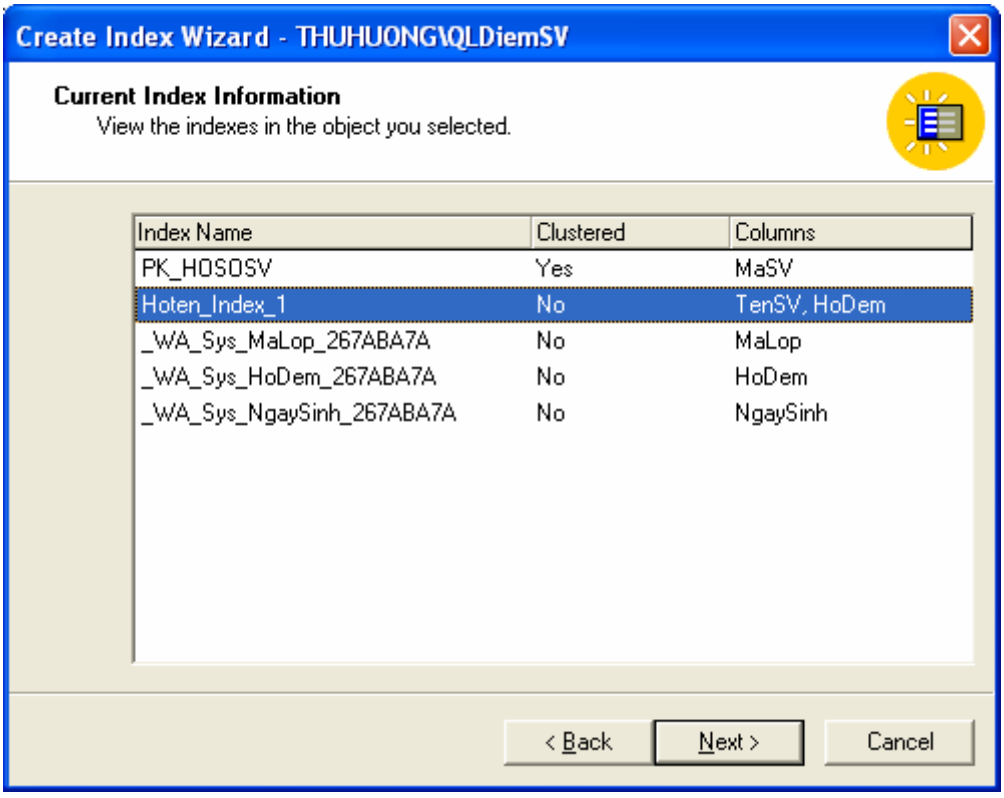
Sau đó chọn Next.

- Xuất hiện cửa sổ *Current index information (hình 3.41)* đưa thông tin các Index đã được tạo trước đó.
- Cửa sổ tiếp theo là cửa sổ *Select columns (hình 3.42).* Trong cửa sổ này ta chọn các cột để tạo chỉ mục. Chọn Next.
- Xuất hiện cửa sổ *Specify Index Option (hình 3.43).* Chỉ định các tham số cho Index như: Tạo chỉ mục duy nhất, tối ưu,.v.v…. Chọn Next.
- Xuất hiện cửa sổ Completing the Create Index Wizard (hình 3.44). Chọn thứ tự, đặt tên cho Index. Cuối cùng chọn Finish để kết thúc.

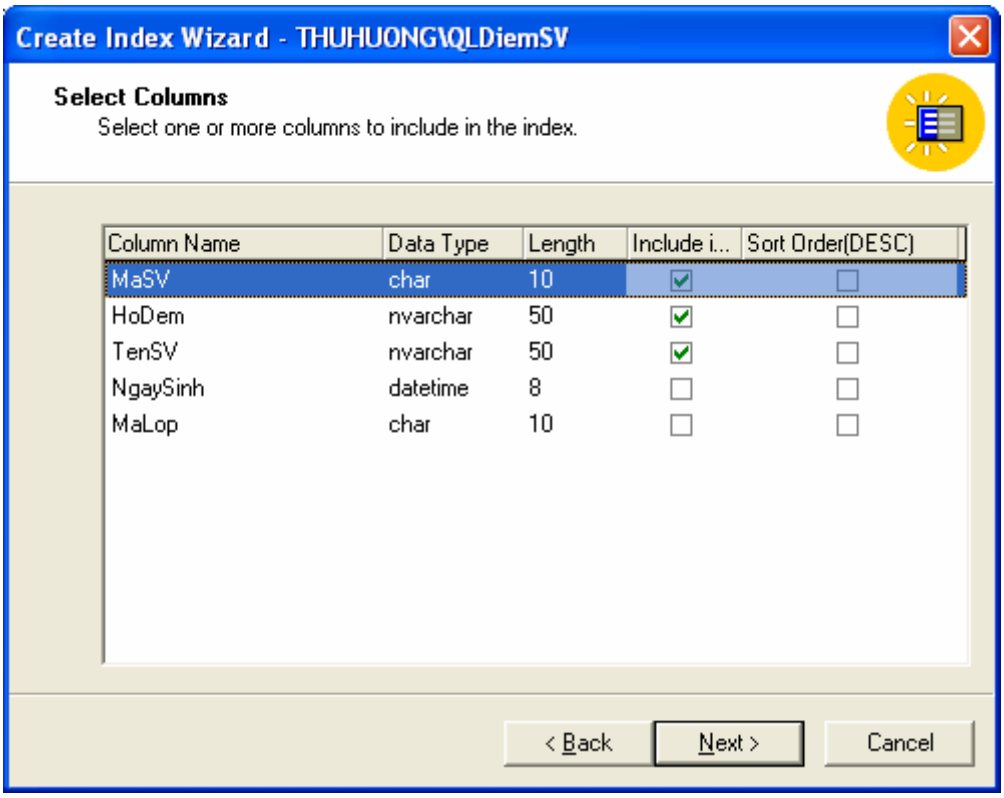
Hình 3.39.*Cửa sổ Welcometo the Create Index Wizard.*



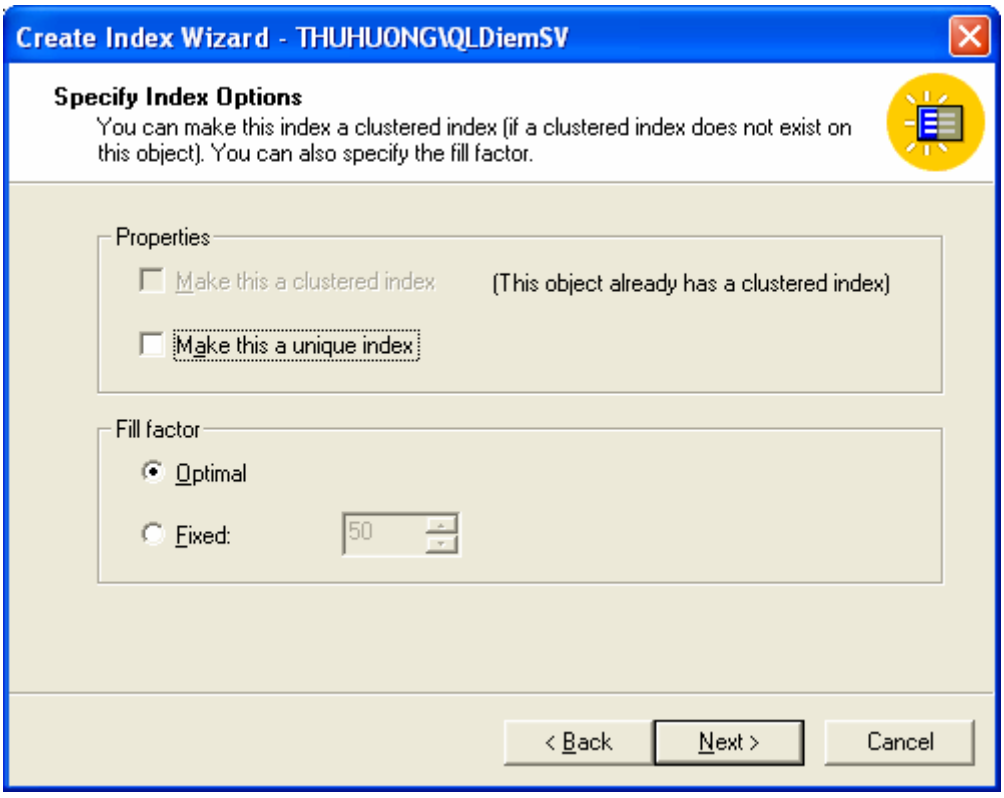
Hình 3.40.*Cửa sổ Select Database and Table.*



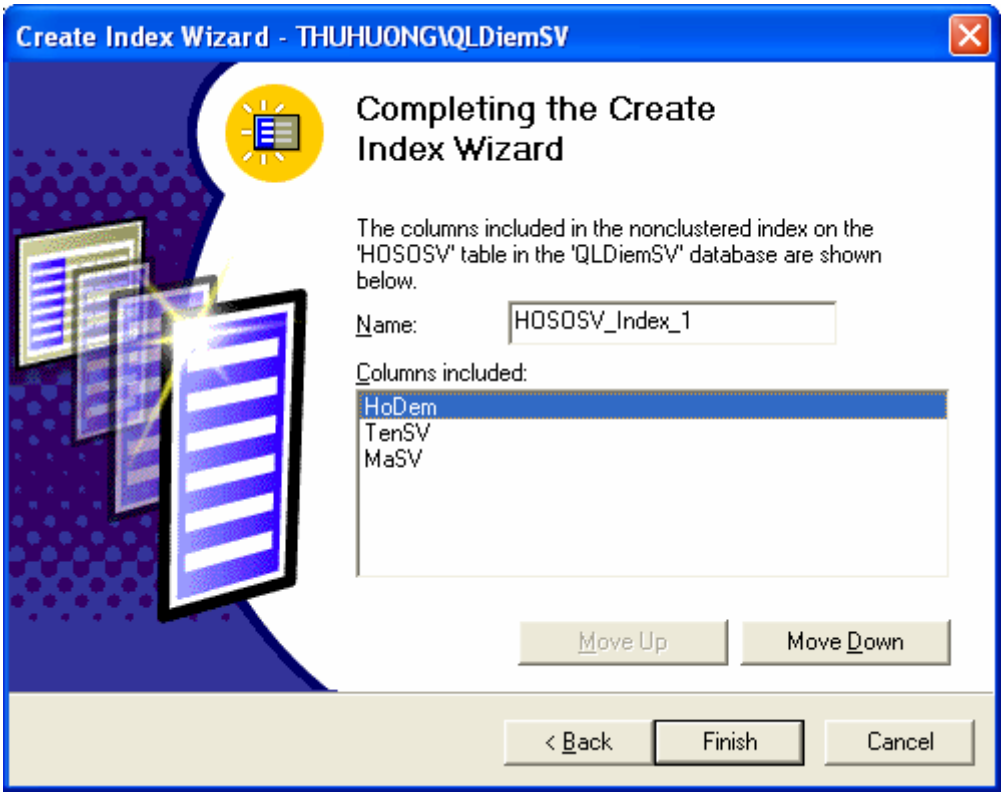
Hình 3.41.*Cửa sổ Current index information.*



Hình 3.42.*Cửa sổ select columns.*



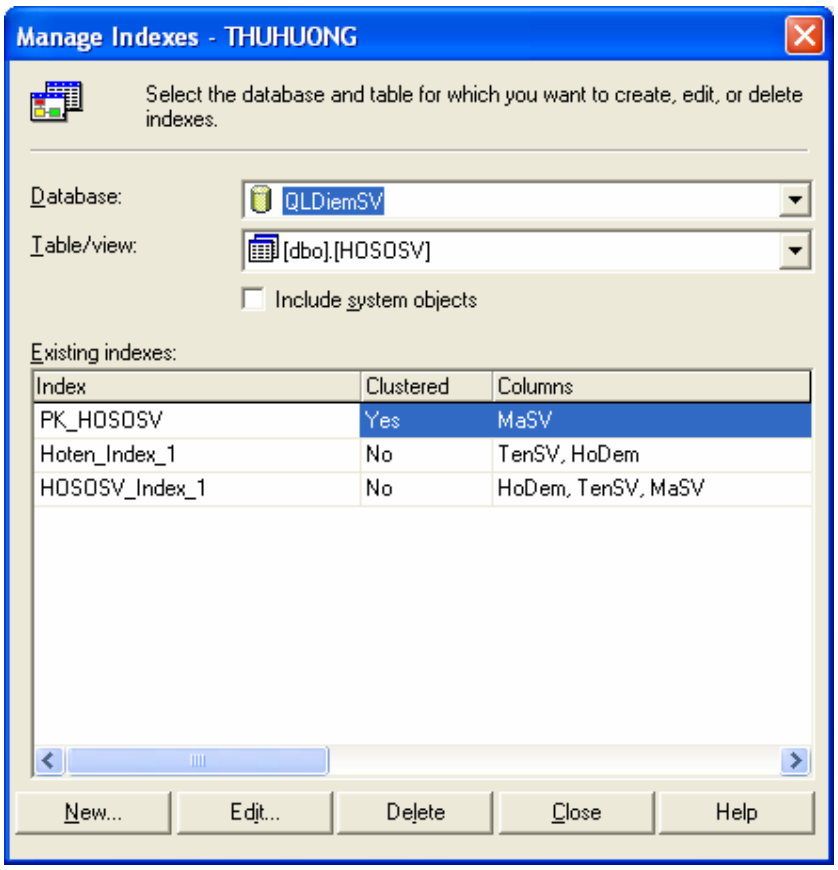
Hình 3.43. *Cửa sổ Specify Index Option.*



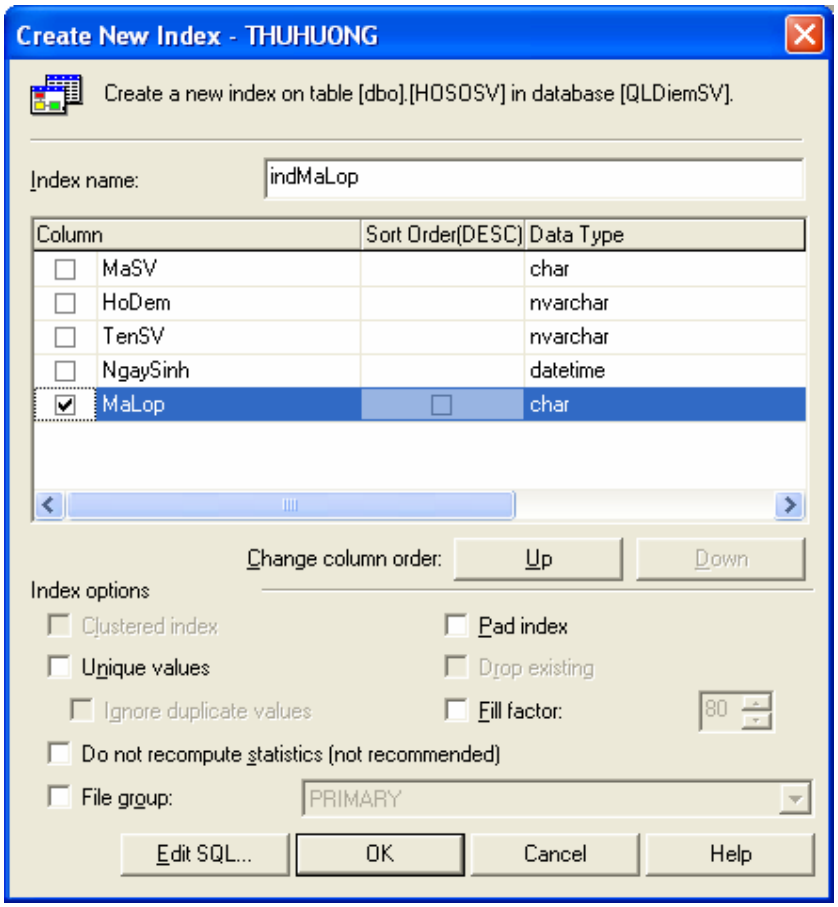
Hình 3.44. *Cửa sổ Completing the Create Index Wizard*

** Dùng Enterprise Manager:*

- Trong cửa sổ Enterprise Manager, mở rộng danh mục Database, mở rộng cơ sở dữ liệu muốn tạo Index, chẳng hạn CSDL QLDiemSV và chọn mục Tables. Sau đó right click lên bảng muốn tạo chỉ mục. Chọn All Task\Manage Indexes. Xuất hiện của sổ Manage Indexes như hình 3.45.
- Trong cửa sổ này chọn CSDL trong mục Database, chọn bảng hoặc View trong mục Table/view và sau đó click nút New xuất hiện cửa sổ Create New Index hình 3.46.
- Trong cửa sổ này ta nhập tên của Index trong mục Index name và chọn các trường của Index trong cột Column.



Hình 3.45. *Cửa sổ Manage Indexes*



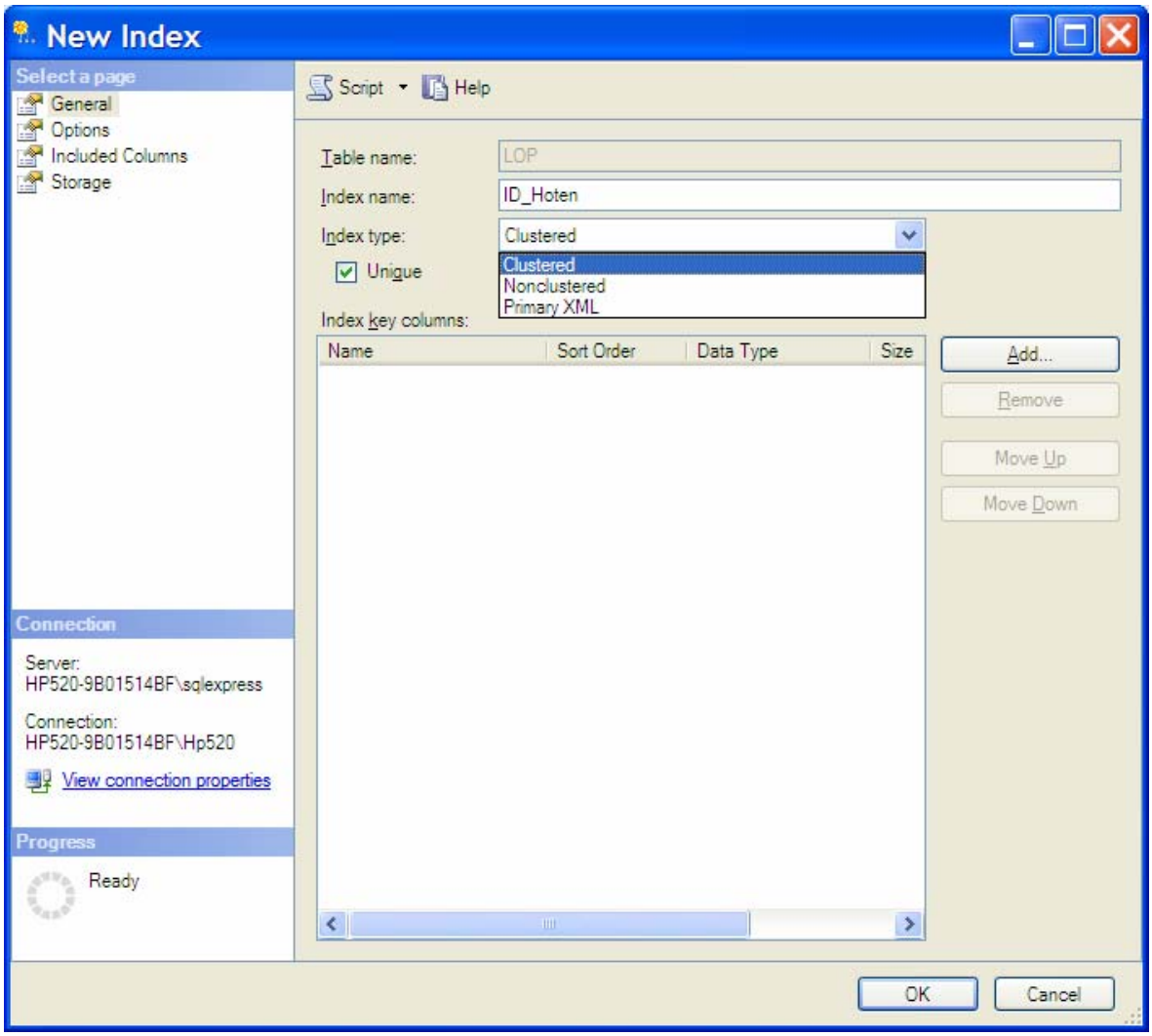
Hình 3.46. *Cửa sổ Create New Index*

Ngoài ra, trong cửa sổ 3.45 Manage Indexes ta có các nút thao tác quản lý chỉ mục (Index):

- Nút New: Dùng để thêm một Index mới.
- Nút Edit: Chọn Index đã có trong danh sách và chọn nút để Edit để thực hiện chỉnh sửa chỉ mục đã có.
- Nút Delete: Chọn Index đã có trong danh sách và chọn nút để Delete để thực hiện xóa chỉ mục đã chọn.

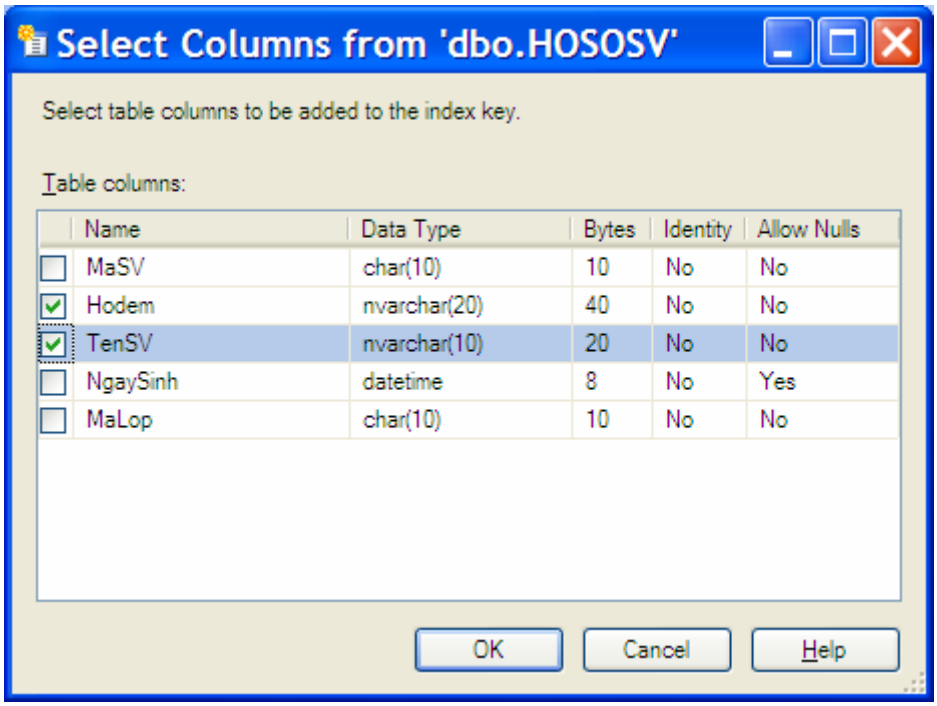
** Dùng SQL Server Management Studio*

- Trong SQL Server Management Studio, mở rộng danh mục Database, mở rộng cơ sở dữ liệu chứa bảng hoặc view muốn tạo chỉ số.



Hình 3.47. *Cửa sổ New Index*

- Mở rộng mục Table hoặc View, mở rộng bảng hoặc view muốn tạo Index. Right Click lên thư mục Indexes chọn New Index, xuất hiện cửa sổ New Index như hình 3.47. gồm các tham số:
  + Table name (hoặc View name đối với view): Tên bảng (hoặc tên view) mà tệp chỉ số được xây dựng trên đó.
  + Index name: Đặt tên tệp chỉ số muốn xây dựng.
  + Index type: Kiểu của index (Clustered, Nonclustered)
  + Unique: Tùy chọn cho phép tạo tệp chỉ số này có là tệp chỉ số duy nhất hay không?
  + Index key columns: Xác định các trường khóa của index bằng cách click nút Add xuất hiện cửa sổ Select Columns Hình 3.48, ta chọn các cột làm khóa cho Index.



Hình 3.48. *Cửa sổ Select Columns*

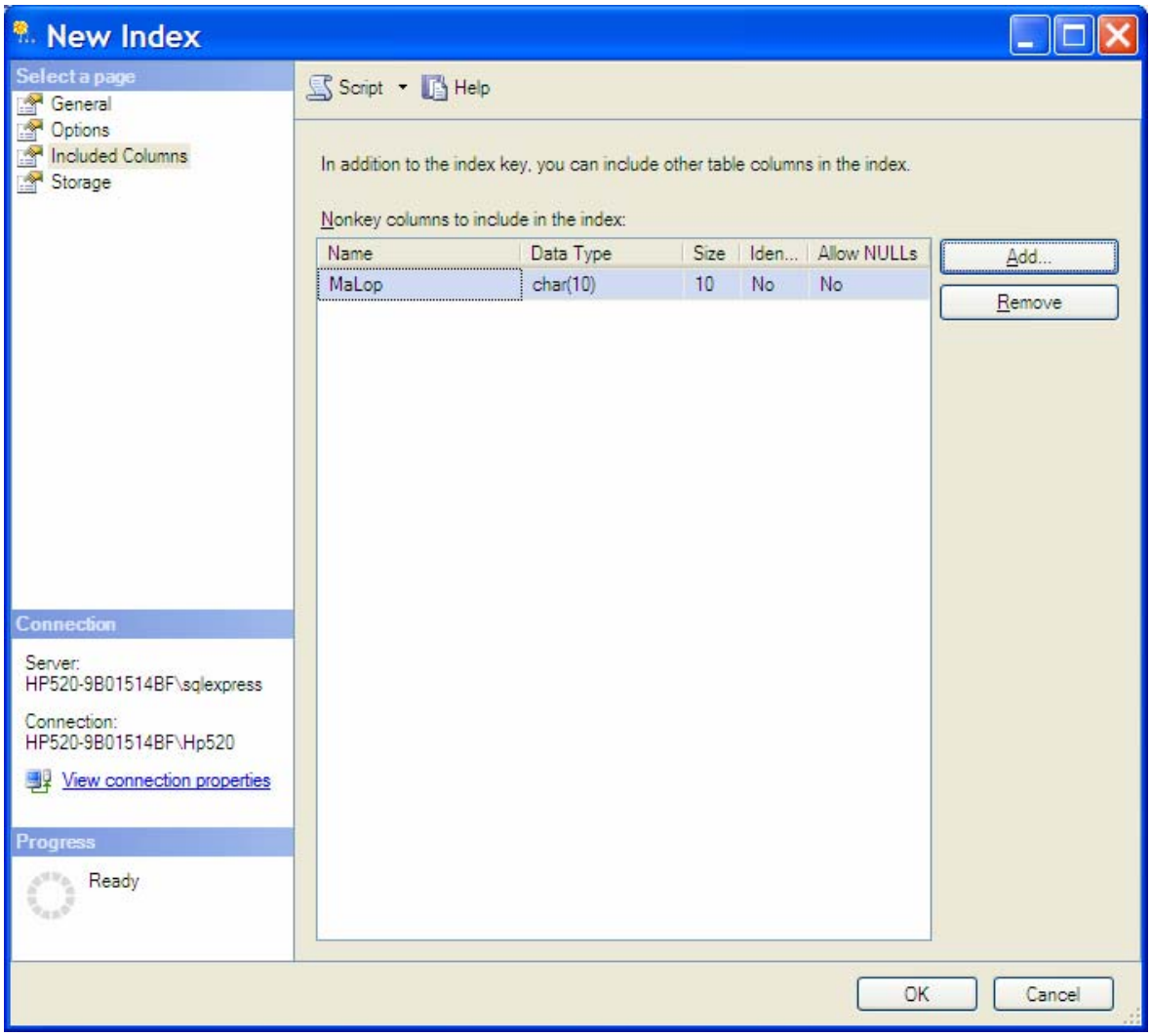
Chú ý: Trong SQL Server 2005, chúng ta có thể mở rộng chức năng của các tệp chỉ số phi liên cung (nonclustered indexes) bằng cách thêm các cột không là khóa (nonkey columns) vào các nút lá của cây nonclustered index. Các cột khóa được lưu trữ tại tất cả các mức còn các cột không khóa chỉ lưu trữ tại mức lá của index. Bằng việc thêm các cột không khóa, ta có thể tạo các

tệp chỉ số phi liên cung phủ nhiều truy vấn hơn bởi vì các cột không khóa có các lợi ích sau:

+ Chúng có thể là các cột có kiểu dữ liệu không được phép làm các cột khóa trong index.
+ Chúng không được Database Engine xét khi tính đến số các cột khóa của Index hay kích thước khóa của index.

Một index được bao gồm tất cả các cột không khóa có thể cải thiện đáng kể sự thực thi truy vấn khi tất cả các cột trong được bao gồm trong index (cả các cột khóa và không khóa của index).

Để thêm các cột không khóa ta chọn trang Included Columns (Hình 3.49). Sau đó click nút Add để xuất hiện cửa sổ Select Columns (Hình 3.48).



Hình 3.49. *Cửa sổ New Index*

** Dùng T - SQL*

- Trên SQL Server 2000:

```
CREATE [ UNIQUE][CLUSTERED|NONCLUSTERED]INDEX
index_name
    ON { table | view } ( column [ ASC | DESC ] [ ,...n ] )
[ WITH < index_option > [ ,...n] ]
[ ON filegroup ]

< index_option > :: =
    { PAD_INDEX |
        FILLFACTOR = fillfactor |
        IGNORE_DUP_KEY |
        DROP_EXISTING |
    STATISTICS_NORECOMPUTE |
    SORT_IN_TEMPDB
    }
```

- Trên SQL Server 2005:

```
CREATE [ UNIQUE ] [ CLUSTERED | NONCLUSTERED ] INDEX
index_name
    ON <object> ( column [ ASC | DESC ] [ ,...n ] )
    [ INCLUDE ( column_name [ ,...n ] ) ]
    [ WITH ( <relational_index_option> [ ,...n ] ) ]
    [ ON { filegroup_name|default }
    ]
[ ; ]

<object> ::=
{
    [ database_name. [ schema_name ] . | schema_name. ]
        table_or_view_name
}

<relational_index_option> ::=
{
    PAD_INDEX  = { ON | OFF }
  | FILLFACTOR = fillfactor
  | SORT_IN_TEMPDB = { ON | OFF }
  | IGNORE_DUP_KEY = { ON | OFF }
  | STATISTICS_NORECOMPUTE = { ON | OFF }
  | DROP_EXISTING = { ON | OFF }
  | ONLINE = { ON | OFF }
  | ALLOW_ROW_LOCKS = { ON | OFF }
  | ALLOW_PAGE_LOCKS = { ON | OFF }
  | MAXDOP = max_degree_of_parallelism
 }
```

Các tham số trong đó:

UNIQUE

Chỉ định tạo một unique index trên bảng hoặc trên view. Một clustered index trên view buộc phải là unique.

CLUSTERED

Chỉ định tạo chỉ mục liên cung.

NONCLUSTERED

Chỉ định tạo chỉ mục phi liên cung. Mặc định là chỉ mục NONCLUSTERED.

index_name

Là tên của tệp chỉ số.

column

Là tên cột hoặc các cột mà index dựa trên đó.

INCLUDE (column [,... n])

Chỉ định các cột không khóa được thêm vào mức lá của chỉ mục phi liên cung.

ON filegroup_name

Tạo index trên filegroup chỉ định. Nếu không có chỉ định này thì index sử dụng cùng filegroup mà table hoặc view dựa trên.

ON "default"

Tạo index dựa trên filegroup mặc định.

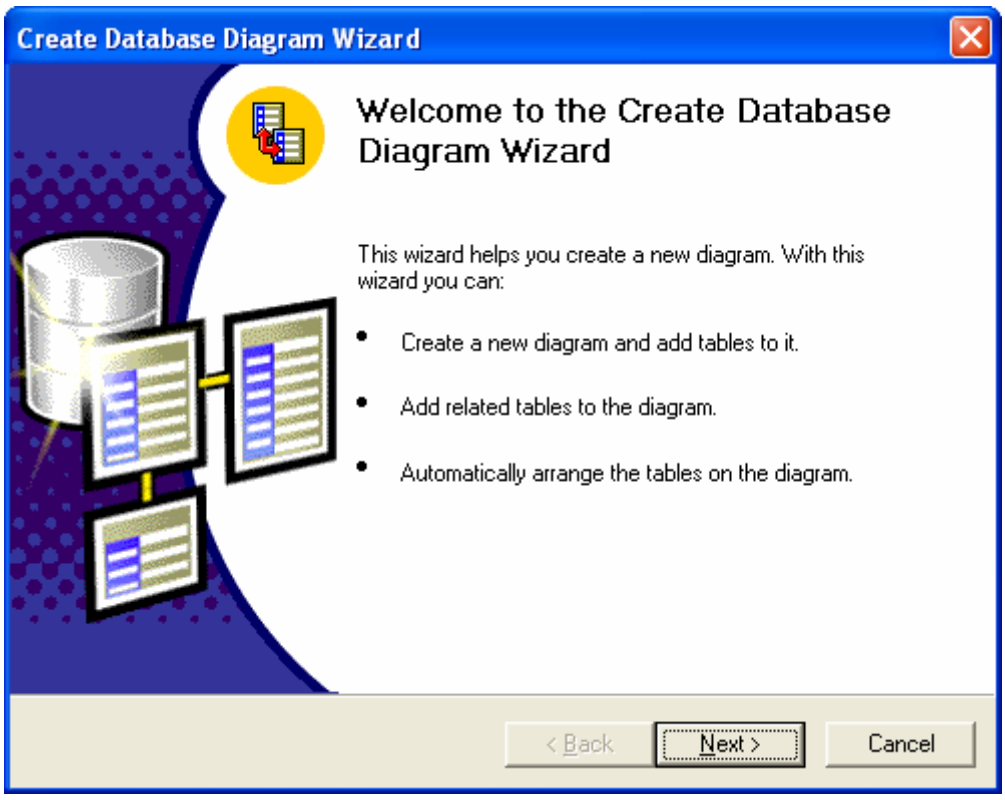
Ví dụ 3.7. Tạo Index trên bảng LOP của CSDL QLDiemSV

```
Use QLDiemSV
create Unique index indTenLopind On LOp(TenLop)
```

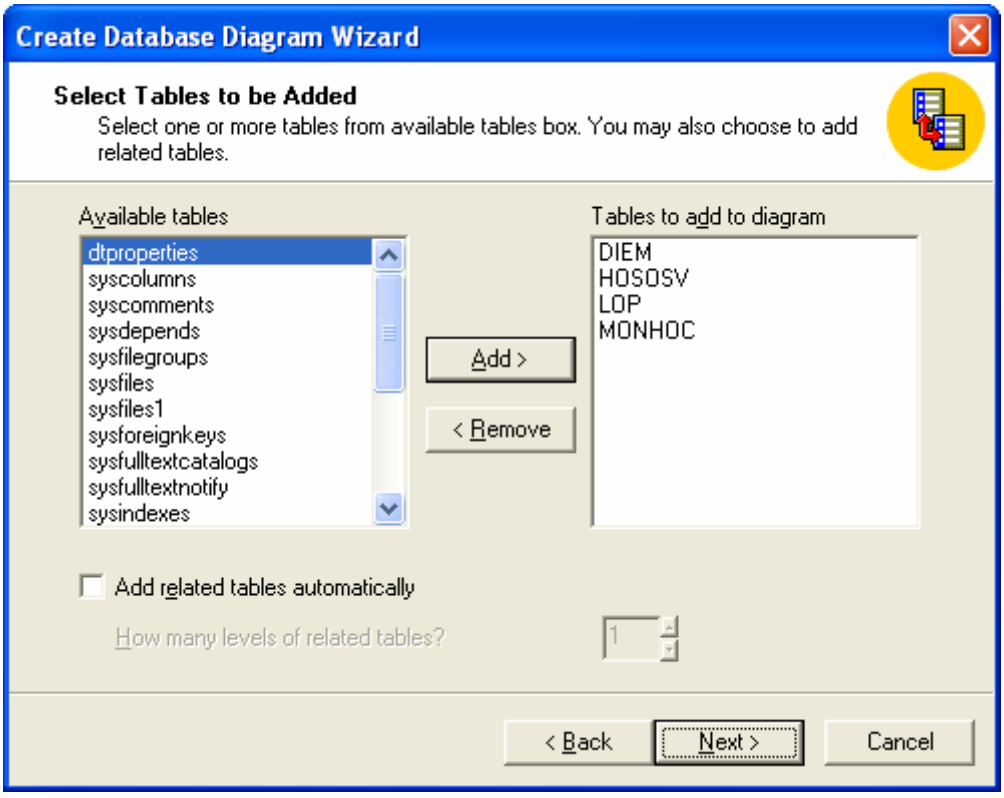
Ví dụ 3.8. Xây dựng lại Index `TenLop_ind` trên bảng LOP của CSDL QLDiemSV

```
Use QLDiemSV
create Unique index TenLop_ind
    On LOp(TenLop)
    With DROP_EXISTING
```

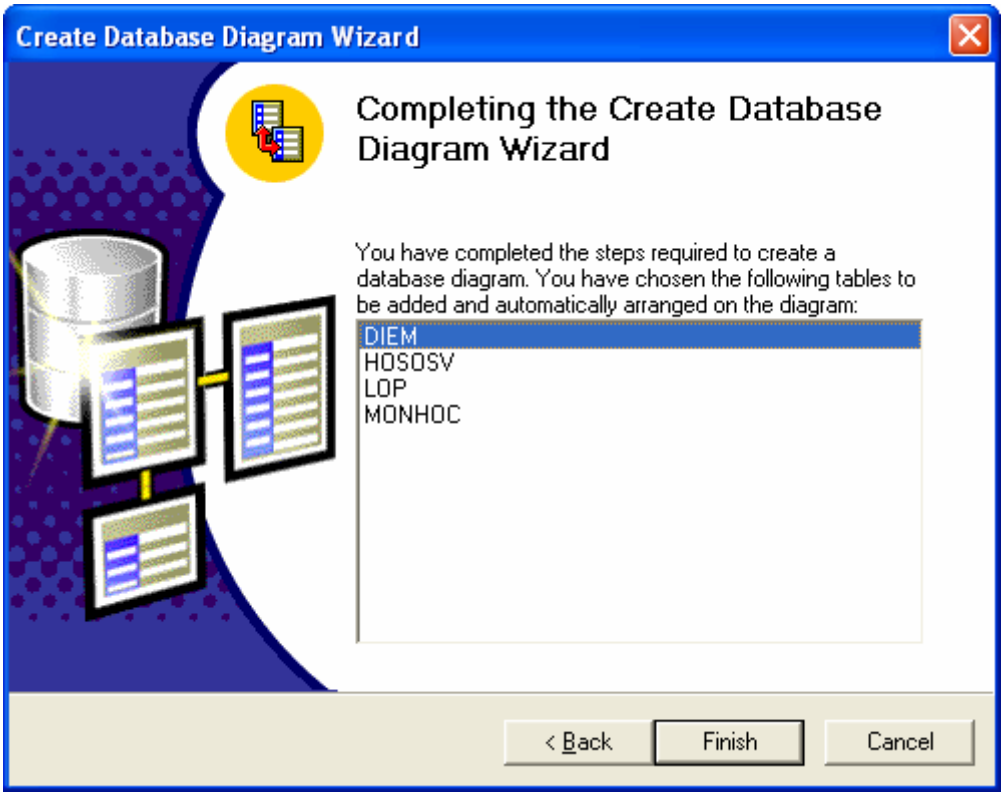
- Right Click lên Diagrams, chọn New Database Design. Xuất hiện cửa sổ Wellcome to Create Database Wizard (Hình 3.50). Chọn Next.
- Xuất hiện cửa sổ *Cửa sổ Select Tables to be Added* (hình 3.51). Chọn các bảng sẽ dùng để xây dựng lược đồ quan hệ. Chọn Next.
- Cửa sổ *Completing the Create Database Wizard* (hình 3.52) chọn Finish.
- Xuất hiện cửa sổ thiết kế Diagrams (hình 3.53). Trong cửa sổ này ta thực hiện thiết kế các mối quan hệ giữa các bảng bằng cách: Kéo và giữ chuột trên trường quan hệ của bảng này sau đó nhả vào trường quan hệ của bảng kia.
- Xuất hiện cửa sổ Relationship (hình 3.54). Trong cửa sổ này ta chọn trường quan hệ giữa hai bảng và các điều kiện cho mối quan hệ đang tạo giữa hai bảng này.



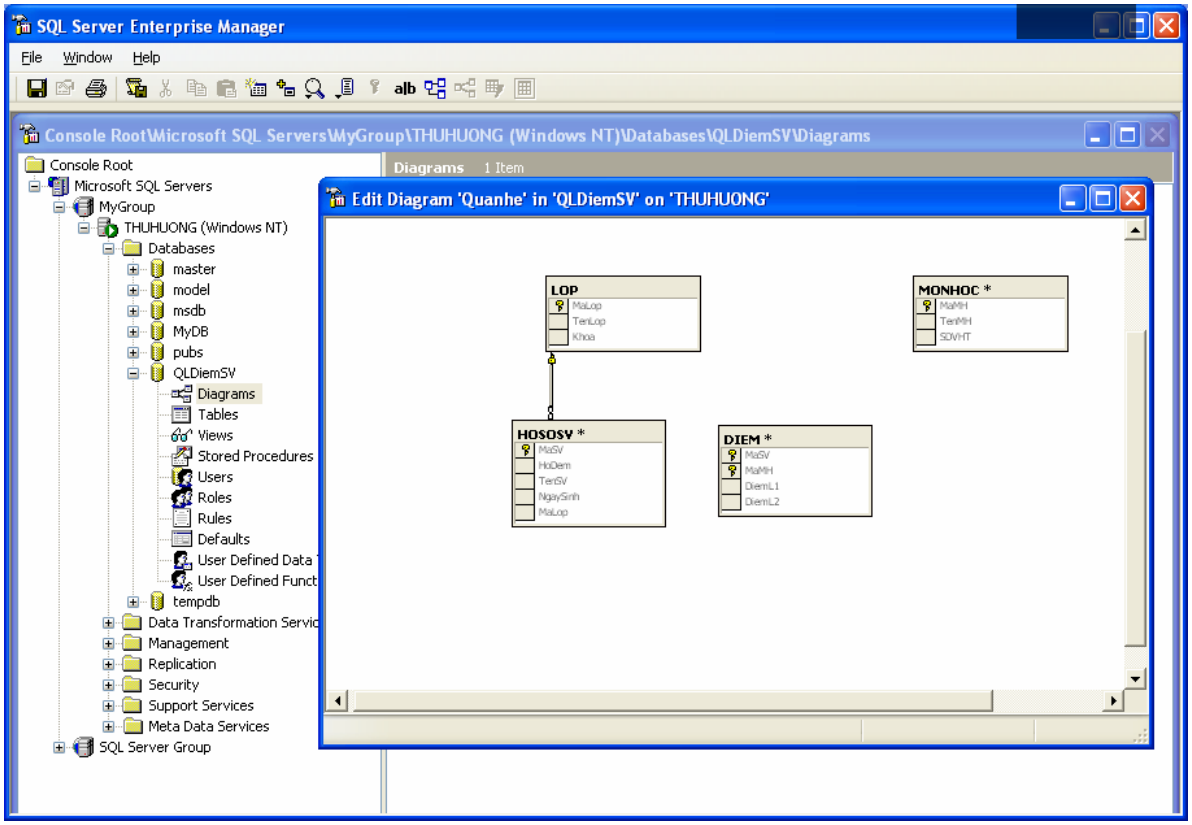
Hình 3.50. *Cửa sổ Wellcome to Create Database Wizard*



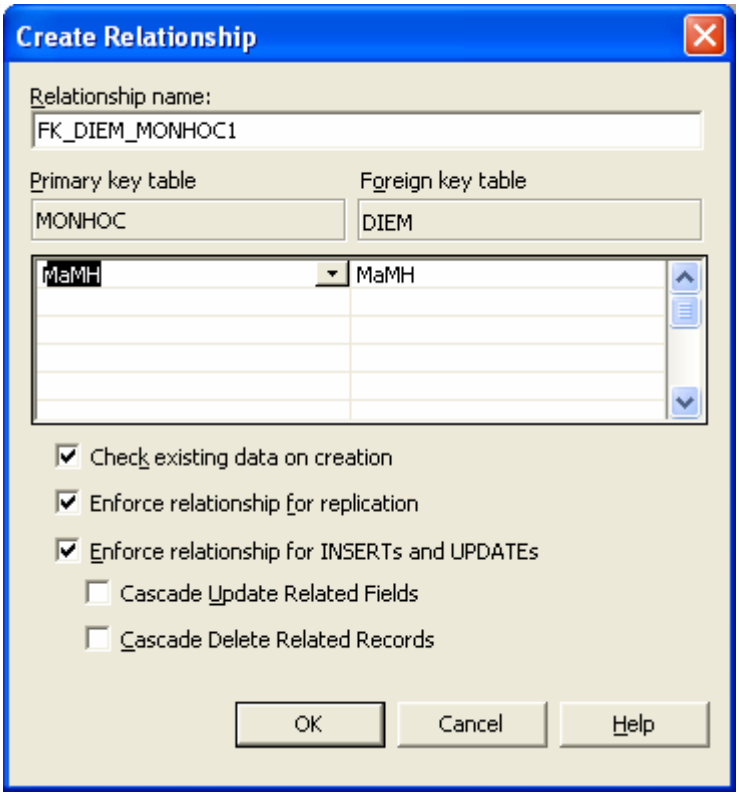
Hình 3.51. *Cửa sổ Select Tables to be Added*



Hình 3.52. *Cửa sổ Completing the Create Database Wizard*



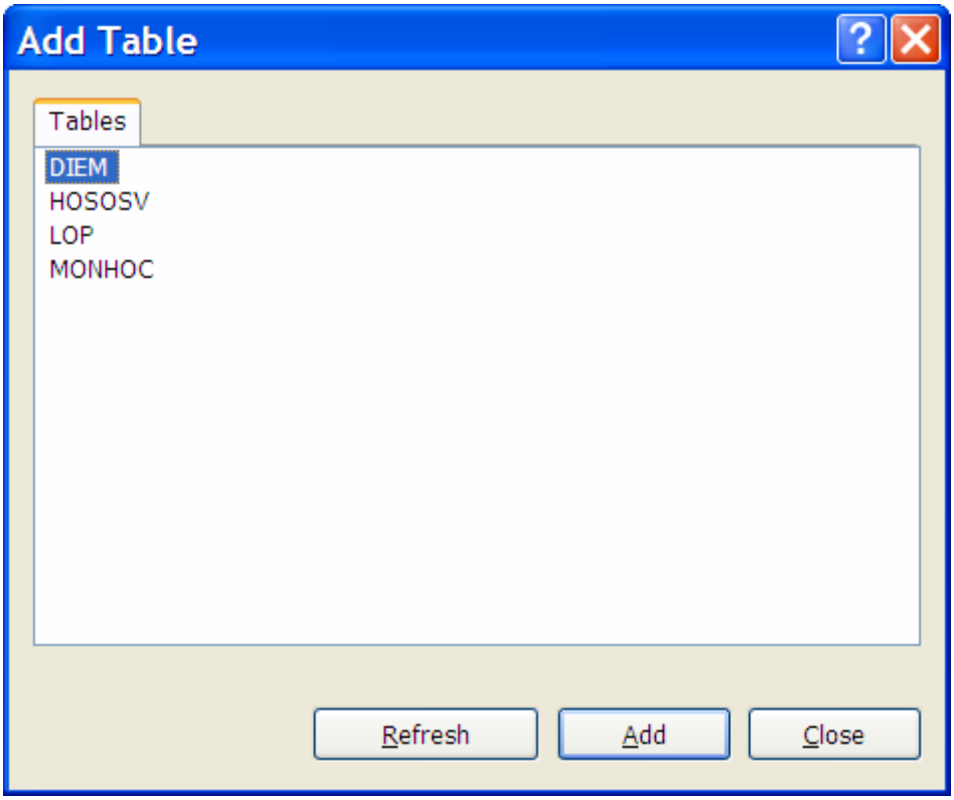
Hình 3.53. *Cửa sổ thiết kế Diagrams*



Hình 3.54. *Cửa sổ Create Relationship*

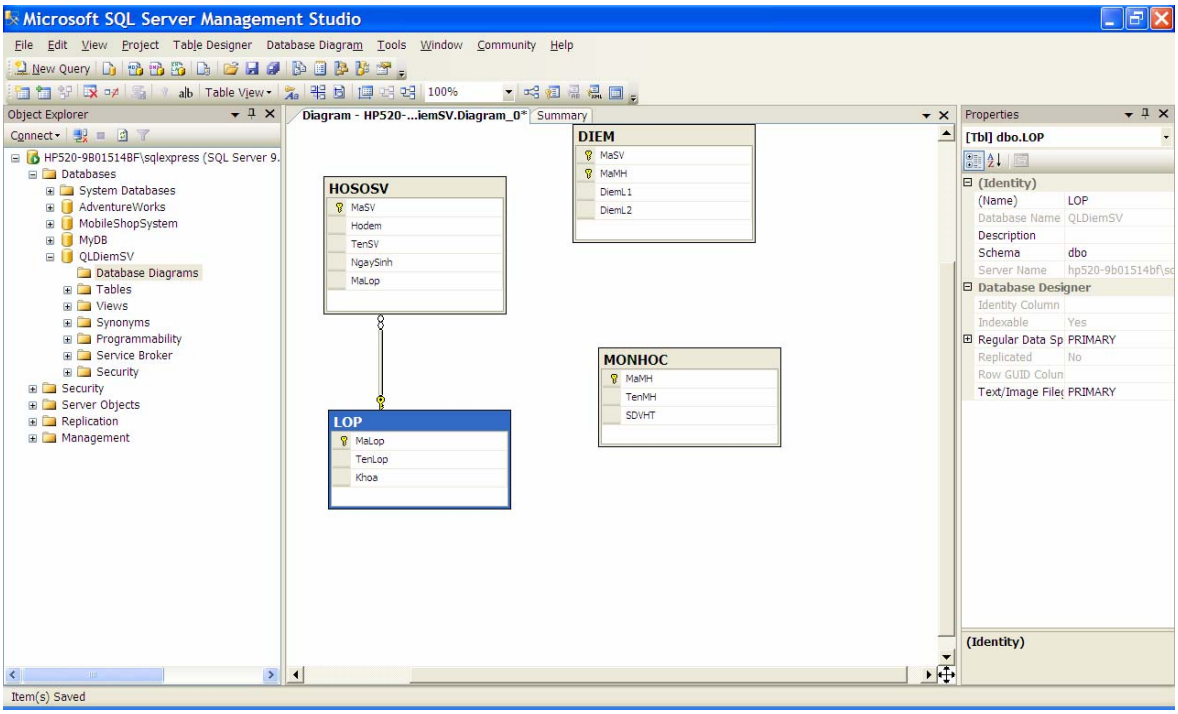
** Dùng SQL Server Management Studio*

- Trong SQL Server Management Studio, mở rộng danh mục Database, mở rộng cơ sở dữ liệu muốn tạo Database Diagrams. Right click và chọn New Database Diagram xuất hiện cửa sổ Add Table hình 3.55.

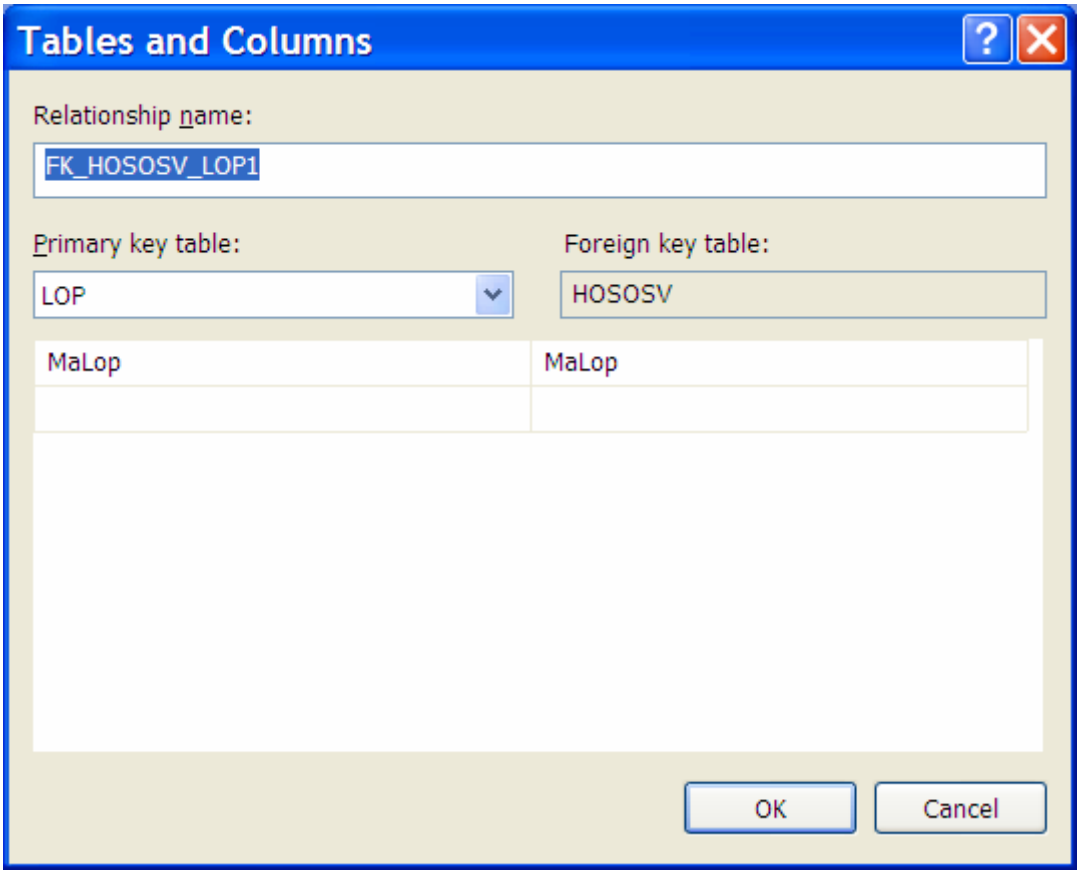


***Hình 3.55.** Cửa sổ Add Table*

- Chọn các bảng xây dựng lược đồ thông qua nút Add. Xuất hiện cửa sổ thiết kế Diagram (Hình 3.56).
- Ta thực hiện thiết kế mối quan hệ giữa các bảng trong cơ sở dữ liệu bằng việc kéo và giữ trường của bảng này thả sang trường tương ứng của bảng khác xuất hiện cửa sổ Table and Columns (Hình 3.57). Ta thực hiện điều chỉnh các tham số cho mối quan hệ đó.



Hình 3.56. *Cửa sổ thiết kế Diagram*



Hình 3.57. *Cửa sổ thiết kế Table and Columns*

b) Chỉnh sửa lược đồ

** Sử dụng Enterprise Manager:*

- Trong Enterprise Manager, mở rộng danh mục Database, mở rộng cơ sở dữ liệu muốn tạo lược đồ, chẳng hạn CSDL QLDiemSV và chọn mục Diagrams.
- Double Click lên lược đồ muốn chỉnh sửa. Xuất hiện cửa sổ Edit Diagram (Hình 3.53). Ta thực hiện các thao tác chỉnh sửa lược đồ trên cửa sổ này:
    + Thêm bảng mới: Right click và chọn Add table
    + Xóa bảng: Right click lên bảng xóa và chọn Remove Table from Diagram.
    + Xóa quan hệ: Right click lên quan hệ muốn xóa và chọn Delete Relationship from Database.
    + Chỉnh sửa lại quan hệ: Right click lên quan hệ muốn chỉnh sửa và chọn Properties. Thực hiện các chỉnh sửa trên cửa sổ này.

** Dùng SQL Server Management Studio*

- Trong SQL Server Management Studio, mở rộng danh mục Database, mở rộng cơ sở dữ liệu sửa đổi Database Diagrams. Right click vào Diagram muốn sửa đổi và chọn Modify xuất hiện cửa sổ thiết kế Database Diagram như hình 3.56 ta thực hiện sửa đổi trên cửa sổ này.

c) Xóa lược đồ

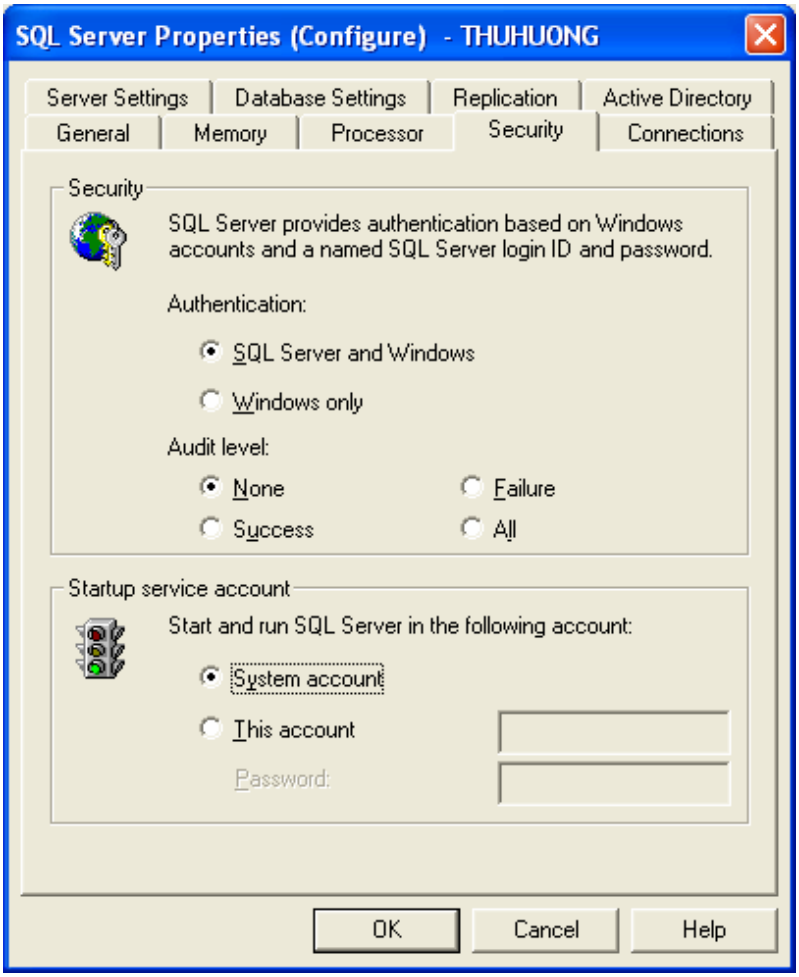
** Sử dụng Enterprise Manager:*

Trong Enterprise Manager, mở rộng danh mục Database, mở rộng CSDL và chọn mục Diagrams. Right click lên lược đồ muốn xóa và chọn Delete.

** Dùng SQL Server Management Studio*

Trong SQL Server Management Studio, mở rộng danh mục Database, mở rộng cơ sở dữ liệu sửa đổi Database Diagrams. Right click vào Diagram muốn sửa đổi và chọn Delete xuất hiện cửa sổ xác nhận xóa và chọn OK.

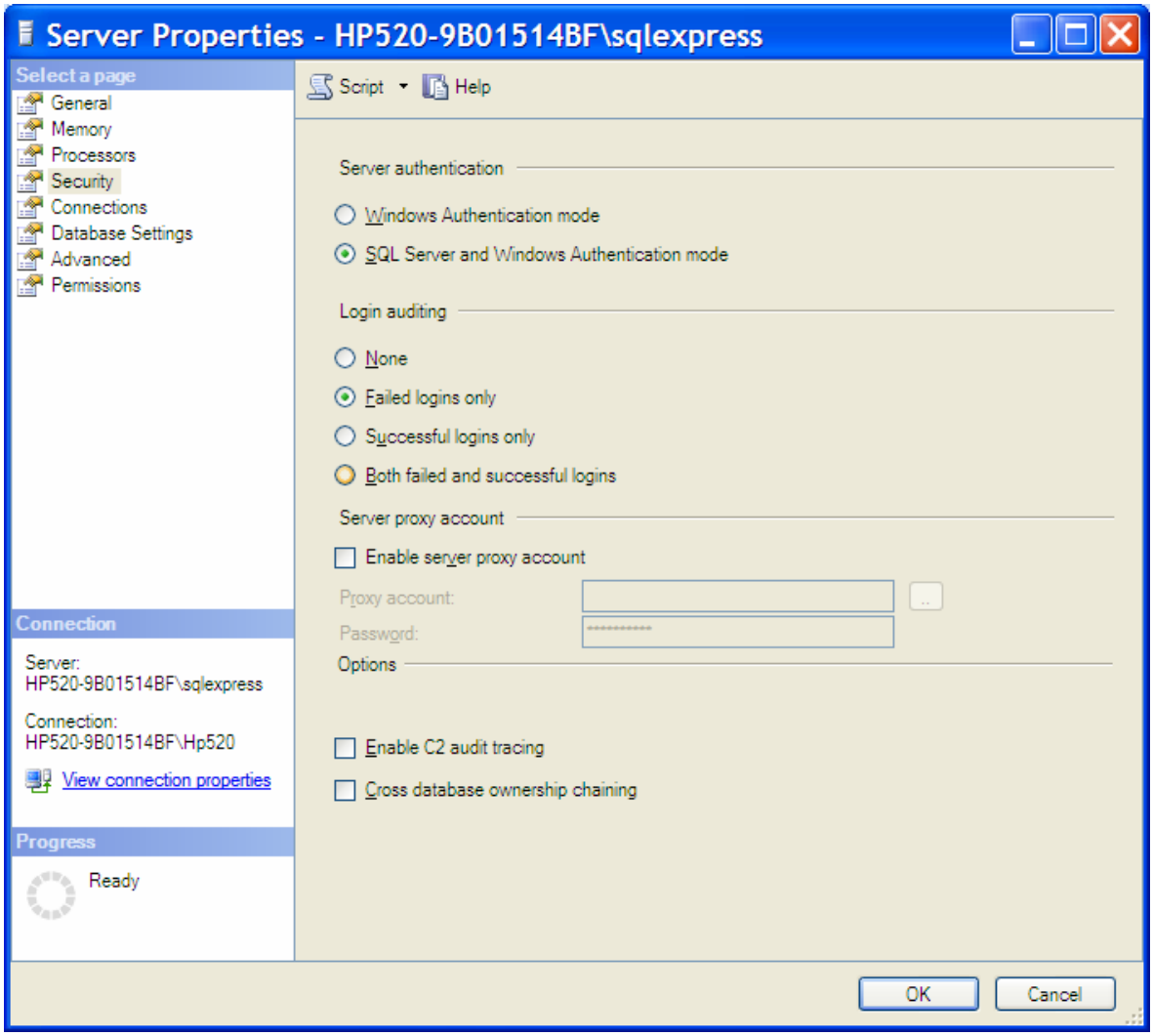
+ Vùng Security ta chọn chế độ xác thực: SQL Server and Windows hoặc Windows Only. Và các mức dò xét hành vị đăng nhập, tùy thuộc vào yêu cầu ảo mật. Có 4 mức cho sẵn đó là:
    - *None*: Không thực hiện dò xét hành vi,
    - *Success*: Ghi nhận tất cả các đăng nhập thành công,
    - *Failure*: Ghi nhận những đăng nhập không thành công,
    - *All*: ghi nhận tất cả các lần đăng nhập.
+ Trong vùng Startup service account: Chỉ ra tài khoản Windows được dùng khi SQL Server khởi động.
    + System account: các tài khoản hệ thống cục bộ được xây dựng sẵn.
    + This account: Chỉ ra một tài khoản cụ thể.



Hình 3.58. *Cửa sổ Properties*

** Đối với SQL Server 2005:*

+ Trong cửa sổ SQL Server Management Studio, right click vào thể hiện của SQL Server muốn thiết lập chế độ xác thực và chọn Properties xuất hiện cửa sổ Server Properties (Hình 3.59), chọn trang Security.



Hình 3.59. *Cửa sổ Server Properties*

+ Mục Server Authentication: Chọn chế độ xác thực Window (Windows Authentication mode) hoặc chế độ xác thực hỗn hợp (SQL Server and Windows Authentication mode)

+ Login Auditing: Kiểu ghi nhận thông tin login (None, Failed logins only, Successful login only, …)

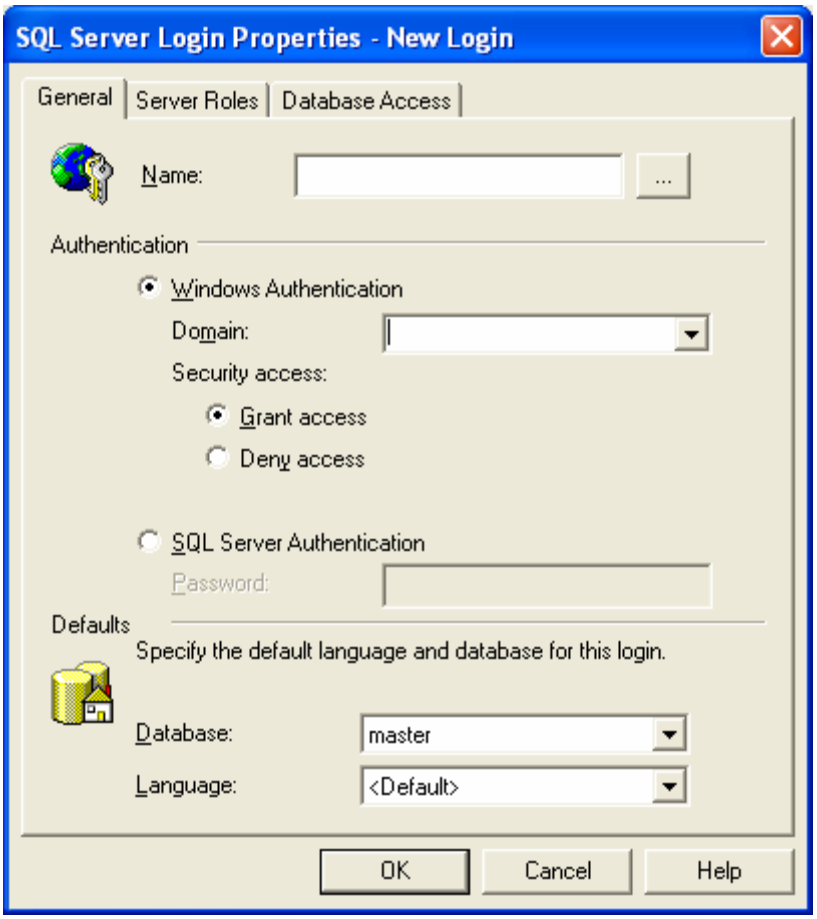
b) Người dùng và đăng nhập

Tài khoản dùng để kết nối tới SQL Server được gọi là tài khoản đăng nhập SQL Server. Cùng với tài khoản đăng nhập SQL Server, mỗi CSDL có một tài khoản người dùng ảo được gán với nó. Những tài khoản ảo này cung cấp một bí danh tới tài khoản đăng nhập SQL Server được gọi là tài khoản người dùng CSDL.

*** Tạo tài khoản đăng nhập SQL Server:**

➢ ***Dùng Enterprise manager:***

+ Trong cửa sổ Enterprise Manager, mở rộng server muốn tạo tài khoản đăng nhập và mở rộng mục Security. Right click lên Logins và chọn New Login để xuất hiện cửa sổ SQL Server Login Properties (hình 3.60).



Hình 3.60. *Cửa sổ New Login*

+ Trên tab General:

+ Name: Nhập tên tài khoản đăng nhập. nếu chọn chế độ xác thực bằng Window thì tên tài khoản đăng nhập phải là tài khoản đã tồn tại trong Windows.

+ Authentication: Chọn chế độ xác thực của Windows là Windows Authentication hay chế độ xác hỗn hợp SQL Server Authentication.

+ Default: Chọn CSDL và ngôn ngữ mặc định sẽ được dùng.

+ Tab Server Roles: Ở đây ta chọn nhóm quyền server cho đăng nhập mới bằng cách chọn các nhóm quyền trong danh sách. Click vào nút Properties để xem và sửa đổi nhóm quyền đã chọn. Nếu là tài khoản đăng nhập thường thì không cần cấp quyền server.

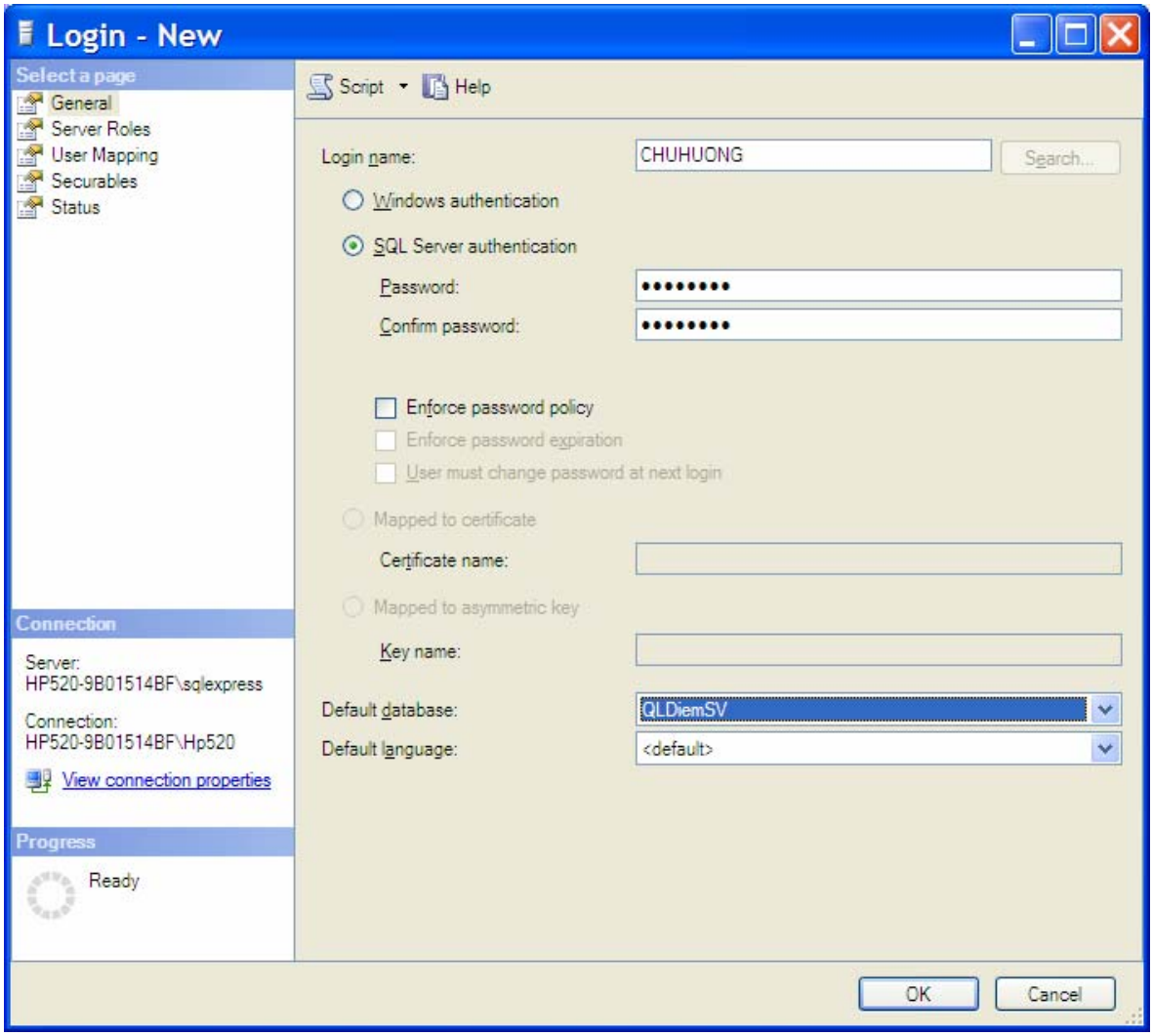
+ Tab Database Access: Cho phép chọn CSDL mà người dùng được phép truy cập.

➢ ***Dùng SQL Server Management Studio:***

+ Trong cửa sổ SQL Server Management Studio, mở rộng thể hiện server muốn tạo tài khoản đăng nhập và mở rộng mục Security. Right click lên Logins và chọn New Login để xuất hiện cửa sổ Login – New (hình 3.61).

+ Trang General có các lựa chọn:

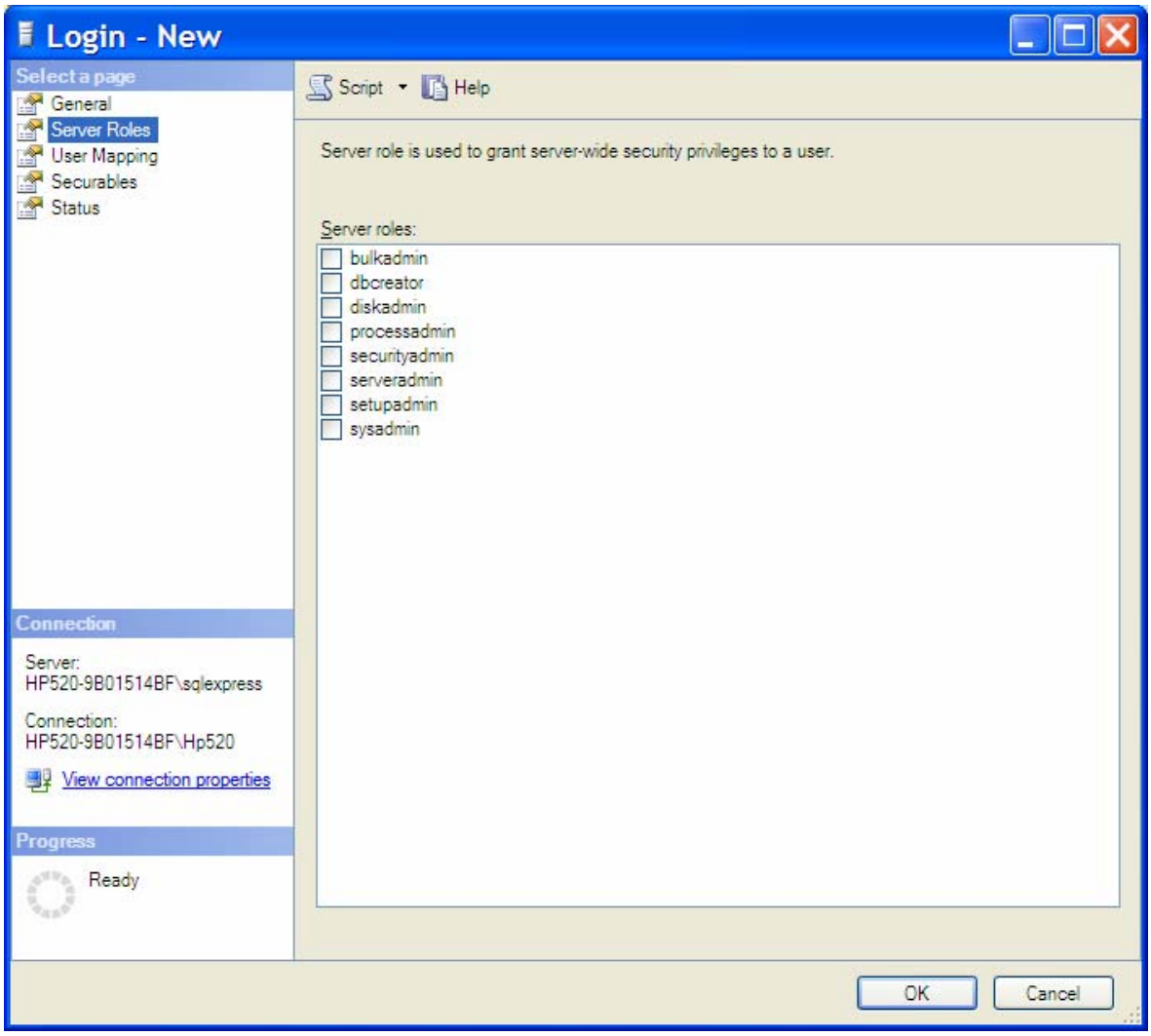
- Login Name: Nhập tên tài khoản đăng nhập. Nếu chọn chế độ xác thực bằng Window thì tên tài khoản đăng nhập phải là tài khoản đã tồn tại trong Windows.
- Default database: Chọn CSDL mặc định được sử dụng.
- Default language: Chọn ngôn ngữ mặc định.



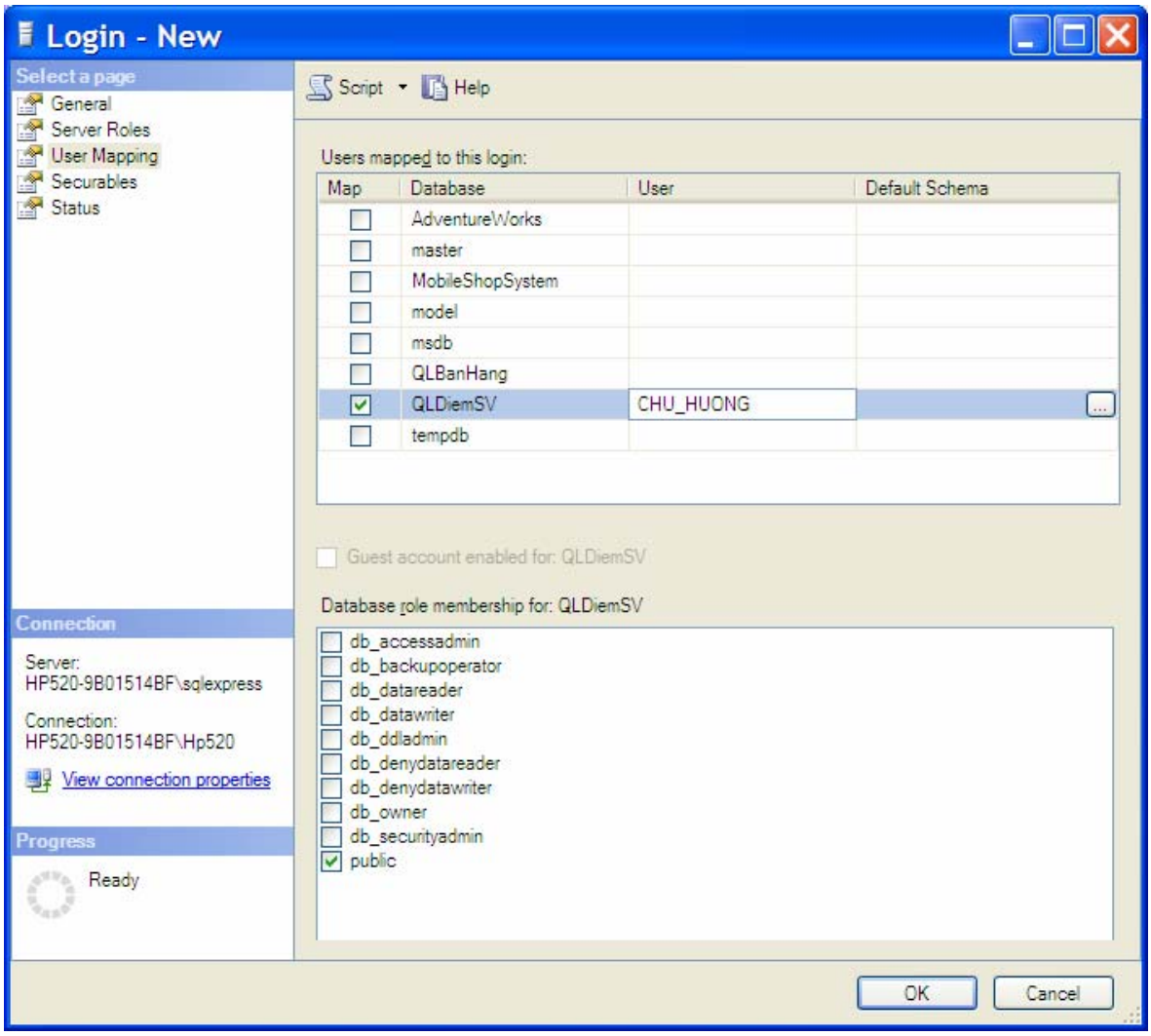
Hình 3.61. *Cửa sổ Login - New*

+ Trang Server Roles có các lựa chọn: Chọn nhóm quyền server cho đăng nhập mới bằng cách chọn các nhóm quyền trong danh sách (Hình 3.62).

+ Trang User Mapping (Hình 3.63): Cho phép chọn CSDL mà người dùng được phép truy cập.



Hình 3.62. *Cửa sổ Login - New*



Hình 3.63. *Cửa sổ Login - New*

➢ ***Dùng T-SQL:***

- Ta có thể tạo tài khoản đăng nhập bằng thủ tục sp_addlogin hoặc sp_grandlogin.

  + Thủ tục sp_addlogin chỉ có thể thêm người dùng được xác thực bằng SQL Server.

  + Thủ tục sp_grandlogin có thể thêm người dùng được xác thực bằng Windows.

Ví dụ 3.11. Tạo tài khoản đăng nhập *Huongct* với Password là *'abcd'* và CSDL mặc định là *'QLDiemSV'*

```
EXEC sp_addlogin  'Huongct1','abcd','QLDiemSV'
```

Ví dụ 3.12. Tạo tài khoản đăng nhập *Huongct* với chế độ xác thực Windows.

```
EXEC sp_grantlogin 'THUHUONG\Huongct'
```

- Ngoài ra, đối với SQL Server 2005 ta có thể sử dụng cú pháp T-SQL sau:

+ Tạo Login

```
CREATE LOGIN login_name { WITH PASSWORD =
'password' [,<option_list>[ ,... ]] | FROM
WINDOWS [ WITH <windows_options> [ ,... ]]}

<option_list> ::=
    DEFAULT_DATABASE = database
    | DEFAULT_LANGUAGE = language
    | CHECK_EXPIRATION = { ON | OFF}
    | CHECK_POLICY = { ON | OFF}
    [ CREDENTIAL = credential_name ]

<windows_options> ::=
    DEFAULT_DATABASE = database
    | DEFAULT_LANGUAGE = language
```

+ Sửa Login

```
ALTER LOGIN login_name
    {
    <status_option>
    | WITH <set_option> [ ,... ]
    }

<status_option> ::=
        ENABLE | DISABLE

<set_option> ::=
    PASSWORD = 'password'
    [
            OLD_PASSWORD = 'oldpassword'
    ]
    | DEFAULT_DATABASE = database
    | DEFAULT_LANGUAGE = language
    | NAME = login_name
    | CHECK_POLICY = { ON | OFF }
    | CHECK_EXPIRATION = { ON | OFF }
```

+ Xóa Login

```
DROP LOGIN login_name
```

Ví dụ 3.13. Tạo các login

- Tạo Login HUONGCT dùng chế độ xác thực SQL

```
USE  master
CREATE LOGIN HUONGCT WITH PASSWORD = '123456';
GO
```

- Tạo Login [HP520-9B01514BF\Hp520] từ domain account Windows.

```
CREATE LOGIN [HP520-9B01514BF\Hp520] FROM WINDOWS;
GO
```

- Sửa Login HUONGCT

```
USE  master
ALTER LOGIN HUONGCT WITH PASSWORD =
'12102006',DEFAULT_DATABASE = QLDiemSV;
```

- Xóa Login HUONGCT

```
USE  master
DROP LOGIN HUONGCT
```

➢ ***Dùng Wizard trong SQL Server 2000:***

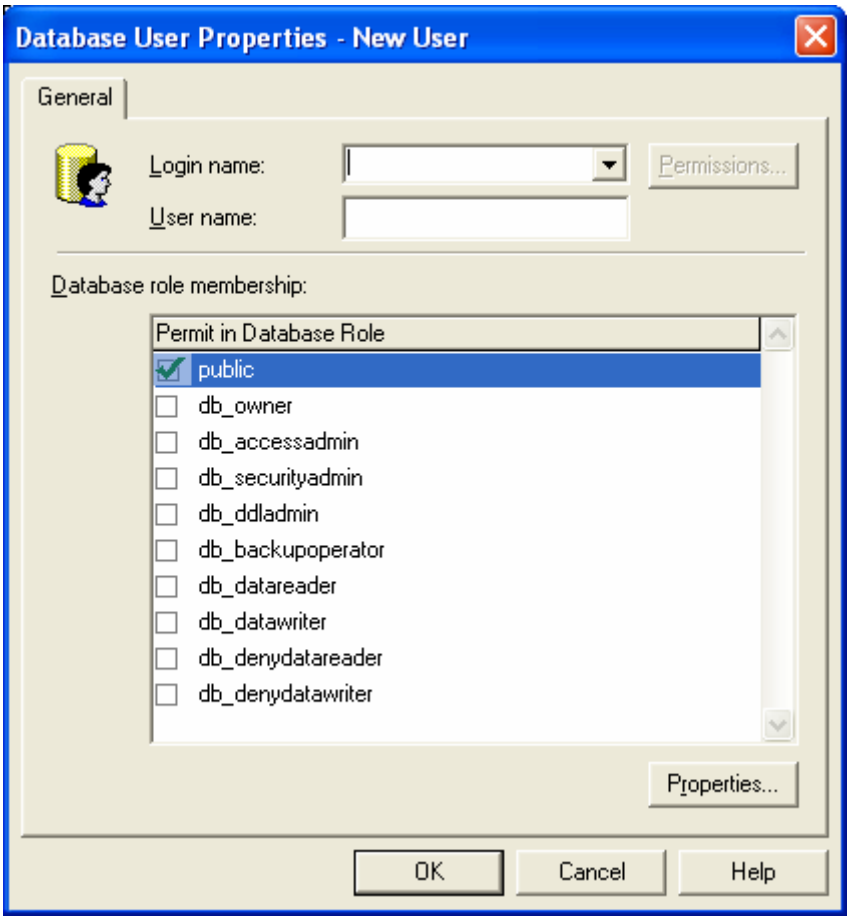
Vào Tools\Wizard xuất hiện cửa sổ hình 3.5. Chọn mục Database và chọn Create Login Wizard. Sau đó thực hiện theo sự chỉ dẫn của trình Wizard.

*** Tạo người dùng SQL Server:**

Để tạo người dùng SQL Server, trước hết ta phải tạo đăng nhập SQL Server cho người dùng đó vì tên người dùng tham chiếu đến tên đăng nhập.

➢ ***Dùng Enterprise manager:***

+ Trong cửa sổ Enterprise Manager, mở rộng mục Database. Right click CSDL muốn tạo người dùng và chọn New\Database User xuất hiện cửa sổ New User (hình 3.64).



Hình 3.64. *Cửa sổ New User*

+ Nhập tên đăng nhập hợp lệ trong danh sách các tên đăng nhập của hộp combo Login Name và nhập tên người dùng mới vào hộp User Name (Mặc định SQL Server tự điền tên User Name trùng tên Login Name, ta có thể thay đổi tên này)

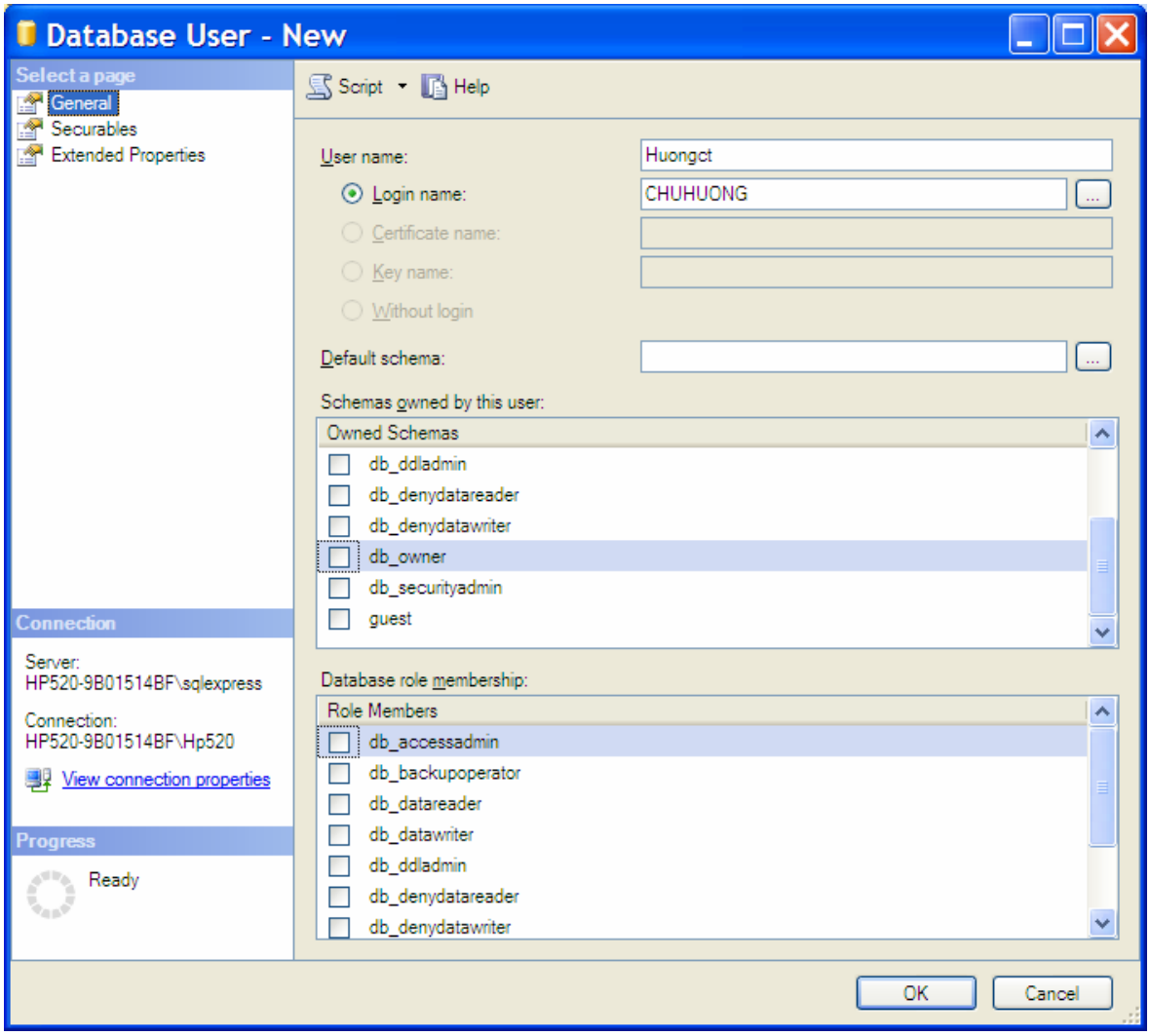
+ Chọn nhóm quyền CSDL mà người dùng mới này là thành viên, sau đó chọn OK.

➢ ***Dùng SQL Server Management Studio:***

+ Trong cửa sổ SQL Server Management Studio, mở rộng thể hiện server và mục Database. Mở rộng mục Security của cơ sở dữ liệu muốn tạo người dùng, Right click lên Users và chọn New User để xuất hiện cửa sổ Database User – New (hình 3.65).

+ Trang General có các lựa chọn:

- User name: Nhập tên người dùng
- Login name: Nhập tên hoặc chọn Login mà người dùng này ánh xạ đến.



Hình 3.65. *Cửa sổ Database User - New*

➢ ***Dùng T-SQL:***

- Ta có thể tạo người dùng mới bằng thủ tục sp_adduser.
- Đối với SQL Server 2005 ta có thể dùng cú pháp sau:

+ Tạo User

```
CREATE USER user_name
    { { FOR | FROM } LOGIN login_name |
WITHOUT LOGIN }
    [ WITH DEFAULT_SCHEMA = schema_name ]
```

+ Sửa User

```
ALTER USER user_name
     WITH <set_item> [ ,...n ]
<set_item> ::=
     NAME = new_user_name
     | DEFAULT_SCHEMA = schema_name
```

+ Xóa User

```
DROP USER user_name
```

Ví dụ 3.14. Tạo các người dùng mới là *Huong* với tên đăng nhập *Huongct* trên CSDL QLDiemSV

```
USE QLDiemSV
Go
sp_adduser 'Huongct','Huong'
```

Ví dụ 3.15. Tạo các người dùng mới trùng với tên đăng nhập *Huongct* trên CSDL QLDiemSV sử dụng xác thực của Windows

```
USE QLDiemSV
Go
sp_adduser 'THUHUONG\Huongct'
```

Ví dụ 3.16. Tạo các User

```
CREATE USER Huongct WITHOUT LOGIN
CREATE USER ChuHuong FOR LOGIN CHUHUONG
```

3.3.1.2. Quản lý nhóm quyền CSDL

Các nhóm quyền CSDL được thiết kế cho phép các nhóm những người dùng nhận các quyền CSDL giống nhau mà không cần phải cấp quyền một cách riêng biệt cho từng người dùng.

a) Các nhóm quyền Server cố định

Một số nhóm quyền ở cấp server đã được định nghĩa trước tại thời điểm cài đặt SQL Server. Những nhóm quyền cố định này được dùng để cấp quyền cho người quản trị CSDL. Các nhóm quyền server cố định được liệt kê trong danh sách sau:

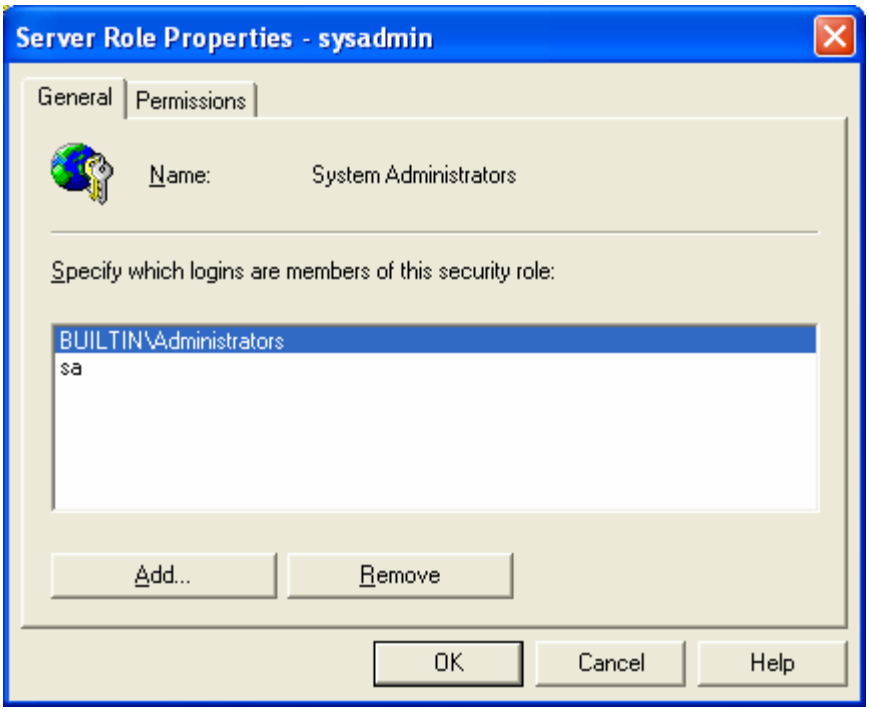
+ bulkadmin: Có thể thực thi lệnh BULK INSERT để thêm lượng lớn dữ liệu vào bảng.

+ dbcreator: Có thể tạo và sửa đổi CSDL.
+ diskadmin: Có thẻ quản lý các tập tin trên đĩa.
+ processadmin: Có thể quản lý các quá trình của SQL Server.
+ securityadmin: Có thể quản lý đăng nhập và tạo các quyền CSDL.
+ serveradmin: Có thể thiết lập bất kỳ tùy chọn server nào và có thể đóng CSDL.
+ setupadmin: Có thể quản lý các server liên kết và có thể đóng CSDL.
+ sysadmin: Có thể thực hiện bất kỳ hoạt động server nào.

Thêm người dùng vào các nhóm quyền server cố định.

➢ **Dùng Enterprise Manager**

+ Trong cửa sổ Enterprise Manager, mở rộng server và mở rộng mục Security. Sau đó chọn server roles và right click lên nhóm quyền server muốn thêm người dùng, chẳng hạn chọn System Administrators xuất hiện cửa sổ hình 3.66.

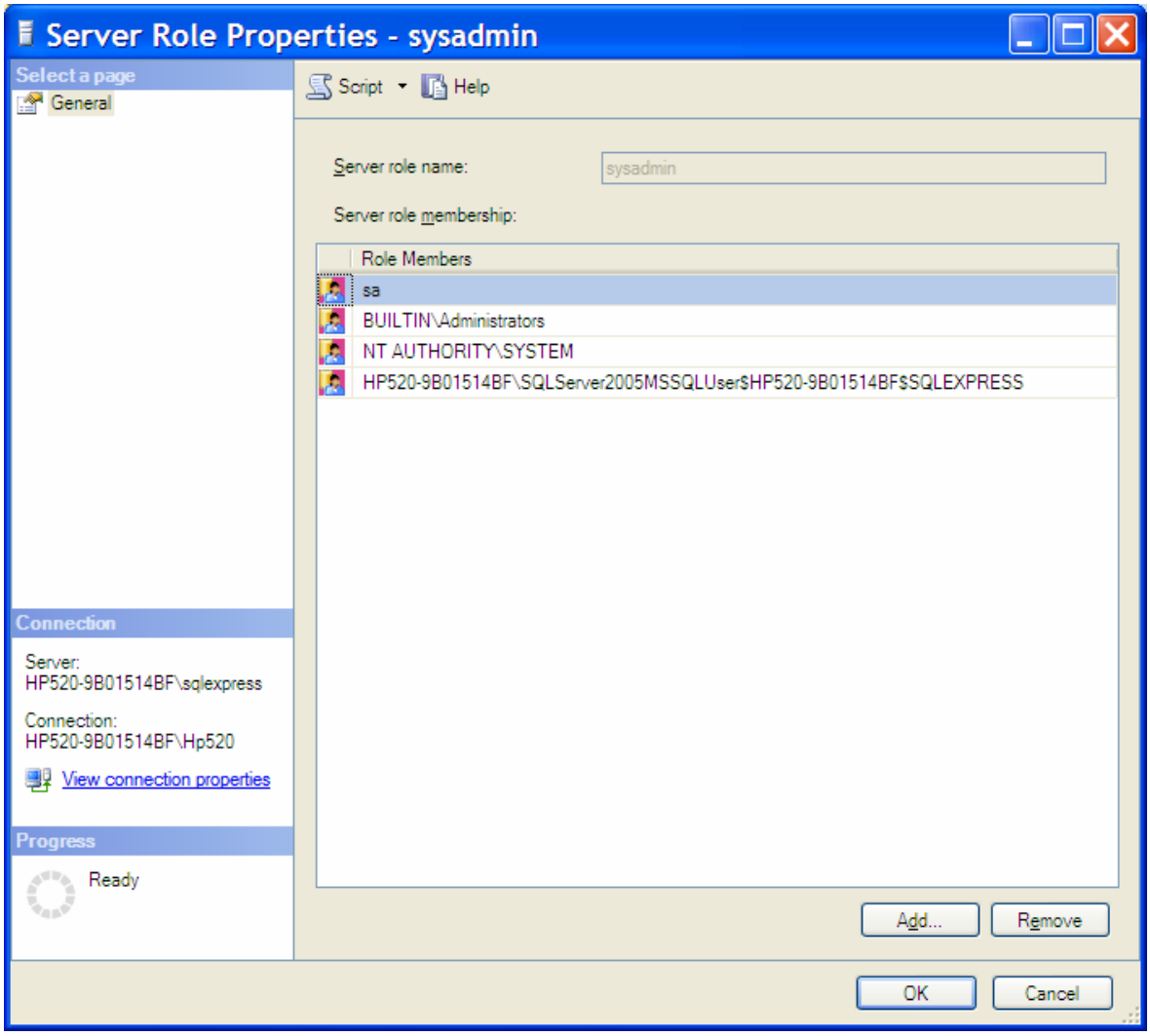


Hình 3.66. *Cửa sổ Server Role Propeties*

+ Click nút Add để thêm người dùng vào trong nhóm.

➢ **Dùng SQL Server Management Studio**

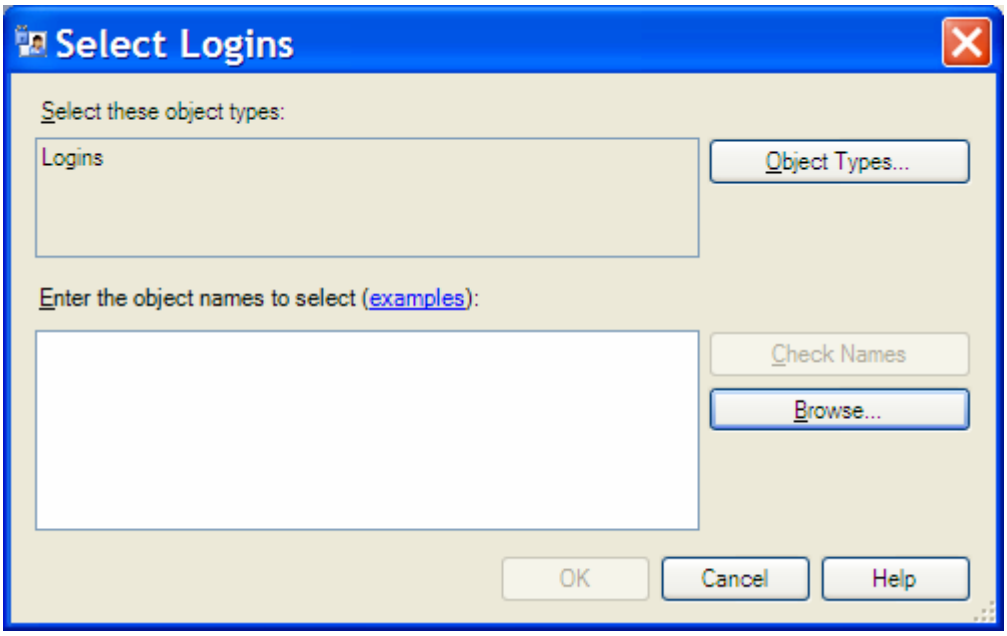
+ Trong cửa sổ SQL Server Management Studio, mở rộng mục Security ở cấp Server. Sau đó chọn Server roles và right click lên lên nhóm quyền server muốn thêm người dùng, chẳng hạn chọn System Administrators xuất hiện cửa sổ hình 3.67.



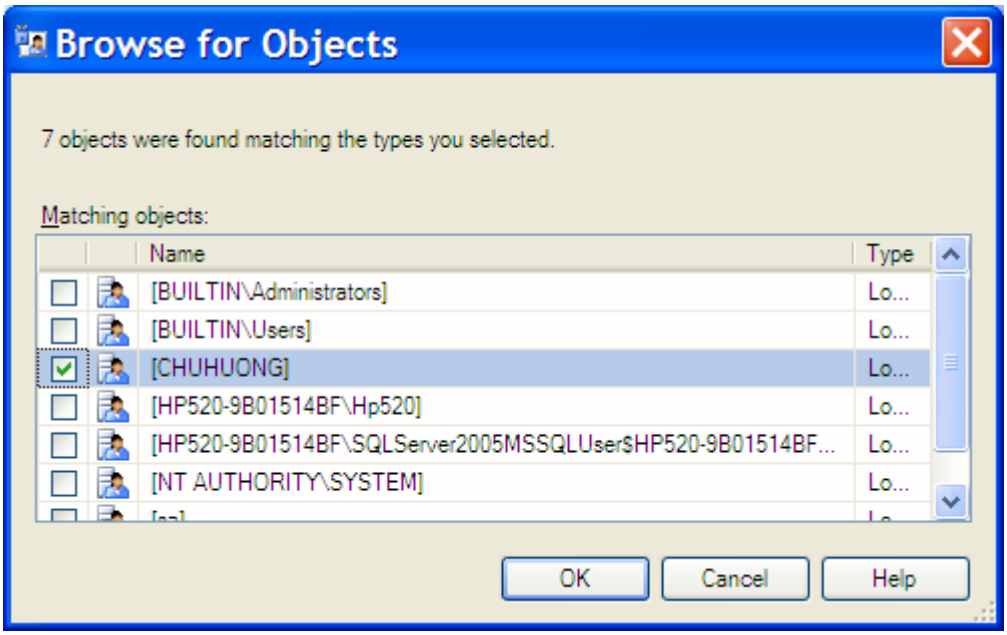
Hình 3.67. *Cửa sổ Server Role Propeties*

+ Click vào nút Add để thêm người dùng vào nhóm. Hộp thoại Select Login xuất hiện (Hình 3.68).

+ Click vào nút Browse để chọn đăng nhập ta muốn thêm vào nhóm. Xuất hiện hộp thoại Browse for Objects (Hình 3.69)



Hình 3.68. *Cửa sổ* Select Login



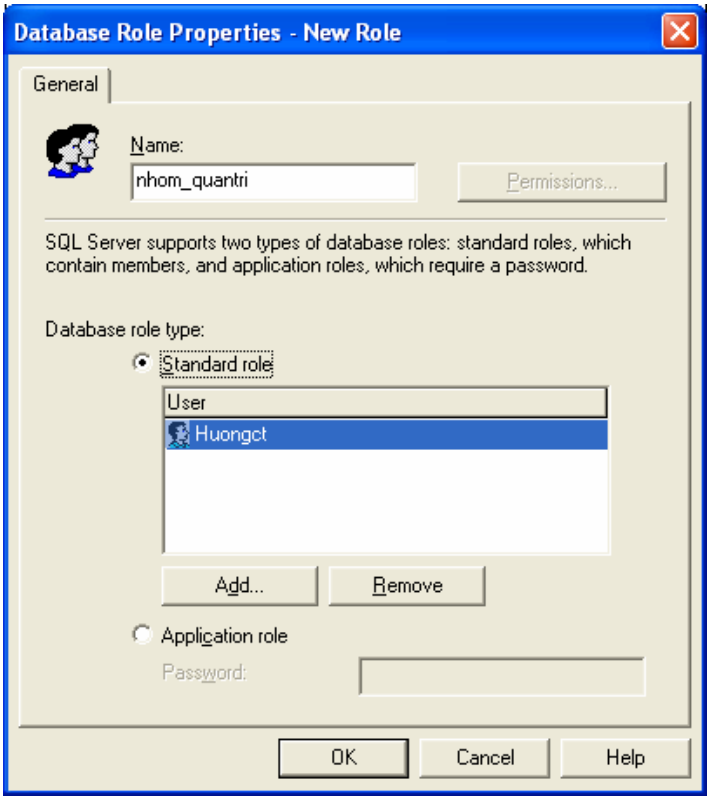
Hình 3.69. *Cửa sổ* Browse for Objects

+ Chọn các Login và click OK để đóng cửa sổ Browse for Objects.
+ Click OK để đóng cửa sổ Select Logins.
+ Click OK để đóng cửa sổ Server Role Propeties để thêm người dùng đã chọn vào nhóm.

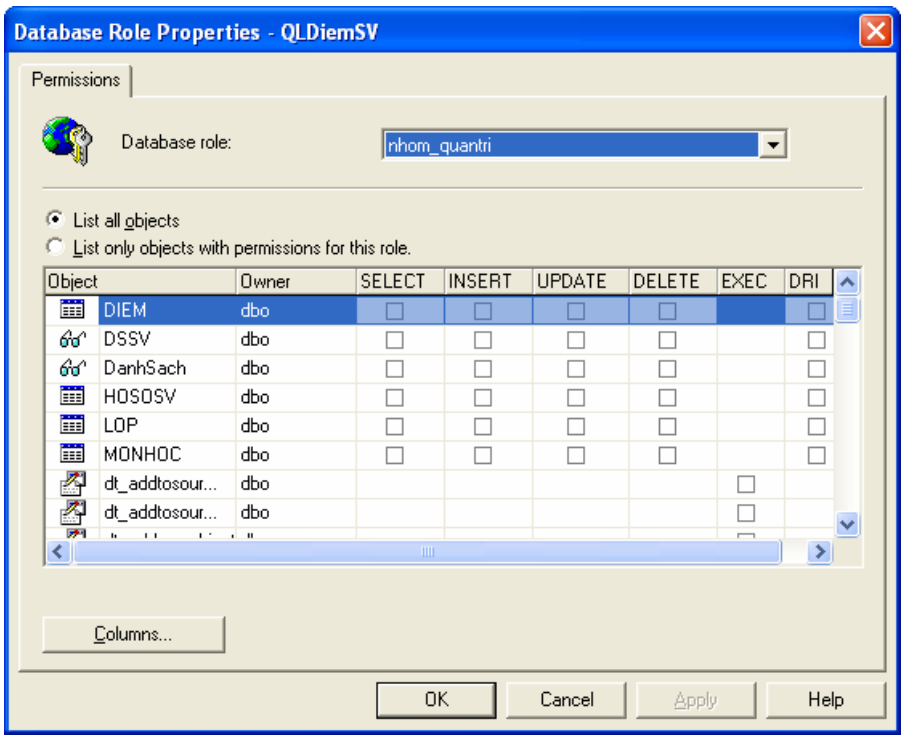
b) Tạo và sửa đổi nhóm quyền

➢ **Dùng Enterprise manager:**

+ Trong cửa sổ Enterprise Manager, mở rộng mục Database. Right click CSDL muốn tạo nhóm người dùng và chọn New\Database Role xuất hiện cửa sổ (hình 3.70).
+ Trong cửa sổ này ta nhập tên mô tả nhóm và Add các tài khoản người dùng thuộc nhóm bằng cách click vào nút Add và chọn các người dùng trong danh sách. Sau đó click nút OK để tạo nhóm.
+ Trở về Enterprise Manage, chọn mục Roles ta sẽ thấy nhóm quyền mới vừa tạo ở danh mục bên phải.
+ Ta gán quyền cho nhóm bằng cách right click lên nhóm quyền và chọn Propeties. Xuất hiện cửa sổ Database Role Propeties và chọn Permissions xuất hiện cửa sổ hình 3.71.



Hình 3.70. *Cửa sổ New Role*



Hình 3.71. *Cửa sổ Database Role Propeties*

+ Ta thực hiện gán các quyền cho các nhóm quyền này trên các đối tượng của CSDL và click OK.

➢ **Dùng SQL Server Management Studio:** SQL Server 2005 cung cấp hai loại nhóm quyền do người dùng định nghĩa

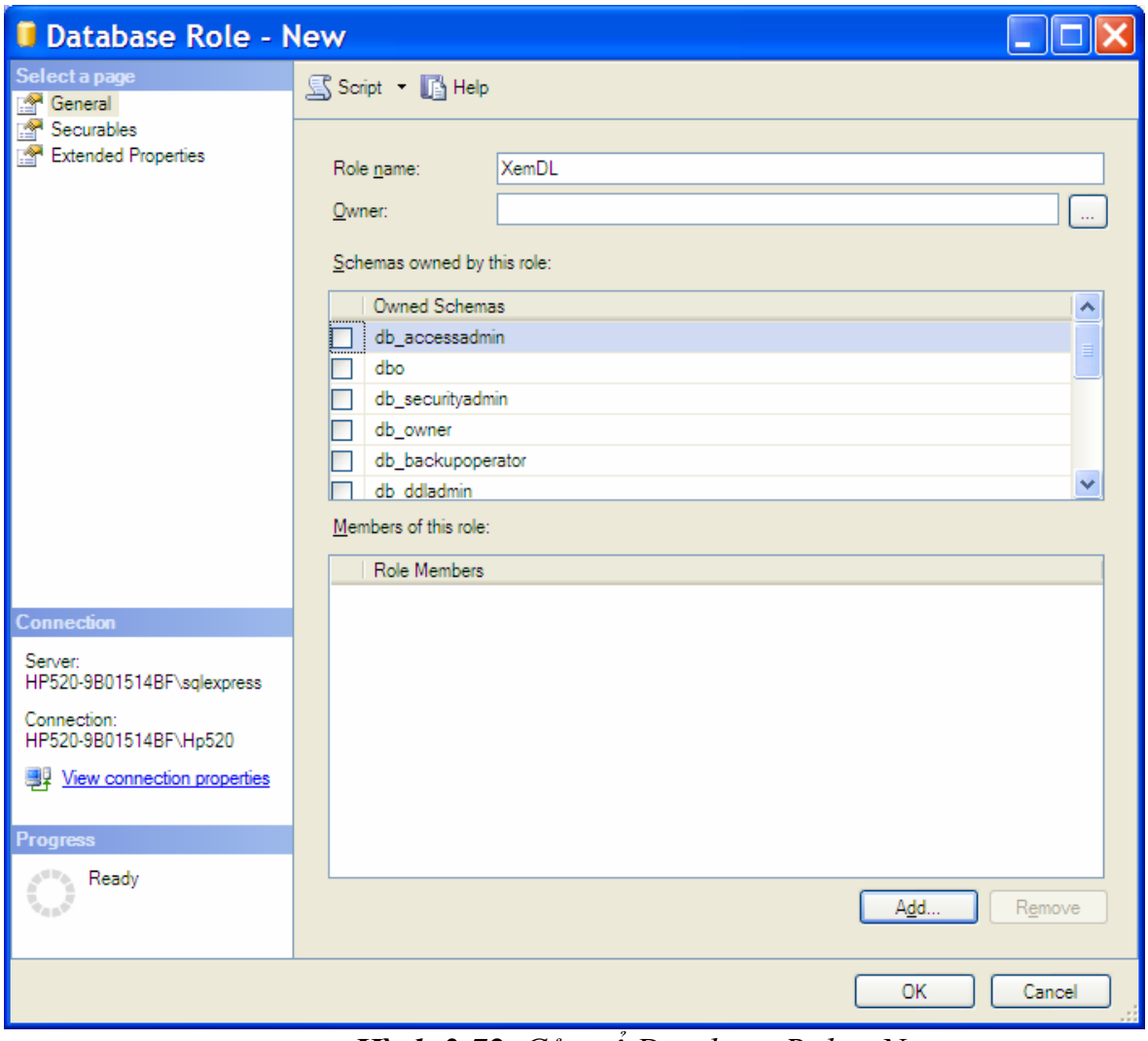
+ Database Roles: Đây là nhóm quyền chuẩn, dùng cho các tác vụ gán quyền tới cơ sở dữ liệu.

+ Application Roles: Dùng cho các quyền liên quan đến ứng dụng.

Xây dựng nhóm quyền Database Role ta tiến hành thực hiện theo các bước sau:

+ Trong SQL Server Management Studio, mở rộng mục Security cấp CSDL của CSDL ta muốn tạo nhóm quyền.

+ Mở rộng mục Roles, right click lên mục Database Roles và chọn New Database Role xuất hiện cửa sổ Database Role – New (Hình 3.72)



Hình 3.72. *Cửa sổ Database Role - New*

- Role name: Nhập tên nhóm
- Owner: Chọn danh sách các người dùng.
- Members of this role: Click vào nút Add để thêm các người dùng vào nhóm.

➢ **Dùng T-SQL:**

+ Ta thực hiện thông qua 2 bước.

1. *Tạo nhóm quyền:* Ta có thể tạo nhóm quyền bằng thủ tục sp_addrole. Đối SQL Server 2005 ta có thể dùng cú pháp sau:

```
CREATE ROLE role_name [AUTHORIZATION owner_name ]
```

Trong đó:

- *role_name:* Là tên của nhóm sẽ được tạo.
- AUTHORIZATION *owner_name* : Là người dùng CSDL hoặc các nhóm mà sở hữu nhóm mới này. Nếu không có người dùng nào chỉ định thì nhóm sẽ được sở hữu bởi người dùng thực hiện câu lệnh CREATE ROLE này.

Ví dụ 3.21. Tạo nhóm quyền 'xem_dl' trên CSDL QLDiemSV.

```
Use QLDiemSV
Go
Sp_addrole 'xem_dl'
```

2. *Thêm quyền vào nhóm:* Ta có thể thêm quyền vào nhóm bằng việc sử dụng lệnh GRANT và thu hồi quyền của nhóm sử dụng lệnh REVOKE.

   Ví dụ 3.22. Ta thêm quyền SELECT bảng HOSOSV vào nhóm 'xem_dl' trên CSDL QLDiemSV.

```
Use QLDiemSV
Go
GRANT SELECT
ON HOSOSV
TO xem_dl
```

3. *Thêm người dùng vào nhóm quyền:* Để thêm người dùng vào nhóm quyền ta sử dụng thủ tục sp_addrolemember

   Ví dụ 3.23. Thêm người dùng Guest vào nhóm quyền 'xem_dl' trên CSDL QLDiemSV.

```
Use QLDiemSV
Go
Sp_addrolemember 'xem_dl','Guest'
Go
```

3.3.1.3. Quản lý quyền CSDL

Xác thực người dùng là quá trình đảm bảo chỉ có những người dùng hợp lệ mới được phép làm việc với cơ sở dữ liệu. Sau khi người dùng truy cập được vào CSDL thì họ có các quyền cụ thể với các đối tượng trong CSDL.

Các quyền trên các đối tượng là:

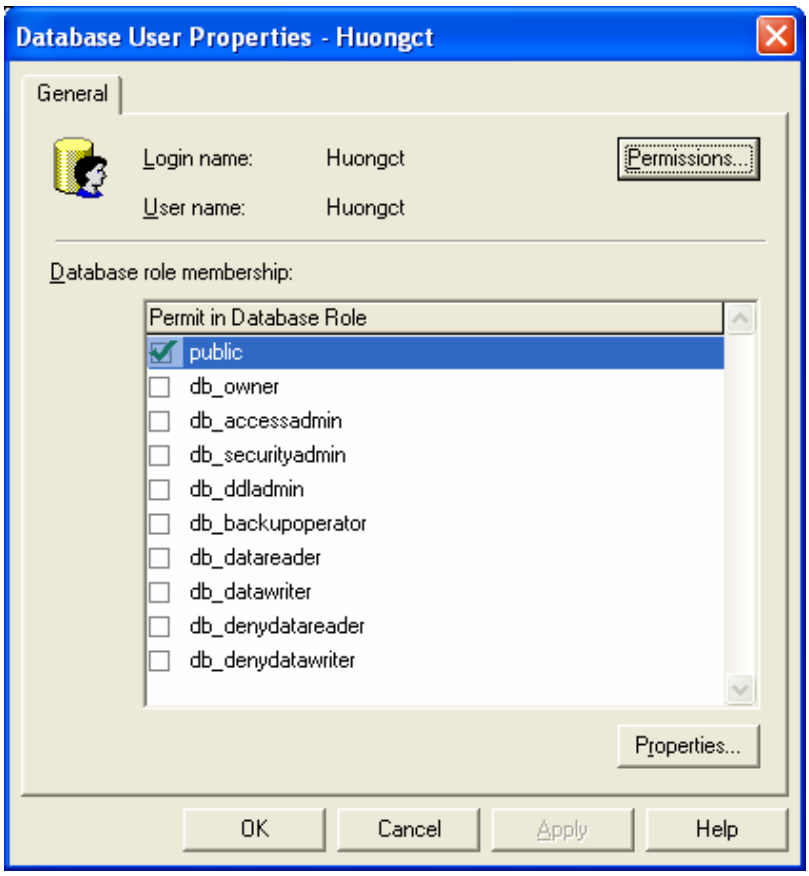
+ Đối tượng database: BACKUP DATABASE, BACKUP LOG, CREATE DATABASE, CREATE DEFAULT, CREATE FUNCTION, CREATE PROCEDURE, CREATE RULE, CREATE TABLE, và CREATE VIEW.
+ Đối tượng scalar function: EXECUTE và REFERENCES.
+ Đối tượng table-valued function: DELETE, INSERT, REFERENCES, SELECT, và UPDATE.
+ Đối tượng stored procedure: DELETE, EXECUTE, INSERT, SELECT, và UPDATE.
+ Đối tượng table: DELETE, INSERT, REFERENCES, SELECT, và UPDATE.
+ Đối tượng view: DELETE, INSERT, REFERENCES, SELECT, and UPDATE.

Cấp quyền trên các đối tượng CSDL ta tiến hành thực hiện như sau:

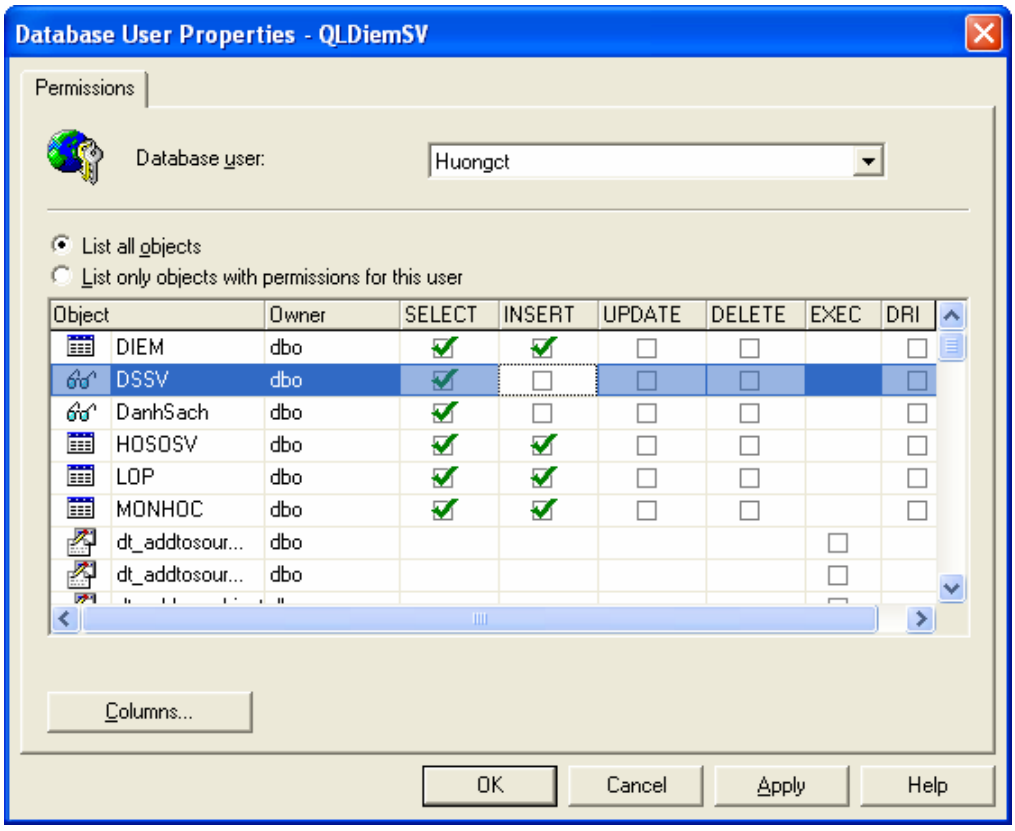
➢ **Dùng Enterprise manager để cấp quyền trên các đối tượng:** Có hai cách quản lý quyền trên các đối tượng.

Cách 1: Gán quyền cho tất cả các đối tượng cho một người dùng hoặc một nhóm người dùng.

+ Trong cửa sổ Enterprise Manager, mở rộng mục Database. Chọn CSDL muốn cấp quyền và mở rộng mục Users. Sau đó right click lên tên người dùng muốn cấp quyền và chọn Properties, xuất hiện cửa sổ như hình 3.73.
+ Click vào nút Permissionss để hiển thị cửa sổ Database Users Properties như hình 3.74. Ta gán các quyền cho người dùng này trên các đối tượng bằng cách chọn hộp kiểm tương ứng. Các tùy chọn *List all Objects* dùng để liệt kê tất cả các đối tượng còn chọn *List only objects with permissions for this user* thì chỉ liệt kê các đối tượng mà người dùng này có quyền truy cập.



Hình 3.73. *Cửa sổ* Database Users Properties

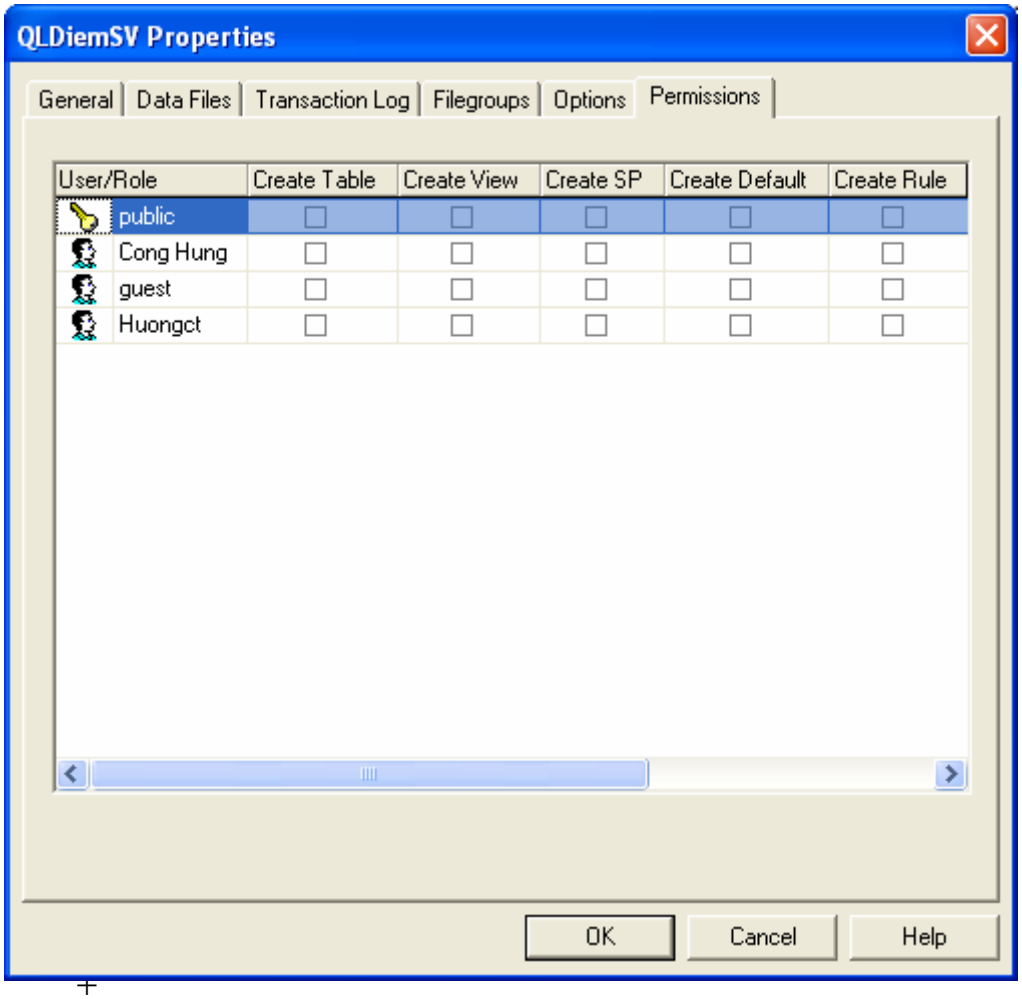


Hình 3.74. *Cửa sổ* Database Users Properties

Cách 2: Gán các quyền trên một đối tượng cho tất cả các người dùng hoặc các nhóm người dùng.

+ Trong cửa sổ Enterprise Manager, mở rộng mục Database. Sau đó right click lên CSDL muốn cấp quyền, ví dụ QLDiemSV và chọn Properties, xuất hiện cửa sổ như hình 3.75. Chọn tab Permissions.

+ Thực hiện cấp quyền thực thi cho các người dùng bằng cách tích vào ô tương ứng với các quyền đó.



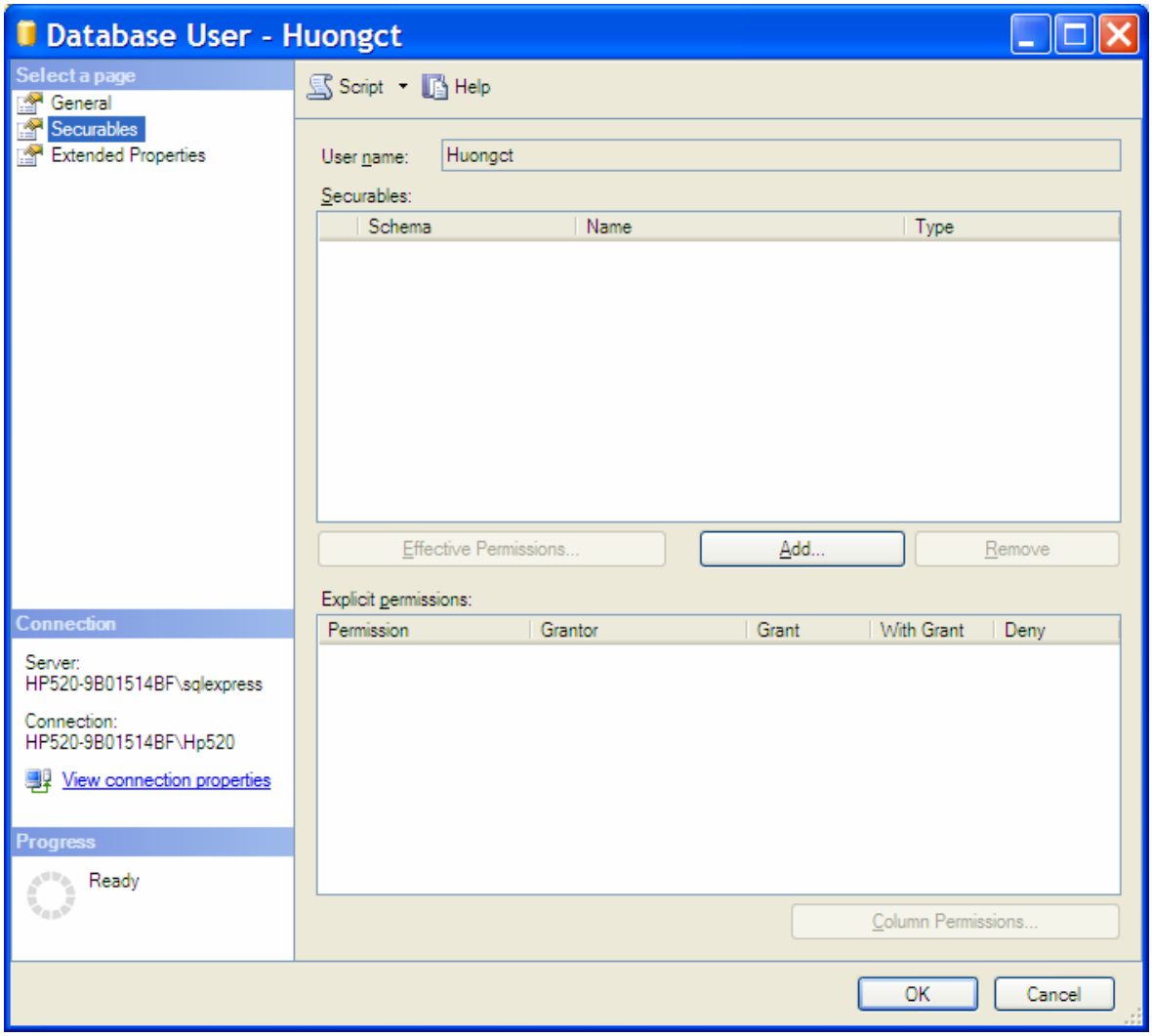
Hình 3.75. Cửa sổ QLDiemSV Properties

➢ **Dùng SQL Server Management Studio để cấp quyền trên các đối tượng:** Có hai cách quản lý quyền trên các đối tượng.

Cách 1: Gán quyền cho tất cả các đối tượng cho một người dùng hoặc một nhóm người dùng.

+ Trong cửa sổ Object Explorer của SQL Server Management Studio, mở rộng mục Database. Chọn CSDL muốn cấp quyền và mở rộng mục Users hoặc Roles\Database Roles.

+ Right click lên người dùng hoặc nhóm người dùng muốn và chọn Properties. Hộp thoại Properties xuất hiện chọn trang Securables (Hình 3.76)

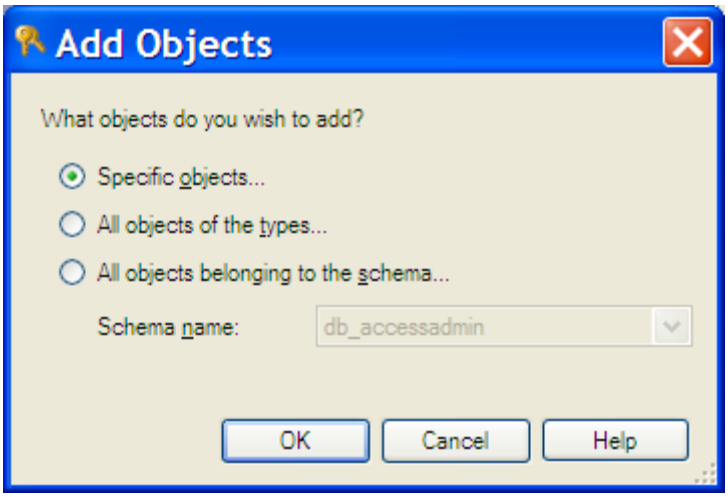


Hình 3.76. Cửa sổ Database User

+ Click nút Add để thêm các đối tượng muốn bảo mật. Hộp thoại Add Objects xuất hiện (Hình 3.77), ta chỉ các đối tượng muốn bảo mật.

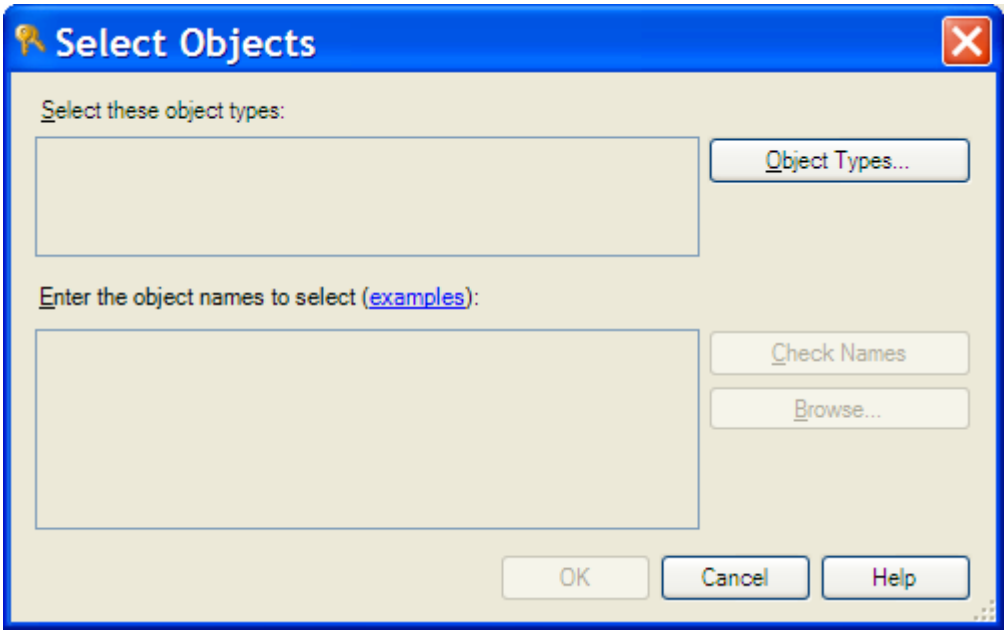
  - Specific objects: Chỉ định các đối tượng cụ thể.

- All objects of the types: Tất cả các đối tượng của các kiểu cụ thể.
- All objects belong to schema: Các đối tượng thuộc giản đồ.



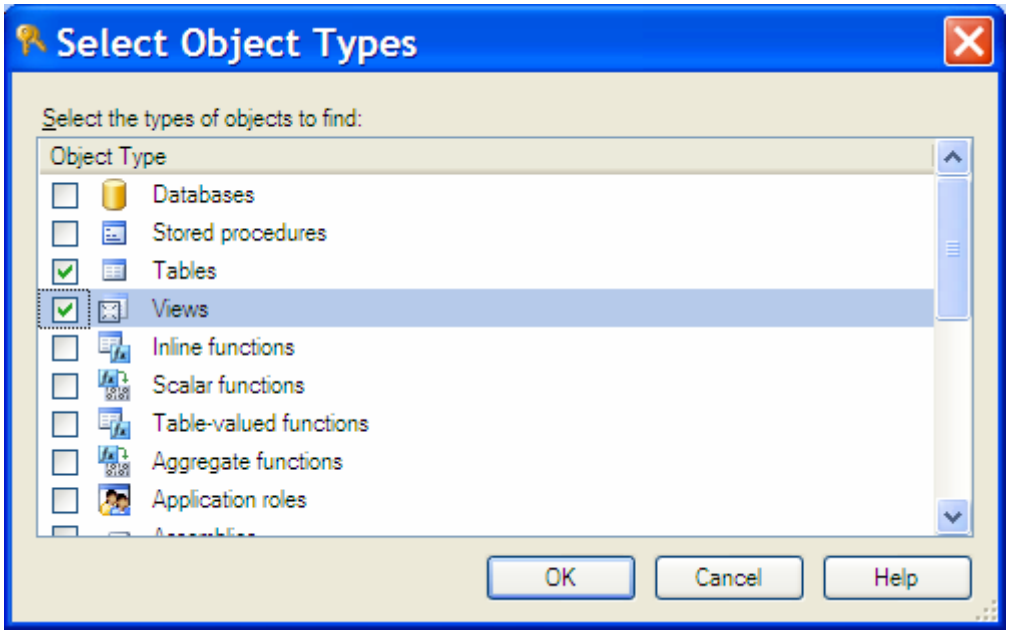
Hình 3.77. Cửa sổ Add Objects

+ Nếu chọn tùy chọn Specific objects, click OK hộp thoại Select Object xuất hiện (Hình 3.78).



Hình 3.78. Cửa sổ Select Objects

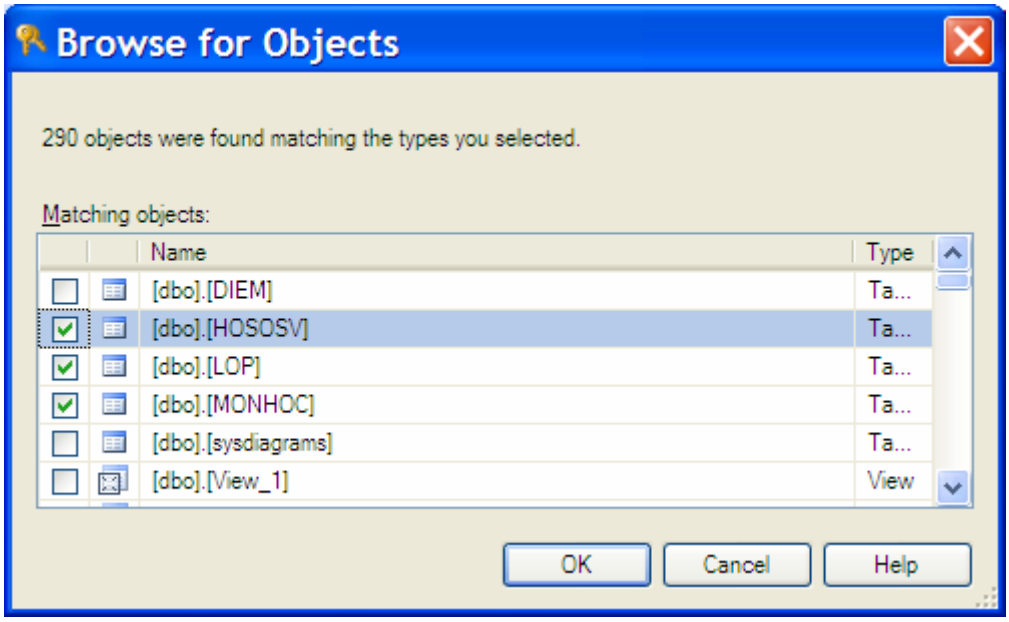
+ Click vào nút Object Types, hộp thoại Select Object Types xuất hiện như hình 3.79. Ta chọn các kiểu đối tượng mà muốn bảo mật cho người dùng đó và click OK.



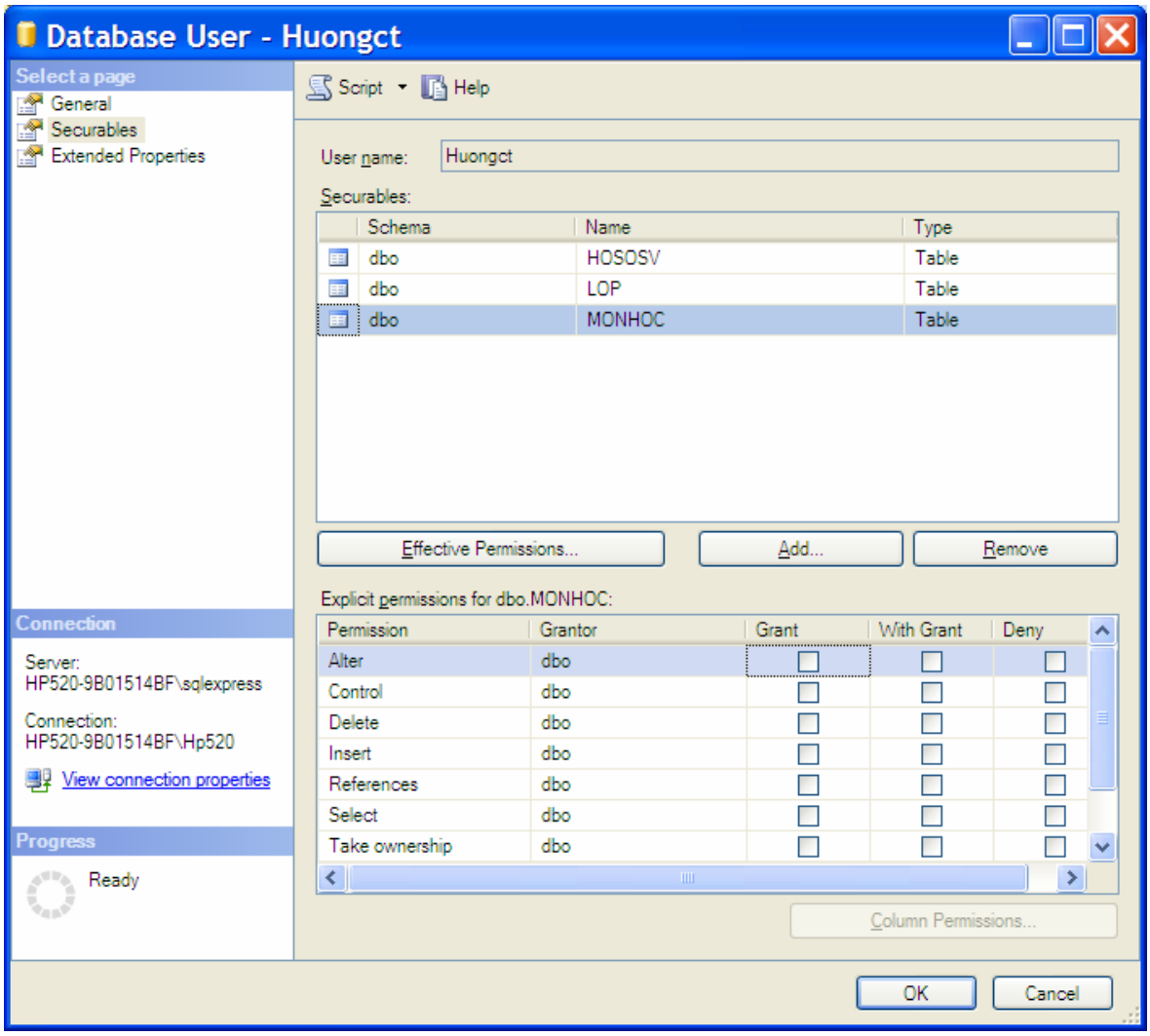
Hình 3.79. Cửa sổ Select Object Type

+ Click vào nút Browse trên hộp thoại Select Objects (Hình 3.78) xuất hiện cửa sổ Browse for Objects hình 3.80. Ta chọn các đối tượng mà muốn bảo mật cho người dùng đó và click OK.

+ Click OK trên cửa sổ Select Objects để quay trở lại trang Securable của hộp thoại Database User như hình 3.81. Ta chọn từng đối tượng và cấp quyền cho người dùng này.



Hình 3.80. Cửa sổ Browse for Objects



Hình 3.81. Cửa sổ Database User

➢ **Dùng T-SQL để cấp quyền:** Ta dùng lệnh các lệnh

+ GRANT: để cấp quyền cho người dùng;

+ DENY: dùng để ngăn cản quyền của người dùng nào đó. Ngăn cản các người dùng từ việc kế thừa các quyền trong nhóm.

+ REVOKE: để thu hồi lại quyền đã cấp.

Có cú pháp như sau:

```
GRANT { ALL }
      | permission [( column [ ,...n ] ) ] [ ,...n ]
      [ ON securable ] TO principal [ ,...n ]
      [ WITH GRANT OPTION ]
```

```
DENY { ALL }
      | permission [( column [ ,...n ] ) ] [ ,...n ]
      [ ON securable ] TO principal [ ,...n ]
      [ CASCADE]
REVOKE [ GRANT OPTION FOR ]
      {
        [ ALL ]
        |permission[( column [ ,...n ] ) ] [ ,...n ]
      }
      [ ON securable ]
      { TO | FROM } principal [ ,...n ]
      [ CASCADE]
```

Trong đó:

+ ALL: Cấp tất cả các quyền ;
+ permission: Tên các quyền cụ thể được cấp;
+ column: Tên các cột của bảng mà các quyền đó được được cấp;
+ securable: Chỉ định đối tượng đang cấp quyền trên đó.
+ Principal: Chỉ định người được cấp quyền.
+ WITH GRANT OPTION: Chỉ định người được cấp quyền có thể cấp quyền này cho người khác.
+ CASCADE: Chỉ định ngăn cản (đối với DENY) hoặc thu hồi (đối với REVOKE) theo dây truyền đối với các người dùng được cấp quyền với từ khóa WITH GRANT OPTION.

Ví dụ 3.15. Trao các quyền INSERT, SELECT, UPDATE cho *Huongct* trên CSDL QLDiemSV.

```
Use QLDiemSV
Go
GRANT INSERT,SELECT, UPDATE
ON HOSOSV
TO Huongct
```

Ví dụ 3.16. Trao tất cả các quyền cho *Huongct* trên CSDL QLDiemSV.

```
Use QLDiemSV
Go
```

được sao lưu. Đây là kỹ thuật phổ biến dùng cho các CSDL có kích thước vừa và nhỏ.

- ***Sao lưu những thay đổi (Differential Database):*** Cho phép chỉ sao lưu những dữ liệu thay đổi kể từ lần sao lưu gần nhất. Kỹ thuật này nhanh hơn và ít tốn không gian lưu trữ hơn so với sao lưu đầy đủ. Nhưng dùng phương pháp này khó khăn hơn và tốn nhiều thời gian gian hơn để khôi phục dữ liệu.
- ***Sao lưu tập tin log giao dịch (Transaction Log):*** Cho phép sao lưu transaction log, sao lưu này rất quan trọng cho phục hồi CSDL.
- ***Sao lưu nhóm tập tin (Full File Group):*** bao gồm sao lưu tất cả các tập tin dữ liệu kết hợp với tập tin đơn trong CSDL. Dùng phương pháp này để sao lưu các nhóm tập tin riêng biệt tùy thuộc vào cách hệ thống được cấu hình
- ***Sao lưu tập tin dữ liệu (Full File):*** Cho phép sao lưu một tập tin đơn trong nhóm tập tin. Phương pháp này kết hợp với khả năng của SQL Server để khôi phục một tập tin dữ liệu đơn riêng biệt.

Thực hiện sao lưu dữ liệu: Ta có thể thực hiện bằng các phương pháp như dùng:

Đối với SQL Server 2000: Enterprise manager, T-SQL, Create Database Backup Wizard.

Đối với SQL Server 2005: Phương pháp sử dụng SQL Server Management Studio, T-SQL

Physical and Logical Devices

SQL Server Database Engine nhận dạng các thiết bị back up (backup devices) hoặc là tên thiết bị vật lý (physical device) hoặc tên logic (logical device) :

- Thiết bị sao lưu vật lý là tên được sử dụng bởi hệ điều hành cho việc nhận dạng thiết bị back up. Ví dụ: C:\Backups\Accounting\Full.bak.

- Thiết bị sao lưu logic là bí danh do người dùng định nghĩa, được sử dụng để nhận dạng một thiết bị sao lưu vật lý. Tên thiết bị logic được lưu trữ thường trực trên các bảng hệ thống trong SQL Server. Tiện lợi của việc sử dụng thiết bị sao lưu logic là tên của nó đơn giản hơn so với tên thiết bị vật lý. Ví dụ, ta sử dụng tên logic là Accounting_Backup nhưng tên vật lý có thể là E:\Backups\Accounting\Full.bak.

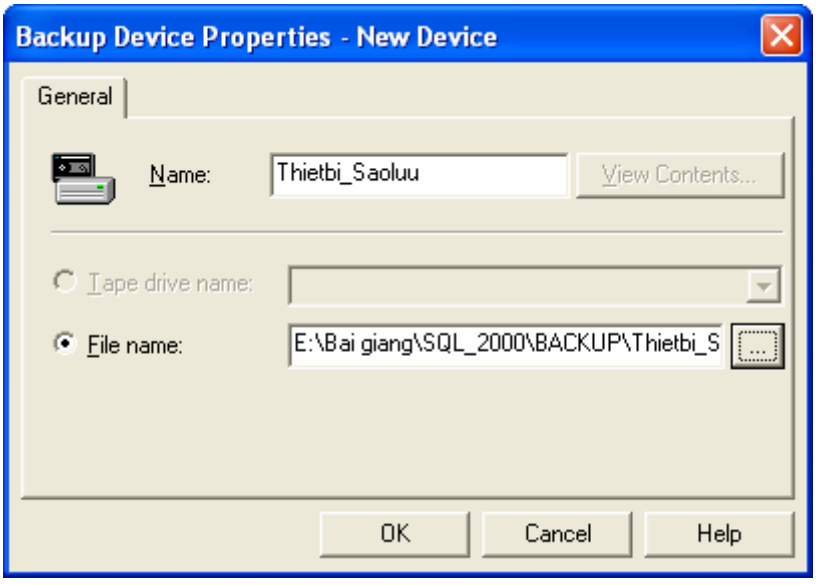
Hoạt động sao lưu có thể hướng tới thiết bị vật lý hoặc thiết bị logic. Thiết bị vật lý là đĩa cứng, băng từ,.v.v… còn thiết bị logic chỉ tồn tại trong SQL Server và chỉ được dùng cho SQL Server thực hiện sao lưu. Để sao lưu tới thiết bị logic, ta phải tạo thiết bị đó trước.

➢ **Dùng Enterprise manager:**

- ***Tạo thiết bị sao lưu logic:***

+ Trong cửa sổ Enterprise Manager, mở rộng server muốn thực hiện sao lưu và mở rộng mục Management.

+ Right click lên danh mục Backup và chọn New Backup Device xuất hiện cửa sổ *Backup Device Propeties (Hình 3.82)*. Ta nhập tên cho thiết bị sao lưu vào hộp Name và đường dẫn lưu file vào hộp thoại File Name. Sau đó click OK để lưu lại.

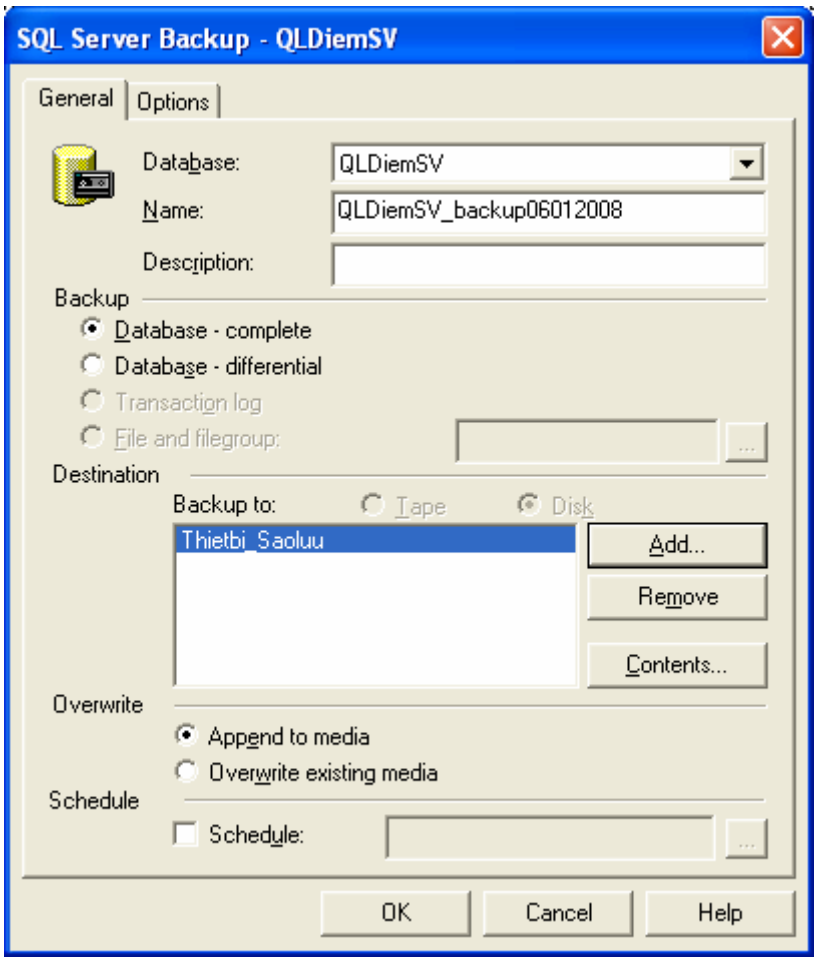


Hình 3.82. *Cửa sổ Backup Device Propeties*

- ***Thực hiện sao lưu:***

  + Trong cửa sổ Enterprise Manager, mở rộng server muốn thực hiện sao lưu và mở rộng mục Management.

  + Right click lên danh mục Backup và chọn New Backup a Database xuất hiện cửa sổ SQL Server Backup *(Hình 3.83)*.



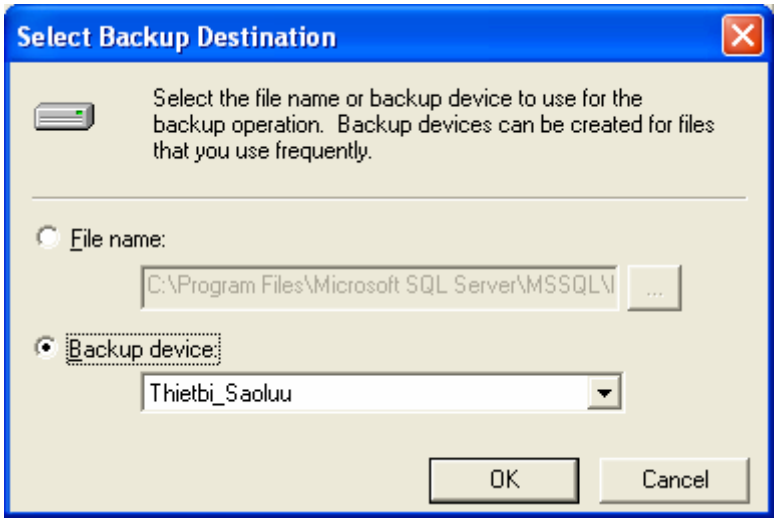
Hình 3.83. *Cửa sổ SQL Server Backup*

  + Trong danh sách Database chọn CSDL muốn thực hiện sao lưu, chẳng hạn chọn QLDiemSV. Tên sao lưu được điền tự động vào hộp Name, ta có thể thay đổi tên này. Nhập mô tả vào hộp Description.

  + Trong vùng BACKUP chỉ ra kiểu sao lưu. Có các tùy chọn có sẵn như là:

- *Database Complete*: Thực hiện sao lưu đầy đủ CSDL.
- *Database Differential*: Thực hiện sao lưu phần thay đổi.
- *Transaction log*: Thực hiện sao lưu tập tin log giao dịch.
- *File and filegroup*: Thực hiện sao lưu tập tin và nhóm tập tin.

+ Trong vùng Destination: Click vào nút Add xuất hiện cửa sổ *Select Backup Destination* hình 3.84.
  - *File Name:* Chọn vào tùy chọn này để thực hiện sao lưu thành một tập tin lưu trực tiếp xuống hệ điều hành (dùng thiết bị vật lý)
  - *Backup device:* Chọn thiết bị sao lưu logic. Chọn Thietbi_Saoluu vừa tạo ở mục trên và chọn OK.



Hình 3.84. *Cửa sổ Select Backup Destination*

+ Trong vùng Overwrite: Có 2 tùy chọn
  - *Append to media*: Ghi nối tiếp vào thiết bị
  - *Overwrite existing media*: để ghi đè lên thiết bị đang tồn tại.

+ Trong vùng Shedule: Dùng khi muốn thiết lập biểu sao lưu tự động.

+ Click OK để thực hiện sao lưu.

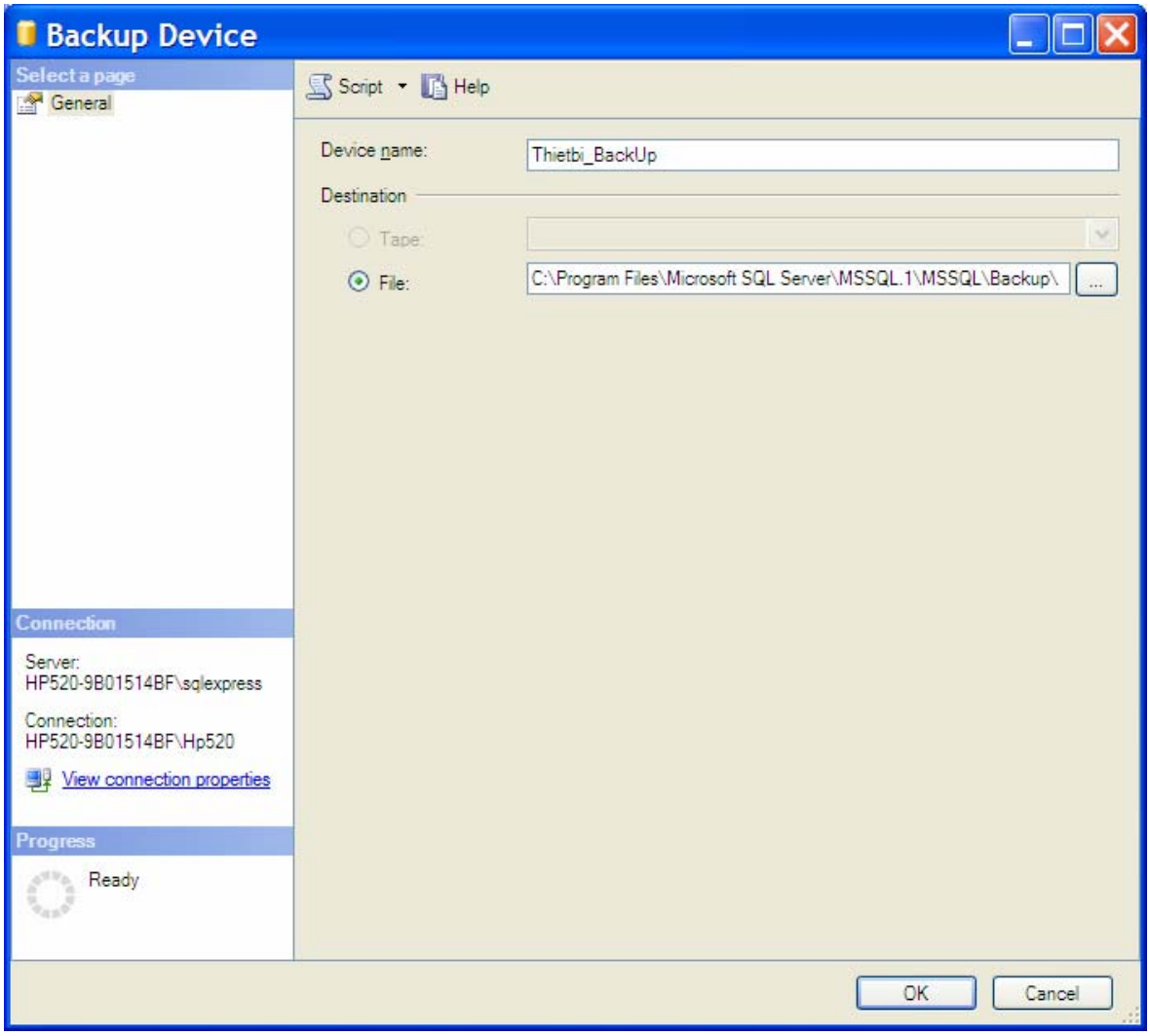
➢ **Dùng SQL Server Management Studio:**

- ***Tạo thiết bị sao lưu logic:***

  + Trong cửa sổ Object Explorer, click vào tên server để mở rộng cây server.

  + Mở rộng mục Server Objects, và right-click Backup Devices và chọn New Backup Device. Xuất hiện hộp thoại Backup Device (Hình 3.85). Trong hộp thoại mục:

    - Device name: Nhập tên thiết bị logic.
    - Destination, click File và chỉ định đường dẫn đầy đủ của file.

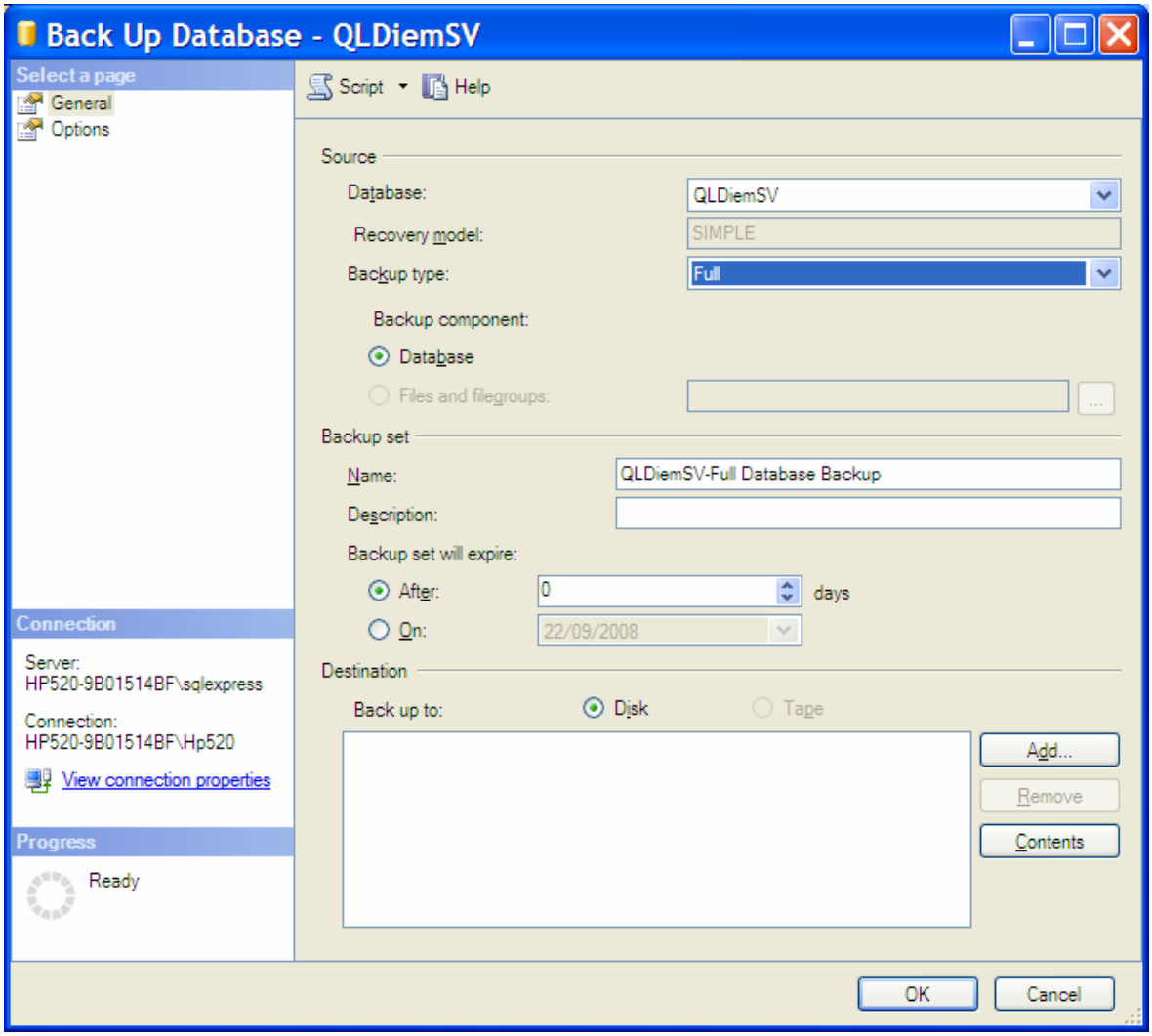


Hình 3.85. *Cửa sổ Backup Device*

- ***Thực hiện sao lưu:***

  + Trong cửa sổ Object Explorer, click vào tên server để mở rộng cây server.

  + Mở rộng mục Databases, và right-click lên cơ sở dữ liệu muốn tạo file Backup và chọn Tasks\Back Up. Xuất hiện hộp thoại Back Up Database (Hình 3.86).

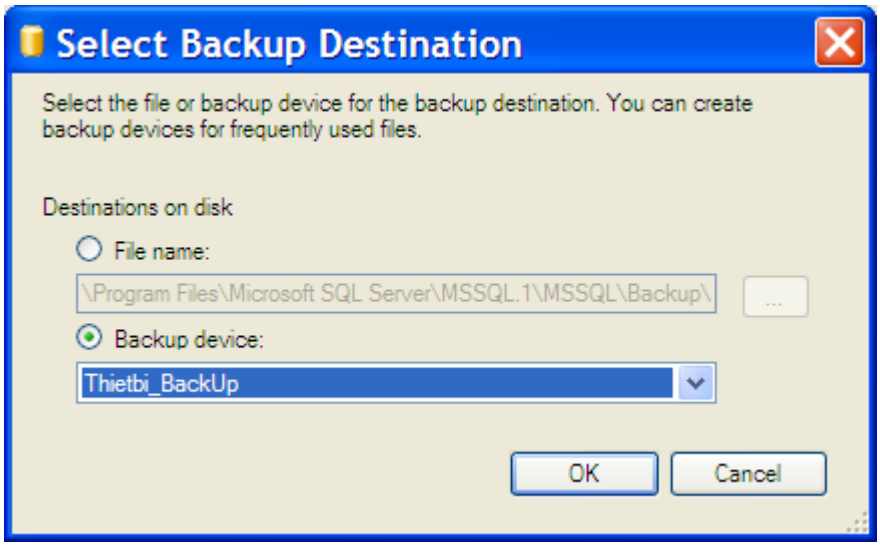


Hình 3.86. *Cửa sổ Back Up Database*

  + Trong hộp thoại ta thực hiện các lựa chọn:

    - Vùng Source: Database: Chọn Database muốn tạo file Back Up; Backup type: Chọn kiểu Back Up.

- Vùng Backup set: Thiết lập tên, mô tả,… file BackUp.
- Vùng Destination: Mục Backup to chọn Disk và click vào nút Add để chọn thiết bị sao lưu. Xuất hiện cửa sổ Select Backup Destination (Hình 3.87) ta sẽ chọn kiểu thiết bị sao lưu vật lý (chọn File name) hoặc logic (Backup device) và click OK.



Hình 3.87. *Cửa sổ* Select Backup Destination

+ Ta click vào nút OK trên cửa sổ Back Up Database để thực hiện quá trinhd sao lưu.

3.3.2.2. Phục hồi dữ liệu

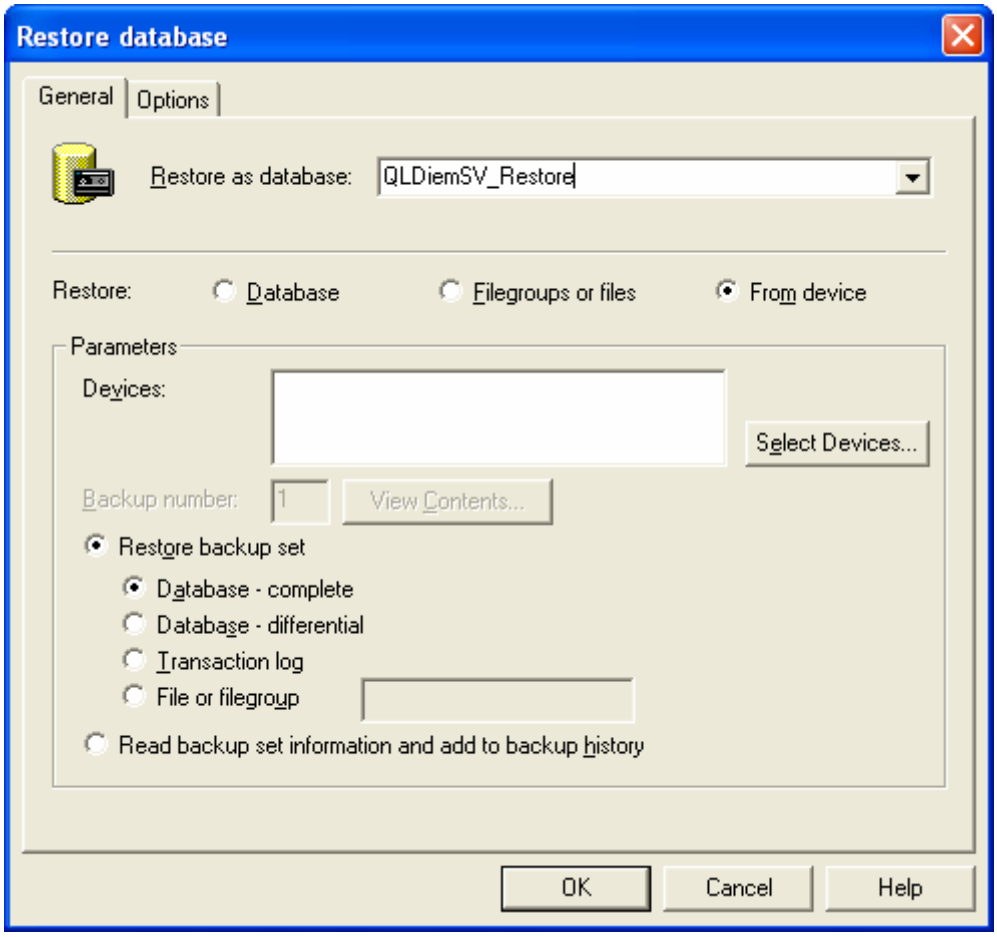
Phục hồi dữ liệu có thể coi là một quá trình ngược với quá trình sao lưu, dữ liệu sao chép được sao chép trở lại CSDL. Loại sao lưu dữ liệu sẽ ảnh hưởng đến cách khôi phục dữ liệu. Ta thực hiện các bước sau để khôi phục dữ liệu.

➢ **Dùng Enterprise manager:**

- Trong cửa sổ Enterprise Manager, right click lên mục Database. Chọn *All Task/Restore Database*. Khi đó xuất hiện hộp thoại Restore Database hình 3.88.

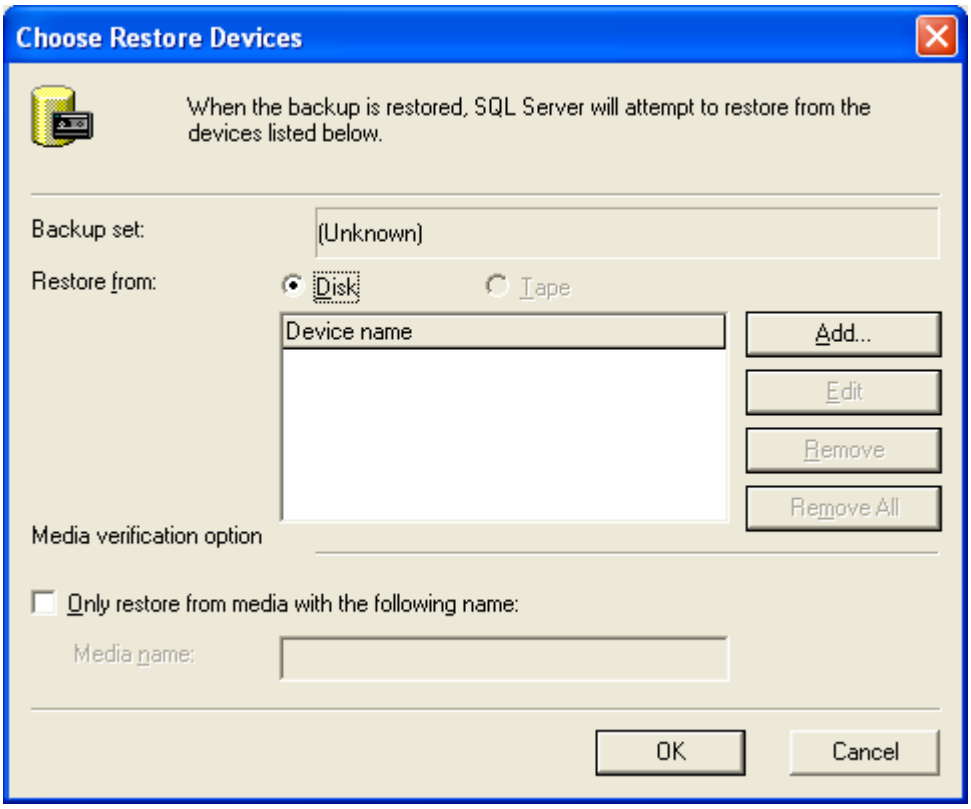
- Chọn tab General, mục Restore As Database cho phép chỉ ra CSDL sẽ được phục hồi từ dữ liệu sao lưu. Ví dụ ta chọn CSDL QLDiemSV thì việc khôi phục dữ liệu sao lưu sẽ thay thế toàn bộ dữ liệu hiện thời trong QLDiemSV. Khi đó ta phải chọn hộp check box Force Restore Over Existing Database trong tab Options.

  SQL Server không đòi hỏi phải khôi phục CSDL trực tiếp mà cho phép ta khôi phục dữ liệu với một cái tên khác. Ví dụ, người dùng xóa nhầm bảng dữ liệu. Như vậy, nếu ta khôi phục toàn bộ dữ liệu thì dữ liệu cũ sẽ thay thế toàn bộ dữ liệu đang có trên các bảng khác. Thay vào đó ta khôi phục với một cái tên khác sau đó trích bảng đã bị xóa và thêm vào CSDL đã bị mất. Ví dụ ta thay với tên QLDiemSV_Restore.



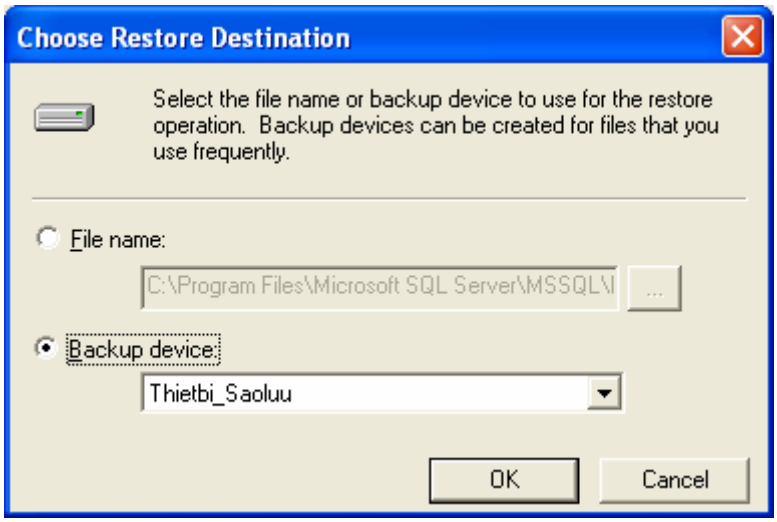
Hình 3.88. *Cửa sổ Restore Database.*

- Trong vùng Restore chọn From Device. Trong vùng Paramaters chọn Restore backup set và chọn Database complete.
- Click vào nút Select Device xuất hiện hộp thoại Choose Restore Device như hình 3.89.



Hình 3.89. *Cửa sổ Choose Restore Device.*

- Và chọn nút Add xuất hiện hộp thoại Choose Restore Destination (Hình 3.90).



Hình 3.90. *Cửa sổ Choose Restore Destination.*

- Trong đó có hai lựa chọn:
  + File name: Chọn tập tin sao lưu trên thiết bị vật lý.
  + Backup Device: Chọn tên thiết bị logic.

  Sau đó chọn OK để trở về cửa sổ hình 3.89. Trong cửa sổ 3.89 click OK để trở về cửa sổ 3.88.
- Chọn tab Options:
  + Force Restore Over Existing Database: Chọn nếu ta thực hiện khôi phục trực tiếp CSDL với dữ liệu cũ sẽ đè lên dữ liệu mới.
  + Mục Move to physical file name: Ta có thể sửa đổi lại thành tên khác với tên tập tin ban đầu. Ví dụ tập tin QLDiemSV_Restore.mdf chuyển thành QLDiemSV_Restore_Data.mdf
- Click OK để bắt đầu thực hiện phục hồi dữ liệu.

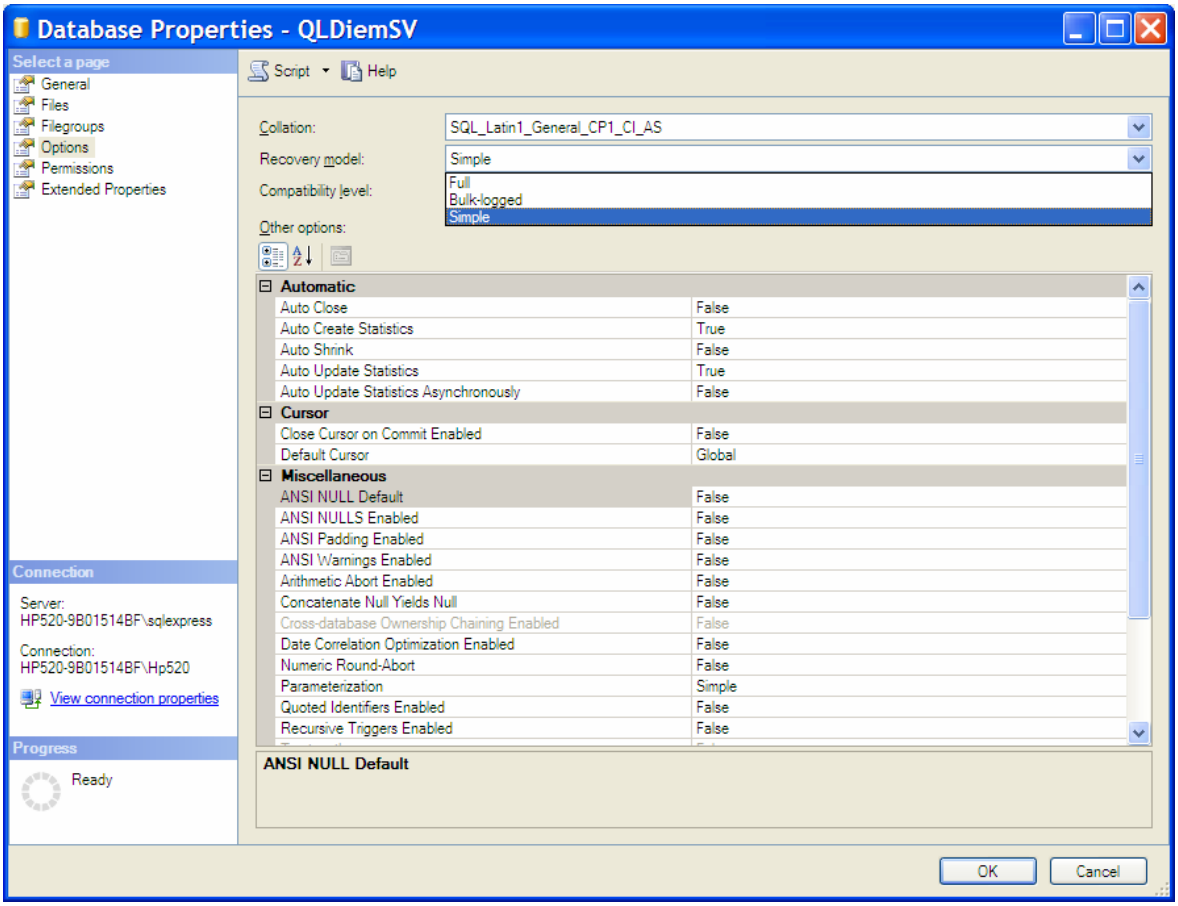
➢ **Dùng SQL Server Management Studio:**

- ***Các mô hình khôi phục dữ liệu:***
  + Full Recovery Model: Đây là mô hình cho phép phục hồi dữ liệu với ít rủi ro nhất. Nếu một database ở trong mô hình này thì tất cả các hoạt động không chỉ insert, update, delete mà kể cả insert bằng Bulk Insert, hay bcp đều được log vào transaction log file. Khi có sự cố thì ta có thể phục hồi lại dữ liệu ngược trở lại tới một thời điểm trong quá khứ. Khi data file bị hư nếu ta có thể backup được transaction log file thì ta có thể phục hồi database đến thời điểm transaction gần nhất được commited.
  + Bulk-Logged Recovery Model: Ở mô hình này này các hoạt động mang tính hàng loạt như Bulk Insert, bcp, Create Index, WriteText, UpdateText chỉ được log minimum vào transaction log file đủ để cho biết là các hoạt động này có diễn ra mà không log toàn bộ chi tiết như trong Full Recovery Mode. Các hoạt động khác như Insert, Update, Delete vẫn được log đầy đủ để dùng cho việc phục hồi sau này.

+ Simple Recovery Model: Ở mô hình này thì Transaction Log File được truncate thường xuyên. Với mo hình này bạn chỉ có thể phục hồi tới thời điểm backup gần nhất mà không thể phục hồi tới một thời điểm trong quá khứ.

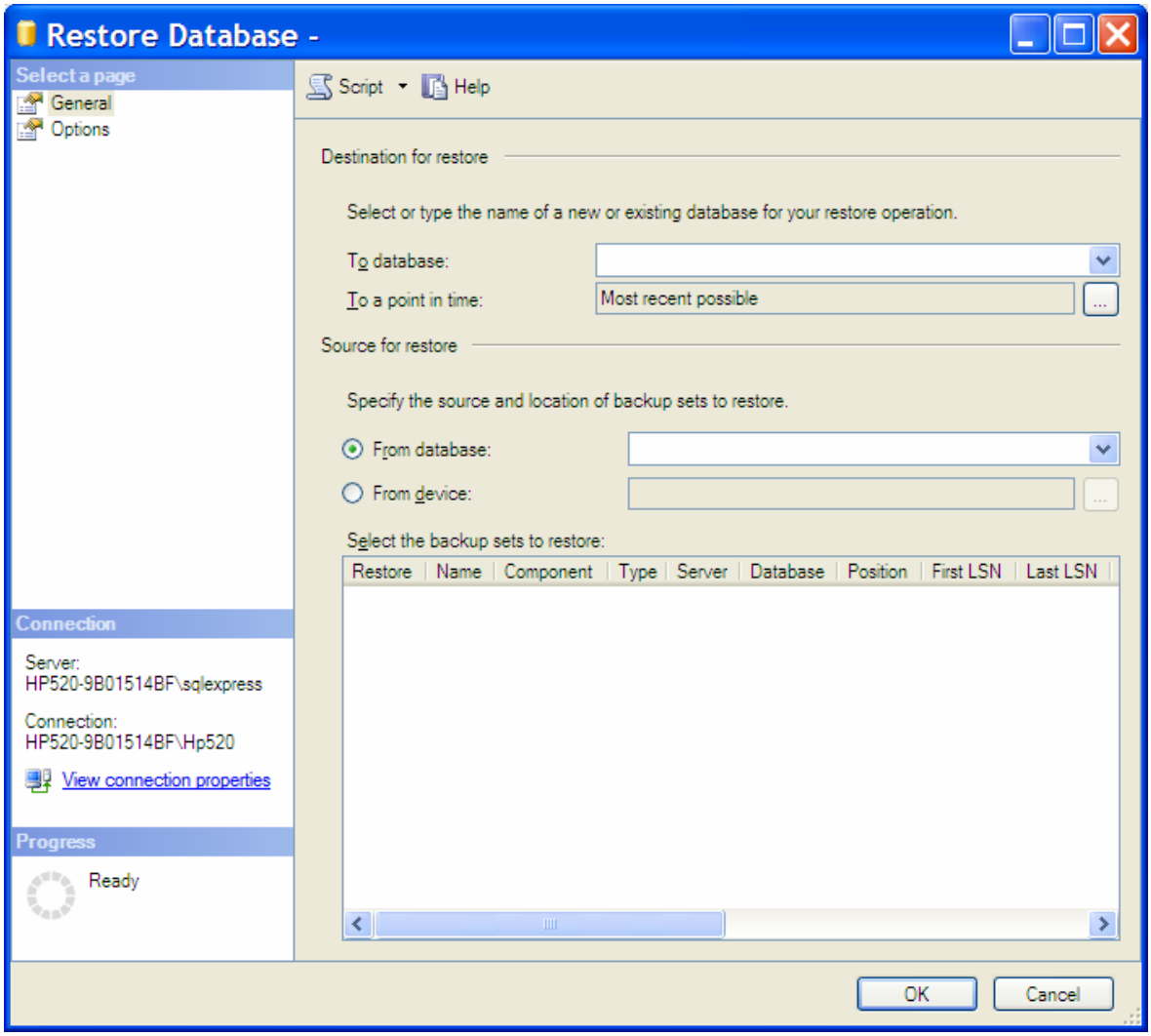
Để thay đổi mô hình Recovery, ta tiến hành thực hiện như sau:

+ Trong cửa sổ Object Explorer, click vào tên server để mở rộng cây server.
+ Mở rộng mục Databases, và right-click lên cơ sở dữ liệu muốn thay đổi mô hình Recovery và chọn Properties. Xuất hiện hộp thoại Database Properties, chọn trang Options (Hình 3.91).
+ Mục Recovery model: Ta chọn mô hình khôi phục dữ liệu.



Hình 3.91. *Cửa sổ Database Properties*

- ***Phục hồi dữ liệu:***
  - Trong cửa sổ Object Explorer, click vào tên server để mở rộng cây server.
  - Right click lên mục Databases, và chọn Restore Database xuất hiện cửa sổ Restore Database (Hình 3.92). Có các lựa chọn.

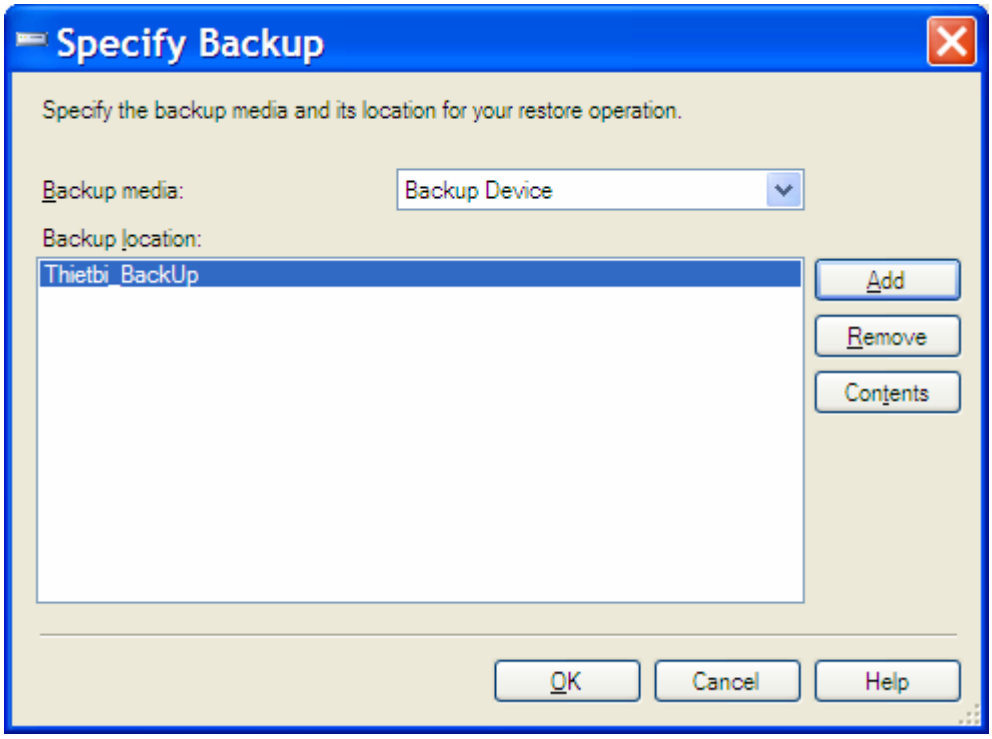


Hình 3.92. *Cửa sổ Restore Database*

+ *To database*: Điền tên Database sẽ được phục hồi. Tên này có thể là một tên Database mới hoặc chọn trong danh sách các Database có trong Server.

+ *To a point in time*: Restore database đến thời điểm sẵn có backup gần đây nhất hoặc đến một thời điểm được chỉ định.
+ *From database*: Chọn database để restore từ danh sách. Danh sách này chỉ chứa các database đã được backed up theo nhật ký của msdb backup.
+ *From device*: Chọn nguồn từ tập các file backup. Click vào nút browse để xuất hiện cửa sổ Specify Backup (Hình 3.93). Tong hộp thoại này ta chọn kiểu thiết bị (Backup media) là thiết bị vật lý hay thiết bị logic. Click nút Add để lấy các file Backup dùng để phục hồi dữ liệu.

o Trong cửa sổ Restore Database, sau khi thiết lập các tham số click OK để thực hiện quá trình phục hồi dữ liệu.



Hình 3.93. *Cửa sổ Restore Database*

trong mệnh đề ORDER BY xuất hiện như là dòng cuối cùng của TOP n (PERCENT).

Ví dụ 4.1. Sử dụng mệnh đề TOP

- Trong câu lệnh Insert

```
INSERT TOP (2) INTO LOP
     SELECT * FROM DMLOP ORDER BY Khoa
```

- Trong câu lệnh Select

```
INSERT INTO LOP
     SELECT TOP (2) WITH TIES * FROM DMLOP
             ORDER BY Khoa
```

b) Điều kiện kết nối - JOIN

Trong khối câu lệnh SELECT, ở mệnh đề FROM ta có thể sử dụng phát biểu JOIN để kết nối các bảng có quan hệ với nhau.

Mệnh đề kết nối Join được phân loại như sau:

- Inner joins (toán tử thường dùng để kết nối thường là các toán tử so sánh = hoặc <>). Inner joins sử dụng một toán tử so sánh để so khớp các dòng từ hai bảng dựa trên các giá trị của các cột so khớp của mỗi bảng. Kết quả trả về của Inner Join là các dòng thỏa mãn điều kiện so khớp.
- Outer joins. Outer joins có thể là left, right, hoặc full outer join.
  - LEFT JOIN hoặc LEFT OUTER JOIN : Kết quả của left outer join không chỉ bao gồm các dòng thỏa mãn điều kiện so khớp giữa hai bảng mà còn gồm tất cả các dòng của bảng bên trái trong mệnh đề LEFT OUTER. Khi một dòng ở bảng bên trái không có dòng nào của bảng bên phải so khớp đúng thì các giá trị NULL được trả về cho tất cả các cột ở bảng bên phải.
  - RIGHT JOIN or RIGHT OUTER JOIN: Right outer join là nghịch đảo của left outer join. Tất cả các dòng của bảng bên phải được trả về. Các giá trị Null cho bảng bên trái khi

bất cứ một dòng nào bên phải không có một dòng nào bảng bên trái so khớp đúng.

+ FULL JOIN or FULL OUTER JOIN: full outer join trả về tất cả các dòng trong cả hai bảng bên trái và phải. Bất kỳ một dòng không có dòng so khớp đúng của bảng còn lại thì bảng còn lại nhận các giá trị NULL. Khi có sự so khớp đúng giữa các bảng thì tập kết quả sẽ chứa dữ dữ liệu các bảng cơ sở đó.

➢ Cross joins: Trả về tất cả các dòng của bảng bên trái và mỗi dòng bên trái sẽ kết hợp với tất cả các dòng của bảng bên phải. Cross joins còn được gọi là tích Đề các (Cartesian products).

Ví dụ 4.2. Sử dụng Join

- Inner Joins:

```
SELECT MONHOC.MaMH, MONHOC.TenMH, MONHOC.SDVHT,
       DIEM.MaSV, DIEM.DiemL1
FROM DIEM INNER JOIN MONHOC
       ON DIEM.MaMH = MONHOC.MaMH
```

- Left Joins:

```
SELECT MONHOC.MaMH, MONHOC.TenMH, MONHOC.SDVHT,
       DIEM.MaSV, DIEM.DiemL1
FROM  MONHOC LEFT JOIN  DIEM
       ON MONHOC.MaMH= DIEM.MaMH
```

- Right Joins:

```
SELECT MONHOC.MaMH, MONHOC.TenMH, MONHOC.SDVHT,
       DIEM.MaSV, DIEM.DiemL1
FROM  DIEM Right JOIN MONHOC
      ON DIEM.MaMH= MONHOC.MaMH
```

- Full Joins:

```
SELECT MONHOC.MaMH, MONHOC.TenMH, MONHOC.SDVHT,
       DIEM.MaSV, DIEM.DiemL1
FROM  DIEM Full JOIN MONHOC
      ON DIEM.MaMH= MONHOC.MaMH
```

- Cross Joins:

```
SELECT MONHOC.MaMH, MONHOC.TenMH, MONHOC.SDVHT,
       DIEM.MaSV, DIEM.DiemL1
FROM MONHOC CROSS JOIN DIEM
```

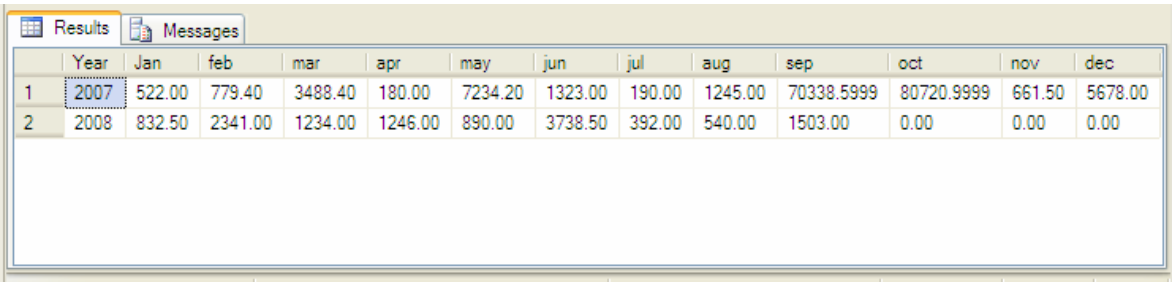
c) Truy vấn Cross tab

Trong một số trường hợp thống kê, ta cần phải xoay bảng kết quả, do đó có các cột được biểu diễn theo chiều ngang và các dòng được biểu diễn theo chiều dọc (được gọi là truy vấn cross tab).

Ví dụ 4.3. Ví dụ ta có một view tính tổng giá trị của một hóa đơn View_Order (OrderID, OrderDate, Month, Year, Total). Ta cần thống kê doanh thu theo từng tháng của các năm.

```
SELECT Year,
    SUM(CASE Month WHEN 1 THEN Total ELSE 0 END) AS Jan,
    SUM(CASE Month WHEN 2 THEN Total ELSE 0 END) AS feb,
    SUM(CASE Month WHEN 3 THEN Total ELSE 0 END) AS mar,
    SUM(CASE Month WHEN 4 THEN Total ELSE 0 END) AS apr,
    SUM(CASE Month WHEN 5 THEN Total ELSE 0 END) AS may,
    SUM(CASE Month WHEN 6 THEN Total ELSE 0 END) AS jun,
    SUM(CASE Month WHEN 7 THEN Total ELSE 0 END) AS jul,
    SUM(CASE Month WHEN 8 THEN Total ELSE 0 END) AS aug,
    SUM(CASE Month WHEN 9 THEN Total ELSE 0 END) AS sep,
    SUM(CASE Month WHEN 10 THEN Total ELSE 0 END) AS oct,
    SUM(CASE Month WHEN 11 THEN Total ELSE 0 END) AS nov,
    SUM(CASE Month WHEN 12 THEN Total ELSE 0 END) AS dec
FROM View_Order
GROUP BY Year
```

Kết quả:



Results | Messages

	Year	Jan	feb	mar	apr	may	jun	jul	aug	sep	oct	nov	dec
1	2007	522.00	779.40	3488.40	180.00	7234.20	1323.00	190.00	1245.00	70338.5999	80720.9999	661.50	5678.00
2	2008	832.50	2341.00	1234.00	1246.00	890.00	3738.50	392.00	540.00	1503.00	0.00	0.00	0.00

Sử dụng toán tử PIVOT và UNPIVOT

SQL Server 2005 đưa ra các toán tử đơn giản hơn cho việc tạo truy vấn cross tab, đó là toán tử PIVOT và UNPIVOT trong mệnh đề FROM của khối câu lệnh SELECT.

+ Toán tử PIVOT thực hiện xoay một biểu thức giá trị bảng (table valued expression) thành một bảng khác bằng việc đưa các giá trị duy nhất của một cột thành các cột và thực hiện các hàm thống kê trên các cột còn lại.

+ Toán tử UNPIVOT thực hiện quá trình ngược lại với quá trình thực hiện của toán tử PIVOT, xoay các cột của biểu thức bảng thành giá trị của một cột.

Cú pháp:

```
FROM { <table_source> } [ ,...n ]
<table_source> ::=
{
     <pivoted_table>
    | <unpivoted_table> [ ,...n ]
}

<pivoted_table> ::=
       table_source PIVOT <pivot_clause> table_alias

<pivot_clause> ::=
        ( aggregate_function( value_column )
        FOR pivot_column
        IN ( <column_list> )
    )

<unpivoted_table> ::=
        table_source UNPIVOT <unpivot_clause>
table_alias

<unpivot_clause> ::=
        ( value_column FOR pivot_column IN (
<column_list> ) )

<column_list> ::=
          column_name [ , ... ]
```

Trong đó:

+ *table_source* PIVOT <pivot_clause> : Chỉ định bảng *table_source* được xoay dựa trên cột *pivot_column*. *table_source* là một bảng hoặc biểu thức bảng. Output là một

bảng chứa tất cả các cột của *table_source* trừ cột *pivot_column* và *value_column*. Các cột của *table_source*, trừ *pivot_column* và *value_column*, được gọi là các cột phân nhóm của toán tử pivot.

+ *aggregate_function:* Là một hàm thống kê của hệ thống hoặc do người dùng định nghĩa. Hàm COUNT(*) không được phép sử dụng trong trường hợp này.

+ *value_column:* Là cột giá trị của toán tử PIVOT. Khi sử dụng với toán tử UNPIVOT, *value_column* không được trùng tên với các cột trong bảng input *table_source*.

+ FOR *pivot_column* : Chỉ định trục xoay của toán tử PIVOT. *pivot_column* là có kiểu chuyển đổi được sang **nvarchar()**. KHông được là các kiểu **image** hoặc **rowversion**.

  Khi UNPIVOT được sử dụng, *pivot_column* là tên của cột output được thu hẹp lại từ *table_source*. Tên cột này không được trùng với một tên nào trong *table_source*.

+ IN **(** *column_list* **)** : Trong mệnh đề PIVOT, danh sách các giá trị trong *pivot_column* sẽ trở thành tên các cột trong bảng output. Danh sách này không được trùng với bất kỳ tên cột nào tồn tại trong bảng input *table_source* mà đang được xoay.

  Trong mệnh đề UNPIVOT, danh sách các cột trong *table_source* sẽ được thu hẹp lại thành một cột *pivot_column*.

+ *table_alias:* Là tên bí danh của bảng output. *pivot_table_alias* phải được chỉ định.

+ UNPIVOT < unpivot_clause > : Chỉ định bảng input được thu hẹp bằng các cột trong *column_list* trở thành một cột gọi là *pivot_column*.

**** Hoạt động của toán tử PIVOT:***

Toán tử PIVOT thực hiện theo tiến trình sau:

+ Thực hiện GROUP BY dựa vào các cột phân nhóm trên bảng input_table và kết quả là ứng với mỗi nhóm cho một dòng out put trên bảng kết quả.

+ Sinh các giá trị ứng với các cột trong danh sách column list cho mỗi dòng output bằng việc thực thi như sau:

  - Nhóm các dòng được sinh từ việc GROUP BY ở bước trước dựa trên cột *pivot_column*.

    Đối với mỗi cột output trong *column_list*, chọn một nhóm con thỏa mãn điều kiện:

    ```
    pivot_column=CONVERT(<data type of pivot_column>, 'output_column')
    ```

  - *aggregate_function* định giá trị dựa tên cột *value_column* trong nhóm con này và kết quả được trả về của nó tương ứng là giá trị của cột *output_column*. Nếu nhóm con là rỗng thì SQL Server sinh giá trị NULL cho cột *output_column* đó. Nếu hàm thống kê là COUNT thì nó sinh giá trị 0.

Ví dụ 4.5. Ví dụ ta có một view tính tổng giá trị của một hóa đơn View_Order (OrderID, OrderDate, Month, Year, Total). Ta cần thống kê doanh thu theo từng tháng của các năm.

```
SELECT Year,[1]AS Jan,[2]AS feb, [3]AS mar,[4] AS apr,[5]
AS may,[6] AS jun,[7] AS jul,[8] AS aug,[9] AS sep,
[10]AS oct,[11] AS nov,[12] AS dec
FROM
    (SELECT Year, Month,Total
    FROM View_Order) p
PIVOT
    (Sum(Total) FOR Month IN
        ([1], [2], [3], [4], [5], [6], [7], [8], [9],
[10], [11], [12])
)AS pvt
```

Ví dụ 4.6. Sử dụng PIVOT

```
USE AdventureWorks
GO
SELECT VendorID, [164] AS Emp1, [198] AS Emp2, [223] AS
Emp3, [231] AS Emp4, [233] AS Emp5
FROM
```

```
(SELECT PurchaseOrderID, EmployeeID, VendorID
FROM Purchasing.PurchaseOrderHeader) p
PIVOT
(
COUNT (PurchaseOrderID)
FOR EmployeeID IN
( [164], [198], [223], [231], [233] )
) AS pvt
ORDER BY VendorID;
```

Ví dụ 4.7. Sử dụng UNPIVOT

```
CREATE TABLE pvt (VendorID int, Emp1 int, Emp2 int,
Emp3 int, Emp4 int, Emp5 int)
GO
INSERT INTO pvt VALUES (1,4,3,5,4,4)
INSERT INTO pvt VALUES (2,4,1,5,5,5)
INSERT INTO pvt VALUES (3,4,3,5,4,4)
INSERT INTO pvt VALUES (4,4,2,5,5,4)
INSERT INTO pvt VALUES (5,5,1,5,5,5)
GO
--Unpivot the table.
SELECT VendorID, Employee, Orders
FROM
    (SELECT VendorID, Emp1, Emp2, Emp3, Emp4, Emp5
    FROM pvt) p
UNPIVOT
    (Orders FOR Employee IN
        (Emp1, Emp2, Emp3, Emp4, Emp5)
)AS unpvt
```

d) UNION và UNION ALL

Toán tử UNION [ALL] dùng để hợp kết quả của hai hoặc nhiều câu truy vấn tương thích với nhau. Hai câu truy vấn tương thích là hai câu có cùng cấu trúc, tức là có cùng số cột và tập các cột tương ứng có cùng kiểu dữ liệu hoặc có các kiểu dữ liệu tương thích nhau. Cú pháp của câu lệnh:

```
select_statement UNION [ALL] select_statement
```

Tên của các cột trong phép toán UNION là tên các cột trong tập kết quả của khối câu lệnh SELECT thứ nhất trong UNION.

Theo mặc định phép toán UNION chỉ lấy đại diện cho tập các dòng trùng nhau. Nếu ta sử dụng từ khóa ALL, thì tất cả các dòng được cho vào bảng kết quả và các dòng trùng nhau sẽ không loại bỏ các dòng trùng nhau.

	giá trị FALSE.	
AND	Kết hợp và so sánh giữa hai biểu thức Boolean, nếu cả hai biểu thức đều TRUE thì nó trả về giá trị TRUE và ngược lại nó trả về giá trị FALSE.	5 > 7 AND 6 < 15
ANY	So sánh một giá trị vô hướng với một tập các giá trị của một cột được lấy từ một câu truy vấn con. Nó sẽ trả về giá trị TRUE nếu có bất cứ giá trị nào trong cột trả về giá trị TRUE. Nếu không có một giá trị nào trả về giá trị TRUE thì nó trả về giá trị FALSE. ANY tương tự như toán tử SOME.	5 > ANY (SELECT qty FROM sales)
BETWEEN	Kiểm tra giá trị có nằm giữa phạm vi được chỉ định hay không. Trả về giá trị TRUE nếu nó nằm trong khoảng giá trị đó và ngược lại trả giá trị FALSE.	5 BETWEEN (3 AND 10)
EXISTS	Kiểm tra xem có giá trị nào trả về khi thực hiện một câu truy vấn. Nếu có các giá trị trả về thì toán tử cho giá trị TRUE, ngược lại trả về giá trị FALSE.	EXISTS (SELECT * FROM test)
IN	Kiểm tra xem một giá trị có tồn tại trong một tập các giá trị hay không. Nếu giá trị mà thuộc tập giá trị đó thì toán tử trả về giá trị TRUE, ngược lại trả về giá trị FALSE.	5 IN (SELECT qty FROM sales)
LIKE	Dùng để so khớp các giá trị với một mẫu theo từ khóa LIKE. Nó sẽ trả về giá trị TRUE nếu khớp với mẫu ngược lại trả về giá trị FALSE. Ký tự % đại diện cho một dãy ký tự bất kỳ, _ đại diện cho một ký tự bất kỳ.	SELECT name WHERE name LIKE ‘S%’

NOT	Dùng để phủ định một biểu thức Boolean.	NOT 5 > 2
OR	Kết hợp và so sánh giữa hai biểu thức Boolean, nếu một trong hai biểu thức là TRUE thì nó trả về giá trị TRUE và ngược lại nó trả về giá trị FALSE.	5 > 2 OR 10 < 3
SOME	So sánh một giá trị vô hướng với một tập các giá trị của một cột được lấy từ một câu truy vấn con. Nó sẽ trả về giá trị TRUE nếu có bất cứ giá trị nào trong cột trả về giá trị TRUE. Nếu không có một giá trị nào trả về giá trị TRUE thì nó trả về giá trị FALSE. SOME tương tự như toán tử ANY.	5 > SOME (SELECT * FROM sales)

- *Toán tử ghép chuỗi (+)*: Dùng để ghép hai chuỗi với nhau thành một chuỗi. Toán tử ghép chuỗi được dùng với các kiểu dữ liệu char, varchar, nchar, nvarchar, text, và ntext.

```
SELECT 'This' + ' is a test.'
```

- *Toán tử bit*: Thực hiện thao tác với các bit-lavel với các kiểu dữ liệu Integer. Các toán tử đó được cho trong bảng 4.2.

Bảng 4.2. Các toán tử Bitwise

Toán tử	*Ý nghĩa*	*Ví dụ*
&	Thực hiện AND giữa các bit tương ứng giữa hai biểu diễn nhị phân của hai số integer.	7 & 51 = 3 (7=111, 51=110011, 3=11)
|	Thực hiện OR giữa các bit tương ứng giữa hai biểu diễn nhị phân của hai số integer.	7 | 51 = 55
^	Thực hiện XOR giữa các bit tương ứng giữa hai biểu diễn nhị phân của hai số integer. (hai bit giống nhau trả về bit 0, khác nhau trả về bit 1)	7 ^51 = 52

~	Thực hiện NOT của biểu thức biểu diễn nhị phân của mọt số nguyên	~7 = -8

b) Cấu trúc lặp

SQL Server cung cấp hai cấu trúc lặp đó là: cấu trúc WHILE và GOTO.

- Cấu trúc lặp WHILE: Câu lệnh WHILE sẽ kiểm tra điều kiện trước khi thực hiện lệnh. Một khối lệnh là một tập các câu lệnh được bao trong cặp từ khóa BEGIN …END. Cú pháp:

```
WHILE Boolean_expression
    {sql_statement| statement_block}
    [BREAK]
    {sql_statement| statement_block}
    [CONTINUE]
```

trong đó:

+ *Boolean_expression:* Là biểu thức điều kiện để kiểm tra điều kiện lặp. Vòng lặp sẽ được thực hiện khi biểu thức trả về giá trị True và kết thúc vòng lặp khi trả về giá trị False.
+ *sql_statement|statement_block*:Đó là câu lệnh SQL hoặc khối các câu lệnh SQL sẽ được lặp lại trong câu lệnh While. Khối các câu lệnh SQL được bao trong cặp từ khóa BEGIN … END
+ *BREAK:* Từ khóa dùng để chỉ định dừng việc thực thi vòng lặp hiện tại. Tất cả các câu lệnh sau từ khóa BREAK và trước từ khóa END sẽ bị bỏ qua.
+ *CONTINUE*: Từ khóa dùng để restart lại vòng lặp hiện tại tại ví trí bắt đầu. Tất cả các câu lệnh sau từ khóa CONTINUE và trước từ khóa END sẽ bị bỏ qua.

Ví dụ 4.5. Sử dụng cấu trúc lặp WHILE đơn giản.

```
Use pubs
go
CREATE TABLE WhileLoopTest
(
    LoopID        INT,
```

```
    LoopValue    VARCHAR(32)
)
GO
SET NOCOUNT ON
DECLARE  @intCounter    INT
DECLARE @vchLoopValue    VARCHAR(32)

SELECT @intCounter = 1
WHILE (@intCounter <= 100)
 BEGIN
    SELECT @vchLoopValue = 'Loop Interation #' +
           CONVERT (VARCHAR(4), @intCounter)
    INSERT INTO WhileLoopTest(LoopID, LoopValue)
           VALUES (@intCounter, @vchLoopValue)
    SELECT @intCounter = @intCounter + 1
 END
```

- Cấu trúc lặp GOTO: Tương tự như cấu trúc WHILE, GOTO có thể cho phép lặp một chuỗi câu lệnh cho đến khi điều kiện được thỏa mãn.

Chú ý: Câu lệnh GOTO không nhất thiết phải sử dụng trong các vòng lặp mà có thể sử dụng để thoát khỏi vòng lặp khác.

Để sử dụng câu lệnh GOTO, trước hết ta phải định nghĩa một nhãn. Nhãn là một câu lệnh chỉ định vị trí mà câu lệnh GOTO sẽ nhảy đến. Để tạo nhãn ta sử dụng cú pháp sau:

```
LABLE:
```

Để nhảy đến nhãn trong code ta sử dụng câu lệnh GOTO theo cú pháp sau:

```
GOTO  LABLE
```

Trong đó: LABLE là nhãn đã được định nghĩa ở trước đó trong code. Bằng việc sử dụng GOTO, ta có thể nhảy đến một vị trí bất kỳ trong code.

Ví dụ 4.6. Sử dụng cấu trúc lặp GOTO đơn giản.

```
Use pubs
Go
CREATE TABLE GotoLoopTest
(
    GotoID        INT,
    GotoValue     VARCHAR(32)
)
GO

SET NOCOUNT ON
```

```
    DECLARE    @intCounter    INT
    DECLARE    @vchLoopValue  VARCHAR(32)

    SELECT @intCounter = 0

    LOOPSTART:
    SELECT @intCounter = @intCounter + 1
    SELECT @vchLoopValue = 'Loop Iteration #' +
               CONVERT(VARCHAR(4), @intCounter)
    INSERT INTO GotoLoopTest(GotoID, GotoValue) VALUES
(@intCounter, @vchLoopValue)
    IF (@intCounter <= 1000)
    BEGIN
        GOTO LOOPSTART
    END
```

c) Cấu trúc rẽ nhánh

- *Cấu trúc IF...ELSE*: Cấu trúc IF…ELSE là một khối các câu lệnh dùng để rẽ nhánh dựa trên các tham số được cung cấp. Cú pháp của khối câu lệnh IF như sau:

```
IF expression
    BEGIN
    sql_statements
    END
[ELSE
    BEGIN
    sql_statements
    END]
```

Chú ý: Ta có thể sử dụng các cấu trúc IF lồng nhau.

Ví dụ 4.7. Sử dụng cấu trúc rẽ nhánh IF.

```
Use pubs
Go

CREATE PROCEDURE uspCheckNumber
    @intNumber    INT
AS

IF @intNumber < 1
   BEGIN
    PRINT 'Number is less that 1.'
    RETURN
   END

ELSE IF @intNumber = 1
  BEGIN
    PRINT 'One'
    RETURN
  END
```

```
ELSE IF @intNumber = 2
 BEGIN
    PRINT 'Two'
    RETURN
 END
ELSE IF @intNumber = 3
 BEGIN
    PRINT 'Three'
    RETURN
 END
ELSE IF @intNumber = 4
 BEGIN
    PRINT 'Four'
    RETURN
 END
ELSE IF @intNumber = 5
 BEGIN
    PRINT 'Five'
    RETURN
 END
ELSE IF @intNumber = 6
 BEGIN
    PRINT 'Six'
    RETURN
 END
ELSE IF @intNumber = 7
 BEGIN
    PRINT 'Seven'
    RETURN
 END
ELSE IF @intNumber = 8
 BEGIN
    PRINT 'Eight'
    RETURN
 END
ELSE IF @intNumber = 9
 BEGIN
    PRINT 'Nine'
    RETURN
 END
ELSE IF @intNumber = 10
 BEGIN
    PRINT 'Ten'
    RETURN
 END
ELSE
 BEGIN
```

```
    PRINT 'Number is greater than 10.'
    RETURN
END
```

- *Cấu trúc CASE:* Cấu trúc này được dùng để đánh giá một biểu thức và trả về một hoặc một số các kết quả dựa vào giá trị của biểu thức. Có 2 kiểu cấu trúc CASE khác nhau như sau:
    - *Simple CASE:* Với cấu trúc này, một biểu thức sẽ được dùng để so sánh với một tập các giá trị để xác định kết quả. Cú pháp như sau:

```
CASE case_expression
    WHEN expression THEN result
        [...n
    [ELSE else_result
END
```

    - Searched CASE: Đánh giá tập các biểu thức Boolean để xác định kết quả. Cú pháp của nó như sau:

```
CASE
    WHEN Boolean_expression THEN result
        [...n
    [ELSE else_result
END
```

Trong đó:

+ `case_expression`: Biểu thức dùng để SQL Server đánh giá giá trị trong câu lệnh Simple CASE.
+ `Expression`: Giá trị dùng để so sánh với biểu thức `case_expression` nếu đúng thì nó sẽ trả về kết quả.
+ `Result`: Kết quả sẽ được trả về nếu như giá trị biểu thức `case_expression` so với `Expression` là đúng.
+ `Boolean_expression`: SQL Server dùng biểu thức Boolean để rẽ nhánh, nếu biểu thực nhận giá trị True thì sẽ thực hiện kết quả `Result.`
+ `else_result`: Thực hiện các kết quả sau ELSE.

Ví dụ 4.8. Sử dụng cấu trúc rẽ nhánh CASE dùng trong cả hai trường hợp Simple Case và Searched Case.

```
Use pubs
Go
CREATE PROCEDURE uspCheckNumberCase
    @chrNumber    CHAR(2)
AS
IF  (CONVERT(INT,  @chrNumber)  <  1)  OR  (CONVERT(INT,
@chrNumber) > 10)
  BEGIN
     SELECT CASE
        WHEN CONVERT(INT, @chrNumber) < 1 THEN 'Number
is less than 1.'
        WHEN CONVERT(INT, @chrNumber) > 10 THEN 'Number
is greater than 10.'
     END
     RETURN
  END
SELECT CASE CONVERT(INT, @chrNumber)
    WHEN 1 THEN 'One'
    WHEN 2 THEN 'Two'
    WHEN 3 THEN 'Three'
    WHEN 4 THEN 'Four'
    WHEN 5 THEN 'Five'
    WHEN 6 THEN 'Six'
    WHEN 7 THEN 'Seven'
    WHEN 8 THEN 'Eight'
    WHEN 9 THEN 'Nine'
    WHEN 10 THEN 'Ten'
END
```

d) Cấu trúc WAITFOR

Cấu trúc WaitFor được dùng để ngăn việc thực thi một lô, thủ tục, hay một giao dịch cho đến một thời điểm nào đó hoặc sau một khoảng thời gian nào đó. Cú pháp của WAITFOR như sau:

```
WAITFOR { DELAY 'time' | TIME 'time' }
```

Trong đó:

+ `DELAY`: Chỉ định khoảng thời gian phải chờ. Tối đa là 24 giờ.
+ `TIME`: Chỉ định thời điểm thực thi một lô, thủ tục, hay một giao dịch.

Ví dụ 4.9. Sử dụng cấu trúc WAITFOR để chờ đến lúc 21^h30 thì thực hiện xóa bản ghi.

```
BEGIN
   WAITFOR TIME '21:30'
   DELETE FROM DMLOP WHERE MALOP='TH6A'
END
```

Ví dụ 4.10. Xây dựng thủ tục time_delay để chờ trong một khoảng thời gian nào đó và đưa ra thông báo khoảng thời gian đã chờ đó.

```
CREATE PROCEDURE time_delay @DELAYLENGTH char(9)
AS
DECLARE @RETURNINFO varchar(255)
BEGIN
    WAITFOR DELAY @DELAYLENGTH
    SELECT @RETURNINFO = 'A total time of ' +
        SUBSTRING(@DELAYLENGTH, 1, 2) +
        ' hours, ' +
        SUBSTRING(@DELAYLENGTH, 4, 2) +
        ' minutes, and ' +
        SUBSTRING(@DELAYLENGTH, 7, 2) +
        ' seconds ' +
        'has elapsed! Your time is up.';
    PRINT @RETURNINFO;
END;
GO
-- This next statement executes the time_delay procedure.
EXEC time_delay '00:05:00'
GO
```

e) Cấu trúc TRY…CATCH

Trong SQL Server 2005, cấu trúc TRY … CATCH được sử dụng để quản lý lỗi tương tự như các ngôn ngữ lập trình VB.NET, $C^{\#}$ và C^{++}. Cú pháp:

```
BEGIN TRY
      { sql_statement | statement_block }
END TRY
BEGIN CATCH
      { sql_statement | statement_block }
END CATCH[ ; ]
```

Hoạt động của cấu trúc TRY… CATCH:

+ Cấu trúc TRY…CATCH gồm hai phần: Khối TRY và khối CATCH. Khi một điều kiện lỗi được dò thấy ở một câu lệnh Transact-SQL thuộc khối TRY, điều khiển được chuyển sang khối

CATCH để xử lý. Sau khi khối CATCH điều khiển ngoại lệ, điều khiển được chuyển cho câu lệnh Transact-SQL ngay sau lệnh END CATCH.

+ Nếu không lỗi trong khối TRY, điều khiển được chuyển ngay lập tức cho câu lệnh sau END CATCH.

Ví dụ 4.11. Sử dụng cấu trúc TRY … CATCH để điều khiển lỗi.

```
BEGIN TRY
INSERT INTO [QLDiemSV].[dbo].[DMLOP]([MaLop],[TenLop], [Khoa])
     VALUES ('TH6A','Tin học 6A','6')
END TRY
BEGIN CATCH
Print ERROR_MESSAGE()
END CATCH
```

Ví dụ 4.11. Xây dựng thủ tục đưa ra thông tin lỗi.

```
USE QLDiemSV;
GO
BEGIN TRANSACTION;

BEGIN TRY
    -- Generate a constraint violation error.
    DELETE FROM LOP
        WHERE MaLop='TH5A';
END TRY
BEGIN CATCH
    SELECT
        ERROR_NUMBER() AS ErrorNumber,
        ERROR_SEVERITY() AS ErrorSeverity,
        ERROR_STATE() as ErrorState,
        ERROR_PROCEDURE() as ErrorProcedure,
        ERROR_LINE() as ErrorLine,
        ERROR_MESSAGE() as ErrorMessage;

    IF @@TRANCOUNT > 0
        ROLLBACK TRANSACTION;
END CATCH;

IF @@TRANCOUNT > 0
    COMMIT TRANSACTION;
GO
```

f) Functions - Hàm

Hàm được dùng hoặc là định dạng và thao tác dữ liệu hoặc là trả về thông tin cho người sử dụng. Có hai loại hàm: hàm do hệ thống định nghĩa

hoặc hàm do người dùng định nghĩa. Hàm do hệ thống định nghĩa được tạo do Microsoft và được cài đặt khi SQL Server cài đặt. Hàm do người dùng định nghĩa được định nghĩa bởi người sử dụng bằng cách sử dụng câu lệnh CREATE FUNCTION . Đối với loại hàm này ta sẽ thảo luận chúng trong phần tiếp theo của chương.

Các hàm do hệ thống định nghĩa được chia thành các kiểu hàm sau: String functions, Date functions, Mathematical functions, aggregate Functions, System functions,.v.v...

- ***String functions:*** Là các hàm thao tác với dữ liệu kiểu ký tự. Sau đây là một số hàm thông dụng.

  + `CHARINDEX(string1, string2, start_position)`: Tìm vị trí bắt đầu của chuỗi ký tự chỉ định string1 trong chuỗi string2 và bắt đầu tìm ở vị trí start_position trong chuỗi string2.

    Ví dụ 4.9. Sử dụng hàm `CHARINDEX`

    ```
    SELECT CHARINDEX('test', 'This is a test', 1)
    ```

    Hàm sẽ trả về giá trị 11, vị trí bắt đầu của chuỗi 'test' trong chuỗi 'This is a test'.

  + `LEFT (string, number_of_characters)`: Trả về chuỗi gồm `number_of_characters` ký tự tính từ trái sang của chuỗi `string`.

    Ví dụ 4.10. Sử dụng hàm `LEFT`

    ```
    SELECT LEFT('This is a test', 4)
    ```

    Hàm sẽ trả về chuỗi 'This'

  + `LEN(string)`: Xác định độ dài của chuỗi ký tự *`string`*.

    Ví dụ 4.11. Sử dụng hàm `LEN`

    ```
    SELECT LEN('This is a test')
    ```

    Hàm sẽ trả về giá trị 14

  + `LOWER(string)`: Hàm trả về chuỗi ký tự thường.

Ví dụ 4.12. Sử dụng hàm `LOWER`

```
SELECT LOWER('This is a TEST')
```

Hàm sẽ trả về chuỗi 'this is a test'

+ `LTRIM(string)`: Cắt bỏ các ký tự trắng bên trái của chuỗi.

Ví dụ 4.13. Sử dụng hàm `LTRIM`

```
SELECT LTRIM('    This is a test    ')
```

Hàm sẽ trả về chuỗi 'This is a test   '

+ `RIGHT (string, number_of_characters)`: trả về chuỗi gồm `number_of_characters` ký tự tính từ phải sang của chuỗi `string`.

Ví dụ 4.14. Sử dụng hàm `RIGHT`

```
SELECT RIGHT('This is a test', 4)
```

Hàm sẽ trả về chuỗi 'test'

+ `RTRIM(string)`: Cắt bỏ các ký tự trắng bên phải của chuỗi.

Ví dụ 4.15. Sử dụng hàm `RTRIM`

```
SELECT RTRIM('    This is a test    ')
```

Hàm sẽ trả về chuỗi '   This is a test'

+ `SUBSTRING ( expression ,start , length )`: Hàm trả về chuỗi con gồm `length` ký tự của `expression` tính từ vị trí `start`.

```
SELECT x = SUBSTRING('abcdef', 2, 3)
```

+ `UPPER(string)`: Chuyển đổi các ký thường thành chữ hoa.

Ví dụ 4.16. Sử dụng hàm `UPPER`

```
SELECT UPPER('This is a TEST')
```

Hàm sẽ trả về chuỗi 'THIS IS A TEST'

Chú ý: Cần phải cẩn thận khi sử dụng các hàm, chẳng hạn khi ta sử dụng hàm UPPER trong vế trái của toán tử so sánh. Khi đó nó sẽ bắt SQL Server

phải thực hiện trên một bảng để tìm kiếm giá trị. Ta xét hai truy vấn trong ví dụ 4.17 sau:

```
select au_lname from authors where au_lname = 'Green'
select au_lname from authors where upper(au_lname) =
'Green'
```

Đối với truy vấn thứ hai, sử dụng hàm Upper mất thời gian lâu hơn so với truy vấn thứ nhất.

- ***Date Functions:*** Là các hàm làm việc với dữ liệu kiểu datetime. Một số hàm làm việc với các kiểu thông tin đặc biệt được gọi là `datepart`. Trước khi đi vào các hàm, ta xét các ký hiệu của `datepart` cho trong bảng 4.3.

Bảng 4.3. Các thành phần `datepart`

Ký hiệu `datepart`	`datepart`
yy	Year (năm)
yyyy	Year (năm)
q	Quarter (quý)
qq	Quarter (quý)
m	Month (tháng)
mm	Month (tháng)
dy	Dayofyear
y	Dayofyear
d	Day
dd	Day
wk	Week
ww	Week
dw	weekday
hh	hour
mi	Minute
n	minute
ss	Second

s	Second
ms	millisecond

Sau đây là một số hàm hay sử dụng:

+ `DATEADD (datepart, amount, date)`: Cộng thêm một số `amount` thời gian thành phần `datepart` của `date`.

Ví dụ 4.17. Sử dụng hàm `DATEADD`

```
SELECT DATEADD(year, 1, GETDATE())
```

Hàm sẽ trả về ngày hiện tại cộng thêm một năm.

+ `DATEDIFF (datepart, date1, date2)`: So sánh điểm khác nhau giữa hai ngày bằng việc sử dụng tham số `datepart`.

Ví dụ 4.18. Sử dụng hàm `DATEDIFF`

```
SELECT DATEDIFF(hour, '1/1/2008 12:00:00',
'1/1/2008 16:00:00')
```

Hàm sẽ trả về giá trị 4. Đây là điểm khác nhau giữa hai ngày, hai ngày chênh nhau 4 giờ.

```
SELECT DATEDIFF(hour, '1/1/2008 12:00:00',
'1/2/2008 16:00:00')
```

Hàm sẽ trả về giá trị 28. Đây là điểm khác nhau giữa hai ngày, hai ngày chênh nhau 28 giờ.

+ `DATEPART(datepart, date)`: Hàm trả về giá trị của thành phần `datepart` trong `date`.

Ví dụ 4.19. Sử dụng hàm `DATEPART`

```
SELECT DATEPART(month, '1/1/2008
16:00:00')
```

Hàm sẽ trả về giá trị tháng 1.

+ `DAY(date)`: Xác định số ngày của tháng trong dữ liệu ngày giờ date.

Ví dụ 4.20. Sử dụng hàm `DAY`

```
SELECT DAY ('7/22/1979 00:04:00')
```

Hàm sẽ trả về giá trị ngày là 22.

+ `GETDATE()`: Trả về giá trị ngày hiện tại của hệ thống.
+ `MONTH(date)`: Tương tự như hàm DAY, hàm MONTH trả về tháng của dữ liệu ngày giờ.
+ `YEAR(date)`: Trả về năm của dữ liệu ngày giờ.

- ***Mathematical Functions:*** Sau đây ta trình bày một số hàm toán học thông thường.
  + `ABS(number)`: Trả về giá trị tuyệt đối của số `number`.
  + `CEILING(number)`: Trả về số nguyên nhỏ nhất lớn hơn hoặc bằng `number`.
  + `FLOOR(number)`: Trả về số nguyên lớn nhất nhỏ hơn hoặc bằng `number`.
  + `ROUND(number,precision)`: Hàm làm tròn số `number` lấy `precision` chữ số sau dấu thập phân.
  + `SQUARE(number)`: Hàm trả về giá trị bình phương số `number`.
  + `SQRT(number)`: Hàm trả về giá trị căn bậc hai số `number`.
- ***Aggregate Functions*:** Các hàm tập hợp thực hiện tính toán trên một tập hợp các giá trị và trả về một giá trị đơn. Ngoại trừ hàm COUNT, hàm tập hợp bỏ qua các giá trị NULL.

  Các hàm tập hợp thường sử dụng với mệnh đề GROUP BY trong khối câu lệnh SELECT. Hàm tập hợp được phép dùng như là các biểu thức trong trường hợp:

  - Trong danh sách select của khối câu lệnh SELECT.

- Trong mệnh đề COMPUTE hoặc COMPUTE BY .
- Trong mệnh đề HAVING

Sau đây là một số hàm tập hợp hay được sử dụng:

+ `AVG ([ALL|DISTINCT]expression)`: Hàm trả về giá trị trung bình của tập các giá trị trong một nhóm.
  - `ALL`: Áp dụng cho các hàm tập hợp để chỉ định cho tất cả các giá trị. ALL là từ khóa mặc định.
  - `DISTINCT:` Chỉ định chỉ lấy một thể hiện duy nhất của một giá trị. Nghĩa là trong tập hợp có nhiều phần tử có cùng một giá trị thì chỉ lấy một giá trị đại diện cho nó.

Ví dụ 4.21. Sử dụng hàm `AVG`

```
USE pubs
SELECT AVG(advance), SUM(ytd_sales)
FROM titles
WHERE type = 'business'
```

+ `COUNT({[ALL|DISTINCT]expression]|*})`: Hàm trả về kiểu int số các phần tử của một nhóm.

Chú ý. Sử dụng hàm `COUNT`

- `COUNT(*)`: Trả về số các phần tử trong một nhóm bao gồm cả giá trị NULL và giá trị duplicates.
- `COUNT(ALL` ***`expression`***`)`: Thực hiện định giá trị cho *expression* tại mỗi dòng trong nhóm và trả về số các giá trị không NULL.
- `COUNT(DISTINCT` ***`expression`***`)`: Thực hiện định giá trị cho *expression* tại mỗi dòng trong nhóm và trả về số các giá trị duy nhất và không NULL.

Ví dụ 4.22. Sử dụng hàm `COUNT`

```
USE pubs
```

```
GO
SELECT COUNT(DISTINCT city)
FROM authors
GO
```

+ COUNT_BIG({[ALL|DISTINCT]expression}|*): Trả về số các phần tử trong một nhóm. Hàm COUNT_BIG làm việc như hàm COUNT. Điểm khác nhau giữa chúng là hàm COUNT trả về giá trị kiểu int còn hàm COUNT_BIG trả về giá trị kiểu bigint.

+ MAX ([ALL|DISTINCT]expression): Trả về giá trị lớn nhất trong biểu thức expression.

+ MIN ([ALL|DISTINCT]expression): Trả về giá trị lớn nhất trong biểu thức expression.

+ SUM([ALL|DISTINCT]expression): Trả về tổng của tất cả các giá trị của biểu thức hoặc tổng các giá trị DISTINCT của biểu thức expression. Hàm SUM chỉ áp dụng cho các cột kiểu số. Các giá trị NULL được bỏ qua.

Ví dụ 4.23. Sử dụng hàm SUM

```
USE pubs
GO
-- Aggregate functions
SELECT type, SUM(price), SUM(advance)
FROM titles
WHERE type LIKE '%cook'
GROUP BY type
ORDER BY type
GO
```

Ví dụ 4.24. Sử dụng hàm SUM để tính điểm trung bình trong CSDL QLDiemSV

```
SELECT DIEM.Masv,(Convert(real, Sum(
dbo.fncDiemCN(DiemL1,DiemL2)*MONHOC.SD
```

```
VHT))/convert(real,Sum(MONHOC.SDVHT)))
AS DTB
FROM DIEM INNER JOIN MONHOC ON
DIEM.MAMH=MONHOC.MaMH
GROUP BY DIEM.MaSV
```

- ***System Functions:*** Các hàm hệ thống là các hàm lấy thông tin hệ thống về các đối tượng và đã thiết lập trong SQL Server.

  + `CONVERT (data_type, expression)`: Chuyển đổi biểu thức `expression` thành kiểu dữ liệu `data_type`.

    Ví dụ 4.25. Sử dụng hàm `CONVERT`

    ```
    SELECT CONVERT(VARCHAR(5), 12345)
    ```

    Hàm sẽ trả về chuỗi '12345'.

  + `CAST (expression AS data_type)`: Chuyển đổi biểu thức `expression` thành kiểu dữ liệu `data_type`.

  + `CURRENT_USER`: Trả về người sử dụng hiện tại.

    Ví dụ 4.26. Sử dụng hàm `CURRENT_USER`

    ```
    SELECT CURRENT_USER
    ```

  + `DATALENGTH(expression)`: Trả về số byte được sử dụng trong biểu thức `expression`.

  + `HOST_NAME()`: trả về tên máy tính mà người sử dụng hiện tại đang login.

    Ví dụ 4.27. Sử dụng hàm `HOST_NAME()`

    ```
    SELECT HOST_NAME()
    ```

  + `SYSTEM_USER`: Hàm trả về tên của các User đang login hệ thống.

    Ví dụ 4.28. Sử dụng hàm `SYSTEM_USER`

    ```
    SELECT SYSTEM_USER
    ```

```
[ WITH <procedure_option> [ ,...n ]
AS { [ BEGIN ] statements [ END ] }
[;]

<procedure_option> ::=
    [ ENCRYPTION ]
    [ RECOMPILE ]
```

Trong đó:

+ *procedure_name*: là tên của store procedure sẽ được tạo.
+ *parameter:* Là các tham số truyền vào store procedure, ta phải định nghĩa chúng trong phần khai báo của store procedure. Khai báo báo gồm tên của tham số (trước tên tham số sử dụng tiền tố @), kiểu dữ liệu của tham số và một số chỉ định đặc biệt phụ thuộc vào mục đích sử dụng của tham số đó.
+ ; number: Là số nguyên tùy chọn được sử dụng trong nhóm các thủ tục có cùng tên.
+ *data type:* Kiểu của tham số trong phần khai báo.
+ *[VARYING]*: Đây là tùy chọn được chỉ định khi cursor trả về như một tham số.
+ *[= default]* : Gán giá trị mặc định cho tham số. Nếu không gán giá trị mặc định thì tham số nhận giá trị NULL.
+ OUTPUT: Đây là từ khóa chỉ định tham số đó là tham số xuất. Tham số xuất không dùng được với kiểu dữ liệu Text và image.
+ [,...n]: Chỉ định rằng có thể khai báo nhiều tham số.
+ RECOMPILE:Chỉ định Database Engine không xây dựng kế hoạch cho thủ tục này và thủ tục sẽ được biên dịch tại thời điểm thực thi thủ tục.
+ ENCRYPTION:Chỉ định SQL Server sẽ mã hóa bản text lệnh CREATE PROCEDURE. Users không thể truy cập vào các bảng hệ thống hoặc file dữ liệu để truy xuất bản text đã mã hóa.

**** Thực thi store procedure trong SQL Server:*** Để thực thi một thủ tục trong SQL Server ta sử dụng cú pháp sau:

```
{ EXEC | EXECUTE }
    { module_name [ ;number ]}
        [ [ @parameter = ] { value
                           | @variable [ OUTPUT ]
                           | [ DEFAULT ]
                           }
        ]
      [ ,...n ]
      [ WITH RECOMPILE ]
```

Trong đó:

+ module_name: Là tên thủ tục cần thực hiện.
+ ;number: Chỉ định thủ tục trong nhóm thủ tục cùng tên.
+ @parameter: Tên tham số trong thủ tục.
+ @variable: Chỉ định biến chứa các tham số hoặc trả về tham số.
+ DEFAULT: Chỉ định lấy giá trị mặc định của biến.

Ví dụ 4.30. Xây dựng thủ tục XemDSSV.

```
Use QLDiemSV
Go
IF EXISTS(Select name from sysobjects
          where name ='p_DSSV' and type='p')
DROP PROCEDURE p_DSSV
GO
CREATE PROCEDURE p_DSSV
AS
SELECT MaSV, Hodem + ' '+TensV as Hoten, Ngaysinh, MaLop
      From HOSOSV
GO
```

- Thực thi thủ tục p_DSSV

```
Use QLDiemSV
Go
EXEC p_DSSV
```

**** Truyền tham số nhập vào trong store procedure.***

Ví dụ 4.32. Xây dựng thủ tục pp_DSSV để hiển thị danh sách sinh viên theo tham số mã lớp. Mã lớp được truyền vào khi thủ tục được thực hiện.

```
Use QLDiemSV
Go
IF EXISTS(Select name from sysobjects
where name ='p_DSSV' and type='p')
DROP PROCEDURE p_DSSV
GO
CREATE PROCEDURE p_DSSV
@parMaLop Varchar(10)='TH%'
AS
 SELECT MaSV, Hodem + ' '+TensV as Hoten, Ngaysinh
From HOSOSV Where MaLop like @parMaLop
GO
```

Gọi thực thi thủ tục trên với truyền giá trị cho tham số nhập như sau:

```
EXEC p_DSSV 'TH03A'
EXEC p_DSSV @parMaLop=DEFAULT
```

**** Sử dụng tham số xuất trong store procedure.***

Ví dụ 4.33. Xây dựng thủ tục pp_Siso để xuất giá trị sĩ số của một lớp theo tham số mã lớp. Mã lớp được truyền vào khi thủ tục được thực hiện.

```
Use QLDiemSV
Go
IF EXISTS(Select name from sysobjects where name
='pp_Siso' and type='p')
DROP PROCEDURE pp_Siso
GO
CREATE PROCEDURE pp_Siso
@parMaLop Char(10), @parSiso Int OUTPUT
AS
 SELECT @parSiso=count(*)
    From HOSOSV Where MaLop=@parMaLop
GO
DECLARE @siso int
```

```
exec pp_Siso 'TH03A',@parSiso=@siso OUTPUT
Print 'Si so lop TH03A là :'+ convert(varchar(3),@siso)
Go
```

Kết quả thực hiện chương trình:

```
Si so lop TH03A là :12
```

**** Sử dụng biến cục bộ:*** Các biến cục bộ được sử dụng trong bó lệnh, trong chương trình gọi (Scipt) hoặc trong thủ tục (xem ví dụ 4.5 và 4.6). Biến cục bộ thường được giữ các giá trị sẽ được kiểm tra trong phát biểu điều kiện và giữ giá trị sẽ được trả về bởi lệnh RETURN. Phạm vị của biến cục bộ trong store procedure là từ điểm biến đó được khai báo cho đến khi thoát store procedure. Ngay khi store procedure kết thúc thì biến đó không được tham chiếu nữa. Cú pháp khai báo biến cục bộ:

```
DECLARE <parameter> [AS] <data type>
```

Giống như khai báo các biến ở trên, trước tên biến phải có tiền tố @. Giá trị khởi tạo ban đầu của biến là NULL.

Để thiết lập giá trị của biến ta sử dụng cú pháp:

```
SET <parameter> = <expression>
SELECT <parameter> = <expression>
```

**** Câu lệnh PRINT:*** Dùng để hiển thị chuỗi thông báo tới người sử dụng. Chuỗi thông báo này nó thể dài tới 8000 ký tự. Cú pháp của lệnh PRINT như sau:

```
PRINT < messages>
```

**** Sử dụng SELECT đề trả về giá trị:*** Ta có thể trả về giá trị bằng việc sử dụng SELECT trong thủ tục hoặc trả về kết quả thiết lập từ truy vấn SELECT.

Ví dụ 4.34. Xây dựng thủ tục pp_Siso để xuất giá trị sĩ số của một lớp theo tham số mã lớp ra ngoài. Mã lớp được truyền vào khi thủ tục được thực hiện.

```
Use QLDiemSV
Go
```

```
    IF EXISTS(Select name from sysobjects where name
='pp_Siso' and type='p')
    DROP PROCEDURE pp_Siso
    GO
    CREATE PROCEDURE pp_Siso
    @parMaLop Char(10), @parSiso Int OUTPUT
    AS
     SELECT     @parSiso=count(*)     From     HOSOSV     Where
MaLop=@parMaLop
    GO
    DECLARE @siso int
    exec pp_Siso 'TH03A',@parSiso=@siso OUTPUT
    SELECT 'Si so lop TH03A là :'= @siso
    Go
```

**** Lệnh RETURN:*** Ta có thể sử dụng lệnh RETURN để thoát không điều kiện khỏi thủ tục. Khi lệnh RETURN được thực thi trong thủ tục, khi đó các câu lệnh sau RETURN trong thủ tục sẽ bị bỏ qua và thoát khỏi thủ tục để trở về dòng lệnh tiếp theo trong chương trình gọi.

Ngoài ra, ta có thể sử dụng lệnh RETURN để trả về giá trị cho chương trình gọi, giá trị trả về phải là một số nguyên, nó có thể là một hằng số hoặc một biến. Cú pháp như sau:

```
RETURN [ integer_expression ]
```

Ví dụ 4.35. Cho CSDL pubs. Xây dựng thủ tục `usp_4_31` kiểm tra một chủ đề có tồn tại trong bảng `titles` hay không? Nếu tồn tại một chủ đề thì hiển thị chủ đề đó. Nếu không tồn tại chủ đề đó thì thủ tục trả về giá trị 1 hoặc có nhiều hơn một chủ đề đó thì trả về giá trị 2.

```
Use pubs
Go
IF EXISTS(Select name from sysobjects
          where name ='usp_4_31' and type='p')
DROP PROCEDURE usp_4_31
GO
CREATE PROCEDURE usp_4_31
    @vchTitlePattern    VARCHAR(80) = '%'
AS
SELECT @vchTitlePattern = '%' + @vchTitlePattern + '%'
IF (SELECT COUNT(*) FROM titles
                   WHERE title LIKE @vchTitlePattern) < 1
BEGIN
    RETURN 1
END
IF (SELECT COUNT(*) FROM titles
                   WHERE title LIKE @vchTitlePattern) > 1
```

```
        From HOSOSV Where MaLop=@parMaLop
GO
```

b) Xóa store procedure

Cú pháp:

```
DROP{ PROC|PROCEDURE}{[schema_name.] procedure }
```

Ví dụ 4.37. Xóa thủ tục p_DSSV:

```
Use QLDiemSV
Go
DROP PROCEDURE p_DSSV
Go
```

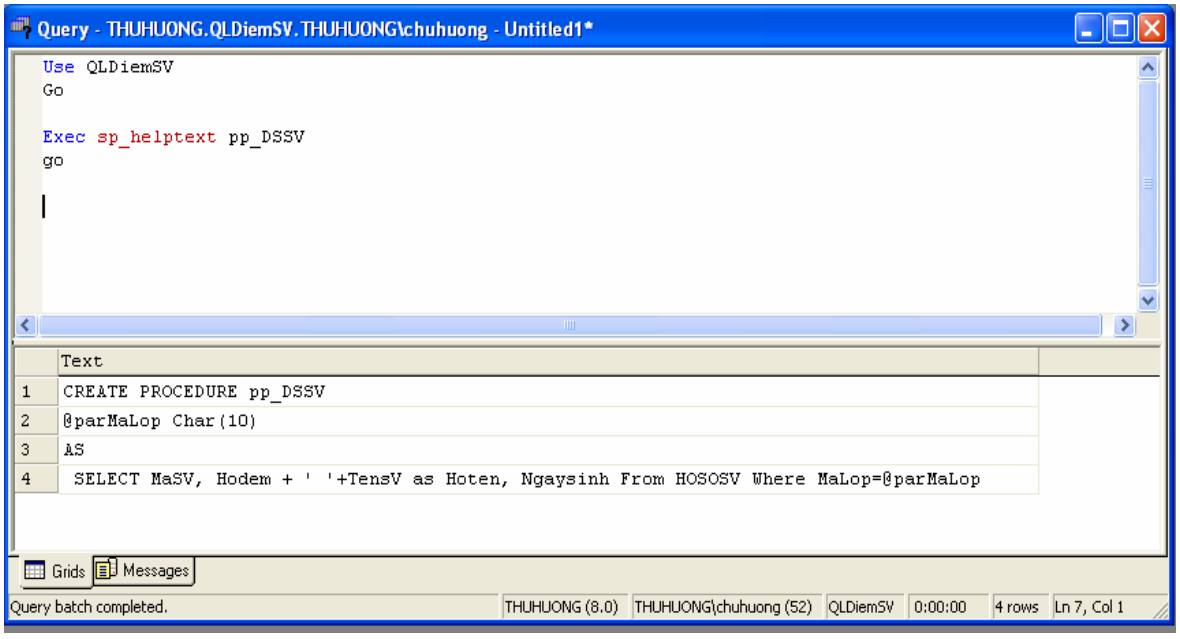
c) Xem nội dung store procedure

Để xem nội dung của thủ tục ta sử dụng thủ tục hệ thống sp_helptext.

Ví dụ 4.38. Xem nội dung thủ tục pp_DSSV:

```
Use QLDiemSV
Go
Exec sp_helptext pp_DSSV
Go
```

Kết quả việc thi hành các câu lệnh này cho trong hình 4.8.



Hình 4.8. Sử dụng thủ tục sp_helptext

+ `parameter_name`: Là các tham số Input cho hàm. Các tham số này xây dựng cũng tương tự như trong stored procedure.
+ `scalar_data_type`: Là kiểu dữ liệu vô hướng của tham số. Một hàm có thể nhận bất kỳ kiểu dữ liệu nào như là tham số trừ các kiểu timestamp, cursor, text, ntext, image.
+ `default`: Chỉ định giá trị mặc định cho tham số, tương tự như trong stored procedure.
+ `[,...n]`: Chỉ định một hàm có thể tạo nhiều tham số. Một hàm trong SQL Server có thể chứa tới 1024 tham số.
+ `RETURNS`: từ khóa này chỉ định kiểu dữ liệu hàm sẽ trả về. Kiểu dữ liệu của hàm có thể là một kiểu dữ liệu vô hướng hoặc một bảng.
+ `scalar_data_type`: Ta sẽ chỉ định kiểu dữ liệu nếu như hàm trả về một giá trị vô hướng. Ở đây ta phải chỉ định kiểu độ dài dữ liệu.
+ `TABLE`: Đây là kiểu dữ liệu cho phép hàm có thể trả về nhiều dòng dữ liệu.
+ `column_definition`: Định nghĩa các cột cho kiểu dữ liệu TABLE. Các cột này được định nghĩa tương tự như định nghĩa các cột trong bảng.
+ `table_constraint`: Định nghĩa các ràng buộc trong kiểu dữ liệu TABLE này.
+ `[,...n]`: Chỉ định có thể có nhiều cột và nhiều ràng buộc trong bảng.
+ `WITH ENCRYPTION`: Từ khóa chỉ định code của hàm sẽ được mã hóa trong bảng `syscomments`.
+ `SCHEMABINDING`: Từ khóa này chỉ định hàm được tạo để buộc vào tất cả các đối tượng mà nó tham chiếu.

```
    ELSE IF DATEPART(dw, @dtmDateStart) = 7
    BEGIN
        RETURN(DATEADD(dw, 4, @dtmDateStart))
    END

    RETURN(DATEADD(dw, 3, @dtmDateStart))
END
```

Thực hiện thử nghiệm hàm trên, ta xây dựng scipt sau:

```
DECLARE @dtmDate    DATETIME
SELECT      @dtmDate = '1/10/2008'
SELECT      DATENAME(dw, @dtmDate)
SELECT      DATENAME(dw, dbo.
                        fncGetThreeBusinessDays(@dtmDate))
```

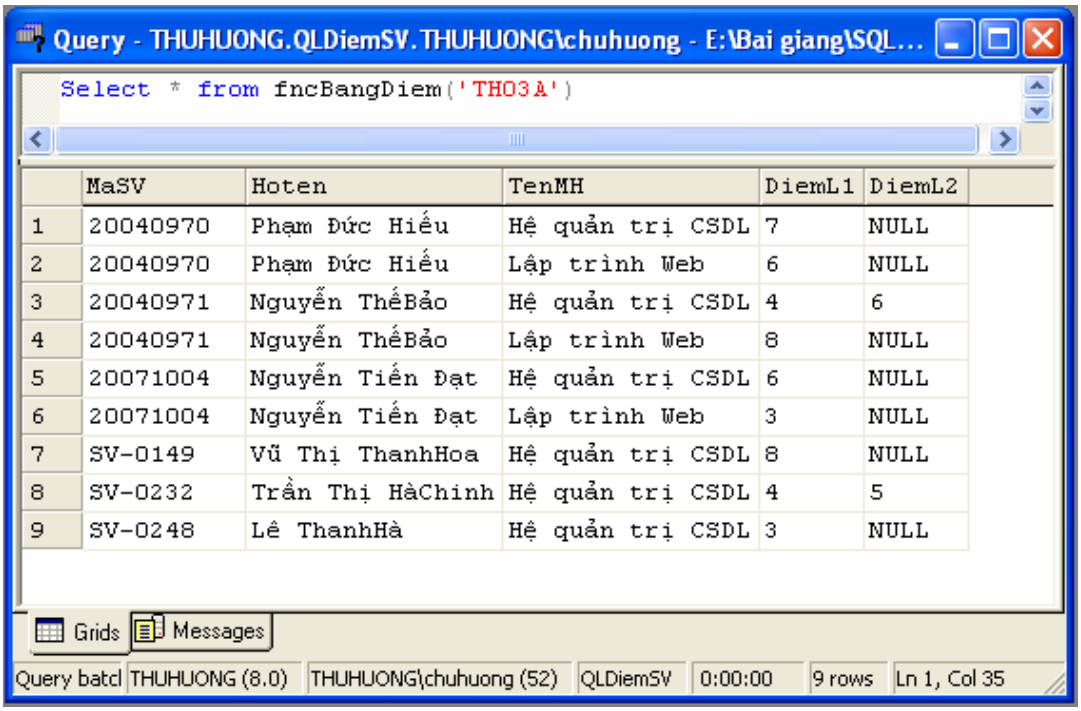
Ví dụ 4.39. Sử dụng hàm `fncGetThreeBusinessDays` trên để tính toán trên một cột trong một bảng.

```
CREATE TABLE OrderInfo
(
    OrderID         INT NOT NULL,
    ShippingMethod VARCHAR(16) NOT NULL,
    OrderDate       DATETIME NOT NULL DEFAULT GETDATE(),
    ExpectedDate    AS(
            dbo.fncGetThreeBusinessDays(OrderDate)
        )
)
GO

INSERT OrderInfo VALUES (1, 'UPS GROUND', GETDATE())
INSERT OrderInfo VALUES (2, 'FEDEX STANDARD',
                            DATEADD(dd, 2, GETDATE()))
INSERT OrderInfo VALUES (3, 'PRIORITY MAIL',
                            DATEADD(dd, 4, GETDATE()))
GO

SELECT OrderID,ShippingMethod,CONVERT(VARCHAR(12),
       OrderDate,1) + '(' + DATENAME(dw, OrderDate) + ')'
 AS 'OrderDate', CONVERT(VARCHAR(12),ExpectedDate,1)+
 '('+ DATENAME(dw, ExpectedDate) + ')' AS 'ExpectedDate'
FROM     OrderInfo
```

Ví dụ 3.40. Xây dựng hàm trả về các dòng dữ liệu gồm thông tin về điểm của các môn học theo Mã lớp.

```
CREATE FUNCTION fncBangDiem(@MaLop CHAR(10))
    RETURNS @TableName TABLE
    (   MaSV    CHAR(10),
        Hoten   nvarchar(100),
        TenMH    NVARCHAR(50),
        DiemL1   INT,
        DiemL2   INT    )
AS
BEGIN
INSERT INTO @TableName
    SELECT Di.MaSV, Ho.HoDem + ' '+ Ho.TenSV, Mo.TenMH,
                                    Di.DiemL1,Di.DiemL2
    FROM   DIEM Di
        JOIN HOSOSV Ho ON (Di.MaSV = Ho.MaSV)
        JOIN MONHOC Mo ON (Di.MaMH = Mo.MaMH)
    WHERE   Ho.MaLop = @MaLop
RETURN
END
GO
```

Thử nghiệm gọi hàm này trong đoạn script chương trình sau và kết quả cho trong hình 4.10: Select * from fncBangDiem('TH03A')



Hình 4.10. Gọi hàm `fncBangDiem`

- **AFTER Triggers:** Khi các câu lệnh thay đổi dữ liệu được thực hiện trên một bảng có định nghĩa trigger, một vài xử lý xuất hiện trước khi trigger thực sự nổ. Đầu tiên bộ truy vấn sẽ kiểm tra trên bảng có bất cứ các ràng buộc nào hay không. Nếu có, SQL Server sẽ tiến hành kiểm tra tính hợp lệ của dữ liệu trên mọi ràng buộc nào. Nếu dữ liệu đang thêm vào hoặc đang sửa đổi mà không thỏa các ràng buộc thì bộ truy vấn sẽ dừng câu lệnh trên và khi đó trigger sẽ không được kích nổ. Ngược lại, khi kiểm tra thành công thì các trigger sẽ được thực thi.

  Trước khi bất cứ câu lệnh nào chứa trigger thực sự được thực hiện, SQL Server tạo ra hai bảng đặc biệt lưu trong bộ nhớ có cùng cấu trúc với bảng mà trên đó trigger được tạo.

  + Table INSERTED chứa các giá trị đang được Add vào bảng.
  + Table DELETED chứa các giá trị đang bị xóa từ bảng.

  Nếu trigger được định nghĩa là INSERT trigger thì SQL Server sẽ chỉ tạo bảng INSERTED, còn nếu là DELETE trigger thì thì SQL Server sẽ chỉ tạo bảng DELETED. Cuối cùng là nếu trigger được định nghĩa là UPDATE trigger thì SQL Server sẽ tạo cả hai bảng vì INSERTED sẽ chứa ảnh của hàng sau khi thay đổi còn bảng DELETED chứa ảnh của hàng trước khi thay đổi.

- ***INSTEAD OF trigger:*** Là một đặc điểm thêm vào của SQL Server 2000. Như tên gọi đã ám chỉ, INSTEAD OF trigger được kích nổ thay cho hành động được sử dụng để kích nó.

  Nghĩa là, nếu một trigger được định nghĩa là INSTEAD OF INSERT trigger thì trigger này sẽ được nổ khi câu lệnh Insert được thực hiện trên bảng. Sau khi câu lệnh sửa đổi dữ liệu được gửi đi thì INSTEAD OF trigger được kích nổ và hiện lập tức ngay lập tức. Các ràng buộc không được kiểm tra trước khi trigger được kích nổ, mặc dù các bảng INSERTED, DELETED vẫn được tạo. Sau khi các bảng này được tạo thì quá trình xử lý trigger tương tự như quá trình xử lý của stored procedure. INSTEAD OF trigger có thể được tạo trên bảng hoặc trên view.

+ *AFTER*: Chỉ định AFTER trigger. Tùy chọn này chỉ định riêng không lẫn với từ khóa INSTEAD OF. AFTER triggers không được định nghĩa trên view.
+ *INSTEAD OF*: Từ khóa chỉ định đây là INSTEAD OF trigger. Tùy chọn này chỉ định để không lẫn với từ khóa AFTER.
+ *DELETE*: Chỉ định rằng trigger ta đang tạo sẽ được kích nổ đáp ứng với hành động DELETE của bảng hoặc view.
+ *INSERT*: Chỉ định rằng trigger ta đang tạo sẽ được kích nổ đáp ứng với hành động INSERT của bảng hoặc view.
+ *UPDATE*: Chỉ định rằng trigger ta đang tạo sẽ được kích nổ đáp ứng với hành động UPDATE của bảng hoặc view.
+ *NOT FOR REPLICATION*: Chỉ định rằng bất cứ bản sao nào của các hành động chạy ngầm dưới bảng này đều không được kích nổ trigger này.
+ *AS*: Chỉ định phần code của trigger bắt đầu từ đây.
+ *IF UPDATE (column)*: Được dùng trong các trigger INSERT, UPDATE. Cấu trúc này được dùng để kiểm tra các sửa đổi trên cột chỉ định và sau đó là các hành động trên nó.
+ *{AND | OR} UPDATE (column)*: Chỉ định rằng ta có thể sử dụng chuỗi các cấu trúc UPDATE với nhau để kiểm tra một vài cột tại cùng một thời điểm.
+ ...n: Chỉ định ta có thể lặp lại các cấu trúc trên nếu cần thiết.
+ IF(COLUMNS_UPDATED():Chỉ dùng trong các trigger INSERT, UPDATE. Hàm trả về một bit chỉ định cột bị chỉnh sửa trong quá trình INSERT, UPDATE trên bảng cơ sở.
+ *bitwise_operator:* Được dùng để so sánh với bit trả về của hàm COLUMNS_UPDATED()

+ *updated_bitmask:* Được sử dụng để kiểm cột nào thực sự được Update trong câu lệnh Insert hoặc Update.
+ *sql_statement:* Các câu lệnh T-SQL
+ *... n:* Chỉ định lặp lại các câu lệnh T-SQL.

- Tạo DDL Trigger:

```
CREATE TRIGGER trigger_name
ON { ALL SERVER | DATABASE }
[ WITH ENCRYPTION ]
{ FOR |AFTER } {event_type } [,...n ]
AS { sql_statement  [ ; ] [ ...n ]}
```

Trong đó:

+ DATABASE: Chỉ định phạm vi của DDL trigger là database hiện thời. Trigger sẽ kích nổ khi sự kiện event_type xảy ra trên database hiện thời.
+ ALL SERVER: Chỉ định phạm vi của DDL trigger là server hiện thời. Trigger sẽ kích nổ khi sự kiện event_type xảy ra trên server hiện thời.
+ event_type: Là tên của sự kiện mà là nguyên nhân kích nổ DDL trigger. Các sự kiện này có thể là: CREATE_FUNCTION, CREATE_INDEX, GRANT_DATABASE, CREATE_TABLE, ALTER_VIEW, ALTER_TABLE, DROP_TABLE, DROP_VIEW, .v.v...

Ví dụ 4.44. Tạo một AFTER INSERT Trigger. Trong ví dụ này ta định nghĩa AFTER INSERT Trigger trên bảng TriggerTableChild. Trigger này có nhiệm vụ kiểm tra xem có sự tương ứng với các dòng của bảng TriggerTableParent hay không. Nếu không có sự tương ứng nó sẽ thực hiện roll back (cuốn lại) lại giao dịch đó và dòng thông báo lỗi hiện lên. Nếu có sự tương ứng thì giao dịch được thực hiện một cách bình thường.

```
CREATE TABLE TriggerTableParent
(
    TriggerID    INT,
```

```
    TriggerText     VARCHAR(32)
)
GO

INSERT INTO TriggerTableParent VALUES (1, 'Trigger Text 1')
INSERT INTO TriggerTableParent VALUES (2, 'Trigger Text 2')
INSERT INTO TriggerTableParent VALUES (3, 'Trigger Text 3')
INSERT INTO TriggerTableParent VALUES (4, 'Trigger Text 4')
INSERT INTO TriggerTableParent VALUES (5, 'Trigger Text 5')
GO

CREATE TABLE TriggerTableChild
(
    TriggerID     INT,
    TriggerSubText     VARCHAR(32)
)
GO

CREATE TRIGGER trTriggerTableChildInsert
ON TriggerTableChild
FOR INSERT
AS
IF (SELECT COUNT(*)FROM TriggerTableParent TTP
    INNER JOIN INSERTED I ON (TTP.TriggerID= I.TriggerID))= 0
BEGIN
    ROLLBACK TRANSACTION
    RAISERROR ('No corresponding record was found in the
           TriggerTableParent table for this insert.', 11, 1)
END
ELSE
    Print 'This vas inserted'
GO
SET NOCOUNT ON
INSERT INTO TriggerTableChild VALUES(1, 'Sub Trigger Text 1')
INSERT INTO TriggerTableChild VALUES(2, 'Sub Trigger Text 2')
INSERT INTO TriggerTableChild VALUES(3, 'Sub Trigger Text 3')
INSERT INTO TriggerTableChild VALUES(6, 'Sub Trigger Text 6')
GO
```

Ví dụ 4.45. Dùng bảng INSERTED và DELETED

```
CREATE TRIGGER trTriggerTableParentUpdate1
ON TriggerTableParent1
AFTER UPDATE
AS
SET NOCOUNT ON

PRINT     'Contents of the INSERTED Table:'
SELECT     *
FROM     INSERTED
PRINT     'Contents of the DELETED Table:'
SELECT     *
FROM     DELETED
PRINT     'Contents of the TriggerTableParent Table:'
SELECT     TTP.* FROM     TriggerTableParent1 TTP
    INNER JOIN INSERTED I ON
    (TTP.TriggerID = I.TriggerID)

ROLLBACK TRANSACTION
GO
UPDATE TriggerTableParent1
SET TriggerText = 'Changed Trigger Text 1'
WHERE TriggerID = 1
```

Khi sử dụng lệnh UPDATE, kết quả cho thấy bảng INSERTED chứa các giá trị mới; bảng DELETED chứa các giá trị cũ và bảng TriggerTableParent1 chứa các giá trị mới.

Ví dụ 4.46. Sử dụng cấu trúc IF UPDATED trong UPDATE Trigger

```
CREATE TRIGGER trTriggerTableChildUpdate
ON TriggerTableChild1
AFTER UPDATE
AS
IF UPDATE(TriggerID)
BEGIN
    IF (SELECT COUNT(*)FROM     TriggerTableParent1 TTP
            INNER JOIN INSERTED I ON
            (TTP.TriggerID = I.TriggerID)) = 0
```

```
        BEGIN
            RAISERROR ('No parent record exists for this
                    modification. Transaction cancelled.', 11,
                1)
            ROLLBACK TRANSACTION
            RETURN
        END
    END
    GO

    UPDATE      TriggerTableChild1
    SET     TriggerID = 7
    WHERE       TriggerID = 1
    GO
```

Ví dụ 4.47. Tạo INSTEAD OF Trigger. Bởi vì INSTEAD OF Trigger được kích nổ trước khi bất cứ dữ liệu nào được sửa đổi trong CSDL, do đó vai trò của hai bảng INSERTED và DELETED bị giảm nhẹ hơn. Ta xét ví dụ sau tương tự như ví dụ 4.45.

```
CREATE TABLE TriggerTableParent2
(
    TriggerID     INT,
    TriggerText     VARCHAR(32)
)
GO
INSERT INTO TriggerTableParent2 VALUES(1,'Trigger Text 1')
INSERT INTO TriggerTableParent2 VALUES(2,'Trigger Text 2')
INSERT INTO TriggerTableParent2 VALUES(3,'Trigger Text 3')
INSERT INTO TriggerTableParent2 VALUES(4,'Trigger Text 4')
INSERT INTO TriggerTableParent2 VALUES(5,'Trigger Text 5')
GO
CREATE TABLE TriggerTableChild2
(
    TriggerID     INT,
    TriggerSubText     VARCHAR(32)
)
GO
CREATE TRIGGER trTriggerTableParent2InsteadOfUpdate
ON TriggerTableParent2
```

```
INSTEAD OF UPDATE
AS
SET NOCOUNT ON

PRINT     'Contents of the INSERTED Table:'
SELECT     *
FROM     INSERTED
PRINT     'Contents of the DELETED Table:'
SELECT     *
FROM     DELETED
PRINT     'Contents of the TriggerTableParent Table:'
SELECT     TTP.*
FROM     TriggerTableParent2 TTP
    INNER JOIN INSERTED I ON
    (TTP.TriggerID = I.TriggerID)

ROLLBACK TRANSACTION
GO

UPDATE     TriggerTableParent2
SET     TriggerText = 'Changed Trigger Text 1'
WHERE     TriggerID = 1
GO
```

Khi thực hiện lệnh Update, ta thấy bảng cơ sở chưa được chèn dữ liệu do trigger đã được thực thi trước khi có sự sửa đổi dữ liệu, đây chính là đặc điểm của INSTEAD OF Trigger. Cụ thể khi sử dụng lệnh UPDATE, kết quả cho thấy bảng INSERTED chứa các giá trị mới; bảng DELETED chứa các giá trị cũ và bảng TriggerTableParent2 chứa các giá trị cũ.

Ví dụ 4.48. View của các bảng và INSTEAD OF Trigger.

Một trong những đặc điểm nổi bật chính của INSTEAD OF Trigger là cho phép người sử dụng thực hiện các câu lệnh thay đổi dữ liệu trên view của nhiều bảng. Trong ví dụ này, ta xây dựng một INSTEAD OF Trigger thực hiện chức năng này.

```
SET NOCOUNT ON
GO
CREATE TABLE ViewTable1
```

```
(
    KeyColumn    INT,
    Table1Column    VARCHAR(32)
)
GO
INSERT INTO ViewTable1 VALUES (1, 'ViewTable1 Value 1')
INSERT INTO ViewTable1 VALUES (2, 'ViewTable1 Value 2')
INSERT INTO ViewTable1 VALUES (3, 'ViewTable1 Value 3')
INSERT INTO ViewTable1 VALUES (4, 'ViewTable1 Value 4')
INSERT INTO ViewTable1 VALUES (5, 'ViewTable1 Value 5')
INSERT INTO ViewTable1 VALUES (6, 'ViewTable1 Value 6')
INSERT INTO ViewTable1 VALUES (7, 'ViewTable1 Value 7')
INSERT INTO ViewTable1 VALUES (8, 'ViewTable1 Value 8')
INSERT INTO ViewTable1 VALUES (9, 'ViewTable1 Value 9')
INSERT INTO ViewTable1 VALUES (10, 'ViewTable1 Value 10')
GO

CREATE TABLE ViewTable2
(
    KeyColumn    INT,
    Table2Column    VARCHAR(32)
)
GO

INSERT INTO ViewTable2 VALUES (1, 'ViewTable2 Value 1')
INSERT INTO ViewTable2 VALUES (2, 'ViewTable2 Value 2')
INSERT INTO ViewTable2 VALUES (3, 'ViewTable2 Value 3')
INSERT INTO ViewTable2 VALUES (4, 'ViewTable2 Value 4')
INSERT INTO ViewTable2 VALUES (5, 'ViewTable2 Value 5')
INSERT INTO ViewTable2 VALUES (6, 'ViewTable2 Value 6')
INSERT INTO ViewTable2 VALUES (7, 'ViewTable2 Value 7')
INSERT INTO ViewTable2 VALUES (8, 'ViewTable2 Value 8')
INSERT INTO ViewTable2 VALUES (9, 'ViewTable2 Value 9')
INSERT INTO ViewTable2 VALUES (10, 'ViewTable2 Value 10')
GO

CREATE VIEW TestView1
AS
SELECT VT1.KeyColumn, VT1.Table1Column, VT2.Table2Column
```

```
FROM ViewTable1 VT1 INNER JOIN ViewTable2 VT2 ON
                              (VT1.KeyColumn = VT2.KeyColumn)
GO
INSERT INTO TestView1 VALUES (11, 'ViewTable1 Value 11',
'ViewTable2 Value 11')
GO
CREATE TRIGGER trTestView1InsteadOfInsert
ON TestView1
INSTEAD OF INSERT
AS
DECLARE    @intKeyColumn         INT
DECLARE @vchTable1Column    VARCHAR(32)
DECLARE @vchTable2Column    VARCHAR(32)
DECLARE @intError        INT

SET NOCOUNT ON
SELECT @intKeyColumn=KeyColumn, @vchTable1Column =
Table1Column,    @vchTable2Column = Table2Column
FROM    INSERTED
BEGIN TRANSACTION
 INSERT INTO ViewTable1
VALUES(@intKeyColumn,@vchTable1Column)
    SELECT @intError = @@ROWCOUNT
    INSERT INTO ViewTable2 VALUES(@intKeyColumn,
@vchTable2Column)
    SELECT @intError = @intError + @@ROWCOUNT
    IF ((@intError < 2) OR (@intError % 2) <> 0)
    BEGIN
        RAISERROR('An error occurred during the multitable
insert.', 1, 11)
        ROLLBACK TRANSACTION
        RETURN
    END
COMMIT TRANSACTION
GO
INSERT INTO TestView1 VALUES (11, 'ViewTable1 Value
11','ViewTable2 Value 11')
GO
```

**** Xem những trigger nào đang tồn tại trên một bảng hoặc một view:*** Dùng thủ tục sp_helptrigger.

Ví dụ 4.50. Xem các trigger trên bảng HOSOSV

```
Use QLDiemSV
Go
Exec sp_helptrigger HOSOSV
go
```

**** Thay đổi nội dung trigger:*** Để thay đổi trigger ta dùng câu lệnh ALTER TRIGGER theo cú pháp sau:

- Sửa DML Trigger:

```
ALTER TRIGGER trigger_name
ON {table | view }
[WITH ENCRYPTION]
{
{{FOR | AFTER | INSTEAD OF} { [DELETE] [,]
[INSERT] [,] [UPDATE] }
    [NOT FOR REPLICATION]
AS
    [ { IF UPDATE ( column )
            [ { AND | OR } UPDATE ( column ) ]
                [ ...n ]
        | IF ( COLUMNS_UPDATED ( ) {
bitwise_operator } updated_bitmask )
                { comparison_operator }
column_bitmask [ ...n ]
        } ]
        sql_statement [ ...n ]
}
```

- Sửa DDL Trigger:

```
ALTER TRIGGER trigger_name
ON { ALL SERVER | DATABASE }
[ WITH ENCRYPTION ]
{ FOR |AFTER } {event_type } [,...n ]
AS { sql_statement  [ ; ] [ ...n ]}
```

Ví dụ 4.51. Thay đổi nội dung trigger trTestView1InsteadOfInsert

```
ALTER TRIGGER trTestView1InsteadOfInsert
ON TestView1
INSTEAD OF INSERT
AS
DECLARE    @intKeyColumn         INT
DECLARE @vchTable1Column    VARCHAR(32)
DECLARE @vchTable2Column    VARCHAR(32)
DECLARE @intError        INT

SET NOCOUNT ON

SELECT @intKeyColumn=KeyColumn, @vchTable1Column =
Table1Column,    @vchTable2Column = Table2Column
FROM    INSERTED

BEGIN TRANSACTION
 INSERT INTO ViewTable1
VALUES(@intKeyColumn,@vchTable1Column)
    SELECT @intError = @@ROWCOUNT
    INSERT INTO ViewTable2 VALUES(@intKeyColumn,
@vchTable2Column)
    SELECT @intError = @intError + @@ROWCOUNT
    IF ((@intError < 2) OR (@intError % 2) <> 0)
    BEGIN
        ROLLBACK TRANSACTION
        RETURN
    END
COMMIT TRANSACTION
GO
```

**** Xóa một trigger***: Dùng câu lệnh DROP TRIGGER.

- Xóa DML Trigger:

```
DROP TRIGGER schema_name.trigger_name [ ,...n ]
```

- Xóa DDL Trigger:

```
DROP TRIGGER trigger_name [ ,...n ]
    ON { DATABASE | ALL SERVER }
```

Ví dụ 4.52. Xóa trigger `trTestView1InsteadOfInsert`

```
DROP TRIGGER trTestView1InsteadOfInsert
go
```

**** Vô hiệu hóa hoặc làm cho có hiệu lực một trigger ta dùng câu lệnh:***

```
DISABLE | ENABLE TRIGGER{[ schema . ]
trigger_name [ ,...n ] | ALL }
ON { object_name | DATABASE | ALL SERVER } [ ; ]
```

Ví dụ 4.53. Vô hiệu hóa trigger `trTriggerTableParent2InsteadOfUpdate` Trên bảng `TriggerTableParent2.`

```
ALTER TABLE TriggerTableParent2
DISABLE TRIGGER trTriggerTableParent2InsteadOfUpdate
GO.
```

Ví dụ 4.54. Vô hiệu hóa trigger `[trTriggerTableChildInsert]` trên bảng `TriggerTableChild.`

```
DISABLE TRIGGER [trTriggerTableChildInsert] ON
[dbo].[TriggerTableChild]
```

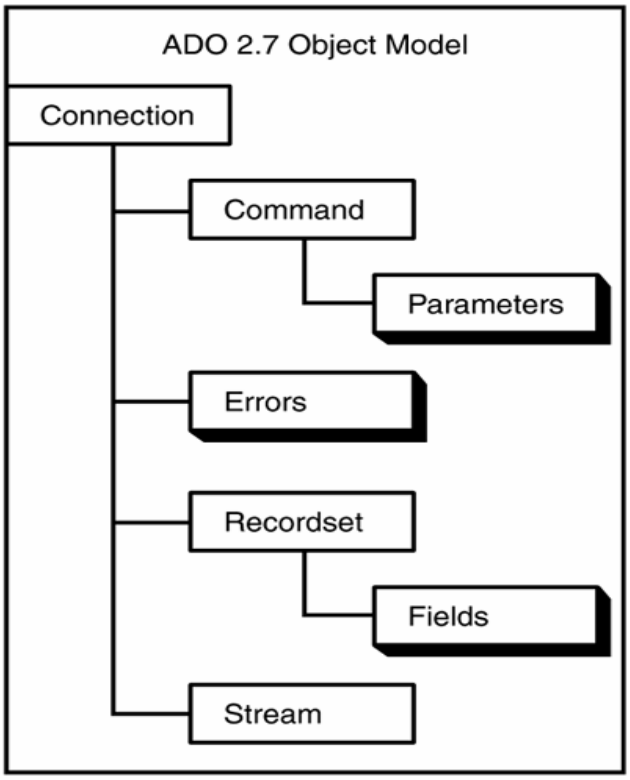
Ví dụ 4.55. Làm cho trigger `trTriggerTableParent2InsteadOfUpdate` trên bảng `TriggerTableParent2` có hiệu lực.

```
ALTER TABLE TriggerTableParent2
ENABLE TRIGGER trTriggerTableParent2InsteadOfUpdate
GO.
```

để nó hoạt động thì ta cần ADODB. Các ứng dụng thường làm việc với các bản ghi dữ liệu.

ActiveX Data Objects 2.7 (ADODB) Object Model

Mô hình đối tượng *ActiveX Data Objects* - *ADO* (Xem hình 5.1) bao gồm:

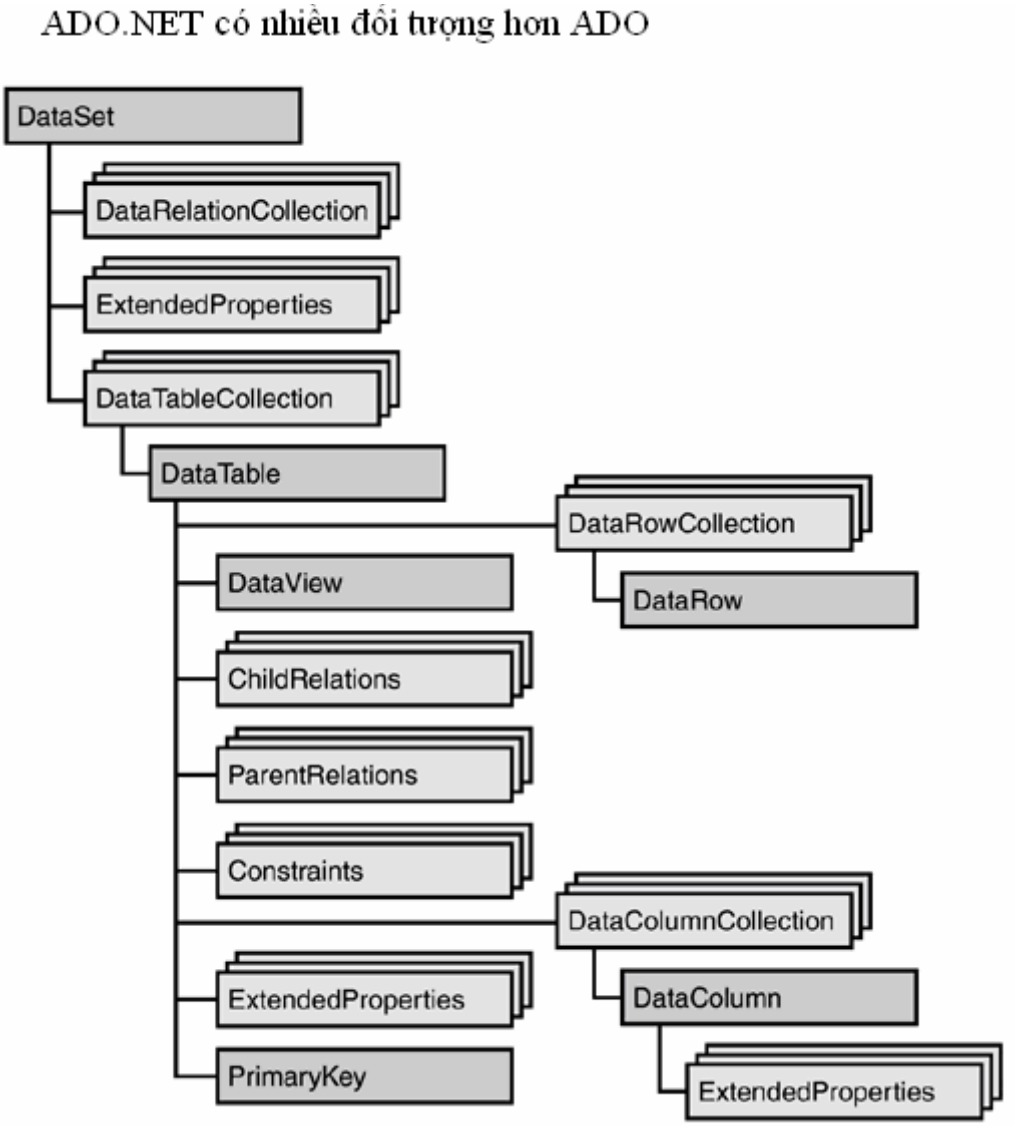


Hình 5.1. **Mô hình ADO**

+ Đối tượng *Connection*. Tương tự như đối tượng Database trong DAO, đó là nơi mà tất cả các thao tác làm việc của ta với ADO được bắt đầu. Tất cả các objects và collections đều được đề cập sau đối tượng *Connection*.

+ Đối tượng Errors collection/Error. Giống như *DAO errors collection* và *error object*, nó cho phép phát triển để quản lý lỗi.

+ Đối tượng *Command*. Cho phép chạy truy vấn trên một database và trả lại các bản ghi trên đối tượng Recordset, thao tác với cấu trúc CSDL, và thực hiện một thao tác dữ liệu. Một tập hợp các tham số sẽ được sử dụng với đối tượng Command.



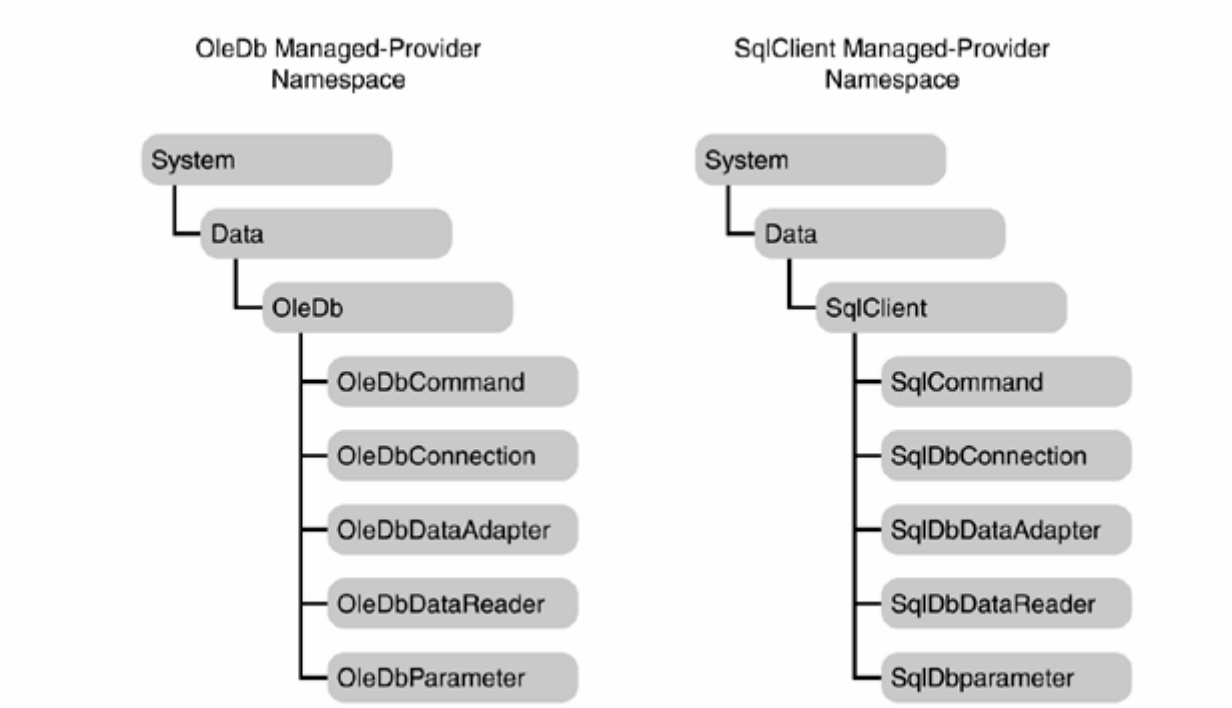
***Hình 5.2.* Mô hình ADO.NET**

Xét bảng Bảng 5.1 để tìm hiểu các mô tả ngắn ngọn về các đối tượng hay dùng.

Bảng5.1. ADO.NET Data Objects hay được dùng để thao tác dữ liệu	
Object	***Mục đích***

`DataSet`	Đối tượng này được sử dụng cùng với các điều khiển dữ liệu khác, lưu trữ các kết quả được trả về bởi các đối tượng *commands* và *data adapters*. Không như recordset của ADO và DAO, *Data set* thực sự mang lại một view phân cấp của dữ liệu. Bằng việc sử dụng các thuộc tính và các collections trong đối tượng `DataSet`, ta có thể nhận được toàn bộ các quan hệ (relations), các tables riêng biệt, rows, và columns.
`DataTable`	Một trong đối tượng của data set, là đối tượng `DataTable` cho phép ta có thể thao tác dữ liệu trong một table riêng biệt. Data table tương tự như đối tượng recordset trong ADO.
`DataView`	Bằng việc sử dụng đối tượng này, ta có thể lọc và sắp xếp dữ liệu, duy trì các views khác nhau của dữ liệu. Mỗi data table có một default view, nó là nơi khởi động data view mà có thể được sửa dổi và lưu trữ trong data view khác.
`DataRow`	Đối tượng này cho phép ta thao tác các rows của dữ liệu trong data tables. Nó có thể được xét như một cache của dữ liệu mà ta có thể thực hiện các thao tác adding, deleting, và modifying records. Ta có thể quay trở lại recordset, nơi mà ta sẽ chạy các câu lệnh SQL để update dữ liệu về server.
`DataColumn`	Như tên đề nghị, ta có thể nhận các thông tin tại các cột bằng việc sử dụng đối tượng `DataColumn`. Ta có thể lấy thông tin lược đồ như là dữ liệu bằng việc sử dụng đối tượng này. Ví dụ: nếu ta muốn tạo một list box của tên các trường, ta có thể lặp thông qua tập DataColumn của các dòng dữ liệu và truy lục tất cả các tên trường.
`PrimaryKey`	Đối tượng này cho phép ta chỉ định primary key cho table dữ liệu. Cách khác, khi ta sử dụng phương thức Find của data table.

.NET cũng cung cấp các classes, được gọi *data providers*, sử dụng tên miền (namespaces) để làm việc với các đối tượng ADO.NET và cung cấp các truy xuất với dữ liệu. Lõi của các lớp ADO.NET tồn tại trong *namespace System.Data*. Namespace này chứa một vài namespaces con. Phần quan trọng nhất của chúng là các tên miền *System.Data.SqlClient* và System.Data.OleDb. Chúng cung cấp các classes sử dụng cho việc truy cập *SQL Server databases* và *OLE (Object Linking and Embedding) DB- compliant databases* (phục hồi dữ liệu). Các lớp của OleDb và SqlClient cho trong hình 5.3. và Bảng 5.2. mô tả vắn tắt các đối tượng thường được sử dụng để thao tác với dữ liệu.



***Hình 5.3.* OleDb và SqlClient classes**

Bảng 5.2. .NET Data Provider Classes thường được sử dụng để thao tác dữ liệu

Object	***Mục đích***
`Command`	Tương tự như đối tượng ADO `Command`, cho phép ta thực hiện các stored procedures trong code. Khác với phiên bản ADO, ta có thể tạo đối tượng `DataReader` bằng việc sử dụng phương thức `ExecuteReader`.

`Connection`	Đây là đối tượng mở kết nối tới server và database mà ta muốn làm việc. Khác với đối tượng ADO `Connection`, cách thức kết nối phụ thuộc vào đối tượng mà ta muốn làm việc, như là đối tượng `DataReader` hay `DataSet`.
`DataAdapter`	Đối tượng `DataAdapter` cho phép ta tạo các câu lệnh SQL và điền dữ liệu vào *datasets*. Nó cũng cho phép tạo các *action queries* cần thiết, như là `Insert`, `Update`, và `Delete`.
`DataReader`	Đối tượng này tạo một *read-only*, *forward-only stream* của dữ liệu mà cho phép chúng đặt trên các điều khiển, như `ListBox` và `ComboBox`.
`Parameter`	Đây là đối tượng cho phép chỉ định các tham số mà các đối tượng `DataAdapter` có thể chỉ định và sử dụng

Ngoài ra, hai tên miền con khác cũng tồn tại trong tên miền System.Data, đó là: *System.Data.OracleClient* và *System.Data.Odbc.* Trong đó*, namespace System.Data.OracleClient* được sử dụng dành cho các CSDL Oracle. Các lớp *SqlClient* cung cấp các kết quả tốt nhất khi làm việc với các SQL Server databases; *OracleClient* cung cấp thực thi tối ưu khi truy cập vào Oracle databases. Namespace System.Data.Odbc cung cấp truy cập tới ODBC (Open Database Connectivity) data sources cũ mà không được support bởi công nghệ OleDb.

Như vậy, các tên miền *System.Data.SqlClient, System.Data.OleDb, System.Data.OracleClient*, và *System.Data.Odbc* được biết như là *data providers* trong ADO.NET.

Trong môn học, ta xây dựng ứng dụng truy xuất SQL Server databases nên sử dụng namespace SqlClient. Tuy nhiên trong ADO.NET, các data providers khác nhau làm việc theo một cách thức tương tự nhau. Vì vậy các kỹ xảo ta sử dụng ở đây có thể được dễ dàng chuyển giao cho các lớp OleDb classes. Với ADO.NET, ta sử dụng *data provider* này là phù hợp nhất cho

Nếu ta không chỉ định chuỗi kết nối trong khi xây dựng, ta có thể thiết lập nó bằng việc sử dụng thuộc tính *SqlConnection.ConnectionString*.

Ta xem xét các chuỗi *connection string*s làm việc như thế nào?

**** Các tham số của chuỗi kết nối (Connection String Parameters)***

Phương thức để chuỗi connection string được xây dựng sẽ phụ thộc vào *data provider* gì ta đang sử dụng. Khi truy xuất SQL Server, ta thường cung cấp Server và Database, được chỉ dẫn theo bảng sau:

Tham số	***Mô tả***
Server	Tên của SQL Server mà ta muốn truy xuất. Nó thường là tên của máy tính đang chạy SQL Server. Ta có thể sử dụng *(local)* hoặc *localhost* nếu SQL Server trên cùng một máy đang chạy ứng dụng. Nếu bạn sử dụng các thể hiện tên của SQL Server, thì tham số nên chứa tên của máy tính theo sau là đấu gạch chéo ngược và tiếp đó là thể hiện tên của SQL Server.
Database	Là tên của database mà ta muốn connect tới.

Ta cũng cần một vài thông tin về authentication và thực hiện theo hai cách: bằng cách cung cấp *username* và *password* trong chuỗi kết nối (connection strings) hoặc bằng cách kết nối tới *SQL Server* sử dụng *NT account* mà ứng dụng đang chạy trong đó.

Nếu ta muốn connect tới server bằng việc chỉ định *username* và *password*, ta cần tính đến các parameters đó trong chuỗi kết nối connection, được cho trong bảng sau:

Tham số	***Mô tả***
User ID	Username sử dụng để connect tới database. account với user ID này phải tồn tại trong SQL Server và được phép truy cập đến database chỉ định.

Password	Là password của user đã chỉ định.

Tuy nhiên, SQL Server có thể được thiết lập sử dụng Windows NT account của user người đang chạy chương trình để mở connection. Trong trường hợp này, ta không cần chỉ định *username* và *password*. Ta chỉ cần phải chỉ định nó khi ta đang sử dụng *integrated security (security kết hợp* - Phương thức này được gọi là security kết hợp bởi vì SQL Server đang được kết hợp với Windows NT's security system và cung cấp connection an toàn nhất bởi vì các tham số User ID và Password parameters phải được chỉ định trong code). Ta sử dụng tham số Integrated Security này, và thiết lập là True khi ta muốn connect SQL Server đang sử dụng user's NT account hiện tại. Tất nhiên, với các làm việc này, user của ứng dụng phải được cho phép sử dụng SQL Server database. Giả dụ như là đang sử dụng SQL Server Enterprise Manager.

Để hiểu chức năng của các tham số này trong chuỗi kết nối ta khởi tạo một đối tượng connection, xét đoạn code sau. Nó sử dụng lớp *SqlConnection* để khởi tạo một đối connection mà được sử dụng chỉ định user ID và password trong chuỗi kết nối:

```
Dim objConnection As New SqlConnection _
    ("server=THUHUONG;database=QLDiemSV;" &_
    "user id=sa;password=pass2008")
```

Chuỗi kết nối này sẽ connects tới SQL Server database. Tham số Server chỉ định là database nằm trên máy THUHUONG. Tham số Database chỉ định database mà ta muốn truy cập, trong trường hợp này là database QLDiemSV. Cuối cùng, các tham số User ID và Password chỉ định User ID và password của user định nghĩa trên database. Ta thấy rằng, mỗi một tham số được gán một giá trị qua dấu **=**, và mỗi cặp *parameter* - *value* được phân cách nhau nhau bởi dấu chấm phảy. Hoặc

```
Dim objConnection As SqlConnection = New _
SqlConnection("Server=localhost;Database=pubs;" & _
"User ID=sa;Password=vbdotnet;")
```

Khi sử dụng các *data adapters* (bộ điều hợp dữ liệu) và *datasets*, không cần gọi nhiều các đối tượng của chúng. Chúng có thể chủ yếu được sử dụng cho việc thực thi một câu lệnh select, delete, insert, hoặc update, mà đó là những gì ta đề cập trong chương này. Ta cũng có thể sử dụng các đối tượng command với *data reader*. Một *data reader* là một lựa chọn khác với DataSet mà nó sử dụng một vài tài nguyên hệ thống nhưng kém mềm dẻo.

**** Thuộc tính Connection***

Một vài thuộc tính phải được thiết lập trên đối tượng SqlCommand trước khi ta có thể thực hiện một câu truy vấn. Thuộc tính đầu tiên trong các thuộc tính này đó là thuộc tính Connection. Thuộc tính này thiết lập tới đối tượng SqlConnection như trong đoạn code sau:

```
objCommand.Connection = objConnection
```

Chú ý: Để câu lệnh được thực hiện thành công, thì connection phải được mở ở thời điểm thực hiện.

**** Thuộc tính CommandText***

Thuộc tính tiếp theo phải được thiết lập đó là thuộc tính CommandText. Thuộc tính này chỉ định chuỗi SQL hoặc *stored procedure* sẽ được thực hiện. Hầu hết các databases quy định đặt các các giá trị của chuỗi trong cặp dấu *" các giá trị của chuỗi "*, ví dụ:

```
Dim objConnection As New SqlConnection _
                ("server=THUHUONG;database=QLDiemSV;" &_
    "user id=sa;password=pass2008")
Dim objCommand As SqlCommand = New SqlCommand()
' Open the connection, execute the command
objConnection.Open()
' Set the SqlCommand object properties...
objCommand.Connection = objConnection
objCommand.CommandText = "INSERT INTO HOSOSV " & _
            "(MaSV, HoDem, TenSV, NgaySinh, MaLop) " & _
"VALUES('SV001','Nguyen Thi','Hoa','12/8/1988','TH05')"
```

Câu lệnh INSERT chỉ đơn giản nghĩa là "chèn một hàng mới vào bảng HOSOSV. Trong đó cột MaSV nhận giá trị 'SV001', HoDem nhận giá trị 'Nguyen Thi', TenSV nhận giá trị 'Hoa', NgaySinh nhận giá trị '12/8/1988' và cột MaLop nhận giá trị ' TH05'."

Với cách cơ bản này các câu lệnh INSERT làm việc trong SQL. Ta có, theo sau INSERT INTO là tên của bảng. Tiếp là một danh sách tên các cột trong cặp dấu ngoặc đơn. Tiếp theo nữa là từ khóa VALUES là danh sách các giá trị được chèn vào cột theo đúng thứ tự.

Trong ví dụ trên, với các giả thiết là ta biết các giá trị để chèn trong khi thực hiện chương trình, thường là không có thực trên hầu hết các ứng dụng. Cũng may, ta cũng có thể tạo các câu lệnh với các tham số và thiết lập giá trị cho các tham số đó một cách độc lập. Ta sẽ tìm hiểu cách sử dụng các tham số đó như thế nào trong phần Parameters Collection.

**** Parameters Collection***

Placeholders, ký hiệu @, là tiền tố các biến trong các câu lệnh SQL; Chúng được điền vào trước các tham số. Như vậy, nếu ta muốn cập nhật bảng HOSOSV được mô tả trong ví dụ trước nhưng không biết các giá trị tại thời điểm thiết kế, ta chỉ cần viết:

```
Dim objConnection As New SqlConnection _
                ("server=THUHUONG;database=QLDiemSV;" &_
    "user id=sa;password=pass2008")

Dim objCommand As SqlCommand = New SqlCommand()
' Open the connection, execute the command
objConnection.Open()
' Set the SqlCommand object properties...
objCommand.Connection = objConnection
objCommand.CommandText = "INSERT INTO HOSOSV " & _
            "(MaSV, HoDem, TenSV, NgaySinh, MaLop) " & _
            "VALUES(@MaSV,@Hodem,@TenSV,@NgaySinh,@MaLop)"
```

Ở đây, thay vì cung cấp các giá trị cụ thể, ta cung cấp các placeholders. Placeholders, như đã đề cập, luôn xuất hiện với ký hiệu @. Chúng không cần được đặt trước tên các trường trong các bảng của database mà chúng thể hiện, nhưng thường đơn giản hơn nếu ta sử dụng tên tham số trùng với tên trường dữ liệu, vì nó thường tốt hơn cho chính các văn bản code của ta.

Tiếp theo, ta cần tạo các tham số mà sẽ được sử dụng để truyền giá trị cho các placeholders khi câu lệnh SQL được thực hiện. Ta cần phải create và add các tham số vào *Parameters collection* của đối tượng *SqlCommand*.

Thuật ngữ *parameters* chỉ các tham số cần thiết cho câu lệnh SQL hoặc *stored procedure*, không chỉ các tham số được quy định chuyển để chuyển sang các phương thức của Visual Basic 2005.

Ta có thể truy cập vào tập Parameters collection của đối tượng SqlCommand bằng việc chỉ định thuộc tính Parameters. Sau khi truy cập vào tập Parameters collection, ta có thể sử dụng các thuộc tính và các phương thức của nó để tạo một hoặc nhiều hơn các tham số trong collection. Cách tốt nhất để add một tham số tới câu lệnh được mô tả trong ví dụ sau:

```
Dim objConnection As New SqlConnection _
                ("server=THUHUONG;database=QLDiemSV;" &_
    "user id=sa;password=pass2008")

Dim objCommand As SqlCommand = New SqlCommand()
' Open the connection, execute the command
objConnection.Open()
' Set the SqlCommand object properties...
objCommand.Connection = objConnection
objCommand.CommandText = "INSERT INTO HOSOSV " & _
            "(MaSV, HoDem, TenSV, NgaySinh, MaLop) " & _
            "VALUES(@MaSV,@Hodem,@TenSV,@NgaySinh,@MaLop)"
' Add parameters for the placeholders in the SQL in the
objCommand.Parameters.AddWithValue("@MaSV", txtMaSV.Text)
objCommand.Parameters.AddWithValue("@Hodem", txtHodem.Text)
objCommand.Parameters.AddWithValue("@TenSV", txtTenSV.Text)
objCommand.Parameters.AddWithValue("@NgaySinh",_
                    txtNgaySinh.Text).DbType = DbType.Date
objCommand.Parameters.AddWithValue("@MaLop", cboMaLop.Text)
```

Phương thức *AddWithValue* thừa nhận tên của tham số và đối tượng ta muốn add. Trong trường hợp này, ta sử dụng thuộc tính Text của đối tượng Text box trên cùng một Form.

**** Phương thức ExecuteNonQuery***

Cuối cùng, ta có thể thực hiện các câu lệnh. Để làm điều này, connection cần phải được mở. Ta có thể gọi phương thức ExecuteNonQuery của đối tượng SqlCommand. Phương thức này thực hiện câu lệnh SQL và là nguyên

nhân dữ liệu được chèn vào database. Để hoàn thành đoạn code, ta phải mở connection, thực hiện query, và đóng connection trở lại:

```
Dim objConnection As New SqlConnection _
                ("server=THUHUONG;database=QLDiemSV;" &_
    "user id=sa;password=pass2008")

Dim objCommand As SqlCommand = New SqlCommand()
' Set the SqlCommand object properties...
objCommand.Connection = objConnection
objCommand.CommandText = "INSERT INTO HOSOSV " & _
            "(MaSV, HoDem, TenSV, NgaySinh, MaLop) " & _
            "VALUES(@MaSV,@Hodem,@TenSV,@NgaySinh,@MaLop)"
' Add parameters for the placeholders in the SQL in the

objCommand.Parameters.AddWithValue("@MaSV", txtMaSV.Text)
objCommand.Parameters.AddWithValue("@Hodem", txtHodem.Text)
objCommand.Parameters.AddWithValue("@TenSV", txtTenSV.Text)
objCommand.Parameters.AddWithValue("@NgaySinh",_
                    txtNgaySinh.Text).DbType = DbType.Date
objCommand.Parameters.AddWithValue("@MaLop", cboMaLop.Text)
' Open the connection, execute the command
objConnection.Open()
' Execute the SqlCommand object to insert the new data...
Try
    objCommand.ExecuteNonQuery()
Catch SqlExceptionErr As SqlException
    MessageBox.Show(SqlExceptionErr.Message)
End Try
' Close the connection...
objConnection.Close()
```

**** Phương thức ExecuteReader***

Trái với phương thức ExecuteNonQuery, phương thức ExecuteReader của đối tượng SqlCommand sẽ thực hiện truy vấn trả về kết quả trên đối tượng DataReader. Và dữ liệu trên DataReader thường được đổ sang các điều khiển ListBox, ComboBox trên form ứng dụng.

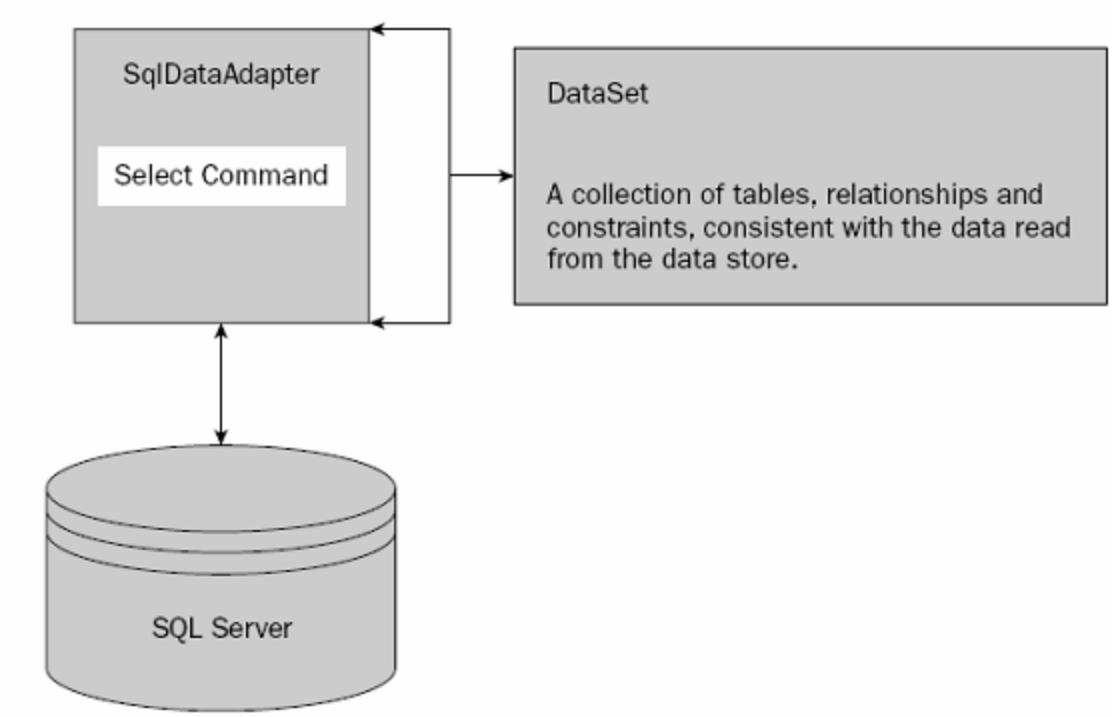
```
    ' Declare local variables and objects...
    Dim objCommand As SqlCommand = New SqlCommand()
    Dim objDataReader As SqlDataReader

    ' Open the connection, execute the command
    objConnection.Open()
    ' Set the SqlCommand object properties...
    objCommand.Connection = objConnection
```

SelectCommand, và các tools được gọi *command builders* mà ta có thể tạo một cách tự động các câu lệnh khác dựa trên chúng.

Xét thuộc tính SelectCommand và xét xem ta có thể tạo các câu lệnh cho việc updating, deleting, và inserting records như thế nào.

**** Thuộc tính SelectCommand***

Thuộc tính SelectCommand của lớp SqlDataAdapter được dùng để đẩy dữ liệu vào từ SQL Server database DataSet, như hình vẽ Hình 5.4.



Hình 5.4. Thuộc tính Select Command

Khi ta muốn đọc dữ liệu từ kho dữ liệu, đầu tiên ta phải thiết lập thuộc tính SelectCommand của lớp SqlDataAdapter. Thuộc tính này là đối tượng SqlCommand và nó được dùng để chỉ định dữ liệu được select và select dữ liệu như thế nào. Vì vậy, thuộc tính SelectCommand cũng có các thuộc tính của nó, và ta cần thiết lập chúng như là thiết lập các thuộc tính trên các câu lệnh thông thường. Ta đã xem xét các thuộc tính sau của đối tượng SqlCommand:

- ✓ Connection: Thiết lập đối tượng SqlConnection để truy xuất vào data store.

- ✓ CommandText: Thiết lập các câu lệnh SQL hoặc tên stored procedure được sử dụng để selectdữ liệu.

Trong các ví dụ trước của các đối tượng SqlCommand, ta đã sử dụng trực tiếp các câu lệnh SQL. Nếu ta muốn sử dụng các stored procedures, ta cần phải sử dụng thêm thuộc tính CommandType, thuộc tính này thiết lập một giá trị xác định xem thuộc tính CommandText được biên dịch như thế nào.

Trong chương này, ta tập trung vào các câu lệnh SQL, nhưng các stored procedures thường cũng rất hữu dụng, đặc biệt là nếu chúng đã tồn tại trên database. Nếu ta muốn sử dụng chúng, thì thiết lập thuộc tính CommandText với tên của stored procedure (chú ý là phải được đóng trong cặp dấu ngoặc kép "tên của stored procedure hoặc chuỗi SQL " bởi trình trình biên dịch biên dịch chúng như là một string), và thiết lập thuộc tính CommandType thành CommandType.StoredProcedure.

- ***Thiết lập SelectCommand với một chuỗi SQL***

Xét xem làm thế nào, ta có thể thiết lập các thuộc tính trên trong code. Đoạn code sau cho ta thấy các cách thiết lập điển hình khi thực hiện chuỗi SQL string:

```
' Declare a SqlDataAdapter object...
Dim objDataAdapter As New SqlDataAdapter()
' Assign a new SqlCommand to the SelectCommand property
objDataAdapter.SelectCommand = New SqlCommand()
' Set the SelectCommand properties...
objDataAdapter.SelectCommand.Connection = objConnection
objDataAdapter.SelectCommand.CommandText = _
"SELECT MaSV, HoDem, TenSV, FROM HOSOSV ORDER BY TenSV,
Hodem"
```

Đầu tiên trong đoạn code trên khai báo đối tượng SqlDataAdapter. Đối tượng này có thuộc tính SelectCommand được thiết lập bằng SqlCommand; Ta chỉ cần thiết lập các thuộc tính của command. Ta thiết lập các thuộc tính bằng việc thiết lập thuộc tính Connection, để đối tượng connection được hợp lệ thì nó phải được tạo trước đoạn code trên. Tiếp theo, ta thiết lập các thuộc tính của thuộc tính CommandText bằng các câu lệnh SQL SELECT.

- ***Thiết lập SelectCommand với một Stored Procedure***

Trong đoạn code tiếp theo, ta tìm hiểu cách thiết lập các thuộc tính trên khi sử dụng để thực hiện một stored procedure. Một stored procedure là một nhóm các câu lệnh SQL và đã được lưu trữ trong database dưới một tên duy nhất và được thực hiện như một unit. stored procedure trong ví dụ này (usp_select_DSSV) sử dụng cùng câu lệnh SQL ta đã sử dụng trong phần trước:

```
' Declare a SqlDataAdapter object...
Dim objDataAdapter As New SqlDataAdapter()
' Assign a new SqlCommand to the SelectCommand property
objDataAdapter.SelectCommand = New SqlCommand()
' Set the SelectCommand properties...
objDataAdapter.SelectCommand.Connection = objConnection
objDataAdapter.SelectCommand.CommandText = "usp_select_DSSV"
objDataAdapter.SelectCommand.CommandType =
CommandType.StoredProcedure
```

Thuộc tính CommandText bây giờ được chỉ định bằng tên của stored procedure mà ta muốn thực hiện thay vì chuỗi SQL đã được chỉ định trong ví dụ trước. Cũng phải chú ý đến thuộc tính CommandType. Trong ví dụ trước ta không cần phải chuyển đổi thuộc tính này, bởi vì nó được mặc định là CommandType.Text, mà ta cần thiết lập để thực hiện các câu lệnh SQL. Trong ví dụ này, ta phải thiết lập giá trị CommandType.StoredProcedure, để chỉ dẫn rằngthuộc tính CommandText chứa tên của stored procedure để thực hiện.

Sử dụng Command Builders để tạo các Commands khác

SelectCommand là tất cả những gì ta cần thiết để chuyển dữ liệu từ database vào DataSet. Sau khi ta cho phép các users thay đổi đối với DataSet, ta muốn ghi những thay đổi đó trở lại database. Ta có thể làm điều này bằng việc điều chỉnh các đối tượng câu lệnh SQL cho các công việc inserting, deleting, and updating. Một lựa chọn khác, ta có thể sử dụng các stored procedures. Cả hai giải pháp phụ thuộc vào sự am hiểu về SQL. Rất may là, có một cách đơn giản hơn, ta có thể sử dụng *command builders* để tạo các câu lệnh này. Nó chỉ có một dòng trong các dòng sau:

```
' Declare a SqlDataAdapter object...
```

```
Dim objDataAdapter As New SqlDataAdapter()
' Assign a new SqlCommand to the SelectCommand property
objDataAdapter.SelectCommand = New SqlCommand()
' Set the SelectCommand properties...
objDataAdapter.SelectCommand.Connection = objConnection
objDataAdapter.SelectCommand.CommandText = "usp_select_DSSV"
objDataAdapter.SelectCommand.CommandType =
CommandType.StoredProcedure
' automatically create update/delete/insert commands
Dim objCommandBuilder As SqlCommandBuilder = New _
SqlCommandBuilder(objDataAdapter)
```

Như vậy, ta có thể sử dụng SqlDataAdapter để ghi những thay đổi dữ liệu trở lại database. Ta tìm hiểu kỹ hơn về vấn đề này trong phần ví dụ minh họa. Để hiểu, ta xét các phương thức mà lấy dữ liệu từ database đẩy vào DataSet:Phương thức Fill .

**** Phương thức Fill***

Ta dùng phương thức Fill để dữ liệu đẩy dữ liệu mà đối tượng SqlDataAdapter sử dụng SelectCommand của nó lấy từ kho lưu trữ dữ liệu vào đối tượng DataSet. Tuy nhiên, trước khi làm điều đó, ta phải khởi tạo đối tượng DataSet. Để sử dụng đối tượng DataSet trong project của ta, ta phải add một tham chiếu tới System.Xml.

```
' Declare a SqlDataAdapter object...
Dim objDataAdapter As New SqlDataAdapter()
' Assign a new SqlCommand to the SelectCommand property
objDataAdapter.SelectCommand = New SqlCommand()
 ' Set the SelectCommand properties...
objDataAdapter.SelectCommand.Connection = objConnection
objDataAdapter.SelectCommand.CommandText = "usp_select_DSSV"
objDataAdapter.SelectCommand.CommandType =
CommandType.StoredProcedure
Dim objDataSet as DataSet = New DataSet()
```

Ta có hai đối tượng DataSet và SqlDataAdapter, có thể điền dữ liệu vào DataSet. Phương thức Fill có nhiều phiên bản, nhưng ta sẽ chỉ xét thảo luận một phiên bản hay được sử dụng nhiều nhất. Cú pháp của phương thức Fill được cho như sau:

```
SqlDataAdapter.Fill(DataSet, string)
```

Đối số *DataSet* chỉ định một đối tượng DataSet hợp lệ mà dữ liệu được lưu trữ ở trong đó. Đối số *string* gán với tên mà ta muốn bảng có trong DataSet. Chú ý rằng, một DataSet có thể chứa nhiều bảng. Ta có thể sử dụng bất cứ tên nào ta muốn, nhưng thường tốt nhất là sử dụng tên của các bảng nơi mà dữ liệu trong database đến. Nó giúp cho văn bản code của ta và làm cho code dễ bảo trì.

Đoạn code sau gọi phương thức Fill. Chuỗi "authors" được chỉ định là đối số string. Nó là tên ta muốn sử dụng khi đang thao tác trong các phiên bản nhớ của bảng này; nó cũng là tên của bảng trong data source.

```
' Declare a SqlDataAdapter object...
Dim objDataAdapter As New SqlDataAdapter()

'Create an instance of a new select command object
objDataAdapter.SelectCommand = New SqlCommand()

 ' Set the SelectCommand properties...
objDataAdapter.SelectCommand.Connection = objConnection
objDataAdapter.SelectCommand.CommandText = "usp_select_DSSV"
objDataAdapter.SelectCommand.CommandType =
CommandType.StoredProcedure
Dim objDataSet as DataSet = New DataSet()

 ' Fill the DataSet object with data...
objDataAdapter.Fill(objDataSet, "DSSV")
```

Phương thức Fill sử dụng thuộc tính SelectCommand.Connection để connect tới database. Nếu connection đã được mở, data adapter sẽ sử dụng nó để thực hiện SelectCommand. Và nếu như connection đã đóng thì data adapter sẽ mở nó, thực hiện SelectCommand, và sau đó đóng trở trở lại. Bây giờ ta đã có dữ liệu trong bộ nhớ, và có thể bắt đầu thao tác nó không phụ thuộc vào data source.

Chú ý: Lớp DataSet không có Sql tại vị trí bắt dầu tên lớp của nó. Bởi vì DataSet không nằm trong namespace System.Data.SqlClient, nó nằm trong lớp cha namespace System.Data. Các lớp trong tên miền này có liên quan căn bản với thao các dữ liệu trong bộ nhớ.

Ta có thể tạo một DataView từ dữ liệu được chứa trong DataTable mà đang chỉ chứa dữ liệu mà ta muốn hiển thị. Ví dụ, nếu dữ liệu trong DataTable chứa tất cả các Sinh viên được sắp xếp bởi *Tên* và *Họ đệm*, ta có thể tạo một DataView chứa tất cả *Sinh viên* được sắp xếp bởi Tên và sau đó là Họ đệm. Hoặc, nếu muốn, ta có thể tạo một DataView chỉ chứa một vài sinh viên chứa tên nào đó.

Mặc dù ta có thể view dữ liệu trong DataView trong nhiều cách khác nhau từ DataTable cơ bản, nó vẫn là cùng dữ liệu. Các thay đổi với DataView ảnh hưởng tới DataTable cơ bản một cách tự động và các thay đổi với DataTable cơ bản tự động ảnh hưởng bất kỳ đối tượng DataView mà đang hiển thị DataTable đó.

Kiến trúc cho lớp DataView khởi tạo một thể hiện mới của lớp DataView và lấy DataTable là một đối số. Đoạn code sau khai báo một đối tượng DataView và khởi tạo nó đang sử dụng bảng HosoSV từ DataSet có tên là objDataSet. Chú ý rằng code đó truy cập vào tập hợp Tables của đối tượng DataSet, bằng việc chỉ định thuộctính Tables và tên bảng:

```
' Set the DataView object to the DataSet object...
Dim objDataView = New DataView(objDataSet.Tables("HosoSV"))
```

**** Thuộc tính Sort***

Khi *DataView* đã được khởi tạo và hiển thị dữ liệu, ta có thể thay đổi view của dữ liệu đó. Ví dụ, giả sử ta muốn sắp xếp dữ liệu khác với thứ tự trong DataSet. Để sắp xếp dữ liệu trong DataView, ta thiết lập thuộc tính Sort và chỉ định cột hoặc các cột mà ta muốn sắp xếp. Đoạn code sau sẽ sắp xếp dữ liệu trong DataView theo Tên và sau đó theo Họ đệm của danh sách Sinh viên:

```
objDataView.Sort = "TenSV, Hodem"
```

Chú ý: Thuộc tính Sort giống như cú pháp của mệnh đề ORDER BY trong câu lệnh SQL. Như trong SQL mệnh đề ORDER BY, thao tác sắp xếp trên DataView thường sắp xếp theo thứ tự mặc định là tăng dần. Nếu ta muốn thay đổi theo thứ tự sắp xếp giảm, ta cần phải có chỉ định từ khóa DESC:

```
objDataView.Sort = "TenSV, Hodem DESC"
```

**** Thuộc tính RowFilter***

Khi đã khởi tạo DataView, ta có thể lọc các dòng dữ liệu mà chúng sẽ chứa. Nó tương tự như chỉ định trong mệnh đề WHERE của câu lệnh SQL SELECT; chỉ những hàng thỏa mãn điều kiện sẽ được giữ lại trên view. Dữ liệu cơ bản không bị ảnh hưởng. Thuộc tính RowFilter chỉ định một điều kiện lọc mà có thể áp dụng được trên DataView. Cú pháp này cũng tương tự như mệnh đề SQL WHERE. Nó chứa ít nhất tên một cột, theo sau là một toán tử và giá trị. Nếu giá trị là string thì nó phải được bao trong cặp dấu 'nháy đơn', ví dụ ta có đoạn code sau:

```
' Set the DataView object to the DataSet object...
objDataView = New DataView(objDataSet.Tables("HosoSV"))
objDataView.RowFilter = "TenSV = 'Hoa'"
```

Nếu bạn muốn lấy lại các ròng trừ các dòng có Tên là ='Hoa'

```
' Set the DataView object to the DataSet object...
objDataView = New DataView(objDataSet.Tables("authors"))
objDataView.RowFilter = "TenSV <> 'Hoa'"
```

Ta cũng có thể chỉ định điều kiện lọc phức tạp hơn như trong SQL. Ví dụ, sử dụng toán tử AND:

```
objDataView.RowFilter = "TenSV <> 'Hoa' AND Hodem LIKE 'N*'"
```

**** Phương thức Find***

Nếu ta muốn tìm kiếm cho một hàng dữ liệu trong DataView, ta gọi phương thức Find. Phương thức Find tìm kiếm dữ liệu trong cột khóa sắp xếp của DataView. Do đó, trước khi gọi phương thức Find, ta trước hết hãy sắp xếp DataView trên cột chứa dữ liệu mà ta muốn tìm kiếm. Cột trong DataView đã được sắp xếp trở thành cột khóa sắp xếp trong đối tượng DataView object.

Ví dụ, giả sử ta muốn tìm kiếm Sinh viên có TenSV là 'Anh'. Ta cần sắp xếp dữ liệu trong DataView theo TenSV và thiết lập cột này thành cột khóa sắp xếp trong DataView, và sau đó gọi phương thức Find, như đoạn code sau:

```
Dim intPosition as Integer
objDataView.Sort = "TenSV"
intPosition = objDataView.Find("Ann")
```

Ta thực hiện xây dựng from trên thông qua các đối tượng sau.

1. Tạo một Project Windows Application tên là QL_DiemSV.

2. Tạo Form Nhập hồ sơ sinh viên (frmNhapHSSV) như hình có các đối tượng sau:

Object	***Property***	***Setting***
Form	Name Text	frmNhapHSSV Nhap Ho so sinh vien
Label	Name Text	Label1 NHẬP DANH SÁCH SINH VIÊN
Label	Name Text	lblMaSV Mã sinh viên
Label	Name Text	lblHodem Họ đệm
Label	Name Text	lblTenSV Tên sinh viên
Label	Name Text	lblMaLop Mã lớp
Textbox	Name	txtMaSV
Textbox	Name	txtHodem
Textbox	Name	txtTenSV
Textbox	Name	txtMaLop
Combo	Name	cboMaLop
Button	Name Text	btnNew New
Button	Name Text	btnAdd Add
Button	Name Text	btnUpdate Update
Button	Name Text	btnDelete Delete

Textbox	Name	txtRecordPosition
Button	Name Text	btnMoveFirst |<
Button	Name Text	btnMovePrevious <
Button	Name Text	btnMoveNext >
Button	Name Text	btnMoveLast >|
StatusStrip	Name Text	StatusStrip1 StatusStrip1
ToolStripStatusLabel	Name Text	ToolStripStatusLabel1 ToolStripStatusLabel1
Button	Name Text	btnClose Close

Sau khi thiết kế các đối tượng trên ta được một form như hình,

3. Import các tên miền cần thiết. Mở code editor và chèn đoạn code sau.

```
' Import Data and SqlClient namespaces...
Imports System.Data

Imports System.Data.SqlClient
```

4. Tiếp theo khai báo các đối tượng global trong phạm vi của form này.

```
' Import Data and SqlClient namespaces...
Imports System.Data
Imports System.Data.SqlClient

Public Class frmNhapHSSV
#Region "Khai bao cac ket noi"
    Dim objConnection As New SqlConnection _
       ("server=THUHUONG;database=QLDiemSV;user id=sa;password=pass2008")
    Dim objDataAdapter As New SqlDataAdapter( _
        "SELECT MaSV, HoDem, TenSV, NgaySinh, MaLop " & _
        "FROM HOSOSV ", objConnection)
    Dim objDataSet As DataSet
    Dim objDataView As DataView
    Dim objCurrencyManager As CurrencyManager

#End Region
```

Chú ý: Khi khai báo kết nối cần phải đảm bảo đúng tên server, user id và password.

5. Thủ tục đầu tiên ta sẽ tạo đó là thủ tục FillDataSetAndView. Thủ tục có thể coi là thủ tục khởi tạo các đối tượng.

```
Private Sub FillDataSetAndView()
    ' Initialize a new instance of the DataSet object...
    objDataSet = New DataSet()

    ' Fill the DataSet object with data...
    objDataAdapter.Fill(objDataSet, "Hoso")

    ' Set the DataView object to the DataSet object...
    objDataView = New DataView(objDataSet.Tables("Hoso"))

    ' Set our CurrencyManager object to the DataView object...
    objCurrencyManager = CType(Me.BindingContext(objDataView),
CurrencyManager)
  End Sub
```

6. Thủ tục tiếp theo sẽ thực hiện kết nối các điều khiển trên form với đối tượng Dataview.

```
Private Sub BindFields()


    ' Clear any previous bindings...
    txtMaSV.DataBindings.Clear()
    txtHodem.DataBindings.Clear()
    txtTenSV.DataBindings.Clear()
    txtNgaySinh.DataBindings.Clear()
    txtMaLop.DataBindings.Clear()

    ' Add new bindings to the DataView object...
    txtMaSV.DataBindings.Add("Text", objDataView, "MaSV")
    txtHodem.DataBindings.Add("Text", objDataView, "HoDem")
    txtTenSV.DataBindings.Add("Text", objDataView, "TenSV")
    txtNgaySinh.DataBindings.Add("Text", objDataView, "Ngaysinh")
    txtMaLop.DataBindings.Add("Text", objDataView, "MaLop")

    ' Display a ready status...
    ToolStripStatusLabel1.Text = "Ready"
  End Sub
```

7. Thủ tục tiếp theo là sẽ hiển thị vị trí của bản ghi hiện thời trên form.

```
Private Sub ShowPosition()
    ' Display the current position and the number of records
    txtRecordPosition.Text = objCurrencyManager.Position + 1 & _
    " / " & objCurrencyManager.Count()

End Sub
```

8. Ta đã xây dựng các thủ tục. Ta sẽ gọi các thủ tục đó. Double-click Form Designer, chọn sự kiện Load form sau đó add đoạn code sau.

```
 Private Sub frmNhapHSSV_Load(ByVal sender As System.Object, ByVal e
As System.EventArgs) Handles MyBase.Load

  ' Fill the DataSet and bind the fields...
```

```
        Me.txtMaSV.Enabled = False
        FillDataSetAndView()
        BindFields()

       ' Show the current record position...
        ShowPosition()

    End Sub
```

Bây giờ ta sẽ chạy thử form: Debug\Start Debugging (F5)

9. Tiếp theo ta sẽ viết code cho các navigation buttons. Để thực hiện duyệt các bản ghi trên form.

```
    Private Sub btnMoveFirst_Click(ByVal sender As Object, ByVal e As
System.EventArgs) Handles btnMoveFirst.Click

        ' Set the record position to the first record...
        objCurrencyManager.Position = 0

        ' Show the current record position...
        ShowPosition()
    End Sub

    Private Sub btnMovePrevious_Click(ByVal sender As Object, ByVal e As
System.EventArgs) Handles btnMovePrevious.Click
        ' Move to the previous record...
        objCurrencyManager.Position -= 1
        ' Show the current record position...
        ShowPosition()
    End Sub

    Private Sub btnMoveNext_Click(ByVal sender As Object, ByVal e As
System.EventArgs) Handles btnMoveNext.Click
        ' Move to the next record...
        objCurrencyManager.Position += 1

        ' Show the current record position...
        ShowPosition()
    End Sub

    Private Sub btnLast_Click(ByVal sender As Object, ByVal e As
System.EventArgs) Handles btnLast.Click
        ' Set the record position to the last record...
        objCurrencyManager.Position = objCurrencyManager.Count - 1

        ' Show the current record position...
        ShowPosition()

        End Sub
```

10. Thêm các bản ghi.

- Trước hết ta đưa đoạn code sau vào thủ tục của nút New.

```
    Private Sub btnNew_Click(ByVal sender As Object, ByVal e As
System.EventArgs) Handles btnNew.Click

        txtMaSV.Text = ""
        txtHodem.Text = ""
```

```
        txtTenSV.Text = ""
        txtNgaySinh.Text = ""
        txtMaLop.Text = ""

        Me.txtMaSV.Enabled = True
        Me.cboMaLop.Visible = True
        Me.txtMaLop.Visible = False

        ' Declare local variables and objects...
        Dim objCommand As SqlCommand = New SqlCommand()
        Dim objDataReader As SqlDataReader

        ' Open the connection, execute the command
        objConnection.Open()
        ' Set the SqlCommand object properties...
        objCommand.Connection = objConnection
        objCommand.CommandText = "Select MaLop, TenLop From Lop"

        ' Execute the SqlCommand object to insert the new data...
        objDataReader = objCommand.ExecuteReader
        cboMaLop.Items.Clear()

        Do While (objDataReader.Read())
            cboMaLop.Items.Add(objDataReader.Item(0))
        Loop

        ' Close the connection...
        objConnection.Close()
    End Sub
```

Đối với thủ tục này, ta sẽ cho các đối tượng Text box về rỗng để nhập dữ liệu. Riêng Text box txtMaLop sẽ được ẩn đi và thay vào đó là Combo box cboMaLop. Do mối quan hệ giữa hai quan hệ HOSOSV và LOP, cboMaLop sẽ chứa MaLop các lớp có trong danh mục Lop.

Trong phần này, ta có sử dụng đối tượng DataReader của lớp SqlDataReader. Khai báo đối tượng:

```
' Declare local variables and objects...
Dim objCommand As SqlCommand = New SqlCommand()
Dim objDataReader As SqlDataReader
```

Thực hiện đối tượng Command và lưu kết quả để đọc vào đối tượng objDataReader. Và truy xuất dữ liệu đó.

```
' Execute the SqlCommand object to insert the new data...
objDataReader = objCommand.ExecuteReader
cboMaLop.Items.Clear()

Do While (objDataReader.Read())
    cboMaLop.Items.Add(objDataReader.Item(0))
Loop
```

- Chèn dữ liệu vào CSDL. Đưa đoạn code sau vào sự kiện click của nút lệnh Add.

```
Private Sub btnAdd_Click(ByVal sender As Object, ByVal e As
System.EventArgs) Handles btnAdd.Click
        ' Declare local variables and objects...
        Dim intPosition As Integer
        Dim objCommand As SqlCommand = New SqlCommand()

        ' Save the current record position...
        intPosition = objCurrencyManager.Position
        ' Open the connection, execute the command
        objConnection.Open()
        ' Set the SqlCommand object properties...
        objCommand.Connection = objConnection
        objCommand.CommandText = "INSERT INTO HOSOSV " & _
            "(MaSV, HoDem, TenSV, NgaySinh, MaLop) " & _
            "VALUES(@MaSV,@Hodem,@TenSV,@NgaySinh,@MaLop)"
        ' Add parameters for the placeholders in the SQL in the
        ' CommandText property...
        objCommand.Parameters.AddWithValue("@MaSV", txtMaSV.Text)
        objCommand.Parameters.AddWithValue("@Hodem", txtHodem.Text)
        objCommand.Parameters.AddWithValue("@TenSV", txtTenSV.Text)
        objCommand.Parameters.AddWithValue("@NgaySinh",_
                                    txtNgaySinh.Text).DbType = DbType.Date
        objCommand.Parameters.AddWithValue("@MaLop", cboMaLop.Text)

        ' Execute the SqlCommand object to insert the new data...
        Try
            objCommand.ExecuteNonQuery()
        Catch SqlExceptionErr As SqlException
            MessageBox.Show(SqlExceptionErr.Message)
        End Try

        ' Close the connection...
        objConnection.Close()

        ' Fill the dataset and bind the fields...
        FillDataSetAndView()
        BindFields()
        ' Set the record position to the one that you saved...
        objCurrencyManager.Position = intPosition
        ' Show the current record position...
        ShowPosition()
        ' Display a message that the record was added...
        ToolStripStatusLabel1.Text = "Record Added"
        Me.txtMaSV.Enabled = False
        Me.cboMaLop.Visible = False
        Me.txtMaLop.Visible = True

    End Sub
```

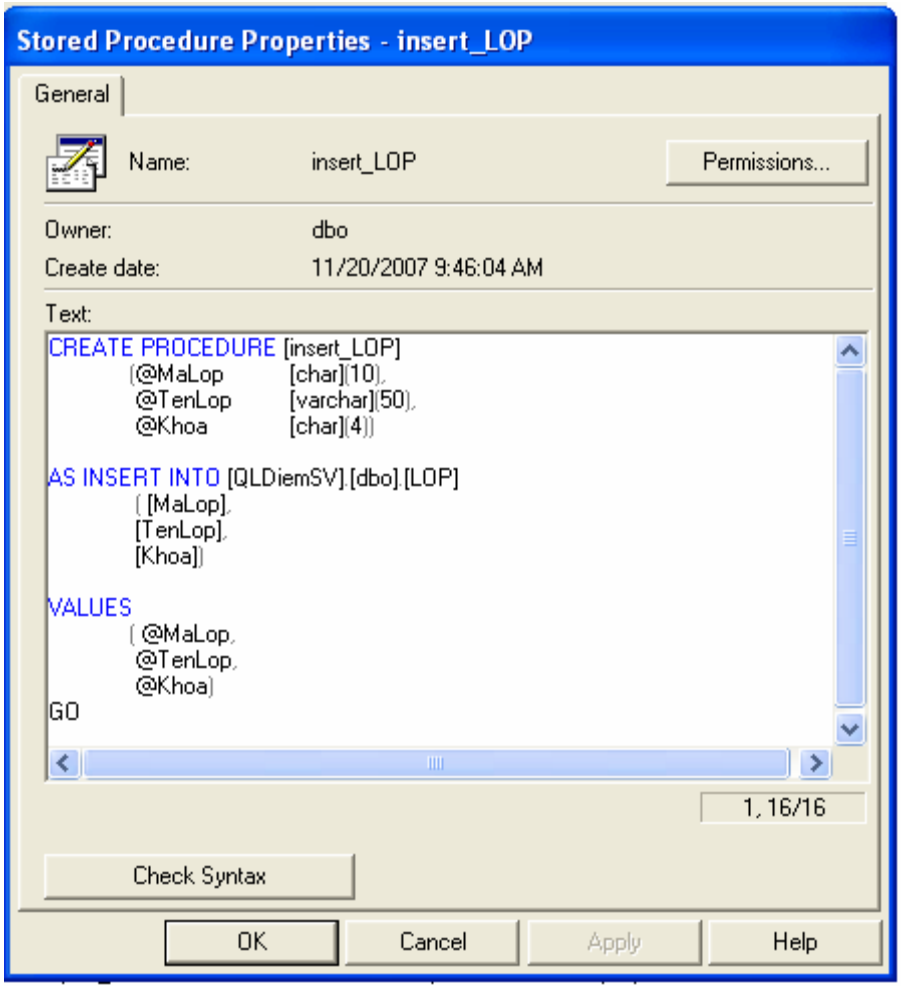
11. Thực hiện Update các bản ghi.

```
Private Sub btnUpdate_Click(ByVal sender As Object, ByVal e As
System.EventArgs) Handles btnUpdate.Click
        ' Declare local variables and objects...
        Dim intPosition As Integer
        Dim objCommand As SqlCommand = New SqlCommand()

        ' Save the current record position...
```

```
        intPosition = objCurrencyManager.Position

        ' Set the SqlCommand object properties...
        objCommand.Connection = objConnection
        objCommand.CommandText = "UPDATE HOSOSV " & _
                "SET  HoDem = @HoDem, TenSV=@TenSV, NgaySinh=@NgaySinh,
MaLop=@MaLop Where MaSV = @MaSV"
        objCommand.CommandType = CommandType.Text
        ' Add parameters for the placeholders in the SQL in the
        ' CommandText property...
        objCommand.Parameters.AddWithValue("@Hodem", txtHodem.Text)
        objCommand.Parameters.AddWithValue("@TenSV", txtTenSV.Text)
        objCommand.Parameters.AddWithValue("@NgaySinh",
txtNgaySinh.Text).DbType = DbType.Date
        objCommand.Parameters.AddWithValue("@MaLop", txtMaLop.Text)
        objCommand.Parameters.AddWithValue_
("@MaSV", BindingContext(objDataView).Current("MaSV"))

        ' Open the connection...
        objConnection.Open()
        ' Execute the SqlCommand object to update the data...
        objCommand.ExecuteNonQuery()
        ' Close the connection...
        objConnection.Close()
        ' Fill the DataSet and bind the fields...
        FillDataSetAndView()
        BindFields()
        ' Set the record position to the one that you saved...
        objCurrencyManager.Position = intPosition
        ' Show the current record position...
        ShowPosition()

        ' Display a message that the record was updated...
        ToolStripStatusLabel1.Text = "Record Updated"

    End Sub
```

12. Thực hiện Delete các bản ghi.

```
Private Sub btnDelete_Click(ByVal sender As Object, ByVal e As
System.EventArgs) Handles btnDelete.Click
        ' Declare local variables and objects...
        Dim intPosition As Integer
        Dim objCommand As SqlCommand = New SqlCommand()

        ' Save the current record position - 1 for the one to be
        ' deleted...
        intPosition = Me.BindingContext(objDataView).Position - 1

        ' If the position is less than 0 set it to 0...
        If intPosition < 0 Then
            intPosition = 0
        End If

        ' Set the Command object properties...
        objCommand.Connection = objConnection
        objCommand.CommandText = "DELETE FROM HOSOSV " & _
                "WHERE MaSV = @MaSV;"
        ' Parameter for the MaSv field...
        objCommand.Parameters.AddWithValue _
                        ("@MaSV",
BindingContext(objDataView).Current("MaSV"))
```



- Viết code cho nút Add này như sau:

```
Private Sub btnAdd_Click(ByVal sender As Object, ByVal e As
System.EventArgs) Handles btnAdd.Click

        ' Declare local variables and objects...
        Dim intPosition As Integer
        Dim objCommand As SqlCommand = New SqlCommand()

        ' Save the current record position...
        intPosition = objCurrencyManager.Position

        ' Open the connection, execute the command
        objConnection.Open()
        ' Set the SqlCommand object properties...
        objCommand.Connection = objConnection
        objCommand.CommandText = "insert_Lop"
        objCommand.CommandType = CommandType.StoredProcedure

        ' Add parameters for the placeholders in the SQL in the
        ' CommandText property...
        objCommand.Parameters.AddWithValue("@MaLop", txtMaLop.Text)
        objCommand.Parameters.AddWithValue("@TenLop", txtTenLop.Text)
        objCommand.Parameters.AddWithValue("@Khoa", txtKhoa.Text)

          ' Execute the SqlCommand object to insert the new data...
```

Mục đích của Form thực hiện hiển thị danh sách sinh viên theo từng lớp. Ngoài ra ra cho phép thực hiện các thao tác:

- Cập nhật dữ liệu trên Grid.
- Sửa đổi dữ liệu trên Grid.
- Xóa dữ liệu trên Grid.
- Sắp xếp dữ liệu.

1. Xây dựng Form như hình trên gồm các control sau:

Object	***Property***	***Setting***
Form	Name Text	frmDanhSach Danh sách sinh viên
Label	Name Text	lblDSL Danh sách lớp
Label	Name Text	lblDSSV DANH SÁCH SINH VIÊN
DataGridView	Name	dgDanhsach
Button	Name Text	btnClose Close
Button	Name Text	btnInsert Insert
Button	Name Text	btnUpdate Update
Button	Name Text	btnDelete Delete
Button	Name Text	btnUndo Undo
Button	Name Text	btnSort Sắp xếp
ComboBox	Name	cboMaLop
ComboBox	Name Items	cboSX Tên; Họ tên; Mã sinh viên

2. Import các tên miền cần thiết. Mở code editor và chèn đoạn code sau.

```
' Import Data and SqlClient namespaces...
Imports System.Data
Imports System.Data.SqlClient
```

3. Tiếp theo khai báo các đối tượng global trong phạm vi của form này.

```
#Region "Khai bao cac ket noi"
    Dim objConnection As New SqlConnection _
            ("server=THUHUONG;database=QLDiemSV;user
id=sa;password=12102006")
    Dim objDataAdapter As New SqlDataAdapter()
    Dim objDataSet As DataSet
    Dim objDataView As DataView
#End Region
```

Chú ý: Khi khai báo kết nối cần phải đảm bảo đúng tên server, user id và password.

4. Thủ tục đầu tiên ta sẽ tạo đó là thủ tục FillDataSetAndView. Thủ tục có thể coi là thủ tục khởi tạo các đối tượng.

```
    Private Sub FillDataSetAndView()

        ' Initialize a new instance of the DataSet object...
        objDataSet = New DataSet()
        objDataAdapter.SelectCommand = New SqlCommand()
        objDataAdapter.SelectCommand.Connection = objConnection
        objDataAdapter.SelectCommand.CommandText = _
        "SELECT MaSV, HoDem, TenSV, NgaySinh, MaLop FROM HOSOSV WHERE
MaLop= '" & cboMaLop.SelectedItem(0) & "'"

        objDataAdapter.SelectCommand.CommandType = CommandType.Text

        ' Open the database connection...
        objConnection.Open()

        ' Fill the DataSet object with data...
        objDataAdapter.Fill(objDataSet, "HosoSV")

        ' Close the database connection...
        objConnection.Close()

        ' Set the DataView object to the DataSet object...
        objDataView = New DataView(objDataSet.Tables("HosoSV"))

    End Sub
```

5. Thủ tục tiếp theo là thủ tục thực hiện trang trí DataGridView

```
Private Sub Grid()
' * trang tri Datagrid
' Declare and set the currency header alignment property...
Dim objAlignRightCellStyle As New DataGridViewCellStyle
objAlignRightCellStyle.Alignment =
DataGridViewContentAlignment.MiddleRight

' Declare and set the alternating rows style...
Dim objAlternatingCellStyle As New DataGridViewCellStyle()
objAlternatingCellStyle.BackColor = Color.WhiteSmoke
dgDanhSach.AlternatingRowsDefaultCellStyle = objAlternatingCellStyle

' Change column names and styles using the column index
dgDanhSach.Columns(0).HeaderText = "Mã sinh viên"
dgDanhSach.Columns(1).HeaderText = "Họ đệm"
dgDanhSach.Columns(2).HeaderText = "Tên"
dgDanhSach.Columns(3).HeaderText = "Ngày sinh"
dgDanhSach.Columns(4).HeaderText = "Mã lớp"

' Clean up
objAlternatingCellStyle = Nothing
objAlignRightCellStyle = Nothing
End Sub
```

6. Viết code cho thủ tục Load form.

```
Private Sub frmDanhSach_Load(ByVal sender As Object, ByVal e As
System.EventArgs) Handles Me.Load

 ' Dien du lieu vào cboMaLop
 Dim objDA_Lop As New SqlDataAdapter()
 Dim objDS_Lop As New DataSet()

 objDA_Lop.SelectCommand = New SqlCommand()

 objDA_Lop.SelectCommand.Connection = objConnection
 objDA_Lop.SelectCommand.CommandText = "SELECT MaLop, TenLop FROM Lop"
 objDA_Lop.SelectCommand.CommandType = CommandType.Text
 objConnection.Open()
 ' Fill the DataSet object with data...
 objDA_Lop.Fill(objDS_Lop, "DMLop")
 ' Close the database connection...
 objConnection.Close()

 Me.cboMaLop.DataSource = objDS_Lop.Tables("DMLop")
 Me.cboMaLop.DisplayMember = "TenLop"
 Me.cboMaLop.ValueMember = "MaLop"

 'Khởi tạo dữ liệu
  FillDataSetAndView()
  Me.dgDanhSach.AutoGenerateColumns = True

  Me.dgDanhSach.DataSource = objDataView
  Grid()
End Sub
```

7. Viết code cho nút lệnh Close để thực hiện đóng form.

```
Private Sub btnClose_Click(ByVal sender As Object, ByVal e As
System.EventArgs) Handles btnClose.Click
        me.Close
End Sub
```

8. Viết code cho nút lệnh Update để thực hiện ghi lại các thay đổi dữ liệu vềcơ sở dữ liệu.

```
   Private Sub btnUpdate_Click(ByVal sender As Object, ByVal e As
System.EventArgs) Handles btnUpdate.Click
        ' automatically create update/delete/insert commands
        Dim objCommandBuilder As SqlCommandBuilder = New
SqlCommandBuilder(objDataAdapter)
        objDataAdapter.UpdateCommand = objCommandBuilder.GetUpdateCommand
        ' Open the connection, execute the command
        objConnection.Open()
        objDataAdapter.Update(objDataSet, "HosoSV")
        objDataSet.Tables("HosoSV").AcceptChanges()
        objConnection.Close()
   End Sub
```

9.Viết code cho nút lệnh Insert để thực hiện chèn dữ liệu thực sự vào cơ sở dữ liệu.

```
Private Sub btnInsert_Click(ByVal sender As Object, ByVal e As
System.EventArgs) Handles btnInsert.Click
        ' automatically create update/delete/insert commands
        Dim objCommandBuilder As SqlCommandBuilder = New
SqlCommandBuilder(objDataAdapter)
        objDataAdapter.InsertCommand = objCommandBuilder.GetInsertCommand
        ' Open the connection, execute the command
        objConnection.Open()
        objDataAdapter.Update(objDataSet, "HosoSV")
        objDataSet.Tables("HosoSV").AcceptChanges()
        objConnection.Close()
End Sub
```

10.Viết code cho nút lệnh Delete để thực hiện xóa thực sự dữ liệu ở CSDL, sau khi thực hiện xóa tạm trên Grid.

```
  Private Sub btnDelete_Click(ByVal sender As Object, ByVal e As
System.EventArgs) Handles btnDelete.Click
        ' automatically create update/delete/insert commands
        Dim objCommandBuilder As SqlCommandBuilder = New
SqlCommandBuilder(objDataAdapter)
        objDataAdapter.DeleteCommand = objCommandBuilder.GetDeleteCommand
        ' Open the connection, execute the command
        objConnection.Open()
```

```
        objDataAdapter.Update(objDataSet, "HosoSV")
        objDataSet.Tables("HosoSV").AcceptChanges()
        objConnection.Close()
        Me.Refresh()
End Sub
```

11.Viết code cho nút lệnh Undo để thực hiện Undo lại các thao tác tạm trên Grid mà chưa thi hành thực sự đối với CSDL.

```
  Private Sub btnUndo_Click(ByVal sender As Object, ByVal e As
System.EventArgs) Handles btnUndo.Click
        Me.Refresh()
        FillDataSetAndView()

        Me.dgDanhSach.AutoGenerateColumns = True
        Me.dgDanhSach.DataSource = objDataView
        Grid()
End Sub
```

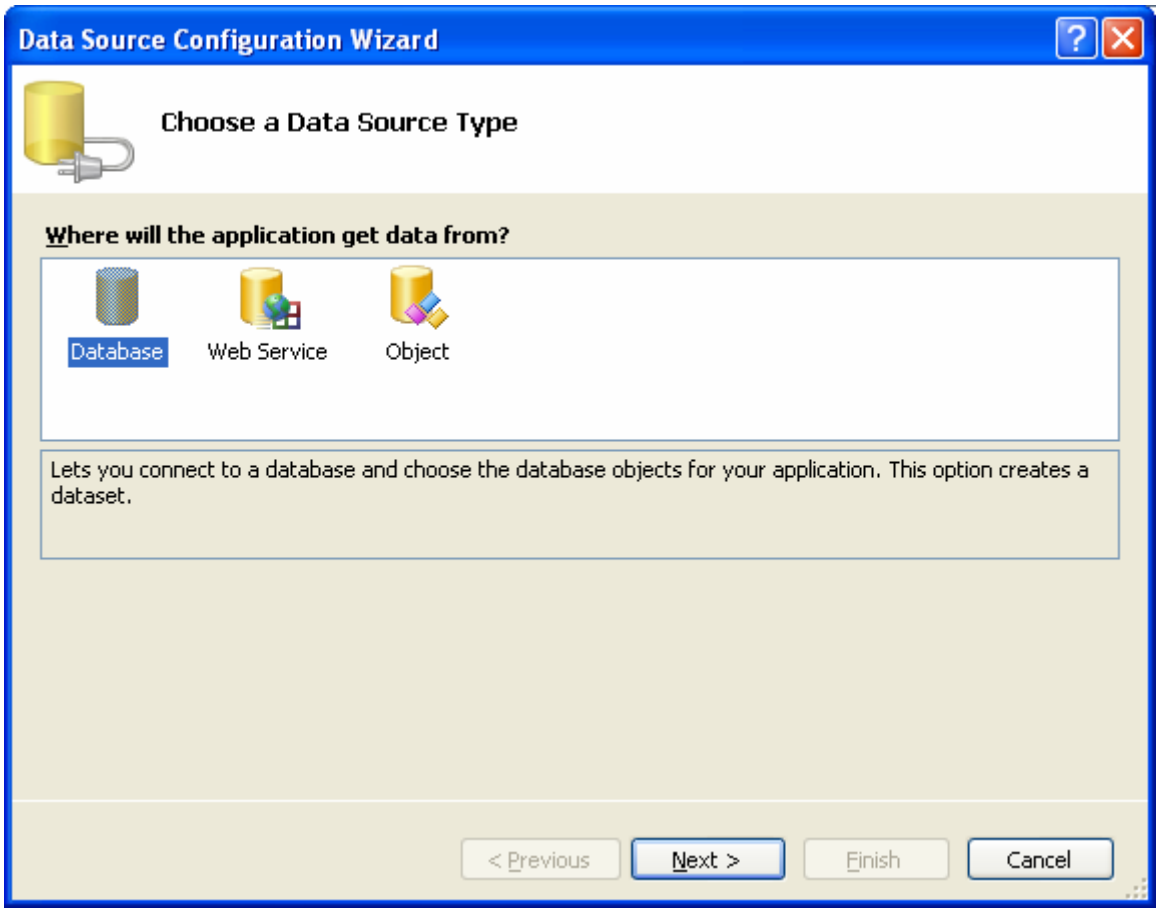
12.Viết code cho nút lệnh Sắp xếp để thực hiện để thực hiện sắp xếp các bản ghi trên Grid theo 3 tiêu chí.

- Sắp xếo theo mã sinh viên
- Sắp xếp theo tên sinh viên
- Sắp xếp theo Họ và tên sinh viên.

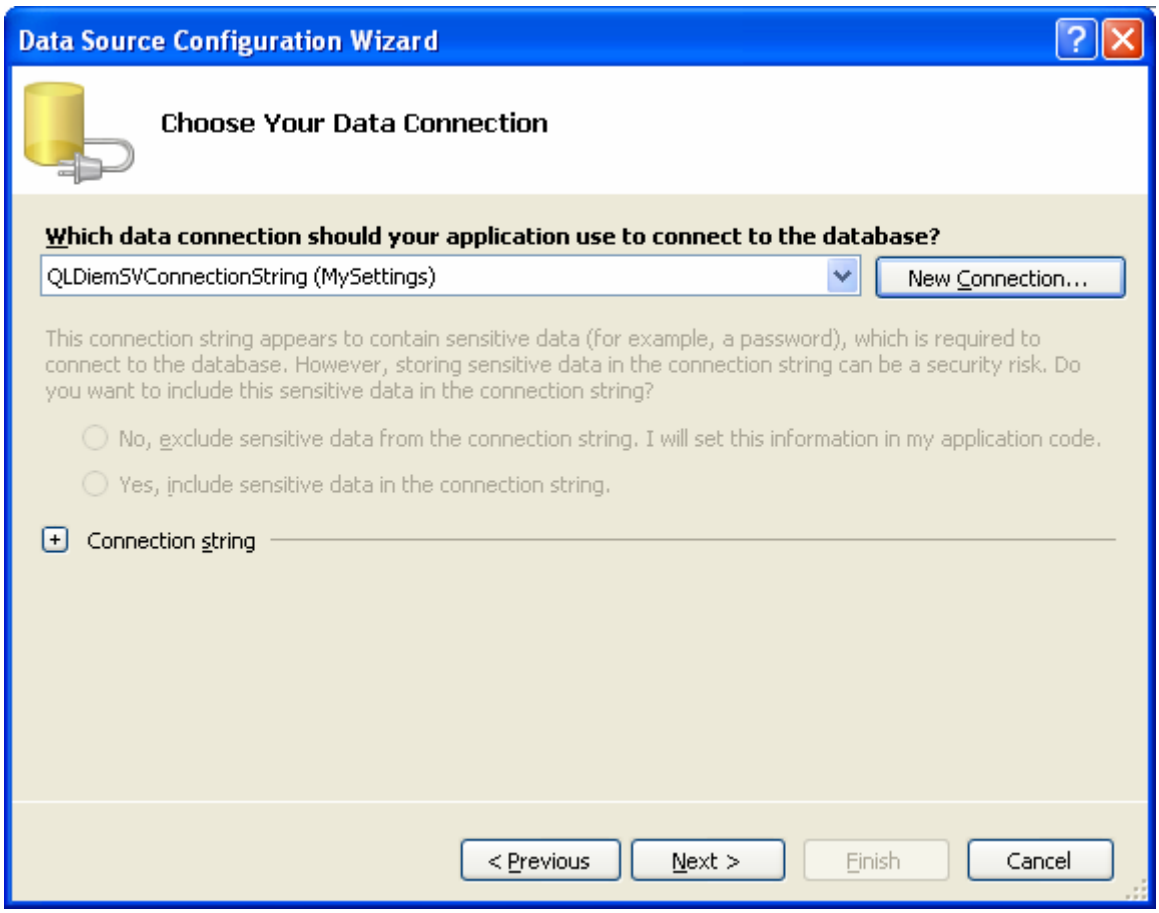
```
Private Sub btnSort_Click(ByVal sender As Object, ByVal e As
System.EventArgs) Handles btnSort.Click
        ' Sort property of the DataView object...
        FillDataSetAndView()
        Select Case cboSX.SelectedIndex
            Case 0 'Ten
                objDataView.Sort = "TenSV"
            Case 1 'Hoten
                objDataView.Sort = "TenSV, Hodem"
            Case 2 'Ma SV
                objDataView.Sort = "MaSV"
        End Select

        Me.dgDanhSach.AutoGenerateColumns = True
        Me.dgDanhSach.DataSource = objDataView
        Grid()
End Sub
```

13.Viết code cho sự kiện chọn dữ liệu trêm cboMaLop. Đoạn code này nhằm mục đích đồng bộ hóa dữ liệu trên Form.

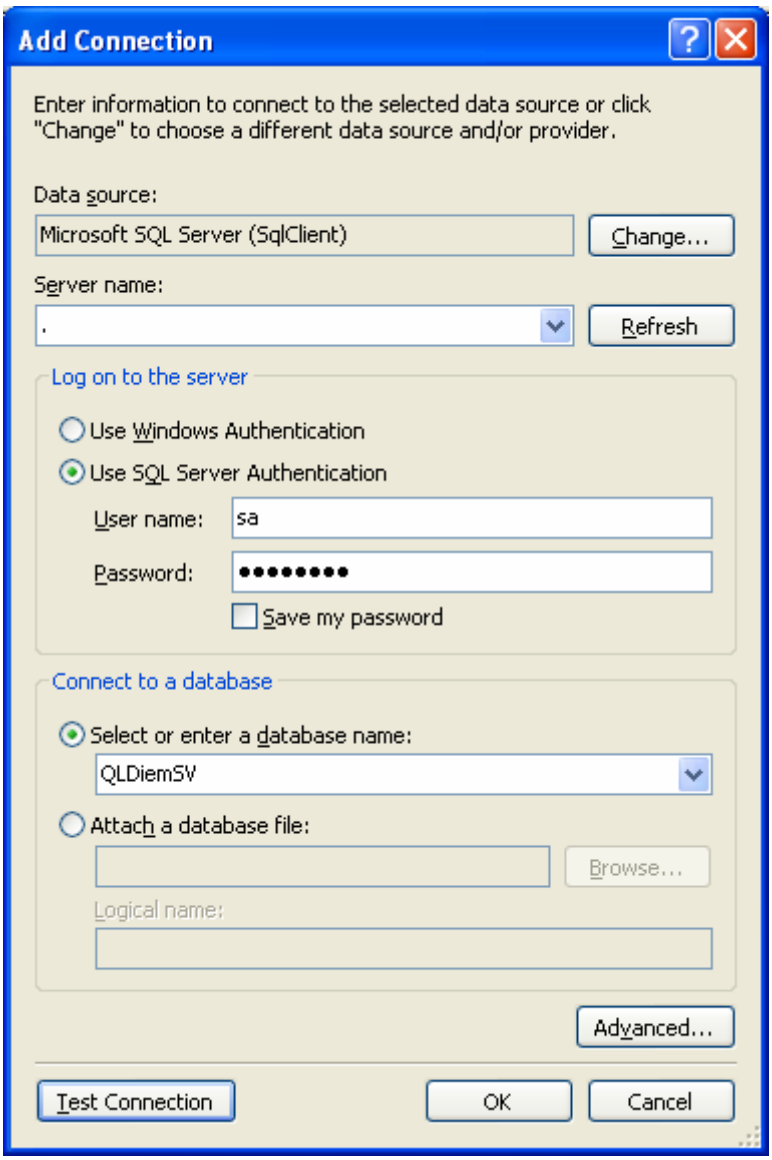
```
  Private Sub cboMaLop_SelectedIndexChanged(ByVal sender As Object, ByVal
e As System.EventArgs) Handles cboMaLop.SelectedIndexChanged
```



Cửa sổ 2: Click nút New Connection

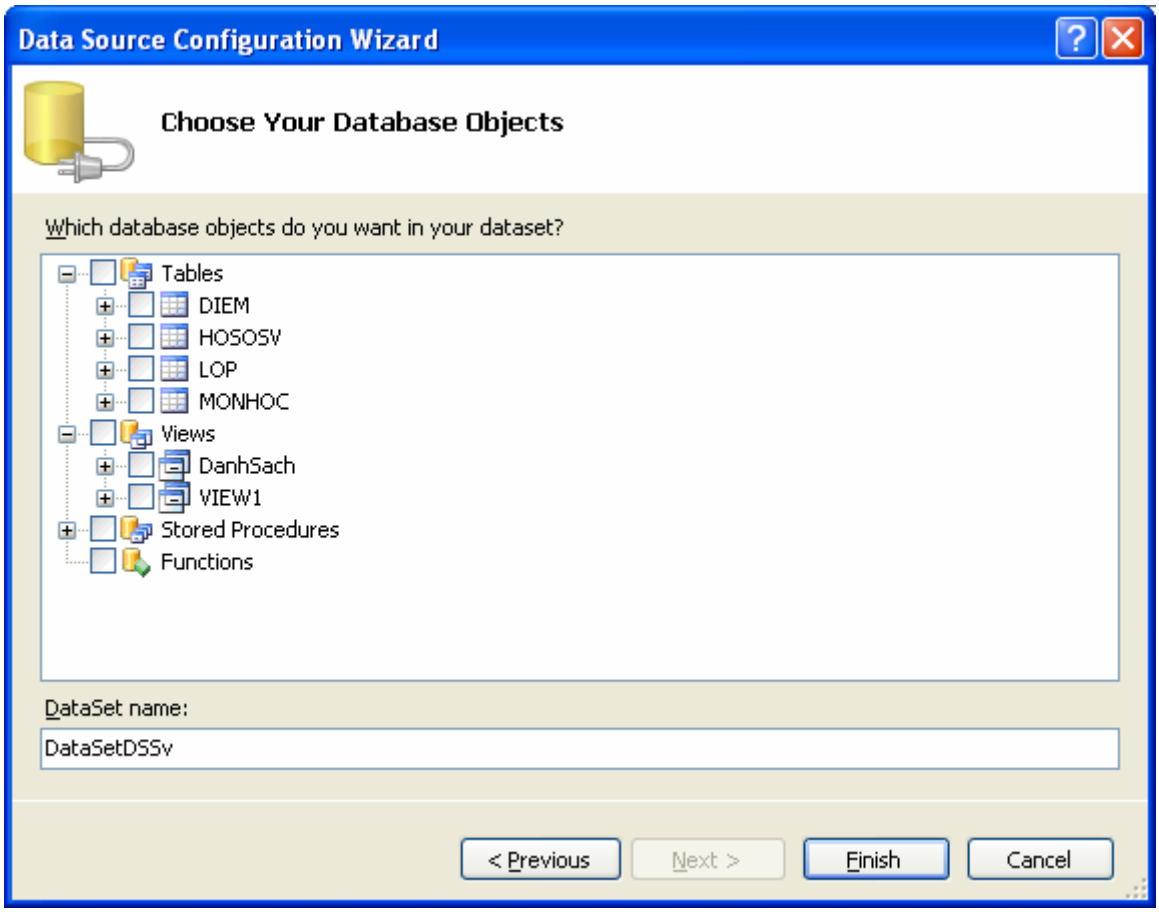

Data Source Configuration Wizard
Choose Your Data Connection
Which data connection should your application use to connect to the database?
QLDiemSVConnectionString (MySettings)
New Connection...
This connection string appears to contain sensitive data (for example, a password), which is required to connect to the database. However, storing sensitive data in the connection string can be a security risk. Do you want to include this sensitive data in the connection string?
No, exclude sensitive data from the connection string. I will set this information in my application code.
Yes, include sensitive data in the connection string.
Connection string
< Previous
Next >
Finish
Cancel

Cửa sổ 3: Cửa sổ Add new Connection. Ta thực hiện điền các tham số như hình.



Sau khi chọn OK, nó sẽ quay trở lại cửa sổ 2.

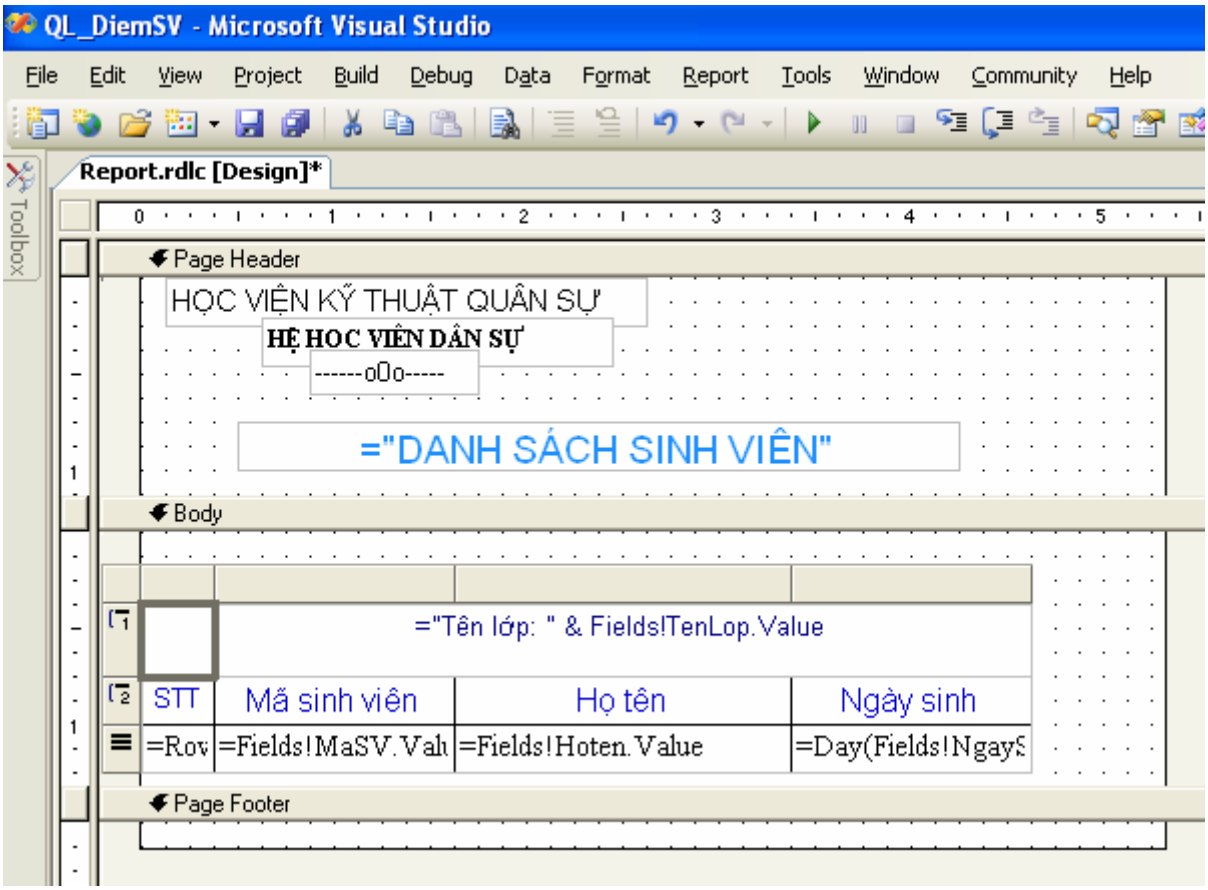
Cửa sổ 4. Chọn dữ liệu sẽ thể hiện trên Report và đặt tên cho Dataset.



Chọn view Danh sach → Click nút Finish.

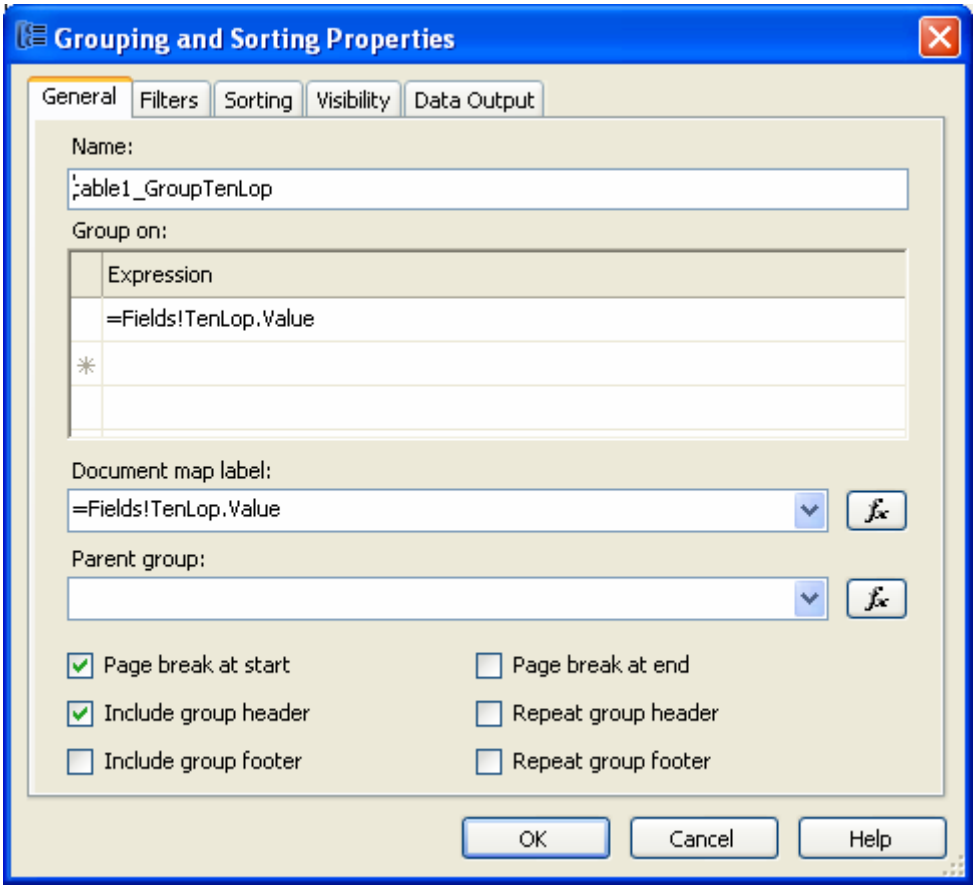
3. Sau khi có nguồn dữ liệu, trong cửa sổ thiết kế Report ta sẽ thiết kế dữ liệu sẽ hiển thị trên Report. Như hình, gồm các đối tượng sau:

Object	*Property*	*Setting*
Textbox	Name Value	textbox1 HỌC VIỆN KỸ THUẬT QUÂN SỰ
Textbox	Name Value	textbox2 **HỆ HỌC VIÊN DÂN SỰ**
Textbox	Name Value	Textbox3 ------o0o-----
Textbox	Name Value	Textbox4 ="DANH SÁCH SINH VIÊN"
Table	Name Datasetname	Table1 DatasetDSSv
Textbox	Name Value	Textbox5 ="Tên lớp: " & Fields!TenLop.Value
Textbox	Name Value	Textbox6 **STT**
Textbox	Name Value	Textbox7 Mã sinh viên
Textbox	Name Value	Textbox8 Họ tên
Textbox	Name Value	Textbox9 Ngày sinh
Textbox	Name Value	Textbox10 `=RowNumber("table1_Group2")`
Textbox	Name Value	Textbox11 `=Fields!MaSV.Value`
Textbox	Name Value	Textbox12 `=Fields!Hoten.Value`
Textbox	Name Value	Textbox13 `=Fields!Ngaysinh.Value`


QL_DiemSV - Microsoft Visual Studio
File Edit View Project Build Debug Data Format Report Tools Window Community Help
Report.rdlc [Design]*
Toolbox
Page Header
HỌC VIỆN KỸ THUẬT QUÂN SỰ
HỆ HỌC VIÊN DÂN SỰ
------oOo-----
="DANH SÁCH SINH VIÊN"
Body
="Tên lớp: " & Fields!TenLop.Value
STT
Mã sinh viên
Họ tên
Ngày sinh
=Rov
=Fields!MaSV.Valı
=Fields!Hoten.Value
=Day(Fields!Ngay
Page Footer

**** Chèn group vào bảng.***

- Trong cửa sổ thiết kế Right-click vào Table tại dòng ta muốn chèn Group và chọn Insert Group.



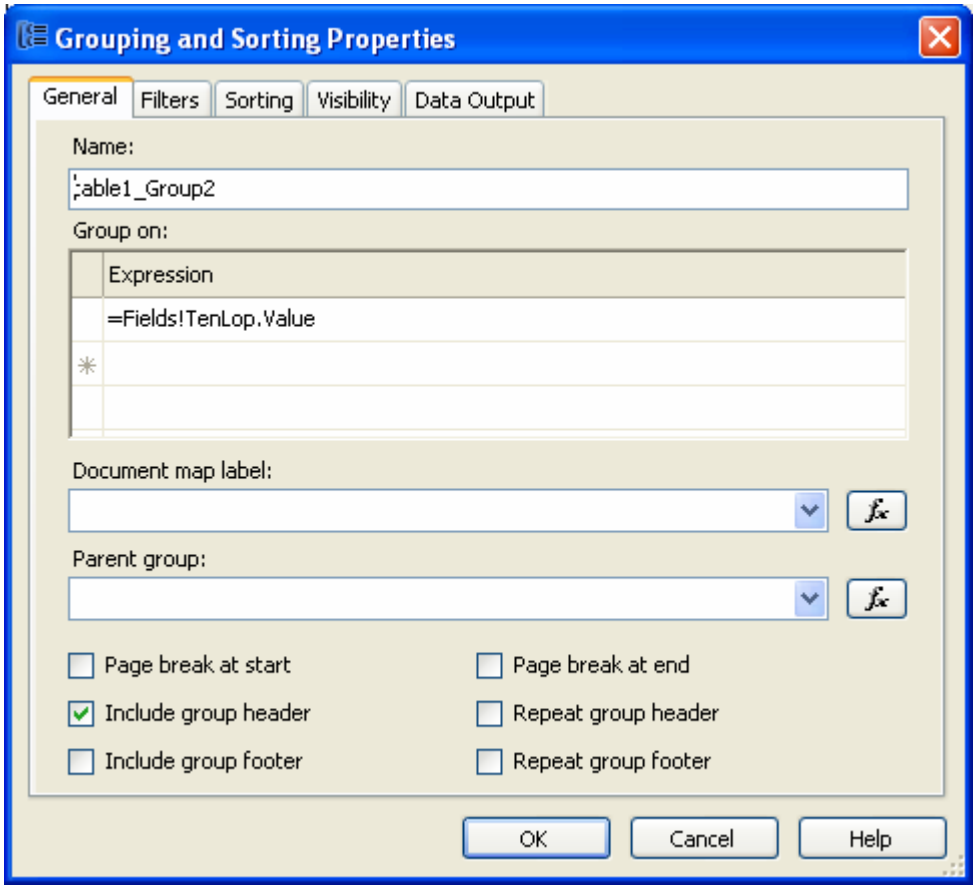
- Trên tab ***General*** ta điền các tham số:
  - + *Name:* Tên của group
  - + *Group on:* Chọn biểu thức mà dựa vào đó dữ liệu gom nhóm.
  - + Document map label: Đánh hoặc chọn biểu thức được sử dụng làm nhãn ánh xạ.
  - + *Parent group:* Nếu group này là nhóm phân cấp ngược (phân cấp đệ quy) ta sẽ đánh hoặc chọn biểu thức làm nhóm cha (recursive group parent).
  - + *Option chọn Page break at start* hoặc *Page break at end*: để thay thế cho một page break tại một vị trí bắt đầu hoặc kết thúc một thể hiện của nhóm.

+ *Include group header* hoặc *Include group footer:* Cho hiển thi hay không hiển thị Header và Footer của nhóm trên Table.

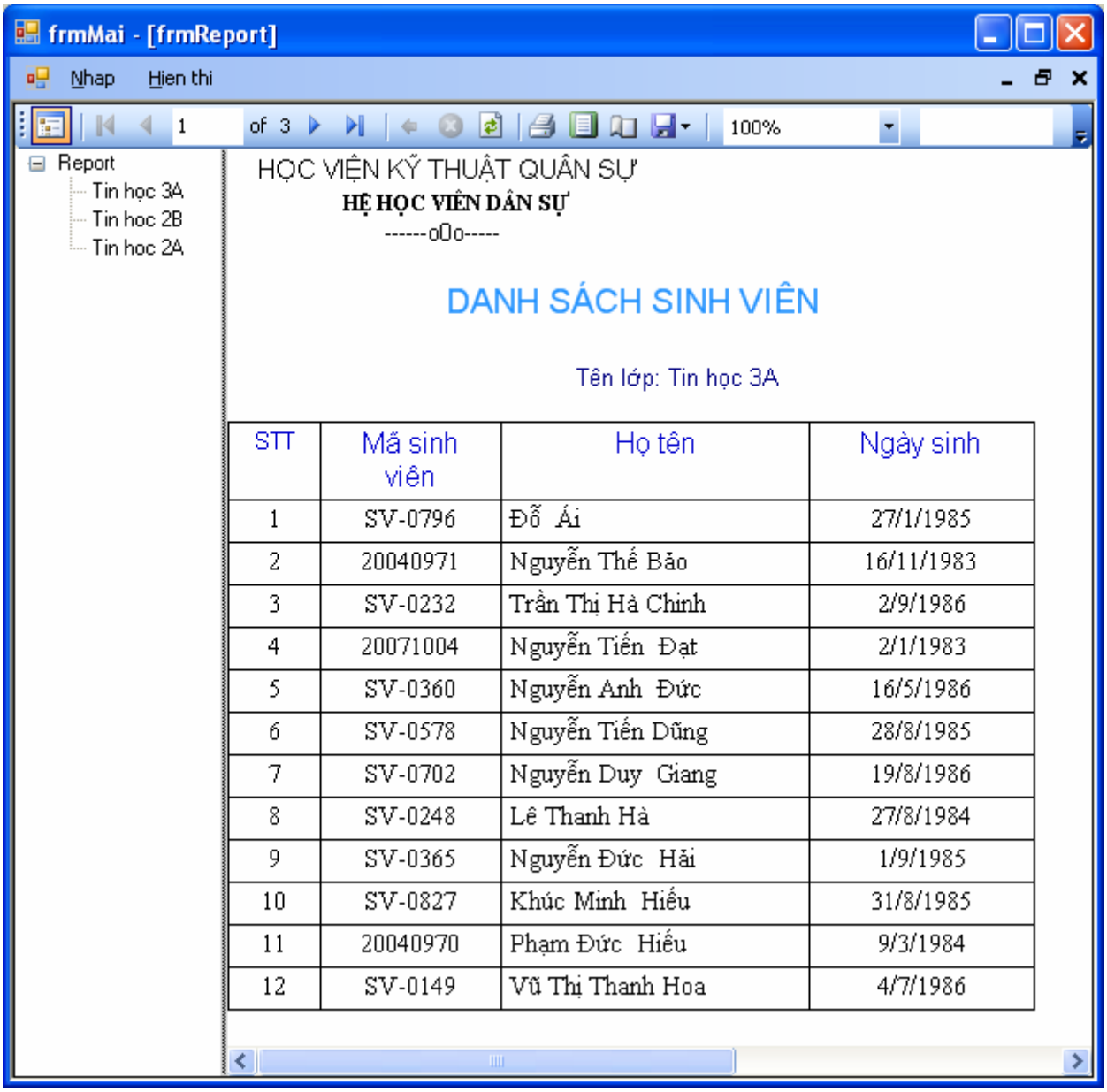
+ *Repeat group header hoặc Repeat group footer:* để lặp lại group header hoặc footer trên mỗi trang mà trong đó bảng xuất hiện.

- Trên tab ***Fillters:*** Chọn hoặc đánh biểu thức dùng để lọc dữ liệu trên nhóm.
- Trên tab ***Sorting***: Chọn hoặc đánh biểu thức dùng để sắp xếp dữ liệu trên nhóm.
- Trên tab ***Visibility***: chọn Visible.
- Trên tab Data Output: Chọn Yes.

Trong ví dụ ta chèn hai Group: table1_GroupTenLop và table1_Group2



5.3.5.2. Hiển thị dữ liệu.



STT	Mã sinh viên	Họ tên	Ngày sinh
1	SV-0796	Đỗ Ái	27/1/1985
2	20040971	Nguyễn Thế Bảo	16/11/1983
3	SV-0232	Trần Thị Hà Chinh	2/9/1986
4	20071004	Nguyễn Tiến Đạt	2/1/1983
5	SV-0360	Nguyễn Anh Đức	16/5/1986
6	SV-0578	Nguyễn Tiến Dũng	28/8/1985
7	SV-0702	Nguyễn Duy Giang	19/8/1986
8	SV-0248	Lê Thanh Hà	27/8/1984
9	SV-0365	Nguyễn Đức Hải	1/9/1985
10	SV-0827	Khúc Minh Hiếu	31/8/1985
11	20040970	Phạm Đức Hiếu	9/3/1984
12	SV-0149	Vũ Thị Thanh Hoa	4/7/1986

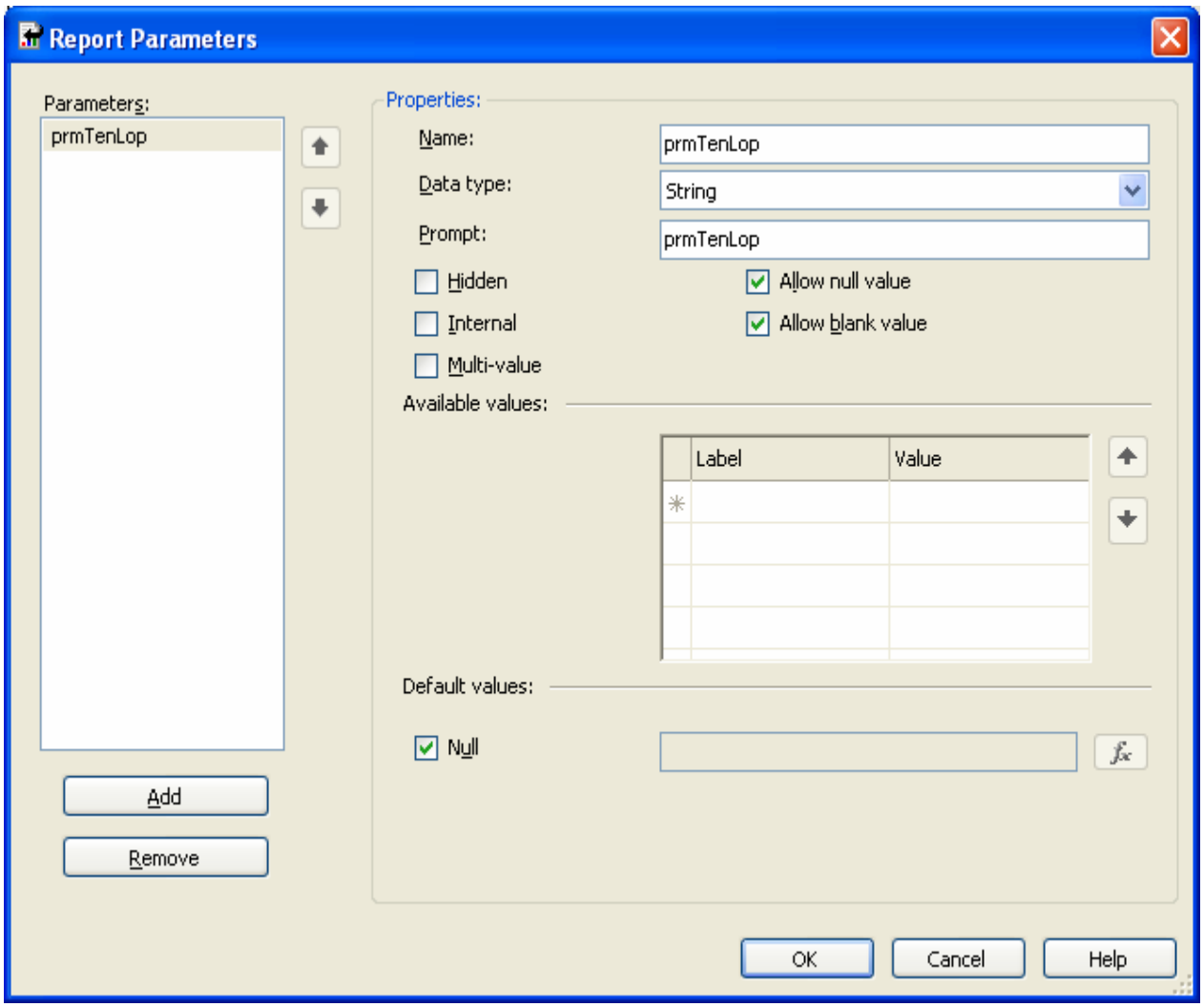
Để kết Report vào form, ta xây dựng form hiển thị. Đặt một điều khiển Report Viewer. Trên Report Viewer Tasks chọn các mục sau:

- Choose report: Chọn report ta vừa thiết kế (report.rdlc)
- Dock in parent container. Chọn.

5.2.5.3. Sử dụng tham số trong Report.

a) Tạo tham số

Trong cửa sổ thiết kế Report, để thiết lập tham số ta thực hiện như sau: Từ menu Report\Report Paramaters để Add các tham số cho Report.



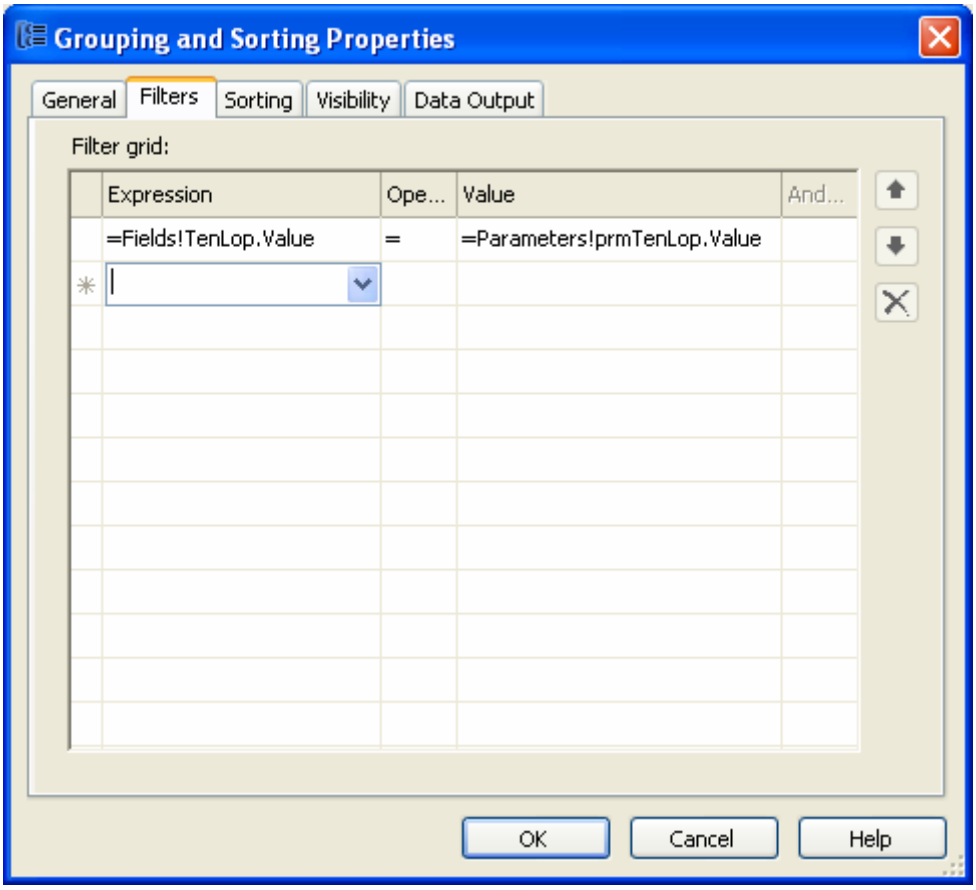
- Chọn nút Add để tạo thêm một tham số mới gồm các thông tin:
  - Name: Tên tham số
  - Data type: Kiểu dữ liệu của tham số
  - Prompt: Điền đoạn text sẽ xuất hiện sau *parameter text box* khi người sử dụng thực hiện chạy report.
  - Allow null value: Chọn khi cho phép tham số nhận giá trị null.
  - Allow blank value: Chọn khi cho phép thma số nhận giá trị blank.
  - Available values: Đưa ra một danh sách các giá trị sẵn có mà người sử dụng có thể lựa chọn.

- Lable: Chứa nhãn sẽ được sử dụng để hiển thị cho người sử dụng.
- Value: Là giá trị sẽ được sử dụng để chuyển qua Report sever cho tham số.

▪ Default values: Giá trị mặc định cho tham số.

- Chọn nút Remove để xóa tham số đã chọn.

Sau khi đã tạo tham số, ta có thể sử dụng tham số trong khi thiết kế Report. Trong ví dụ ta này ta xây dựng một report như trên, ngoài ra add thêm tham số prmTenLop và lọc dữ liệu theo tham số.



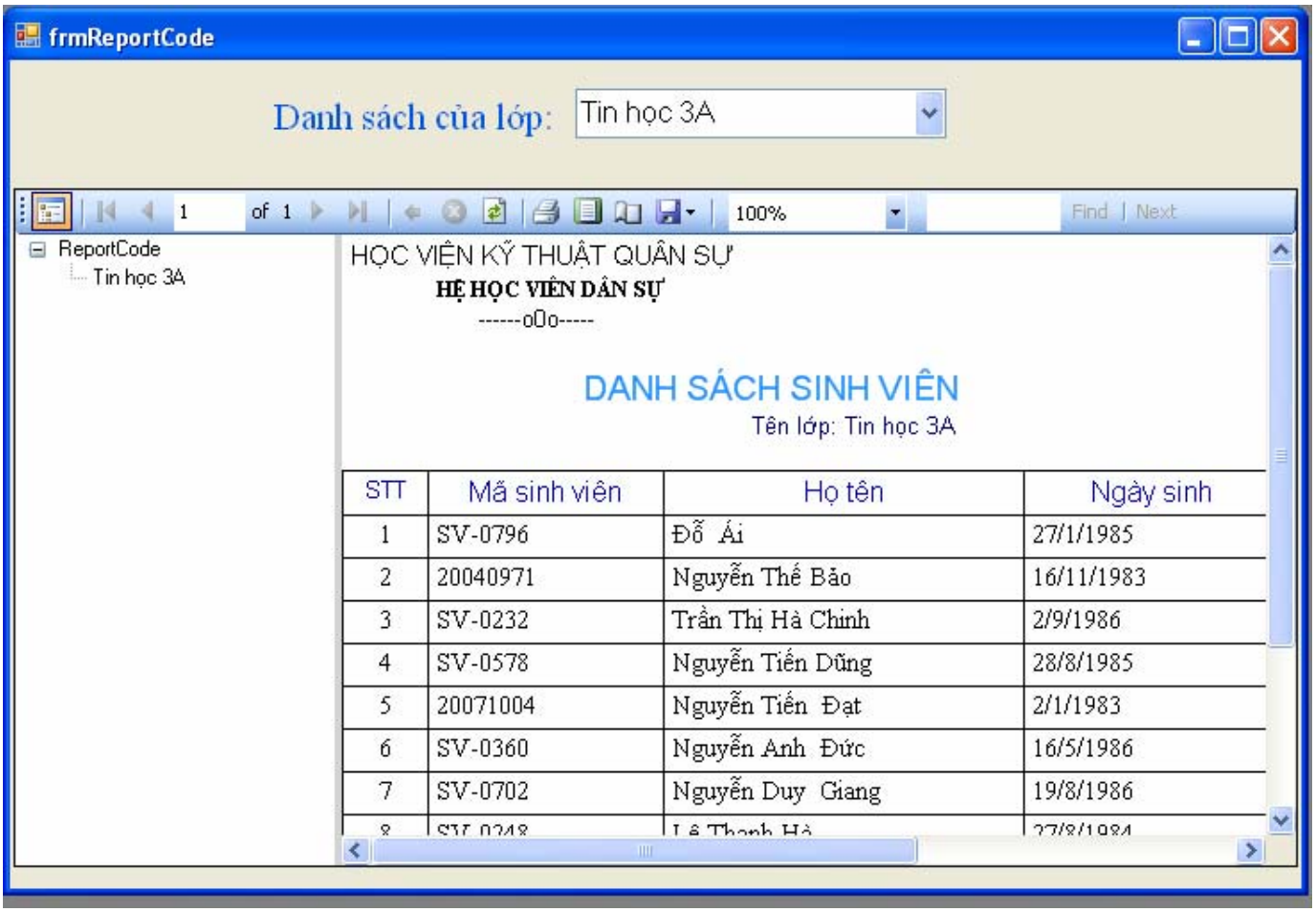
b)Truyền giá trị cho tham số:

Để truyền giá trị cho tham số ta sử dụng phương thức SetParameters của lớp Report trong tên miền Microsoft.Reporting.WinForms. Cụ thể ta sử dụng như sau:

```
' Gán giá trị cho tham số
```

```
Dim p As New ReportParameter("prmTenLop",
Me.cboMaLop.SelectedItem(1).ToString)
' Truyền giá trị vào report
Me.rvwDSReport.LocalReport.SetParameters(New
ReportParameter() {p})
```

Giả sử ta có form hiển thị Report như sau:



Có các đối tượng sau:

Object	***Property***	***Setting***
Form	Name Text	frmReportCode frmReportCode
Label	Name Text	lblDSL Danh sách của lớp
ComboBox	Name	cboMaLop
Report Viewer	Name Choose report	rvwDSL Chọn Report đã tạo như trên

1. Import các tên miền cần thiết.

```
' Import Data and SqlClient namespaces...
Imports System.Data
Imports System.Data.SqlClient
Imports Microsoft.Reporting.WinForms
```

2. Khai báo các kết nối

```
    Dim objConnection As New SqlConnection _
            ("server=THUHUONG;database=QLDiemSV;user
id=sa;password=12102006")
```

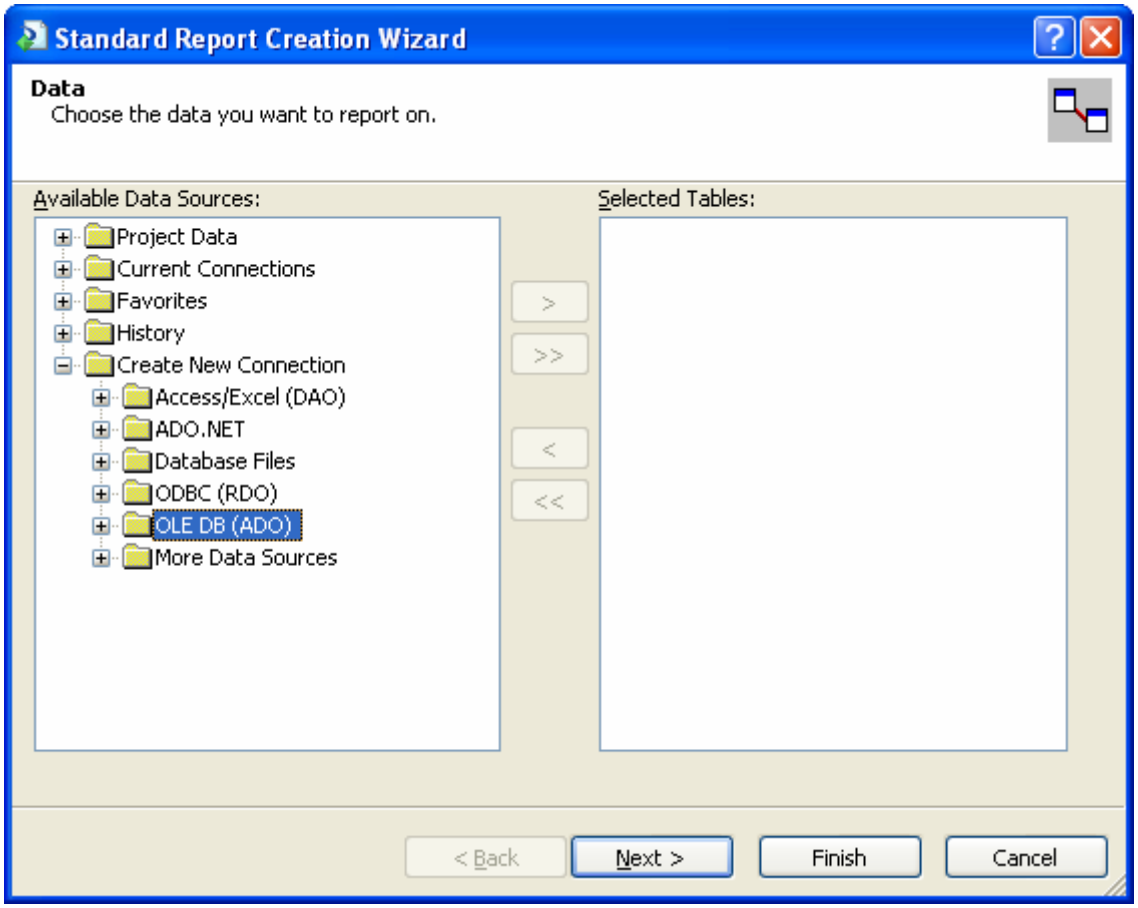
3. Code so sự kiện load form.

```
    Private Sub frmReportCode_Load(ByVal sender As System.Object, ByVal e
As System.EventArgs) Handles MyBase.Load

    ' Dien dư lieu vào cboLop
    Dim objDA_Lop As New SqlDataAdapter()
    Dim objDS_Lop As New DataSet()
    objDA_Lop.SelectCommand = New SqlCommand()
    objDA_Lop.SelectCommand.Connection = objConnection
    objDA_Lop.SelectCommand.CommandText = "SELECT MaLop, TenLop FROM Lop"
    objDA_Lop.SelectCommand.CommandType = CommandType.Text
    objConnection.Open()
    ' Fill the DataSet object with data...
    objDA_Lop.Fill(objDS_Lop, "DMLop")
    ' Close the database connection...
    objConnection.Close()

    Me.cboMaLop.DataSource = objDS_Lop.Tables("DMLop")
    Me.cboMaLop.DisplayMember = "TenLop"
    Me.cboMaLop.ValueMember = "MaLop"

     ' Set tham so cho Report
     Dim p As New ReportParameter("prmTenLop",
     Me.cboMaLop.SelectedItem(1).ToString)
      Me.rvwDSReport.LocalReport.SetParameters(New ReportParameter() {p})

     'This line of code loads data into the 'QLDiemSVDataSet.DanhSach'
     Me.objTableAdapterDSSV.Fill(Me.objDataSetDSSV.DanhSach)
        Me.rvwDSReport.RefreshReport()

    End Sub
```

4. Đồng bộ hóa dữ liệu trên form.

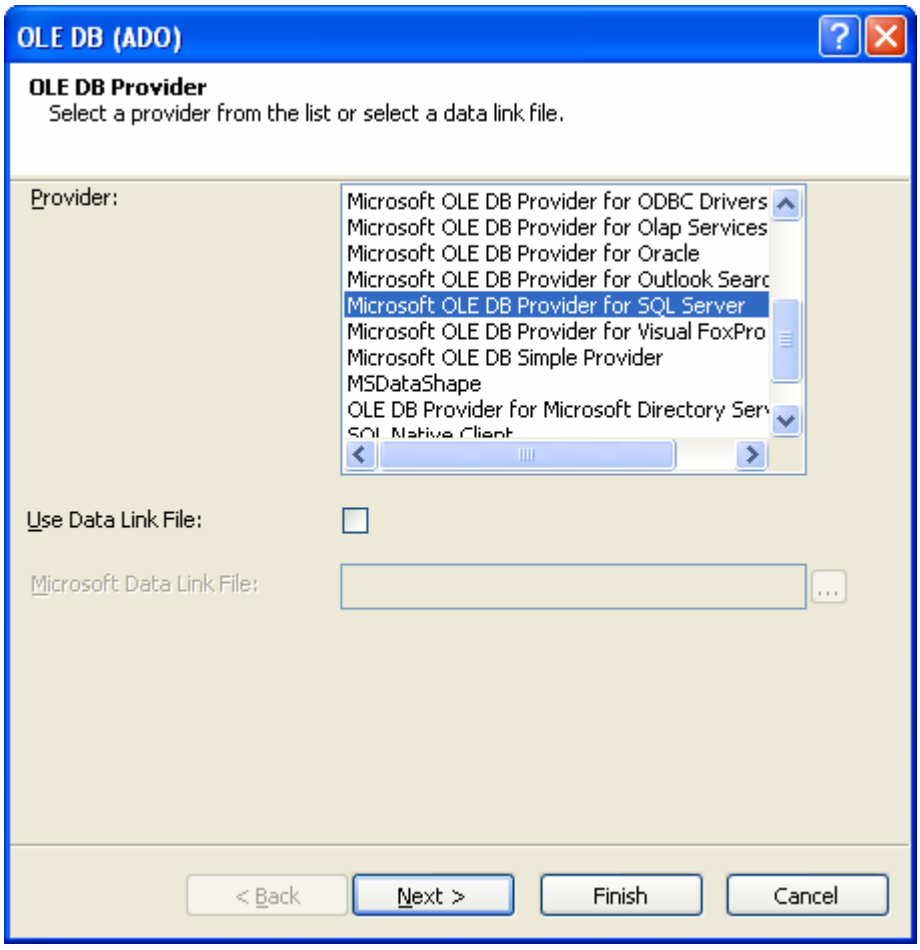
```
    Private Sub cboMaLop_SelectedIndexChanged(ByVal sender As Object,
ByVal e As System.EventArgs) Handles cboMaLop.SelectedIndexChanged
     Dim p As New ReportParameter("prmTenLop",
     Me.cboMaLop.SelectedItem(1).ToString)
     Me.rvwDSReport.LocalReport.SetParameters(New ReportParameter() {p})
     Me.rvwDSReport.RefreshReport()
    End Sub
```

Cửa sổ 2.



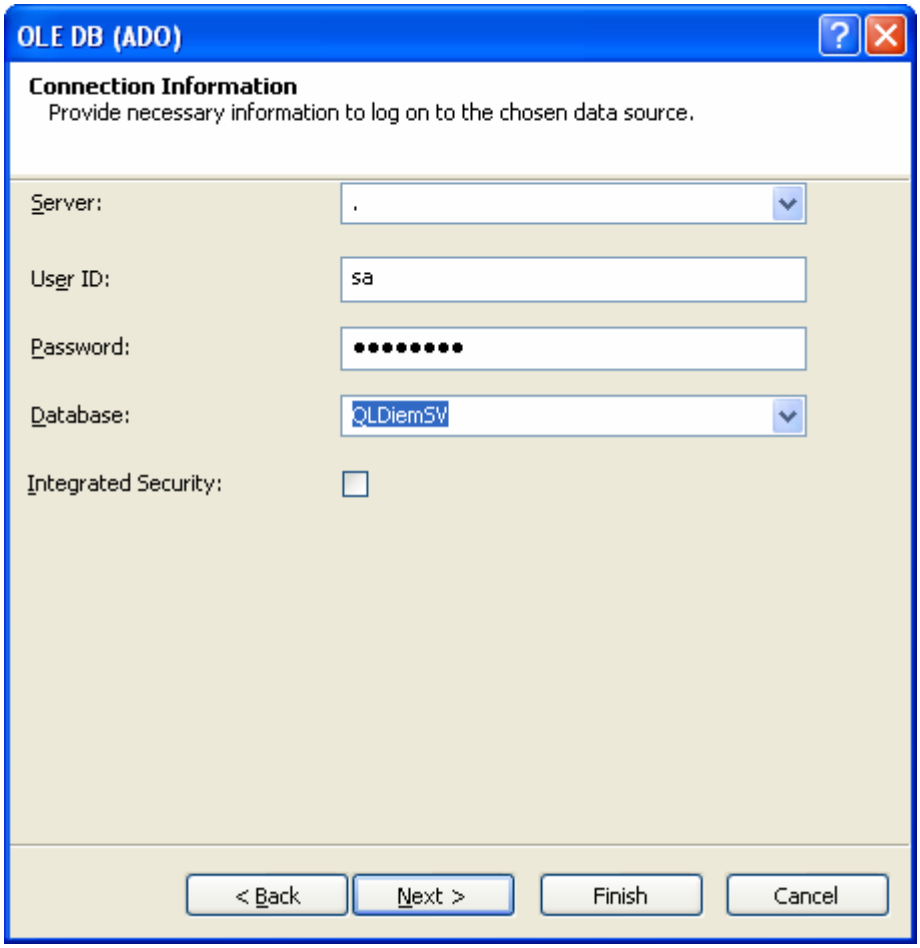
Chọn OLE DB (ADO) → Xuất hiện cửa sổ 3.

Cửa sổ 3:



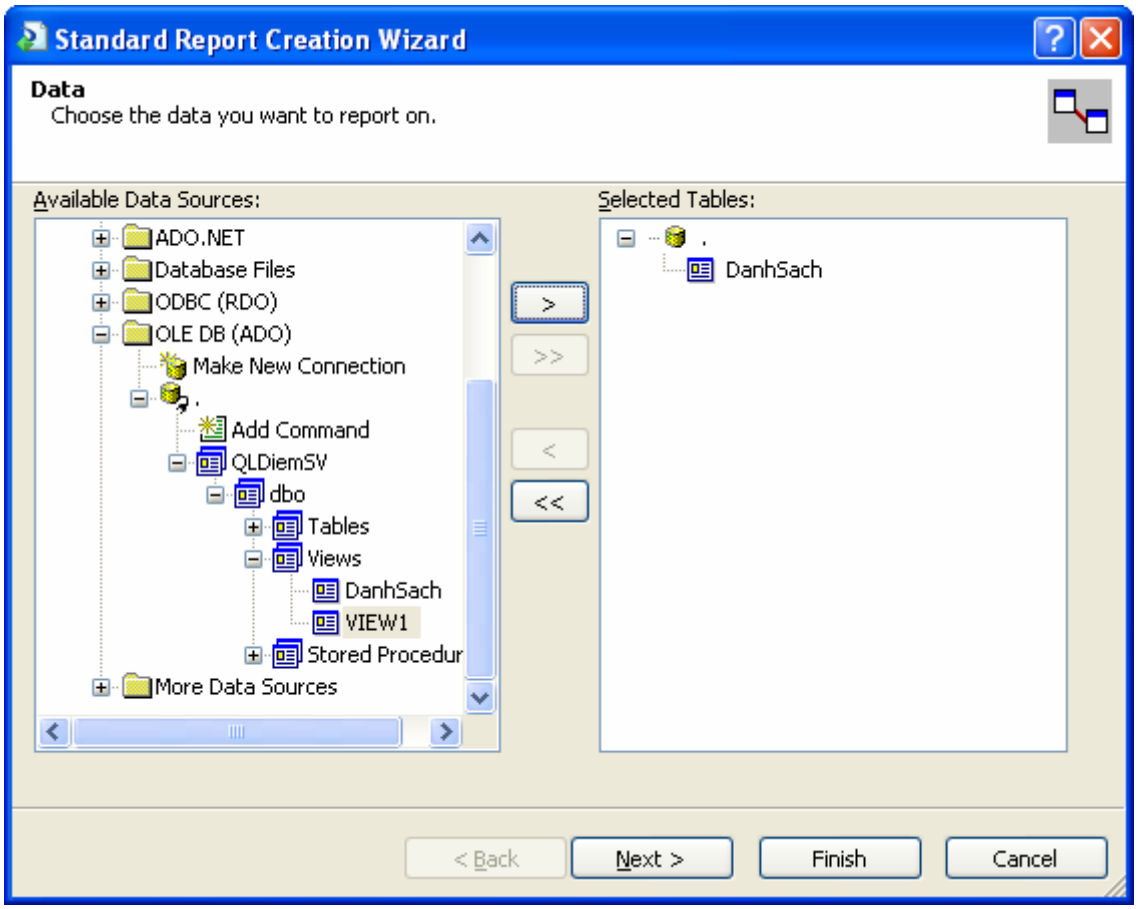
Chọn Microsoft OLE DB Provider for SQL Server. → Next.

Cửa sổ 4:

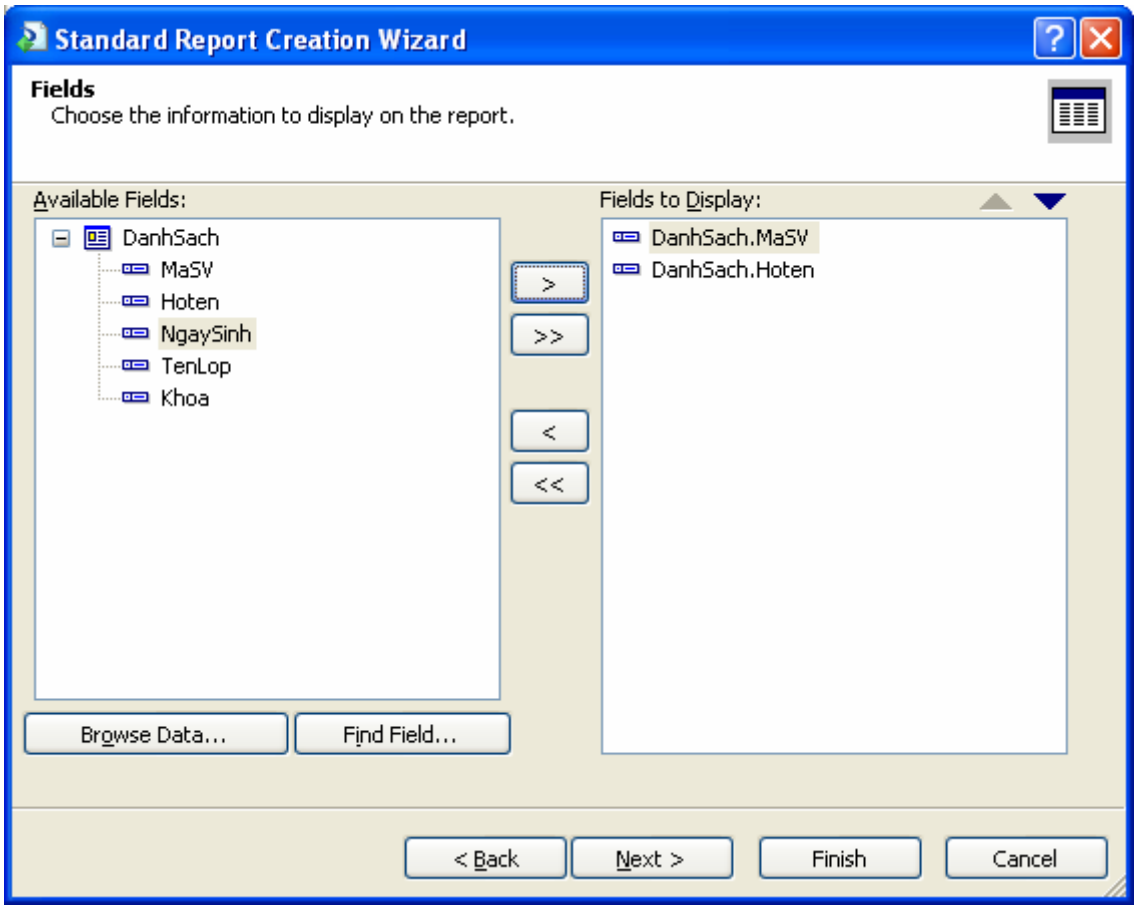


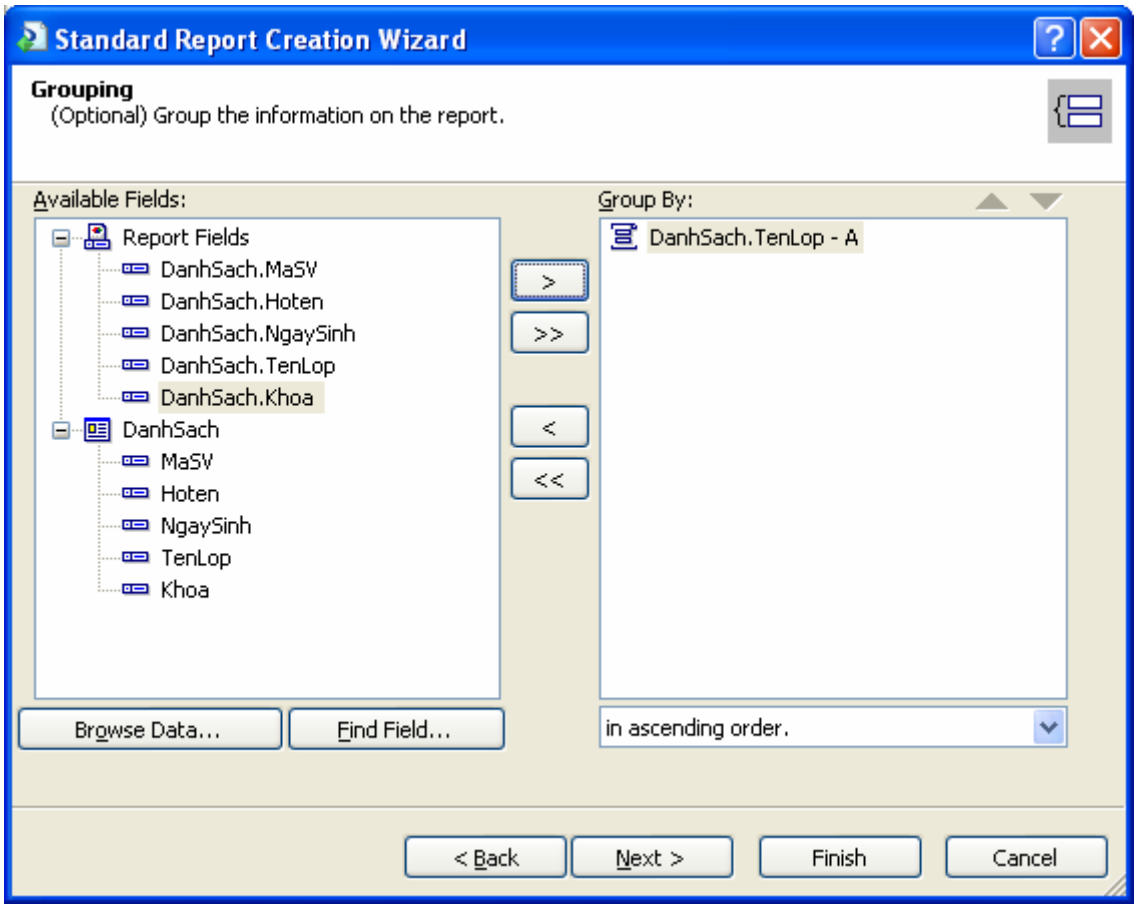
Chọn server, User, password và Database. →Finish. Để quay lại cửa sổ 2. Chọn dữ liệu sẽ được hiển thị trên report.

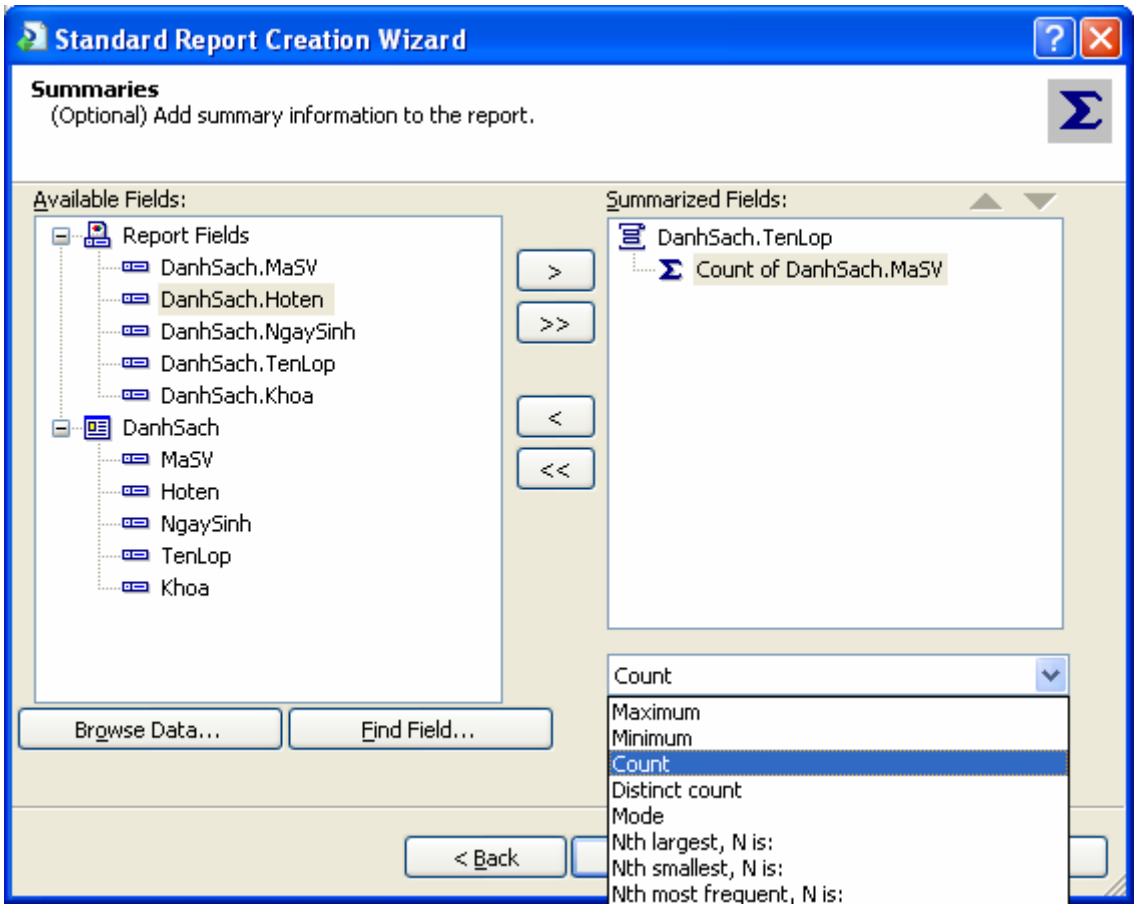
Cửa sổ 2: Chọn nguồn dữ liệu cho Report.

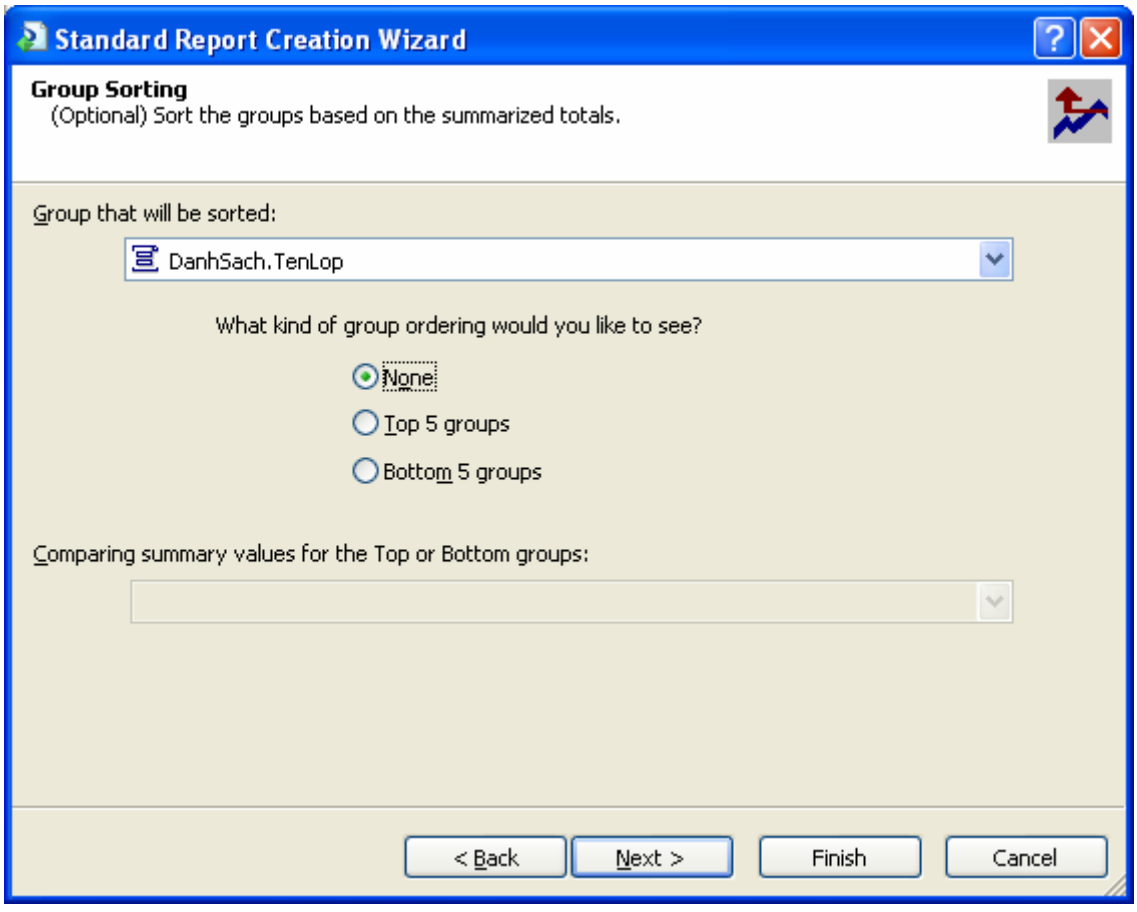


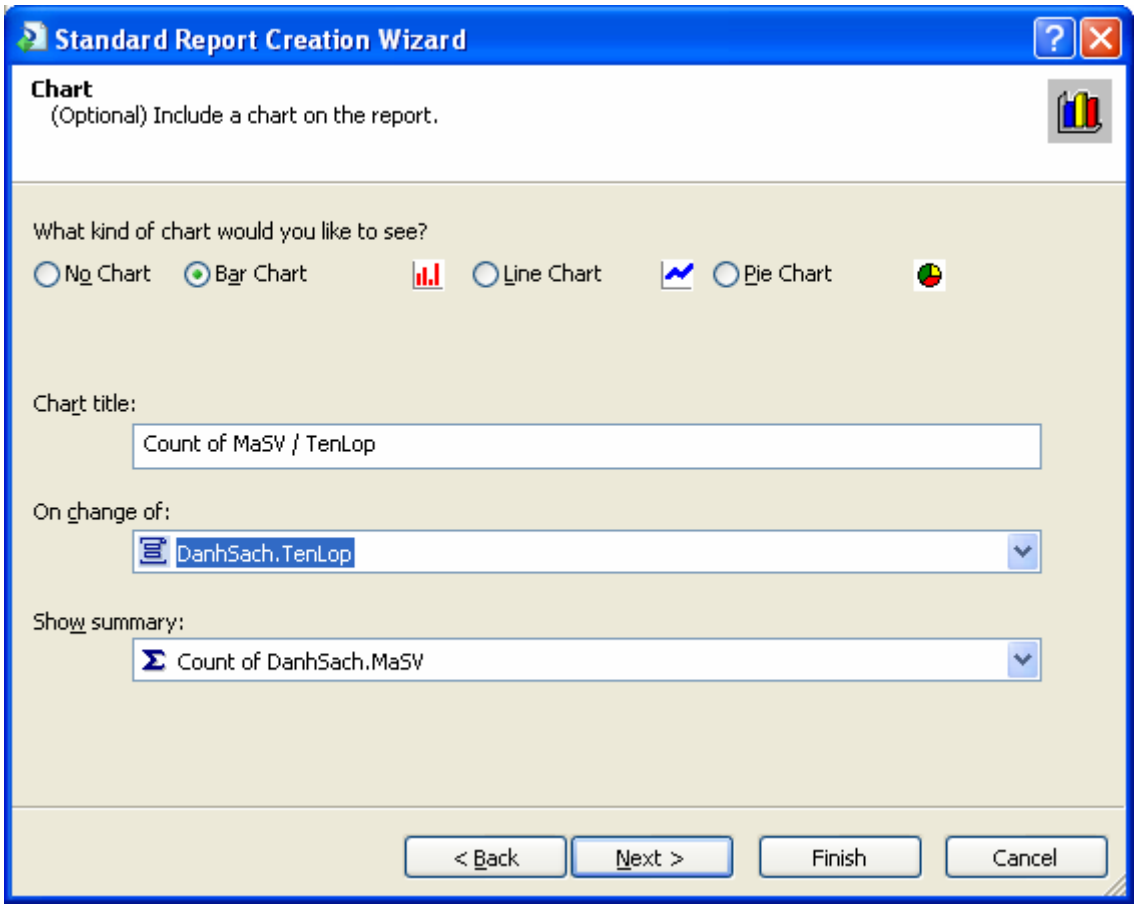
Chọn Next để tiếp tục theo sự chỉ dẫn của wizard.

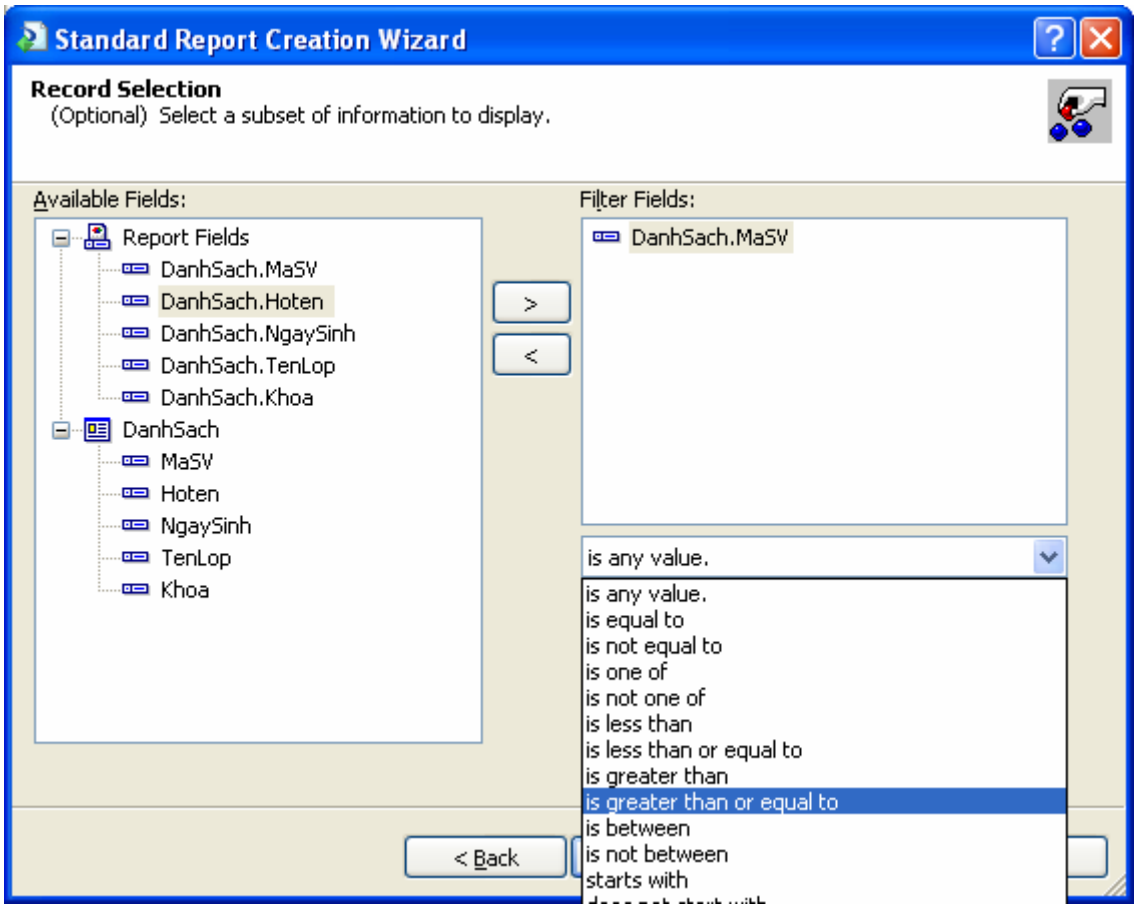

Standard Report Creation Wizard
Fields
Choose the information to display on the report.
Available Fields:
DanhSach
MaSV
Hoten
NgaySinh
TenLop
Khoa
Fields to Display:
DanhSach.MaSV
DanhSach.Hoten
>
>>
<
<<
Browse Data...
Find Field...
< Back
Next >
Finish
Cancel

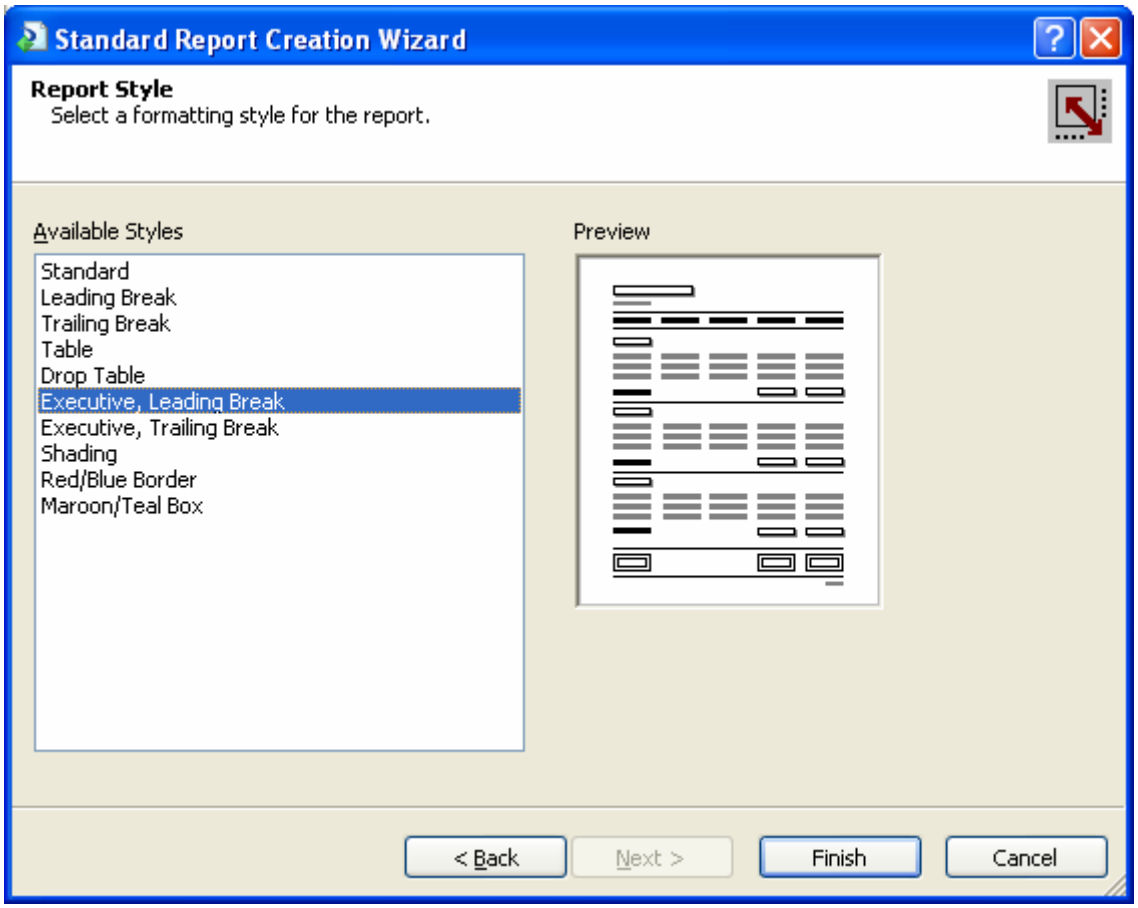

Standard Report Creation Wizard
Grouping
(Optional) Group the information on the report.
Available Fields:
Report Fields
DanhSach.MaSV
DanhSach.Hoten
DanhSach.NgaySinh
DanhSach.TenLop
DanhSach.Khoa
DanhSach
MaSV
Hoten
NgaySinh
TenLop
Khoa
Group By:
DanhSach.TenLop - A
>
>>
<
<<
Browse Data...
Find Field...
in ascending order.
< Back
Next >
Finish
Cancel

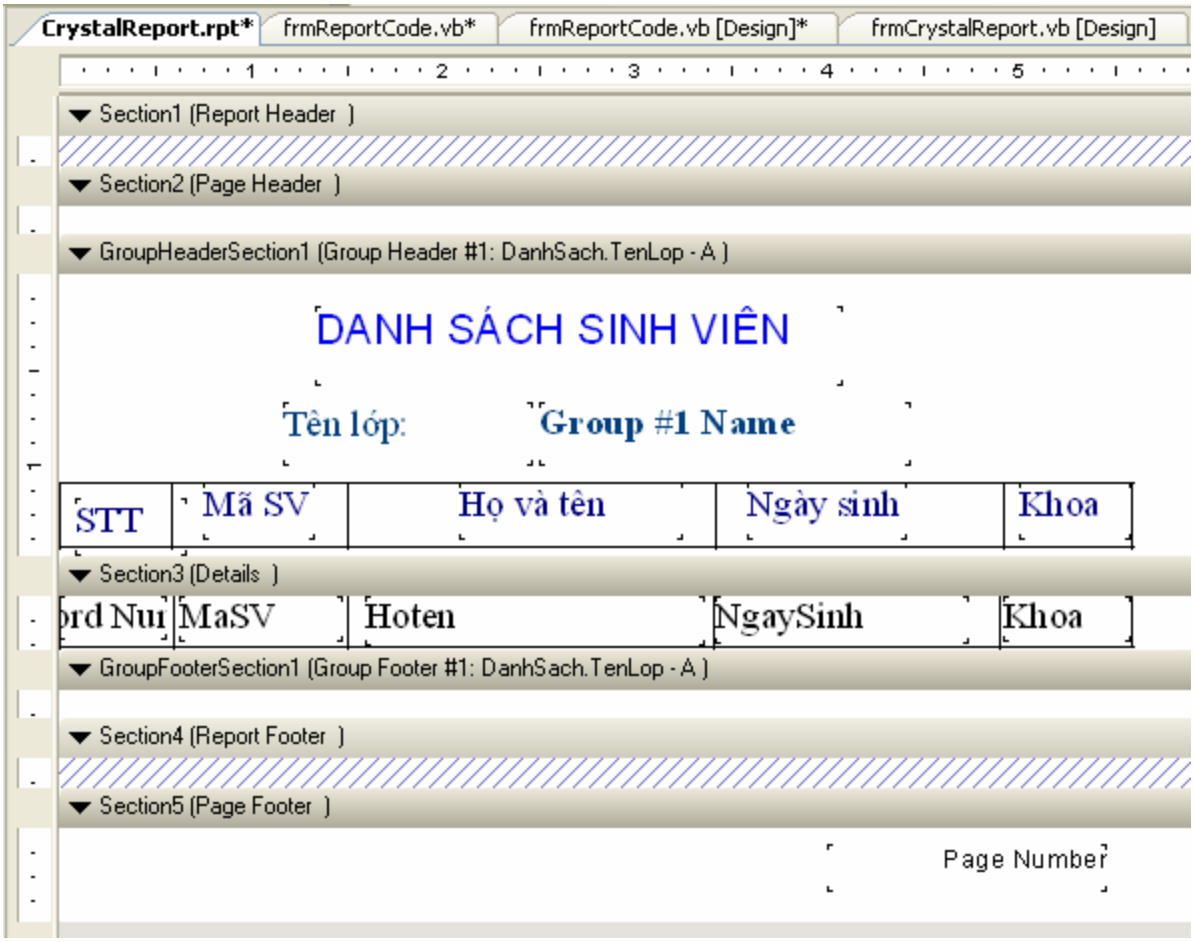

Standard Report Creation Wizard
Summaries
(Optional) Add summary information to the report.
Available Fields:
Report Fields
DanhSach.MaSV
DanhSach.Hoten
DanhSach.NgaySinh
DanhSach.TenLop
DanhSach.Khoa
DanhSach
MaSV
Hoten
NgaySinh
TenLop
Khoa
>
>>
<
<<
Summarized Fields:
DanhSach.TenLop
Count of DanhSach.MaSV
Count
Maximum
Minimum
Count
Distinct count
Mode
Nth largest, N is:
Nth smallest, N is:
Nth most frequent, N is:
Browse Data...
Find Field...
< Back


Standard Report Creation Wizard
Group Sorting
(Optional) Sort the groups based on the summarized totals.
Group that will be sorted:
DanhSach.TenLop
What kind of group ordering would you like to see?
None
Top 5 groups
Bottom 5 groups
Comparing summary values for the Top or Bottom groups:
< Back
Next >
Finish
Cancel


Standard Report Creation Wizard
Chart
(Optional) Include a chart on the report.
What kind of chart would you like to see?
No Chart
Bar Chart
Line Chart
Pie Chart
Chart title:
Count of MaSV / TenLop
On change of:
DanhSach.TenLop
Show summary:
Count of DanhSach.MaSV
< Back
Next >
Finish
Cancel

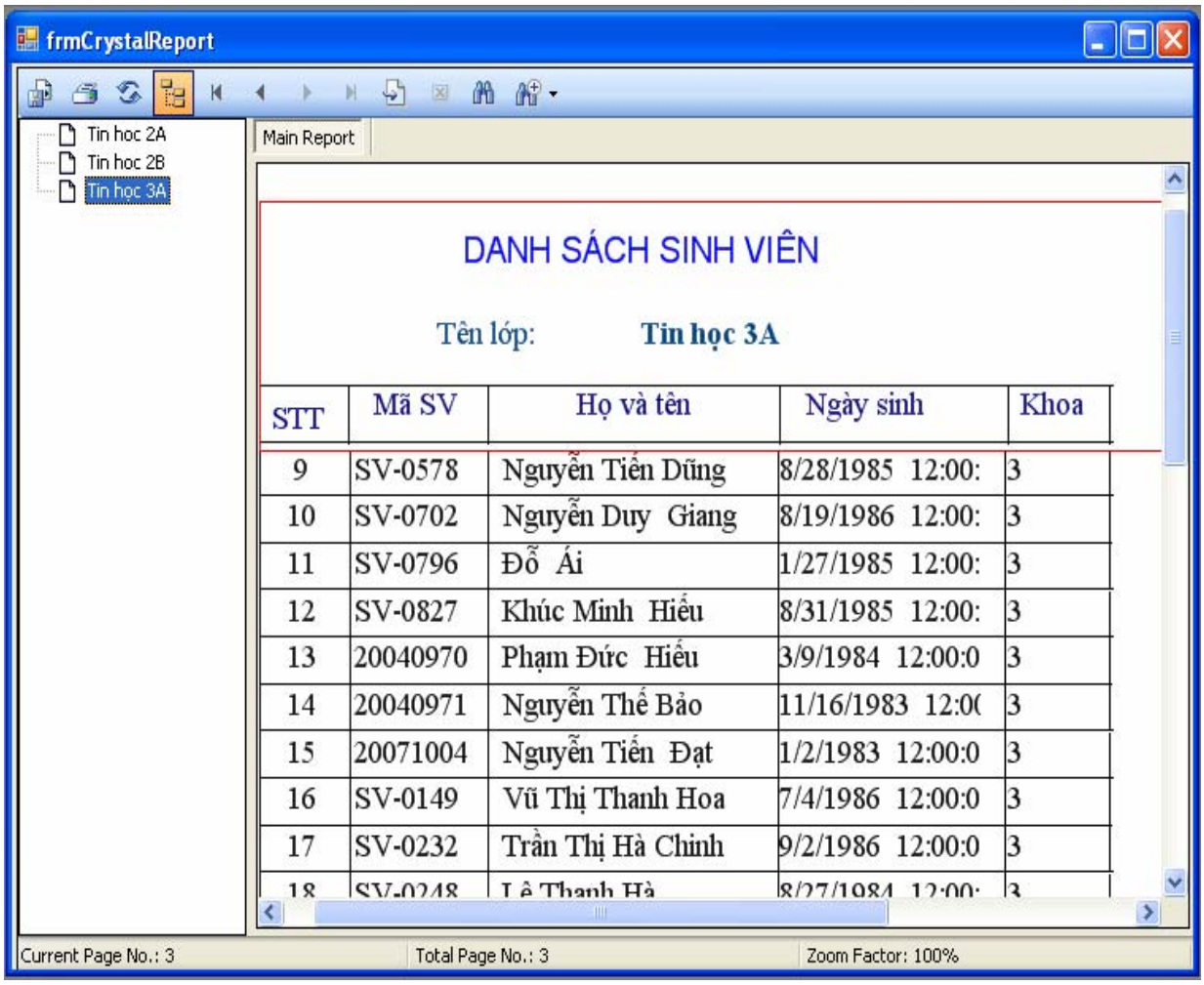

Standard Report Creation Wizard
Record Selection
(Optional) Select a subset of information to display.
Available Fields:
Report Fields
DanhSach.MaSV
DanhSach.Hoten
DanhSach.NgaySinh
DanhSach.TenLop
DanhSach.Khoa
DanhSach
MaSV
Hoten
NgaySinh
TenLop
Khoa
Filter Fields:
DanhSach.MaSV
is any value.
is any value.
is equal to
is not equal to
is one of
is not one of
is less than
is less than or equal to
is greater than
is greater than or equal to
is between
is not between
starts with
< Back


Standard Report Creation Wizard
Report Style
Select a formatting style for the report.
Available Styles
Standard
Leading Break
Trailing Break
Table
Drop Table
Executive, Leading Break
Executive, Trailing Break
Shading
Red/Blue Border
Maroon/Teal Box
Preview
< Back
Next >
Finish
Cancel


CrystalReport.rpt*
frmReportCode.vb*
frmReportCode.vb [Design]*
frmCrystalReport.vb [Design]
Section1 (Report Header)
Section2 (Page Header)
GroupHeaderSection1 (Group Header #1: DanhSach.TenLop - A)
DANH SÁCH SINH VIÊN
Tên lớp:
Group #1 Name
STT
Mã SV
Họ và tên
Ngày sinh
Khoa
Section3 (Details)
MaSV
Hoten
NgaySinh
Khoa
GroupFooterSection1 (Group Footer #1: DanhSach.TenLop - A)
Section4 (Report Footer)
Section5 (Page Footer)
Page Number

Sau khi đã xây dựng được file *.rpt. Ta xây dựng form để hiển thị report trên. Thiết kế form như hình gồm một đối tượng Crystal Report Viewer. Trên Crystal Report Viewer tasks thực hiện các mục:

- Choose a crystal report: Chọn Report vừa tạo trên.
- Dock in parent container: Chọn.



TRƯỜNG ĐẠI HỌC HÀNG HẢI
KHOA CÔNG NGHỆ THÔNG TIN

BÀI GIẢNG
HỆ QUẢN TRỊ CƠ SỞ DỮ LIỆU

Biên soạn: K.S Nguyễn Vương Thịnh

Hải Phòng – 2008

BỘ GIAO THÔNG VẬN TẢI
TRƯỜNG ĐẠI HỌC HÀNG HẢI
KHOA CÔNG NGHỆ THÔNG TIN
BỘ MÔN HỆ THỐNG THÔNG TIN

BÀI GIẢNG

HỆ QUẢN TRỊ CƠ SỞ DỮ LIỆU

TÊN HỌC PHẦN : **CƠ SỞ DỮ LIỆU**
MÃ HỌC PHẦN : **17402**
TRÌNH ĐỘ ĐÀO TẠO : **ĐẠI HỌC CHÍNH QUY**
DÙNG CHO SV NGÀNH : **CÔNG NGHỆ THÔNG TIN**

HẢI PHÒNG - 2008

Tên học phần: Hệ quản trị Cơ sở dữ liệu **Loại học phần**: 4
Bộ môn phụ trách giảng dạy: Hệ thống Thông tin **Khoa phụ trách**: CNTT.
Mã học phần: 17402 **Tổng số TC**: 4

TS tiết	Lý thuyết	Thực hành/ Xemina	Tự học	Bài tập lớn	Đồ án môn học
90	45	45	0	x	0

Điều kiện tiên quyết:

Không yêu cầu.

Mục tiêu của học phần:

Cung cấp cho sinh viên những khái niệm cơ bản về hệ quản trị cơ sở dữ liệu, vai trò và chức năng của hệ quản trị cơ sở dữ liệu cũng như quá trình tương tác giữa hệ quản trị cơ sở dữ liệu với các phần mềm ứng dụng cơ sở dữ liệu. Giúp sinh viên tiếp cận với nền tảng hệ quản trị cơ sở dữ liệu Microsoft SQL Server.

Nội dung chủ yếu:

Khái niệm về CSDL và hệ quản trị CSDL; Các thành phần cơ bản trong MS SQL Server; Ngôn ngữ T-SQL và các đối tượng CSDL; Các tác vụ quản trị hệ thống.

Nội dung chi tiết:

TÊN CHƯƠNG MỤC	PHÂN PHỐI SỐ TIẾT				
	TS	**LT**	**BT**	**TH**	**KT**
Chương 1. Giới thiệu	**6**				
1.1. Một số khái niệm cơ bản					
1.2. Giới thiệu về SQL Server và mô hình Client/Server					
1.3. Cài đặt và cấu hình SQL Server					
1.4. Các thành phần cơ bản trong SQL Server					
Chương 2. Làm việc với CSDL và bảng	**9**				**1**
2.1. Làm việc với CSDL					
2.1.1. Tổ chức tập tin trong CSDL SQL Server					
2.1.2. Định nghĩa CSDL					
2.1.3. Sửa đổi CSDL					
2.1.4. Xóa CSDL					
2.2. Làm việc với bảng dữ liệu					
2.2.1. Các kiểu dữ liệu cơ bản					
2.2.2. Ràng buộc (Constraint) và thuộc tính cột					
2.2.3. Định nghĩa bảng dữ liệu					
2.2.3. Sửa đổi bảng dữ liệu					
2.2.4. Xóa bảng dữ liệu					
2.3. Làm việc với các bản ghi					
2.3.1. Thêm bản ghi mới (Insert)					
2.3.2. Cập nhật bản ghi (Update)					
2.3.3. Xóa bản ghi (Delete)					
2.4. Khái niệm về chỉ mục (Index)					
Chương 3. Truy vấn dữ liệu và bảng ảo (View)	**9**				**1**
3.1. Cấu trúc truy vấn cơ bản					
3.1.1. Câu lệnh SELECT					
3.1.2. Mệnh đề WHERE và biểu thức điều kiện					
3.1.3. Một số hàm thường dùng					
3.1.4. Truy vấn thống kê với GROUP BY và HAVING					
3.1.5. Sắp xếp với ORDER BY					
3.2. Truy vấn dữ liệu từ nhiều bảng					
3.2.1. Inner Join					
3.2.2. Left Outer Join					

TÊN CHƯƠNG MỤC	PHÂN PHỐI SỐ TIẾT				
	TS	**LT**	**BT**	**TH**	**KT**
3.2.3. Right Outer Join					
3.2.4. Full Outer Join					
3.2.5. Cross Join					
3.2.6. Self Join					
3.2.7. Non - Equal Join					
3.2.8. Union					
3.3. Truy vấn lồng nhau (Subquery)					
3.3.1. Nested Scalar Subquery					
3.3.2. Correlated Subquery					
3.4. Bảng ảo (View)					
3.4.1. Định nghĩa View					
3.4.2. Sửa đổi View					
3.4.3. Xóa View					
Chương 4. Thủ tục thường trú và hàm người dùng	**9**				**1**
4.1. Lập trình với T - SQL					
4.2. Thủ tục thường trú (Stored Procedure)					
4.3. Hàm người dùng định nghĩa (User Defined Fuction)					
Chương 5. Trigger và Transaction	**6**				
5.1. Trigger và ứng dụng của Trigger					
5.1.1. After Trigger					
5.1.2. Instead Of Trigger					
5.2. Các loại Transaction và ứng dụng					
5.2.1. Transaction tự động					
5.2.2. Transaction tường minh					
5.2.3. Transaction ẩn					
5.2.4. Transaction lồng nhau					
Chương 6: Các tác vụ quản trị hệ thống	**6**				
6.1. Quản lý đăng ký SQL Server (Server Registration)					
6.2. Bảo mật và phân quyền người dùng					
6.3. Sao lưu (Backup) và phục hồi (Restore) dữ liệu					
6.4. Hoạch định tác vụ quản trị					

Nhiệm vụ của sinh viên:

Tham dự các buổi học lý thuyết và thực hành, làm các bài tập được giao, làm các bài thi giữa học phần và bài thi kết thúc học phần theo đúng quy định.

Tài liệu học tập:

1. Dương Quang Thiện, *SQL Server 2000: Lập trình T - SQL*, NXB Văn hóa Sài Gòn, 2007.
2. Paul Turley & Dan Wood, *Beginning Transact-SQL with SQL Server 2000 and 2005***,** Wrox Press, 2006.
3. Brian Knightet al, *Professional SQL Server 2005 Administration*, Wrox Press, 2007.
4. Ray Rankins, Paul Bertucci, Chris Gallelli, Alex T.Silverstein, *Microsoft SQL Server 2005 Unleashed*, Sams Publishing, 2007.

Hình thức và tiêu chuẩn đánh giá sinh viên:

- Hình thức thi: thi viết hoặc vấn đáp.
- Tiêu chuẩn đánh giá sinh viên: căn cứ vào sự tham gia học tập của sinh viên trong các buổi học lý thuyết và thực hành, kết quả làm các bài tập được giao, kết quả của các bài thi giữa học phần và bài thi kết thúc học phần.

Thang điểm: Thang điểm chữ A, B, C, D, F.

Điểm đánh giá học phần: $Z = 0,4X + 0,6Y$.

Bài giảng này là tài liệu **chính thức và thống nhất** của Bộ môn Hệ thống Thông tin, Khoa Công nghệ Thông tin và được dùng để giảng dạy cho sinh viên.

Trưởng Bộ môn **Ngày phê duyệt: / / .**

MỤC LỤC

Phần 1. QUẢN TRỊ SQL SERVER
BẮT ĐẦU VỚI SQL SERVER

TÌM HIỂU VỀ HỆ QUẢN TRỊ CSDL SQL SERVER

Giới thiệu SQL Server.

SQL Server là hệ thống quản trị cơ sở dữ liệu quan hệ (Relational DataBase Management System- RDBMS) sử dụng các lệnh giáo chuyển Transaction-SQL để trao đổi dữ liệu giữa Client Computer và Server Computer.

SQL Server có một số đặc tính sau:

- Cho phép quản trị một hệ CSDL lớn (lên đến vài tega byte), có tốc độ xử lý dữ liệu nhanh đáp ứng yêu cầu về thời gian.
- Cho phép nhiều người cùng khai thác trong một thời điểm đối với một CSDL và toàn bộ quản trị CSDL (lên đến vài chục ngàn user).
- Có hệ thống phân quyền bảo mật tương thích với hệ thống bảo mật của công nghệ NT (Network Technology), tích hợp với hệ thống bảo mật của Windows NT hoặc sử dụng hệ thống bảo vệ độc lập của SQL Server.
- Hỗ trợ trong việc triển khai CSDL phân tán và phát triển ứng dụng trên Internet
- Cho phép lập trình kết nối với nhiều ngôn ngữ lập trình khác dùng xây dựng các ứng dụng đặc thù (Visual Basic, C, C++, ASP, ASP.NET, XML,...).
- Sử dụng câu lệnh truy vấn dữ liệu Transaction-SQL (Access là SQL, Oracle là PL/SQL).

Các ấn bản của SQL Server.

SQL Server có các ấn bản chính sau:

- Enterpise Manager: Là ấn bản đầy đủ của SQL Server có thể chạy trên 32CPU và 64GB RAM. Có các dịch vụ phân tích dữ liệu Analysis Service.
- Standard: Giống như Enterprise nhưng bị hạn chế một số tính năng cao cấp, có thể chạy trên 2CPU, 4GB RAM.

- Personal: Phiên bản này chủ yếu để chạy trên PC, nên có thể chạy trên các hệ điều hành Windows 9x, Windows XP, Windows 2000, Windows 2003…
- Developer: Là phiên bản tương tự như Enterprise nhưng bị giới hạn bởi số user kết nối đến.
- Desktop Engine: Là phiên bản một engine chỉ chạy trên desktop và không có giao diện người dùng (GUI), kích thước CSDL giới hạn bởi 2GB.
- Win CE: Sử dụng cho các ứng dụng chạy trên Windows CE.
- Trial: Phiên bản dùng thử, bị giới hạn bởi thời gian.
- SQL Client: Là phiên bản dành cho máy khách, khi thực hiện khai thác sẽ thực hiện kết nối đến phiên bản SQL Server, phiên bản này cung cấp giao diện GUI khai thác cho người sử dụng.
- SQL Connectivity only: Là phiên bản sử dụng chỉ cho các ứng dụng để kết nối đến SQL Server, phiên bản này không cung cấp công cụ GUI cho người dùng khai thác SQL Server.

Các phiên bản này được cài đặt phụ thộc vào bộ cài đặt mà bạn chọn hoặc lựa chọn khai cài đặt (ví dụ phiên bản Enterprise, Standard, Personal,… bạn phải chọn theo bộ cài đặt, phiên bản SQL Client, Connectivity,… do bạn chọn trong các hộp thoại trong quá trình cài đặt).

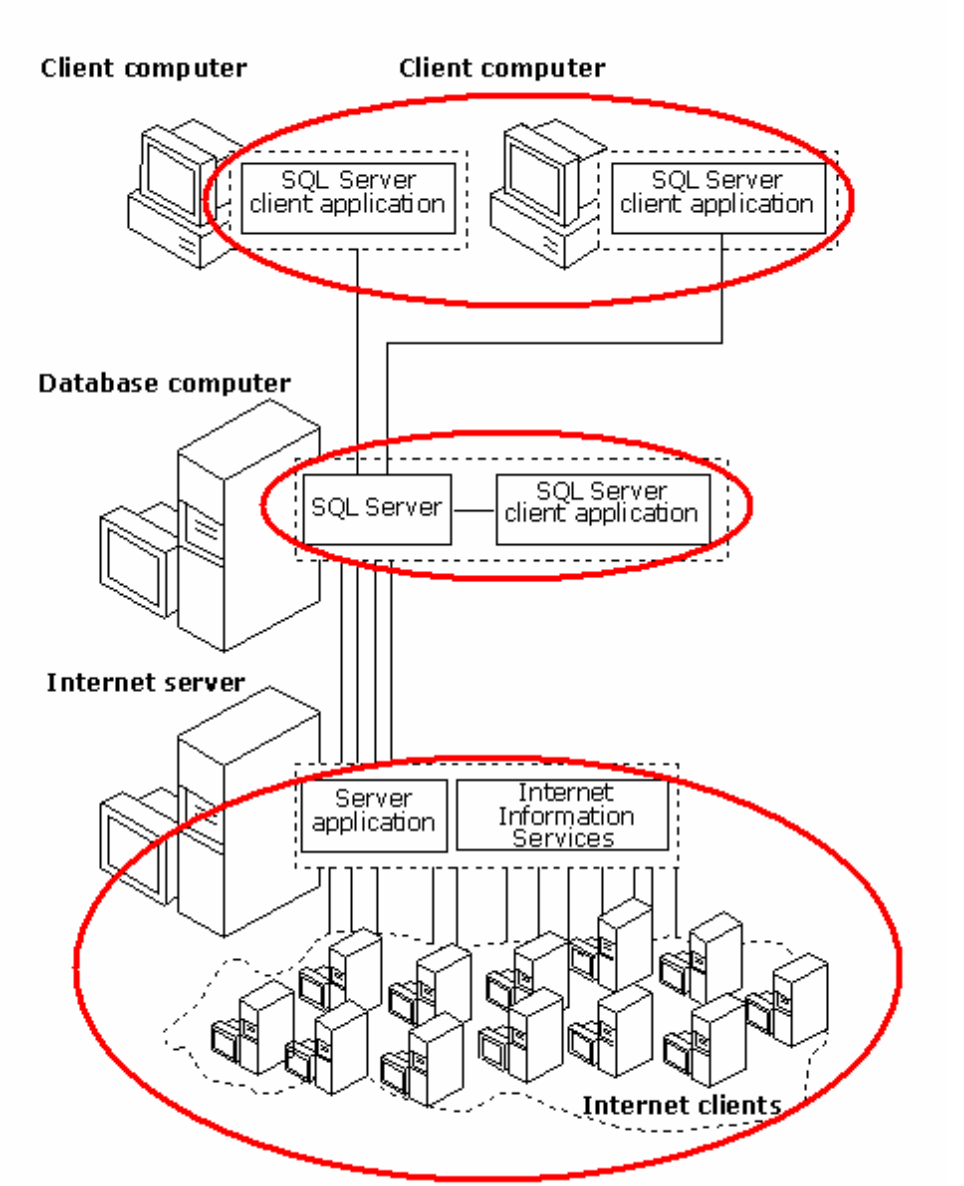
Một số tính năng của Enterprise manager.

- Dễ cài đặt
- Hỗ trợ mô hình Client/Server.
- Thích hợp trên các hệ điều hành Windows.
- Hoạt động với nhiều giao thức truyền thông.
- Hỗ trợ dịch vụ Data Warehousing.
- Thích hợp với chuẩn ANSI/ISO SQL-92.
- Hỗ trợ nhân bản dữ liệu.
- Cung cấp dịch vụ tìm kiếm Full-Text.
- Sách trợ giúp- Book Online.

MÔ HÌNH HOẠT ĐỘNG CỦA SQL SERVER TRÊN MẠNG MÁY TÍNH.

Mô hình chung SQL Server trên mạng.

SQL Server là hệ quản trị CSDL hoạt động trên mạng, có thể thực hiện trao đổi dữ liệu theo nhiều mô hình mạng khác nhau, nhiều giao thức và phương thức truyền tin khác nhau.



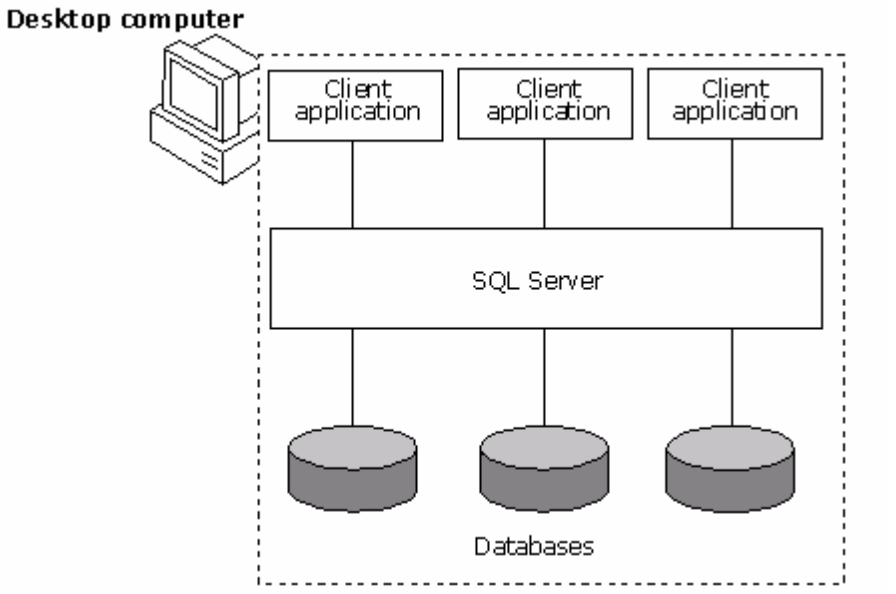
Trong sơ đồ trên thể hiện ba kiểu kết nối ứng dụng đến SQL Server:

- Kết nối trên Desktop: Có thể trên cùng máy tính với SQL Server hoặc kết nối qua mạng nội bộ.

- Kết nối qua mạng diện rộng: Thông qua đường truyền mạng xa kết nối đến SQL Server.
- Kết nối qua mạng Internet: Các ứng dụng kết nối thông qua máy chủ Internet, dịch vụ IIS thực hiện ứng dụng trên Internet (ASP, JSP, ASP.net,…)

Mô hình Desktop.

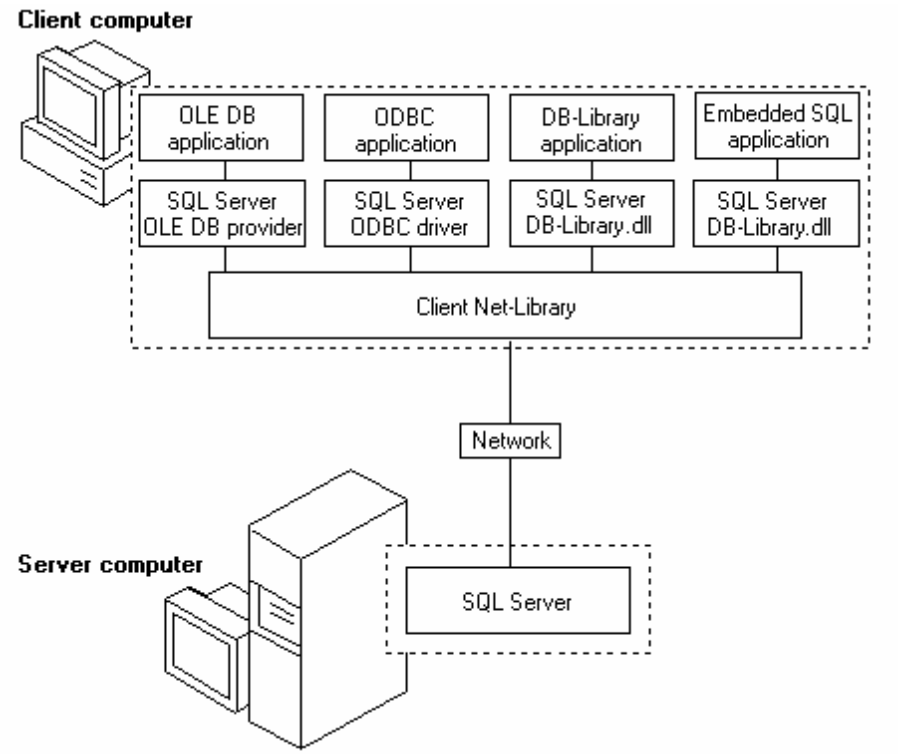
Nếu xét trên một máy Desktop sơ đồ kết nối trao đổi dữ liệu được thể hiện như sau:



Trên một Desktop có thể có nhiều ứng dụng, mỗi ứng dụng có thể thực hiện thao tác với nhiều CSDL.

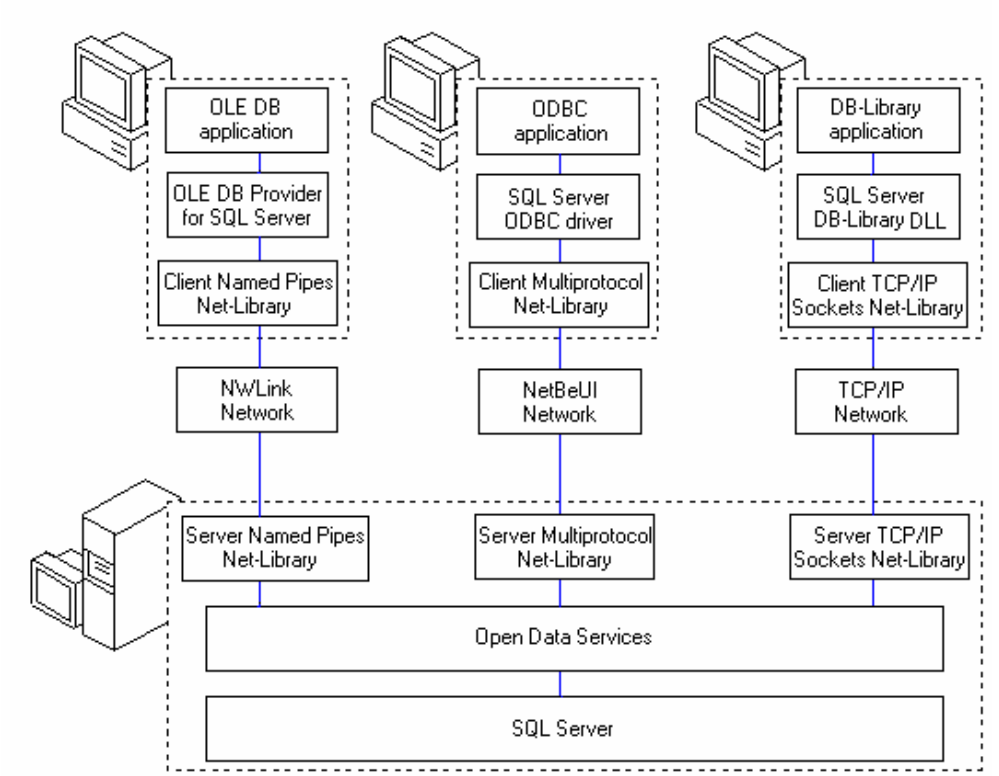
Mô hình Client/Server.

Nếu xét theo mô hình client/server, ứng dụng trao đổi với SQL Server theo sơ đồ sau:



Như sơ đồ trên nhận thấy SQL Server cho phép các ứng dụng kết nối theo các phương thức sau: OLE DB, ODBC, DB-Library, Embedded SQL, đây là các phương thức kết nối hữ ích cho những nhà phát triển ứng dụng.

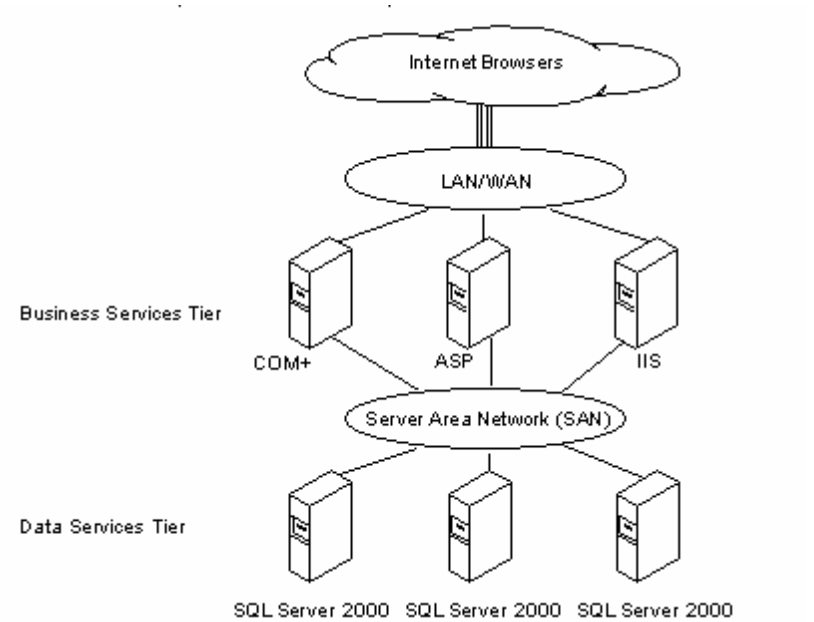
Nếu xem xét cụ thể hơn ta có thể xem sơ đồ sau:



Trong sơ đồ trên cho thấy, SQL Server có thể thực hiện trao đổi dữ liệu với các ứng dụngt heo nhiều giao thực truyền tin khác nhau (TCP/IP, NetBeUI, Names Pipes,…), các ứng dụng có thể sử dụng nhiều phương thức kết nối khác nhau (OLE DB, ODBC, DB-Library).

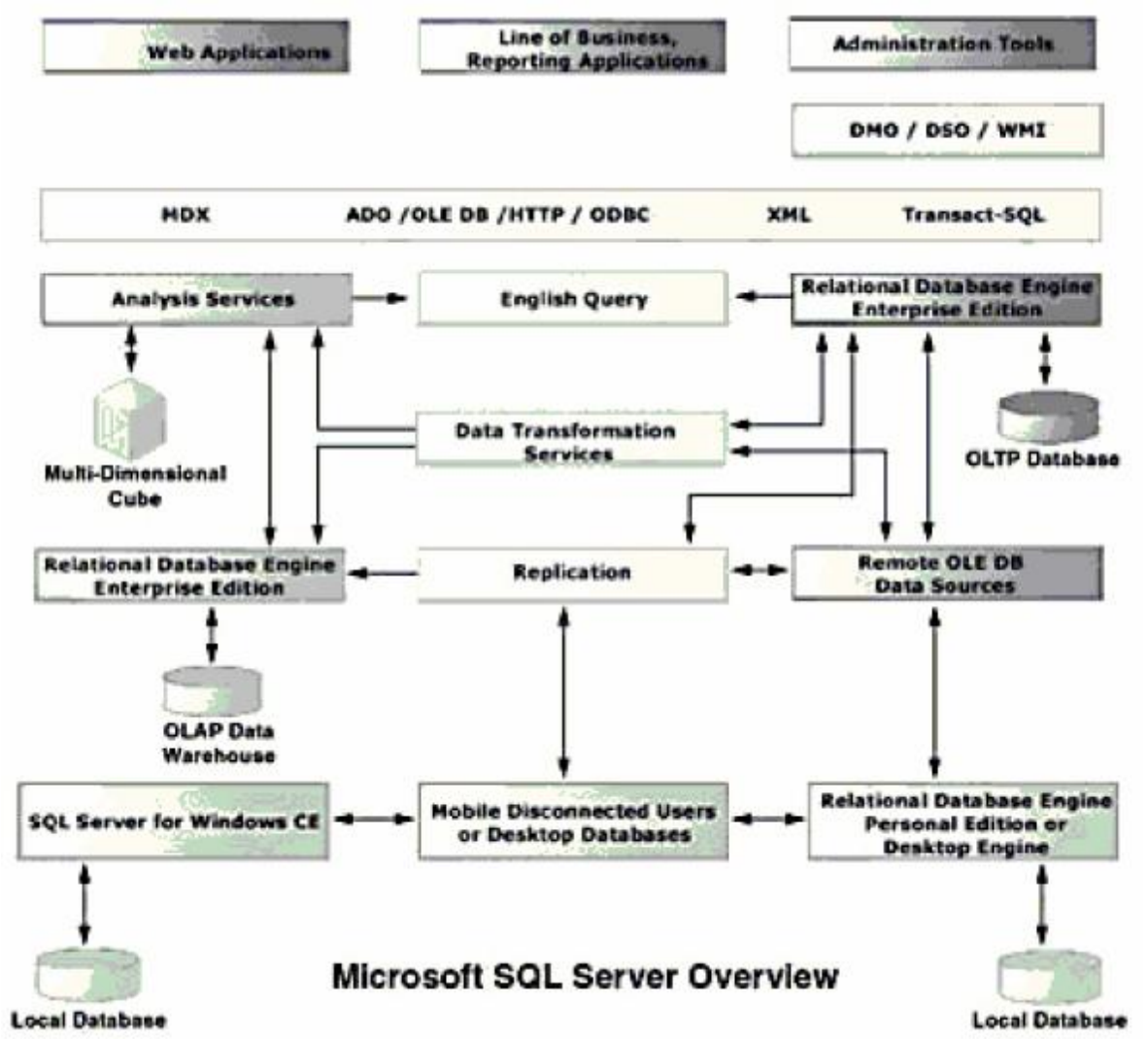
Mô hình kết nối ứng dụng trên mạng Internet.

Nếu xét riêng các ứng dụgn kết nối với SQL Server trên mạng Internet, các máy chủ SQL Server sẽ được quản lý thông qua các hệ thống máy chủ mạng, hệ điều hành mạng, các ứng dụng (COM+, ASP, IIS) sẽ thông qua máy chủ mạng kết nối đến SQL Server, mô hình này có thể áp dụng cho các mạng nội bộ, diện rộng, ứng dụng được khai thác trên trình duyệt Internet Browser. Xem xét mô hình dưới đây:



CÁC THÀNH PHẦN CỦA SQL SERVER.

SQL Server được cấu thành bởi nhiều thành phần khác nhau, các thành phần có mối quan hệ trong một hệ thống, phối hợp với nhau để tạo thành một giải pháp hoàn chính, nâng cao hiệu quả quản trị, phân tích, lưu trữ dữ liệu.



Relational DataBase Engine.

Đây là một engine có khả năng chứa dữ liệu dưới nhiều quy mô khác nhau, theo dạng bảng, hỗ trợ nhiều phương thức kết nối ADO, OLE DB, ODBC.

Replication.

Là công cụ dùng nhân bản dữ liệu, bạn có thể tạo một Server khác với bộ dữ liệu giống bộ dữ liệu trên Server chính. Công cụ tạo cơ chế tự đồng bộ dữ liệu giữa Server chính và Server nhân bản. Mục đích của việc tạo Server nhân bản là giảm tải

cho Server chính, nâng cao hiệu quả phục vụ với số lượng người, phiên giao dịch lớn.

Data Transformation Service – DTS.

Là công cụ giúp bạn chuyển dữ liệu giữa các Server quản trị CSDL khác nhau, DTS có thể chuyển dữ liệu từ SQL Server sang Oracle, Access, DB,… trước khi chuyển dữ liệu DTS định dạng kiểu dữ liệu để chuyển sang hệ quản trị CSDL khác.

Analysis service.

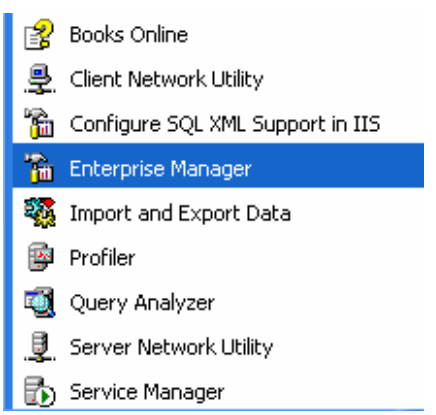
Là công cụ giúp khai thác phân tích dữ liệu, hay khai phá dữ liệu theo phương thức đa chiều. Từ một tập dữ liệu sẵn có bạn có thể khai phá rồi từ đó đưa ra những nhận định, phân tích, đánh giá và dự đoán theo lĩnh vực nào đố, mỗi chiều trong ngữ cảnh này được coi là một tiêu chí xem xét của dữ liệu.

English query.

Đây là công cụ tra cứu dữ liệu bằng tiếng anh, cú pháp có thể sử dụng theo văn phạm tiếng anh thông thường.

SQL Server tools.

Là bộ công cụ cung cấp giao diện cho người quản trị như Enterprise amanger, Query Analyzer ,…SQL Server sau khi cài đặt SQL Server group gồm những thành phần cơ bản trong group như sau:



Một số công cụ quan trong: Enterprise manager, Query Analyzer, Profiler…, các công cụ sẽ được giới thiệu khai thác sau.

CÀI ĐẶT SQL SERVER.

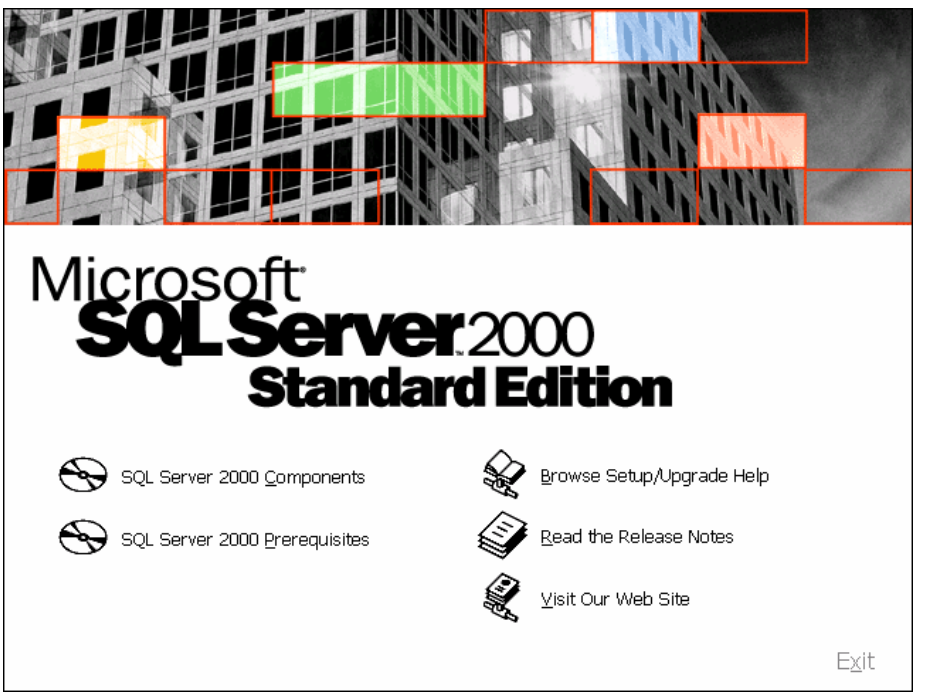
Chuẩn bị cài đặt.

Tùy theo môi trường của máy tính của bạn mà thực hành cài đặt phiên bản nào, bảng sau là tham số với SQL Server 2000 phiên bản Standard.

Computer	Intel® hoặc tương đương Pentium 166 MHz hoặc cao hơn
Memory (RAM)	Enterprise Edition: Tối thiểu 64 MB, 128 MB hoặc nhiều hơn. Standard Edition: Tối thiểu 64 MB. Personal Edition: Tối thiểu 64 MB trên Windows 2000, tối thiểu 32 MB trên các hệ điều hành khác. Developer Edition: Tối thiểu 64 MB. Desktop Engine: Tối thiểu 64 MB trên Windows 2000, tối thiểu 32 MB trên hệ điều hành khác.
Hard disk	SQL Server database components: Từ 95 đến 270 MB, thông thường 250 MB. Analysis Services: Tối thiểu 50 MB, thông thường130 MB. English Query: 80 MB Desktop Engine: 44 MB
Monitor	VGA hoặc độ phân dải cao hơn. 800x600 hoặc độ phân dải cao hơn.

Thực hành cài đặt.

- Sử dụng đĩa CD ROM có bộ cài đặt SQL Server 2000 (tuỳ theo yêu cầu của bạn là Standard, Personal hay Enterprise,...)
- Chạy trình Autorun.exe (thường tự chạy khi đưa đĩa vào máy tính)



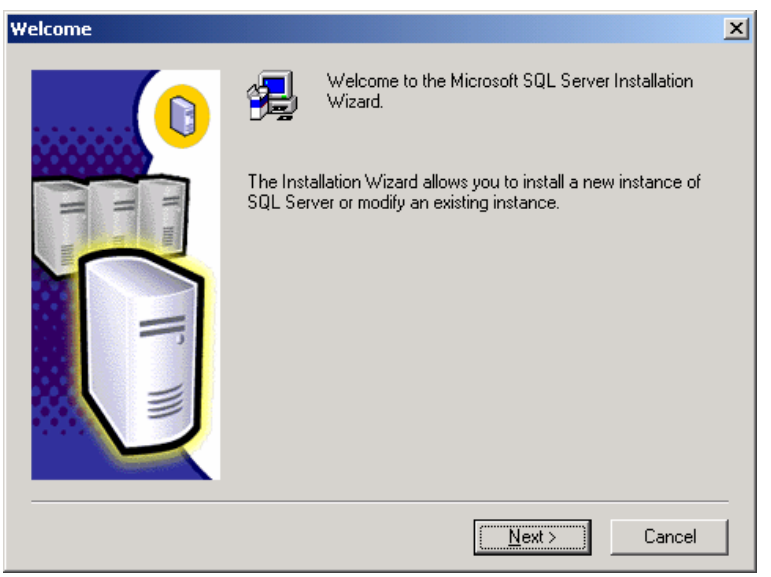
Trong màn hình trên ta có một số lựa chọn:

SQL Server Components: Sẽ thực hành trong bước tiếp.

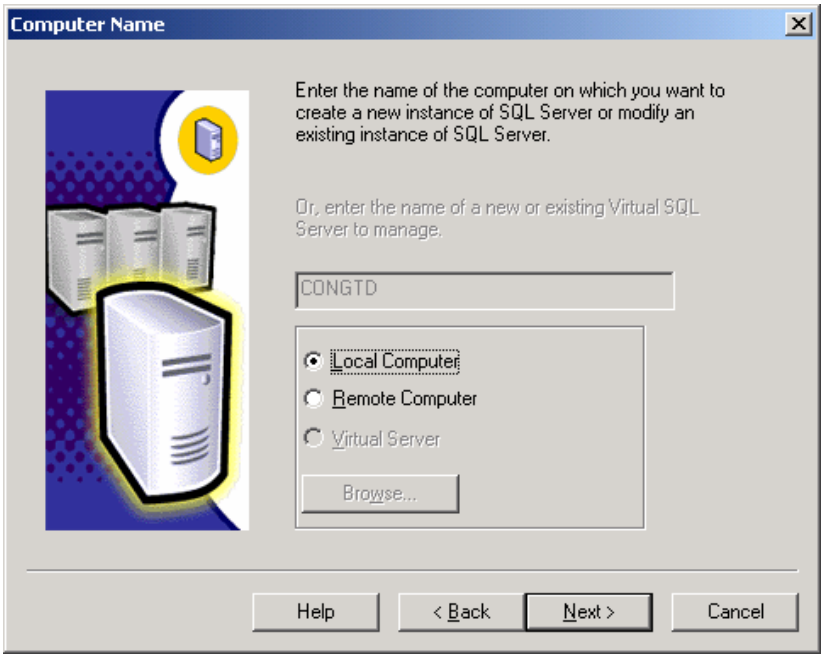
SQL Server 2000 Prerequisites: Dùng cài đặt những yêu cầu được cung cấp sẵn cho việc cài đặt nếu hệ thống trong máy cài đặt chưa đủ.

- Chọn SQL Server Components.

- Chọn Install Database Server.



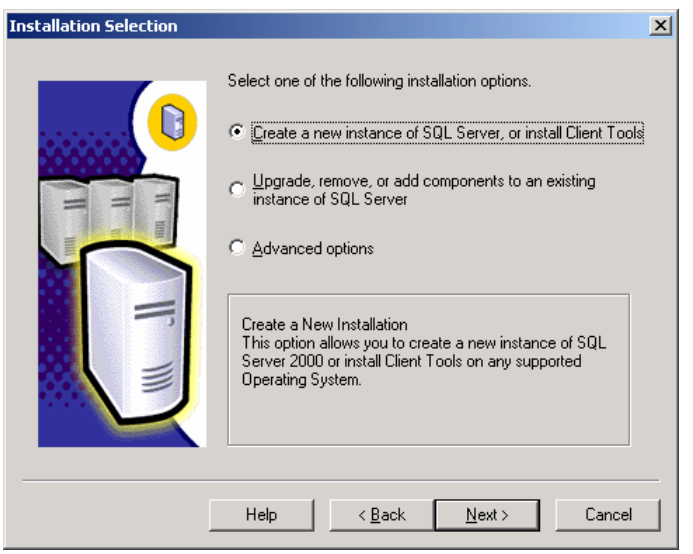
- Chọn Next.



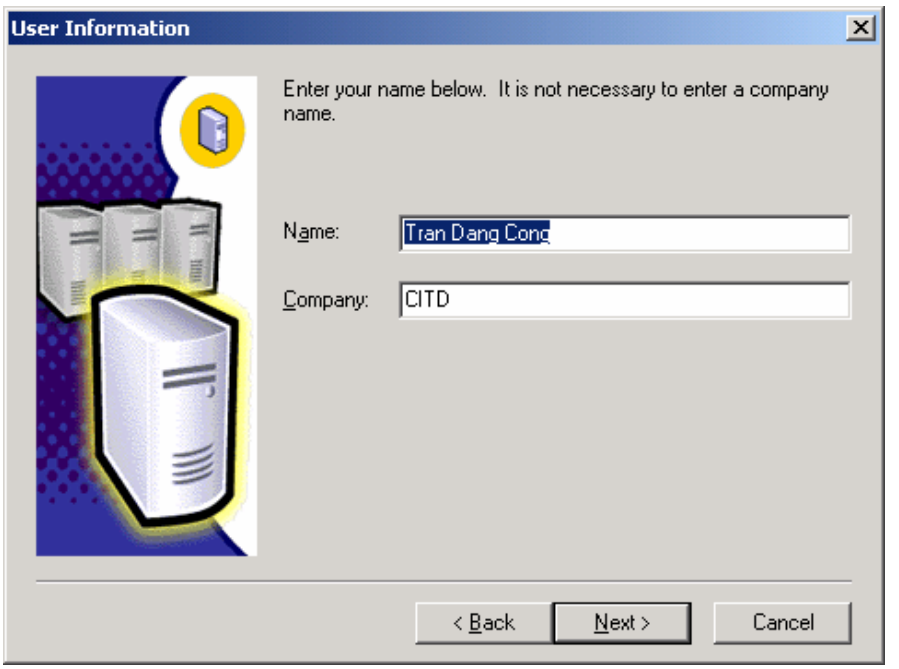
Nếu cài đặt SQL Server trên chính máy bạn đang ngồi thì sử dụng Local Computer

Nếu cài đặt dùng kết nối với máy khác thì sử dụng Remote Computer sau đó nhập tên máy hoặc chọn vị trí máy bằng cách sử dụng Browse

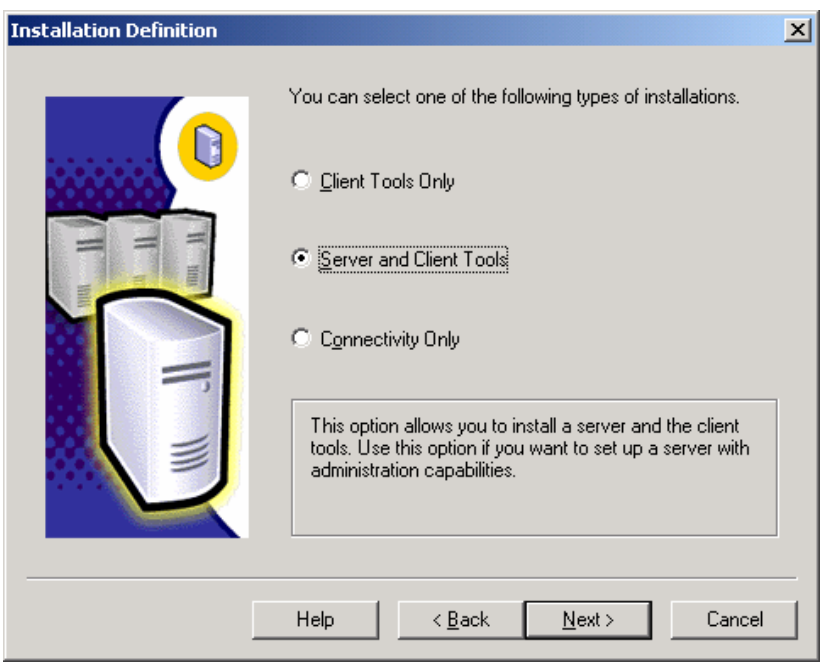
- Chọn next.



- Chọn tuỳ chọn theo chỉ dẫn (tạo mới, thay đổi cái đã có, thêm các chức năng khác,...).
- Trong trường hợp chọn tạo mới (lựa chọn thứ nhất) sau đó ấn Next.



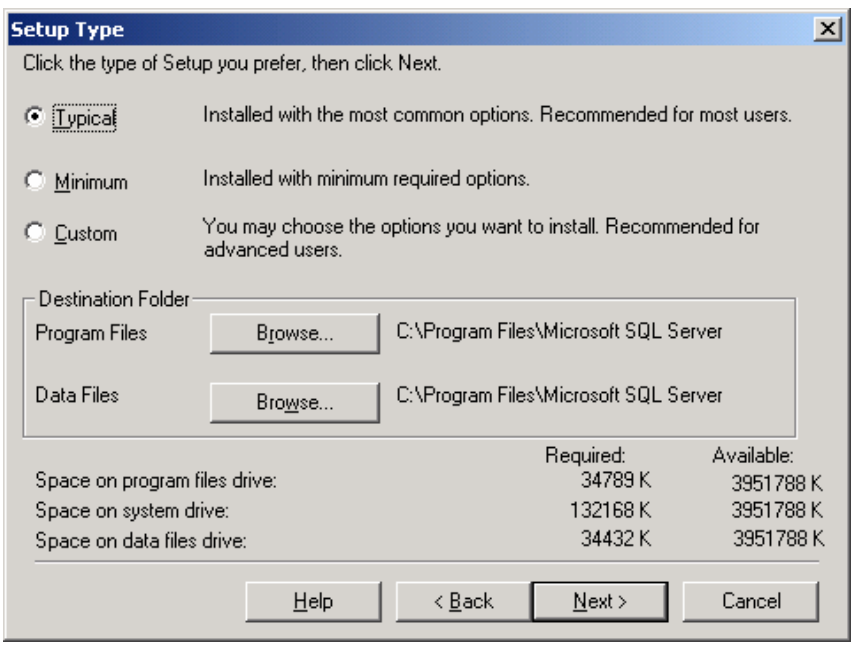
- Nhập tên của bạn, tên cơ quan, sau ấn Next, Yes.



Trong cửa sổ hiện lên 3 lựa chọn:

1. Cài đặt các công cụ truy vấn: Sử dụng cho các máy khách không lưu trữ dữ liệu nhưng có chức năng truy vấn dữ liệu đến SQL Server có CSDL
2. Cài đặt Server và các công cụ truy vấn: Cài đặt SQL Server có dữ liệu và các công cụ của máy khác truy vấn dữ liệu
3. Cài đặt kết nối: Dùng cho các máy chỉ sử dụng kết nối đến Server, thường dùng cài đặt cho các máy sử dụng các ứng dụng kết nối đến server

- Chọn lựa chọn 2, sau ấn Next



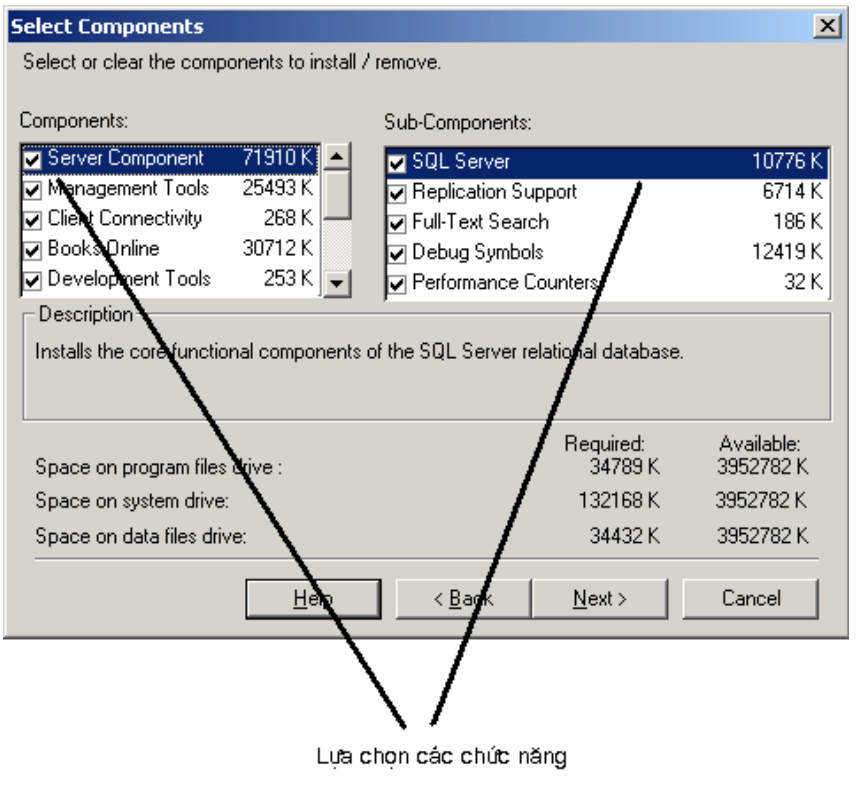
Dùng các lựa chọn để chọn kiểu cài đặt:

+ *Typical*: Cài đặt những chức năng cơ bản được hệ thống định sẵn (chức năng thông thường).

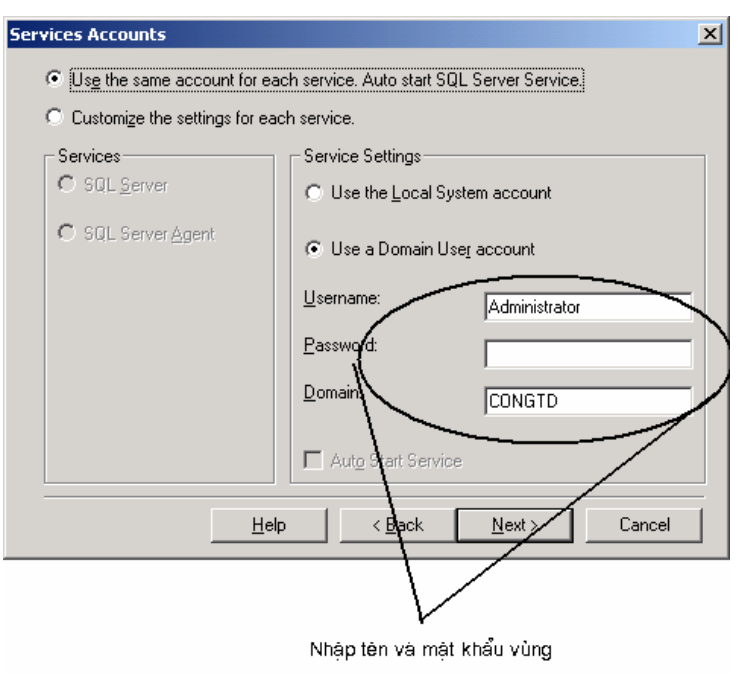
+ *Minimum*: Cài đặt những chức năng tối thiểu của hệ thống.

+ *Custom*: Lựa chọn những chức năng cần cài đặt theo yêu cầu của người dùng.

Trong cách lựa chọn Custom ta cần thêm bước chọn các chức năng như sau:

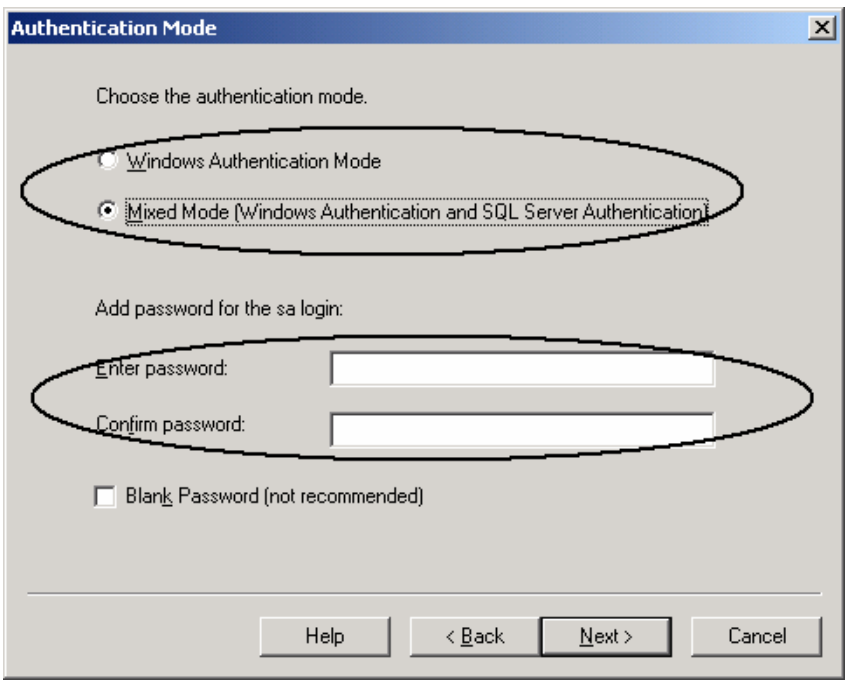


- Ấn nút Next để tiếp tục.



Trong cửa sổ trên ta cần nhập tên, mật khẩu của user được đăng ký truy nhập vùng, thông thường SQL Server cài đặt được thực hiện theo quyền Administrator của máy tính chủ, khi đó bạn lựa chọn nút chọn Use a Domain User account.

- Ấn nút next để tiếp tục.



Trong cửa sổ trên cho phép ta sử dụng 2 lựa chọn:

+ *Lựa chọn thứ nhất*: Người dùng sử dụng hệ thống bảo mật của Windows (hệ điều hành của máy chủ cài đặt – thông thường khi cài đặt dùng lựa chọn này).

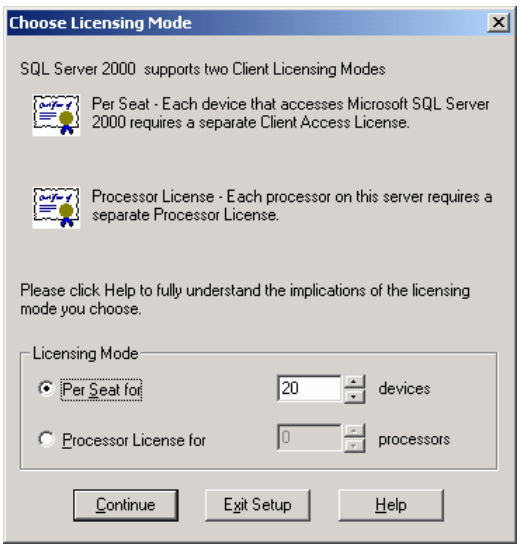
+ *Lựa chọn thứ hai*: Người dùng sử dụng hệ thống bảo mật của Windows và của hệ quản trị CSDL SQL Server.

Trong các trường hợp trên đều có thể sử dụng tên và mật khẩu được cung cấp theo vùng (domain) của hệ điều hành. Nếu sử dụng lựa chọn thứ 2 ta sử dụng tên và mật khẩu của người quản trị vùng (Administrator).

Đối với SQL Server ta có thể thay tên Administrator bằng tên sa (viết tắt của từ System Administrator).

Vấn đề thực hiện chọn chế độ bảo mật nào sẽ được bàn trong những bài sau.

- Ấn next để tiếp tục.

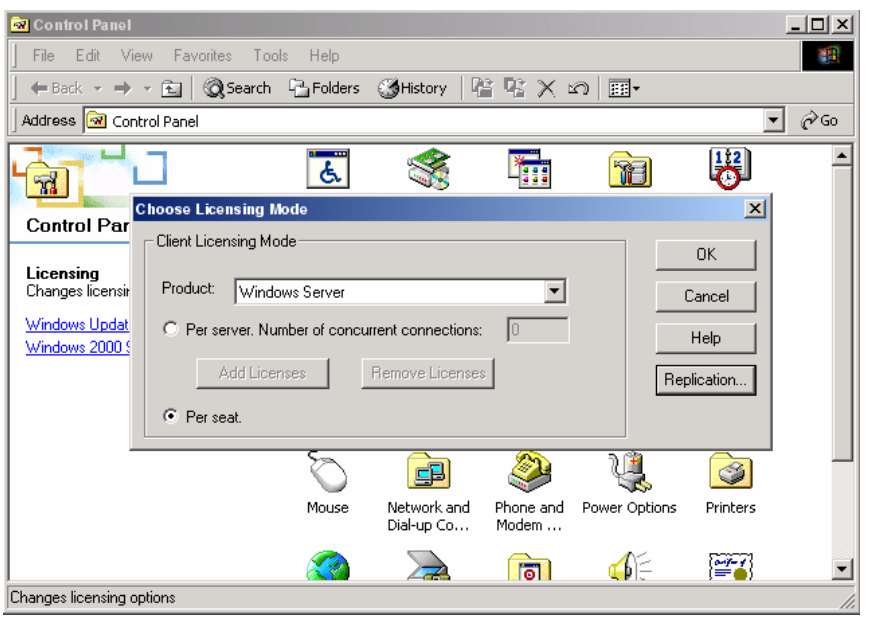


Trong cửa sổ trên ta có 2 lựa chọn:

+ *Per Seat for*: Lựa chọn cho phép xác định số thiết bị (khái niệm sẽ được giới thiệu sau) trên mỗi vị trí khai thác hệ thống theo bản quyền được phép của Microsoft.

+ *Processor License for*: Xác định số Processor cho phép sử dụng theo bản quyền được cung cấp bởi Microsoft.

Ngoài việc đăng kỹ bản quyền tại thời điểm này, ta có thể đăng ký bản quyền trong công cụ điều khiển của Control Panel.



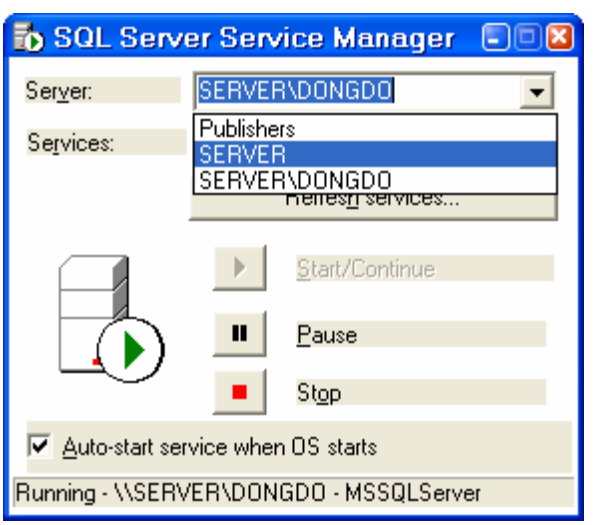
QUẢN TRỊ SERVER

INSTANCE

SQL Server hỗ trợ nhiều hoạt động trên mạng, như các mô hình đã xem xét trước ta có thể thiết lập nhiều máy tính cài đặt SQL Server, các máy tính có thể liên kết với nhau, trao đổi dữ liệu với nhau.

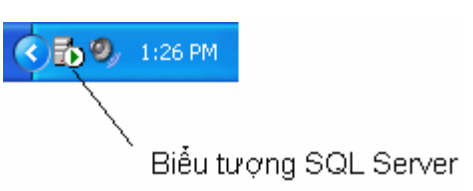
Tuy nhiên một máy tính cũng có thể thiết lập nhiều hệ thống SQL Server khác nhau, mỗi hệ thống đều có một tên quy định, mỗi hệ thống như vậy gọi là một Instance.

Mỗi Instance trên một máy tính được coi như một hệ thống SQL Server độc lập, tương tự như các hệ thống SQL Server cài đặt trên các máy tính khác nhau.

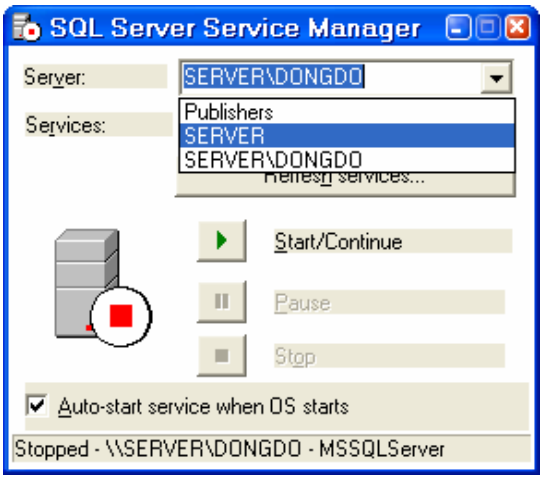


ĐIỀU KHIỂN CÁC DỊCH VỤ CỦA SQL SERVER.

SQL Server sau khi cài đặt xong, khởi động máy thông thường sẽ được thiết lập có biểu tượng ở góc dưới, trái màn hình như sau:



Biểu tượng mày chỉ có với máy tính cài đặt phiên bản SQL Server và là biểu tượng của trình quản lý dịch vụ Service Manager.



Gồm các dịch vụ cơ bản sau:

+ Distributed Transaction Coordinator - DTC.

+ Microsoft Search.

+ SQL Server.

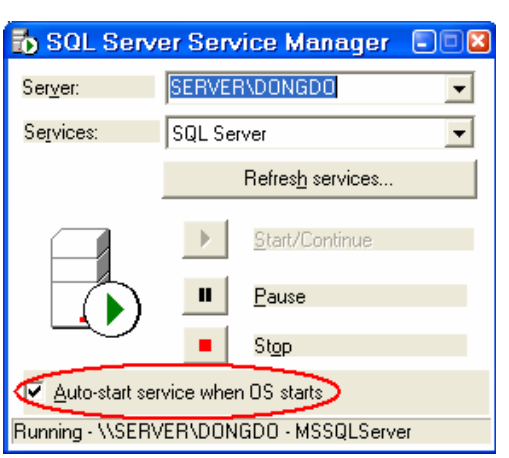
+ SQL Server Agent.

Các dịch vụ này ta có thể bắt đầu, tạm dựng hoặc kết thúc, mỗi dịch vụ đều điều khiển các ứng dụng, công cụ quản trị của SQL Server.

Để thực hiện điều khiển dịch vụ đầu tiên ta làm như sau:

Services -> Start/Continue (Pause, Stop)

Để dịch vụ khởi động tự động khi khởi động hệ điều hành hãy chọn vào nút chọn ***Auto-start service when OS starts***.

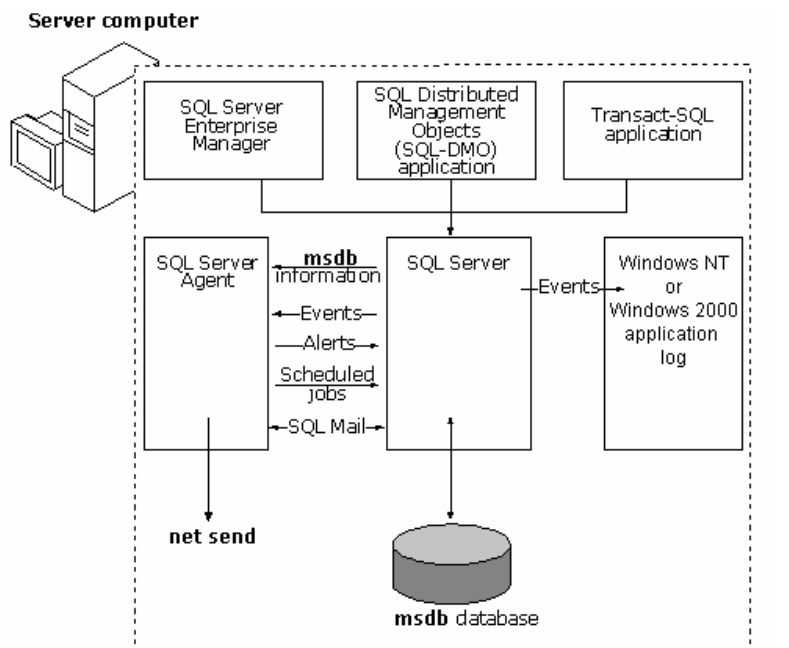


Dịch vụ MS SQLServer.

Dùng quản lý tất cả các file gồm các CSDL mà SQL Server quản lý, là thành phần xử lý tất cả các lệnh của Transact-SQL được gửi từ các trình ứng dụng client, phân phối các nguồn tài nguyên khi có nhiều user cùng truy nhập một lúc. Đây là dịch vụ quản trị cơ bản, khi ngắt dịch vụ này hệ thống sẽ ngưng tất cả các công việc khai thác dữ liệu.

Dịch vụ SQLServerAgent.

Hỗ trợ các tính năng cho phép lập thời biểu các hoạt dộng theo từng giai đoạn tren SQL Server, hoặc thông báo cho người quản lý hệ thống về những sự cố của hệ thống, bao gồm các thành phần Jobs, Alerts, Operator.

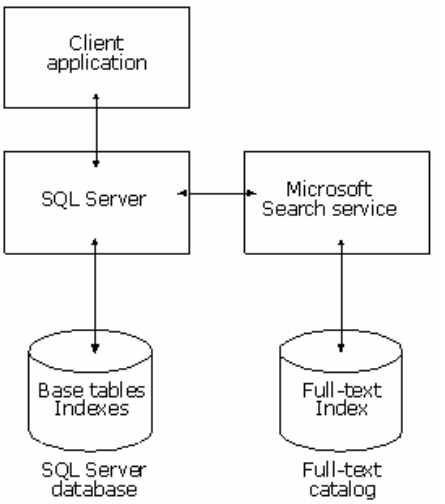


Dịch vụ Microsoft Search.

Cung cấp dịch vụ tìm kiếm và tìm kiếm văn bản với các phép toán cơ bản sau:

+ Ký tự (chuỗi): =, >, >=, <, <= được so sánh với một chuỗi hằng.

+ So sánh chuỗi nhỏ trong văn bản hoặc chuỗi có kích thước lớn, văn bản.



Dịch vụ MS DTC.

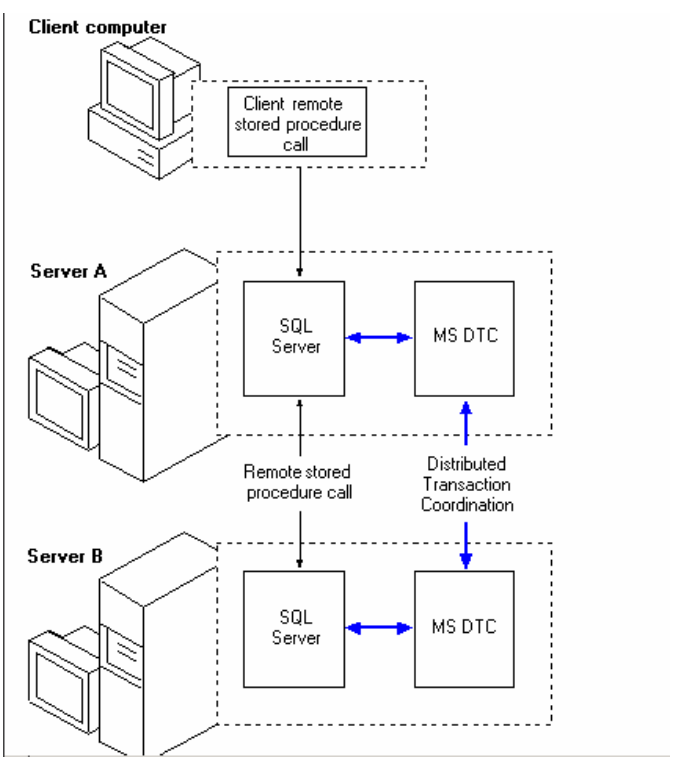
Là dịch vụ cho phép trong một phiên giao vận có thể sử dụng dữ liệu được phân phối trên nhiều server khác nhau, thực hiện theo các bước cơ bản sau:

+ Gọi các thủ tục lưu trữ trên các server xa sử dụng SQL Server

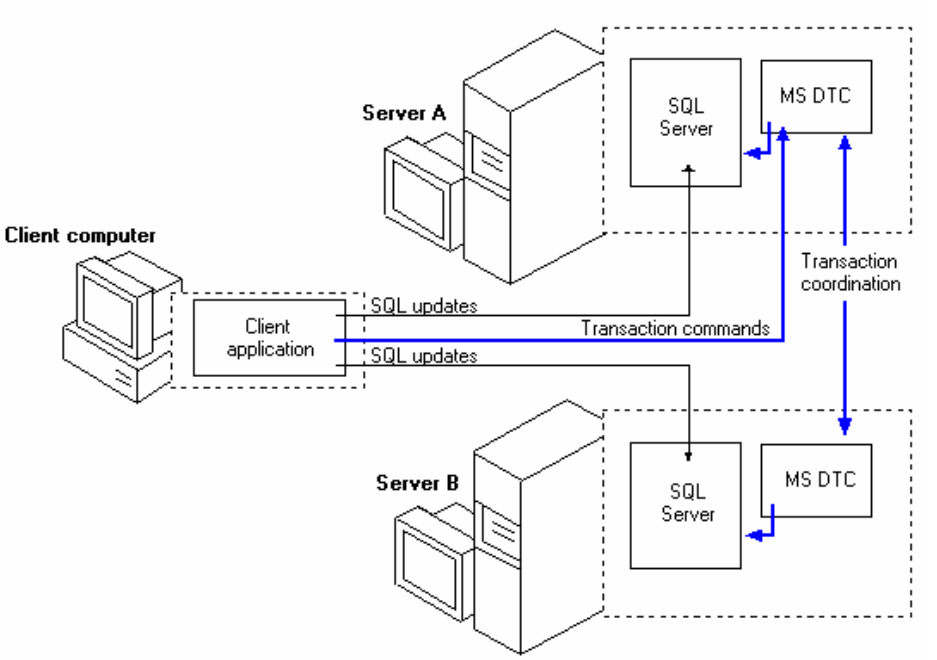
+ Tự động gọi hoặc tạo các phiên giao vận cục bộ và các giao vận với các máy chủ từ xa

+ Tạo bộ dữ liệu được cập nhật hoặc được phân phối bởi các server xa.

Xem xét sơ đồ hoạt động sau:



Như sơ đồ trên khi client triệu gọi một thủ tục có sẵn đến server cục bộ, khi có yêu cầu dữ liệu trên server khác, thông qua dịch vụ MS DTC server cục bộ sẽ triệu gọi các thủ tục từ server từ xa, kết quả có thể tạo được bộ dữ liệu được tập trung từ nhiều server khác nhau.



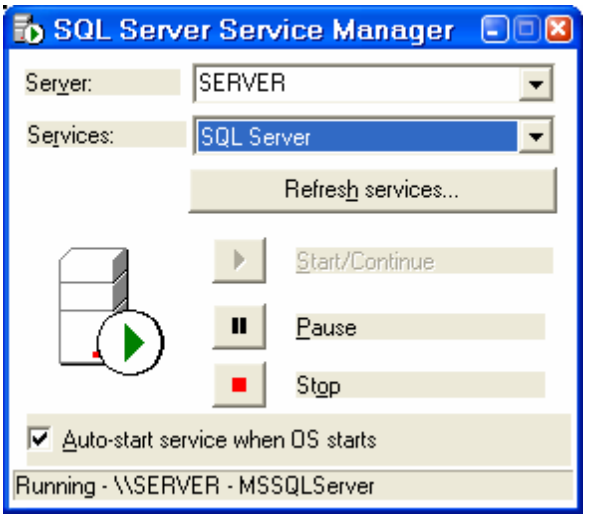
QUẢN TRỊ SERVER.

Như đã nếu trên mỗi Instance được coi là một hệ thống quản trị CSDL SQL Server và có thể gọi tắt là Server. Server có chức năng quản trị toàn bộ hệ thống của SQL Server (dữ liệu, bảo mật, người dùng, tác vụ, các dịch vụ khác,...).

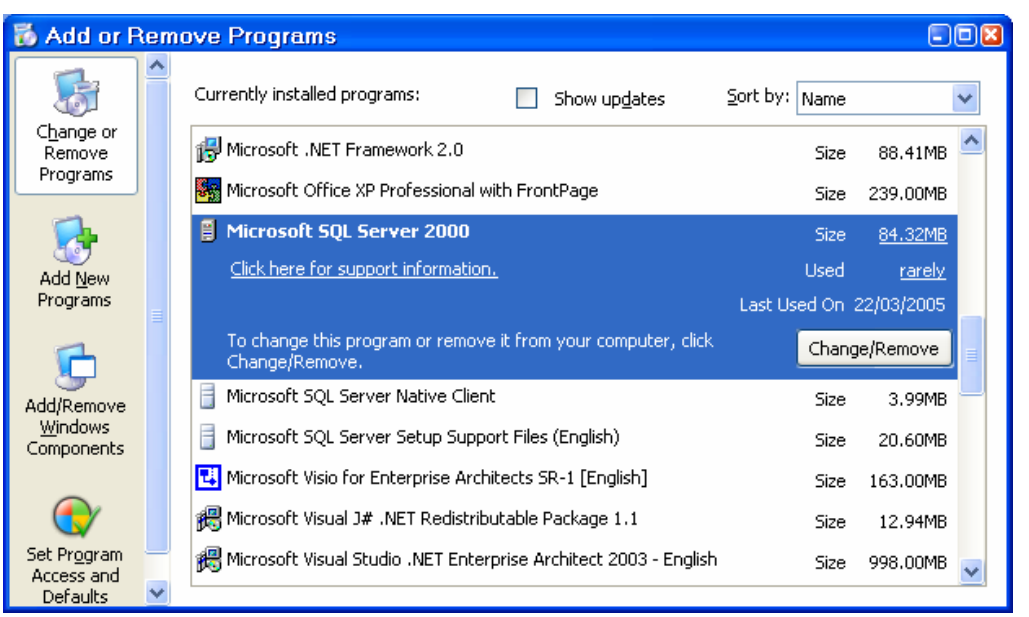
Các ứng dụng hoặc các công cụ khai thác dữ liệu (GUI) sẽ thực hiện khai thác dữ liệu do Server quản lý khi có kết nối đến Server. Tuy nhiên một vấn đề quan trọng là Server đó phải sẵn sàng phục vụ.

Thực hiện quản trị Server là việc thực hiện các công việc sau:

+ Bật/tắt/tạm dừng dịch vụ của SQL Server.



+ Gỡ bỏ hoặc cài đặt Server (Instance).



+ Thay đổi, nâng cấp phiên bản.

THIẾT LẬP KẾT NỐI ĐẾN SERVER.

Để khai thác được dữ liệu của hệ thống SQL Server ta phải thực hiện kết nối (connect) đến Server, việc kết nối có thể thực hiện từ các vị trí: Ứng dụng, công cụ khai thác của SQL Server là SQL Client. Trong phần này ta sẽ xem xét việc kết nối từ SQL Client đến Server.

Mỗi Server khi cài đặt đã có một tên là tên của Instance được đặt. Trên một mạng máy tính nếu có đủ quyền hạn ta hoàn toàn có thể thực hiện kết nối đến Server nói trên.

Từ một máy SQL Client có thể thực hiện đồng thời kết nối đến nhiều Server khác nhau, đây cũng chính là ưu điểm của SQL Server.

Nếu bạn cài đặt phiên bản SQL Server trên máy tính bạn cũng phải làm tiòan bộ các bước kết nối như SQL Client, phiên bản SQL Server được coi như gồm 2 phần: Hệ thống quản trị, công cụ khai thác SQL Client.

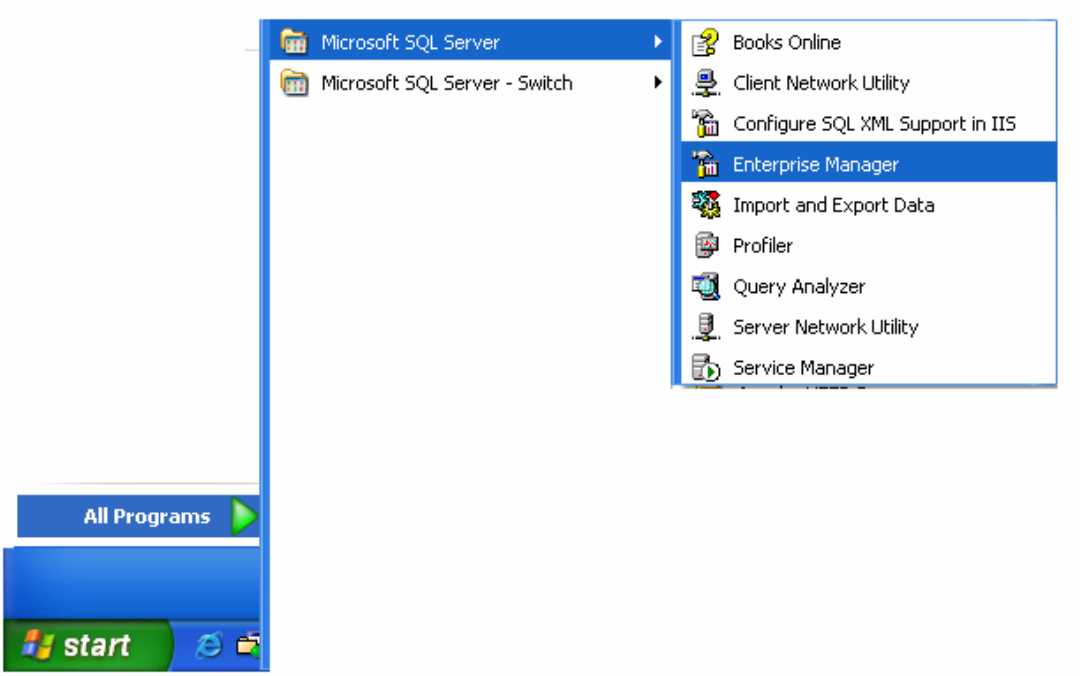
Dù sử dụng công cụ nào để khai thác đã được cài đặt trên máy tính của bạn, quyền hạn khai thác, quản trị phụ thuộc vào user thực hiện kết nối.

Quản trị Server Group.

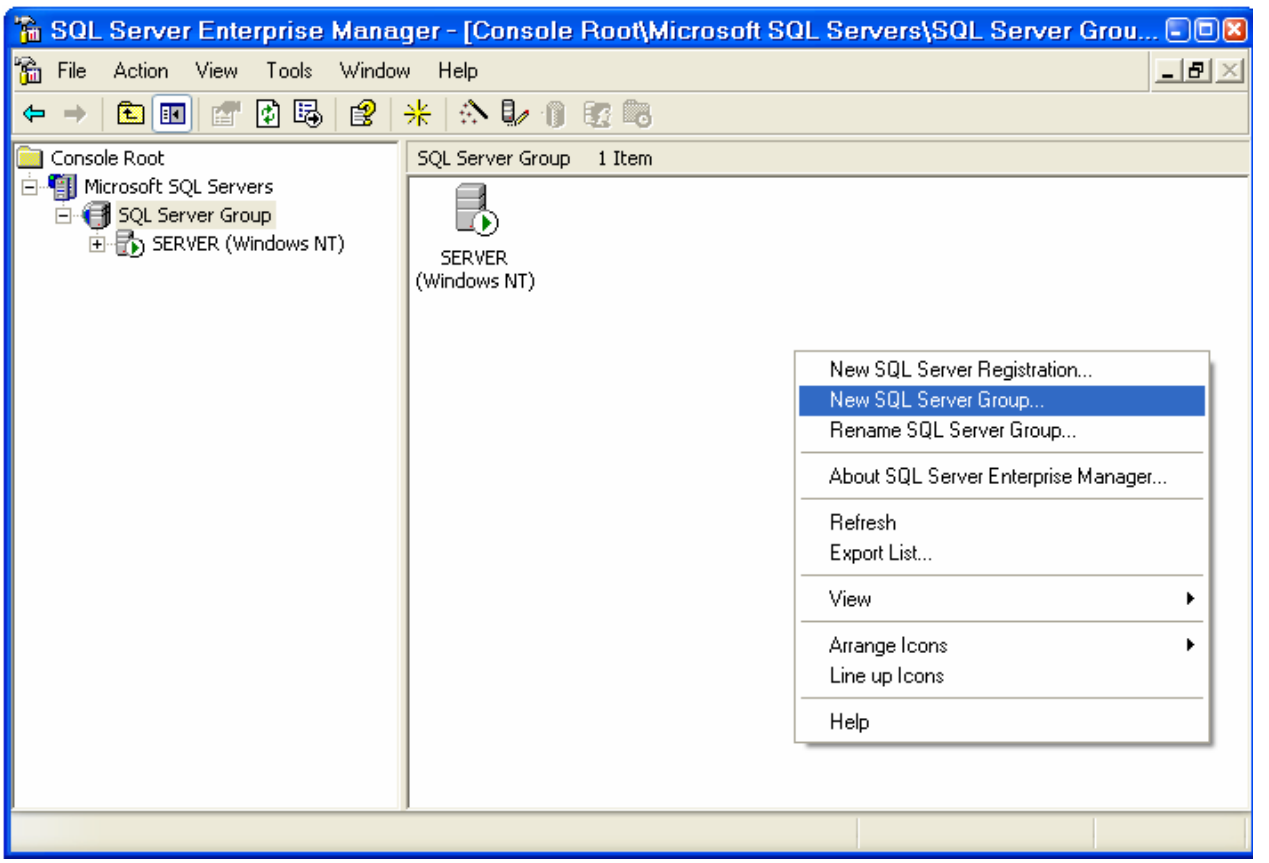
Server Group là công cụ dùng quản lý các kết nối (sẽ thực hành sau) tương tự như khái niệm thư mục trong hệ điều hành, trong các Server Group chứa các Server Group con hoặc các kết nối đến Server.

Các bước thực hiện như sau:

- Vào chức năng Enterprise manager như hình dưới

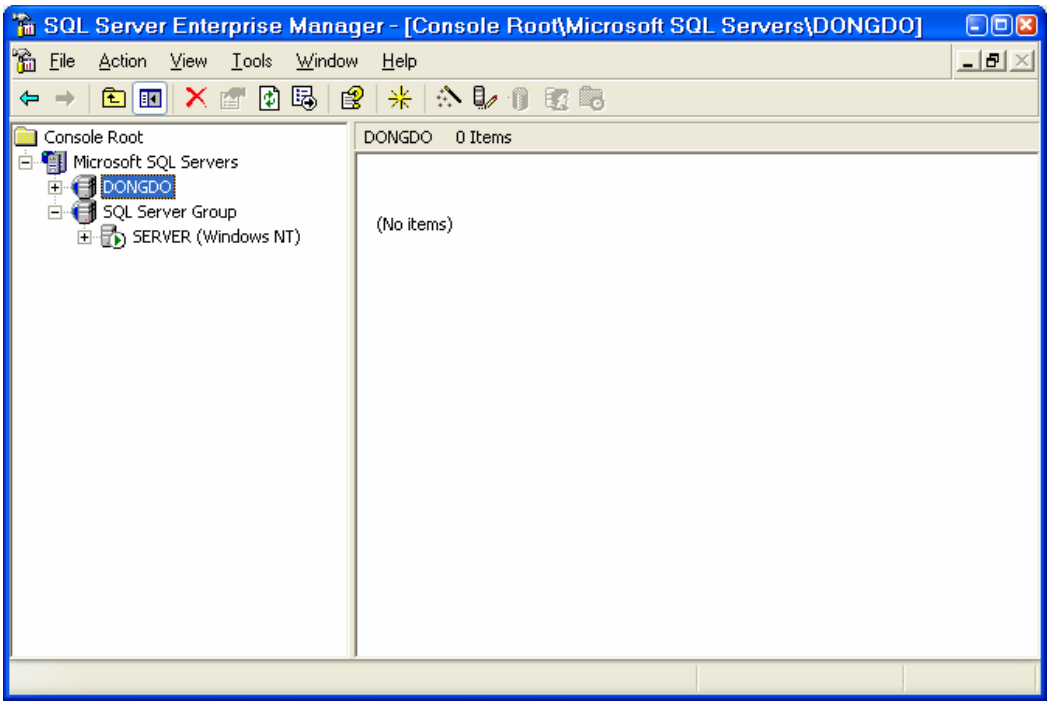


- Di chuyển vào mức trong bằng cách nhấn vào dấu + của cây các đối tượng.

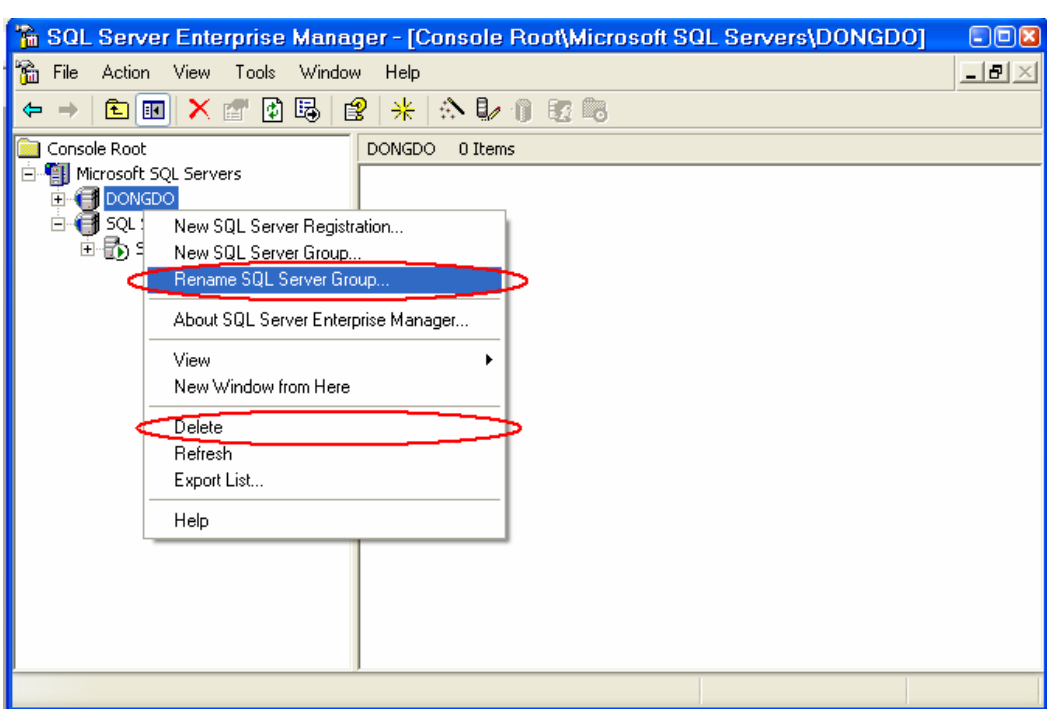


- Chọn New SQL Server group để tạo group mới

- Nhập tên group -> Ok



Các thao tác đổi tên, xóa được thực hiện bằng cách nhấn phải chuột vào group cần thao tác.



Thiết lập kết nối đến Server (thiết lập Server).

Là bước thiết lập kết nối đến Server từ Client, Server khác. Các kết nối được thể hiện bằng tên của Server kết nối đến (hay còn là tên của Instance), chính vì vậy nên tên các kết nối trên một Client là duy nhất, không trùng nhau trong toàn bộ client.

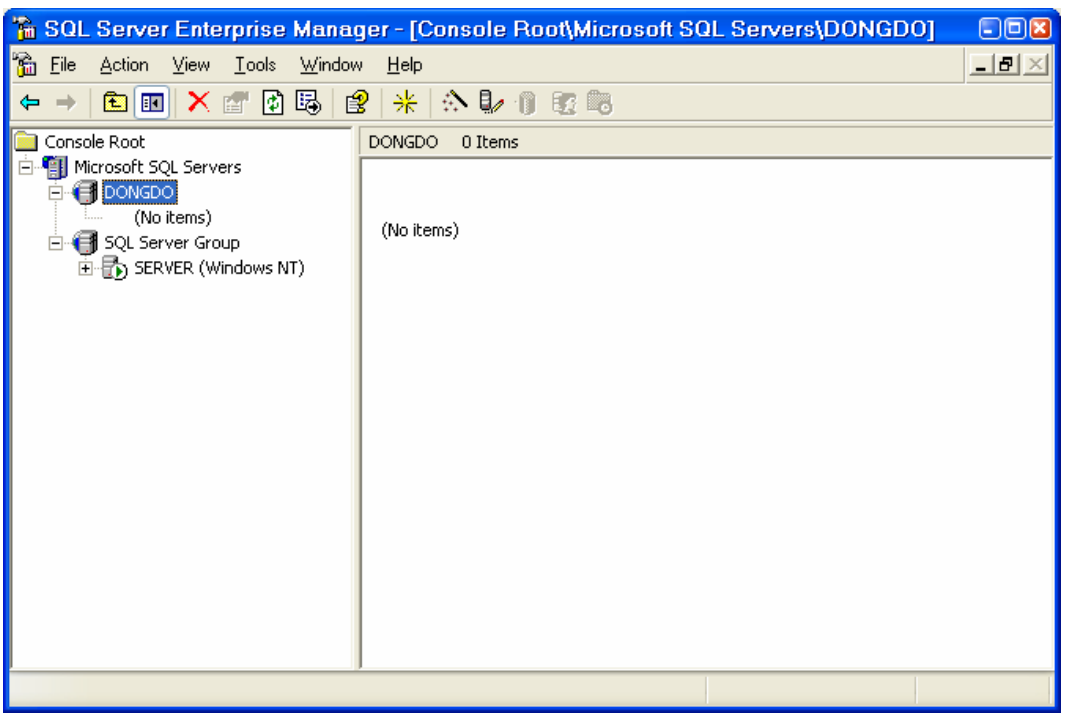
Trước khi thực hiện tạo kết nối ta phải chuẩn bị các tham số sau:

+ Tên Server (Instance) muốn kết nối đến

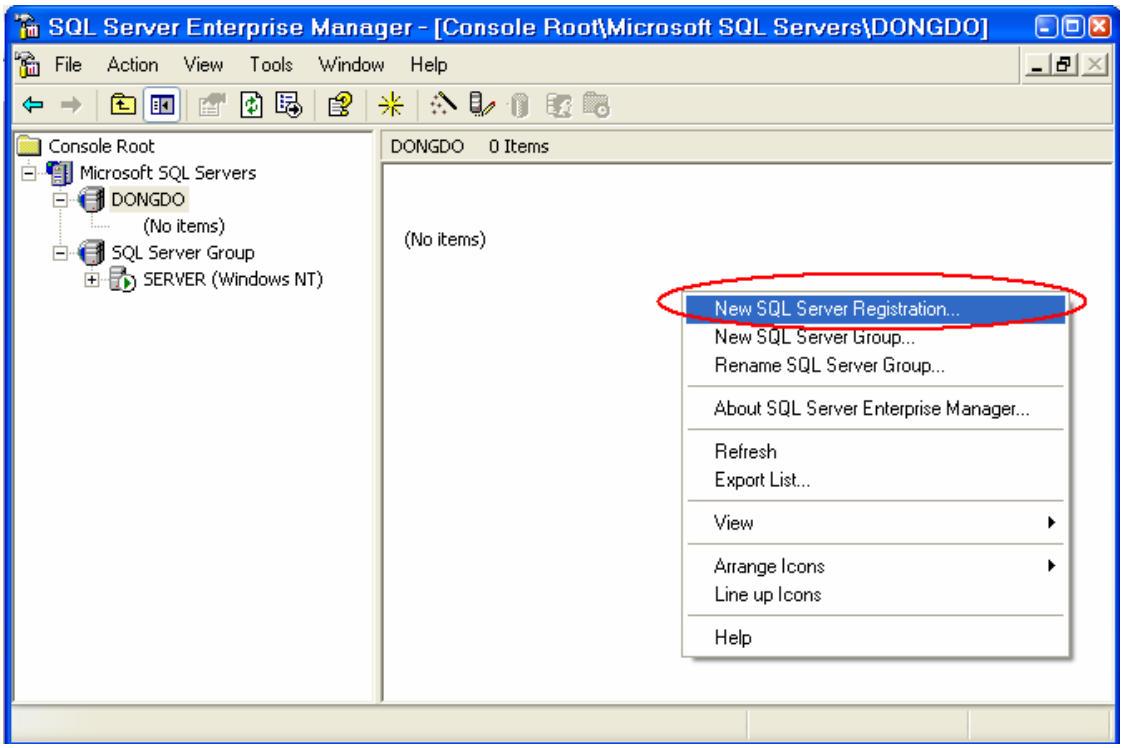
+ User name và Password của Server ta cần kết nối đến (tham số này do người quản trị Server cấp).

Cách làm như sau:

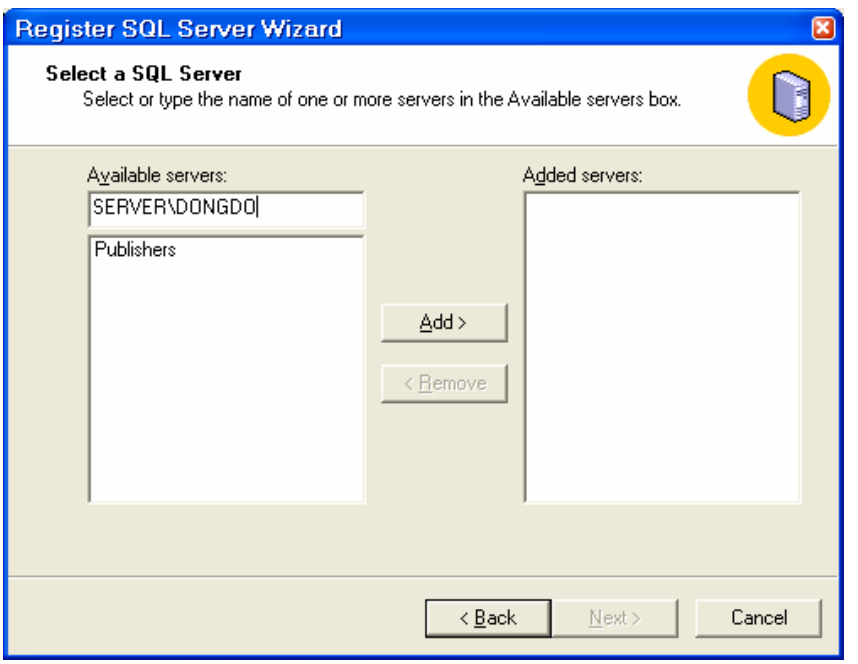
- Vào Enterprise và chọn Server group



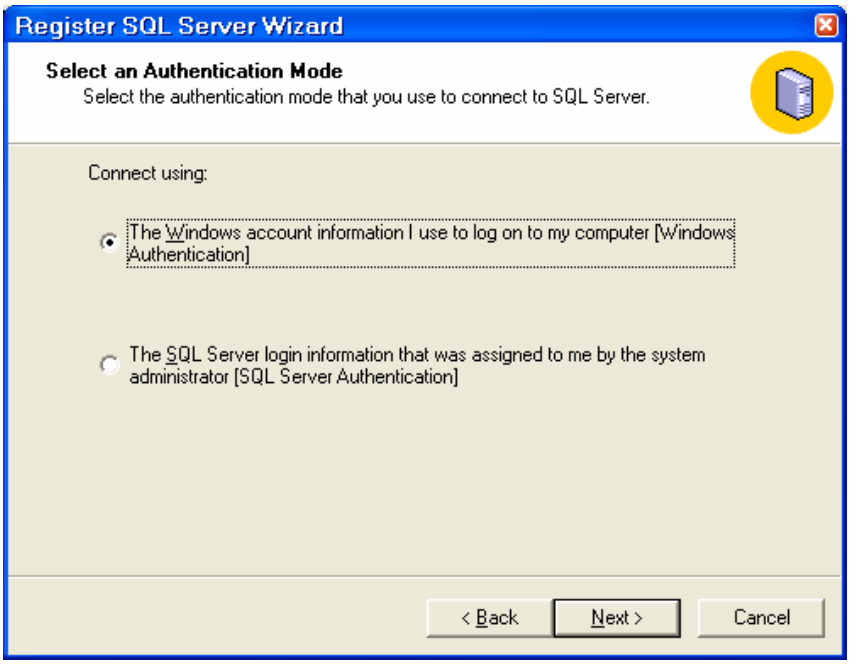
- Nhấn nút phải chuột vào cửa sổ bên phải, chọn New SQL Server Registration.



- Nhập tên Server.

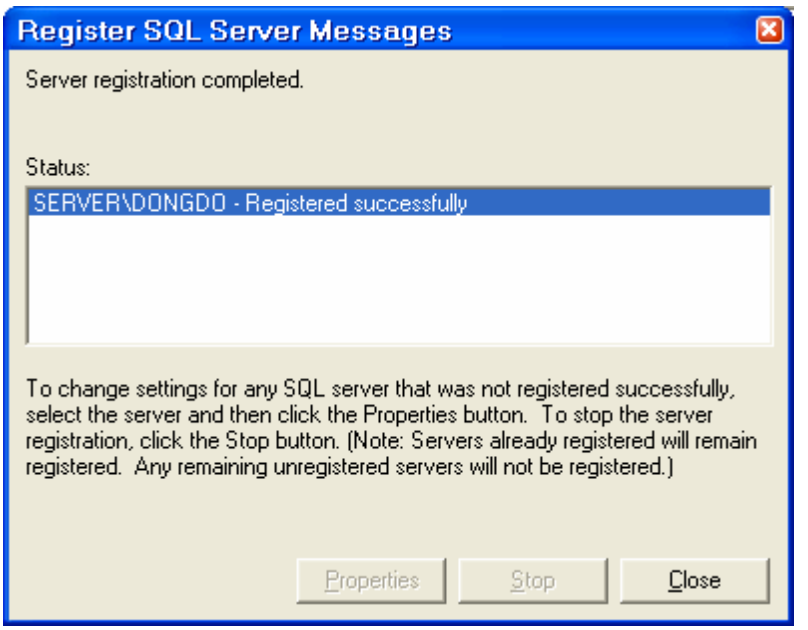


- Nhấn Add -> Next



- Chọn chế độ bào mật (thông thường chọn The Windows account information để chọn chế độ bảo mật của Widnows, phần này sẽ xem xét kỹ trong bài sau) -> Next -> Chọn Server Group.

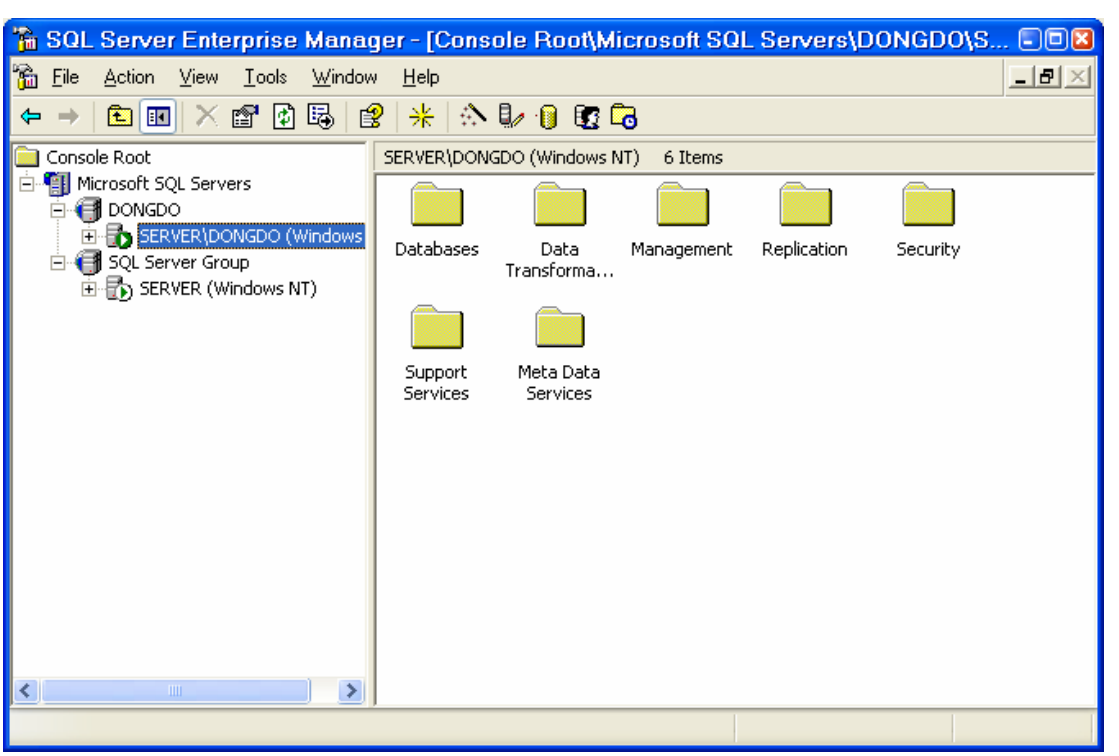
- Nhấn Finish.



Khi màn hình xuất hiện thông báo Registered successfully là việc thiết lập đã thành công.

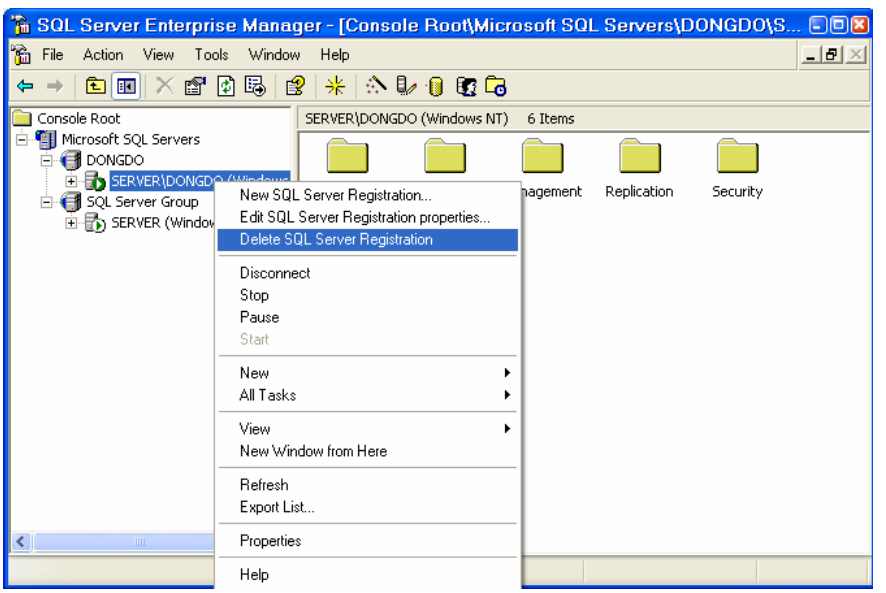
- Nhấn Close.

Sau khi thiết lập xong kết nối xuất hiện trên danh sách các kết nối.



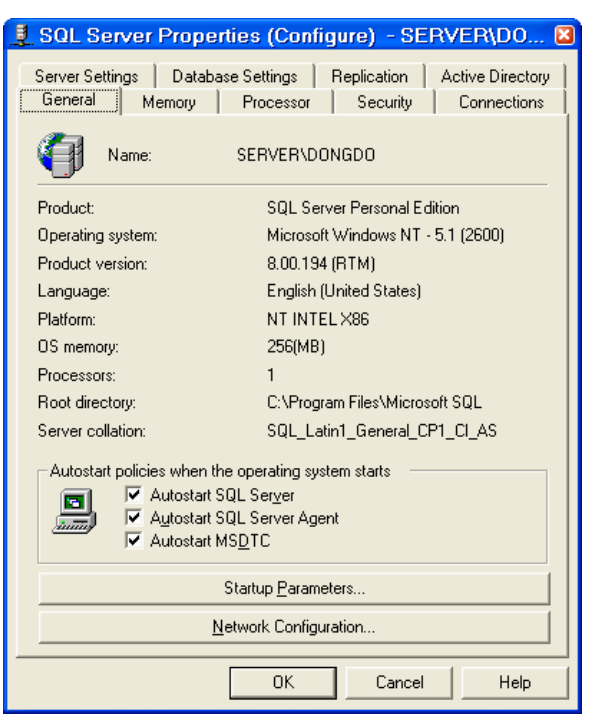
Kết nối như đã thiết lập có quyền hạn khai thác phụ thuộc vào user kết nối, trong ví dụ trên quyền hạn phụ thuộc vào user đã truy nhập vào Windows, tuy nhiên trong những bài sau sẽ giới thiệu cách thứac tạo user, sử dụng user của SQL Server để thực hiện tạo kết nối và khai thác.

Để xóa hoặc sửa thông tin cho kết nối chọn Delete hoặc Edit khi nhấn nút phải chuột vào tên kết nối cần thao tác.



Xem và thay đổi tham số cho Server.

Để thay đổi tham số cho Server, hãy chọn tên kết nối -> nhấn nút phải chuột -> chọn Properties.



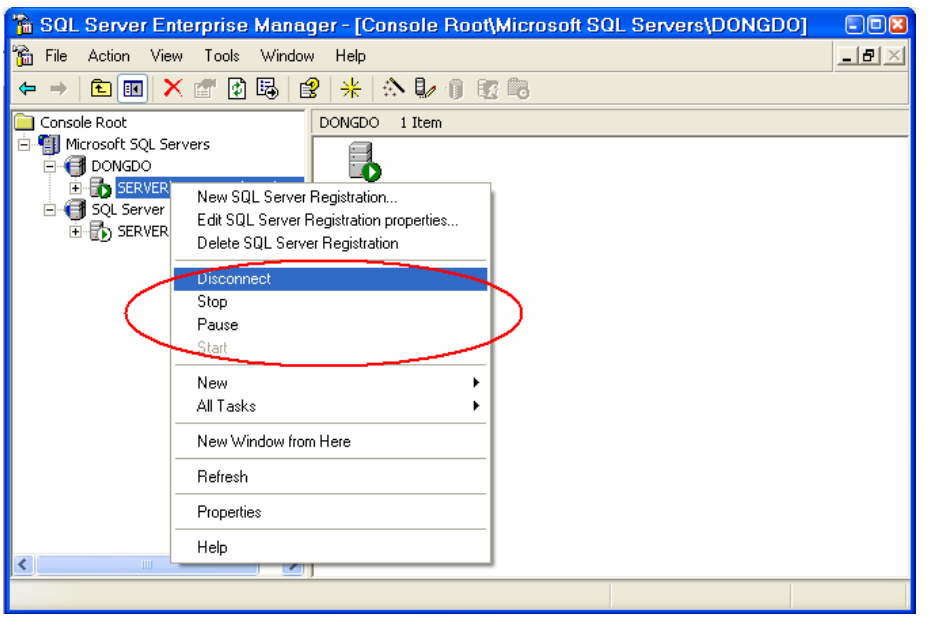
Tuy nhiên các tham số trên có thể bạn chưa xem xét, nên trong bài này chỉ giới thiệu mã sẽ xem xét một số tham số cơ bản trong như bài liên quan.

Bật/tắt/tạm dừng/kết nối/ngắt kết nối Server.

Ta có thể thực hiện tạm điều khiển dịch vụ MS SQL Server từ kết nối.

- Chọn tên kết nố

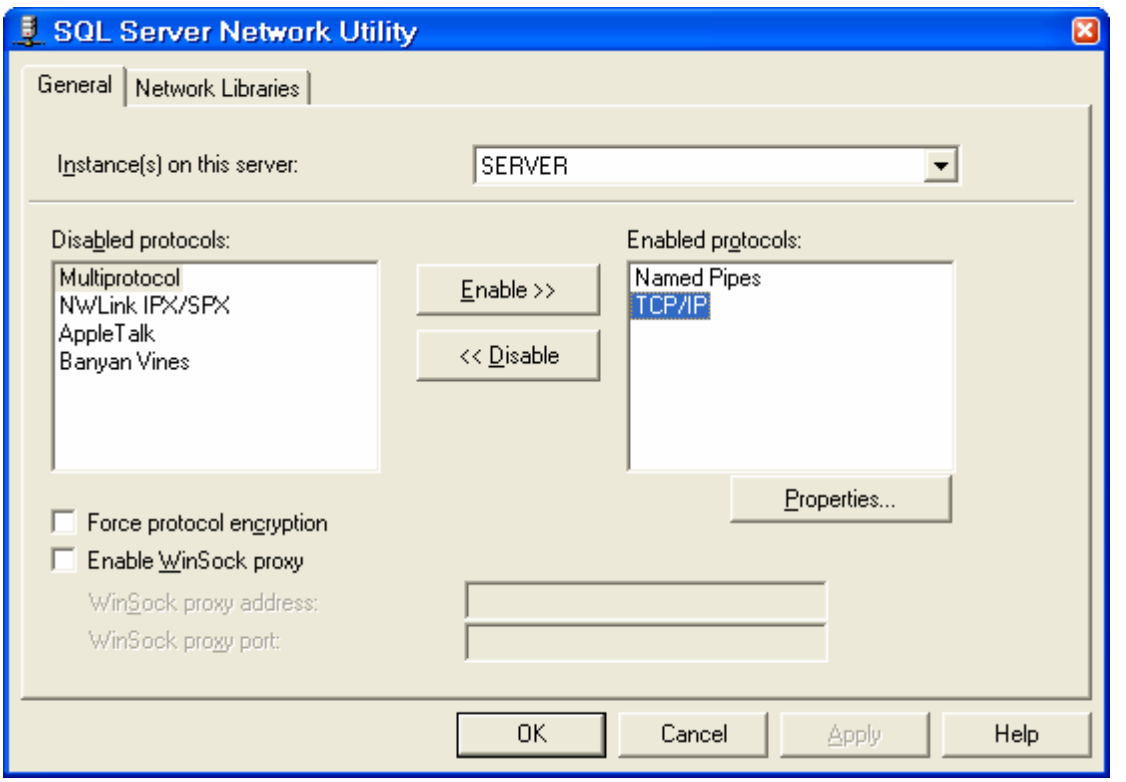
- Nhấn nút phải chuột



CẤU HÌNH KẾT NỐI MẠNG CỦA SERVER.

Để các Client hoặc các ứng dụng kết nối được đến Server, ta phải cấu hình các phương thức kết nối phù hợp với kết nối mạng. Kết nối mạng có thể sử dụng kết nối thông qua Proxy, thông qua mạng Internet.

Khi sử dụng kết nối nào đi nưa thì trước hết ta phải chọn giao thức phù hợp với giao thức mạng đạng sử dụng. Thực hiện bằng cách chọn Server network utility



- Chọn giao thức đưa vào danh sách enabled để sử dụng và đưa vò danh sách Disabled để không sử dụng.

- Chọn Properties để chọn cổng, tham số của giao thức.

- Chọn Enable WinSock proxy để thực hiện kết nối qua Proxy.

- Chọn Force protocol encryption để sử dụng kết nối qua Internet không dùng Fire Wall với SQL Server.

QUẢN TRỊ CÁC CLIENT.

Khi Server đã sẵn sàng cho kết nối, việc tiếp theo là xem xét đến các client kết nối đến server. Trong phần này ta sẽ xem xét cấu hình client kết nối đến server.

Các client kết nối đến server đều thực hiện trên cở sở hệ thống truyền tin của mạng máy tính, tuy nhiên các ứng dụng client kết nối đến server để thực hiện khai thác dữ liệu trên server thông qua một số phương thức kết nối sau:

- OLE DB: Có 2 kiểu Microsoft OLE DB Provider for SQL Server và Microsoft OLE DB Provider for ODBC.

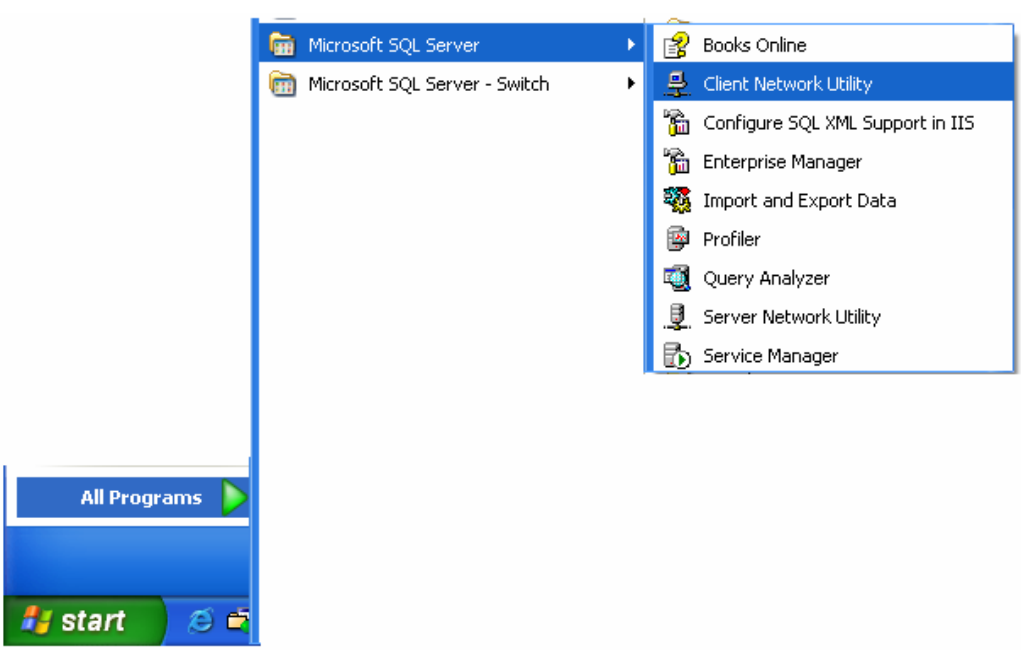
- ODBC: Kết nối thông qua SQL Server Enterprise Manager và SQL Query Analyzer sử dụng SQL Server ODBC.

- DB-Library: Sử dụng lệnh SQL Server **isql**.

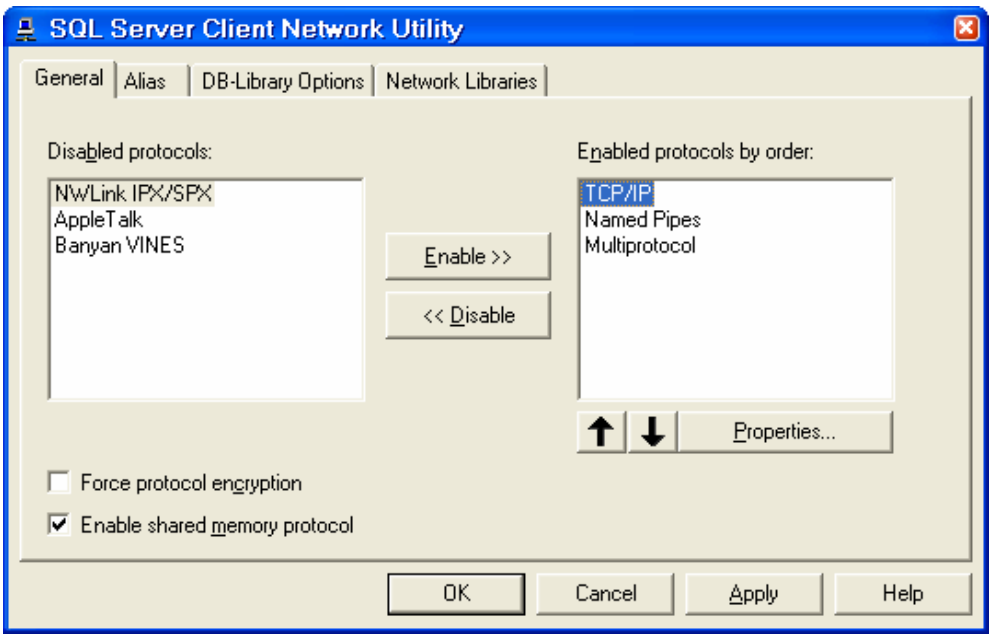
Cấu hình Net-Library.

Như đã xem xét trước mỗi Instance khi cấu hình xác định một địa chỉ và số hiệu cổng riêng, nên việc kết nối thông qua Net-Library là kết nối thông qua địa chỉ và như vậy kết nối đã xácđịnh được đến Instance.

Trên Server thông thường được cấu hình theo TCP/IP Sockets và Named Pipes Net-Libraries, trên client thông thường cấu hình theo Thực hiện cấu hình ta sử dụng Client network utility.



- Chọn Client network utility.



- Chọn giao thức và các tham số liên quan tương ứng với server, có thể thực hiện định tên Server sang tên mới trên bảng Alias.

Cấu hình ODBC.

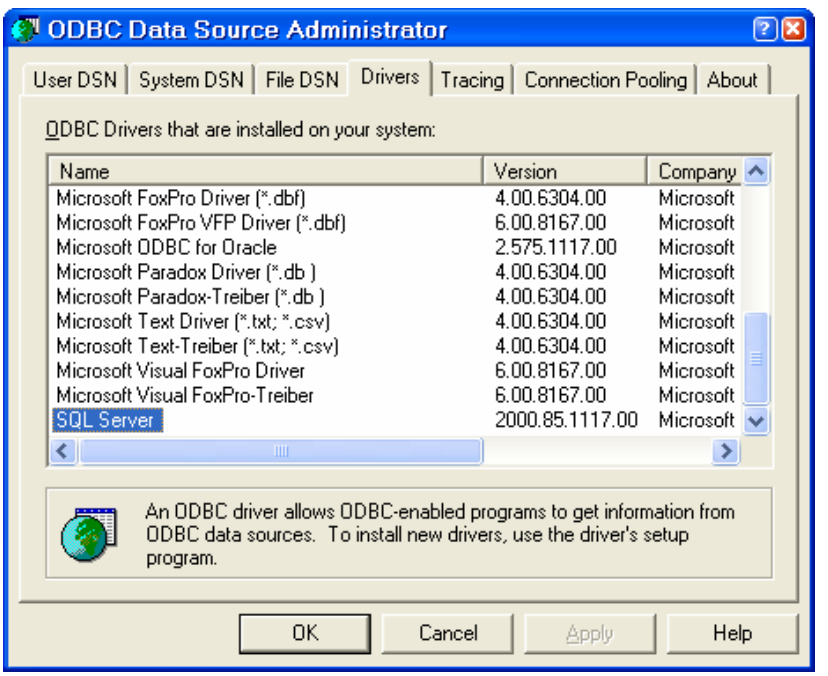
ODBC viết tắt của từ Open DataBase Connectivity, là công cụ kết nối mở. ODBC được Windows cũng cấp dẫn khi cài đặt, được sử dụng làm kết nối trung gian giữa ứng dụng và các hệ quản trị CSDL (Dbase, Access, SQL Server, Oracle,…).

Thông qua ODBC ứng dụng chỉ cần xác định tên của nguồn trong ODBC (gọi là Data Source) và tài khoản khi truy nhập để thực hiện quyền mà không cần quan tâm đến cơ sở dữ liệu đang nằm ở đâu.

Thông thường khi cài đặt hệ quản rị CSDL mới thì Windows sẽ tự cập nhật vào danh sách các Driver điều khiển ODBC của hệ quản trị CSDL đó. Thực hiện tạo ODBC cho SQL Server như sau:

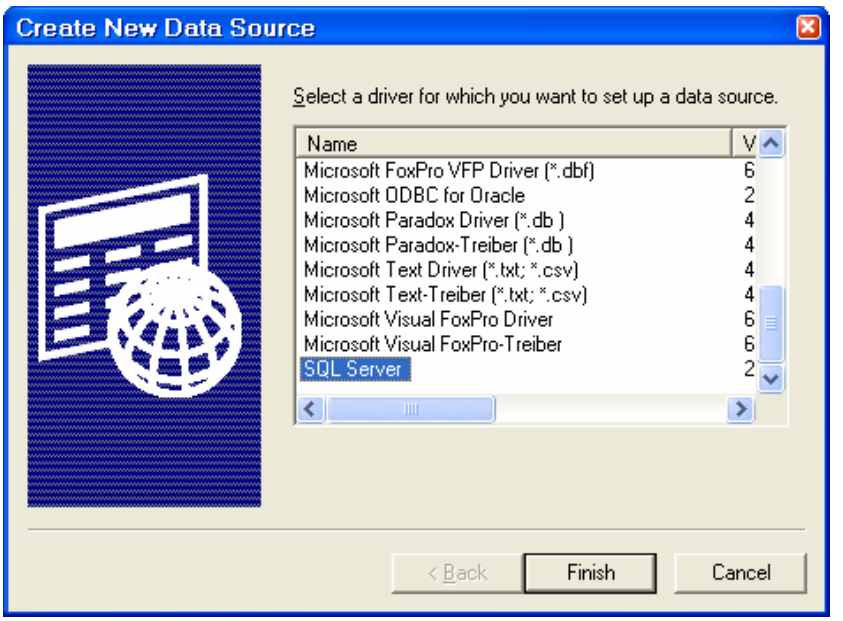
- Chọn ODBC trong Administrative tools (Control panel).

- Chọn bảng Drivers, trong danh sách kiểm tra xem đã có SQL Server chưa

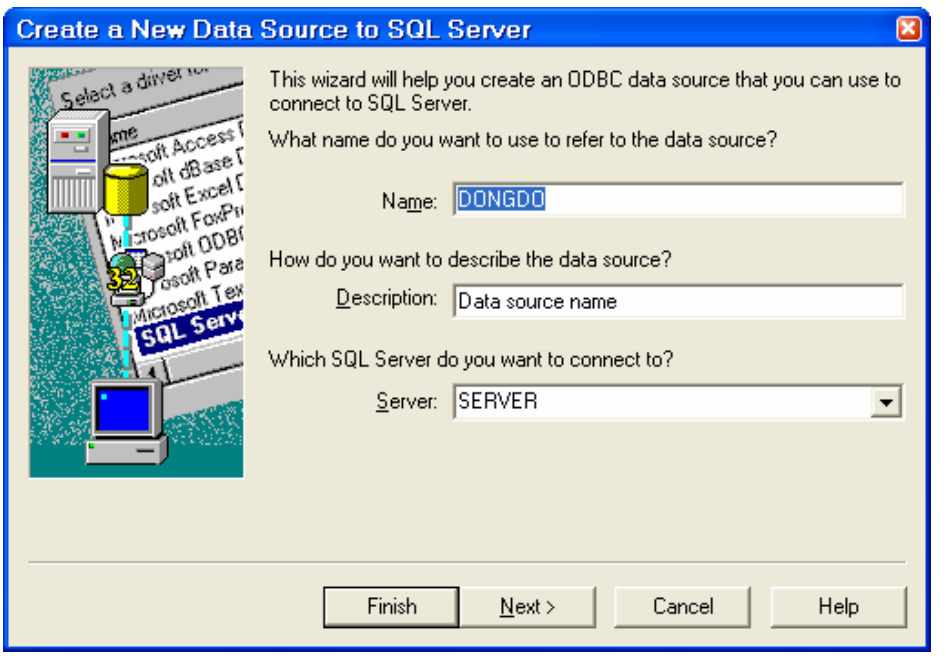


Nếu chưa có kiểm tra cách cài đặt SQL Server (thông thường Windows tự cập nhật).

- Chọn bảng User DSN (Data Source Name) -> Add.

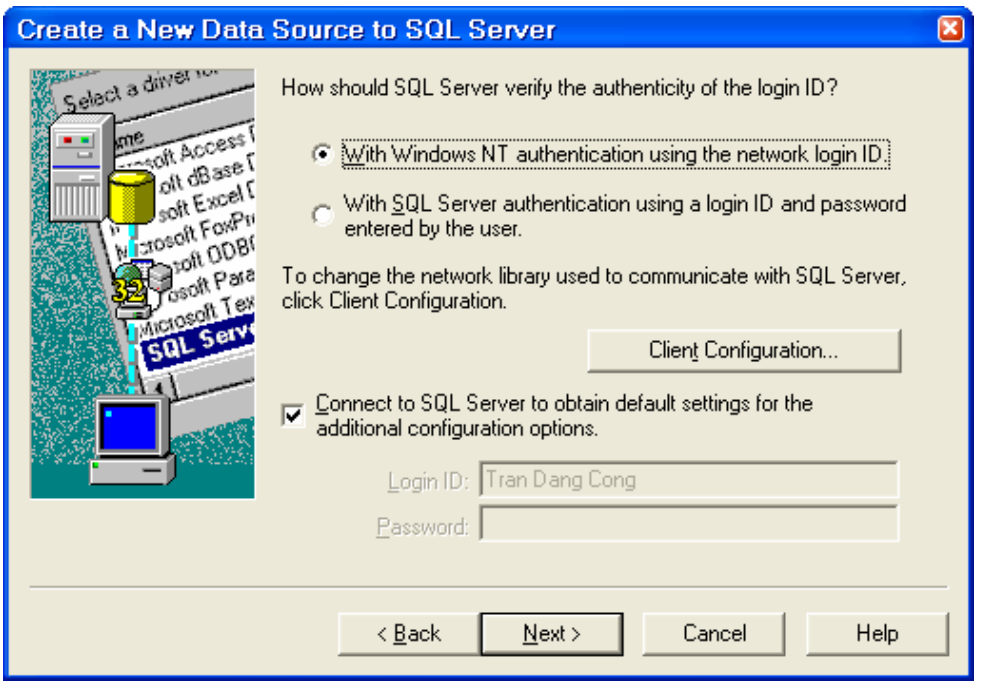


- Nhấn Finish.



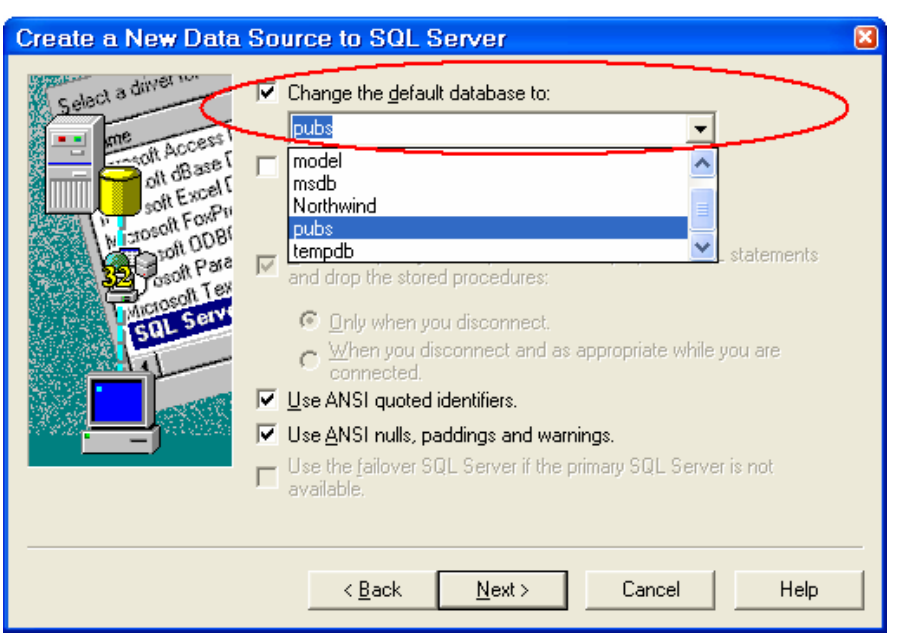
- Nhập tên DSN (đây là tên sẽ được sử dụng cho ứng dụng), thông tin mô tả, tên Server (Instance).

- Nhấn Next.

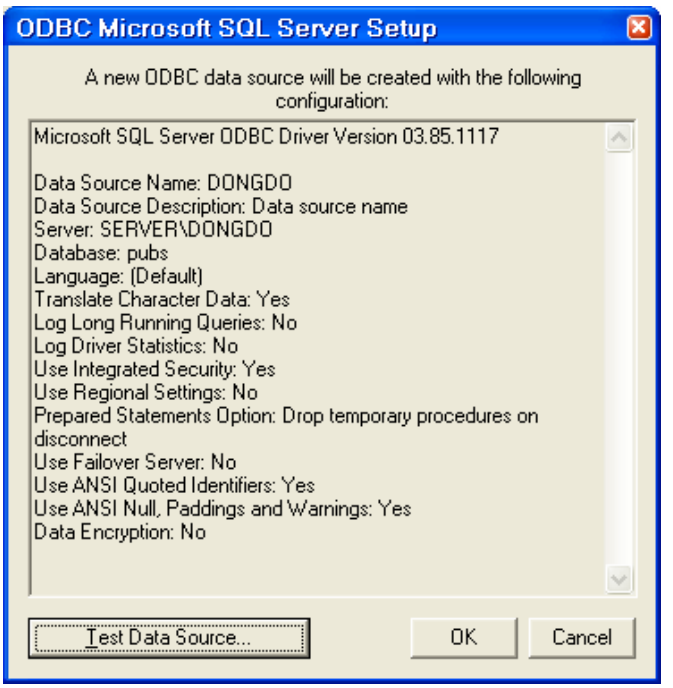


Trong cửa sổ có 2 lựa chọn: Sử dụng chế độ bảo mật kế thừa của Windows NT hoặc của SQL Server (sẽ xem xét sau), trước hết bạn hãy chọn lựa chọn kế thừa của Windows NT (lúc này quyền khai thác là quyền của người truy nhập vào Windows).

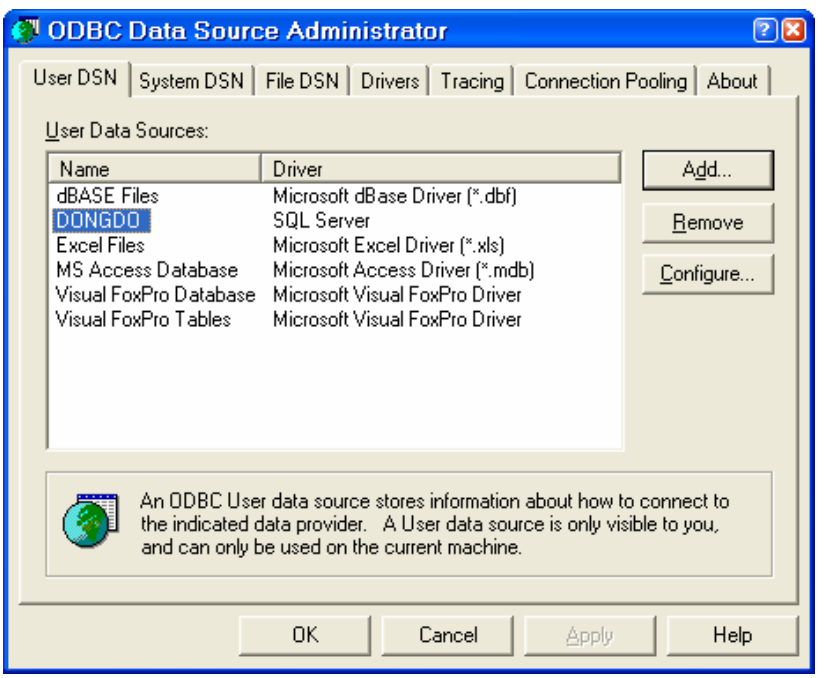
- Nhấn Next, chọn hộp chọn Change the default database to -> chọn cơ sở dữ liệu (việc tạo cơ sở dữ liệu sẽ xem xét bài sau, tại bước này bạn hãy chọn một cơ sở dữ liệu ví dụ có tên là Pubs, đây là cơ sở dữ liệu ví dụ do SQL Server tự thiết lập để làm mẫu).



- Nhấn Next -> Finish.



Để kiểm tra kết nối có thành công không bạn nhấn Test Data Source sau đo nhấn Ok để kết thúc và thu được màn hình sau:



Trên danh sách các Data Source có tên DONGDO vừa được tạo, Data Source tạo ra sẽ được sử dụng trong ứng dụng client.

Cấu hình OLE DB.

OLE DB là phương thức khá quen thuộc đối với người lập tình CSDL (lập trình trên Desktop hoặc trên Internet). OLE DB sử dụng với nhiều hệ quản trị CSDL khác nhau, mỗi hệ quản trị có cú pháp riêng và chỉ định driver điều khiển cho nó. Với SQL Server thông thường sử dụng 2 phương thức kết nối sử dụng OLE DB:

- Microsoft OLE DB Provider for SQL Server (SQLOLEDB): Không sử dụng ODBC, xác định driver cho SQL Server.

- Microsoft OLE DB Provider for ODBC: Sử dụng ODBC đã tạo (trong phần trước).

QUẢN TRỊ CƠ SỞ DỮ LIỆU

Trong chương này ta sẽ xem xét cấu trúc vật lý, tạo, xóa, sửa tham số của cơ sở dữ liệu.

CẤU TRÚC CƠ SỞ DỮ LIỆU.

Chắc hẳn khi nghiên cứu đến hệ quản trị CSDL SQL Server bạn đã xem xét đến các hệ quản trị CSDL như DBase hoặc Access, với hệ quản trị CSDL như trên mỗi cơ sở dữ liệu khi sử dụng (thực hiện mở CSDL) sẽ mở trực tiếp từ tập tin chứa CSDL, tập tin chứa CSDL sẽ có một tập tin chính (ví dụ *.dbf hoặc *.mdb) và tập tin phụ nhưng khi ta thao tác ta chỉ cần quan tâm đến tập tin chính. Nên trong các ứng dụng thông thường ta thường dùng các thao tác mở (open) để mở tập tin chính chứa CSDL và đóng (close) đóng tập tin chính chứa CSDL mà không cần quan tâm đến việc đã kết nối đến CSDL chưa (không có phương thức kết nối).

SQL Server quản lý trực tiếp các CSDL, danh sách mỗi Server sẽ gồm danh sách các tên CSDL, tên các CSDL là duy nhất, không trùng nhau. Mỗi CSDL SQL Server sẽ quản lý các cấu trúc vật lý của nó. Chính từ cách thức quản lý như trên mà việc quản trị cơ sở dữ liệu có một số đặc điểm sau:

+ Để Client khai thác CSDL trước hết phải thực hiện kết nối đến Server quản trị CSDL đó.

+ Chỉ thực hiện khai thác với các CSDL có tên trong danh sách các CSDL mà Server quản lý.

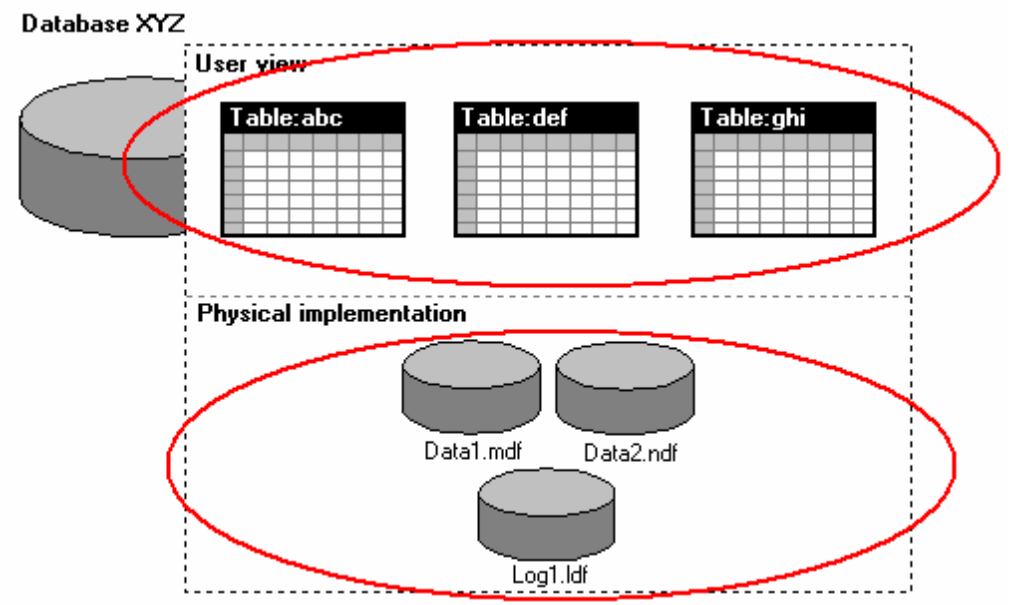
+ Không có các phương thức mở CSDL trực tiếp từ tập tin như Dbase hoặc Access.

+ Khi đã kết nối đến Server, Client chỉ thực hiện được quyền khai thác theo quy định đã định sẵn trong CSDL (phân quyền trong CSDL).

Cấu trúc cơ sở dữ liệu.

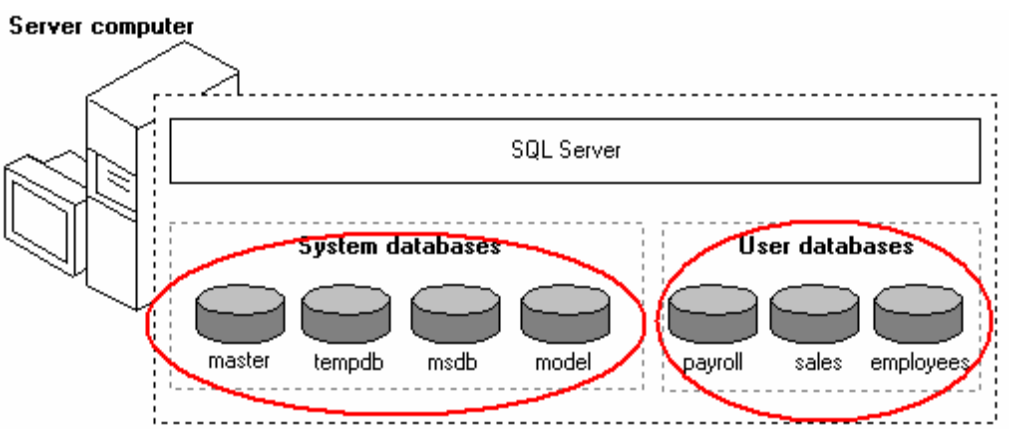
Cơ sở dữ liệu trong SQL Server lưu trữ theo 2 phần: phần dữ liệu (gồm một tập tin bắt buộc *.mdf và các tập tin phụ *.ndf) và phần nhật ký (*.ldf). Như vậy một cơ sở dữ liệu có ít nhất 2 tập tin.

Cấu trúc logic trong CSDL gồm các table, view và các object khác. Sau đây là cấu trúc một CSDL.



Sơ đồ quản trị cơ sở dữ liệu của SQL Server.

Cơ sở dữ liệu trong SQL Server chia thành 2 loại: Cơ sở dữ liệu hệ thống (do SQL Server sinh ra khi cài đặt) và cở sở dữ liệu người dùng (do người dùng tạo ta).



Cơ sở dữ liệu hệ thống gồm:

- Master: Lưu trữ các thông tin login account, cấu hình hệ thống, thông tin quản trị các CSDL, là CSDL quan trọng nên thường được sao lưu để bảo đảm an toàn cho hệ thống.
- Tempdb: Chứa các table tạm thời và các thủ tục được lưu trữ tạm thời. Các table và thủ tục nói trên được lưu trữ trong CSDL này phục vụ cho các user.

- Model: Được sử dụng khi template được sử dụng cho các CSDL được tạo trên một hệ thống.
- Msdb: Sử dụng bởi SQL Agent.

Tập tin của các CSDL nói trên như sau:

Tập tin CSDL	Tên tập tin vật lý	Kích thước ngầm định
master primary data	Master.mdf	11.0 MB
master log	Mastlog.ldf	1.25 MB
tempdb primary data	Tempdb.mdf	8.0 MB
tempdb log	Templog.ldf	0.5 MB
model primary data	Model.mdf	0.75 MB
model log	Modellog.ldf	0.75 MB
msdb primary data	Msdbdata.mdf	12.0 MB
msdb log	Msdblog.ldf	2.25 MB

Cấu trúc vật lý của CSDL.

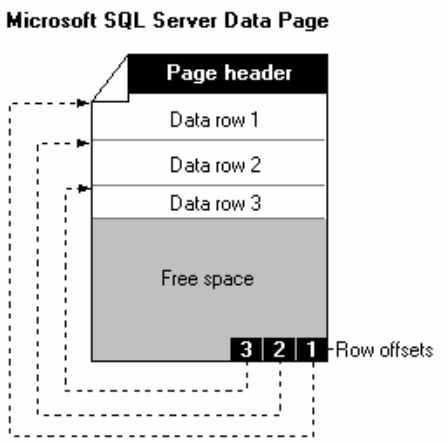
Như các cấu trúc các CSDL hệ quản trị CSDL thông thường (Dbase, Access), SQL Server cũng quản lý tập tin dữ liệu của CSDL ở dạng vật lý theo trang (page) và phân đoạn (extent).

Page.

SQL Server quản lý một page có kích thước là 8KB, như vậy 1MB có 128 page, trong mỗi trang có 96 byte chứa thông tin của trang. Có 8 kiểu page như sau:

Tên	Nội dung
Data	Chứa tất cả các kiểu dữ liệu loại trừ **text**, **ntext** và **image**
Index	Các khóa Index.
Text/Image	**Text**, **ntext**, and **image** data.
Global Allocation Map, Secondary Global Allocation Map	Chứa các thông tin định vị của các extent.
Page Free Space	Chứa thông tin khoảng trống của page.
Index Allocation Map	Chứa các thông tin về Extent đã sử dụng cho Index và Page.
Bulk Changed Map	Chứa thông thông tin về các lệnh BACKUP LOG.
Differential Changed Map	Chứa các thông tin lệnh BACKUP DATABASE.

Đối với các tập tin nhật ký (*.ldf), các bản ghi được ghi lại liên tục, không phân trang.



Dữ liệu trong một trang sẽ bắt đầu lưu trữ từ sau phần thông tin Header, và lưu trữ liên tiếp, mỗi hàng có kích thước tối đa là 8060byte. Riêng đối với dữ liệu kiểu text, ntext, image đây là kiểu dữ liệu phức tạp và có kích thước lơn, SQL Server sẽ có chiến lược quản lý khác, phân tran riêng nhằm tăng hiệu quả truy vấn dữ liệu.

Dữ liệu trong SQL Server được lưu trữ trên đĩa và tạo chỉ mục Index theo cấu trúc dữ liệu kiểu B-Tree Plus (có thể tham khảo thêm trong những nội dung cấu trúc dữ liệu nâng cao).

Extent.

Extent là đơn vị dùng chứa các table và index, mỗi extent có 8 page hay 64KB. SQL Server có 2 kiểu extent:

- Uniform: Chỉ dùng lưu trữ cho một đối tượng,.
- Mixform: Có thể dùng lưu trữ 8 đối tượng.

Cấu trúc Extent như sau:

File.

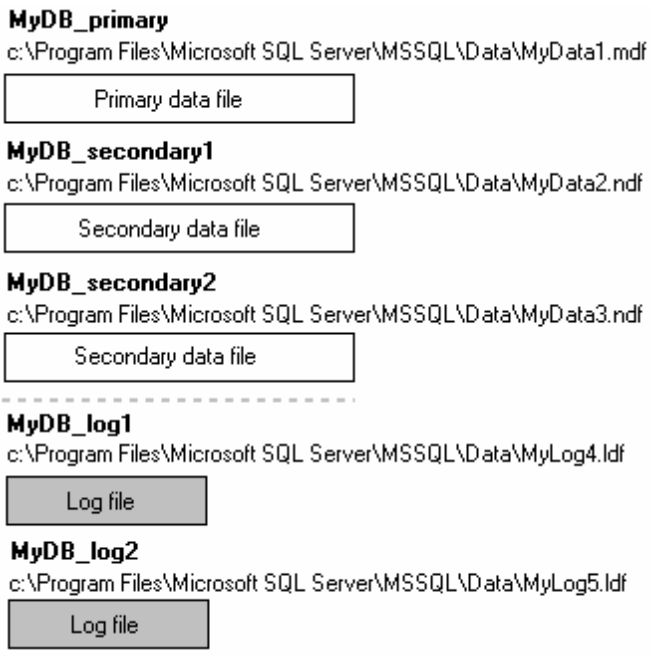
Tập tin lưu trữ một CSDL trong SQL Server có 3 loại.

Primary data file: Là file chính lưu trữ dữ liệu (*.mdf = Master Data File), mỗi CSDL có một file primary, lưu trữ điểm bắt đầu của một CSDL và các điểm kết nối đến các file lưu trữ tiếp theo (sencondary).

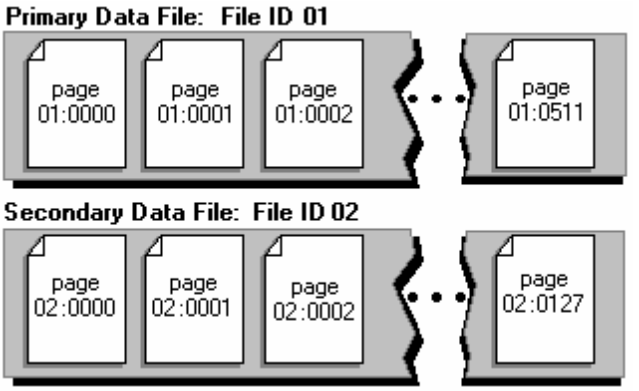
Secondary data file: Là tập tin lưu trữ dữ liệu sau Primary data file, một CSDL có thể có nhiều tập tin sencondary. Loại tập tin này cho phép một CSDL có thể phân tán dữ liệu ở nhiều nơi trên máy tính hoặc trên mạng.

Log file: Là loại tập tin lưu trữ thông tin nhật ký của CSDL.

Giả sử tạo một CSDL có tên MyDB, thông thường hệ thống ngầm định các tập tin như sau:



Các tập tin lưu trữ dữ liệu phân thành từng trang, các trang đánh số id liên tiếp theo từng file:



File group.

SQL Server sử dụng công cụ file group để giúp người dùng dễ dàng quản lý file, các file lưu trữ dữ liệu của một CSDL có thể nhóm thành từng nhóm, gồm 2 kiểu nhóm chính:

- Primary: Là nhóm bắt buộc có, dùng xác định cho file primary (*.mdf) và những file khác.
- User-defined: Nhóm do người dùng tạo ta, tự đặt tên để dễ quản lý.

QUẢN LÝ CƠ SỞ DỮ LIỆU.

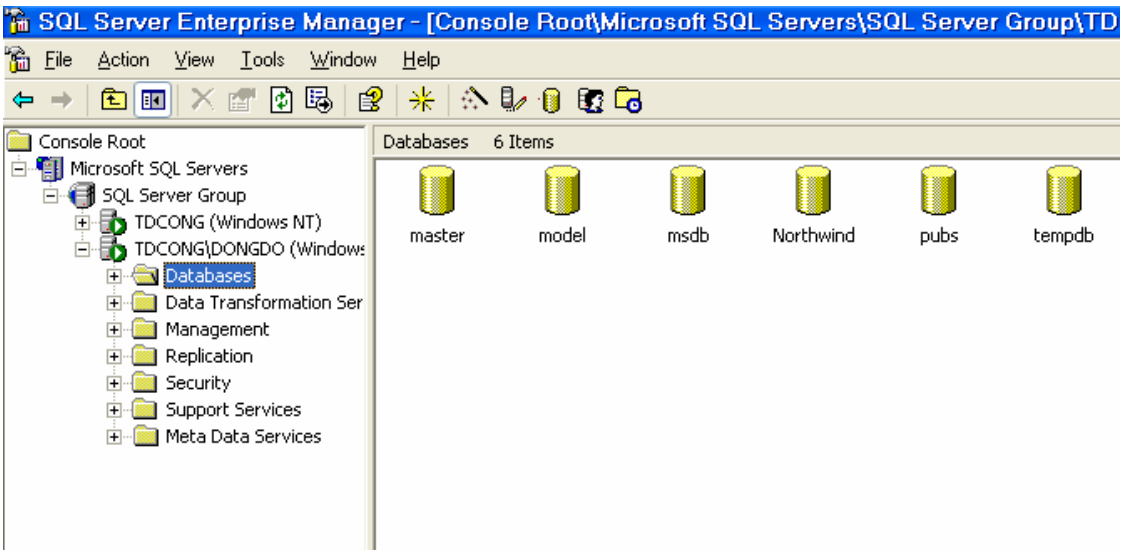
Tạo cơ sở dữ liệu.

Theo lý thuyết cơ sở dữ liệu, trước khi tạo CSDL ta phải thực hiện phân tích các thông tin liên quan mục đích sử dụng CSDL cho ài toán của mình: Tên CSDL, các table, ràng buộc,… tuân theo các chuẩn CSDL (phần này sẽ bàn kỹ trong bài sau)

Trong các thao tác với CSDL và đối tượng khác sẽ gồm 2 phần: Phần thao tác theo công cụ wizard và câu lệnh T-SQL.

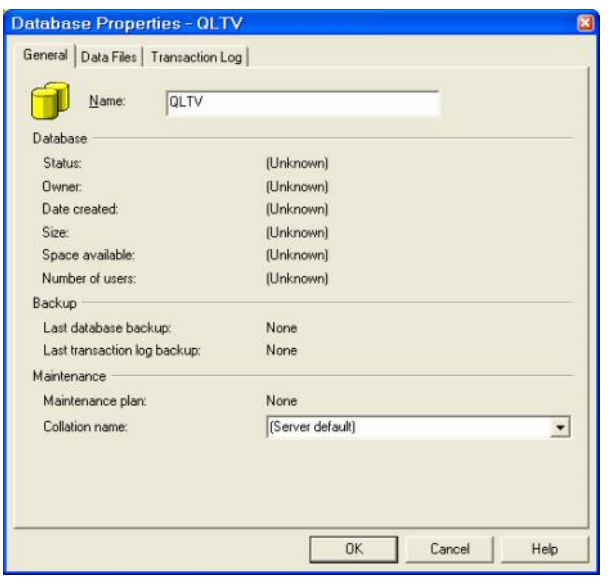
Tạo theo công cụ:

- Vào Enterprise Manager -> Databases.

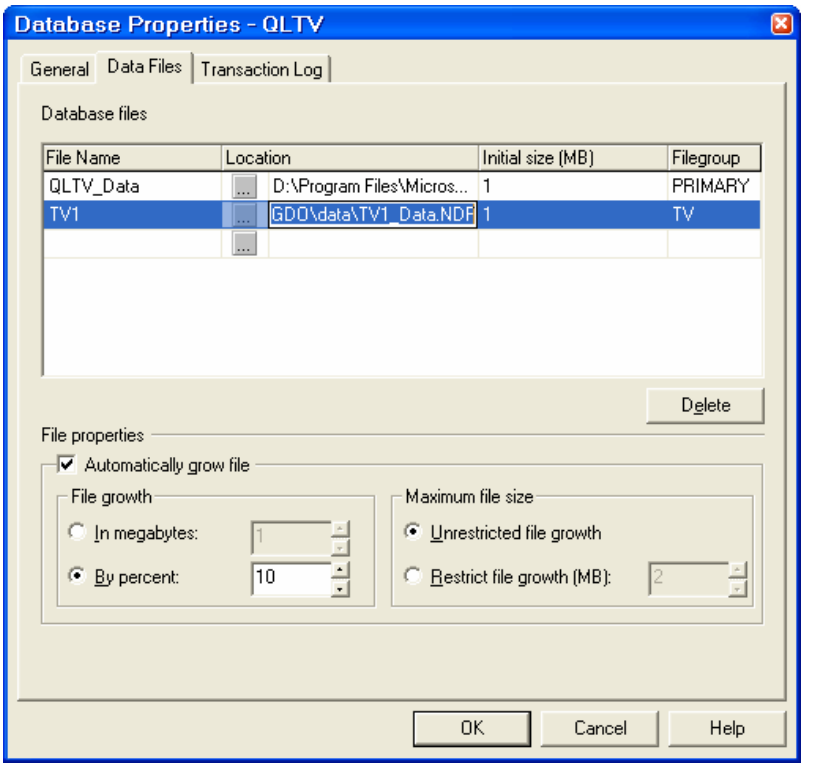


- Nhấn nút phải chuột/hoặc menu Action -> New Database…

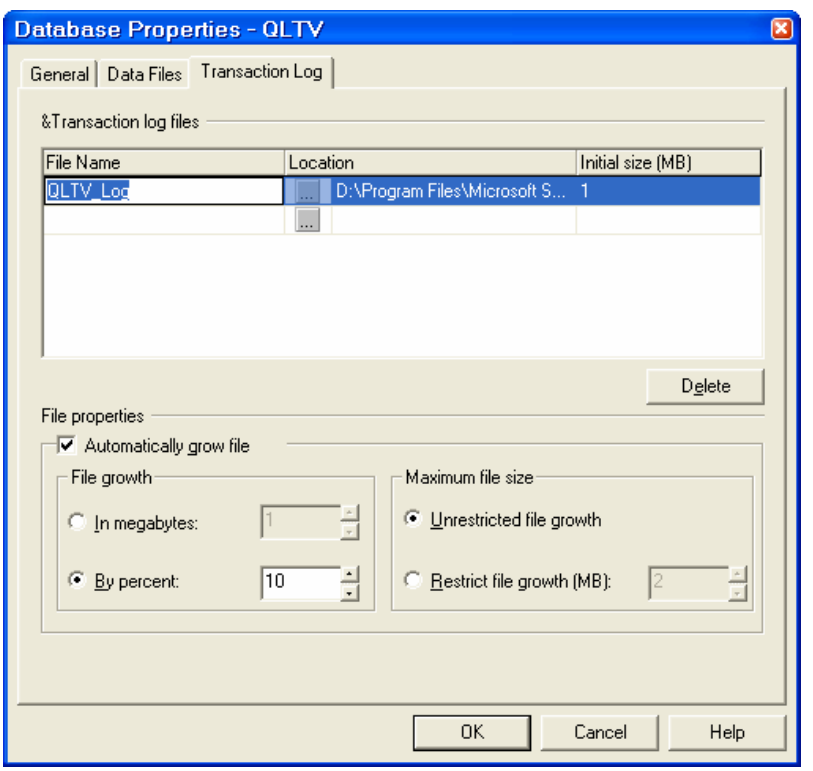
- Nhập tên CSDL.



- Xác định tên logic, tên vật lý, tên nhóm của tập tin và các tham số khác.



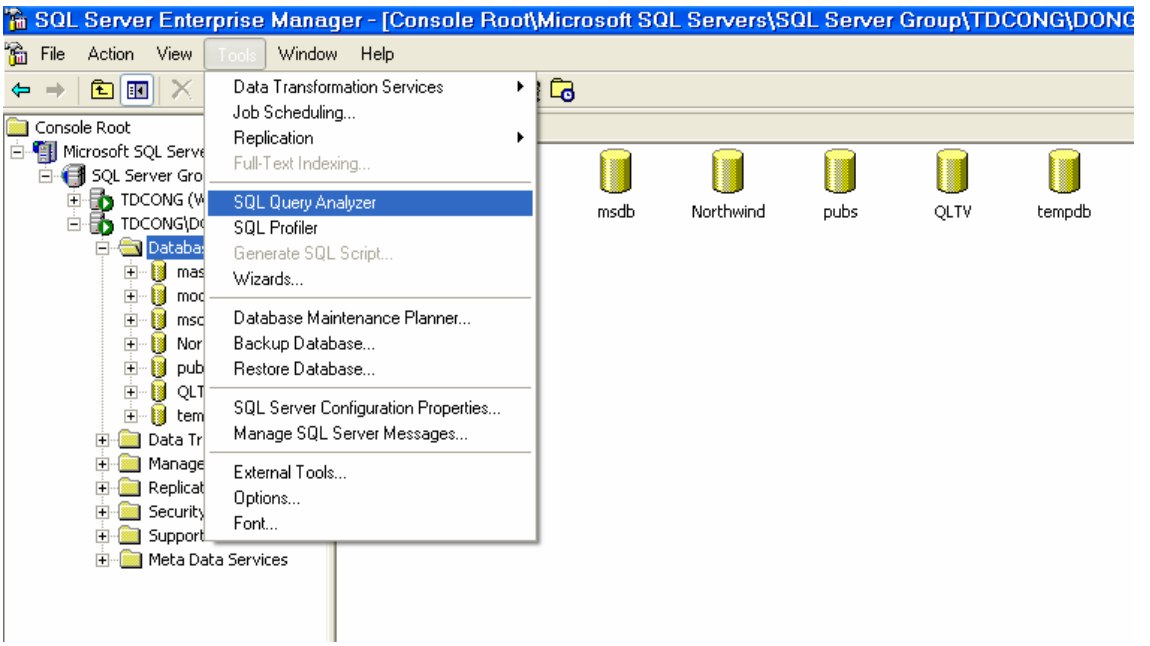
- Xác định tên logic, vật lý, tham số khác tập tin nhật ký.



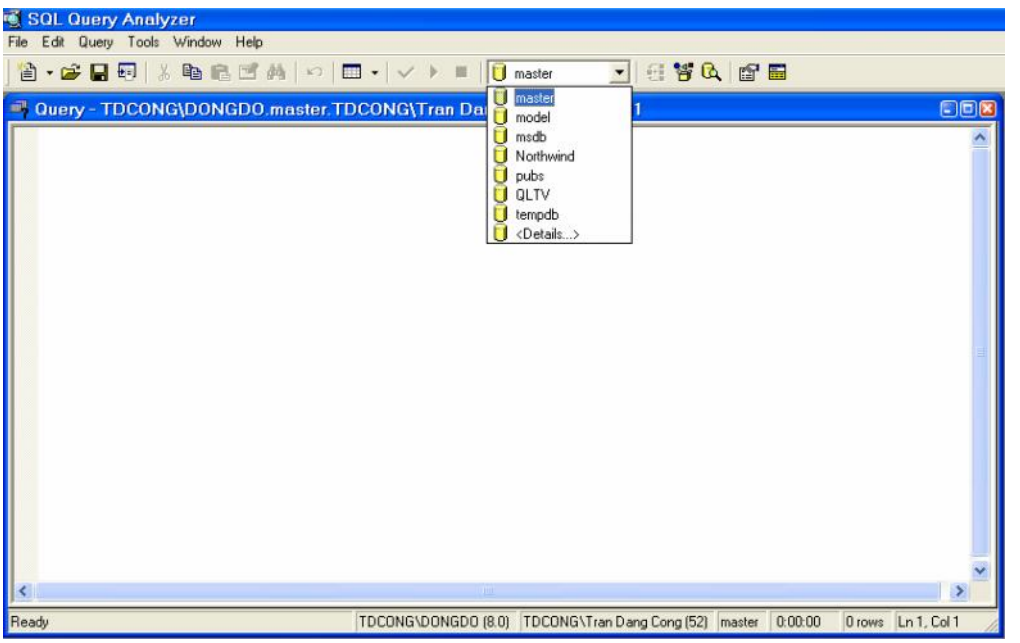
Tạo theo câu lệnh.

Sử dụng câu lệnh Create Database tạo CSDL, công cụ thực hiện lệnh:

- Trong Enterprise Manager -> Databases -> Tools -> SQL Query Analyzer



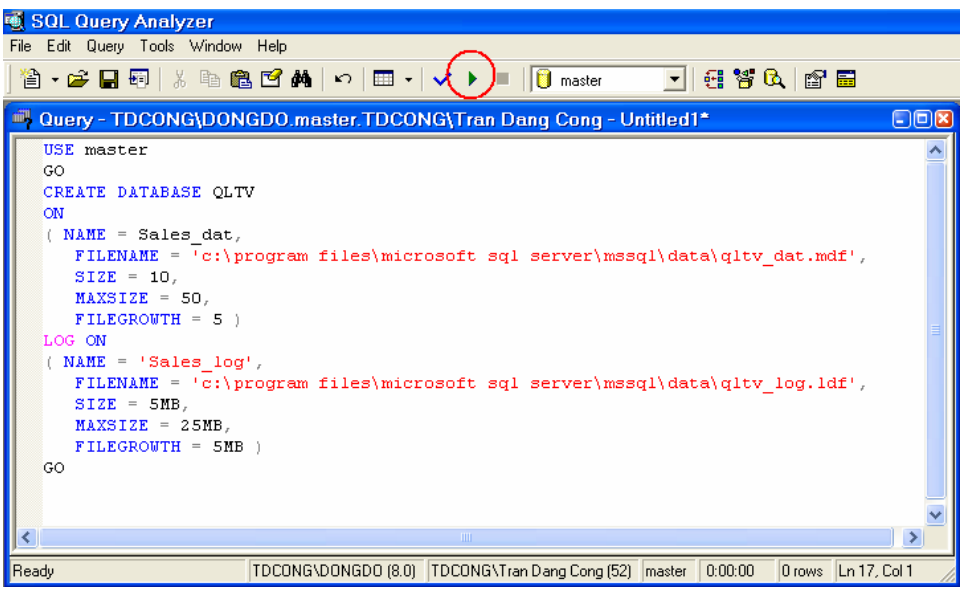
- Chọn CSDL Master.



- Soạn lệnh trong cửa sổ lệnh:

```
USE master
GO
CREATE DATABASE QLTV
ON
( NAME = Sales_dat,
   FILENAME      =      'c:\program      files\microsoft      sql
server\mssql\data\qltv_dat.mdf',
   SIZE = 10,
   MAXSIZE = 50,
   FILEGROWTH = 5 )
LOG ON
( NAME = 'Sales_log',
   FILENAME      =      'c:\program      files\microsoft      sql
server\mssql\data\qltv_log.ldf',
   SIZE = 5MB,
   MAXSIZE = 25MB,
   FILEGROWTH = 5MB )
GO
```

- Nhấn F5 hoặc nút thực hiện.



Công cụ SQL Query Analyzer cho phép bạn thực hiện từng câu lệnh bằng cách bôi đen vào đoạn lệnh cần thực hiện sau đó nhấn F5 hoặc nút thực hiện.

Khi tạo CSDL mới thì ạn phải đứng ở vị trí CSDL Master, khi muốn thực hiện lệnh với một CSDL cụ thể đã có nào đó bạn phải chọn vào CSDL đó và thực hiện lệnh.

Trong đoạn lệnh trên lệnh User Master thực hiện chọn CSDL Master bằng câu lệnh; lệnh Use xác định CSDL thực hiện.

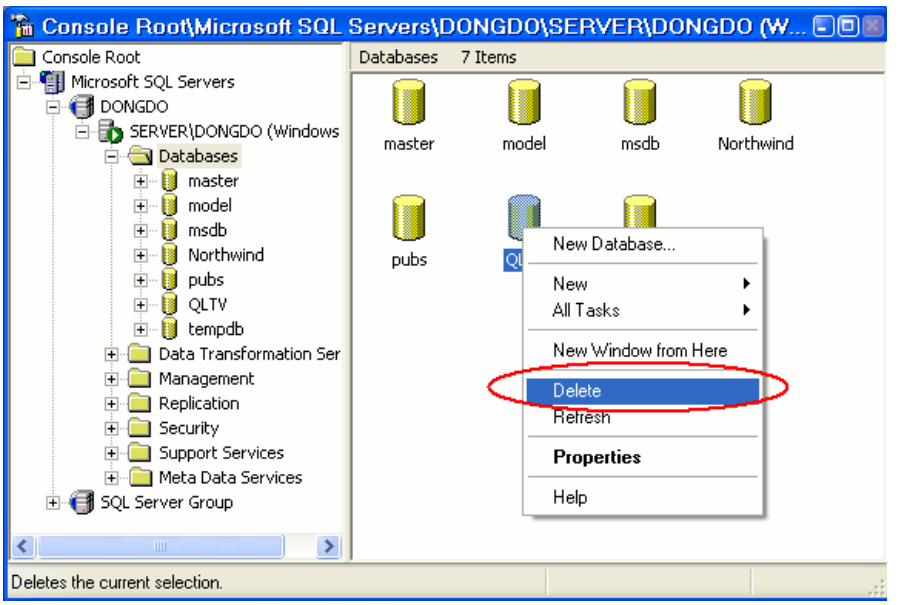
Lệnh Go xác định câu lệnh kết thúc và bắt đầu câu lệnh khác, câu lệnh được hiểu là dòng lệnh. Trong lệnh T-SQL một số lệnh khác nhau vẫn có thể nằm trên một dòng lệnh nên trong một số tình huống kich bản câu lệnh không cần sử dụng lệnh Go.

Xóa cơ sở dữ liệu.

Xóa theo công cụ.

- Chọn vào CSDL.

- Nhấn nút phải chuột -> Delete.



- Chọn Yes.

Xóa theo câu lệnh.

- Sử dụng lệnh Drop Database

```
Drop Database QLTV
```

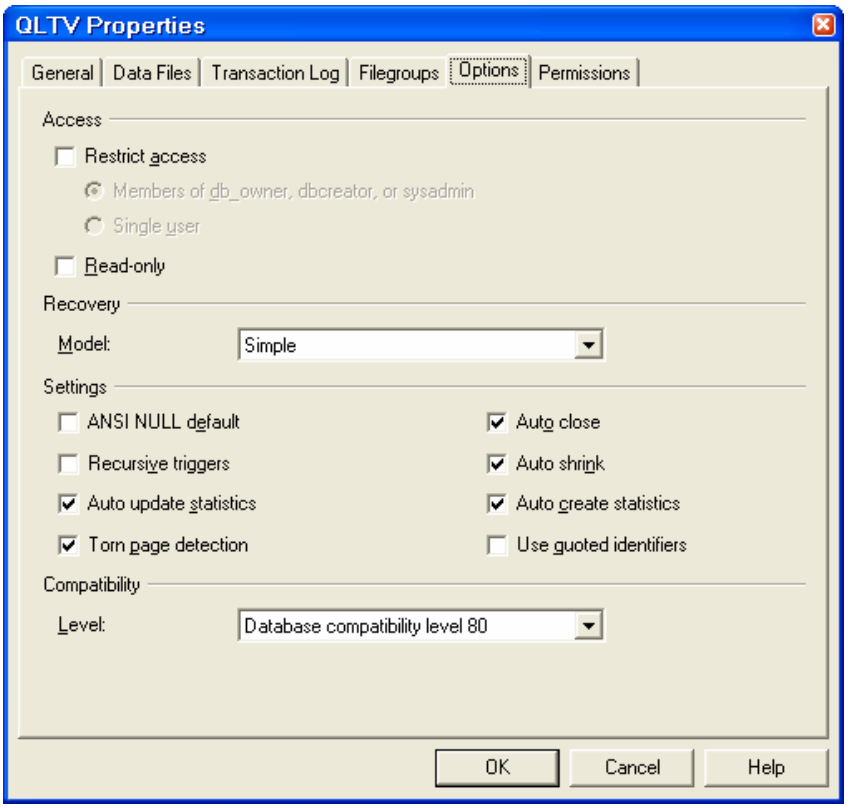
Sửa tham số.

Sửa theo công cụ.

- Chọn CSDL.

- Nhấn nút phải chuột

- Chọn Properties.



- Thay đổi tham số khi cần thiết:

+ Restrict access: Ngăn truy nhập.

+ Read only: Đặt thuộc tính chỉ đọc.

Sửa theo câu lệnh.

- Sử dụng câu lệnh Alter Database, ví dụ sau thực hiện thêm tập tin secondary vào CSDL.

```
ALTER DATABASE QLTV
ADD FILE
(
 NAME = QLTV_newfile,
 FILENAME = 'c:\Program Files\Microsoft SQL
Server\MSSQL\Data\newf.ndf',
 SIZE = 5MB,
 MAXSIZE = 100MB,
 FILEGROWTH = 5MB
)
GO
```

- Sửa tham số dựa vào thủ tục hệ thống sp_dboption:

+ Sửa đổi thuộc tính read only:

```
USE master
EXEC sp_dboption 'qltv', 'read only', 'TRUE'
```

+ Sửa thuộc tính autoshring

:

```
USE master
EXEC sp_dboption 'qltv', autoshring, TRUE
```

+ Sửa thuộc tính single user:

```
USE master
EXEC sp_dboption 'qltv', single_user
```

Mọi câu lệnh liên quan bạn có thể tra cứu, tham khảo trong Book Onlines.

BẢNG DỮ LIỆU – TABLE

CÁC CHUẨN TẮC.

Trong thiết kế cơ sở dữ liệu, việc tuân thủ ngặt ngèo những chuẩn là việc hết sức quan trọng, nó giúp cho việc quản trị dữ liệu có hiệu quả, khắc phục dư thừa, thuận lợi trong quản trị dữ liệu lớn, hiệu quả với dữ liệu phức tạp.

Gồm 3 chuẩn cơ bản:

Chuẩn thứ nhất.

Chuẩn thứ nhất xác định cấu trúc của một bảng không thể chứa các trường lặp lại.

Ta có thể lấy ví dụ như sau giả sử muốn lưu trữ thông tin một quyển sách, mỗi quyển sách có thể có một hoặc nhiều tác giả tham gia biên soạn, nếu không tuân theo chuẩn thứ nhất như nếu trên thì trong một bảng dữ liệu sách có thể có nhiều trường dữ liệu xác định thông tin tác giả.

ID	Tên sách	NXB	Tác giả 1	Tác giả 2

Trong ví dụ trên bạn nhận thấy thống tin Tác giả được lặp lại 2 lần, khắ phục bằng cách tạo ra một bảng lưu trữ danh sách tác giả của sách (sẽ bàn trong chuẩn sau).

Chuẩn thứ hai.

Chuẩn thứ hai xác định trong các hàng dữ liệu, mỗi cột đều phụ thuộc vào cột khóa chính. Ta xem xét một trường hợp vi phạm chuẩn thứ hai như sau:

Giả sử xét tình huống sinh viên mượn sách trong một thư viện, việc mượn sách được nhật ký theo bảng như sau:

Id_sach	Id_Sinhvien	Ngày mượn	Sức khỏe sinh viên

Xem xét trong bảng trên ta thấy mỗi hàng phụ thuộc vào khóa id_sach và id_sinhvien, nhưng thông tin Sức khỏe sinh viên không phụ thuộc vào id_sach, nên thông tin này cần chuyển sang bảng về thông tin của sinh viên.

Chuẩn thứ ba.

Chuẩn thứ ba xác định bản ghi tuân thủ theo chuẩn thứ hai và không có bất kỳ phần phụ thuộc chuyển tiếp nào. Phần phụ thuộc chuyển tiếp tồn tại khi một bảng chứa một cột đặc trưng. Cột này không phải là khóa nhưng vẫn xác định các cột khác.

Ta xem xét một ví dụ vi phạm chuẩn như sau:

Giả sử trong thư viện có một bảng liệt kê sách tồn trong kho, khi sinh viên mượn sách số lượng sách mà sinh viên mượn sẽ tăng, nếu nhật ký mượn sách được thực hiện theo bảng sau:

Id_sach	Id_Sinhvien	Ngày mượn	Số lượng đã mượn

Bảng trên bạn thấy mỗi lần sinh viên mượn sách số lượng sách có mã id_sach mà sinh viên có mã id_sinhvien sẽ tăng lên và tổng số là Số lượng đã mượn, thông tin này là thông tin tích lũy theo id_sach, id_sinhvien, ngày mượn.

Theo bảng trên ta thấy không vi phạm chuẩn thứ hai nhưng vi phạm chuẩn thứ ba vì cột Số sách đã mượn là cột phụ thuộc chuyển tiếp, cột này cần phải được chuyển sang bảng khác là bảng Sinh viên mượn sách:

Id_sach	Id_Sinhvien	Số lượng đã mượn

Khi nào cần chuẩn tắc.

Một cơ sở dữ liệu cần được chuẩn tắc khi:

- Dữ liệu lớn, phân tán.
- Không xác định rõ nhóm dữ liệu.
- Dữ liệu phức tạp.
- Bước đầu tiên khi xây dựng ứng dụng.

Khi nào không cần chuẩn tắc hóa.

Một số tình huống sẽ không cần chuẩn tắc hóa, nếu theo như thiết kế theo chuẩn thì việc đưa ra một mẫu tin truy vấn có thể phải thực hiện truy xuất từ nhiều bảng với nhau, điều này có nghĩa ta phải thực hiện kết hợp các bảng với nhau (tuy theo luật) nên thời gian truy xuất có thể rất lớn mà yêu cầu thực tế đặt ra trong tình

huống này là phải nhanh, thì truy xuất theo một bảng đã có sẵn là nhanh hơn, sau đây là một số trường hợp không cần chuẩn tác hóa (tùy theo tình huống):

- Thông tin tính toán.

- Thông tin sự kiện.

- Sự phân hoạch.

THIẾT KẾ BẢNG DỮ LIỆU.

Table (bảng dữ liệu) là một thành phần cơ bản của CSDL, một CSDL được thiết kế từ một hoặc nhiều bảng dữ liệu, mỗi bảng dữ liệu được cấu trúc từ các hàng và cột dữ liệu, mỗi hàng dùng mô tả một đối tượng, vấn đề, sự kiện,... cột thể hiện thuộc tính của các đối tượng, sự kiện,... của hàng. Dữ liệu cùng cột có cùng kiểu (data type). Ngoài các hàng, cột bảng còn có các khóa, liên kết, ràng buộc,...

Trước khi bắt tay vào thiết lập bảng dữ liệu trước hết ta phải xác định xem bảng sẽ xây dựng như thế nào, dựa trên một số thông tin sau:

- Kiểu dữ liệu trong bảng.

- Các cột, kiểu dữ liệu tương ứng (và độ dài nếu cần thiết).

- Cột nào cho phép giá trị NULL (là giá trị mà phần dữ liệu thuộc hàng, cột xác định không được gán giá trị nào, vì vậy nên 2 phần tử có cùng giá trị NULL là không bằng nhau).

- Giá trị ngầm định (là giá trị mà khi chưa nhập vào nó nhận giá trị này).

- Chỉ số Index, khóa chính, khóa ngoài.

Kiểu dữ liệu.

SQL Server gồm những kiểu dữ liệu sau:

Binary: Là kiểu dữ liệu chứa dạng số ở hệ hexa, gồm 3 kiểu dữ liệu Binary, Varbinary, Image.

Text: Là kiểu ký tự, chứa chữ cái, ký hiệu, số, gồm những kiểu dữ liệu sau:

- Char: Kiểu ký tự, khi xác định độ dài thì độ dài trong CSDL sẽ xác định theo độ dài đặt trước mà không theo độ dài dữ liệu thực có, không sử dụng với ký tự dạng Unicode, độ dài tối đa là 8000.

- Nchar: Tương tự như Char nhưng sử dụng với ký tự Unicode, độ dài tối đa 4000.

- Nvarchar: Tương tự như NChar nhưng kích thước trong CSDL sẽ là kích thước thực dữ liệu hiện có, không tính theo kích thước đặt trước, kích thước tối đa là 4000.

- Varchar: Tương tự như Nvarchar nhưng không hỗ trợ Unicode.

- Text: Kiểu văn bản, chứa cả ký tự xuống dòng, lưu trữ theo dạng văn bản, có kích thước lớn, có thể lên đến vài Gb, cơ chế quản lý kiểu dữ liệu theo dạng con trỏ và cách thức chèn và cập nhật sẽ khác, kiểu dữ liệu này không hỗ trợ cho Unicode.

- Ntext: Tương tự như Text nhưng có hỗ trợ Unicode.

Data/Time: Kiểu dữ liệu ngày, thời gian, ngày và thời gian, gồm 2 kiểu:

- DateTime: Đầy đủ cả ngày và thời gian.

- SmallDataTime: Chỉ ngày hoặc thời gian.

Numeric: Dữ liệu kiểu số, gồm các kiểu dữ liệu sau:

- Int, smallint, tinyint, bigint: Số nguyên

- Float, real, decimal, numeric: Số thực.

Monetary: Tiền tệ:

- Money, Smallmoney.

Bit: Kiểu số 0, 1.

Sql_variant: Là kiểu dữ liệu xác định theo kiểu dữ liệu khác, một cột dữ liệu được định nghĩa dữ liệu kiểu này có thể lưu trữ nhiều dữ liệu có kiểu khác nhau trong cùng một bảng. Ví dụ có thể lưu trữ nhiều kiểu dữ liệu **int**, **binary**, **char**, nhưng không chứa dữ liệu kiểu **text**, **ntext**, **image**, **timestamp**, **sql_variant**.

Timestamp: Là kiểu dữ liệu có kích thước 8 byte, lưu trữ dạng số nhị phân do hệ thống tự sinh ra, mỗi giá trị timestamp trong CSDL là duy nhất.

Table: Là kiểu dữ liệu đặc biệt lưu trữ tập hợp các hàng (dạng bảng), mục đích sử dụng chính là lưu trữ tạm thời tập hợp các hàng sau truy vấn.

Text in row.

Như xem xét trước, dữ liệu kiểu char, varchar có độ dài tối đa là 8000byte, dữ liệu kiểu text, ntext có 2 kiểu lưu trữ: lưu trữ trực tiếp, lưu trữ quản lý theo kiểu con trỏ.

- Đối với lưu trữ theo kiểu trực tiếp, kích thước tối đa đối với text là 8000, đối với ntext là 4000 (kích thước 1 ký tự ở mã Unicode là 2 byte, mã không Unicode là 1 byte).

- Lưu trữ, quản lý theo con trỏ kích thước lên đến GB.

Để lưu trữ dữ liệu theo kiểu con trỏ đầu tiên ta phải đặt chức năng **Text in row** về trạng thái **On**, thuộc tính này hiệu ứng cả với kiểu dữ liệu image.

Sử dụng thủ tục sp_tableoption để thay đổi thuộc tính, thuộc tính thay đổi theo bảng dữ liệu.

Giả sử bật chức năng text in row như sau:

```
Sp_tableoption N'TacGia', 'text in row', 'ON'
```

Tắt chức năng text in row như sau:

```
Sp_tableoption N'TacGia', 'text in row', 'OFF'
```

Để cập nhật dữ liệu khi thuộc tính được bật, ta phải dùng lệnh READTEXT, UPDATETEXT, WRITETEXT (sẽ bàn kỹ câu lệnh này sau).

Auto number.

Đặt cột dữ liệu kiểu số, tăng tự động khi một hàng được thêm, cột kiểu này không sửa dữ liệu. Dữ liệu kiểu này tương ứng với việc khi thêm hàng dữ liệu chèn thêm giá trị dạng số theo hàm NewID().

Ràng buộc dữ liệu.

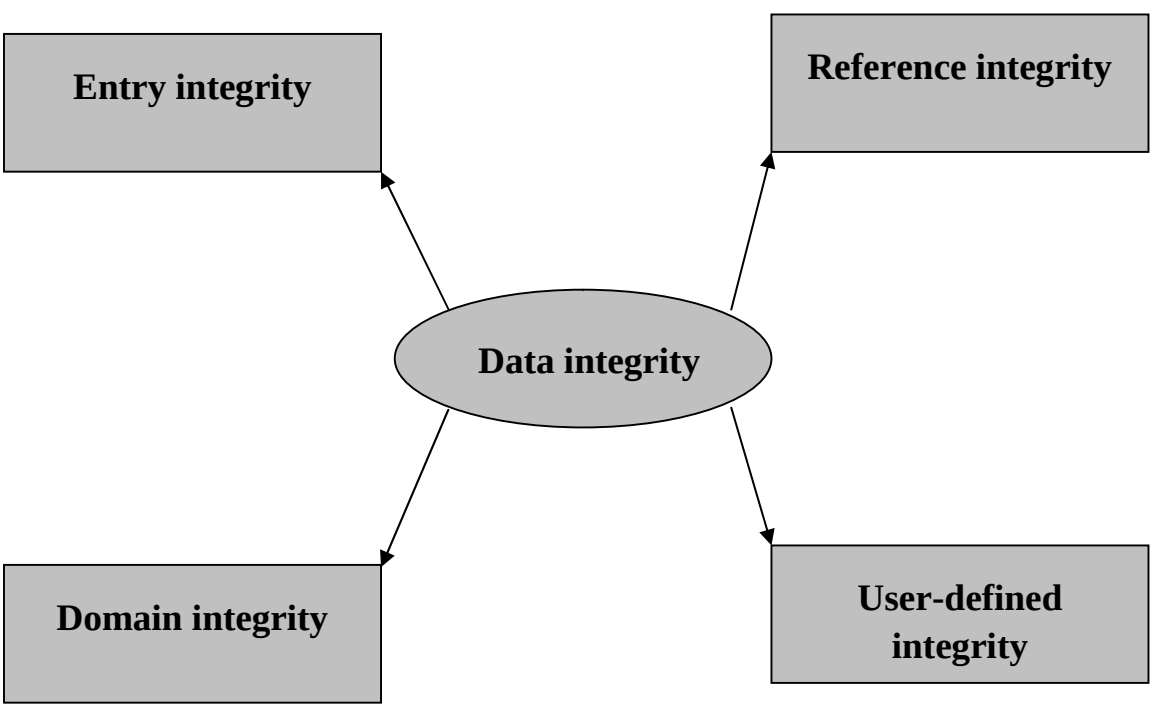
Để có một CSDL khi lưu trữ dữ liệu có độ tin cậy, độ chính xác cao, nhanh và thuận tiện trong khai thác dữ liệu thì toàn vẹn dữ liệu là vấn đề hết sức quan trọng. Khi ràng buộc được thiết lập, dữ liệu khi nhập vào CSDL sẽ được kiểm soát, độ tin cậy thông tin sẽ được bảo đảm.

Có nhiều kiểu ràng buộc dữ liệu, một CSDL có thể gồm một hoặc nhiều ràng buộc, ràng buộc có thể trên một bảng, trên nhiều bảng.

Toàn vẹn dữ liệu chia thành 4 loại:

- Toàn vẹn thực thể (Entry integrity): Mỗi thực thể đều được xác định theo một khóa, khi biết khóa ta hoàn toàn có thể xác định được thực thể tương ứng. Khóa như vậy coi là khóa chính.

- Toàn vẹn theo miền (Domain integrity): Là loại toàn vẹn có hiệu ứng với các cột dữ liệu trong một phạm vi nào đó, ví dụ kiểu dữ liệu cũng là một dạng của toàn vẹn miền, ràng buộc theo khóa check cũng là toàn vẹn theo miền.
- Toàn vẹn dạng tham chiếu (Referential integrity): Khi một bảng có quan hệ với một bảng khác theo một mối quan hệ, trong mối quan hệ đó sẽ có một khóa chính (như phần toạn vẹn thực thể) và một khóa ngoài, khóa ngoài sẽ là khóa tham chiếu của khóa chính, giá trị của khóa ngoài sẽ thuộc tập các giá trị của khóa chính hoặc giá trị NULL. Ràng buộc kiểu quan hệ (Relationship) gọi là toàn vẹn kiểu tham chiếu.
- Toàn vẹn do người dùng định nghĩa (User-defined integrity): Là toàn vẹn do người dùng định nghĩa, quy định dữ liệu nhập vào theo quy cách, giá trị được kiểm soát chặt chẽ, toàn vẹn kiểu này cũng có thể xây dựng trên cơ sở các toàn vẹn trước.



Bốn loại toàn vẹn nói trên ta có thể thống kê tương ứng với các khóa, quy tắc, ràng buộc trong SQL Server như sau:

Kiểu toàn vẹn	Công cụ trong SQL Server
Entry integrity	1. Ràng buộc Primary key 2. Ràng buộc Unique 3. Cột Identity
Domain integrity	1. Giá trị ngầm định Default 2. Ràng buộc khóa ngoài Foreign Key 3. Ràng buộc Check 4. Thuộc tính NOT NULL
Referential integrity	1. Ràng buộc Forelgn Key 2. Ràng buộc Check
User-defined integrity	1. Rules 2. Stored procedures 3. Triggers

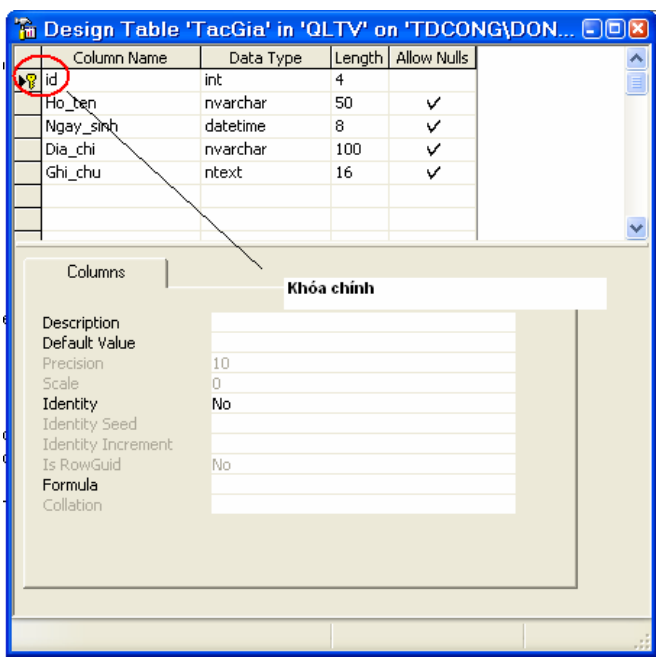
Các khóa.

Khóa chính – Primary Key.

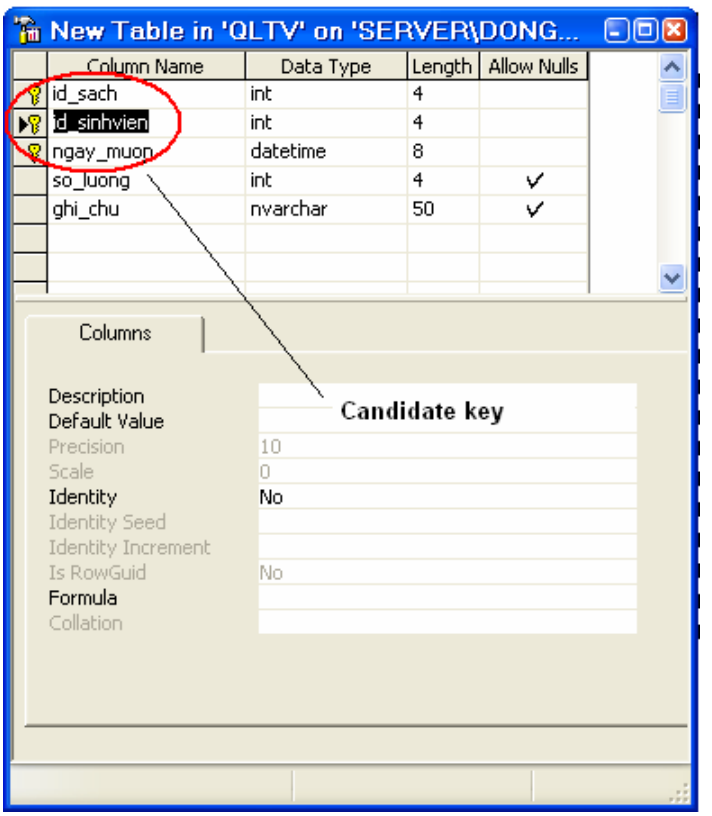
Là một hoặc tổ hợp nhiều cột dữ liệu xác định duy nhất trong một bảng, giá trị khóa chính luôn khác NULL.

Ví dụ: Bảng danh sách tác giả viết sách.

Trong ví dụ trên bảng dữ liệu có khóa chính là một cột dữ liệu id, khi cột xác định là khóa chính bên cạnh xuất hiện biểu tượng chìa khóa, thuộc tính Allow Nulls không được đanh dấu.



Ví dụ: Bảng dữ liệu lưu trữ thông tin nhật ký mượn sách.



Trong ví dụ trên bảng dữ liệu có khóa chính được tổ hợp từ 3 cột dữ liệu id_sach, id_sinhvien, ngay_muon, ba cột trên xác định duy nhất một sinh viên được mượn một loại sách trong một ngày (giả sử quy chế xác định như vậy), các cột tham gia khóa chính gọi là candidate key.

Khóa ngoài.

Theo chuẩn thiết kế CSDL, khi lưu trữ thông tin sách phải có một cột chứa thông tin nhà xuất bản. Một nhà xuất bản có thể xuất bản nhiều quyển sách và một quyển sách chỉ xuất bản ở một nhà xuất bản. Nên trong thiết kế ta phải có:

+ Bảng dữ liệu lưu trữ danh sách các nàh xuất bản: Có khóa chính đại diện cho nhà xuất bản.

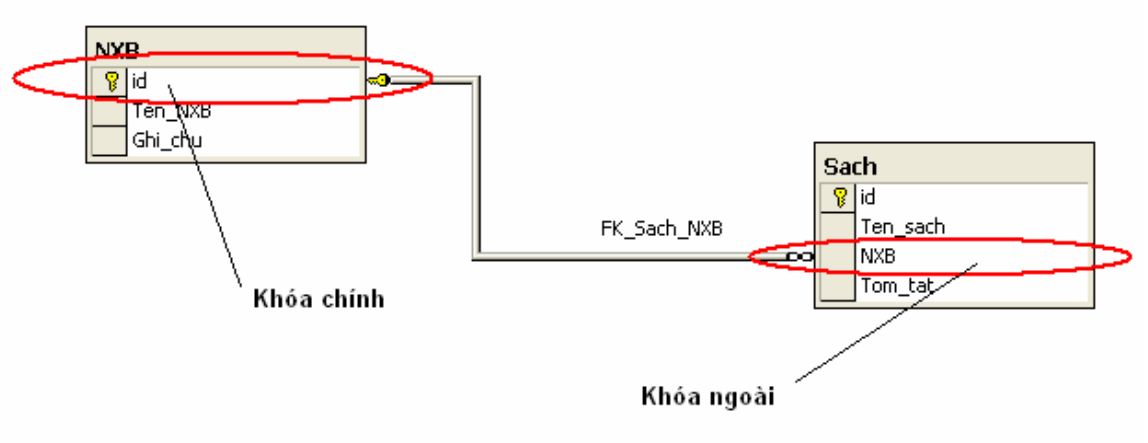
+ Bảng dữ liệu lưu trữ sách: Có chứa thông tin nhà xuất bản.

+ Quan hệ giữa nhà xuất bản và sách: Mã khóa nhà xuất bản thuộc bảng nhà xuất bản và thông tin nhà xuất bản thuộc bảng sách, cột thông tin nhà xuất bản thuộc bảng sách tham gia quan hệ trên gọi là khóa ngoài (Foreign key).

Cột dữ liệu là khóa ngoài có thể có quan hệ với nhiều khóa chỉnh ở nhiều bảng, một bảng có thể có nhiều khóa ngoài, khóa ngoài có thể có giá trị NULL, giá

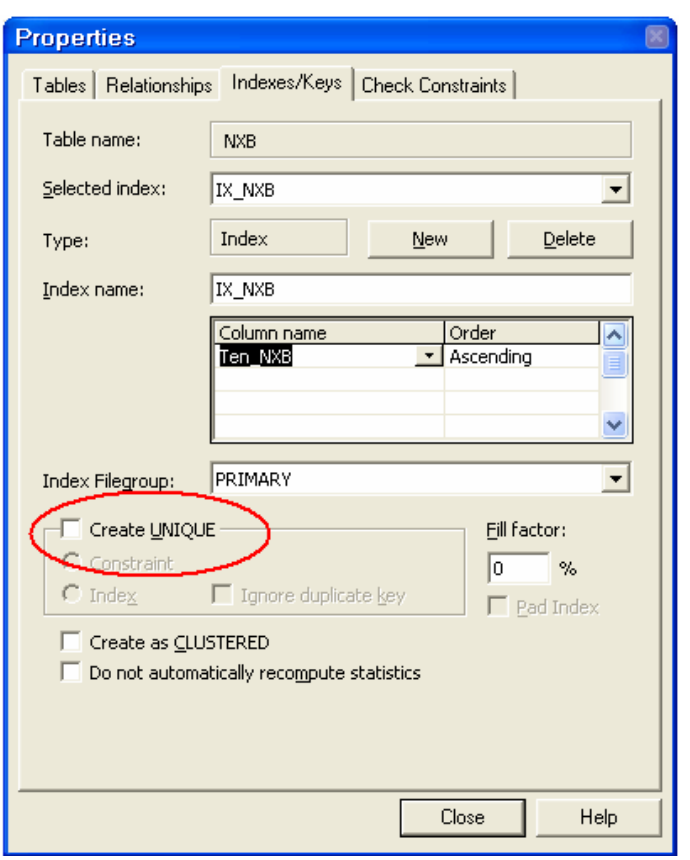
trị của khóa ngoài luôn nằm trong tập giá trị của khóa chính trong mối quan hệ đã thiết lập.

Khóa ngoài và khóa chính phải có cùng kiểu dữ liệu, cùng kích thước.



Ràng buộc Unique.

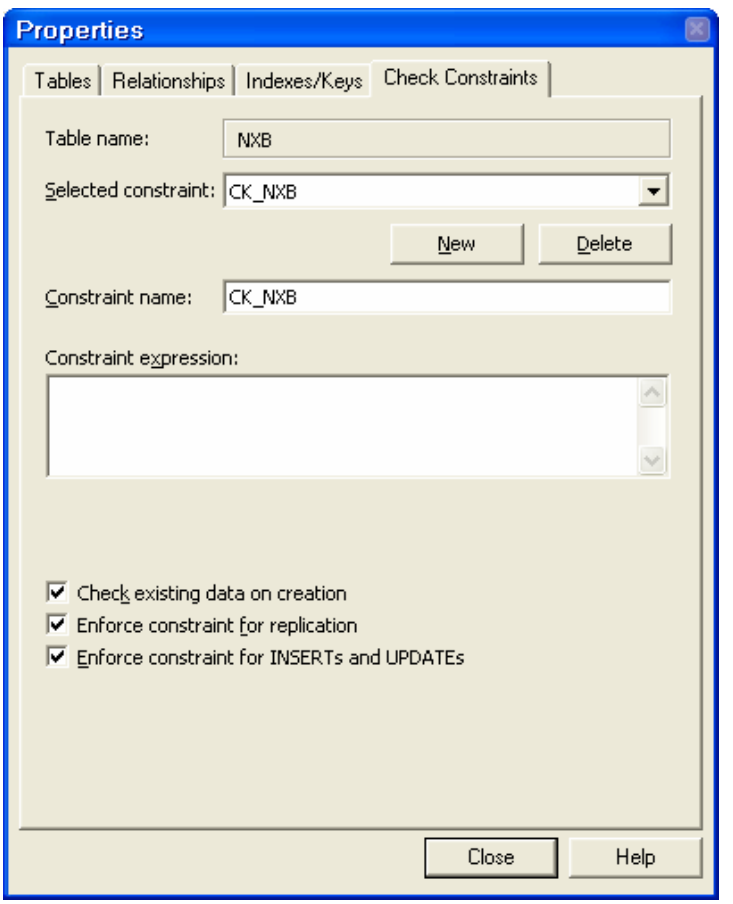
Unique là ràng buộc xác định trên một hoặc tổ hợp cột dữ liệu, cột hoặc tổ hợp cột dữ liệu được xác định ràng buộc loại này là duy nhất.



Một bảng dữ liệu có thể có nhiều ràng buộc duy nhất, một cột trongmrang buộc loại này cho phép nhận giá trị NULL, rang buộc duy nhất có thể sử dụng làm tham chiếu cho khóa ngoài.

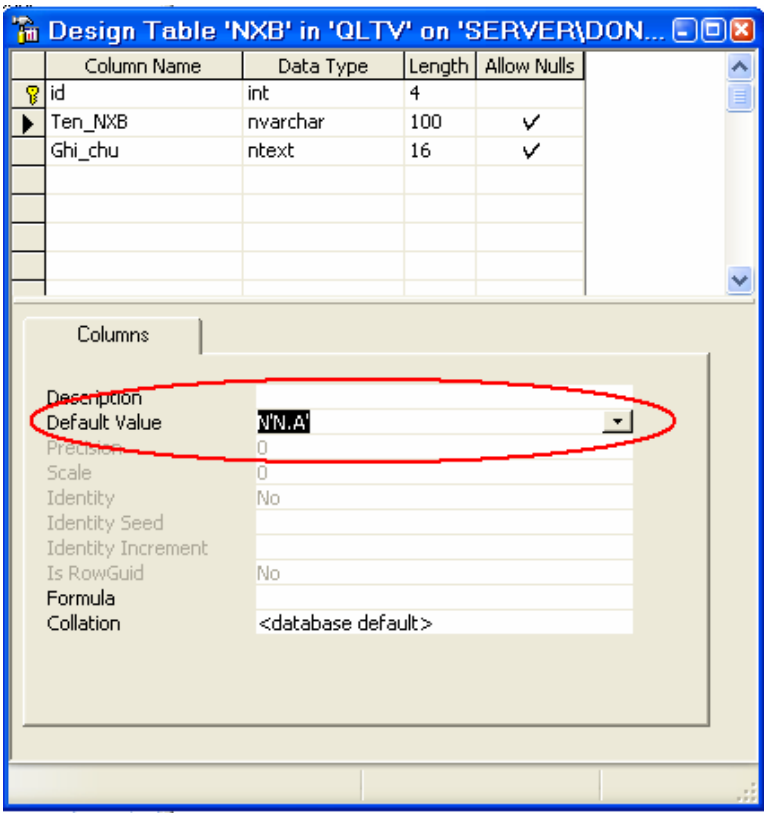
Ràng buộc Check.

Là ràng buộc khống chế dữ liệu nằm trong một phạm vi nào đó. Ràng buộc này sẽ kiểm tra dữ liệu khi nhập vào.



Giá trị ngầm định – Default.

Giá trị gán cho cột dữ liệu khi thêm bản ghi và chứa nhập dữ liệu vào cột này.



TẠO BẢNG DỮ LIỆU.

Sau khi đã xác định đầy đủ các thông tin thiết kế CSDL, bước tiếp theo là thực hiện tạo cấu trúc CSDL. Để tạo cấu trúc CSDL bước quan trọng là tạo bảng dữ liệu.

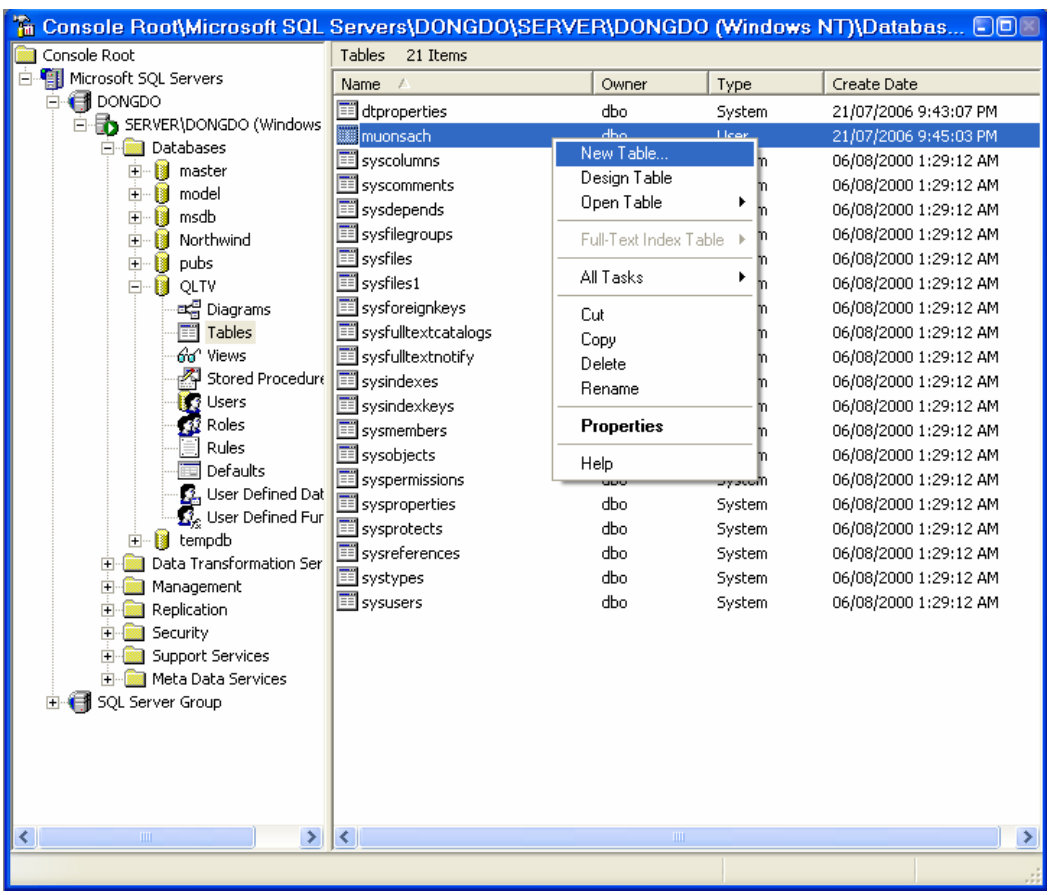
Khi tạo CSDL hệ thống tự động tạo ra một số bảng dữ liệu ngầm định, các bảng dữ liệu này sẽ cung cấp, quản lý thông tin quản trị của CSDL, cung cấp một số hàm hệ thống trợ giúp bgười dùng.

Tạo bằng công cụ.

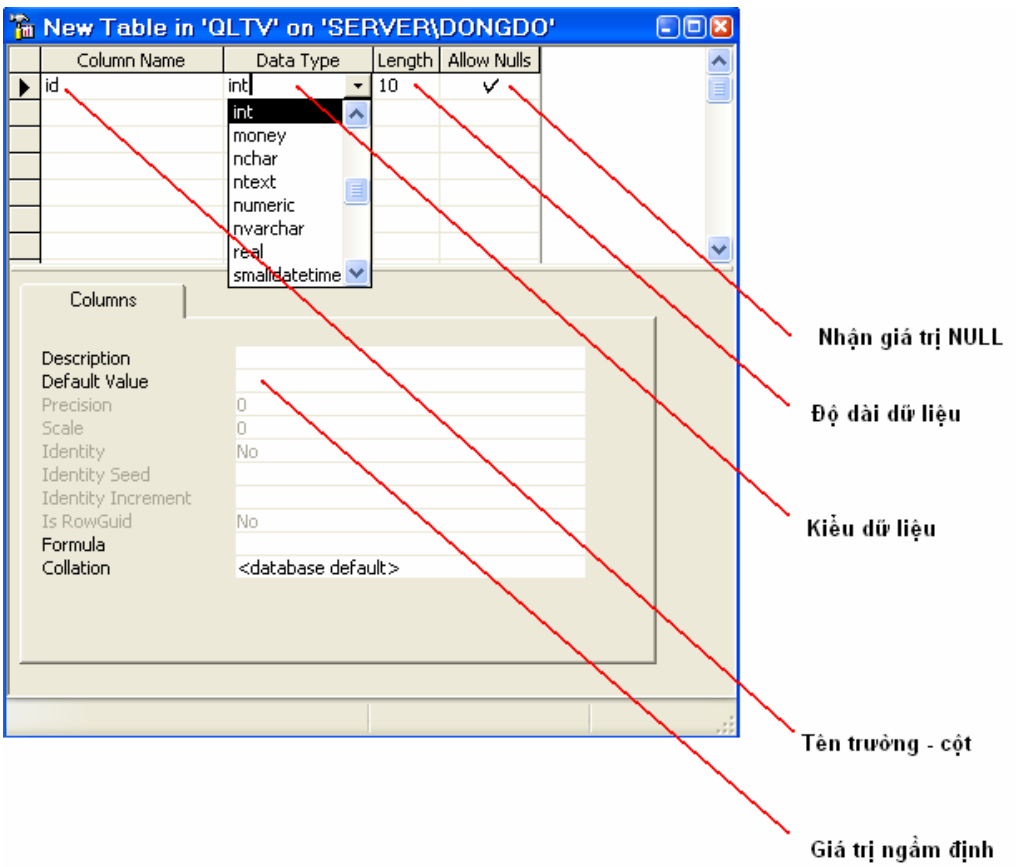
- Chọn CSDL

- Chọn Tables

- Nhấn phải chuột ở cửa sổ bên phải

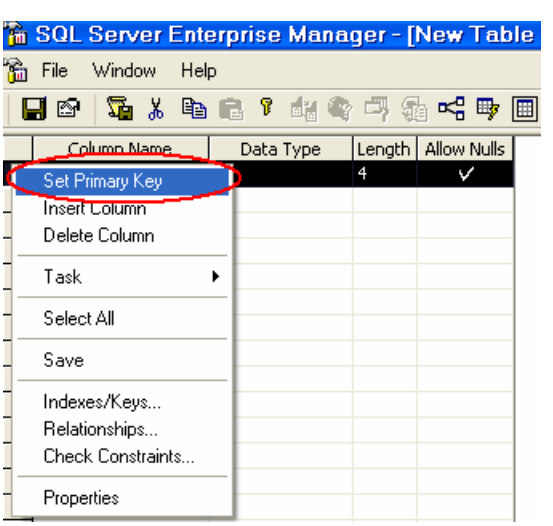


- Chọn New Table.



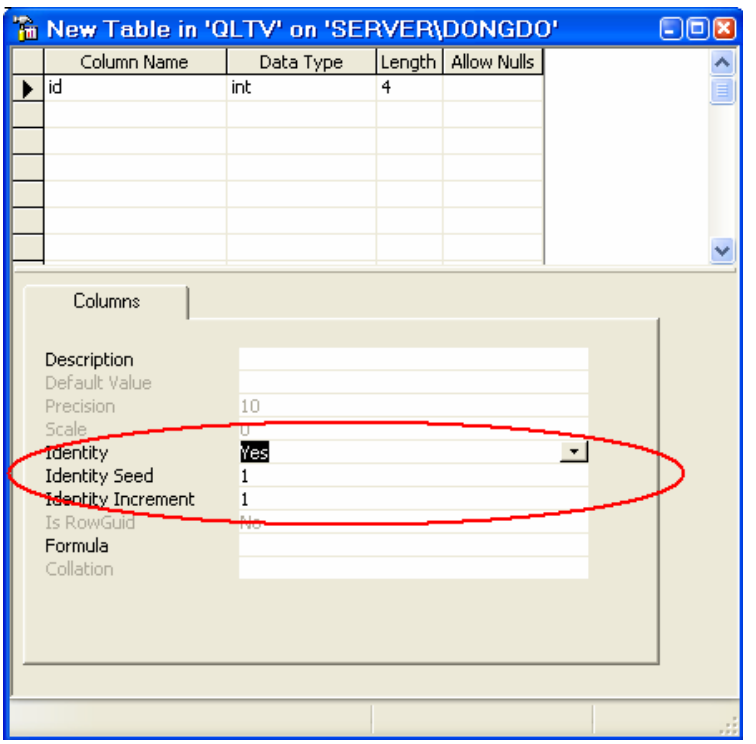
Đặt khóa chính.

Để xác định khóa chính ta thực hiện chọn những cột tham gia khóa bằng cách giữ phím shift và chọn chuột -> nhấn chuột phải -> chọn Set primary key.



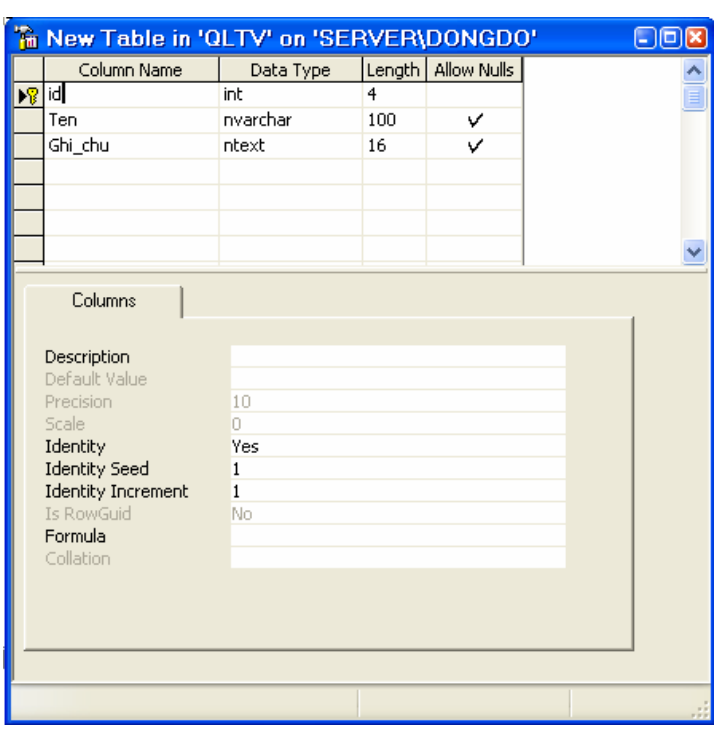
Xác định Identity.

- Chọn cột dữ liệu -> Chọn yes trong mục Identity -> đặt seed (giá trị khởi đầu) -> đặt increment (bước tự động tăng).



Tạo bảng bằng câu lệnh.

Giả sử cần tạo bảng tên NXB có cấu trúc như sau:



Sử dụng lệnh Create table, kịch bản câu lệnh như sau:

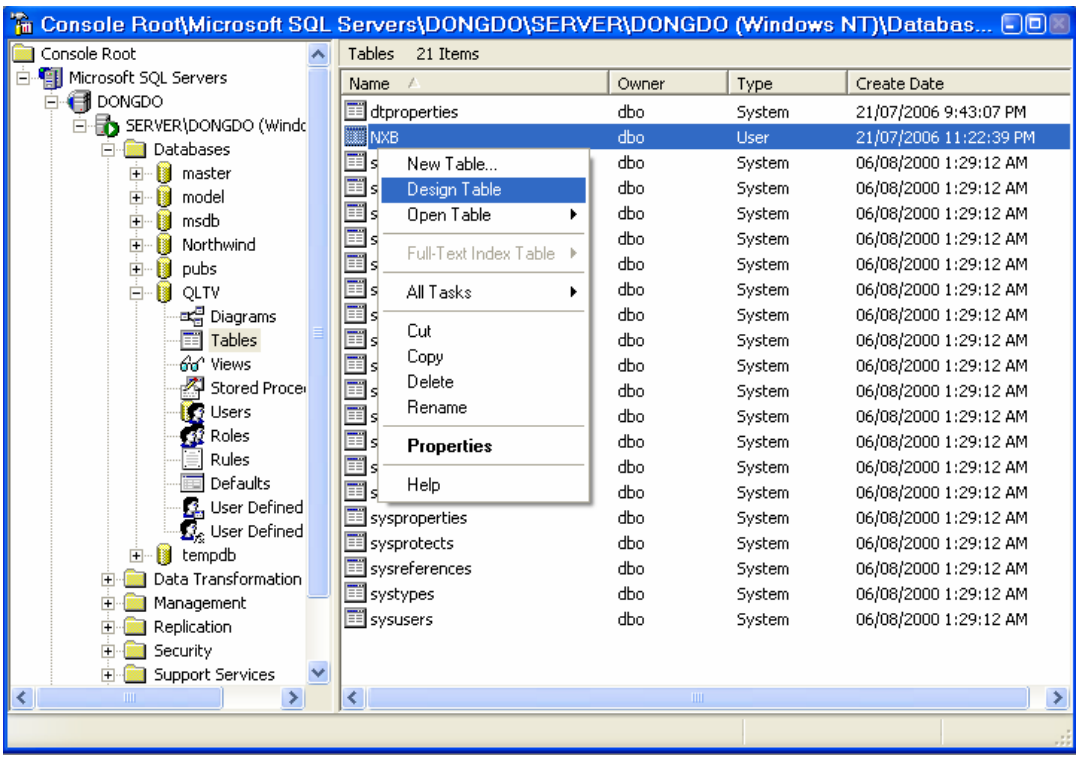
```
Create table NXB(id int not null primary key identity(1,1), Ten Nvarchar(100), Ghi_chu Ntext)
```

Sửa cấu trúc bảng.

Sử dụng công cụ.

- Chọn bảng cần sửa đổi của CSDL.

- Nhất phải chuột -> chọn Design Table.



- Thực hiện sửa cấu trúc bảng.

Sửa dụng câu lệnh.

Để sửa cấu trúc bảng dữ liệu ta sử dụng câu lệnh Alter table.

- Thêm một cột vào bảng đã có:

```
ALTER TABLE NXB ADD Dia_chi NVARCHAR(100) NULL
```

- Xóa cột từ bảng đã có.

```
ALTER TABLE NXB_Drop column_Dia_chi
```

Xóa bảng.

Sử dụng công cụ.

- Chọn bảng

- Nhất chuột phải

- Chọn Delete -> Yes.

Sử dụng lệnh. (Drop Table)

```
Drop Table NXB
```

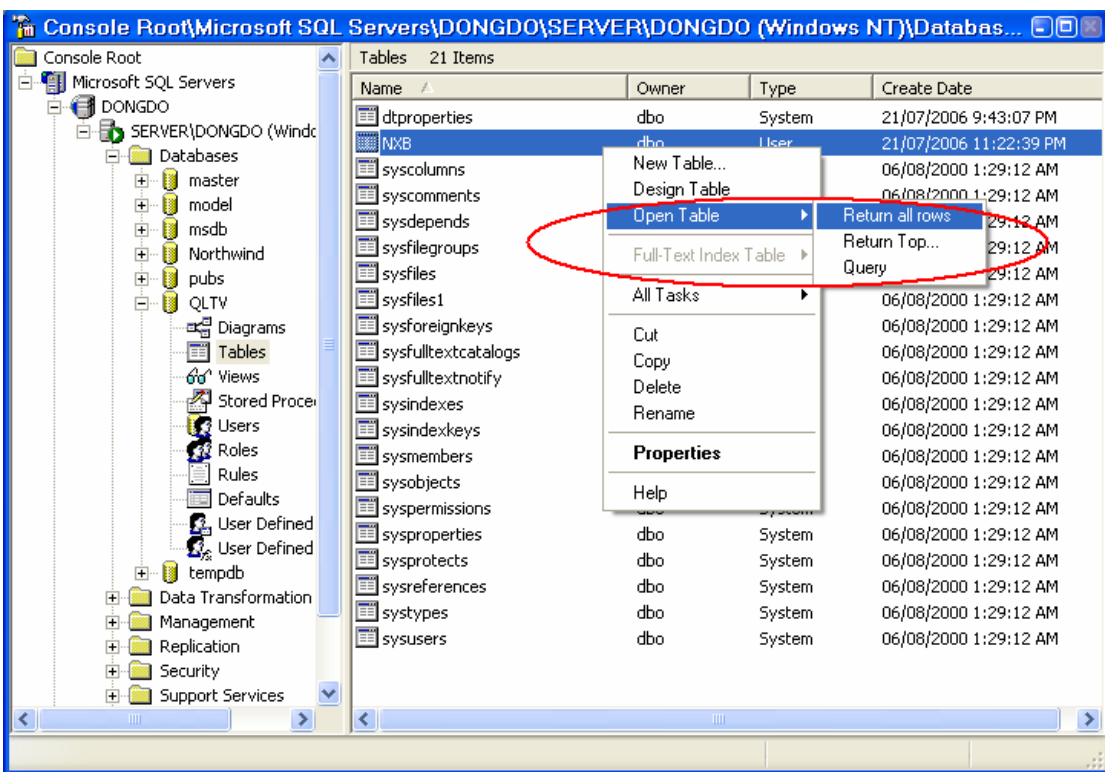
Bảng dữ liệu có tham gia mối quan hệ Relationship khi xóa bạn cần chú ý: Nếu bảng chứa khóa ngoài thì việc xóa thực hiện bình thường, nếu bảng chứa khóa chính của mối quan hệ thì không xóa được.

Nhập dữ liệu vào bảng.

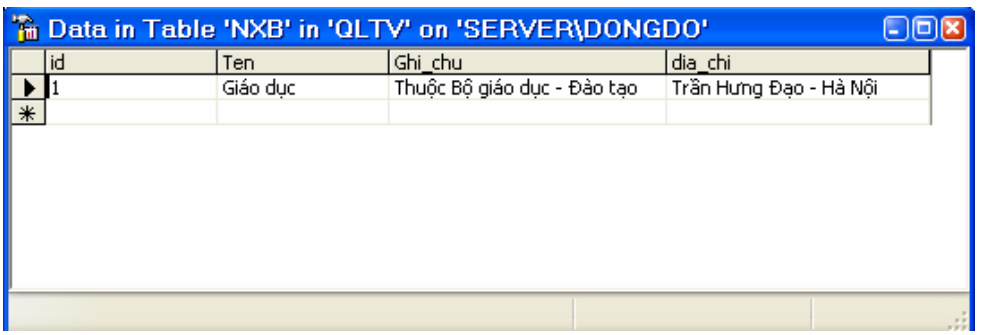
Sử dụng công cụ.

- Chọn bảng dữ liệu

- Nhất chuột phải -> Open Table -> Return all rows



- Nhập dữ liệu theo đúng quy cách kiểu dữ liệu, ràng buộc nếu có.



Việc sửa, xóa được thực hiện trực tiếp. Đối với các cột là dạng số, tăng tự động không cần nhập dữ liệu. Để lưu lại dữ liệu đã nhập bạn di chuyển con trỏ sang hàng khác.

Sử dụng câu lệnh.

Sử dụng lệnh Inert into.

Insert into NXB(Ten, Dia_chi) values(N'Kim Đồng', N'hà Nội')

Nếu cột dữ liệu hỗ trợ Unicode thì trước giá trị nhập vào bạn phải thêm kèm ký tự N (như ví dụ trên).

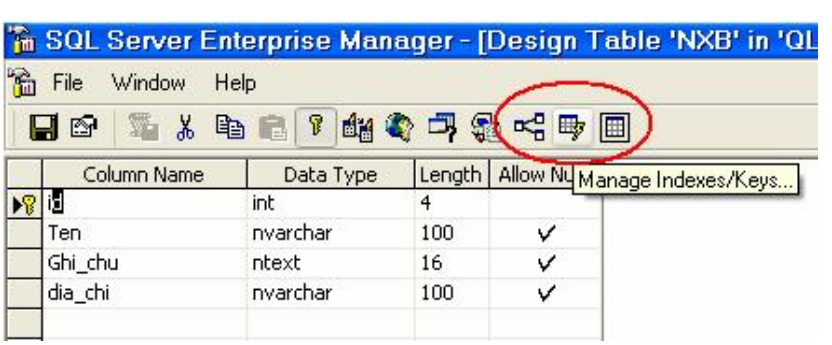
Tạo, sửa ràng buộc, khóa.

Phần này nhằm thực hiện thao tác với các rang buộc, khóa: ralationship, check, unique,...

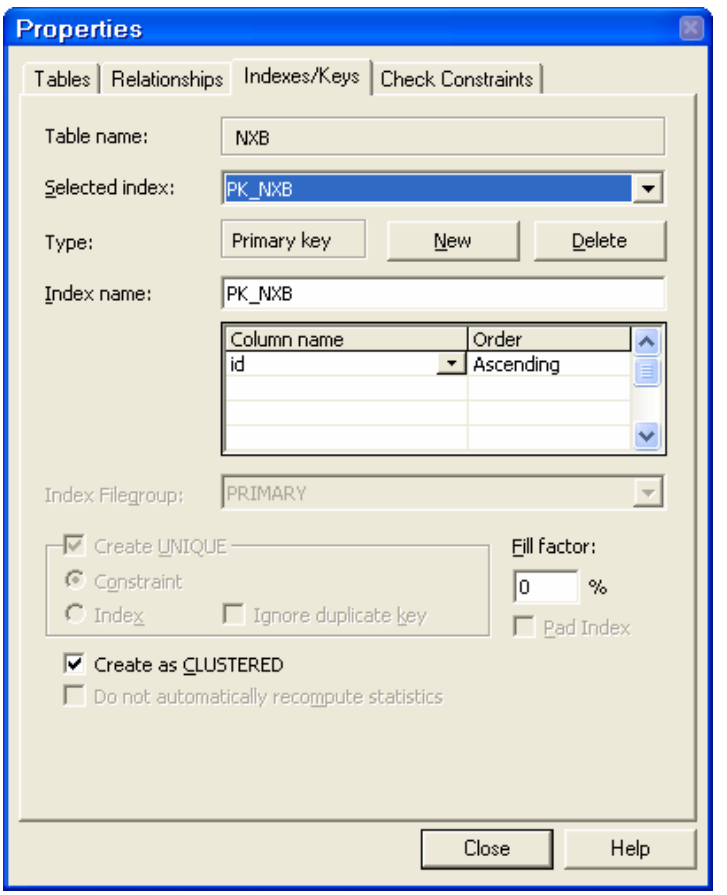
Sử dụng công cụ.

- Chọn chức năng Design table.

- Chọn biểu nút Manage Indexes/Keys...

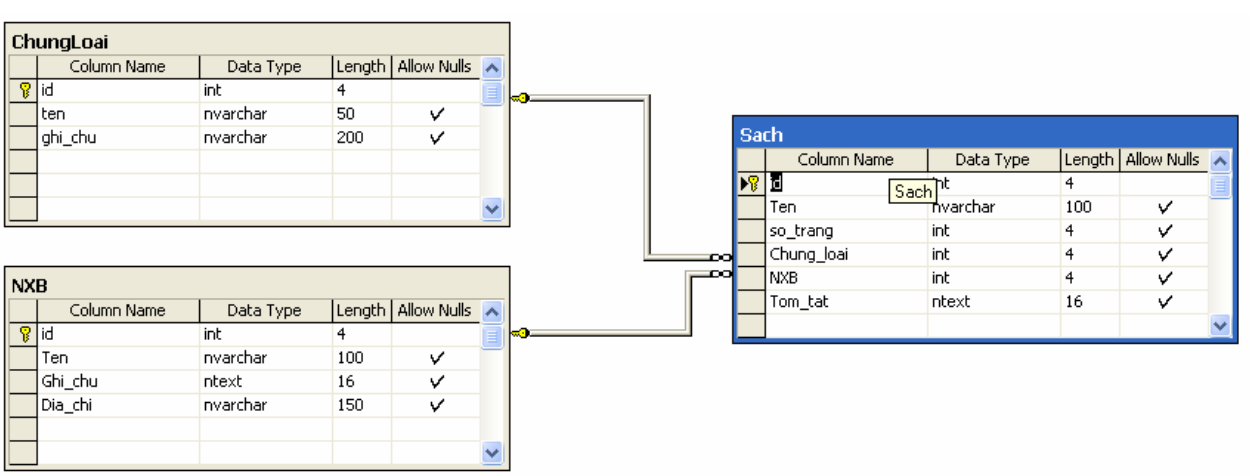


- Chọn bảng tương ứng.



Sử dụng câu lệnh.

Để cụ thể hơn ta thực hiện theo ví dụ có sơ đồ cấu trúc sau:

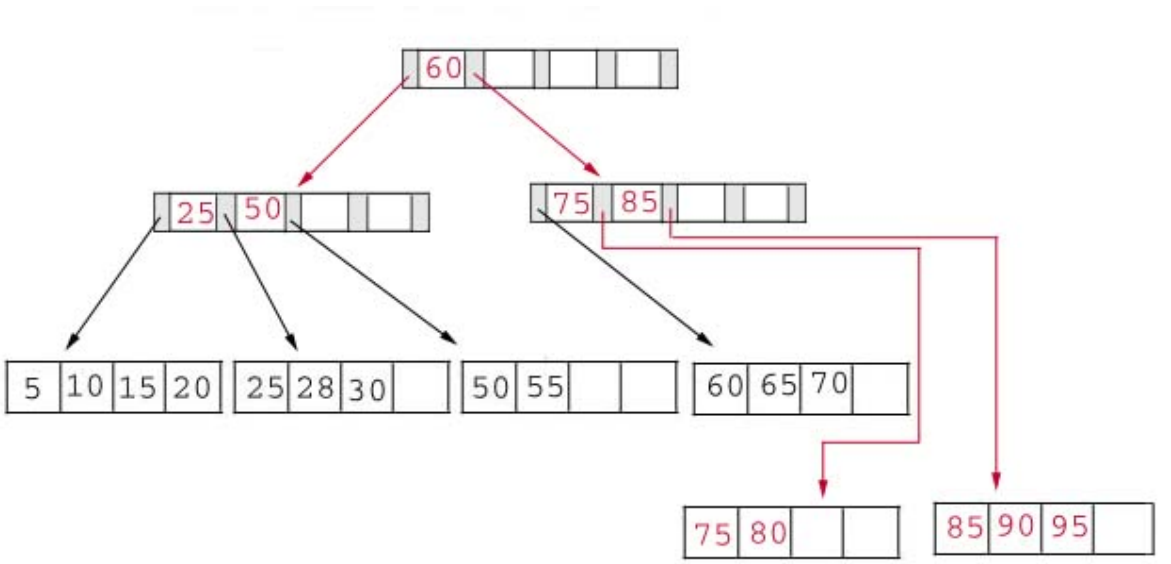


```
Create Table NXB(id int not null primary key identity(1,1), Ten Nvarchar(100), Ghi_chu Ntext, Dia_chi nvarchar(150))

Go

Create Table ChungLoai(id int not null primary key identity, ten nvarchar(50), ghi_chu nvarchar(200))

Go

Create Table Sach(id int not null primary key identity, Ten nvarchar(100), so_trang int default(0), Chung_loai int references Chungloai(id), NXB int references NXB(id), Tom_tat ntext)
```

KHÓA INDEX

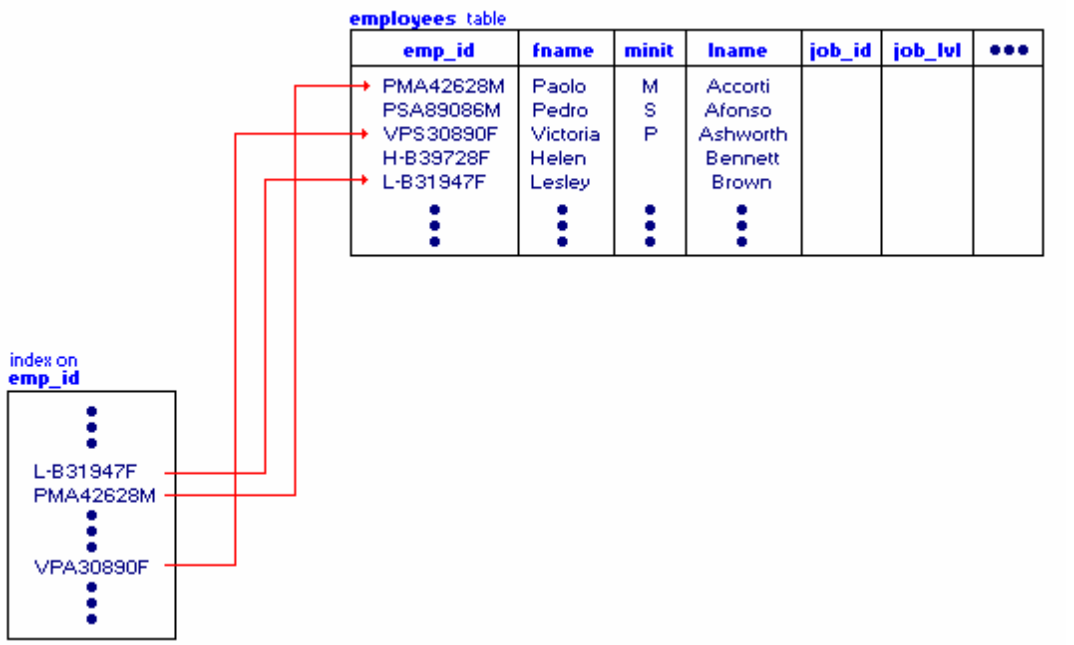
THIẾT KẾ KHÓA INDEX.

Index là một khóa quan trọng đối với CSDL đặc biệt là cơ sở dữ liệu lớn. Index được thiết lập từ một hoặc nhiều cột dữ liệu của bảng dữ liệu. các giá trị của khóa Index sẽ được sắp xếp và lưu trữ theo một danh sách (bảng khác). Mỗi giá trị trong khóa Index là duy nhất trong danh sách, mỗi giá trị khóa Index sẽ liên kết đến giá trị trong bảng dữ liệu (liên kết dạng con trỏ). Việc lưu trữ dữ liệu của bảng có khóa Index được thực hiện theo cấu trúc cấy B-Tree nhằm tăng tốc độ truy xuất dữ liệu đối với ổ đĩa (thiết bị thứ cấp).



Khi tìm kiếm một giá trị trong cột dữ liệu, mà cột này tham gia tạo khóa Index, đầu tiên câu lệnh xác định vị trí của giá trị nằm trong khóa Index bằng phép duyệt cây, sau đó thực hiện tìm theo liên kết đến bản ghi chứa giá trị tương ứng với khóa trong bảng.

Sơ đồ ví dụ dưới đây gồm khóa Index được tạo từ cột emp_id của bảng employees.



Việc thiết kế khóa Index dựa trên nhu cầu truy vấn, chèn dữ liệu trên một bảng, xác định dựa vào một số tham số sau:

+ Cột thường được sử dụng làm khóa truy vấn dữ liệu (xác định cột tham gia khóa Index).

+ Tập lệnh thường sử dụng try vấn cần tốc độ cao (xác định tập cột tham gia truy vấn).

+ Dữ liệu nhập vào bảng có khóa Index cần nhanh hơn hay truy vấn cần nhanh hơn (xác định đặt clustered hoặc nonclustered).

+ Lượng dữ liệu nhập đồng loạt nhiều hay ít (xác định tham số fillfactor).

Clustered Index.

Khi khóa đặt thuộc tính Clustered, dữ liệu của bảng sẽ được sắp xếp vật lý trên đĩa, như vậy khi thiết kế khóa dạng này dữ liệu được chèn và sẽ tìm đúng vị trí trên địa để lưu trữ (vùng đĩa dành cho bảng dữ liệu), chính vì vậy mà có thể xảy ra trường hợp phải dịch chuyển danh sách các giá trị đã có ở đĩa. Những việc tạo khóa Index dạng này sẽ không cần sắp xếp giá trị ở dạng logic mà khi truy nhập đĩa đã bảo đảm dữ liệu được sảo xếp.

Bảng dữ liệu chỉ có thể tạo tối đa một khóa Lustered Index.

Nonclustered Index.

Dữ liệu Index không sắp xếp ở dạng vật lý mà chỉ sắp xếp logic, dữ liệu của bảng lưu trữ giá trị khóa Index được sắp xếp, nhanh trong nhập dữ liệu.

Unique Index.

Xác định dữ liệu của cột tham gia khóa Index không lặp lại.

Fill Factor.

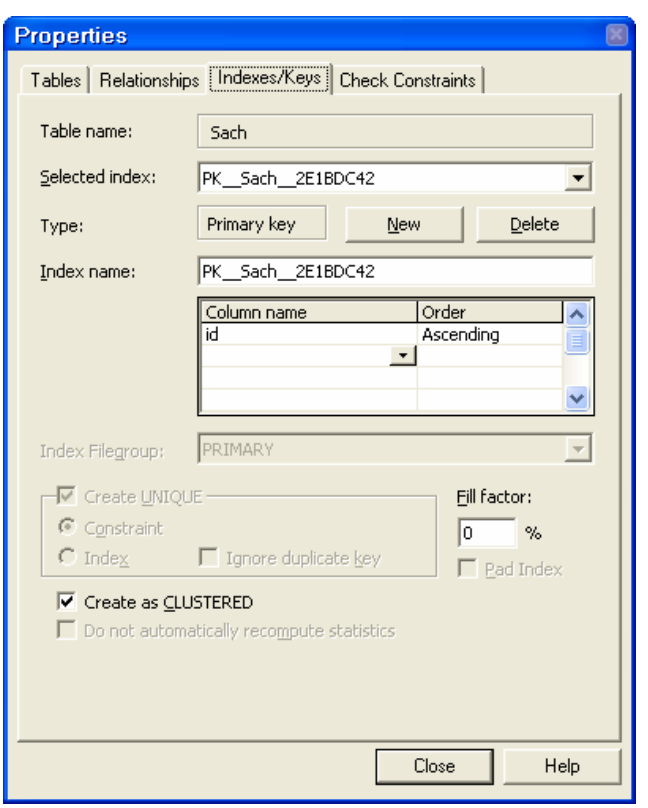
Khi tạo khóa Index, dữ liệu tham giá tạo khóa Index sẽ được phân theo mức của B-Tree, các mức được phần theo page dữ liệu, giá trị Fill factor xác định phần khoảng trống tối đa của page theo tỷ lệ phần trăm. Nhờ khoảng trống này mà tốc độ bố trí cấu trúc Index, tốc độ truy lục thông tin trong cây được cải thiện.

TẠO KHÓA INDEX.

Tạo theo công cụ.

- Chọn chức năng Design table

- Vào bảng Index manager.



- New

- Chọn các cột tham gia tạo khóa Index
- Đặt tham số.

Tạo theo câu lệnh.

- Sử dụng trong câu lệnh Create Table, Alter Table.
- Sử dụng lệnh Create Index.

CREATE [UNIQUE] [CLUSTERED | NONCLUSTERED] INDEX *index_name*
ON { *table* | *view* } **(** *column* [ASC | DESC] [**,**...*n*] **)**

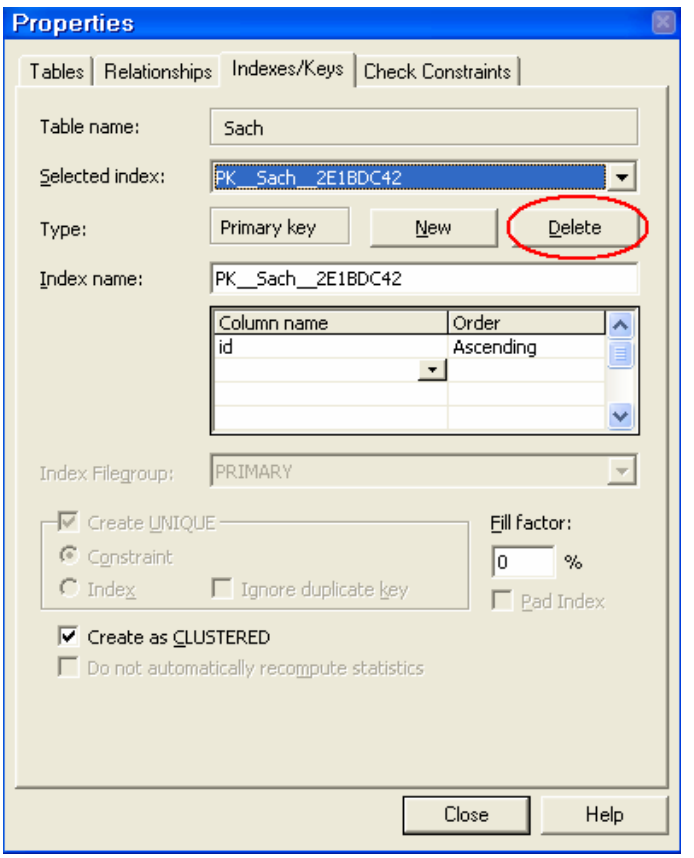
Ví dụ:

```
CREATE INDEX sach_idx  ON sach (id)
```

XÓA INDEX.

Sử dụng công cụ.

- Vào Index amnager
- Chọn khóa Index -> Delete



Sử dụng câu lệnh.

Sử dụng lệnh Drop Index.

Drop Index Sach(sach_idx)

KHUNG NHÌN – VIEW

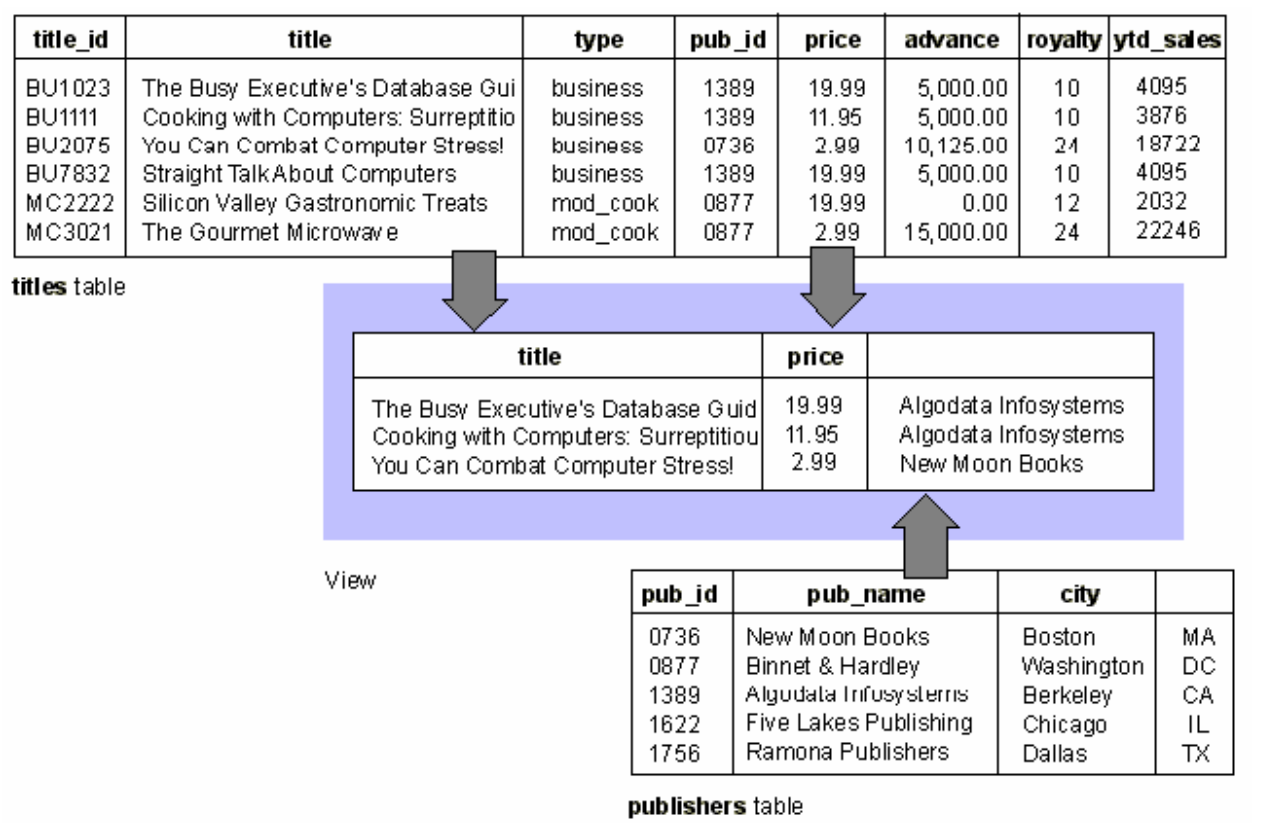
KHÁI NIỆM KHUNG NHÌN.

Khung nhìn (View) là một bảng tạm thời, có cấu trúc như một bảng, khung nhìn không lưu trữ dữ liệu mà nó được tạo ra khi sử dụng, khung nhìn là đối tượng thuộc CSDL.

Khung nhìn được tạo ra từ câu lệnh truy vấn dữ liệu (lệnh Select), truy vấn từ một hoặc nhiều bảng dữ liệu.

Khung nhìn được sử dụng khai thác dữ liệu như một bảng dữ liệu, chia sẻ nhiều người dùng, an toàn trong khai thác, không ảnh hưởng dữ liệu gốc.

Có thể thực hiện truy vấn dữ liệu trên cấu trúc của khung nhìn.

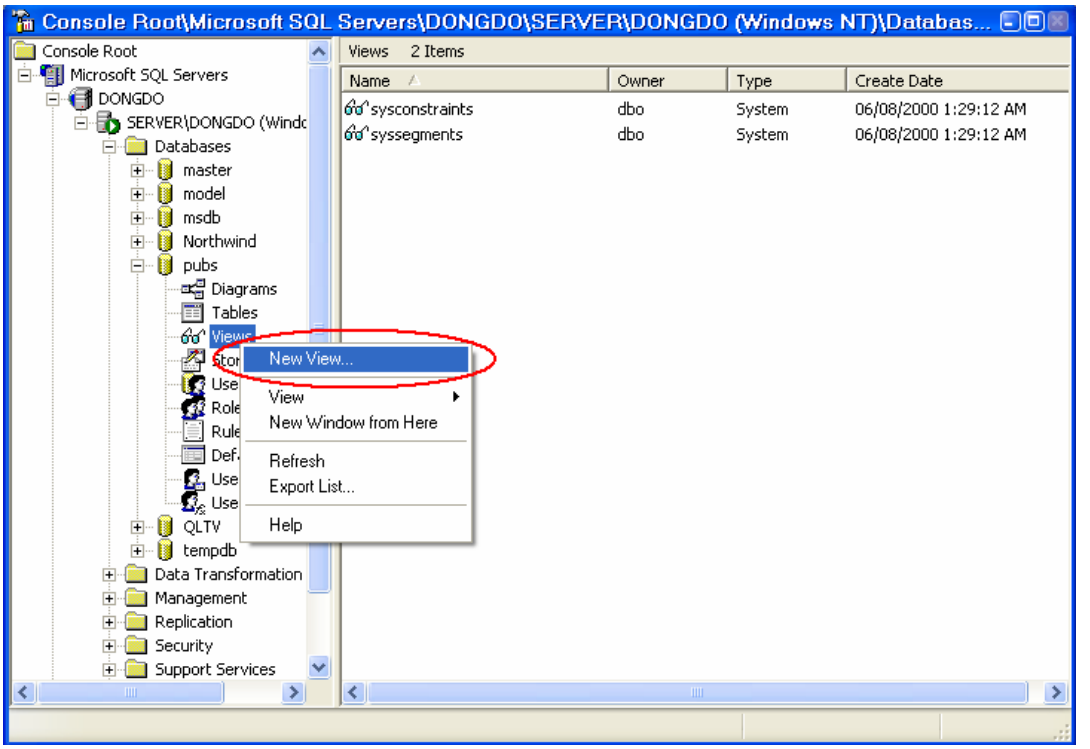


TẠO KHUNG NHÌN.

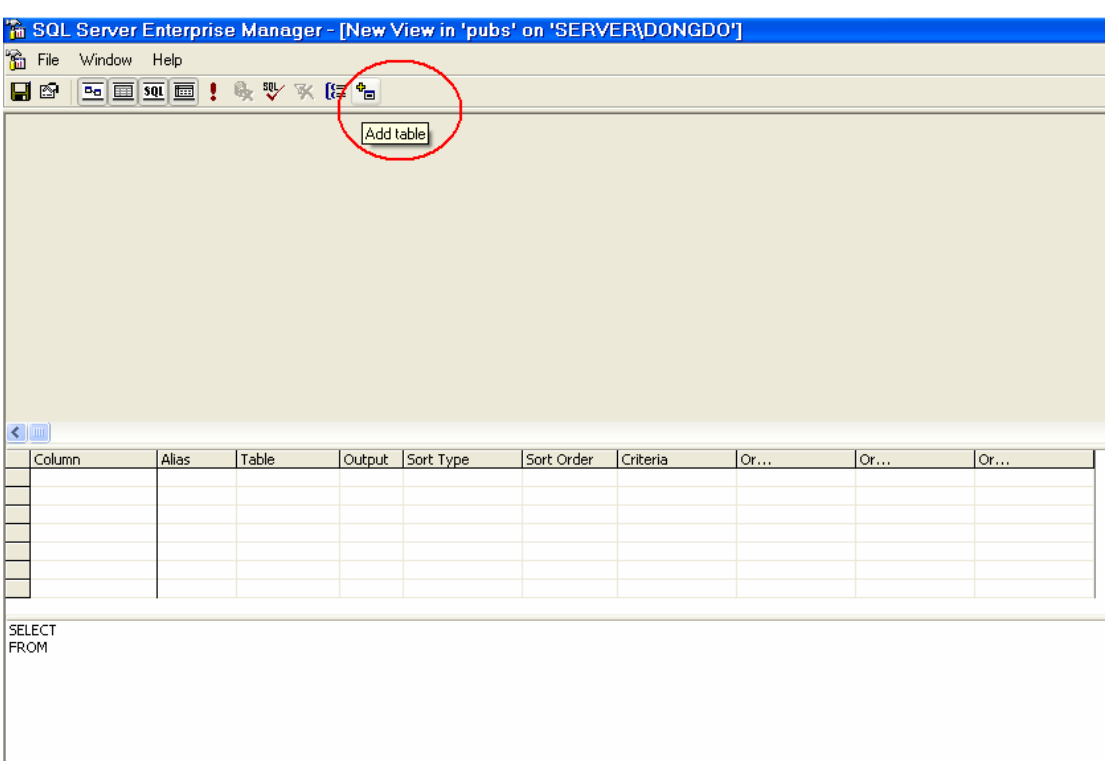
Sử dụng công cụ.

- Chọn chức năng Views của CSDL.

- Nhấn phải chuột.

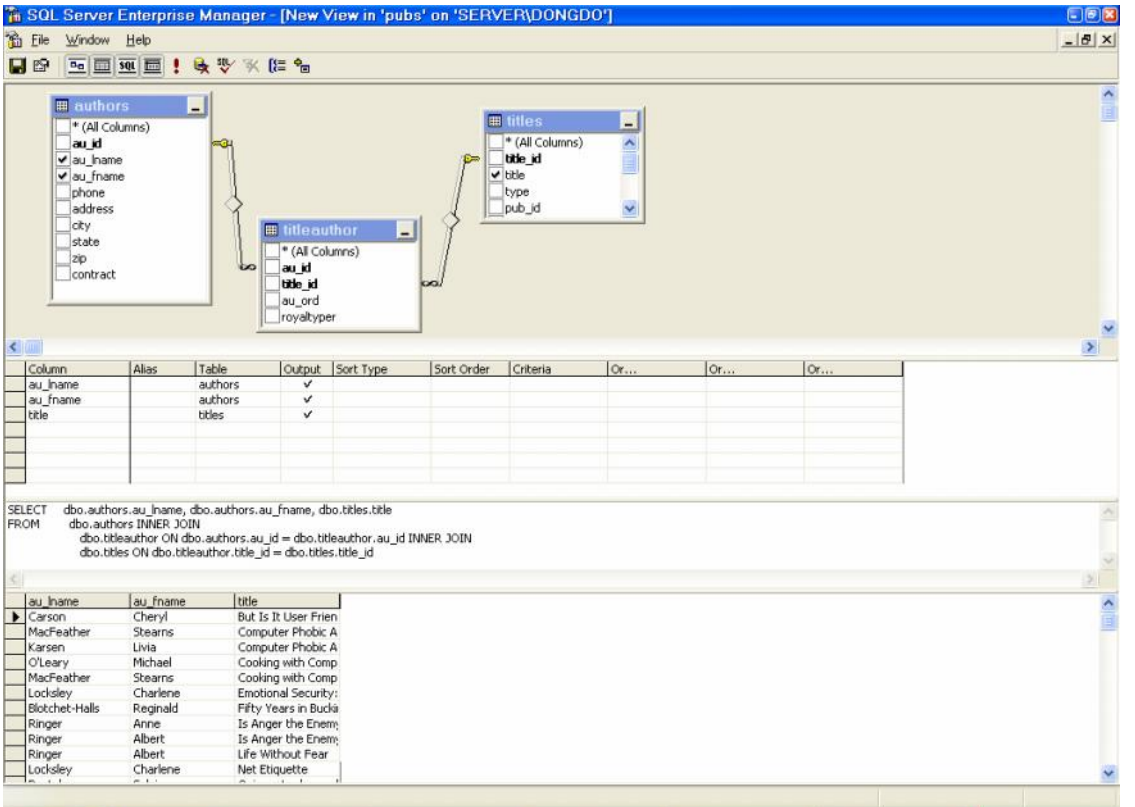


- Chọn New View.



- Thêm các bảng tham gia câu lệnh truy vấn dữ liệu cho View

- Soạn lệnh truy vấn hoặc đánh dấu các cột tham gia tạo View.



- Sửa đổi lệnh Select theo ý muốn.

- Ghi kịch bản -> đặt tên view.

Tạo theo câu lệnh.

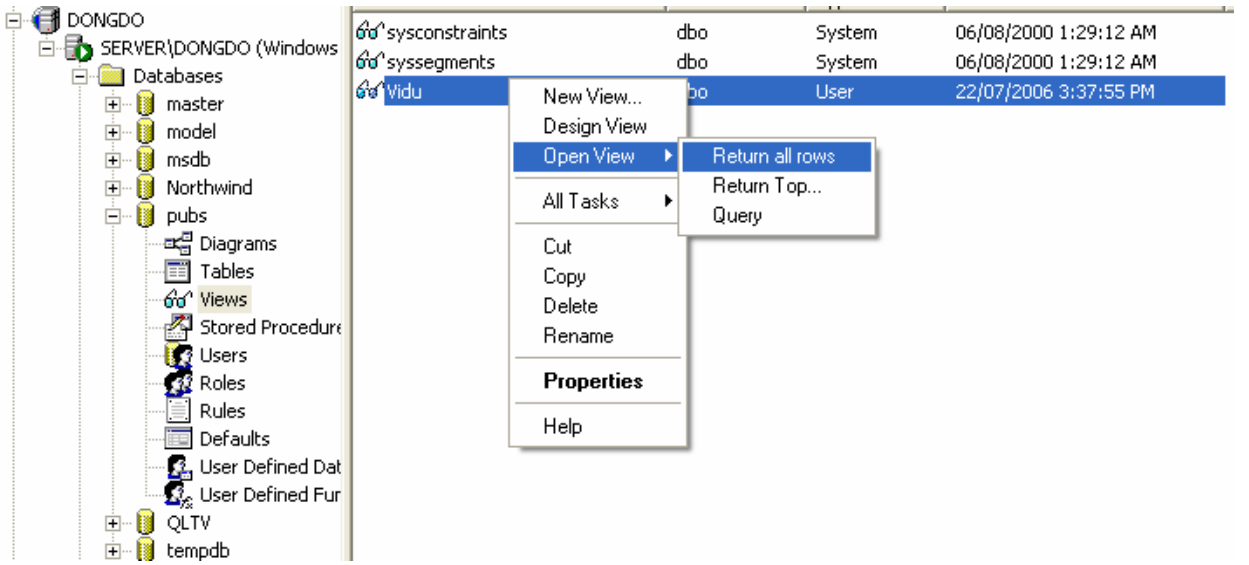
Sử dụng lệnh Create View:

```
CREATE VIEW VIDU as
SELECT     dbo.authors.au_lname, dbo.authors.au_fname, dbo.titles.title
FROM       dbo.authors INNER JOIN
                dbo.titleauthor ON dbo.authors.au_id = dbo.titleauthor.au_id INNER JOIN
                dbo.titles ON dbo.titleauthor.title_id = dbo.titles.title_id
```

SỬ DỤNG VIEW.

- Chọn View

- Nhấn nút phải chuột.



Thực hiện các chức năng tương tự table.

THỦ TỤC LƯU TRỮ

KHÁI NIỆM THỦ TỤC LƯU TRỮ VÀ HÀM.

Thủ tục lưu trữ có thuật ngữ Stored Procedure, là một đối tượng của CSDL tương tự như khung nhìn, thủ tục lưu trữ có thể tạo ra từ công cụ và câu lệnh. THủ tục được thực hiện như câu lệnh (có thể thực hiện từ SQL Query analyzer, các vị trí gọi câu lệnh T-SQL).

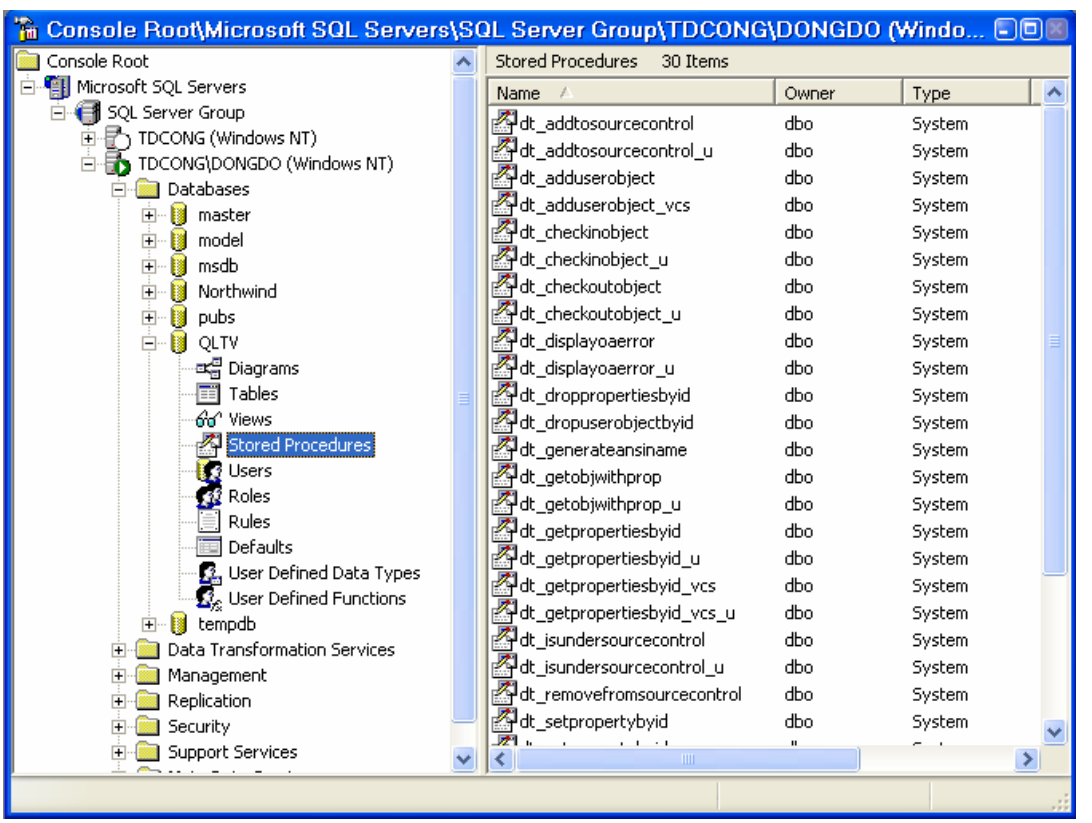
Thủ tục lưu trữ được kết cấu từ một kịch bản câu lệnh T-SQL, thủ tục có những đặc điểm cơ bản sau:

+ Truyền tham số.

+ Gọi thủ tục khác.

+ Trả về các giá trị tham số, chuyển giá trị tham số cho các thủ tục được gọi.

+ Trả về giá trị trạng thái thủ tục là thành công hay không thành công.



Thủ tục lưu trữ có nhiều ưu điểm so với thực hiện câu lệnh T-SQL từ các máy khách:

+ Lập trình theo module: Thủ tục được thiết lập trong từng CSDL một lần, có thể gọi thực hiện nhiều lần trong một ứng dụng, có thể gọi từ nhiều ứng dụng.

+ Thực hiện nhanh hơn: Khi cần thực hiện một lượng lớn câu lệnh T-SQL, thủ tục lưu trữ thực hiện nhanh hơn vì khi máy chủ nhận được nhiều cầu lệnh cùng một lúc đều phải kiểm tra tính hợp lệ quyền của tài khoản từ máy khách và các tham số khác. Khi thủ tục cần gọi nhiều lần trên các máy khách thì thủ tục thực hiện một lần đầu tiên, những lần sau máy khách sẽ chạy thủ tục đã được biên dịch.

+ Làm giảm lưu lượng trên mạng: Thay cho vì máy khách phải gửi nhiều dòng lệnh từ các ứng dụng đến máy chủ, khi sử dụng thủ tục thì nó chỉ cần gửi một lệnh, từ đó dẫn đến lưu lượng thông tin lệnh truyền qua mạng giảm.

+ An ninh bảo mật hơn: Khi không muốn cho một user trực tiếp khai thác một đối tượng hay bảng dữ liệu nào đó, mà cần cho user đó được khai thác thì thủ tục có thể giúp bạn gán quyền khai thác cho người đó. Việc gán quyền khai thác như nói trên sẽ giúp cho vấn đề an ninh bảo mật trong CSDL tốt hơn.

PHÂN LOẠI THỦ TỤC LƯU TRỮ.

Thủ tục lưu trữ được phân thành 5 loại như sau:

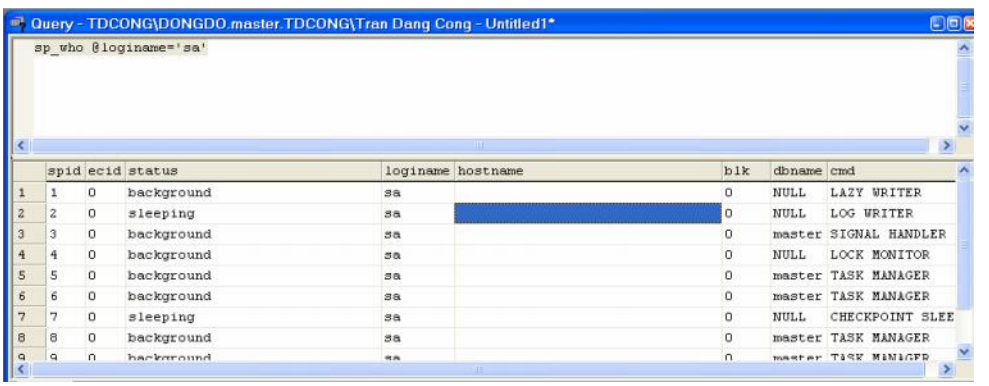
System Stored Procedure.

Là thủ tục được lưu trữ tỏng CSDL Master, thủ tục loại này được bắt đầu bằng chữ **sp_** thủ tục loại này thường được sử dụng trong quản trị CSDL và an ninh bảo mật.

Ví dụ: Muốn biết tất cả các tiến trình đang thực hiện bởi user nào:

```
sp_who @loginame='sa'
```

Kết quả:



	spid	ecid	status	loginame	hostname	blk	dbname	cmd
1	1	0	background	sa		0	NULL	LAZY WRITER
2	2	0	sleeping	sa		0	NULL	LOG WRITER
3	3	0	background	sa		0	master	SIGNAL HANDLER
4	4	0	background	sa		0	NULL	LOCK MONITOR
5	5	0	background	sa		0	master	TASK MANAGER
6	6	0	background	sa		0	master	TASK MANAGER
7	7	0	sleeping	sa		0	NULL	CHECKPOINT SLEE
8	8	0	background	sa		0	master	TASK MANAGER
9	9	0	background	sa		0	master	TASK MANAGER

Local Stored Procedure.

Đây là loại thủ tục thường dùng nhất, nằm trong CSDL do người dùng tạo ra, thực hiện một công việc nào đó. Thủ tục loại này thường được tạo bởi DBA (Database Administrator) hoặc người lập trình.

Temporary Stored Procedure.

Có chức năng tương tự như Local Stored Prcedure nhưng thủ tục loại này tự hủy khi kết nối tạo ra nó ngắt hoặc SQL Server ngưng hoạt động và nó được tạo ra trên CSDL TempDB.

Extended Stored Procedure.

Đây là loại thủ tục sử dụng chương trình ngoại vi đã được biên dịch thành DLL. Tên thủ tục được bắt đầu bằng **xp_**. Ví dụ thủ tục xp_sendmail dùng gửi mail, thủ tục xp_cmdshell dùng thực hiện lệnh của DOS (xp_cmdshell 'dir c:\').

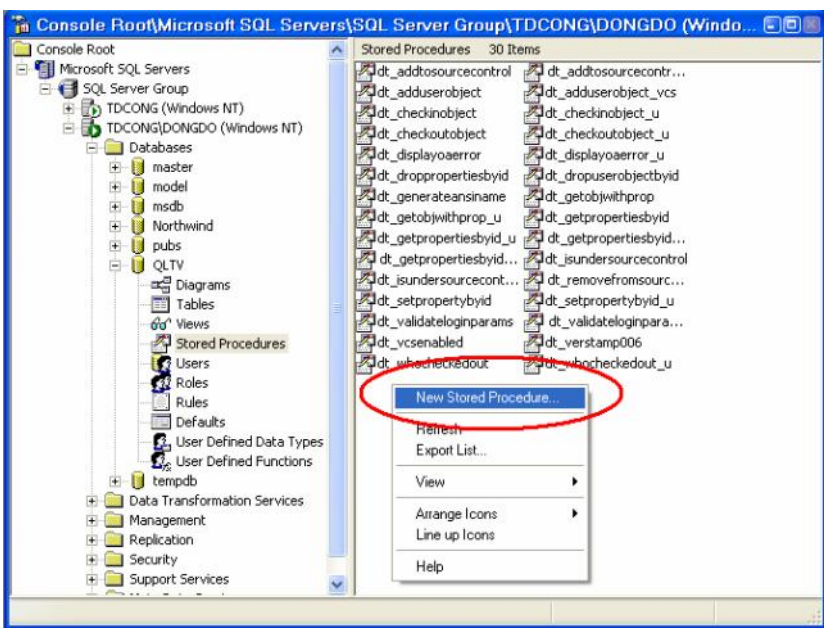
Remote Stored Procedure:

Là loại thủ tục sử dụng thủ tục của một server khác.

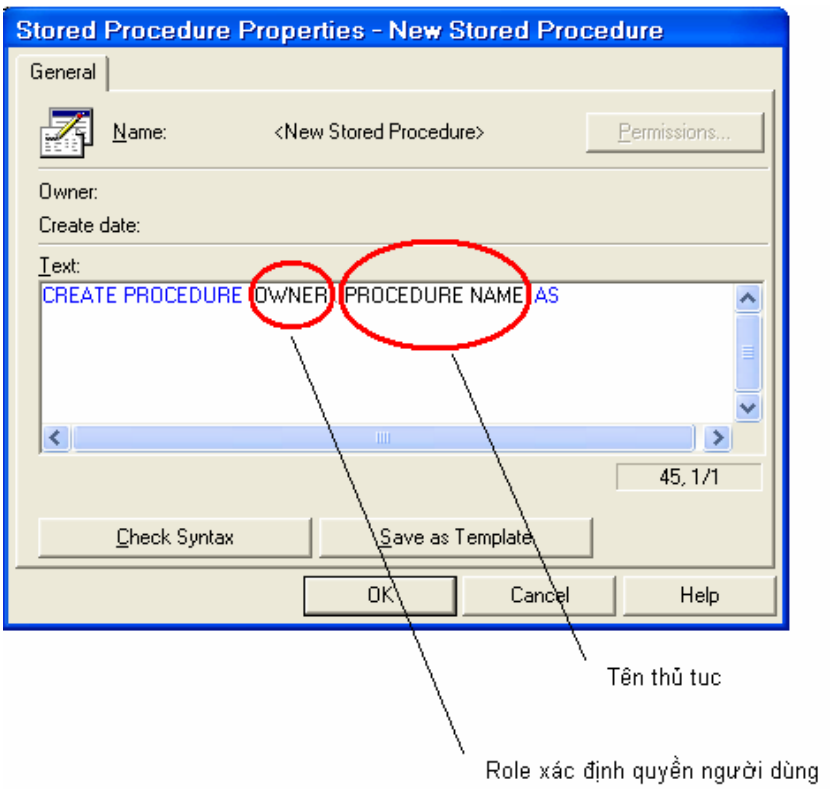
THIẾT LẬP THỦ TỤC LƯU TRỮ.

Sử dụng công cụ.

- Chọn CSDL cần tạo thủ tục trong Enterprise Manager –> Stored Procedures
- Nhấn nút phải chuột -> New Stored Procedure…



- Đặt tên thủ tục, xác định role người khai thác, soạn kịch bản câu lệnh.



Sử dụng câu lệnh.

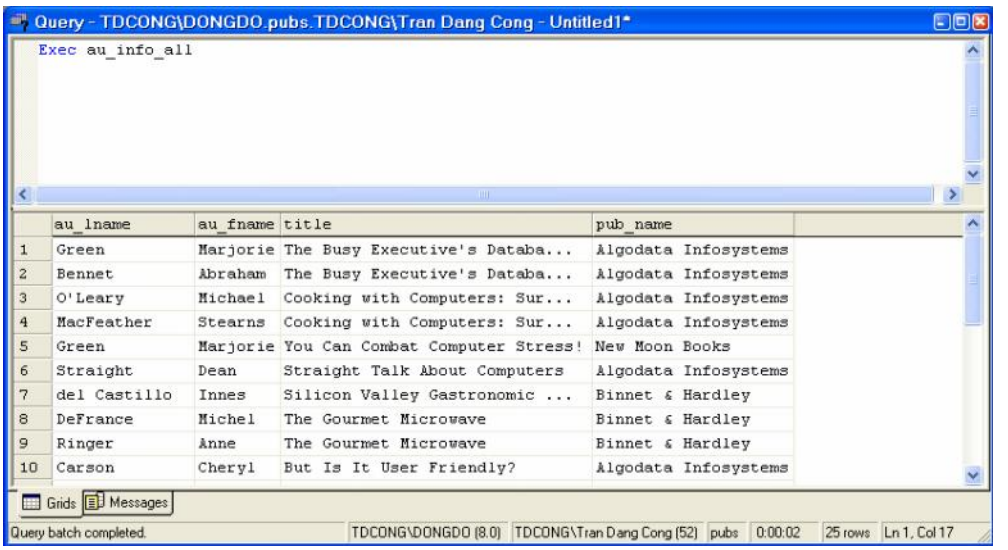
Sử dụng lệnh Create Procedure, để tiện xem xét ta xét theo các ví dụ , các ví dụ dưới đây thực hiện tạo thủ tục và thao tác với CSDL pubs để tiện trong dữ liệu mẫu, để tìm hiểu cú pháp câu lệnh T-SQL bạn xem phần câu lệnh T-SQL trong cùng tài liệu này.

Thủ tục không có tham số.

Thủ tục sau sẽ thực hiện liệt kê tất cả các tác giả, sách và nhà xuất bản mà tác giả viết sách.

```
Use Pubs
CREATE PROCEDURE au_info_all
AS
SELECT au_lname, au_fname, title, pub_name
  FROM authors a INNER JOIN titleauthor ta
    ON a.au_id = ta.au_id INNER JOIN titles t
    ON t.title_id = ta.title_id INNER JOIN publishers p
    ON t.pub_id = p.pub_id
GO
```

Kết quả thực hiện:



Thủ tục có tham số.

Thủ tục sau thực hiện lọc tìm tác giả có tên, họ truyền theo tham số.

```
USE pubs
GO
CREATE PROCEDURE au_info
   @lastname varchar(40),
   @firstname varchar(20)
AS
SELECT au_lname, au_fname, title, pub_name
   FROM authors a INNER JOIN titleauthor ta
      ON a.au_id = ta.au_id INNER JOIN titles t
      ON   t.title_id   =   ta.title_id   INNER   JOIN
publishers p
      ON t.pub_id = p.pub_id
   WHERE  au_fname = @firstname
      AND au_lname = @lastname
GO
```

Cách truyền tham số:

+ Gán giá trị theo thứ tự:

```
     EXECUTE au_info 'Dull', 'Ann'
+ Gán giá trị theo tên biến
    EXECUTE au_info @lastname = 'Dull', @firstname =
    'Ann'
+ Gán giá trị theo tên biến, không theo thứ tự
EXECUTE au_info @firstname = 'Ann', @lastname = 'Dull'
```

Thủ tục có tham số tùy lựa theo giá trị đưa vào.

Ví dụ này sẽ đề cập đến việc truyền tham số theo mẫu, giá trị tham số được ngầm định khi tạo thủ tục và thủ tục khi thực hiện sẽ kiểm tra giá trị tham số nhập vào.

```
USE pubs
GO
CREATE PROCEDURE au_info2
   @lastname varchar(30) = 'D%',
   @firstname varchar(18) = '%'
AS
SELECT au_lname, au_fname, title, pub_name
FROM authors a INNER JOIN titleauthor ta
   ON a.au_id = ta.au_id INNER JOIN titles t
   ON t.title_id = ta.title_id INNER JOIN publishers p
   ON t.pub_id = p.pub_id
WHERE au_fname LIKE @firstname
   AND au_lname LIKE @lastname
GO
```

Tham số % xác định giá trị tùy ý nhập vào tham số, tham số D% xác định giá trị đầu tiên của chuỗi phải bằng chữ D. Khi ngầm định các giá trị như trên tham có không được truyền giá trị sẽ tự nhận giá trị ngầm định.

Cách truyền tham số như sau:

```
+ Không truyền tham số:
     EXECUTE au_info2
```

```
+ Chỉ truyền tham số đầu, tham số sau sẽ nhận giá trị
ngầm định.
     EXECUTE au_info2 'Wh%'
+ Chỉ truyền một tham số, tham số xòn lại sẽ nhận giá
trị ngầm định.
    EXECUTE au_info2 @firstname = 'A%'
+ Tham số thứ nhất xác định giá trị một ký tự thuộc vị
trí có [CK] chỉ nhận ký tự ‘C’ hoặc ‘K’, [OE] chỉ nhận
giá trị ‘O’ hoặc ‘E’.
    EXECUTE au_info2 '[CK]ars[OE]n'
+ Xác định rõ giá trị tham số
    EXECUTE au_info2 'Hunter', 'Sheryl'
+ Xác định kiểu giá trị tham số.
    EXECUTE au_info2 'H%', 'S%'
```

Thủ tục sử dụng tham só láy giá trị ra (tham trị).

Ví dụ sau sẽ mô tả kỹ thuật sử dụng tham trị, như trong các ví dụ trước ta sử dụng tham số để truyền giá trị vào tên tham số bắt đầu bằng 1 chữ @, tham số được bắt đầu bằng 2 chữ @@. Sẽ được sử dụng trên nhiều dòng lệnh, sử dụng cùng từ khóa OUTPUT xác định là tham trị để lấy giá trị ra.

Ví dụ sau thực hiện truyền tham số vào và lấy giá trị ra:

```
USE pubs
GO
CREATE PROCEDURE titles_sum @@TITLE varchar(40) = '%',
@@SUM money OUTPUT
AS
SELECT 'Title Name' = title
FROM titles
WHERE title LIKE @@TITLE
SELECT @@SUM = SUM(price)
FROM titles
WHERE title LIKE @@TITLE
GO
```

Tham số bắt đầu bằng 2 ký tự @@ xác định được sử dụng cho nhiều câu lệnh, sử dụng cùng từ khóa OUTPUT xác định là biến tham trị.. Ví dụ trên sử dụng biến @@Title xác định điều kiện đưa ra tên sách đây là loại biến truyền vào sử

dụng cho hai câu lệnh Select, biến @@Sum xác định là biến tham trị dùng lấy giá trị ra.

Cách sử dụng tham số như sau:

```
DECLARE @@TOTALCOST money
EXECUTE titles_sum 'The%', @@TOTALCOST OUTPUT
IF @@TOTALCOST < 200
BEGIN
   PRINT ' '
   PRINT 'All of these titles can be purchased for less
than $200.'
END
ELSE
   SELECT 'The total cost of these titles is $'
         + RTRIM(CAST(@@TOTALCOST AS varchar(20)))
```

Ví dụ trên sử dụng biến @@TOTALCOST vào vị trí biến @@SUM trong thủ tục. Kết quả thực hiện như sau:

```
Title Name
--------------------------------------------------
The Busy Executive's Database Guide
The Gourmet Microwave
The Psychology of Computer Cooking

(3 row(s) affected)

Warning, null value eliminated from aggregate.

All of these titles can be purchased for less than
$200.
```

Thủ tục sử dụng biến OUTPUT kiểu con trỏ (Cursor).

Ví dụ sau tạo thủ tục có biến kiểu Cursor, biến này sẽ quản lý một bảng dữ liệu được truy vấn bằng câu lệnh Select.

```
CREATE PROCEDURE titles_cursor @titles_cursor CURSOR
VARYING OUTPUT
```

```
AS
SET @titles_cursor = CURSOR
FORWARD_ONLY STATIC FOR
SELECT *
FROM titles

OPEN @titles_cursor
GO
```

Con trỏ được đưa vào biến kiểu Cursor có tên @Titles_cursor, hướng dịch chuyển Forward (tiến) và Static. Sử dụng biến như ví dụ sau:

```
USE pubs
GO
DECLARE @MyCursor CURSOR
EXEC titles_cursor @titles_cursor = @MyCursor OUTPUT
WHILE (@@FETCH_STATUS = 0)
BEGIN
   FETCH NEXT FROM @MyCursor
END
CLOSE @MyCursor
DEALLOCATE @MyCursor
GO
```

Biến con trỏ được đưa vào viến @MyCursor, khi mở con trỏ vị trí bản ghi đầu tiên của bảng được xác định. Trong ví dụ trên sử dụng vòng lặp duyệt từng bản ghi, việc xử lý dữ liệu thực hiện trong vòng lặp.

Thủ tục đặt thuộc tính ẩn kịch bản câu lệnh.

Ví dụ sau sẽ đặt thuộc tính WITH ENCRYPTION ẩn văn bản trong thủ tục với người sử dụng.

```
CREATE PROCEDURE encrypt_this
WITH ENCRYPTION
AS
SELECT *
FROM authors
```

```
GO
```

Khi sử dụng thủ tục hệ thống sp_helptext để xem nội dung thủ tục:

```
EXEC sp_helptext encrypt_this
```

Kết quả như sau:

```
The object's comments have been encrypted.
```

SỬA, XÓA THỦ TỤC

Sử dụng công cụ.

- Chọn thủ tục cần sửa, xóa -> thực hiện sửa nội dung hoặc chức năng xóa.

Sử dụng câu lệnh.

- Sửa sử dụng lệnh Alter Procedure
- Xóa sử dụng lệnh Drop Procedure

Bạn đọc có thể tự tìm hiểu về User Defined Function tong Book Online, là đối tượng gọi là hàm thuộc CSDL, có chức năng và cách thức hoạt động gần giống thủ tục.

TRIGGER

KHÁI NIỆM TRIGGER.

Trigger là một thủ tục đặc biệt mà việc thực thi của nó tự động khi có sự kiện xảy ra, các sự kiện gọi thủ tục đặc biệt này được định nghĩa trong câu lệnh, thông thương được thực hiện với các sự kiện liên quan đến Insert, Update, Delete dữ liệu.

Trigger được sử dụng trong việc bảo đảm toàn vẹn dữ liệu theo quy tắc xác định, được quản lý theo bảng dữ liệu hoặc khung nhìn.

NHỮNG TRƯỜNG HỢP SỬ DỤNG TRIGGER.

- Sử dụng Trigger khi các biện pháp toàn vẹn dữ liệu như Constraint, rule,… không bảo đảm. Khác với các công cụ bao đảm toàn vẹn dữ liệu đã nêu, các công cụ này sẽ thực hiện kiểm tra tính toán vẹn trước khi đưa dữ liệu vào CSDL (còn gọi là Declarative Data Integrity), còn Trigger thực hiện kiểm tra tính toàn vẹn khi công việc đã thực hiện rồi (còn gọi là Procedural Data Integrity).

- Khi CSDL chưa được chuẩn hóa (Normalization) thì có thể xảy ra dữ liệu thừa, chứa ở nhiều vị trí trong CSDL thì yêu cầu đặt ra là dữ liệu cần cập nhật thống nhất trong mọi nơi. Trong trường hợp này ta phải sử dụng Trigger.

- Khi thay đổi day chuyền dữ liệu giữa các bảng với nhau (khi dữ liệu bảng này thay đổi thì dữ liệu trong bảng khác cũng được thay đổi theo).

ĐẶC ĐIỂM CỦA TRIGGER.

- Một trigger có thể thực hiện nhiều công việc (theo kịch bản), có thể nhiều sự kiện kích hoạt thực thi trigger, có thể tách rời các sự kiện trong một trigger.

- Trigger không được tạo trên bảng temprate hay system.

- Trigger chỉ thực thi tự động thông qua các sự kiện mà không thực hiện bằng tay.

- Trigger sử dụng được với khung nhìn.

- Khi trigger thực thi theo các sự kiện Insert hoặc Delete thì dữ liệu khi thay đổi sẽ được chuyển sang các bảng Inserted Table, Deleted Tabla, là 2 bảng tạm thời chỉ chứa trong bộ nhớ, các bảng này chỉ được sử dụng với các lệnh trong trigger.

Các bảng này thường được sử dụng để khôi phục lại phần dữ liệu đã thay đổi (roll back).

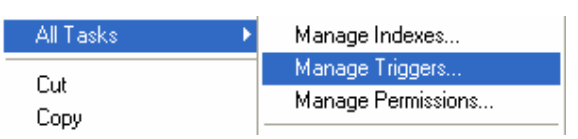
- Trigger chia thành 2 loại Instead of và After: Instead of là loại trigger mà hoạt động của sự kiện gọi nó sẽ bỏ qua và thay vào nó là các lệnh thực hiện trong trigger. After (tương đương với từ khóa For) đây là loại ngầm định, khác với loại Instead of thì loại trigger này sẽ thực hiện các lệnh trong nó sau khi đã thực hiện xong sự kiện gọi nó.

TẠO TRIGGER.

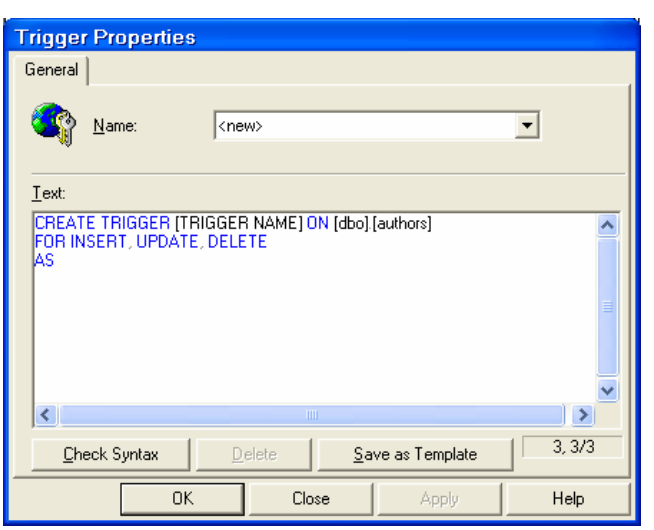
Tạo trigger được thực hiện thông công cụ và câu lệnh:

Tạo trigger bằng công cụ.

- Chọn bảng dữ liệu hoặc khung nhìn.
- Nhấn nút phải chuột.
- Chọn All tasks -> Manage Triggers...



- Soạn kịch bản tạo trigger.



(Cú pháp cụ thể hơn bạn xem trong phần tiếp theo)

Tạo trigger bằng câu lệnh.

Sử dụng lệnh Create Trigger, cú pháp chung như sau:

```
CREATE TRIGGER trigger_name
ON { table | view }
[ WITH ENCRYPTION ]
{
   { { FOR | AFTER | INSTEAD OF } { [ INSERT ] [ , ] [ UPDATE ] }
      [ WITH APPEND ]
      [ NOT FOR REPLICATION ]
      AS
      [ { IF UPDATE ( column )
         [ { AND | OR } UPDATE ( column ) ]
            [ ...n ]
      | IF ( COLUMNS_UPDATED ( ) { bitwise_operator } updated_bitmask )
            { comparison_operator } column_bitmask [ ...n ]
      } ]
      sql_statement [ ...n ]
   }
}
```

Các tham số cơ bản:

+ trigger_name: Tên trigger.

+ table/view: Tên bảng hoặc khung nhìn.

+ For/After/Instead Of: Loại trigger.

+ { [DELETE] [,] [INSERT] [,] [UPDATE] }: Sự kiện khi tự động gọi thực thi trigger.

+ *sql_statement* [...*n*]: Kịch bản các câu lệnh xử lý của trigger.

Các câu lệnh sau không được thực thi trong kịch bản các câu lệnh xử lý của trigger:

ALTER DATABASE	CREATE DATABASE	DISK INIT
DISK RESIZE	DROP DATABASE	LOAD DATABASE
LOAD LOG	RECONFIGURE	RESTORE DATABASE
RESTORE LOG		

Để cụ thể hơn ta xét một số ví dụ sau:

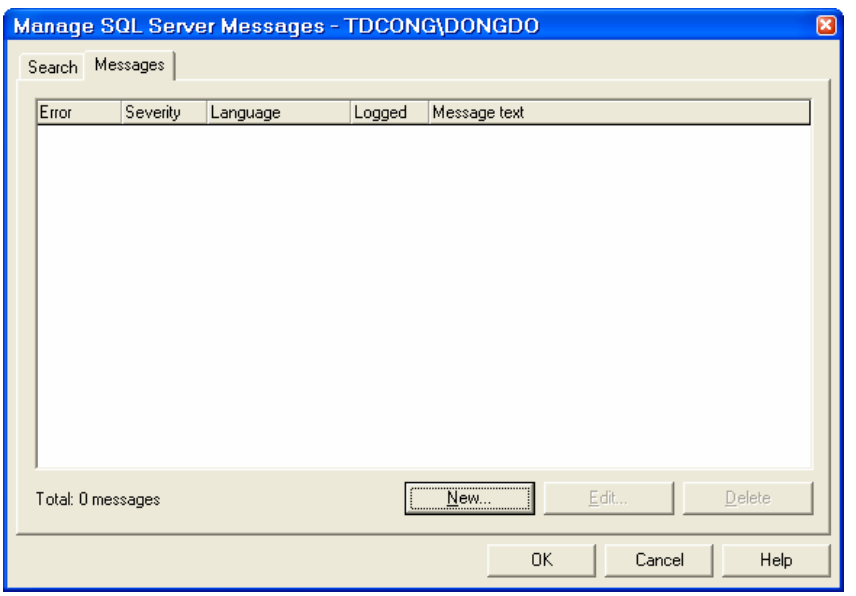
Ví dụ tạo một trigger thông báo.

```
CREATE TRIGGER reminder
ON titles
FOR INSERT, UPDATE
AS RAISERROR (50001, 16, 10)
GO
```

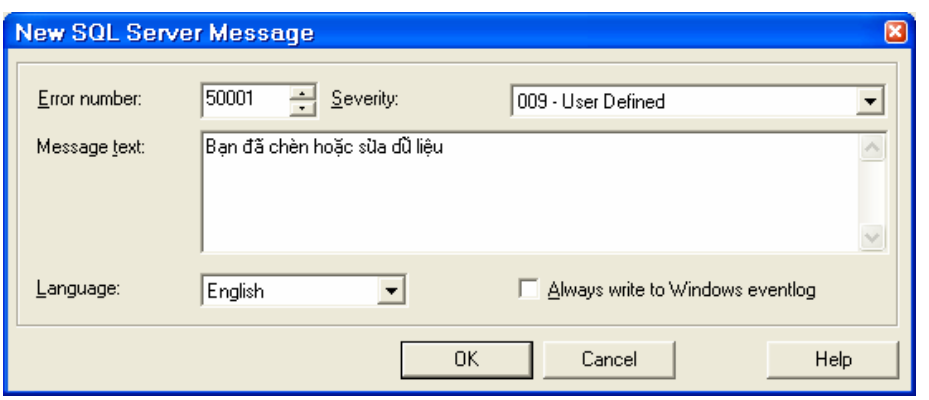
Ví dụ trên tạo một thông báo cho các client khi thực hiện thêm hoặc sửa dữ liệu trên bảng Titles, mã thông báo là 50001, là mã thông báo do người dùng định nghĩa.

Để tạo thông báo bạn thao tác như sau:

- Vào menu Tools -> Manage SQL Server Messages…



- Chọn bảng Messages -> New…



- Đặt mã, soạn nội dung, kiểu thông báo (Serverity), mã thông báo sẽ được sử dụng trong các ứng dụng hoặc câu lệnh yêu cầu.

Ví dụ tạo trigger tự động gửi Email khi được thực thi.

```
CREATE TRIGGER reminder
ON titles
FOR INSERT, UPDATE, DELETE
AS
   EXEC master..xp_sendmail 'MaryM',
      'Don''t forget to print a report for the
distributors.'
GO
```

Ví dụ tạo trigger kiểm soạn khoảng giá trị giữa 2 bảng.

Ví dụ sau sẽ tạo trigger thực hiện kiểm soát phạm vị mức lương của một nhân viên vừa chèn vào có thuộc giá trị định mức lương trong bảng mức lương hay không.

```
CREATE TRIGGER employee_insupd
ON employee
FOR INSERT, UPDATE
AS
DECLARE @min_lvl tinyint,
   @max_lvl tinyint,
   @emp_lvl tinyint,
   @job_id smallint
SELECT @min_lvl = min_lvl,
   @max_lvl = max_lvl,
   @emp_lvl = i.job_lvl,
   @job_id = i.job_id
FROM employee e INNER JOIN inserted i ON e.emp_id =
i.emp_id
   JOIN jobs j ON j.job_id = i.job_id
IF (@job_id = 1) and (@emp_lvl <> 10)
BEGIN
   RAISERROR ('Job id 1 expects the default level of
10.', 16, 1)
   ROLLBACK TRANSACTION
END
ELSE
IF NOT (@emp_lvl BETWEEN @min_lvl AND @max_lvl)
```

```
BEGIN
   RAISERROR ('The level for job_id:%d should be
between %d and %d.',
      16, 1, @job_id, @min_lvl, @max_lvl)
   ROLLBACK TRANSACTION
END
```

SỬA, XÓA TRIGGER.

Sử dụng công cụ.

- Chọn trigger trong mục Manage Triggers…
- Thực hiện sửa nội dung hoặc xóa.

Sửa, xóa theo câu lệnh.

- Sử dụng lệnh Alter trigger để sửa.
- Sử dụng lệnh Drop Trigger để xóa.

(Bạn có thể tìm hiểu chi tiết hơn trong Book Online)

XUẤT – NHẬP DỮ LIỆU

Trong chương này bạn đọc sẽ tìm hiểu kỹ thuật trao đổi dữ liệu với các mỗi trường ngoài Server của SQL, có thể với một CSDL khác, một Server SQL khác, một hệ quản trị CSDL khác hoặc nối ghép tập tin CSDL.

SERVER LIÊN KẾT – LINKED SERVER.

Tương tự như các hệ quản trị CSDL lớn khác (Access, Oracle), hệ thống cung cấp công cụ liên kết với hệ quản trị CSDL khác. Khi liên kết đã được thiết lập, với quyền hạn của user liên kết bạn có thể thực hiện khai thác dữ liệu liên kết trên SQL Server.

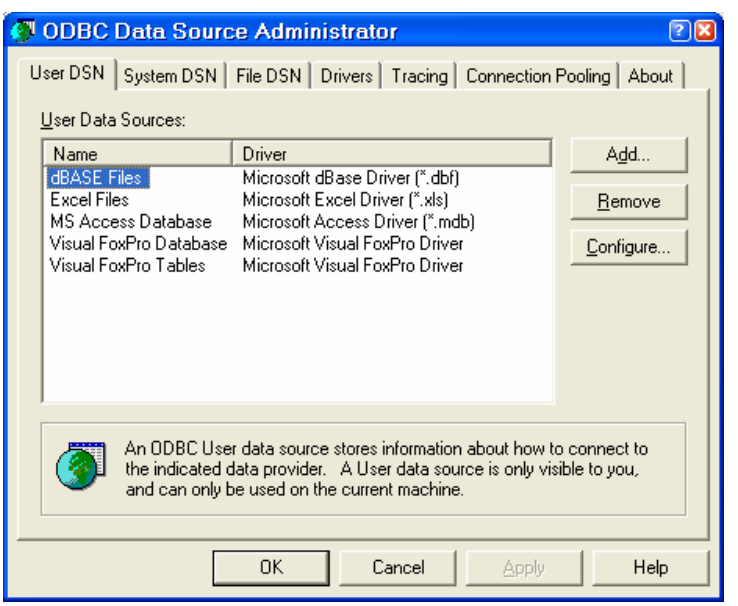
Tạo ODBC.

ODBC viết tắt của cụm từ Open DataBase Connectivity, là công cụ được Windows cung cấp với mục đích làm môi trường trao đổi dữ liệu giữa những CSDL, giữa nhiều hệ quản trị CSDL với ứng dụng.

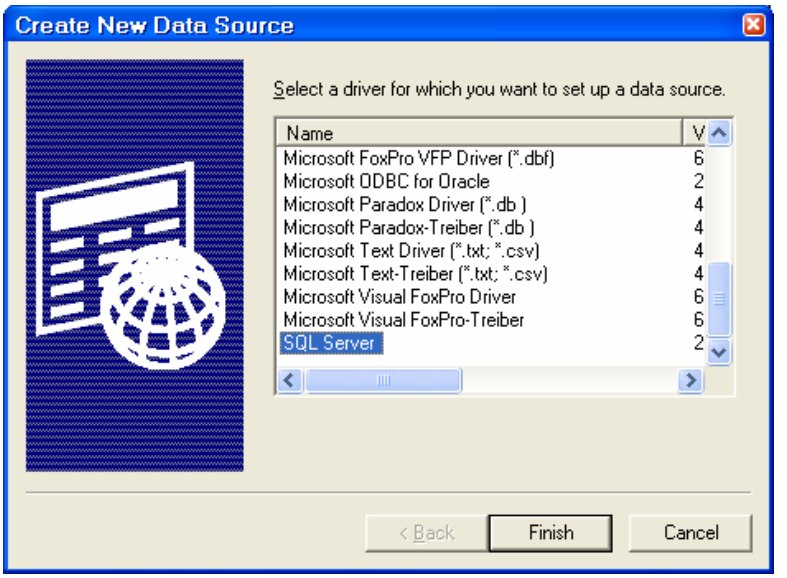
ODBC tạo những chuẩn chung nhất kết nối đến CSDL và ứng dụng. Khi thực hiện khai thác dữ liệu thông qua ODBC, ứng dụng liên kết theo tên ODBC, quyền hạn khai thác thực hiện khi tạo kết nối tự ODBC đến CSDL.

Cách tạo ODBC:

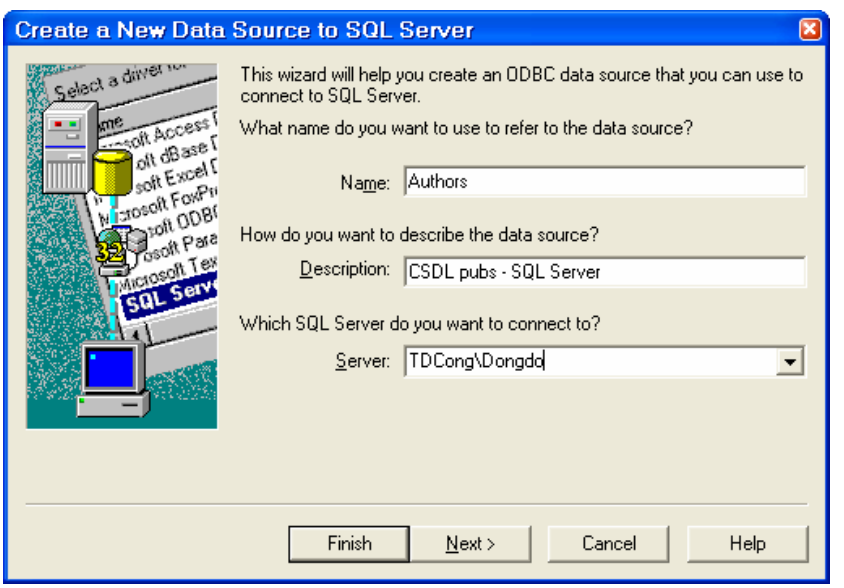
- Chọn ODBC trong Control panel.



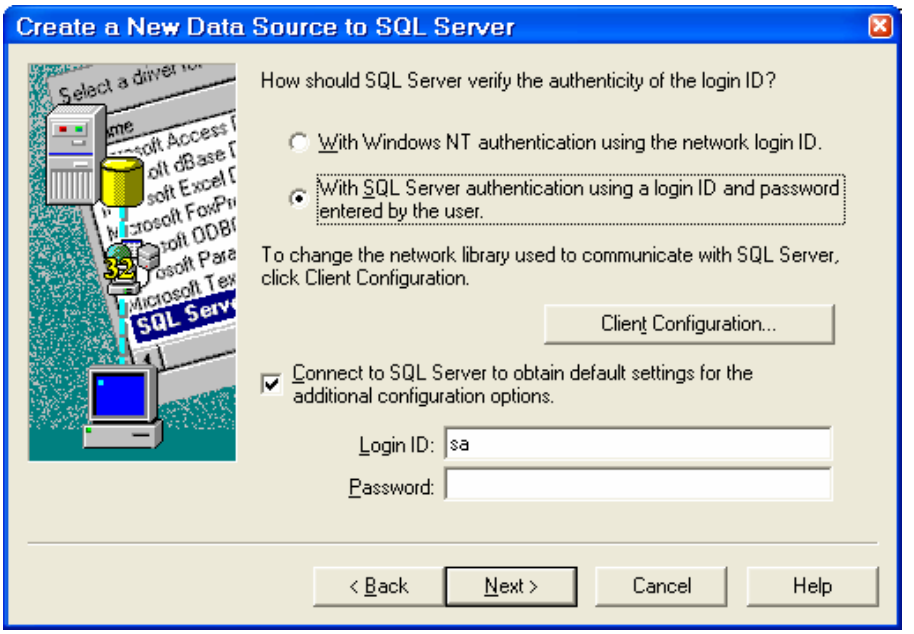
- Chọn bảng User DSN -> Add…



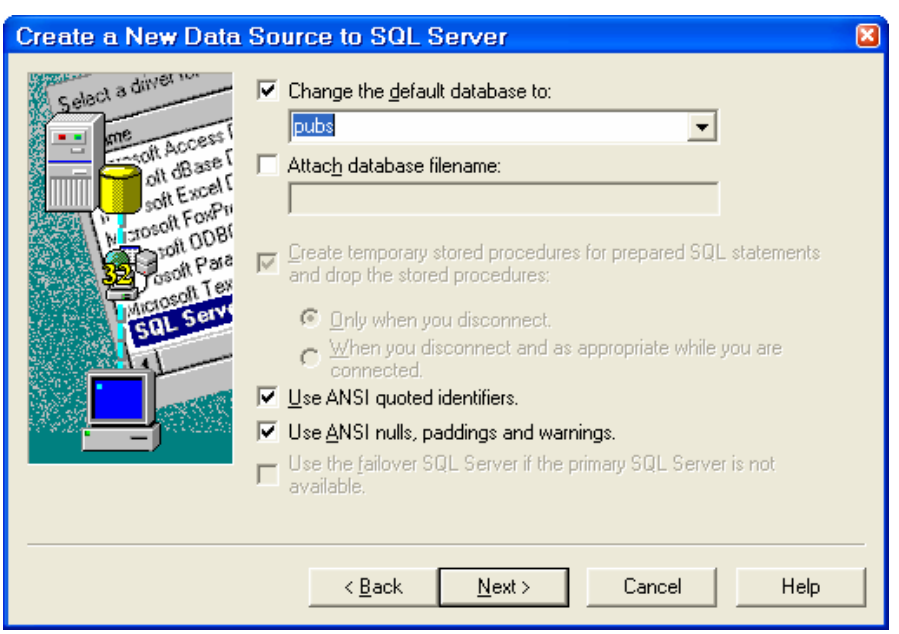
- Chọn Driver của hệ quản trị CSDL của CSDL cần liên tạo ODBC.
- Chọn Finish.



- Nhập tên ODBC (tên này sẽ sử dụng cho ứng dụng khác, nên nhập theo chuẩn chung để dễ sử dụng), các tham số khác (ở đây trên hình sử dụng Driver của SQL Server nên bạn phải chọn Server).
- Next.

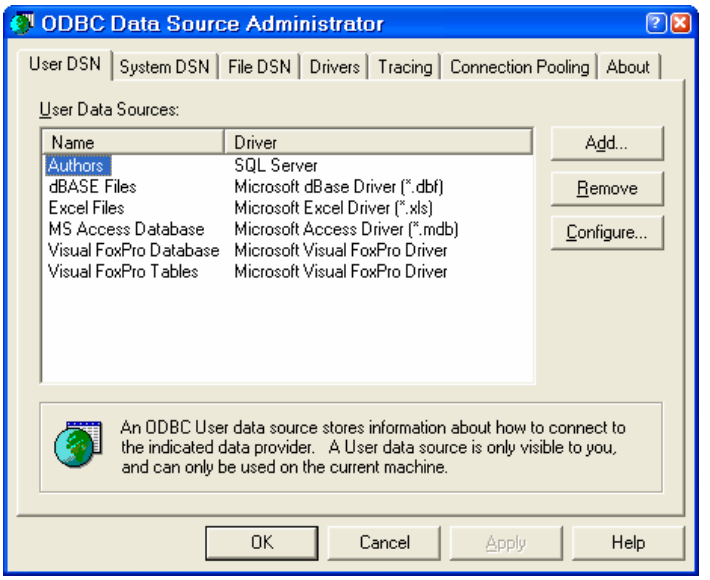


- Nhập Login ID, mật khẩu -> Next.



- Đánh dấu Change the default database to -> Chọn CSDL -> Next -> Finish

Sau khi tạo xong trong danh sách xuất hiện ODBC bạn vừa tạo, tư danh sách bạn có thể sửa đổi, xóa ODBC khi cần thiết.



Tạo liên kết từ Access.

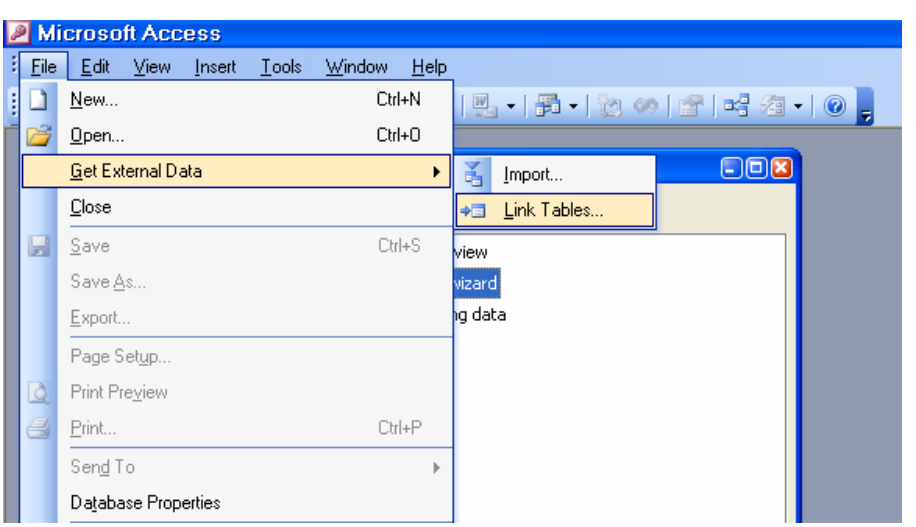
Từ hệ quản trị CSDL Access bạn có thể tạo liên kết đến các hệ quản trị CSDL khác (Acess, Dbase,…), hoặc thông qua ODBC. Trong ví dụ minh họa sử dụng liên kết từ Access với ODBC (đối với SQL Server hoặc Oracle, My SQL thì Access phải liên kết thông qua ODBC vì các hệ quản trị CSDL này không thực hiện khai thác dữ liệu qua tập tin chỉ khai thác thông qua tên CSDL, mà Access chỉ thực hiện theo phương thức mở tập tin).

Các bước thực hiện như sau:

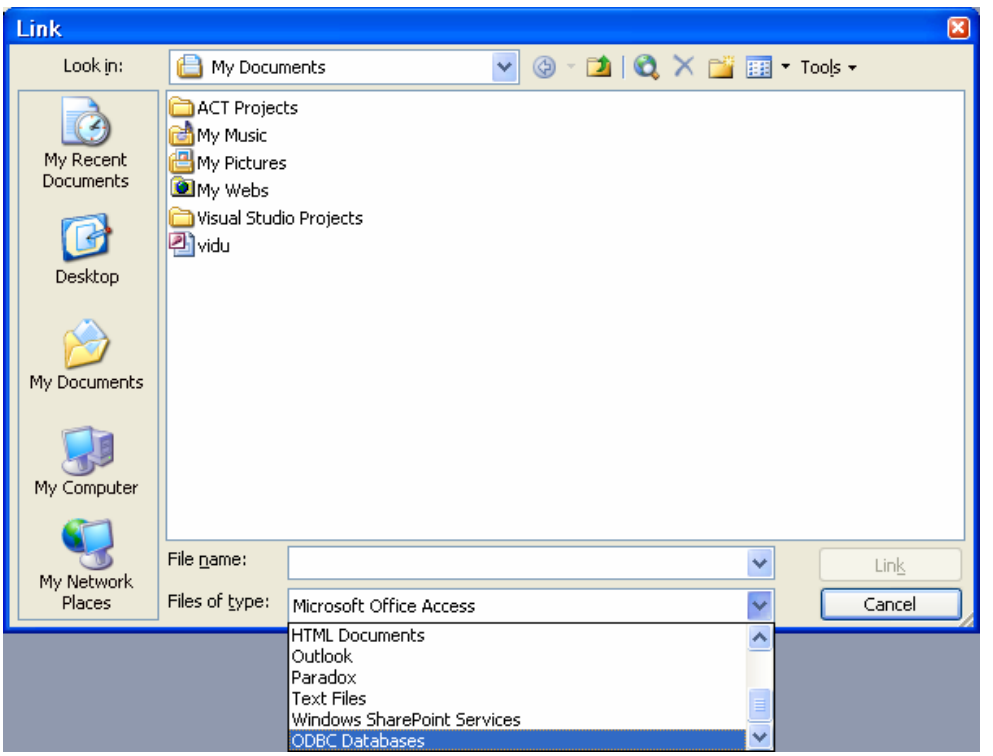
- Mở hệ quản trị CSDL Access.

- Mở hoặc tạo CSDL mới từ Access

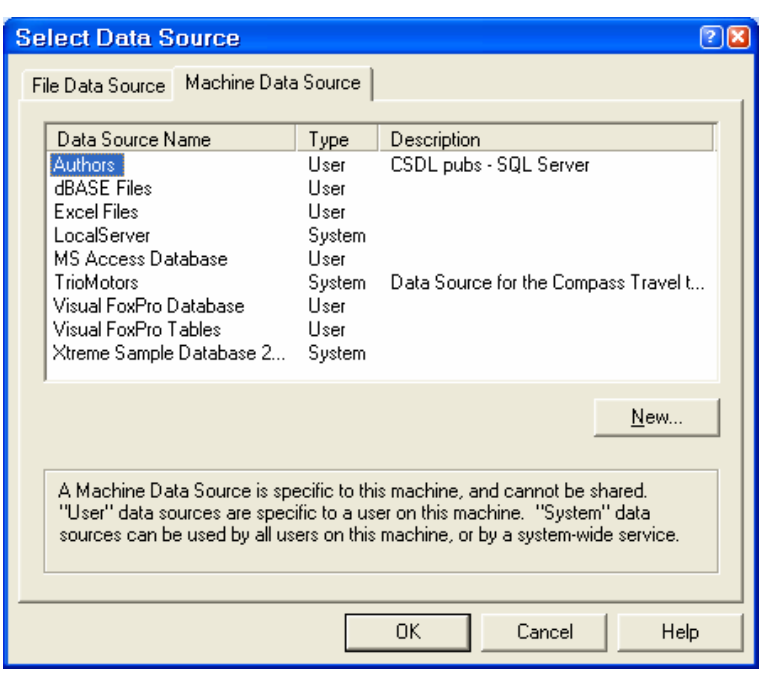
- Chọn File -> Get External Data -> Link Tables.



- Chọn ODBC Databases.



- Chọn ODBC cần liên kết (Authors).

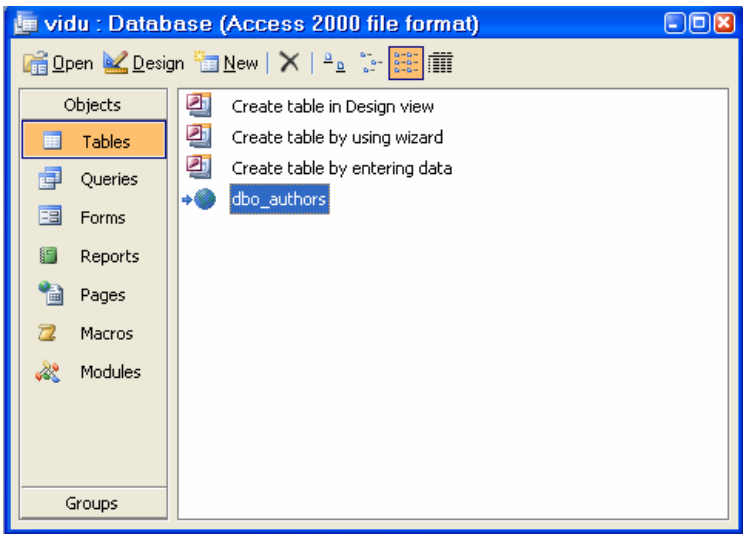


- Ok.

- Nhập Login ID và mật khẩu.

- Chọn bảng hoặc khung nhìn cần liên kết trong danh sách.

- Nhấn Ok, danh sách các bảng trong Access được khai thác tương tự như các bảng khác.

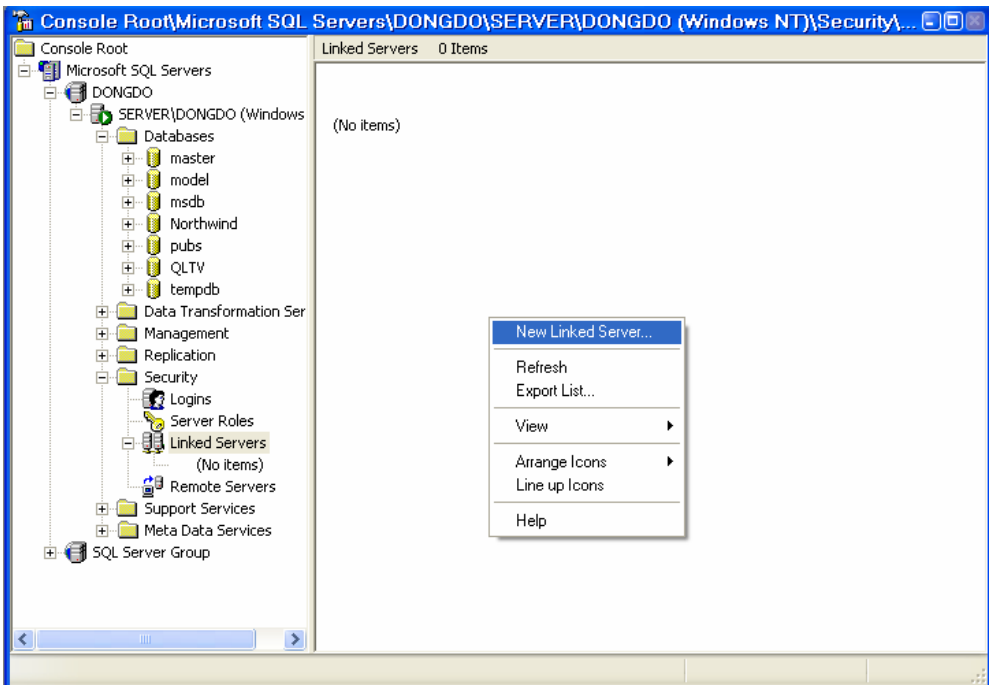


Tạo Server liên kết – Linked Server.

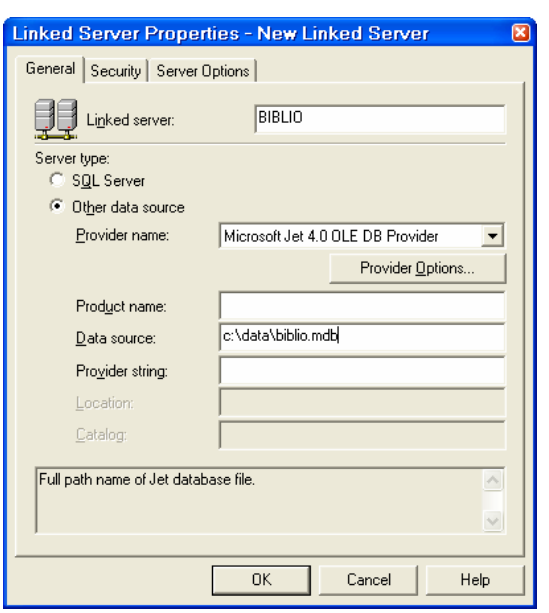
Từ SQL Server có thể tạo liên kết trực tiếp đến các hệ quản trị CSDL khác (Access, SQL Server, Oracle, My SQL,...) mà không cần thiết phải thông qua ODBC như Access đã xét trước.

Tạo bằng công cụ.

- Vào mục Security -> Linked Server.



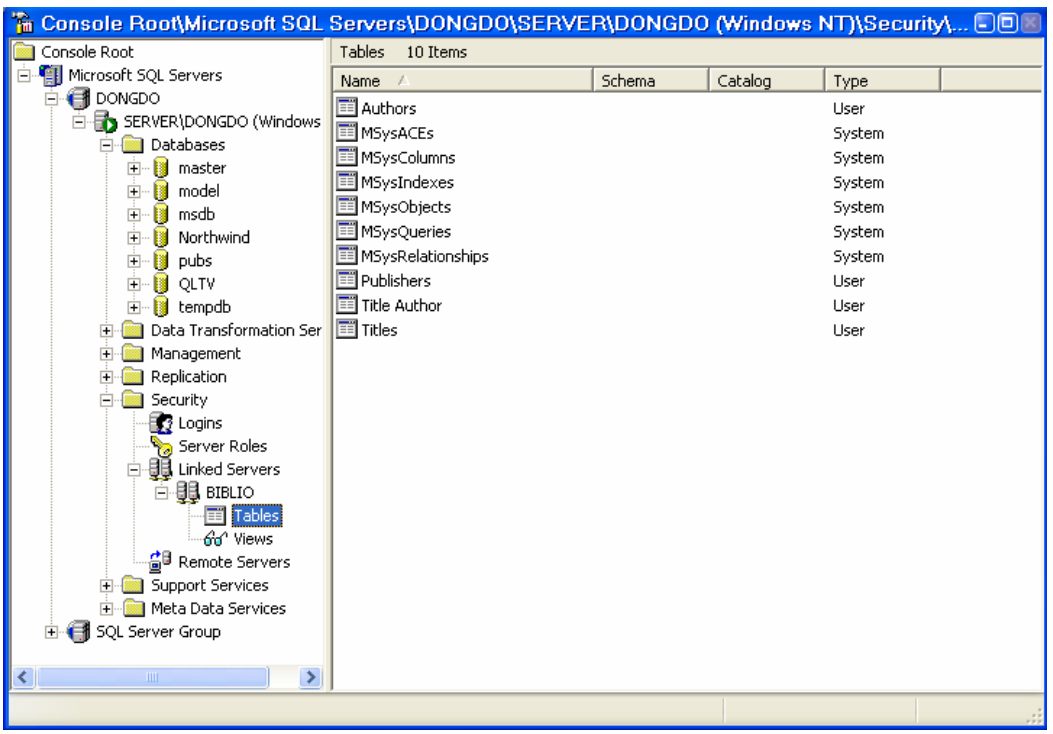
- Nhấn nút phải chuột -> New Linked Server.



- Nhập các tham số:

+ Tên Server.

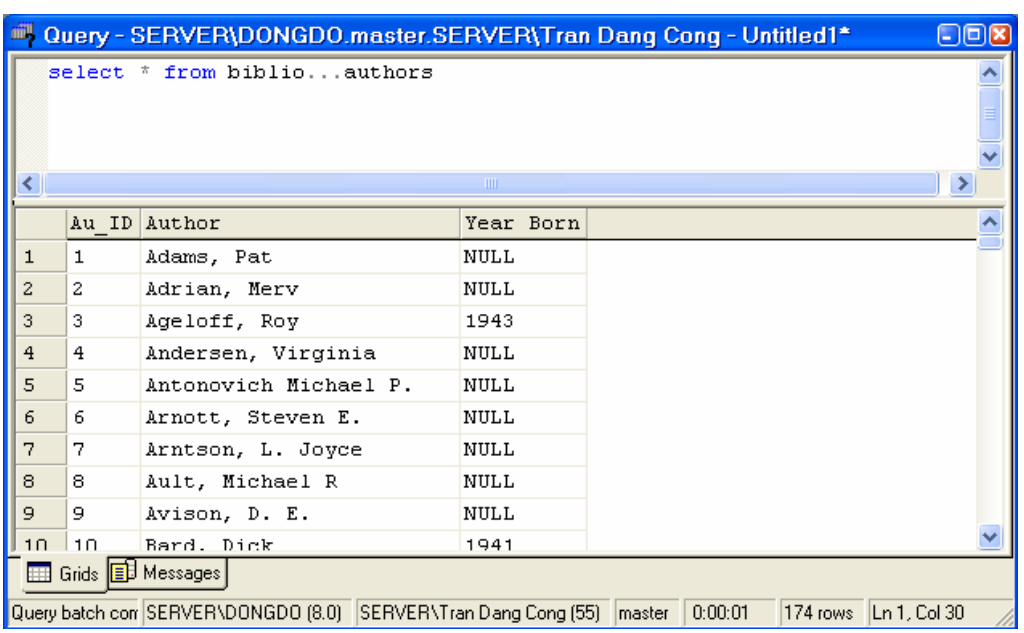
+ Provider (Driver của hệ quản trị CSDL cần thiết lập liên kết, trong ví dụ minh họa thực hiện với Access).



- Thực hiện khai thác thông qua câu lệnh, trong câu lSQL phải chỉ đủ đường dẫn, ví dụ

Select * from Biblio...Authors

sẽ thực hiện liệt kê toàn bộ danh sách các bản ghi của bảng authors.



Tạo bằng câu lệnh.

Sử dụng lệnh `sp_addlinkedserver` tạo server liên kết.

Ví dụ tạo Linked Server Biblio:

```
sp_addlinkedserver 'Biblio',
     'Access 97', 'Microsoft.Jet.OLEDB.4.0',
                              'c:\data\biblio.mdb'
```

(Đường dẫn phải phù hợp với Server)

Xóa Linked Server.

- Sử dụng công cụ: Chọn Linked Server cần xóa -> thực hiện xóa.

- Sử dụng lệnh :

sp_dropserver [**@server =**] '*server*'
[**,** [**@droplogins =**] { '**droplogins**' | NULL}]

SỬ DỤNG BCP VÀ BULK INSERT NHẬP DỮ LIỆU.

Bcp là câu lệnh dạng command prompt, dùng xuất (export) và nhập (import) dữ liệu giữa SQL Server và tập tin (dạng text hoặc excel). Các tập tin tham gia xuất nhập phải có cấu trúc dữ liệu kiểu bảng (hàng, cột), bảng dữ liệu của SQL Server khi thực hiện nhập dữ liệu phải có cấu trúc tương đương có sẵn.

Bulk insert là câu lệnh tương tự bcp nhưng chỉ thực hiện import dữ liệu mà không export.

Cú pháp lệnh bcp.

Lệnh bcp được thực hiện tại cửa sổ lệnh (command prompt).

bcp {[[*database_name***.**][*owner*]**.**]{*table_name* | *view_name*} | "*query*"}
{**in** | **out** | **queryout** | **format**} *data_file*
[**-m** *max_errors*] [**-f** *format_file*] [**-e** *err_file*]
[**-F** *first_row*] [**-L** *last_row*] [**-b** *batch_size*]
[**-n**] [**-c**] [**-w**] [**-N**] [**-V** (**60** | **65** | **70**)] [**-6**]
[**-q**] [**-C** *code_page*] [**-t** *field_term*] [**-r** *row_term*]

[**-i** *input_file*] [**-o** *output_file*] [**-a** *packet_size*]
[**-S** *server_name*[*****instance_name*]] [**-U** *login_id*] [**-P** *password*]
[**-T**] [**-v**] [**-R**] [**-k**] [**-E**] [**-h** "*hint* [**,**...*n*]"]

(bạn tìm hiểu thêm trong book online)

Ví dụ sử dụng lệnh bcp.

+ Sử dụng lệnh có từ khóa out để copy toàn bộ dữ liệu từ một bảng hoặc khung nhìn ra tập tin.

```
bcp pubs..titleview out titleview.txt -c -Sservername -
Uusername -Ppassword
```

+ Sử dụng lệnh Select để copy một tập ra tập tin, có từ khóa queryout.

```
bcp "SELECT au_fname, au_lname FROM pubs..authors ORDER
BY au_lname" queryout c:\Authors.txt -c -Sservername -
Uusername -Ppassword
```

Kết quả thực hiện: Nội dung tập tin Authors.txt

Abraham	Bennet
Reginald	Blotchet-Halls
Cheryl	Carson
Michel	DeFrance
Innes	del Castillo
Ann	Dull
Marjorie	Green
Morningstar	Greene
Burt	Gringlesby
Sheryl	Hunter
Livia	Karsen

Charlene Locksley

Stearns MacFeather

Heather McBadden

Michael O'Leary

Sylvia Panteley

Albert Ringer

Anne Ringer

Meander Smith

Dean Straight

Dirk Stringer

Johnson White

Akiko Yokomoto

Một số tham số cơ bản:

- Out: Copy toàn bộ một Table hoặc view ra tập tin.

- Queryout: Copy tập dữ liệu được truy vấn theo câu lệnh.

- c: Chỉ ra rằng câu lệnh dùng kiểu ký tự để phân định các cột, nếu không chỉ thì câu lệnh tự nhận tab (\t) để phân định và dùng new line để xuống dòng mới.

Cú pháp lệnh Bulk Insert.

Lệnh Bulk Insert gần giống lệnh bcp nhưng Bulk Insert chỉ sử dụng để nhập dữ liệu vào SQL Server (Insert), lệnh này được thực hiện bằng SQl Query Analyzer.

Cú pháp chung:

```
BULK INSERT [ [ 'database_name'.] [ 'owner' ].] { 'table_name' FROM 'data_file' }
  [ WITH
    (
      [ BATCHSIZE [ = batch_size ] ]
      [ [ , ] CHECK_CONSTRAINTS ]
      [ [ , ] CODEPAGE [ = 'ACP' | 'OEM' | 'RAW' | 'code_page' ] ]
      [ [ , ] DATAFILETYPE [ =
```

```
        { 'char' | 'native'| 'widechar' | 'widenative' } ] ]
      [ [ , ] FIELDTERMINATOR [ = 'field_terminator' ] ]
      [ [ , ] FIRSTROW [ = first_row ] ]
      [ [ , ] FIRE_TRIGGERS ]
      [ [ , ] FORMATFILE = 'format_file_path' ]
      [ [ , ] KEEPIDENTITY ]
      [ [ , ] KEEPNULLS ]
      [ [ , ] KILOBYTES_PER_BATCH [ = kilobytes_per_batch ] ]
      [ [ , ] LASTROW [ = last_row ] ]
      [ [ , ] MAXERRORS [ = max_errors ] ]
      [ [ , ] ORDER ( { column [ ASC | DESC ] } [ ,...n ] ) ]
      [ [ , ] ROWS_PER_BATCH [ = rows_per_batch ] ]
      [ [ , ] ROWTERMINATOR [ = 'row_terminator' ] ]
      [ [ , ] TABLOCK ]
    )
  ]
```

Ví dụ thực hiện copy toàn bộ dữ liệu tập tin newpubs.dat vào bảng publishers2 (bảng này đã có cấu trúc tương ứng), tập tin kiểu ký tự, ngăn cách giữa các cột là dấu phẩy ‘,’, xuống dòng mới bằng ký tự ‘\n’ (xuống dòng dưới và chuyển về đầu dòng).

```
BULK INSERT pubs..publishers2 FROM 'c:\newpubs.dat'
WITH (
   DATAFILETYPE = 'char',
   FIELDTERMINATOR = ',',
   ROWTERMINATOR = '\n'
)
```

DETTACH VÀ ATTACH CƠ SỞ DỮ LIỆU.

Mục này sẽ nói về ký thuật hủy và nối ghép tập tin CSDL với Server. Giả sử bạn đã có các tập tin của CSDL (gồm tập tin dữ liệu và nhật ký có thể được copy từ vị trí khác).

Copy tập tin của CSDL.

Trước tiên ta xem kỹ thuật copy các tập tin CSDL sang một vị trí khác (mà vẫn giữ vị trí), sau khi copy sang vị trí khác bạn có thể sử dụng sang Instance mới. Các bước thực hiện như sau:

- Stop dịch vụ SQL của Instance có CSDL.

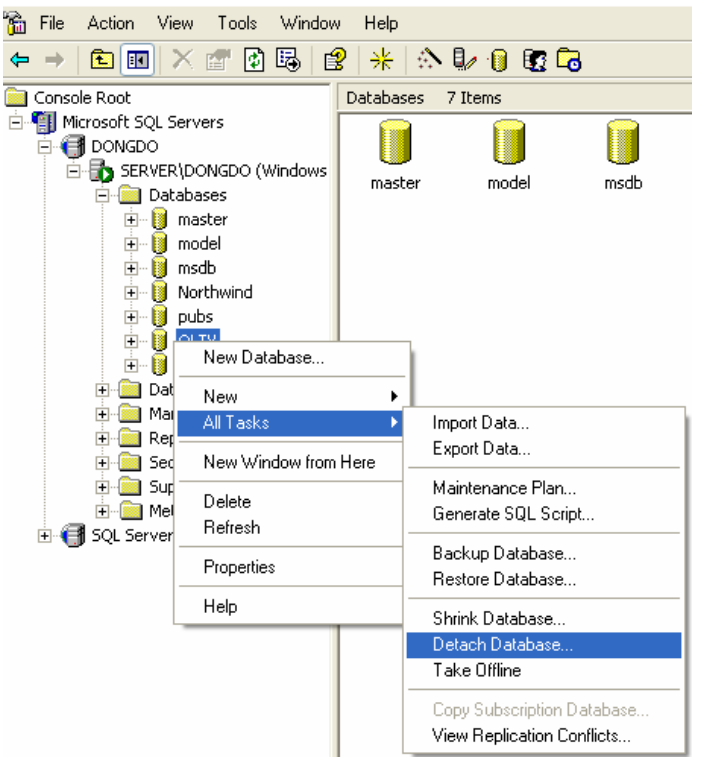
- Copy các tập tin của CSDL sang vị trí cần thiết.

- Start dịch vụ SQL của Instance để tiếp tục làm việc.

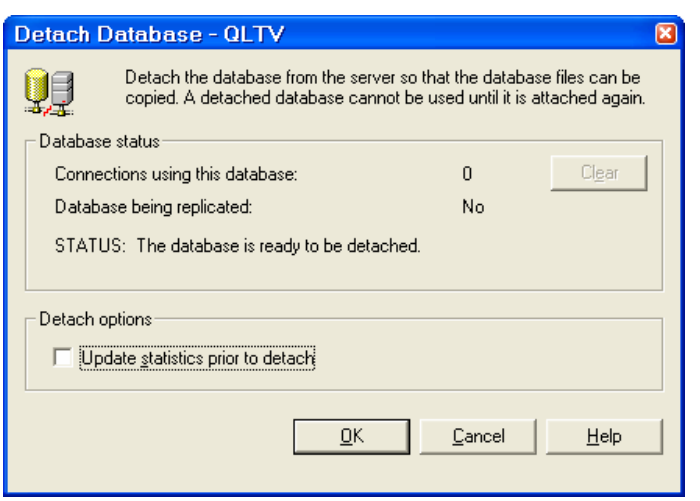
Dettach cơ sở dữ liệu.

Là bước thực hiện tách CSDL khỏi Instance, Ínntance không quản lý CSDL nhưng khác với xóa CSDL là các tập tin chứa CSDL vẫn còn.

- Chọn CSDL cần dettach. -> All tasks -> Dettach Database



- Nhấn Ok.

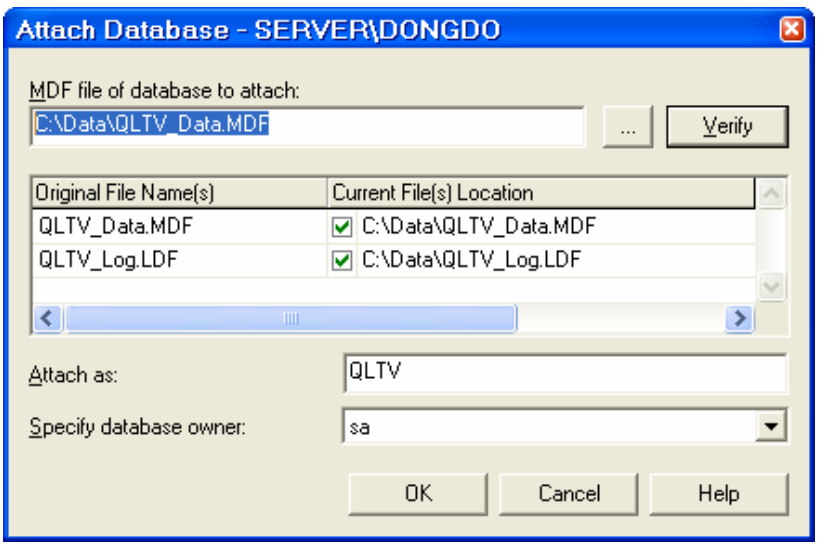


Attach tập tin CSDL vào Instance.

Mục này giới thiệu kỹ thuật ghép nối tập tin CSDL vào Instance, là bước tiếp theo của các bước Copy và Dettach. Các bước thực hiện như sau:

- Chọn Instance cần Attach CSDL -> Databases -> all tasks -> attach database...

- Chọn nút browse (...)



- Chọn tập tin mdf của CSDL cần attach.

- Đặt tên CSDL.

- Xác định User owner.

- Ok.

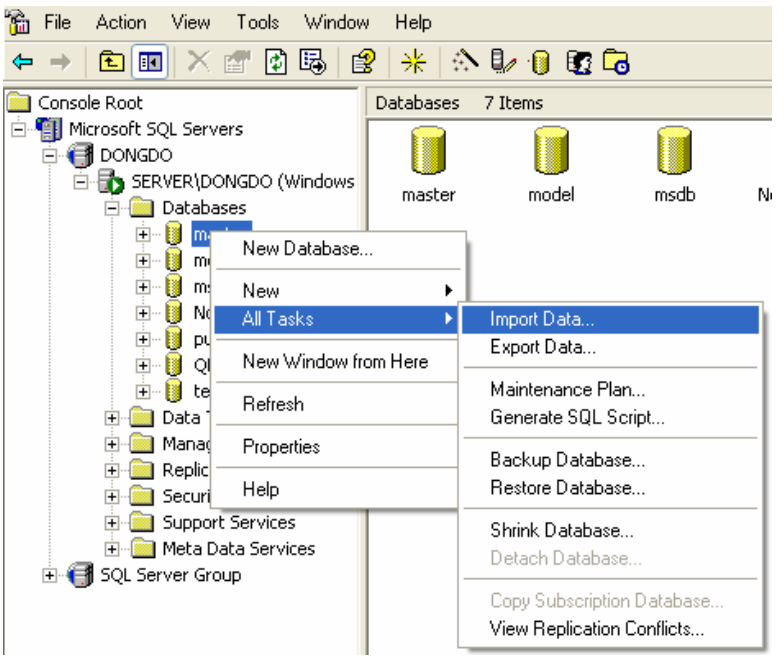
IMPORT VÀ EXPORT CƠ SỞ DỮ LIỆU.

Phần này sẽ trình bày kỹ thuật nhập và xuất dữ liệu từ CSDL với các hệ quản trị CSDL khác hoặc Instance, CSDL khác của SQL Server.

Import – Nhập dữ liệu.

Dùng nhập dữ liệu từ ngoài vaof CSDL từ hệ quản trị CSDL khác hoặc CSDL khác của SQL Server.

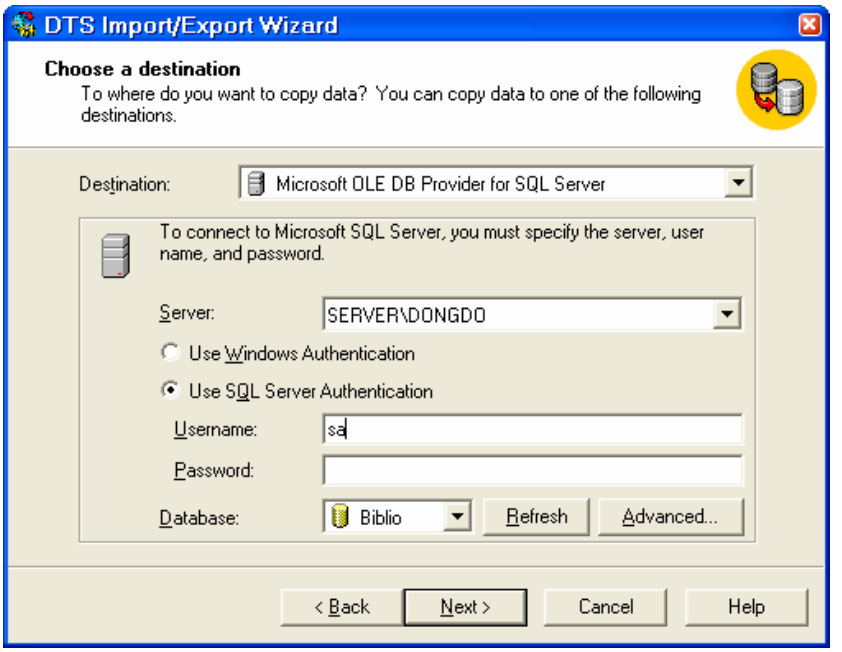
- Chọn Databases -> All tasks -> Import Data...



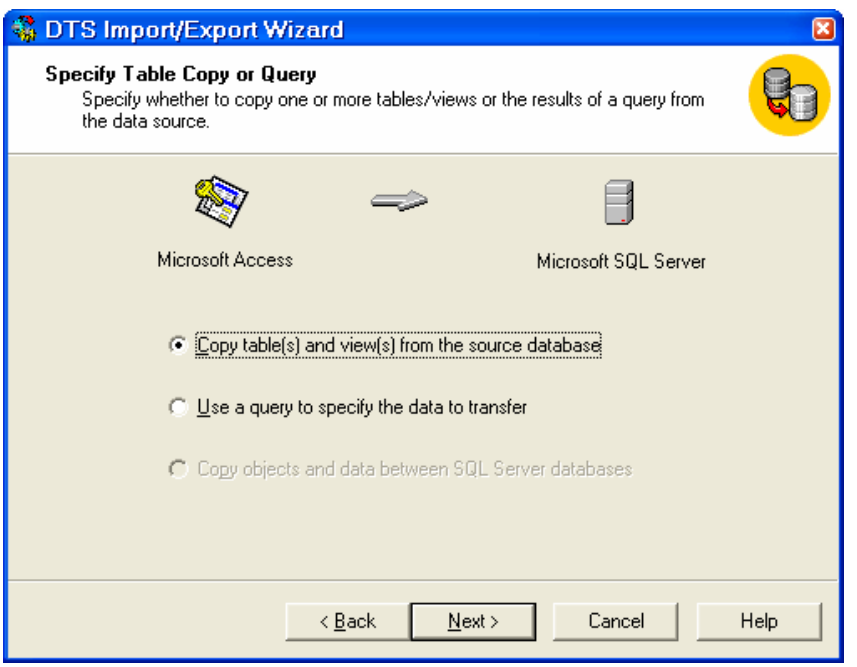
- Next -> Chọn Data Source (Có thể là SQL Server, Oracle, Access,...), trong ví dụ minh họa chọn Access.

- Chọn tập tin (file name) -> Next

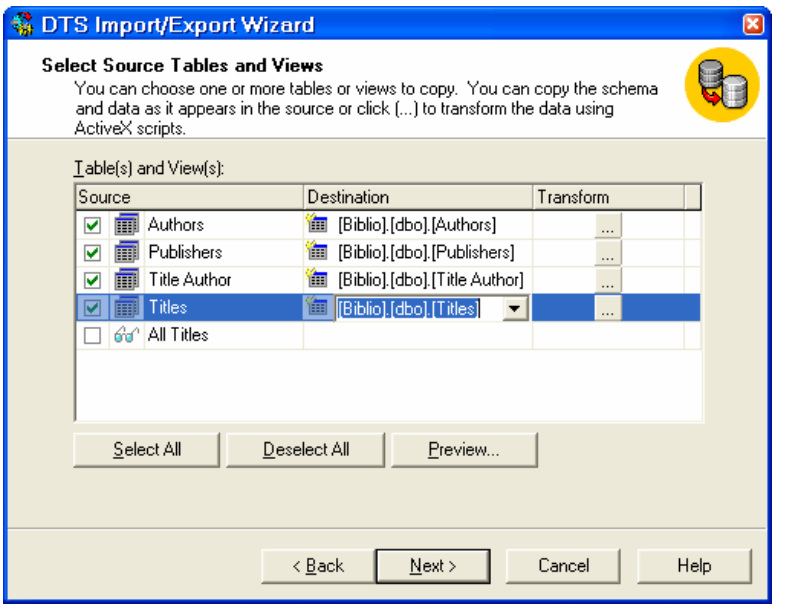
- Chọn Instance cần chuyển dữ liệu vào, user name., tên CSDL (có trước hoặc tạo tại thời điểm này bằng cách chọn New) -> Next



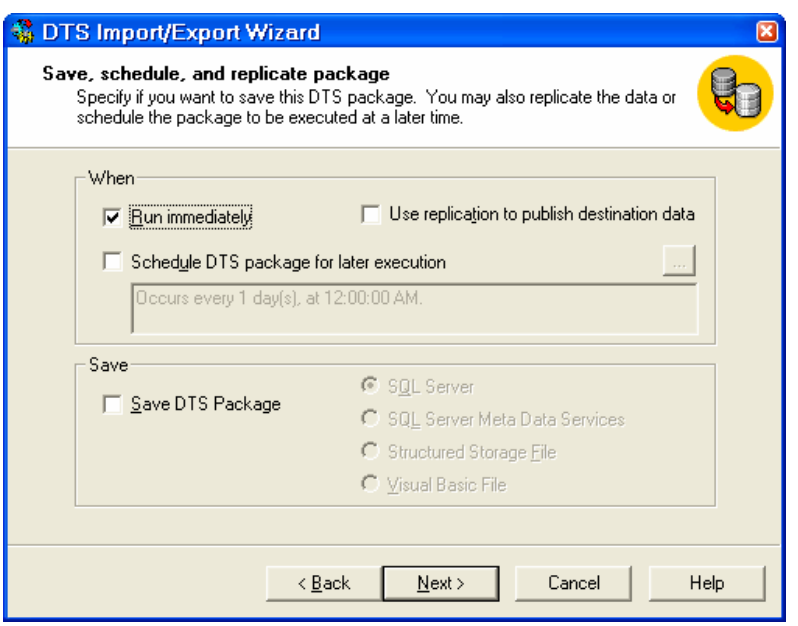
- Chọn cách chuyển toàn bộ bảng dữ liệu hay thông qua câu lệnh truyên vấn (trong ví dụ minh họa chọn bảng dữ liệu) -> Next



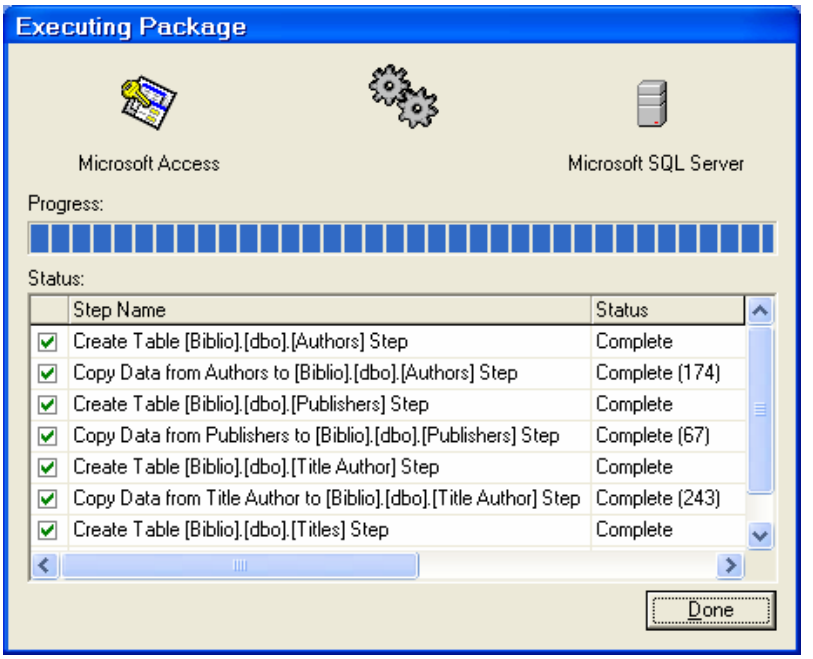
- Chọn các bảng, khung nhìn cần Import (có thể lựa chọn một số chức năng khác cụ thể hơn, bạn đọc tự tìm hiểu), tên các bảng, khung nhìn của SQL Server nhận dữ liệu -> Next.



- Chọn chức năng thực hiện ngày hay theo lịch –> Next -> Finish

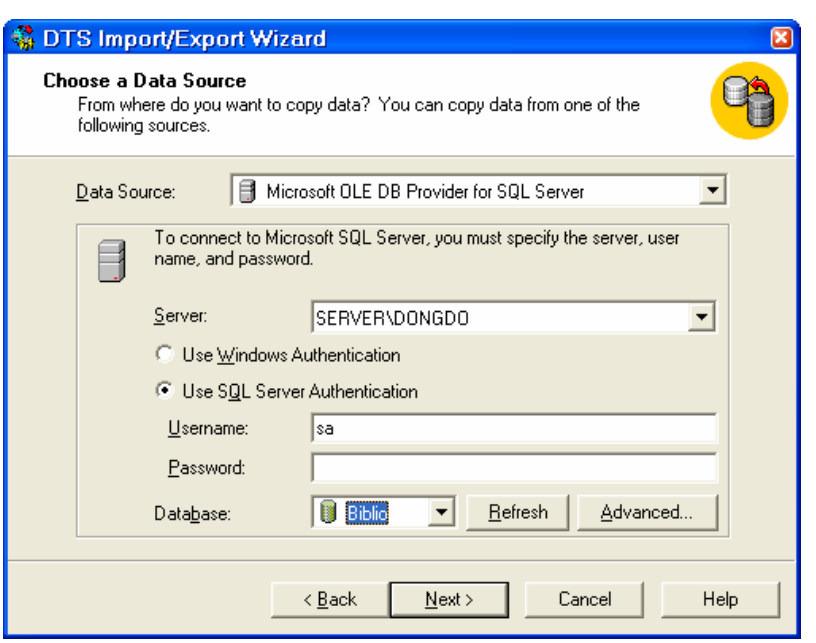


- Xem thông báo sau khi chuyển -> Done



EXPORT – XUẤT DỮ LIỆU.

Phần này giới thiệu kỹ thuật xuất dữ liệu từ một CSDL của SQL Server ra một hệ quản trị CSDL khác hoặc một CSDL khác của SQL Server. Tương tự như Import nhưng Export thực hiện Data Source là SQL Server, còn Destination là hệ quản trị CSDL khác hoặc CSDL khác của SQL Server (phần này bạn đọc tự xem xét).



SAO LƯU, KHÔI PHỤC DỮ LIỆU

Chương này sẽ giới thiệu kỹ thuật sao lưu (backup) và khôi phục (restore) dữ liệu, là kỹ thuật thường được sử dụng bảo đảm an toàn dữ liệu phòng trường hợp CSDL bị hỏng, nhật ký dữ liệu. Chức năng này được thực hiện bằng 2 phương pháp: Bằng công cụ và câu lệnh T-SQL.

NHỮNG LÝ DO PHẢI SAO LƯU VÀ KHÔI PHỤC DỮ LIỆU.

Trong quá trình thực hiện quản trị CSDL SQL Server thì một số nguyên nhân sau đây bắt buộc bạn phải xem xét đến kỹ thuật sao lưu và khôi phục dữ liệu:

+ Ổ đĩa bị hỏng (chứa các tập tin CSDL).

+ Server bị hỏng.

+ Nguyên nhân bên ngoài (thiên nhiên, hỏa hoạn, mất cắp,...)

+ User vô tình xóa dữ liệu.

+ Bị vô tình hay cố ý làm thông tin sai lệch.

+ Bị hack.

CÁC LOẠI BACKUP.

Backup dữ liệu trong SQL Server gồm các loại sau:

+ Full Database Backups: Copy toàn bộ CSDL (các tập tin bao gồm các bảng, khung nhìn, các đối tượng khác).

+ Differential Database Backups: Copy những dữ liệu thay đổi trong Data file kể từ lần full backup gần nhất.

+ File or file group backups: Copy một file đơn hay file group.

+ Differential File or File Group Backups: Thực hiện như Differential Database nhưng copy phần dữ liệu thay đổi của file đơn hoặc file group.

+ Transaction log backups: Ghi nhận tất cả các transaction chứa trong transaction log file kể từ lần transaction log backup gần nhất. Với loại sao lưu này ta có thể khôi phục dữ liệu tại một thời điểm.

CÁC MÔ HÌNH PHỤC HỒI DỮ LIỆU.

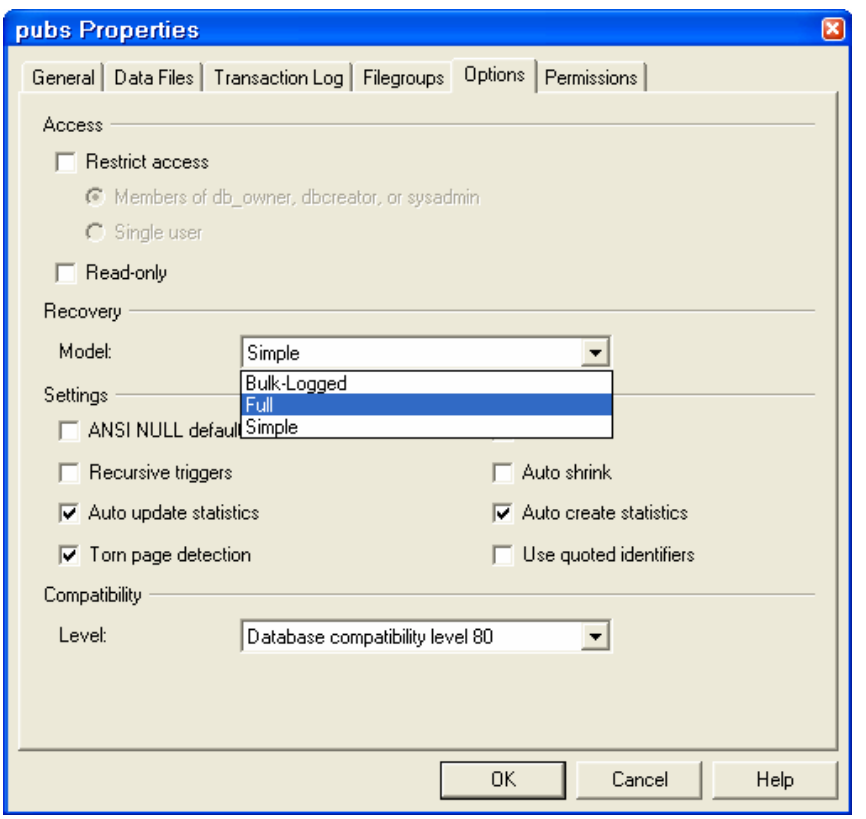
+ Full Recovery model: Là mô hình phục hồi toàn bộ hoạt động giao dịch của dữ liệu (Insert, Update, Delete, hoạt động bởi lệnh bcp, bulk insert). Với mô hình này ta có thể phục hồi dữ liệu tại một thời điểm trong quá khứ đã được lưu trong transaction log file.

+ Bulk-Logged Recovery Model: Mô hình này được thực thi cho các thao tác bcp, bulk insert, create index, writetext, updatetext, các hoạt động này chỉ nhật ký sự kiện vào log để biết mà không sao lưu toàn bộ dữ liệu, chi tiết như trong full recover. Các sự kiện Insert, Update, Delete vẫn được nhật ký và khôi phục bình thường.

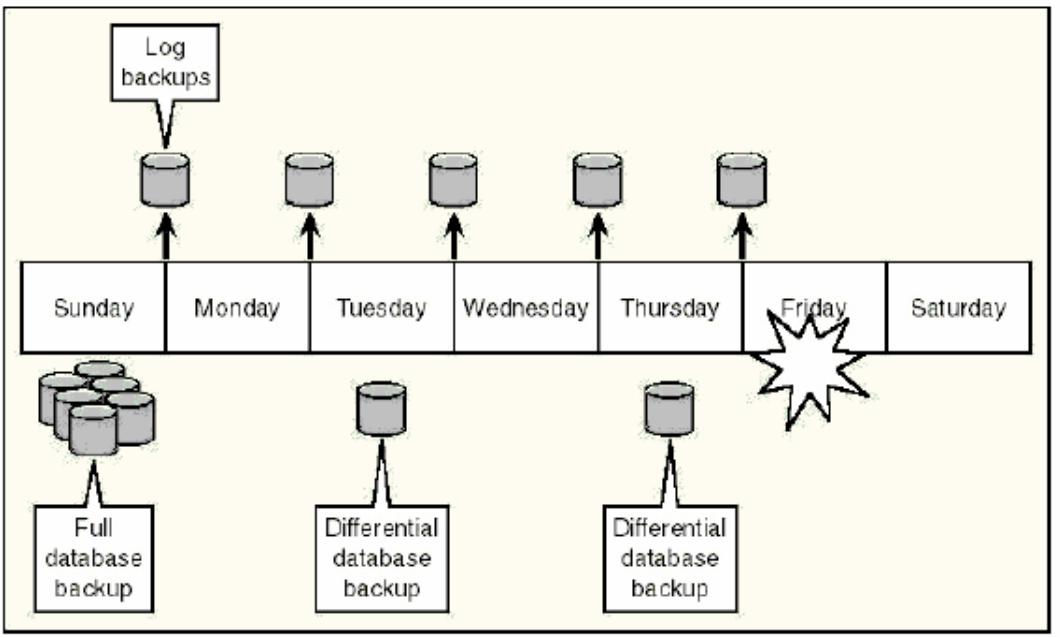
+ Simple Recovery Model: Với mô hình này bạn chỉ phục hồi lại thời điểm backup gần nhất mà không theo thời điểm khác trong quá khứ.

Cách đặt mô hình khôi phục:

- Chọn CSDL.
- Nhấn nút phải chuột -> Properties -> Options -> Recovery



Xét ví dụ sau: Giả sử ta có một CSDL được backup theo chiến lược như hình vẽ:



Nhìn hình trên ta thấy CSDL được lập lịch Full Database Backup vào ngày chủ nhật, Differential Database Backup vào ngày thứ ba và thứ năm, còn Log Database Backup vào 5 ngày trong tuần, ngày thứ sáu có sự cố với CSDL data file bị hỏng, vấn đề đặt ra là phải phục hồi dữ liệu và CSDL hoạt động bình thường. Ta phải làm các bước sau:

+ Thực hiện Backup log file (giả sử log file không bị hỏng).

+ Khôi phục Full Database của ngày chủ nhật.

+ Phục hồi Differential Database của ngày thứ năm.

+ Khôi phục Transaction log backup ngày thứ năm.

SAO LƯU CƠ SỞ DỮ LIỆU - BACKUP DATABASE.

Trước khi xem xét kỹ thuật sao lưu CSDL, ta thống nhất một số thuật ngữ bằng tiếng Anh như sau:

+ Backup: Là quá trình copy toàn bộ hoặc một phần database, transaction log, file, file group thành lập một backup set được chứa trong backup media (disk hoặc tape) bằng cách sử dụng một backup device (tape drive name hoặc physical filename).

+ Backup Device: Một file vật lý hoặc một drive tape.

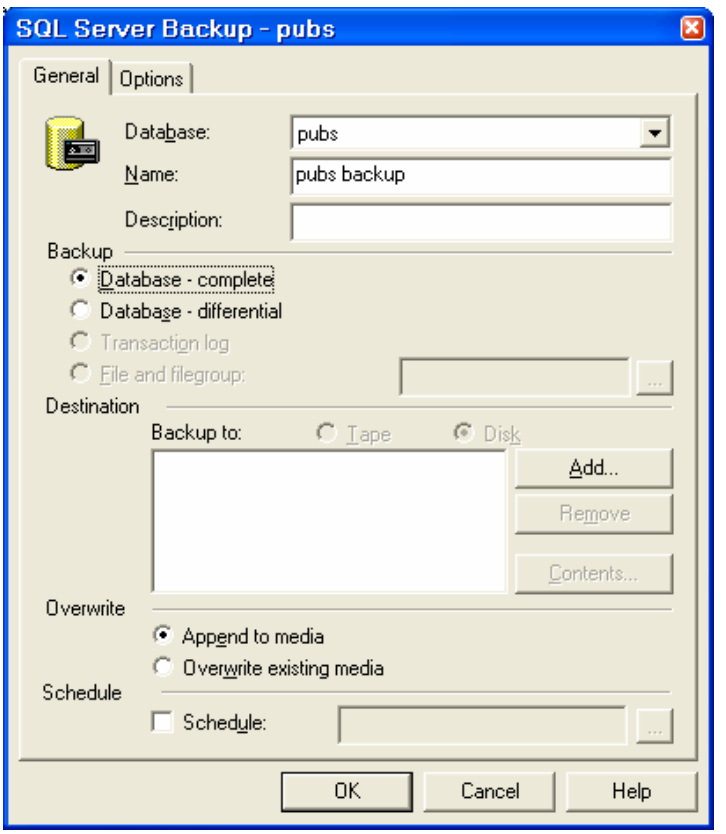
+ Backup file: Một file chứa Backup set.

+ Backup media: LÀ Disk hoặc tape.

+ Backup set: Một bộ backup một lần backup đơn chứa trên backup media.

Các bước thực hiện backup như sau:

- Chọn CSDL cần backup.
- Nhấn phải chuột -> All Tasks -> Backup Database…



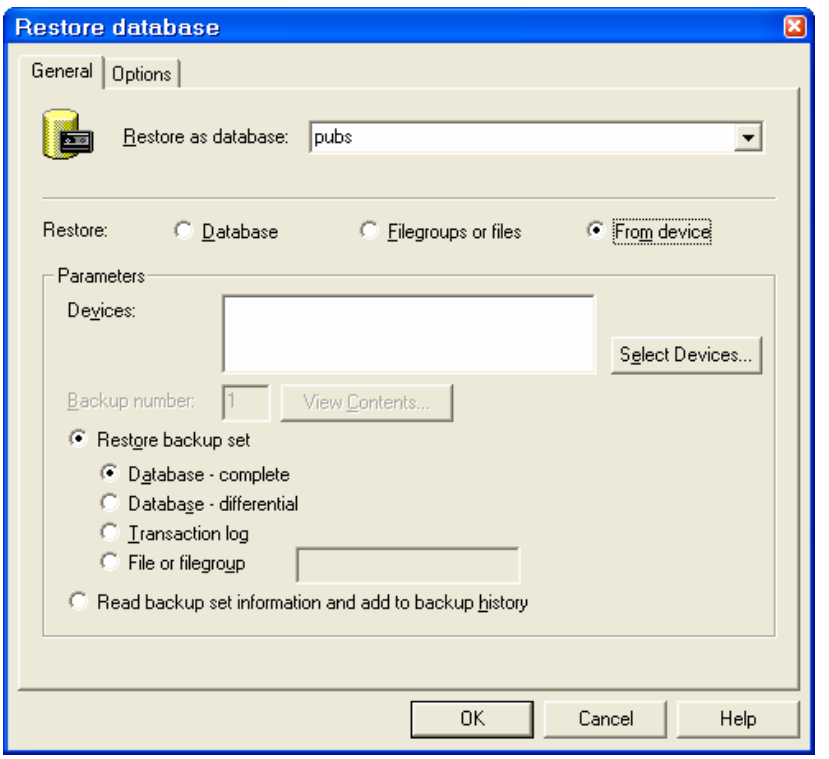
- Nhập các tham số, lựa chọn kiểu.

KHÔI PHỤC DỮ LIỆU – RESTORE DATABASE.

Là chức năng thực hiện khôi phục dữ liệu đã sao lưu, tùy theo chiến lược backup mà bạn có thể phục hồi đến thời điểm nào, thu được bộ dữ liệu trong quá khứ như thế nào. Khôi phục dữ liệu được thực hiện theo thứ tự backup, thông tin này được lưu trữ trong msdb

Các bước thực hiện như sau:

- Chọn mục Databases -> Nhấn nút phải chuột -> All Tasks -> Restore Database…



- Nhập tham số, chọn mô hình khôi phục.

PHÂN QUYỀN, BẢO MẬT

Chương này sẽ giới thiệu bạn đọc kỹ thuật phân quyền, quản lý người dùng, đặt các mức bảo mật cho CSDL.

CHẾ ĐỘ BẢO MẬT – SECURITY MODE.

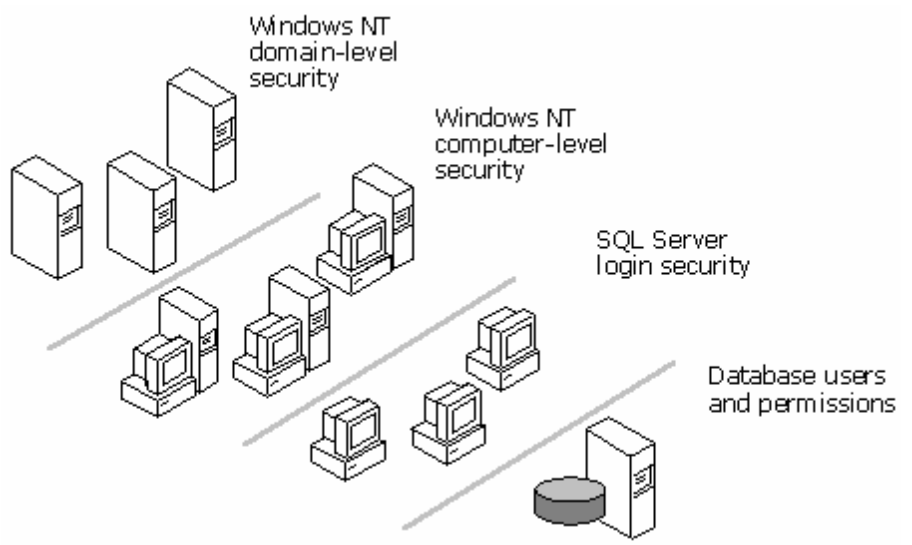
Như đã gặp trong phần cài đặt SQL Server, SQL Server có 2 chế độ bảo mật:

+ Windows Authentication Mode (Windows Authentication)

+ Mixed Mode (Windows Authentication and SQL Server Authentication)

Windows Authentication.

Là chế độ bảo mật mà những User truy nhập SQL Server phải là những User của Windows. Khi Server đặt ở chế độ bảo mật này, những User phải là những User được Windows quản lý mới được truy nhập.

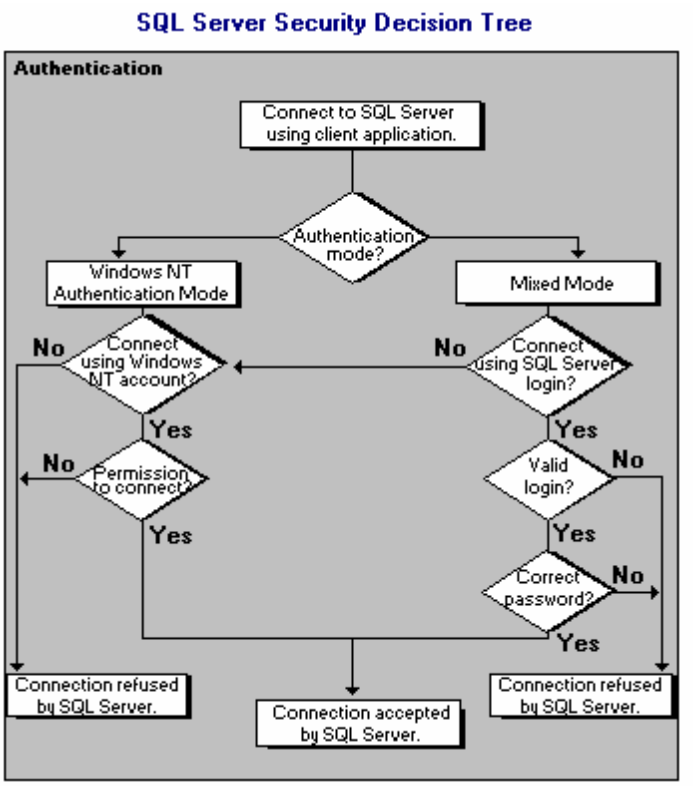


Nhìn trên hình ta thấy khi thực hiện chế độ này người sử dụng muốn khai thác SQL Server phải thông qua 4 bước xác thực (1- Domain, 2- Computer, 3- SQL Server, 4- Database).

SQL Server Authentication.

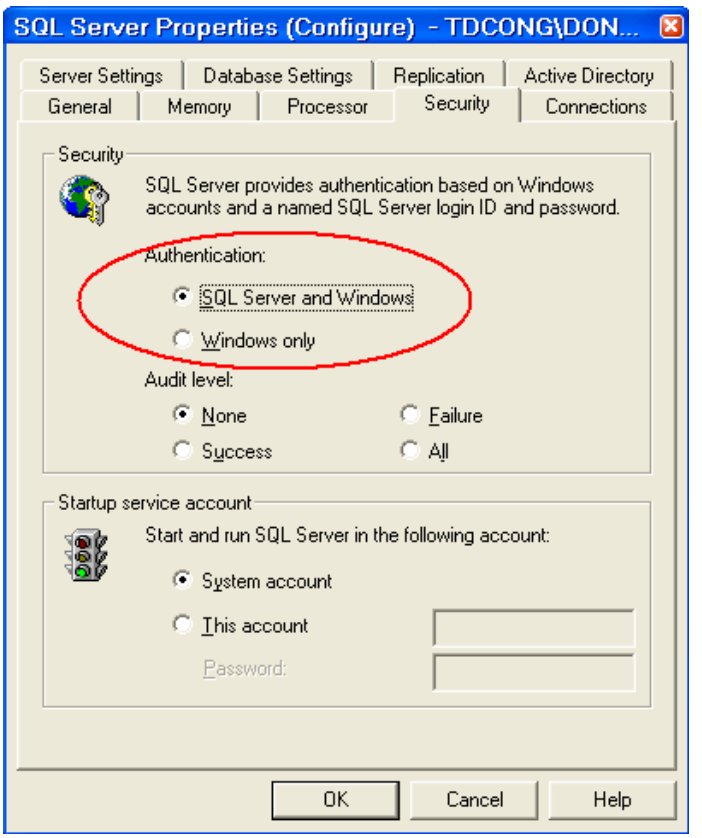
Khi thiết lập ở chế độ bảo mật này, những User được quyền khai thác phải là những User do quản trị SQL Server tạo ra, mà những user của Windows không được khai thác.

Tuy nhiên, SQL Server cho phép thiết lập hai chế độ Windows Authentication Mode (Windows Authentication) và Mixed Mode (Windows Authentication and SQL Server Authentication), chế độ Mixed Mode là sự kết hợp của Windows Authentication và SQL Server Authentication, ở chế độ này cả user của Windows và SQL Server để có thể thiết lập để truy nhập SQL Server.



Đặt chế độ.

- Nhấn phải chuột chọn tên Server (Instance).
- Chọn Properties.
- Chọn bảng Security.



- Chọn chế độ bảo mật -> Ok

SERVER ROLE, DATABASE ROLE.

Role là đối tượng xác định nhóm thuộc tính để gán quyền cho các user tham gia khai thác SQL Server.

Server Role.

Nhóm các quyền thực hiện quản trị hệ thống, gồm các nhóm sau:

+ Bulk Insert Administrators: Được phép thực hiện Bulk Insert.

+ Database Creators: Được phép tạo và sửa đổi cấu trúc CSDL.

+ Disk Administrators: Có thể quản trị các file trên đĩa.

+ Proccess Administrator: Quản trị các dịch vụ đang chạy của SQL Server.

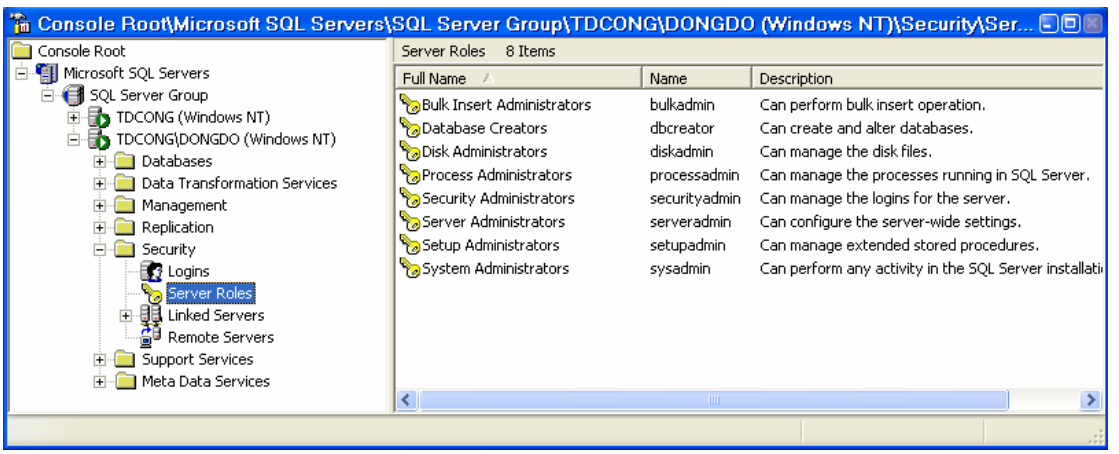
+ Security Administrators: Quản trị hệ thống bảo mật.

+ Setup Administrators: Quản trị các thủ tục mở rộng (xp_).

+ System Administrators: Quản trị hệ thống SQL Server.

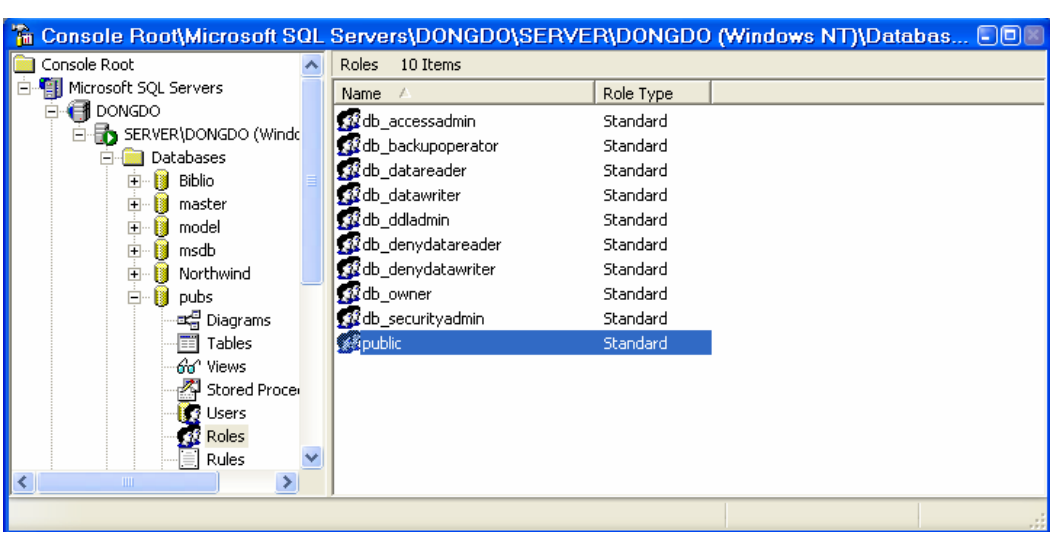
Xem cụ thể như sau:

- Mở rộng Server (nhấn dấu '+' phần tên Server).
- Mở rộng Security.
- Chọn Server Roles:



Database Role.

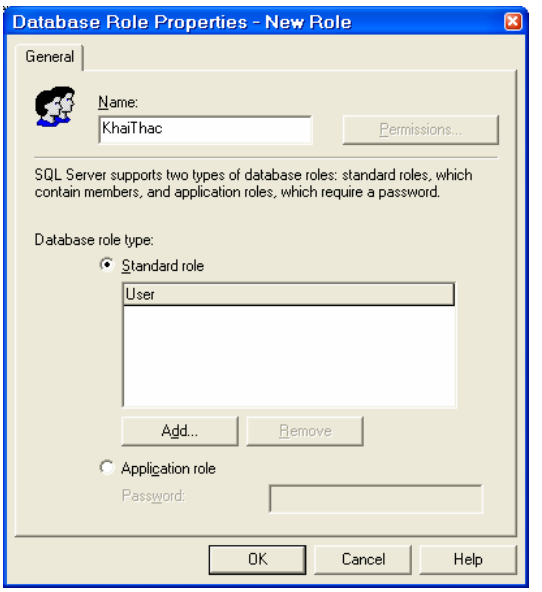
Role là đối tượng mà thông qua nó người quản trị có thể gán quyền khai thác cho người sử dụng. Role do CSDL quản lý, khi tạo CSDL hệ thống tự đặt một số Role ngầm định.



Người những Role ngầm định ta có thể tạo Role mới.

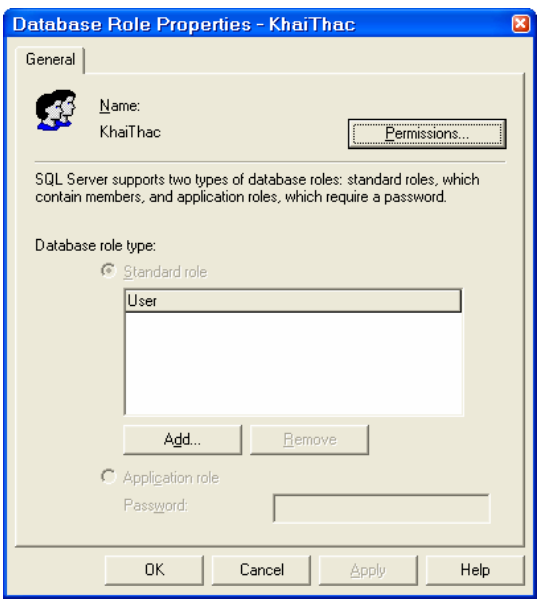
Tạo Role theo công cụ.

- Chọn Roles trong CSDL -> Nhấn phải chuột -> New Database Role..
- Đặt tên, chọn user (chọn user có thể làm sau).
- Nhấn Ok.



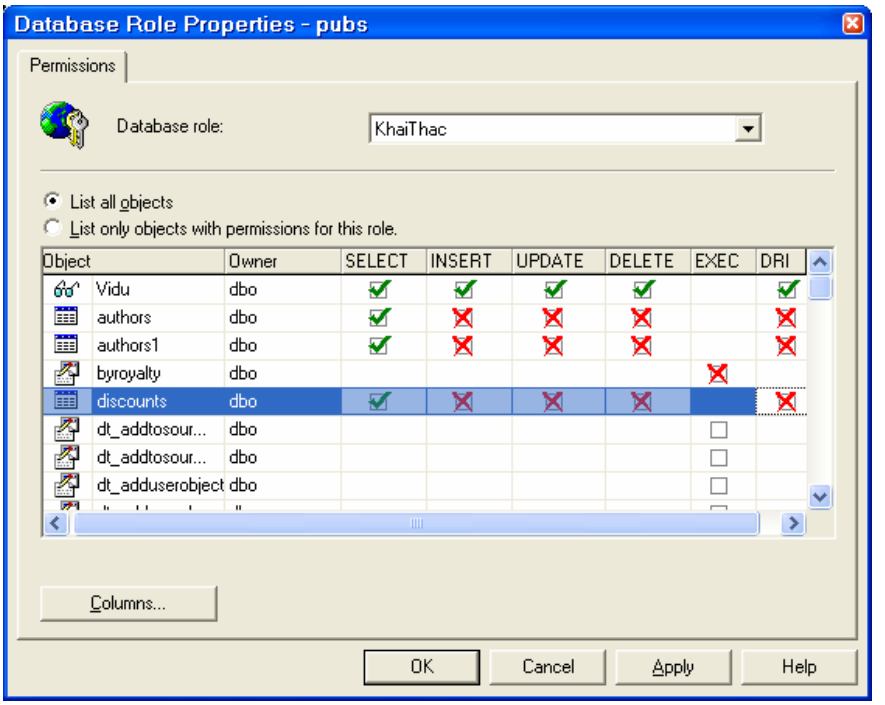
Sau khi tạo xong, thực hiện gán quyền khai thác cho Role.

- Chọn Role cần gán quyền.



- Chọn Perrmissions...

- Đặt các quyền cho từng đối tượng trong CSDL.



Nếu chọn quyền nhấn ô chọn xuất hiện dấu chọn màu xanh, nếu cấm nhấn ô chọn xuất hiện dấu màu đỏ. Có thể đặt quyền khai thác đối với role cho từng cột của bảng dữ liệu.

Mọi thao tác xóa, sửa được thực hiện như các đối tượng khác.

Tạo theo câu lệnh.

Sử dụng câu lệnh

sp_addrole [**@rolename** =] '*role*'
[**,** [**@ownername** =] '*owner*']

Ví dụ: Thêm Role có tên Managers:

```
EXEC sp_addrole 'Managers'
```

QUẢN TRỊ NGƯỜI DÙNG.

Người dùng trong SQL Server được chia thành 2 mức: Người truy nhập vào SQL Server gọi là Login, người khai thác CSDL gọi là User.

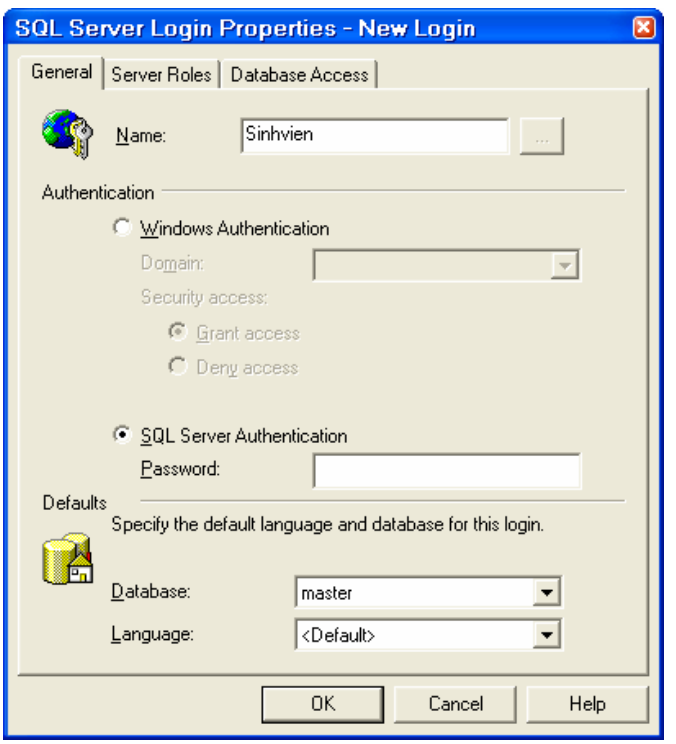
Login.

Là đối tượng được quyền truy nhập vào SQL Server, tùy theo chế độ bảo mật của SQL Server mà những login là account của Windows NT hay của SQL Server, login do Server quản lý trực tiếp.

Tạo bằng công cụ.

- Chọn chức năng Security của Server -> Logins

- Nhấn phải chuột -> New Login...



- Nhập các tham số: Nếu chọn Account của Windows NT thì bạn có thể chọn trong danh sách. Nếu tạo login của SQL Server thì bạn nhập tên mới, mật khẩu, chọn login thuộc server role nào, có thể gán quyền truy nhập khai thác CSDL nào.

Tạo bằng câu lệnh. Sử dụng câu lệnh

sp_addlogin [**@loginame** =] '*login*'
[**,** [**@passwd** =] '*password*']
[**,** [**@defdb** =] '*database*']
[**,** [**@deflanguage** =] '*language*']

[, [**@sid** =] *sid*]
[, [**@encryptopt** =] '*encryption_option*']

Ví dụ: Tạo login có tên ‘Albert’, mật khẩu ‘corporate’

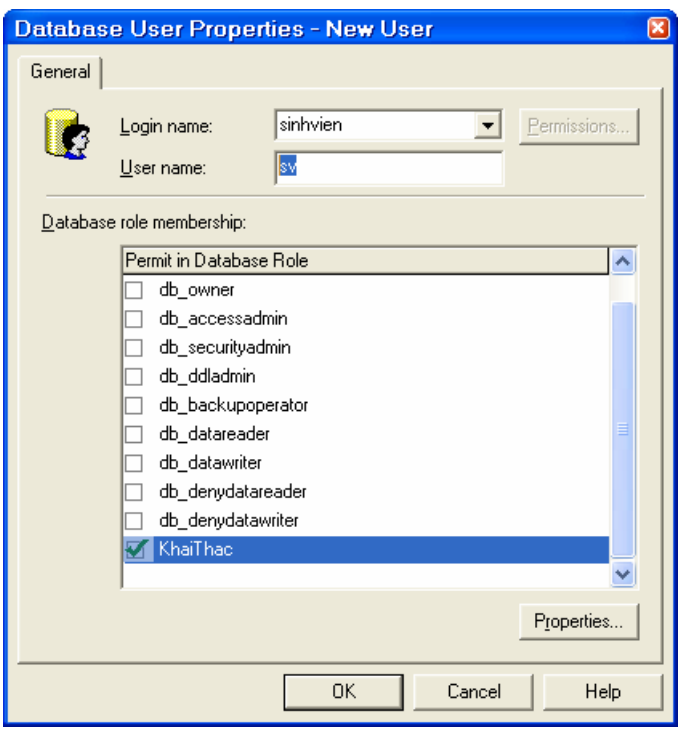
```
EXEC sp_addlogin 'Albert', 'food', 'corporate'
```

Mọi thao tác sửa, xóa được thực hiện như các đối tượng khác.

User.

User là đối tượng khai thác CSDL, nếu login chỉ xác định truy nhập vào SQL Server thì User là login ID tham gia khai thác CSDL, user do CSDL quản lý trực tiếp.

- Chọn CSDL -> users
- Nhấn phải chuột -> new user...



- Chọn Login, nhập user name, chọn role mà user thuộc ->Ok

Các thao tác xóa, sửa thực hiện như các đối tượng khác, để gán quyền cho user bạn có thể chọn lại user vừa tạo cho CSDL sau đó vào nhấn vào Permissions.

NHÂN BẢN DỮ LIỆU

Chương này bạn sẽ giới thiệu với bạn kỹ thuật làm giảm lưu lượng dữ liệu giao dịch với SQL Server khi đã cấu hình nhiều Server trên mạng.

GIỚI THIỆU VỀ NHÂN BẢN DỮ LIỆU.

Nhân bản dữ liệu tên tiếng anh gọi là Replication, là công cụ được sử dụng copy một hoặc nhiều CSDL đến một hoặc nhiều server (SQL Server) khác, các Server được đặt trong mạng máy tính nội bộ (LAN), người khai thác có thể thực hiện truy nhập đến CSDL có trong Server được chuyển dữ liệu đến. Dữ liệu giữa các máy được thực hiện đồng bộ với nhau theo lịch hoặc theo sự kiện, khi có yêu cầu. Nhân bản dữ liệu có những ưu điểm sau:

+ Dữ liệu được lưu trữ ở nhiều nơi, hiệu quả trong việc có nhiều ứng dụng cùng truy nhập, khai thác.

+ Thích hợp các ứng dụng phân tích dữ liệu OLTP của DataWare House.

+ Có thể khai thác dữ liệu khi không kết nối đến Server.

+ Giảm thiểu xung khắc do số lượng lớn các giao dịch trên mạng.

+ Là một giải pháp an toàn khi Server bị lỗi hoặc bảo dưỡng.

Mô hình nhân bản.

Dịch vụ nhân bản dữ liệu gồm các thành phần cơ bản sau: Publisher, Distributor, Subscribers, Publications, Articles, Subscriptions.

Publisher: Là server cung cấp dữ liệu nhân bản cho các server khác. Một publisher có thể thiết lập nhiều bộ dữ liệu nhân bản (gọi là publication).

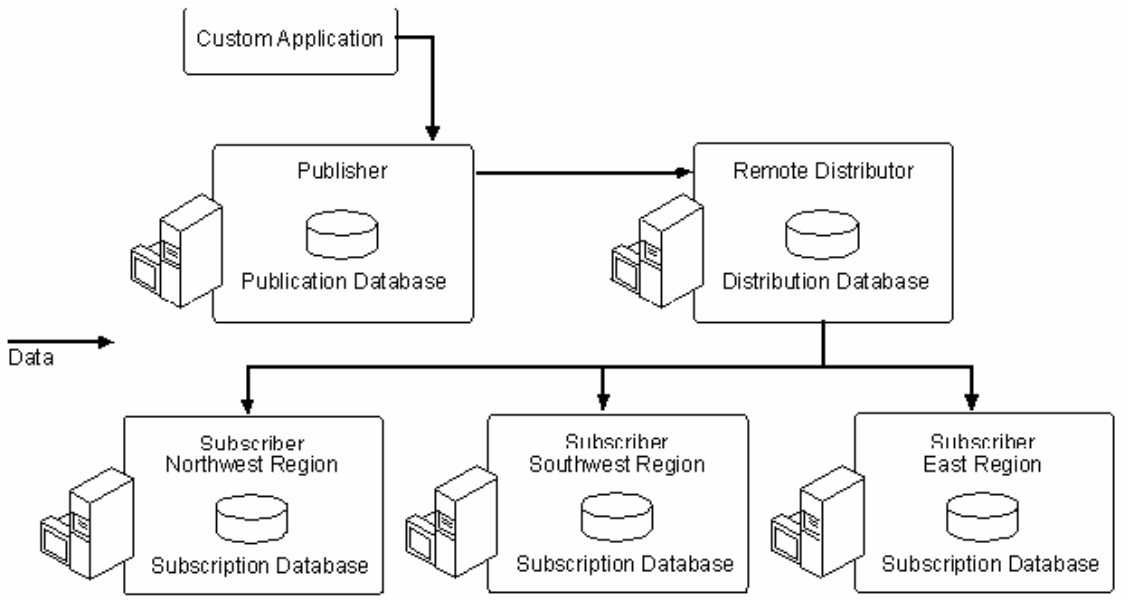
Distributor: Là server quản lý các thông tin nhân bản, lưu trữ dữ liệu trong các giao dịch thực hiện nhận và chuyển dữ liệu từ Publisher đến các Subscriber. Remote distributor là server tách rời khỏi publisher và được cấu hình là distributor. Local distributor là một server được cấu hình là Publisher và Distributor.

Subscriber: Là server nhận dữ liệu nhân bản. Subscriber gắn liền với publication (là máy chủ nhận dữ liệu nhân bản của một bộ dữ liệu cấu hình nhân bản).

Article: Là một bảng, tập dữ liệu hoặc đối tượng của CSDL cấu hình để nhân bản.

Publication: Là một tập gồm một hoặc nhiều article.

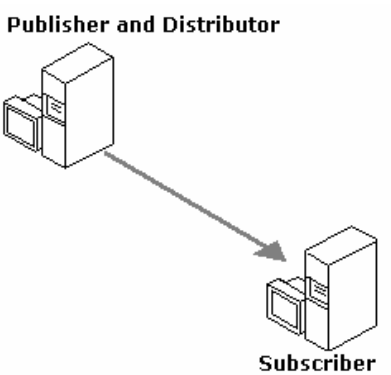
Subscription: Là một giao dịch yêu cầu bản sao bộ dữ liệu hoặc các đối tượng của CSDL thực hiện nhân bản. Trong mỗi giao dịch publisher thực hiện dẩy (push subscription) dữ liệu, subscriber thực hiện kéo (pull subscription).



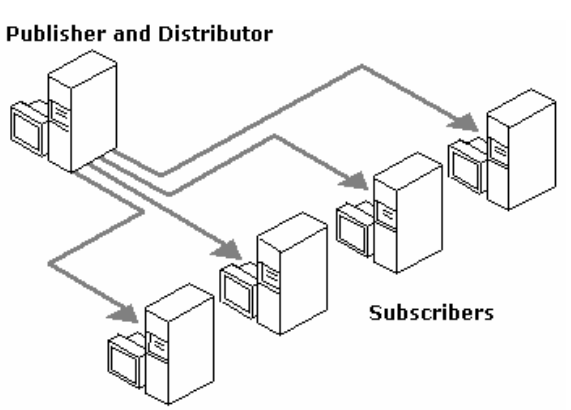
Nhân bản dữ liệu được thực hiện theo những mô hình cơ bản sau:

+ *Central Publisher*: Là mô hình Publisher và Distributor thiết lập trên một máy. Gồm các mô hình sau:

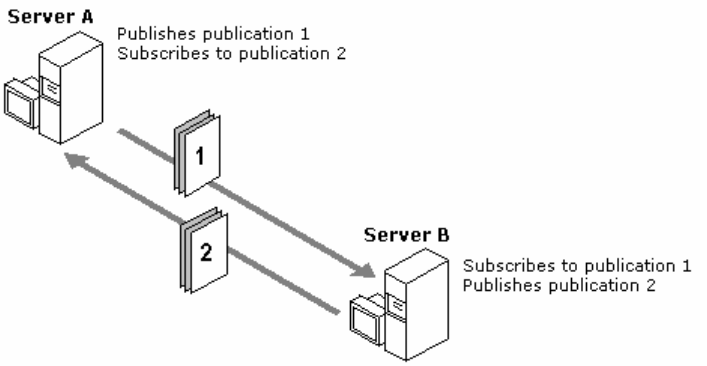
- Một Publishers và một Subscriber:



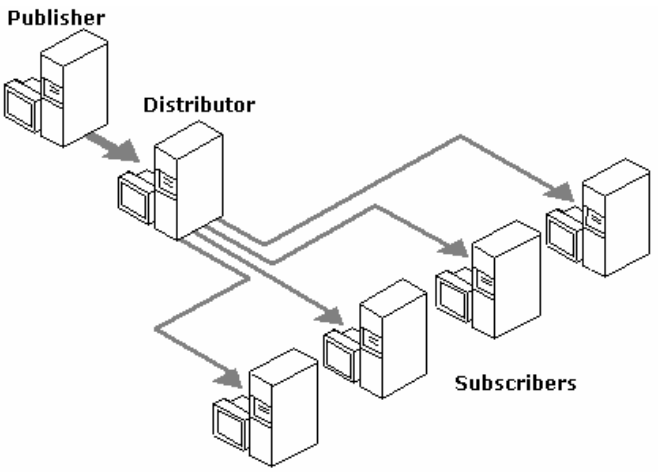
- Một Publisher và nhiều Subscriber.



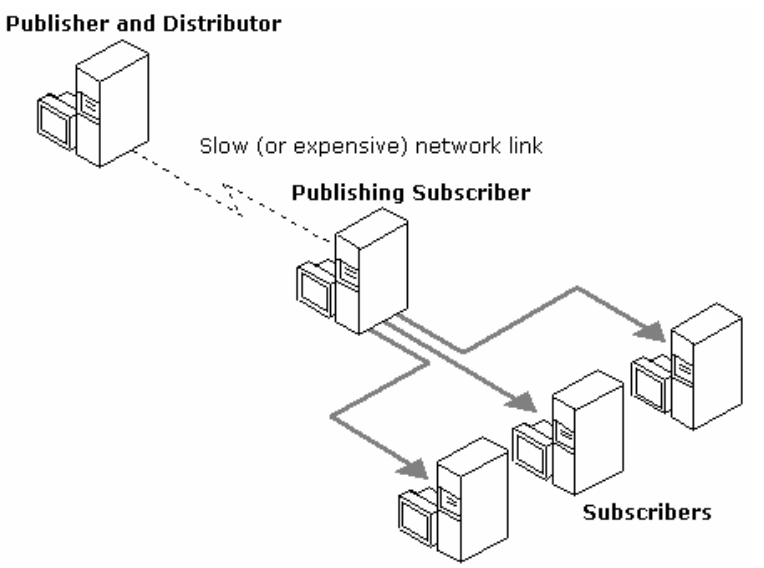
- Publisher và Subscriber được thiết lập trên một máy:



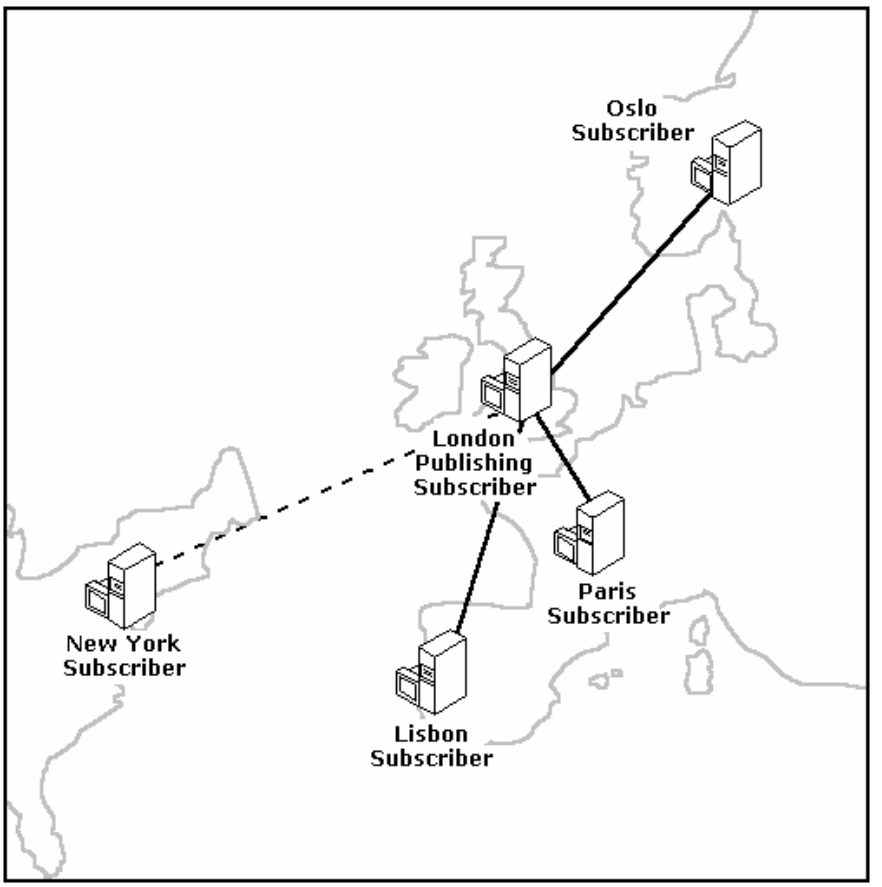
+ *Publisher và Distributor không thiết lập trên một máy*:



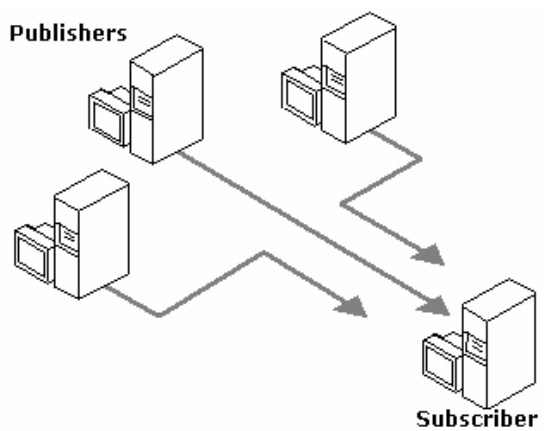
+ *Republisher*: Là mô hình Publisher xuất bản dữ liệu đến Subscriber, sau đó Subscriber được thiết lập là Publisher xuất bản dữ liệu đến Subscriber khác.



Đường truyền giữa hai máy được thiết lập là Publisher có thể tốc độ thấp, phù hợp với vị trí xa nhau. Ví dụ mô hình giữa các vùng cách xa nhau:



+ *Central Subscriber*: Là mô hình Subscriber thiết lậpn nhận dữ liệu xuất bản từ nhiều Publisher.



Những kiểu nhân bản dữ liệu.

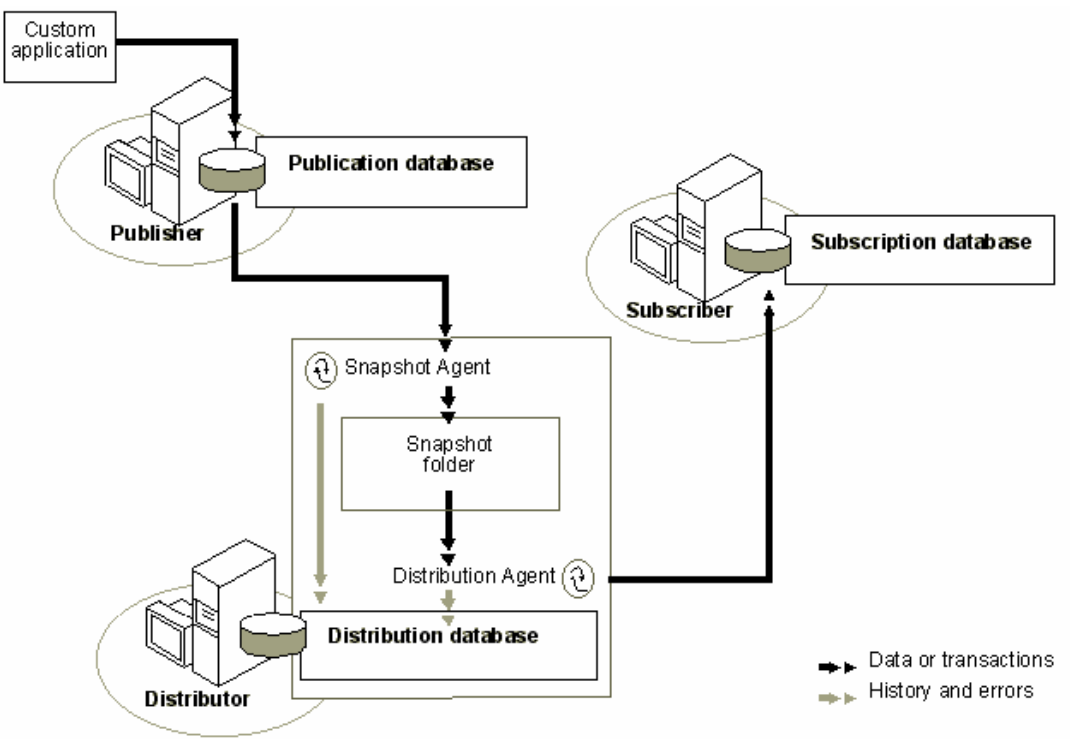
Có 3 kiểu nhân bản dữ liệu Snaphot, Transaction, Merge.

Snapshot replication: Là kiểu nhân bản thực hiện sao chép, phân tán dữ liệu hoặc các đối tượng của CSDL tạo một thời điểm.

Snapshot thường được sử dụng cho những tình huống sau:

+ Dữ liệu thường là tĩnh, ít thay đổi.

+ Nhân bản số lượng dữ liệu nhỏ.



Transaction replication: Là kiểu nhân bản mà bắt đầu bằng nhân bản snapshot, sau đó sẽ thực hiện nhân giao dịch dữ liệu theo các sự kiện insert, update, delete và những thay đổi liên quan đến thực hiện stored procedure, index view.

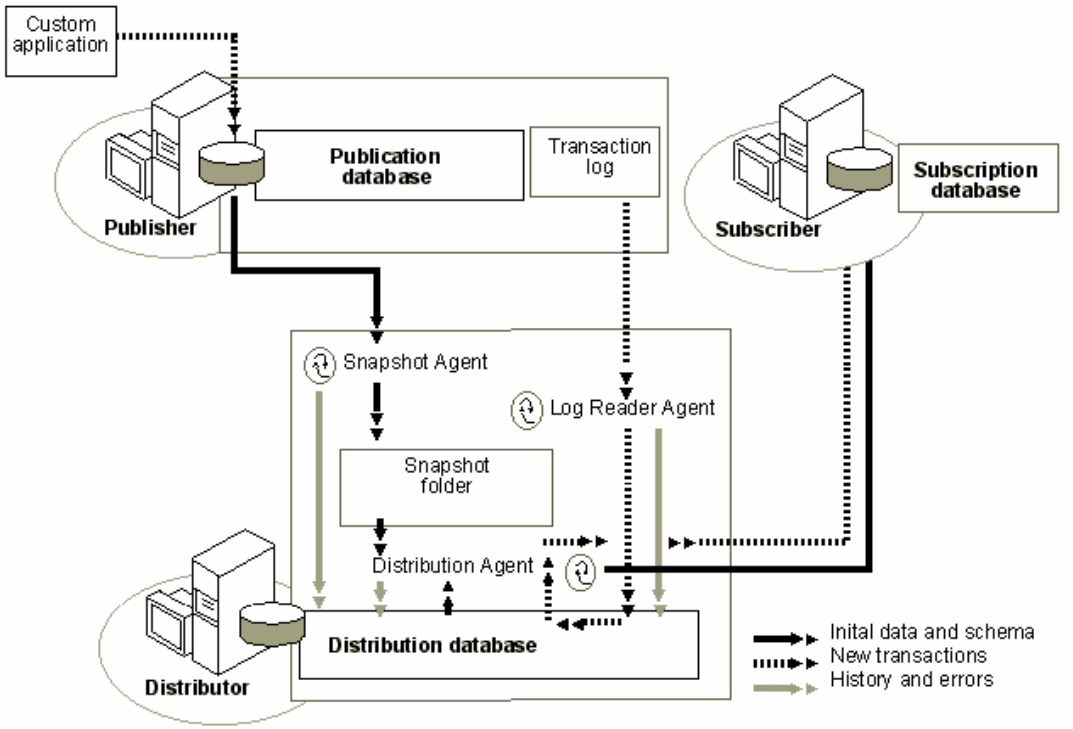
Nhân bản kiểu này cho phép thực hiện lọc dữ liệu tại xuất bản, cho phép user sửa đổi dữ liệu nhân bản tại subscriber và chuyển dữ liệu đã sửa đổi đến Publisher hoặc Subscriber khác, dữ liệu sửa đổi này có thể coi là dữ liệu được xuất bản.

Nhân bản kiểu này được thực hiện khi:

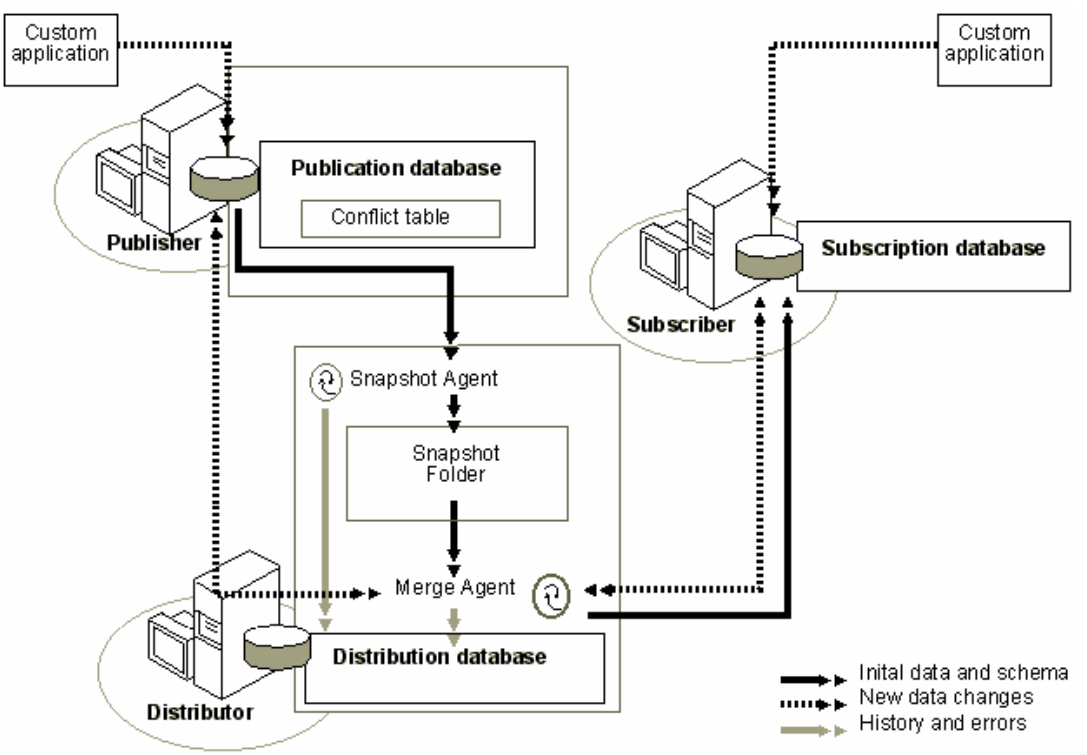
+ Muốn sửa đổi dữ liệu được xuất bản chuyển đến Subscriber, thời gian thực hiện theo giây, hoặc tức thời.

+ Cần giao dịch trên toàn bộ hệ thống nhân bản dữ liệu (dữ liệu có thể chuyển đến tất cả các Subscriber hoặc không chuyển đến Subscriber nào).

+ Subscriber thường xuyên kết nối với Publisher.



Merge replication: Là kiểu nhân bản dữ liệu cho phép thực hiện nhân sửa đổi dữ liệu trên nhiều Subscriber, có thể kết nối (online) hoặc không kết nối (offline) đến



Publisher. Dữ liệu sẽ được đồng bộ theo lịch hoặc theo yêu cầu, dữ liệu cập nhật có thời điểm sau sẽ được chấp nhận.

Kiểu nhân bản này thực hiện khi:

+ Nhiều Subscriber có nhu cầu cập nhật dữ liệu và chuyển dữ liệu cập nhật đến Publisher hoặc Subscriber khác.

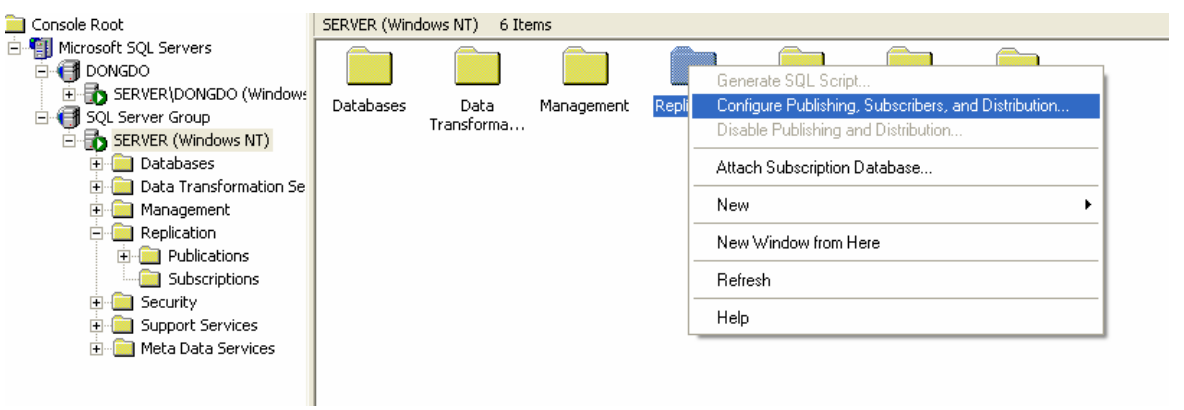
+ Subscriber yêu cầu nhận hoặc chuyển dữ liệu khi offline, đồng bộ dữ liệu với các Subscriber và Publisher sau.

CẤU HÌNH PUBLISHER VÀ DISTRIBUTOR.

Trước khi thực hiện cấu hình các máy thành Publisher hay Distributor ta phải thực hiện chạy dịch vụ SQL Server Agant trong chức năng Service manager. các bước cấu hình như sau:

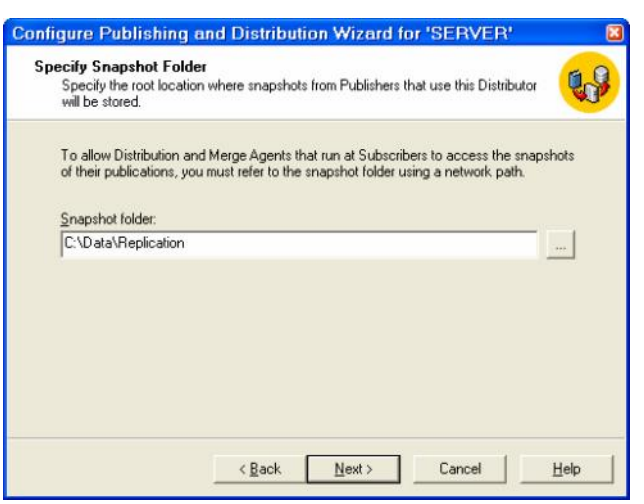
- Chọn Server cần cấu hình -> Replication

- Nhấn phải chuột -> Configure Publishing Subscription and Distribution...

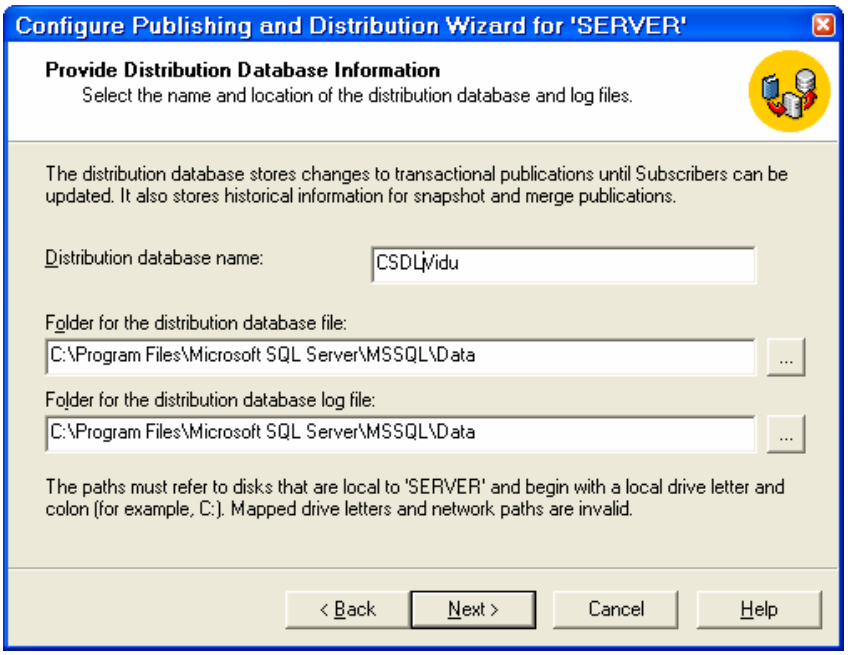


- Thực hiện thao các bước:

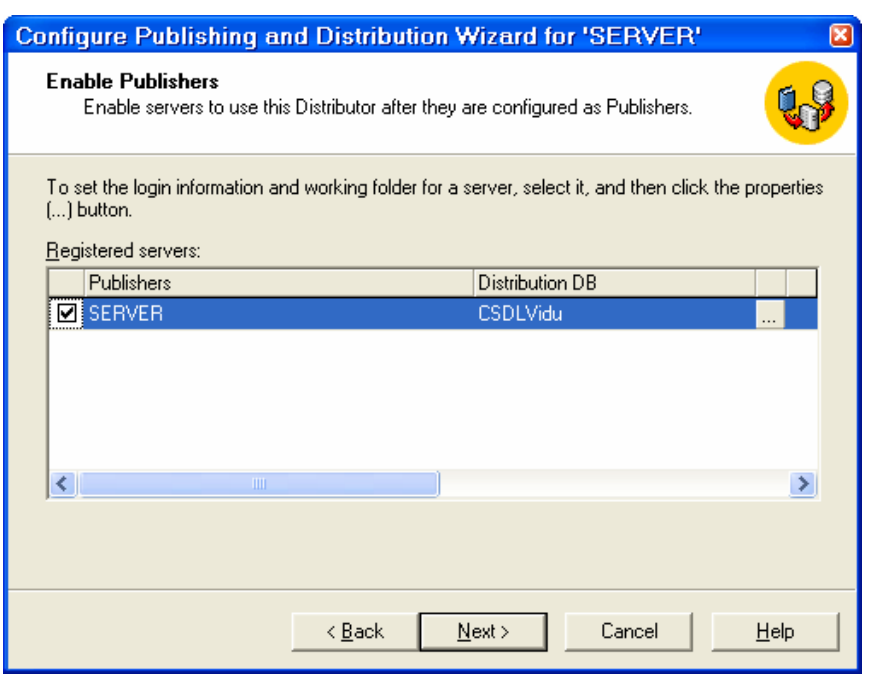
+ Chọn thư mục Snapshot: Thư mục này sẽ sử dụng cho



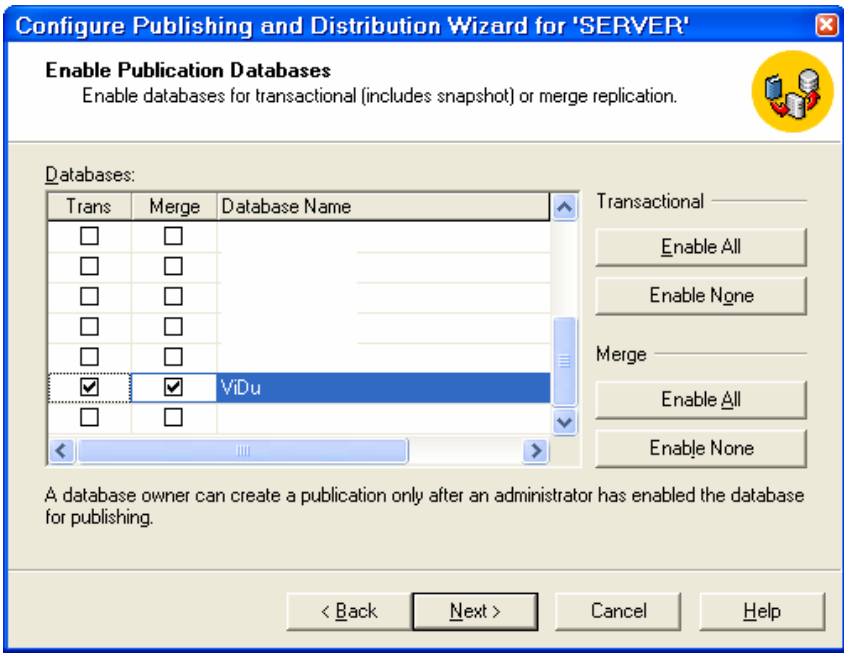
- Đặt tên CSDL của Distribution.



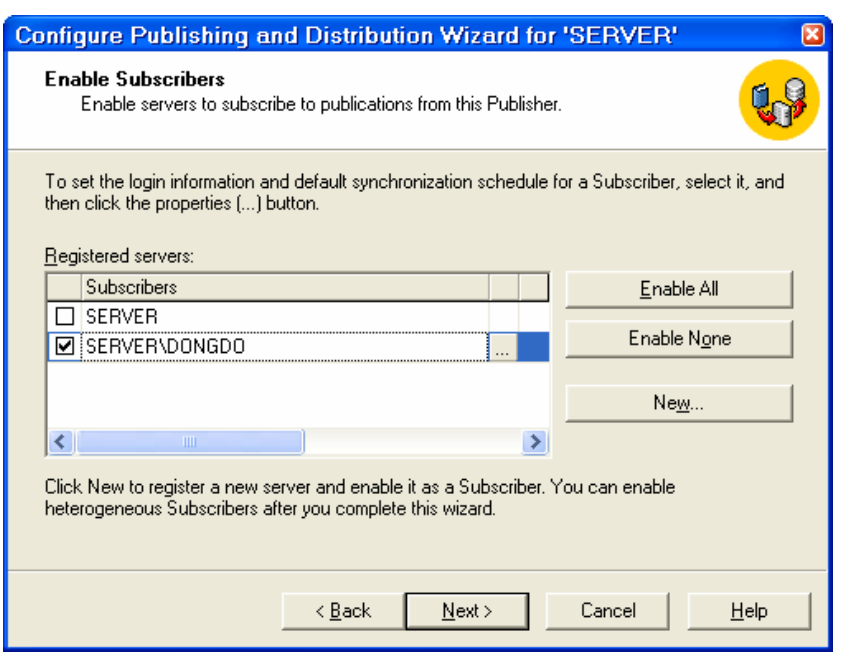
- Chọn Server cấu hình thành Publisher.



- Chọn CSDL tham gia nhân bản, kiểu nhân bản.



- Chọn Server được cấu hình là Subscriber của Publisher đang thiết lập.



- Kết thúc.

TẠO PUBLICATION.

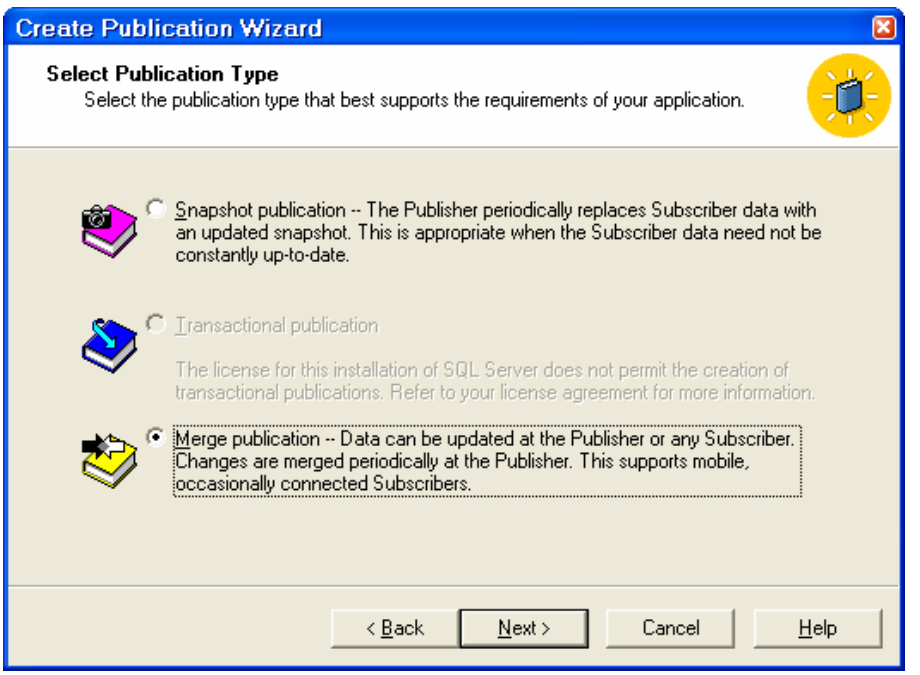
Bước này sẽ thực hiện tạo Publication, cách thực hiện như sau:

+ Chọn Publication trong Replication của Publisher.

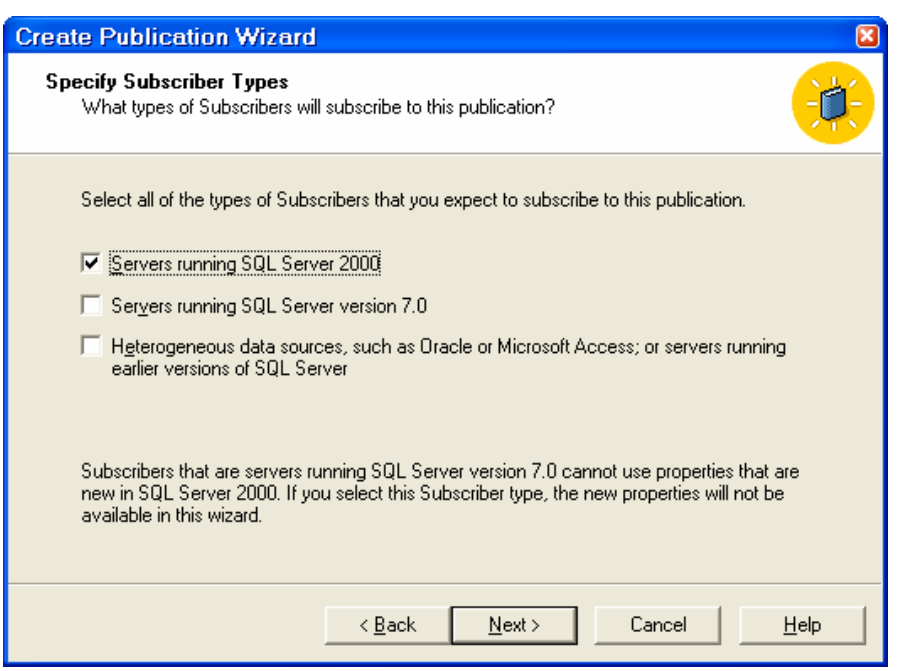
+ Nhấn phải chuột -> New Publication...

+ Thực hiện theo các bước:

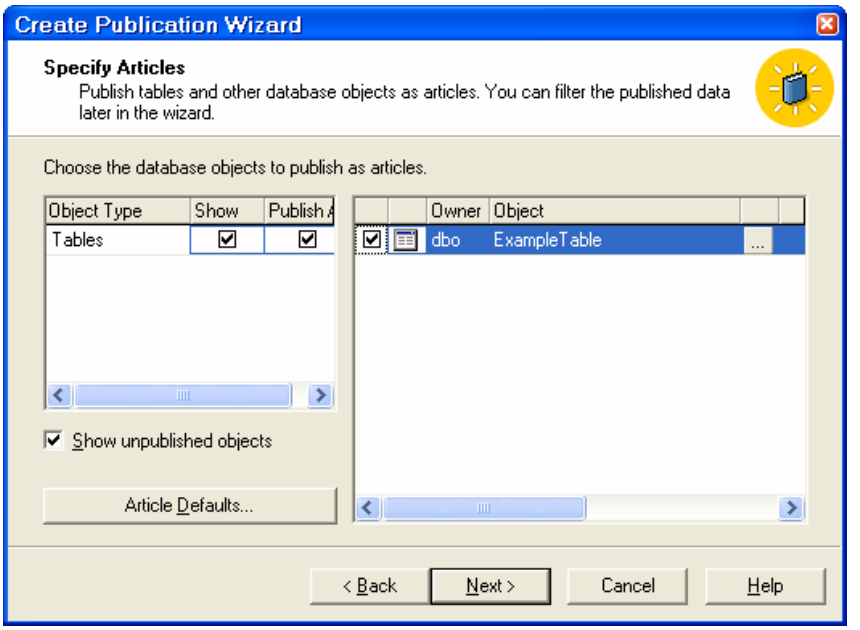
- Chọn CSDL cần xuất bản dữ liệu hoặc đối tượng.
- Chọn kiểu nhân bản (trong ví dụ này thực hiện kiểu Merge)



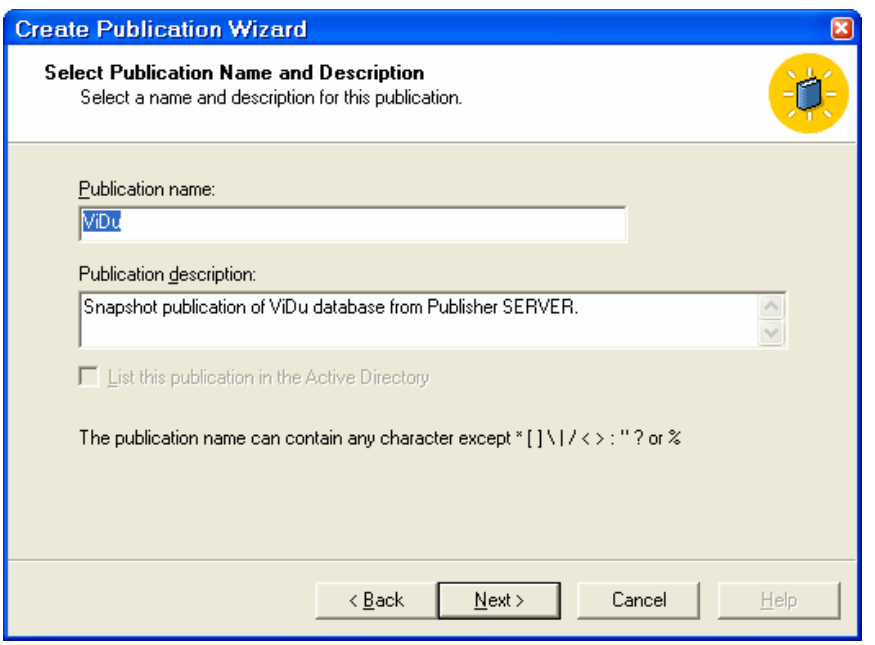
- Chọn phiên bản SQL Server của Subscriber.



- Chọn Article tham gia Publication.



- Đặt tên cho Publication.



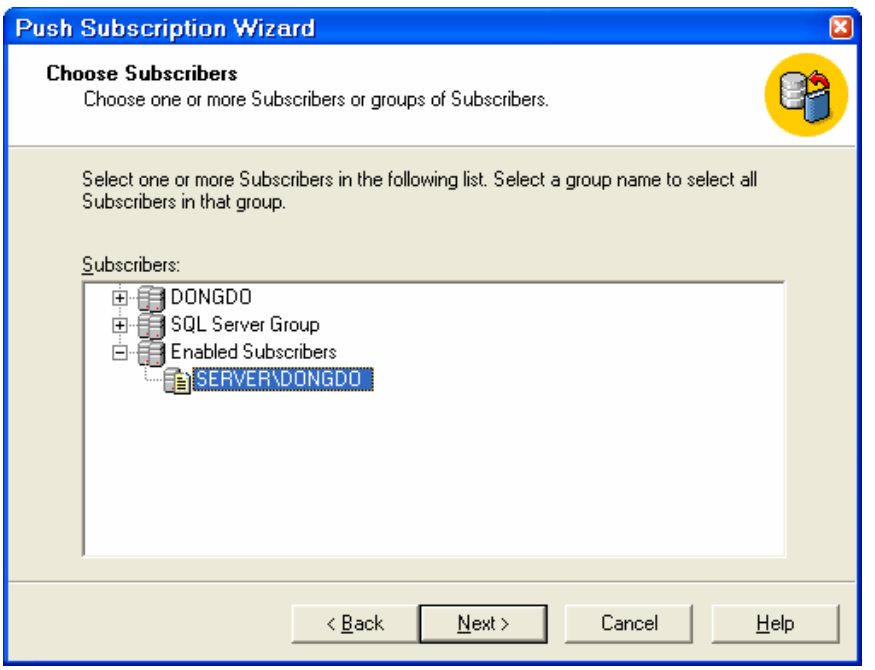
- Kết thúc.

TẠO PUSH SUBSCRIPTION.

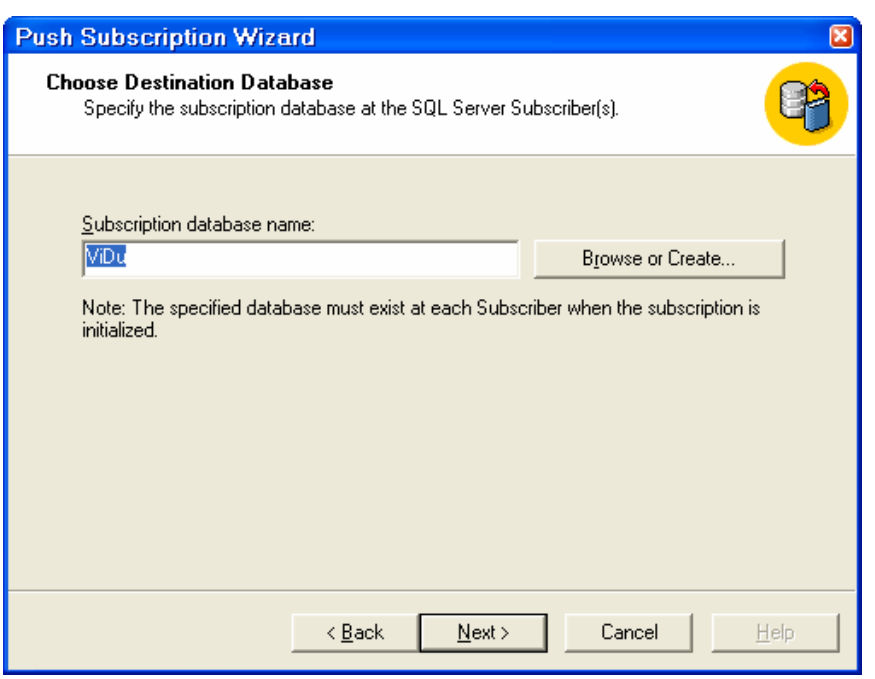
Bước này thực hiện tạo thủ tục đẩy (push) từ Publisher (Distributor trong ví dụ này) đến Subscriber, được thực hiện trên Publisher. Các bước thực hiện như sau:

- Chọn Publication của Publisher -> Nhấn phải chuột -> Push new Subscription...

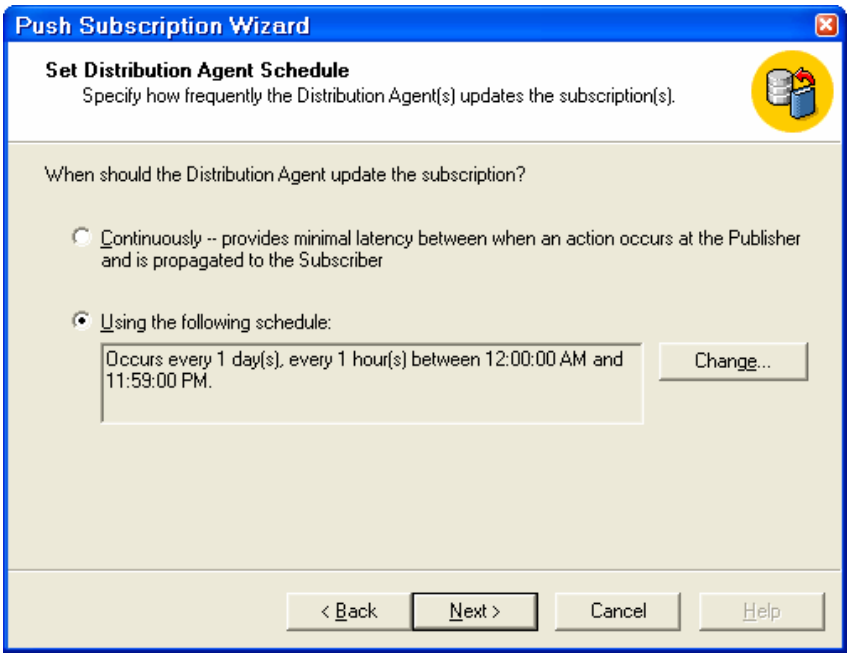
- Chọn Subscriber.



- Chọn CSDL trên Subscriber nếu đã có, nếu chưa có thực hiện chọn chức năng tạo mới.



- Chọn lịch thực hiện đồng bộ dữ liệu.



- Kết thúc. Sau khi thiết lập xong trên Subscriber sẽ có CSDL theo tên đã tạo.

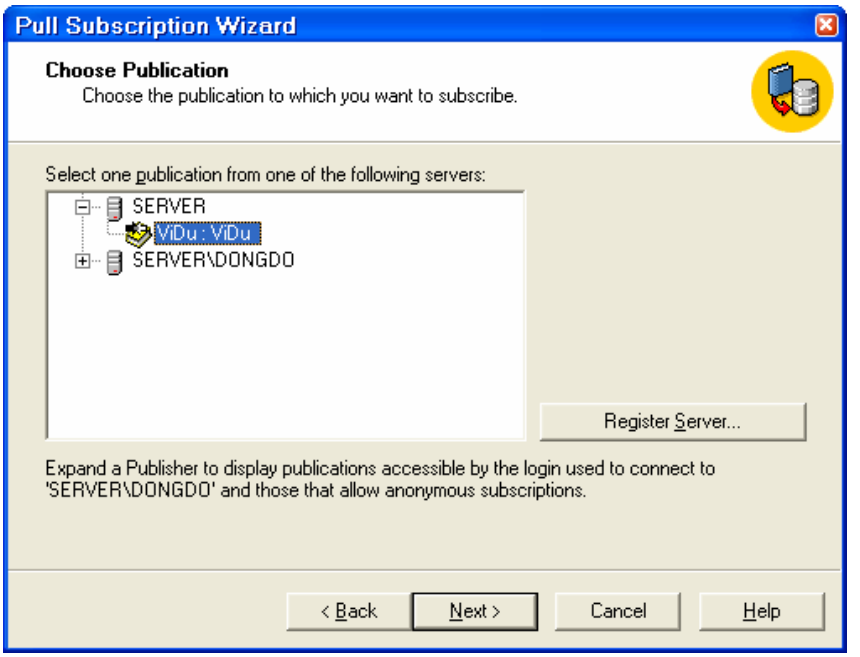
TẠO PULL SUBSCRIPTION.

Bước này thực hiện tạo công cụ kéo dữ liệu nhân bản từ Publisher về Subscriber, được thực hiện trên Subscriber.

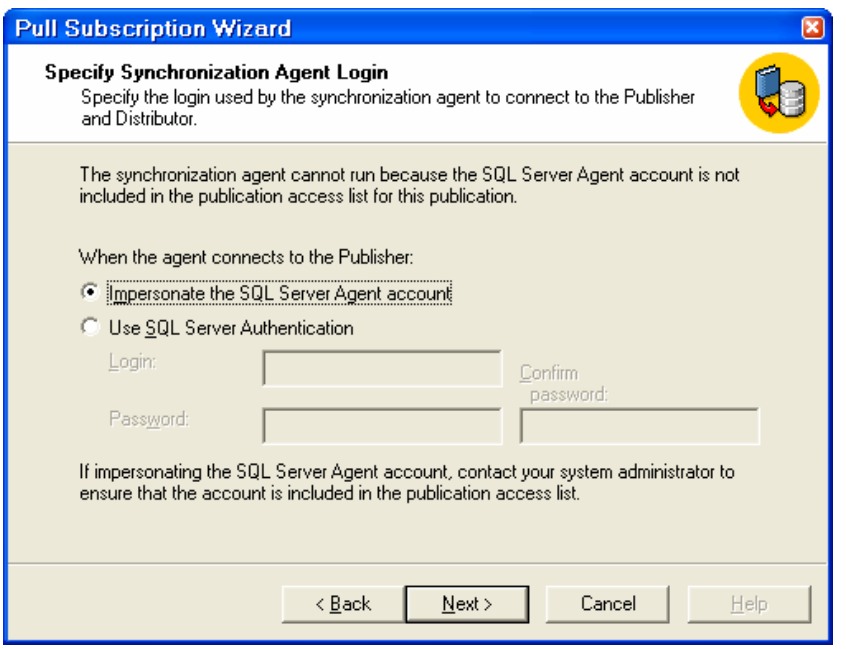
- Chọn Subscription của Subscriber -> Nhấn phải chuột -> New Pull Supcription...

- Thực hiện theo các bước:

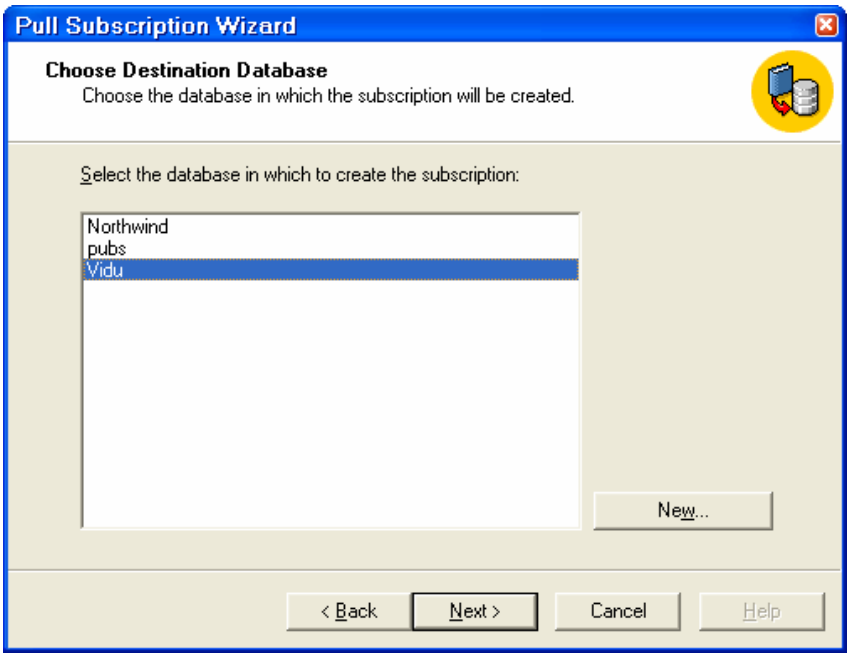
+ Chọn Publication.



- Chọn Agent tham gia kết nối Publisher.



- Chọn CSDL đích.



- Thực hiện tiếp các bước và kết thúc.

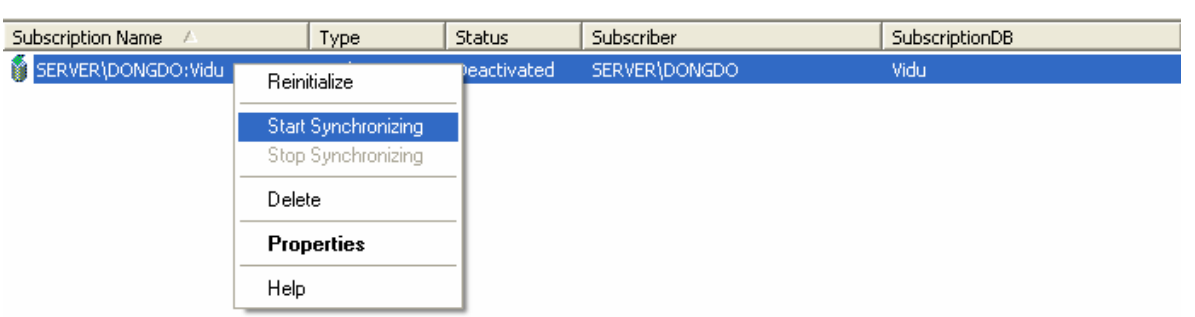
Nếu đã tạo Push Subscription với một CSDL sẽ không được tạo Pull Subscription với CSDL đó.

THỰC HIỆN ĐỒNG BỘ DỮ LIỆU.

Sau khi thiết lập theo các mô hình nhân bản xong, bạn có thể thực hiện đồng bộ dữ liệu bằng cách:

- Thực hiện theo lịch.

- Theo yêu cầu: Chọn Subscription (Push hoặc Pull) -> Nhấn phải chuột -> Start Synchronizing



Sau khi thực hiện xong dữ liệu sẽ được đồng bộ giữa Publisher và Subscriber. Ngoài thực hiện theo công cụ bạn có thể tìm hiểu thực hiện nhân bản theo câu lệnh T-SQL hoặc Stored Procedure.

Phần 2. CÂU LỆNH T-SQL

Trong phần này sẽ giới thiệu cấu trúc, kỹ thuật soạn kịch bản lệnh T-SQL, đối với các hệ quản trị CSDL Foxfro, Access thì câu lệnh thực hiện truy vấn, khai thác CSDL là ngôn ngữ truy vấn SQL (Structure Query Language), các lệnh được thực hiện theo từng câu lệnh mà không thực hiện theo kịch bản hoặc theo tập hợp nhiều câu lệnh với nhau. Đối với hệ quản trị CSDL Oracle thì ngôn ngữ truy vấn dữ liệu là SQL/PL (SQL Plus), còn SQL Server ngôn ngữ có tên Transact-SQL viết tắt là T-SQL.

ĐỊNH NGHĨA DỮ LIỆU (DATA DEFINITION LAGUAGE - DDL).

Phần này sẽ xem xét các lệnh liên quan đến tạo mới, sửa đổi, xóa các đối tượng liên quan đến Table, View và các đối tượng khác.

Tạo kiểu dữ liệu mới.

Tạo kiểu dữ liệu dạng user-defined.

Cú pháp:

sp_addtype [**@typename** =] *type***,**
[**@phystype** =] *system_data_type*
[**,** [**@nulltype** =] '*null_type*']
[**,** [**@owner** =] '*owner_name*']

Ví dụ:

sp_addtype ssn, ‘varchar(11)’, ‘NOT NULL’

Xóa kiểu dữ liệu đã tạo.

Cú pháp:

sp_droptype [**@typename** =] '*type*'

Ví dụ:

Sp_droptype ssn

Tạo ràng buộc (Constraint).

Tạo ràng buộc được thực hiện trong 2 câu lệnh Create Table hoặc Alter Table: Check, Default, Foreign Key, Primary Key, Unique.

Xét một số ví dụ sau:

+ Tạo một Check. trong bảng authors.

ALTER TABLE authors ADD CONSTRAINT chau_id CHECK(au_id LIKE ‘[0-9][0-9][0-9]-[0-9][0-9]- [0-9][0-9] [0-9][0-9]’)

+ Tạo Check trong bảng Publishers.

ALTER TABLE publishers ADD chpub_id CHECK(pub_id IN (‘1389’, ‘0736’, ‘0877’, ‘1622’, ‘1756’) OR pub_id LIKE ‘99[0-9][0-9]’)

+ Tạo ràng buộc Default.

ALTER TABLE authors ADD DEFAULT ‘UNKNOWN’ for au_lname

+ Tạo ràng buộc Foreign Key.

ALTER TABLE titles ADD CONSTRAINT FK_pub_id FOREIGN KEY(pub_id) REFERENCES publishers(pub_id)

+ Tạo ràng buộc Primary Key.

ALTER TABLE authors ADD CONSTRAINT UPKCL_auidind PRIMARY KEY CLUSTERED (au_id)

+ Tạo ràng buộc Unique.

ALTER TABLE stores ADD CONSTRAINT UNC_name_city UNIQUE NONCLUSTERED(store_name, city)

Xóa ràng buộc.

Sử dụng Drop trong các câu lệnh Create Table hoặc Alter Table.

+ Ví dụ xóa Constraint sử dụng câu lệnh Alter Table.

ALTER TABLE authors DROP CONSTRAINT UPKCL_auidind

Hiển thị ràng buộc.

sp_helpconstraint titltes

Tạo bảng.

Để tạo bảng dữ liệu có thể sử dụng 2 câu lệnh Create Table hoặc Select Into.

+ Tạo bảng tạm thời local (là bảng chỉ hiện với phiên hiện thời, tên bảng được bắt đầu bằng một dấu #).

```
CREATE TABLE #MyTempTable (cola INT PRIMARY KEY)
INSERT INTO #MyTempTable VALUES (1)
```

+ Tạo bảng tạm thời global (hiện với tất cả các phiên, tên bảng được bắt đầu bằng 2 dấu #).

```
CREATE TABLE ##MyTempTable (cola INT PRIMARY KEY)
```

```
    INSERT INTO ##MyTempTable VALUES (1)
```

+ Tạo bảng dữ liệu.

```
/* ************************ jobs table ********* */
CREATE TABLE jobs
(
   job_id  smallint
      IDENTITY(1,1)
      PRIMARY KEY CLUSTERED,
   job_desc        varchar(50)     NOT NULL
      DEFAULT 'New Position - title not formalized
yet',
   min_lvl tinyint NOT NULL
      CHECK (min_lvl >= 10),
   max_lvl tinyint NOT NULL
      CHECK (max_lvl <= 250)
)

/* ************************ employee table  */
CREATE TABLE employee
(
   emp_id  empid
      CONSTRAINT PK_emp_id PRIMARY KEY NONCLUSTERED
      CONSTRAINT CK_emp_id CHECK (emp_id LIKE
         '[A-Z][A-Z][A-Z][1-9][0-9][0-9][0-9][0-9][FM]'
or
         emp_id LIKE '[A-Z]-[A-Z][1-9][0-9][0-9][0-
9][0-9][FM]'), fname   varchar(20)     NOT NULL,
   minit   char(1) NULL,
   lname   varchar(30)     NOT NULL,
   job_id  smallint        NOT NULL
      DEFAULT 1
      REFERENCES jobs(job_id),
   job_lvl tinyint
      DEFAULT 10,
   pub_id  char(4) NOT NULL
      DEFAULT ('9952')
      REFERENCES publishers(pub_id),
   hire_date       datetime        NOT NULL
      DEFAULT (getdate())
)

/* ***************** publishers table *** */
```

```
CREATE TABLE publishers
(
   pub_id  char(4) NOT NULL
         CONSTRAINT UPKCL_pubind PRIMARY KEY CLUSTERED
         CHECK (pub_id IN ('1389', '0736', '0877',
'1622', '1756')
            OR pub_id LIKE '99[0-9][0-9]'),
   pub_name      varchar(40)     NULL,
   city         varchar(20)     NULL,
   state      char(2) NULL,
   country      varchar(30)     NULL
            DEFAULT('USA')
)
```

Xóa bảng.

Sử dụng lệnh Drop Table.

+ Xóa bảng trong CSDL hiện thời:

Drop Table MyTable

+ Xóa bảng trong CSDL khác.

```
DROP TABLE pubs.dbo.authors2
```

Đổi tên bảng.

Sử dụng thủ tục sp_rename

+ Đổi tên bảng:

Sp_rename titltes, books

Sửa cấu trúc bảng.

Sử dụng lệnh Alter Table.

+ Thêm một cột vào bảng.

```
CREATE TABLE doc_exa ( column_a INT)
GO
ALTER TABLE doc_exa ADD column_b VARCHAR(20) NULL
GO
EXEC sp_help doc_exa
GO
DROP TABLE doc_exa
GO
```

+ Xóa một cột khỏi bảng.

```
CREATE TABLE doc_exb ( column_a INT, column_b
VARCHAR(20) NULL)
```

```
GO
ALTER TABLE doc_exb DROP COLUMN column_b
GO
EXEC sp_help doc_exb
GO
DROP TABLE doc_exb
GO
```

Tạo Index.

Sử dụng lệnh Create Index.

+ Tạo Index.

```
SET NOCOUNT OFF
USE pubs
IF EXISTS (SELECT name FROM sysindexes
      WHERE name = 'au_id_ind')
   DROP INDEX authors.au_id_ind
GO
USE pubs
CREATE UNIQUE CLUSTERED INDEX au_id_ind
   ON authors (au_id)
GO
```

Xem thông tin Index.

Sử dụng thủ tục sp_helpindex

+ Xem Index của bảng authors.

sp_helpindex authors

Xóa Index.

Sử dụng lệnh Drop Index.

+ Xóa Index của bảng authors.

DROP INDEX authors.au_id_ind

Tạo khung nhìn.

Sử dụng lệnh Create View.

+ Tạo View.

```
USE pubs
IF EXISTS (SELECT TABLE_NAME FROM
INFORMATION_SCHEMA.VIEWS
      WHERE TABLE_NAME = 'titles_view')
```

```
   DROP VIEW titles_view
GO
CREATE VIEW titles_view
AS
SELECT title, type, price, pubdate
FROM titles
GO
```

Xóa khung nhìn.

Sử dụng lệnh Drop View.

+ Xóa khung nhìn.

```
USE pubs
IF EXISTS (SELECT TABLE_NAME FROM
INFORMATION_SCHEMA.VIEWS
         WHERE TABLE_NAME = 'titles_view')
   DROP VIEW titles_view
GO
```

Đổi tên khung nhìn.

Sử dụng lệnh thủ tục sp_rename.

+ Đổi tên view.

sp_rename titles_view, view_titles

THAO TÁC VỚI DỮ LIỆU (DATA MANIPULATION LANGUAGE - DML).

Phần này sẽ xem xét các câu lệnh thao tác với dữ liệu như Insert, Select, Delete.

Lệnh Insert - Chèn dữ liệu vào bảng.

Sử dụng câu lệnh Insert.

+ Chèn dữ liệu vào tất cả các cột, theo thứ tự của trong bảng.

```
IF EXISTS(SELECT TABLE_NAME FROM
     INFORMATION_SCHEMA.TABLES
           WHERE TABLE_NAME = 'T1')
        DROP TABLE T1
     GO
CREATE TABLE T1 ( column_1 int, column_2 varchar(30))
INSERT T1 VALUES (1, 'Row #1')
```

+ Chèn dữ liệu vào các cột không theo thứ tự.

```
IF EXISTS(SELECT TABLE_NAME FROM
INFORMATION_SCHEMA.TABLES
      WHERE TABLE_NAME = 'T1')
   DROP TABLE T1
GO
CREATE TABLE T1 ( column_1 int, column_2 varchar(30))
INSERT T1 (column_2, column_1) VALUES ('Row #1',1)
```

+ Chèn dữ liệu số giá trị ít hơn số cột.

```
IF EXISTS(SELECT TABLE_NAME FROM
INFORMATION_SCHEMA.TABLES
      WHERE TABLE_NAME = 'T1')
   DROP TABLE T1
GO
CREATE TABLE T1
( column_1 int identity,
  column_2 varchar(30)
    CONSTRAINT default_name DEFAULT ('column default'),
  column_3 int NULL,
  column_4 varchar(40)
)
INSERT INTO T1 (column_4)
   VALUES ('Explicit value')
INSERT INTO T1 (column_2,column_4)
   VALUES ('Explicit value', 'Explicit value')
INSERT INTO T1 (column_2,column_3,column_4)
   VALUES ('Explicit value',-44,'Explicit value')
SELECT *
FROM T1
```

+ Chèn dữ liệu với bảng có cột dữ liệu IDENTITY.

Ví dụ sau sẽ thực hiện chèn dữ liệu vào bảng có cột kiểu IDENITY, cột có kiểu IDENTITY sẽ tự động gán giá trị khi hàng mới được tạo, nên người nhập sẽ không nhập và sửa đổi. Tuy nhiên vó thể sử dụng câu lệnh SET IDENTITY_INSERT để nhập giá trị.

```
IF EXISTS(SELECT TABLE_NAME FROM
INFORMATION_SCHEMA.TABLES
      WHERE TABLE_NAME = 'T1')
   DROP TABLE T1
GO
CREATE TABLE T1 ( column_1 int IDENTITY, column_2
varchar(30))
INSERT T1 VALUES ('Row #1')
INSERT T1 (column_2) VALUES ('Row #2')
SET IDENTITY_INSERT T1 ON
INSERT INTO T1 (column_1,column_2)
   VALUES (-99,'Explicit identity value')
SELECT *
FROM T1
```

Lệnh Insert - Chèn dữ liệu vào bảng kết hợp lệnh SELECT.

Câu lệnh này được thực hiện gần tương tự như câu lệnh chèn dữ liệu sử dụng từ khóa VALUES, nhưng giá trị chèn vào được truy vấn từ câu lệnh SELECT. Đối với các cột dữ liệu có kiểu Nchar, Nvarchar hỗ trợ Unicode thì khi chèn dữ liệu trực tiếp phải gán thêm tiền tố N, ví dụ Lname=N'John Smith'.

+ Chèn dữ liệu được truy vấn từ các cột trong lệnh SELECT.

```
USE pubs
INSERT INTO MyBooks
   SELECT *
   FROM titles
   WHERE type = 'mod_cook'
```

+ Chèn dữ liệu được truy vấn từ một số cột.

```
USE pubs
INSERT INTO MyBooks
   SELECT title_id, title, type
   FROM titles
   WHERE type = 'mod_cook'
```

Lệnh Update – Sửa dữ liệu.

Lệnh Update sử dụng sửa dữ liệu trong bảng hoặc View, xem xét cu pháp thông qua các ví dụ sau.

+ Sửa dữ liệu sử dụng lệnh Update sử dụng mệnh đề SET.

```
UPDATE Northwind.dbo.Products
SET UnitPrice = UnitPrice * 1.1
WHERE CategoryID = 2
```

Hoặc gán giá trị trực tiếp:

```
UPDATE authors
    SET authors.au_fname = 'Annie'
    WHERE au_fname = 'Anne'
```

Hoặc gán giá trị NULL cho một cột.

```
UPDATE publishers
SET pub_name = NULL
```

+ Sửa dữ liệu sử dụng mệnh đề Where xác định hàng được sửa dữ liệu.

```
UPDATE authors
   SET state = 'PC', city = 'Bay City'
      WHERE state = 'CA' AND city = 'Oakland'
```

+ Sửa dữ liệu sử dụng mệnh đề From, sử dụng thông tin từ một bảng khác.

```
UPDATE titles
    SET ytd_sales = t.ytd_sales + s.qty
    FROM titles t, sales s
    WHERE t.title_id = s.title_id
    AND s.ord_date = (SELECT MAX(sales.ord_date) FROM
sales)
```

Hoặc ví dụ giá trị xác định là tổng từ bảng khác.

```
UPDATE titles
   SET ytd_sales =
      (SELECT SUM(qty)
```

```
         FROM sales
            WHERE sales.title_id = titles.title_id
            AND sales.ord_date IN (SELECT MAX(ord_date)
FROM sales))
   FROM titles, sales
```

+ Sửa dữ liệu sử dụng mệnh đề Top, xác định số lượng hàng đầu tiên được sửa dữ liệu.

```
UPDATE authors
SET state = 'ZZ'
FROM (SELECT TOP 10 * FROM authors ORDER BY au_lname)
AS t1
WHERE authors.au_id = t1.au_id
```

Lệnh WriteText – Sửa dữ liệu Text, Image.

Lệnh WriteText được sử dụng cập nhật cột có kiểu Text hoặc Image. Dữ liệu kiểu Text và Image thường có kích thước lớn, có thể đến Gyga byte, nên làm việc với kiểu dữ liệu này phải sử dụng con trỏ. Để sử dụng được lệnh này trước hết người quản trị (Administrator) phải đặt thuộc tính select into/bulk copy là true, thực hiện đặt như sau:

```
USE master
EXEC sp_dboption 'pubs', 'select into/bulkcopy', 'TRUE'
```

Với cột dữ liệu kiểu Text, Image ta có thể gán giá trị NULL hoặc sử dụng các lệnh WriteText, UpdateText để gán giá trị, khi sử dụng các lệnh trên, hàng dữ liệu có cột cần chèn đã tồn tại (không đồng thời với câu lệnh Insert). Riêng đối với cột dữ liệu kiểu Text bạn có thể sử dụng lệnh thêm dữ liệu như các cột kiểu chuỗi khác nhưng kích thước của dữ liệu tối đa chỉ được 4096 ký tự.

+ Thực hiện chèn đoạn văn bản vào cột dữ liệu kiểu Text.

```
DECLARE @ptrval binary(16)
SELECT @ptrval = TEXTPTR(pr_info)
FROM pub_info pr, publishers p
WHERE p.pub_id = pr.pub_id
   AND p.pub_name = 'New Moon Books'
```

```
WRITETEXT pub_info.pr_info @ptrval 'New Moon Books
(NMB) has just released another top ten publication.
With the latest publication this makes NMB the hottest
new publisher of the year!'
GO
```

Xem ví dụ trên ta thấy, để chèn dữ liệu vào cột Text hoặc Image ta phải sử dụng con trỏ kiểu binary hoặc varbinarry, con trỏ sẽ được xác định vào cột text, image và hàng tương ứng đã có trong bảng dữ liệu, sau đó sử dụng lệnh WriteText để gán giá trị. Trong thực tế khi thực hiện lệnh này ta thường thực hiện thông qua thủ tục lưu trữ của CSDL, giá trị được gán qua biến. Lệnh WriteText thường được sử dụng khi cột dữ liệu đó là NULL hoặc đè toàn bộ dữ liệu đã có (không chèn thêm).

Lệnh UpdateText – Sửa dữ liệu Text, Image.

Lệnh UpdateText có chức năng thực hiện sửa dữ liệu kiểu Text, Image, tuy nhiên UpdateText khác WriteTex, UpdateText có thể sửa., xóa dữ liệu theo từng đoạn hoặc thêm dữ liệu vào phần dữ liệu đã có của cột dữ liệu.

+ Cú pháp chung:

UPDATETEXT { *table_name.dest_column_name dest_text_ptr* }
{ NULL | *insert_offset* }
{ NULL | *delete_length* }
[WITH LOG]
[*inserted_data*
| { *table_name.src_column_name src_text_ptr* }]

Trong đó:

- Insert_offset: Xác định vị trí theo byte dữ liệu sẽ được đặt vào hoặc bắt đầu xóa.

- Delete_length: Xác định độ dài dữ liệu được xóa tính từ vị trí insert_offset.

Việc chèn, xóa, sửa dữ liệu được điều khiển thông qua các tham số insert_offset, delete_offset, ví dụ muốn sửa dữ liệu, đầu tiên phải xác định vị trí bắt đầu cần sửa dữ liệu (insert_offset) và độ dài dữ liệu cần sửa, bắt đầu từ vị trí cần xóa dữ liệu mới sẽ được chèn vào.

+ Ví dụ sửa nội dung cột kiểu Text.

```
USE pubs
GO
EXEC sp_dboption 'pubs', 'select into/bulkcopy', 'true'
GO
DECLARE @ptrval binary(16)
SELECT @ptrval = TEXTPTR(pr_info)
   FROM pub_info pr, publishers p
      WHERE p.pub_id = pr.pub_id
      AND p.pub_name = 'New Moon Books'
UPDATETEXT pub_info.pr_info @ptrval 88 1 'b'
GO
EXEC sp_dboption 'pubs', 'select into/bulkcopy',
'false'
GO
```

Cursor - Điều khiển con trỏ.

Cursor là kiểu biến xác định con trỏ cho một tập dữ liệu, là kết quả của câu lệnh Select. Cursor được kết hợp cùng lệnh Fetch để xác định vị trí hàng trong tập dữ liệu. Cursor có 2 kiểu Cursor thông thường và Scroll Cursor.

Các thao tác thực hiện với Cursor:

+ Declare: Khai báo.

+ Open: Mở con trỏ để làm việc với tập dữ liệu.

+ Fetch: Dịch chuyển vị trí hàng trong tập dữ liệu.

+ Close: Đóng con trỏ.

+ DeAllocate: Giải phóng con trỏ.

+ Ví dụ sử dụng Curcor, liệt kê danh sách các hàng của bảng Authors.

```
USE pubs
GO
DECLARE authors_cursor CURSOR FOR
SELECT au_lname FROM authors
WHERE au_lname LIKE "B%"
ORDER BY au_lname

OPEN authors_cursor

-- Perform the first fetch.
FETCH NEXT FROM authors_cursor
```

```
-- Check @@FETCH_STATUS to see if there are any more
rows to fetch.
WHILE @@FETCH_STATUS = 0
BEGIN
   -- This is executed as long as the previous fetch
succeeds.
   FETCH NEXT FROM authors_cursor
END

CLOSE authors_cursor
DEALLOCATE authors_cursor
GO
```

+ Ví dụ sử dụng Cursor, giá trị cột được đưa vào biến.

```
USE pubs
GO

-- Declare the variables to store the values returned
by FETCH.
DECLARE @au_lname varchar(40), @au_fname varchar(20)


DECLARE authors_cursor CURSOR FOR
SELECT au_lname, au_fname FROM authors
WHERE au_lname LIKE "B%"
ORDER BY au_lname, au_fname

OPEN authors_cursor

-- Perform the first fetch and store the values in
variables.
-- Note: The variables are in the same order as the
columns
-- in the SELECT statement.

FETCH NEXT FROM authors_cursor
INTO @au_lname, @au_fname

-- Check @@FETCH_STATUS to see if there are any more
rows to fetch.
```

```
WHILE @@FETCH_STATUS = 0
BEGIN

   -- Concatenate and display the current values in the
variables.
   PRINT "Author: " + @au_fname + " " +  @au_lname

-- This is executed as long as the previous fetch
succeeds.
   FETCH NEXT FROM authors_cursor
   INTO @au_lname, @au_fname
END

CLOSE authors_cursor
DEALLOCATE authors_cursor
GO
```

+ Ví dụ sử dụng Scroll Cursor, con trỏ cho phép sử dụng các phương thức: LAST, PRIOR, RELATIVE, ABSOLUTE.

```
USE pubs
GO

-- Execute the SELECT statement alone to show the
-- full result set that is used by the cursor.
SELECT au_lname, au_fname FROM authors
ORDER BY au_lname, au_fname

-- Declare the cursor.
DECLARE authors_cursor SCROLL CURSOR FOR
SELECT au_lname, au_fname FROM authors
ORDER BY au_lname, au_fname

OPEN authors_cursor

-- Fetch the last row in the cursor.
FETCH LAST FROM authors_cursor

-- Fetch the row immediately prior to the current row
in the cursor.
FETCH PRIOR FROM authors_cursor
```

```
-- Fetch the second row in the cursor.
FETCH ABSOLUTE 2 FROM authors_cursor

-- Fetch the row that is three rows after the current
row.
FETCH RELATIVE 3 FROM authors_cursor

-- Fetch the row that is two rows prior to the current
row.
FETCH RELATIVE -2 FROM authors_cursor

CLOSE authors_cursor
DEALLOCATE authors_cursor
GO
```

Lệnh Delete – Xóa dữ liệu.

Sử dụng lệnh Delete để xóa dữ liệu, kết hợp cùng điều kiện để xóa một hay nhiều hàng dữ liệu trong bảng.

+ Xóa tất cả các hàng của bảng.

```
USE pubs
DELETE authors
```

+ Xóa một tập các hàng.

```
USE pubs
DELETE FROM authors
WHERE au_lname = 'McBadden'
```

+ Xóa một hàng tạ vị trí con trỏ.

```
USE pubs
DELETE FROM authors
WHERE CURRENT OF complex_join_cursor
```

Trong ví dụ trên con trỏ đã được mở có tên complex_join_curcor.

+ Xóa các hàng dựa vào lệnh truy vấn khác hoặc liên kết các bảng.

```
/* SQL-92-Standard subquery */
USE pubs
DELETE FROM titleauthor
WHERE title_id IN
   (SELECT title_id
   FROM titles
   WHERE title LIKE '%computers%')

/* Transact-SQL extension */
USE pubs
DELETE titleauthor
FROM titleauthor INNER JOIN titles
   ON titleauthor.title_id = titles.title_id
WHERE titles.title LIKE '%computers%'
```

+ Xóa dữ liệu sử dụng từ khóa Top.

```
DELETE authors
FROM (SELECT TOP 10 * FROM authors) AS t1
WHERE authors.au_id = t1.au_id
```

Lệnh Truncate Table – Xóa dữ liệu toàn bảng.

Tương tự như câu lệnh Delete, lệnh Truncate Table sử dụng xóa dữ liệu toàn bảng, thao tác này giống lệnh Delete khi không có điều kiện Where nhưng lệnh Truncate Table thực hiện nhanh hơn.

```
TRUNCATE TABLE authors
```

Lệnh Go – Nhóm lệnh.

Lệnh Go không tham gia thao tác với CSDL, lệnh Go xác định nhóm các lệnh với nhau, nhóm lệnh được xác định từ vị trí đầu tiên hoặc từ từ lệnh Go trước đó đến lệnh Go tiếp theo. Khi gặp lệnh Go nhóm lệnh sẽ được gửi ngay đến SQL Server để thực hiện.

```
USE pubs
GO
DECLARE @MyMsg VARCHAR(50)
SELECT @MyMsg = 'Hello, World.'
GO -- @MyMsg is not valid after this GO ends the batch.

-- Yields an error because @MyMsg not declared in this
batch.
PRINT @MyMsg
GO

SELECT @@VERSION;
-- Yields an error: Must be EXEC sp_who if not first
statement in
-- batch.
sp_who
GO
```

Contrrol-of-Flow - Điều khiển luồng.

Tương tự như các ngôn ngữ lập trình thiết kế ứng dụng, T-SQL cho phép thiết lập kịch bản câu lệnh, cho phép sử dụng các lệnh điều khiển khối, luồng, vòng lặp, điều kiện, rẽ nhánh,… Sau đây là bảng các lệnh:

Từ khóa	Mô tả
BEGIN…END	Khối lệnh
GOTO	Lệnh nhảy
IF…ELSE	Lệnh điều kiện
RETURN	Thoát
WAITFOR	Chờ thực hiện lệnh
WHILE..BREAK..CONTINUE	Vòng lặp, thoát khỏi vòng lặp, quay lại lặp
CASE	Rẽ nhánh
DECLARE	Khai báo
PRINT	In thông báo
RAISEERROR	Trả lại mã lỗi
EXECUTE (EXEC)	Thực hiện lệnh

TRUY VẤN DỮ LỆU.

Trong trước ta đã xem xét những câu lệnh thao tác với dữ liệu như Insert, Update, Delete, phần này ta sẽ xem xét các câu lệnh khai thác truy vấn dữ liệu như Select, các phép Join,…

Lệnh Use - Chọn Cơ sở dữ liệu.

Sử dung lệnh Use để chọn CSDL trong kịch bản câu lệnh.

```
Use Pubs
```

Select - Truy vấn tất cả các cột từ một bảng.

Lệnh Select được sử dụng truy vấn dữ liệu từ một hoặc nhiều bảng, từ khung nhìn, kết quả đưa lại một tập dữ liệu gồm các hàng, cột.

```
USE Northwind
GO
SELECT *
FROM Shippers
GO
```

Order by - Truy vấn sắp xếp danh sách theo thứ tự.

ASC là sắp xếp tăng, DESC là sắp xếp giảm, khi xác định sắp xếp tăng bạn có thể không cần đặt từ khóa ASC mà hệ thống tự xác định là ASC.

```
USE Northwind
GO
SELECT *
FROM Shippers
ORDER BY CompanyName DESC
GO
```

Truy vấn một số cột, xác định thứ tự các cột.

```
USE Northwind
GO
SELECT OrderID, ProductID, UnitPrice, Quantity,
Discount
FROM [Order Details]
ORDER BY OrderID ASC
GO
```

Đổi tên các cột khi truy vấn.

```
USE Northwind
GO
SELECT OrderID as [Order ID], ProductID as [Product
ID], UnitPrice as [Unit Price], Quantity, Discount
FROM [Order Details]
ORDER BY OrderID ASC
GO
```

Lệnh Case - Phân lớp dữ liệu.

Case là câu lệnh rẽ nhánh, thường được sử dụng phân lớp dữ liệu trong câu lệnh Select.

Ví dụ sử dụng lệnh Case đơn giản:

```
USE pubs
GO
SELECT   Category =
      CASE type
         WHEN 'popular_comp' THEN 'Popular Computing'
         WHEN 'mod_cook' THEN 'Modern Cooking'
         WHEN 'business' THEN 'Business'
         WHEN 'psychology' THEN 'Psychology'
         WHEN 'trad_cook' THEN 'Traditional Cooking'
         ELSE 'Not yet categorized'
      END,
   CAST(title AS varchar(25)) AS 'Shortened Title',
   price AS Price
FROM titles
WHERE price IS NOT NULL
```

```
ORDER BY type, price
COMPUTE AVG(price) BY type
GO
```

Ví dụ sử dụng lệnh Case tìm kiếm:

```
USE pubs
GO
SELECT    'Price Category' =
      CASE
         WHEN price IS NULL THEN 'Not yet priced'
         WHEN price < 10 THEN 'Very Reasonable Title'
         WHEN price >= 10 and price < 20 THEN 'Coffee
Table Title'
         ELSE 'Expensive book!'
      END,
   CAST(title AS varchar(20)) AS 'Shortened Title'
FROM titles
ORDER BY price
GO
```

Kết quả thực hiện như sau:

```
Price Category        Shortened Title
--------------------- --------------------
Not yet priced        Net Etiquette
Not yet priced        The Psychology of Co
Very Reasonable Title The Gourmet Microwav
Very Reasonable Title You Can Combat Compu
Very Reasonable Title Life Without Fear
Very Reasonable Title Emotional Security:
Coffee Table Title    Is Anger the Enemy?
Coffee Table Title    Cooking with Compute
Coffee Table Title    Fifty Years in Bucki
Coffee Table Title    Sushi, Anyone?
Coffee Table Title    Prolonged Data Depri
Coffee Table Title    Silicon Valley Gastr
Coffee Table Title    Straight Talk About
Coffee Table Title    The Busy Executive's
```

```
Expensive book!       Secrets of Silicon V
Expensive book!       Onions, Leeks, and G
Expensive book!       Computer Phobic And
Expensive book!       But Is It User Frien

(18 row(s) affected)
```

Đặt tên cho cột.

Sử dụng dấu phẩy xác định tên cột, tối đa là 30 ký tự.

```
SELECT 'sum'= SUM(ytd_sales) FROM titles
```

Khi cần thể hiện dấu phẩy trên giá trị hoặc tên cột ta cần sử dụng 2 dấu liền nhau. Ví dụ `'I don''t understand.'`

Chuỗi ký tự trong kết quả truy vấn.

Sử dụng dấu phẩy trong chuỗi ký tự.

```
SELECT 'The publisher''s name is', publisher=pub_name
FROM publishers
```

Các giá trị tính toán được.

Đối các kiểu dữ liệu tính toán được sử dụng các phép toán +, -, *, /, %.

```
SELECT title_id, ytd_sales*2 FROM titles
```

Truy vấn kiểu dữ liệu Text, Image.

Để truy vấn dữ liệu Text, Image có thể sử dụng 2 lệnh Select hoặc ReadText. Khi sử dụng lệnh Select để truy vấn kiểu dữ liệu này thì chỉ truy vấn được dữ liệu có độ dài xác định trước bằng câu lệnh SET TEXTSIZE.

```
SET TEXTSIZE 25
```

```
SELECT pub_id, pr_info FROM pub_info
```

Ngầm định kích thước sử dụng cho truy vấn là 4096 (4K).

Từ khóa Distinct – Truy vấn các hàng khác nhau theo cột.

Để truy vấn các hàng dữ liệu khác nhau theo cột ta sử dụng từ khóa Distinct.

```
USE pubs
SELECT DISTINCT au_id
FROM titleauthor
```

Xác định bảng trong mệnh đề From.

```
USE pubs
SELECT p.pub_id, p.pub_name
FROM publishers p
```

Mệnh đề Where.

Mệnh đề Where xác định điều kiện các hàng được truy vấn, biểu thức trong mệnh đề Where xác định theo biểu thức logic. Các phép toán, câu lệnh xác định gồm:

- Các phép toán so sánh: =, <>, <, >, !<, !>.
- Từ khóa xác định phạm vi: Between, Not Between.
- Danh sách: In, Not In.
- Theo mẫu đinh dạng: Like, Not Like.
- Giá trị NULL: Is Null, Is Not Null.
- Các phép toán logic: And, Or.

+ Từ khóa Between:

```
SELECT UnitsInStock, ProductID, ProductName
```

```
FROM Northwind.dbo.Products
WHERE UnitsInStock BETWEEN 15 AND 25

ORDER BY UnitsInStock
```

+ Từ khóa Not Between.

```
SELECT UnitsInStock, ProductID, ProductName
FROM Northwind.dbo.Products
WHERE UnitsInStock NOT BETWEEN 15 AND 25
ORDER BY UnitsInStock
```

+ Từ khóa In, Not In.

```
USE pubs

SELECT au_lname, state
FROM authors
WHERE state IN ('CA', 'IN', 'MD')
```

```
USE pubs
SELECT au_lname, au_fname
FROM authors
WHERE au_id IN
   (SELECT au_id
   FROM titleauthor
   WHERE royaltyper < 50)
```

```
USE pubs
SELECT au_lname, au_fname
FROM authors
WHERE au_id NOT IN
   (SELECT au_id
   FROM titleauthor
   WHERE royaltyper < 50)
```

+ Từ khóa Like.

Từ khóa Like được sử dụng tương tự như phép so sánh, phép Like được thực hiện cho dữ liệu kiểu chuỗi, phép Like được xem là phép so sánh theo định dạng của chuỗi, việc định dạng xác định theo một số từ khóa sau:

%	Xác định bất kỳ chuỗi ký tự nào hoặc không có ký tự nào tại vị trí.
_	Một ký tự bất kỳ nào đó.
[]	Một ký tự nào đó nằm trong phạm vi, ví dụ [a-f].
[^]	Xác định một ký tự không thuộc phạm vi nào đó, ví dụ [^a-f].

Ví dụ sử dụng từ khóa Like với %:

```
USE pubs
GO
SELECT phone
FROM authors
WHERE phone LIKE '415%'
ORDER by au_lname
GO
```

Ví dụ từ khóa Not Like với %:

```
USE pubs
GO
SELECT phone
FROM authors
WHERE phone NOT LIKE '415%'
ORDER BY au_lname
GO
```

Ví dụ sử dụng từ khóa Like với mệnh đề Escape: Escape được sử dụng loại bỏ một ký tự hoặc chuỗi khỏi phép so sánh.

```
USE pubs
GO
```

```
IF EXISTS(SELECT TABLE_NAME FROM
INFORMATION_SCHEMA.TABLES
      WHERE TABLE_NAME = 'mytbl2')
   DROP TABLE mytbl2
GO
USE pubs
GO
CREATE TABLE mytbl2
(
 c1 sysname
)
GO
INSERT mytbl2 VALUES ('Discount is 10-15% off')
INSERT mytbl2 VALUES ('Discount is .10-.15 off')
GO
SELECT c1
FROM mytbl2
WHERE c1 LIKE '%10-15!% off%' ESCAPE '!'
GO
```

Ví dụ sử dụng từ khóa Like với []:

```
USE pubs
GO
SELECT au_lname, au_fname, phone
FROM authors
WHERE au_lname LIKE '[CK]ars[eo]n'
ORDER BY au_lname ASC, au_fname ASC
GO
```

+ Giá trị NULL.

Giá trị NULL được nhập bằng cách đặt ngầm định hoặc gán theo câu lệnh. Để tìm giá trị NULL trong bảng sử dụng từ khóa Is Null hoặc Is Not Null.

```
SELECT title_id, type, advance
FROM pubs.dbo.titles
WHERE advance IS NULL
```

TẠO BẢNG BẰNG LỆNH SELECT INTO.

Lệnh Select Into truy vấn dữ liệu, dữ liệu được đưa vào một bảng mới. Nếu thuộc tính select into/bulkcopy được đặt có thể tạo bảng cố định, nếu thuộc tính không được đặt ta có thể tạo bảng tạm thời.

```
SELECT Shippers.*, Link.Address, Link.City,
                   Link.Region, Link.PostalCode
INTO NewShippers
FROM Shippers
     JOIN LinkServer.DB.dbo.Shippers AS Link
       ON (Shippers.ShipperID = Link.ShipperID)
```

LỆNH COMPUTE BY.

Khi thực hiện với các hàm tính toán SUM, AVG, MIN, MAX, COUNT thường được sử dụng với các mệnh đề GROUP BY, COMPUTE BY (không áp dụng các hàm tính toán với dữ liệu kiểu Text, Image).

+ Sử dụng Group By: Từ khóa Group By được sử dụng nhóm theo cột, có thể kết hợp các hàm tính toán.

```
USE Northwind
SELECT OrdD.ProductID AS ProdID,
       SUM(OrdD.Quantity) AS AmountSold
FROM [Order Details] AS OrdD JOIN Products as Prd
     ON OrdD.ProductID = Prd.ProductID
     AND Prd.CategoryID = 2
GROUP BY OrdD.ProductID
```

+ Sử dụng mệnh đề Compute: Tính toán toàn bộ giá trị.

```
USE pubs
SELECT type, price, advance
FROM titles
ORDER BY type
COMPUTE SUM(price), SUM(advance)
```

+ *Sử dụng mệnh đề Compute By*: Tính toán theo nhóm (tương tự Group By).

```
USE pubs
SELECT type, price, advance
FROM titles
ORDER BY type
COMPUTE SUM(price), SUM(advance) BY type
```

TOÁN TỬ UNION.

Toán tử Union thực hiện hợp 2 tập với nhau, phép toán này thực hiện chỉ lấy đại diện khi có hai hàng của hai tập trùng nhau.

Giả sử có 2 bảng dữ liệu như sau:

Table1			Table2	
ColumnA	ColumnB		ColumnC	ColumnD
char(4)	int		char(4)	int
-------	---		-------	---
abc	1		ghi	3
def	2		jkl	4
ghi	3		mno	5

Thực hiện toán tử Union:

```
SELECT * FROM Table1
UNION
SELECT * FROM Table2
```

Kết quả thực hiện:

```
ColumnA  ColumnB
-------  --------
abc      1
def      2
ghi      3
jkl      4
```

```
mno      5
```

Khi sử dụng từ khóa ALL toàn bộ các hàng của hai tập dữ liệu sẽ được hợp lại, không loại bỏ những hàng trùng nhau.

```
SELECT * FROM TableA
UNION ALL
(   SELECT * FROM TableB
   UNION
   SELECT * FROM TableC
)
GO
```

Sử dụng toán tử Union với Select Into:

```
USE Northwind
IF EXISTS(SELECT TABLE_NAME FROM
INFORMATION_SCHEMA.TABLES
      WHERE TABLE_NAME = 'CustomerResults')
   DROP TABLE CustomerResults
GO
USE Northwind
SELECT ContactName, CompanyName, City, Phone INTO
CustomerResults
FROM Customers
WHERE Country IN ('USA', 'Canada')
UNION
SELECT ContactName, CompanyName, City, Phone
FROM SouthAmericanCustomers
ORDER BY CompanyName, ContactName ASC
GO
```

TRUY VẤN DỮ LIỆU TỪ NHIỀU BẢNG.

Truy vấn dữ liệu từ nhiều bảng được xác định theo quan hệ giữa các cột của các bảng với nhau. Có thể truy vấn thông qua điều kiện liên kết trong mệnh đề Where hoặc từ khóa Join.

Theo điều kiện liên kết.

Sử dụng điều kiện liên kết theo cột giữa các bảng, thông tin cần truy vấn được đặt ở nhiều bảng khác nhau, để truy vấn được các thông tin như trên phải xác định điều kiện liên kết giữa các bảng.

+ Liên kết bằng nhau.

```
SELECT P.ProductID,
       S.SupplierID,
       S.CompanyName
FROM Suppliers AS S, Products AS P
WHERE S.SupplierID = P.SupplierID
  AND P.UnitPrice > $10
  AND S.CompanyName LIKE N'F%'
```

Đối với câu lệnh truy vấn theo điều kiện liên kết nói trên, các hàng chứa giá trị Null của cột tham gia liên kết sẽ không được liệt kê, câu lệnh này tương đương với lệnh Inner Join (sẽ xem trong phần sau).

+ Liên kết không bằng nhau.

Liên kết dạng này sử dụng các phép toán so sánh >, >=, <, <=, <>, !>, !<

```
USE pubs
SELECT p.pub_name, p.state, a.au_lname, a.au_fname,
a.state
FROM publishers p, authors a
   WHERE a.state > p.state and
p.pub_name = 'New Moon Books'
ORDER BY au_lname ASC, au_fname ASC
```

+ Tự liên kết bằng nhau.

Tự liên kết trong một bảng, câu lệnh dạng này thường được sử dụng trong việc xác định những cặp giá trị nào các cột trong bảng có quan hệ với nhau theo liên kết.

```
USE pubs
SELECT au1.au_fname, au1.au_lname, au2.au_fname,
au2.au_lname
FROM authors au1, authors au2
   WHERE au1.zip = au2.zip and au1.city = 'Oakland'
ORDER BY au1.au_fname ASC, au1.au_lname ASC
```

+ Tự liên kết không bằng nhau.

```
USE pubs
SELECT au1.au_fname, au1.au_lname, au2.au_fname,
au2.au_lname
FROM authors au1, authors au2
WHERE  au1.zip = au2.zip
   AND au1.city = 'Oakland'
   AND au1.state = 'CA'
   AND au1.au_id < au2.au_id
ORDER BY au1.au_lname ASC, au1.au_fname ASC
```

+ Truy vấn dữ liệu từ nhiều hơn 2 bảng dữ liệu.

Truy vấn dạng này thực hiện điều kiện liên kết theo từng cặp các bảng với nhau.

```
USE pubs
SELECT a.au_lname, a.au_fname, t.title
FROM authors a, titleauthor ta, titles t
     WHERE a.au_id = ta.au_id
      AND ta.title_id = t.title_id
      AND t.type = 'trad_cook'
      ORDER BY t.title ASC
```

+ Liên kết ngoài trái.

Như những điều kiện liên kết nói trên, những hàng có cột là Null sẽ không được đưa ra tập kết quả, câu lệnh liên kết ngoài sẽ đưa ra những hàng chứa giá trị Null nói trên. Xác định liên kết ngoài bằng toán tử *.

```
USE pubs
SELECT a.au_fname, a.au_lname, p.pub_name
```

```
FROM authors a, publishers p
WHERE a.city *= p.city
ORDER BY p.pub_name ASC, a.au_lname ASC, a.au_fname ASC
```

Bảng authors sẽ đưa ra cả những hàng có cột có cột city là Null, khi đó chưa có pub_name, kết quả như sau:

```
au_fname             au_lname                       pub_name
----------         ----------------                 --------
Reginald             Blotchet-Halls                 NULL
Michel               DeFrance                       NULL
Innes                del Castillo                   NULL
Ann                  Dull                           NULL
Marjorie             Green                          NULL
Morningstar          Greene                         NULL
Burt                 Gringlesby                     NULL
Sheryl               Hunter                         NULL
Livia                Karsen                         NULL
Charlene             Locksley                       NULL
Stearns              MacFeather                     NULL
Heather              McBadden                       NULL
Michael              O'Leary                        NULL
Sylvia               Panteley                       NULL
Albert               Ringer                         NULL
Anne                 Ringer                         NULL
Meander              Smith                          NULL
Dean                 Straight                       NULL
Dirk                 Stringer                       NULL
Johnson              White                          NULL
Akiko                Yokomoto                       NULL
Abraham              Bennet                 Algodata Infosystems
Cheryl               Carson                 Algodata Infosystems

(23 row(s) affected)
```

+ Liên kết ngoài phải.

```
USE pubs
SELECT a.au_fname, a.au_lname, p.pub_name
FROM authors AS a, publishers AS p
WHERE a.city = p.city
ORDER BY p.pub_name ASC, a.au_lname ASC, a.au_fname ASC
```

Kết quả thực hiện:

```
au_fname             au_lname                 pub_name
-------------------- ----------------------  --------------------
Abraham              Bennet                   Algodata
Infosystems
Cheryl               Carson                   Algodata
Infosystems
NULL                 NULL                     Binnet & Hardley
NULL                 NULL                     Five Lakes
Publishing
NULL                 NULL                     GGG&G
NULL                 NULL                     Lucerne Publishing
NULL                 NULL                     New Moon Books
NULL                 NULL                     Ramona Publishers
NULL                 NULL                     Scootney Books

(9 row(s) affected)
```

Lệnh Join – Truy vấn từ nhiều bảng.

Phần trên ta đã xem xét kỹ thuật truy vấn dữ liệu từ nhiều bảng sử dụng điều kiện liên kết, tương tự như các phép toán so sánh, *=, =* SQL Server cung cấp câu lệnh Join thay thế các phép toán nói trên.

+ Inner Join – Liên kết trong.

Thay vì xác định điều kiện liên kết trong mệnh đề Where thì ở đây ta chỉ cần xác định liên kết trong mệnh đề From.

Liên kết bằng:

```
USE pubs
SELECT *
FROM authors AS a INNER JOIN publishers AS p
   ON a.city = p.city
ORDER BY a.au_lname DESC
```

Liên kết không bằng:

```
USE pubs
```

```
SELECT p.pub_name, p.state, a.au_lname, a.au_fname,
a.state
FROM publishers p INNER JOIN authors a
   ON a.state > p.state
WHERE p.pub_name = 'New Moon Books'
ORDER BY au_lname ASC, au_fname ASC
```

+ Tự liên kết trong bảng.

Tự liên kết bằng:

```
USE pubs
SELECT au1.au_fname, au1.au_lname, au2.au_fname,
au2.au_lname
FROM authors au1 INNER JOIN authors au2
   ON au1.zip = au2.zip
WHERE au1.city = 'Oakland'
ORDER BY au1.au_fname ASC, au1.au_lname ASC
```

Tự liên kết không bằng:

```
USE pubs
SELECT au1.au_fname, au1.au_lname, au2.au_fname,
au2.au_lname
FROM authors au1 INNER JOIN authors au2
   ON  au1.zip = au2.zip
WHERE au1.city = 'Oakland'
   AND au1.state = 'CA'
   AND au1.au_id < au2.au_id
ORDER BY au1.au_lname ASC, au1.au_fname ASC
```

+ Liên kết nhiều hơn 2 bảng.

```
USE pubs
SELECT a.au_lname, a.au_fname, t.title
FROM authors a INNER JOIN titleauthor ta
```

```
   ON a.au_id = ta.au_id JOIN titles t
   ON ta.title_id = t.title_id
WHERE t.type = 'trad_cook'
ORDER BY t.title ASC
```

+ Liên kết ngoài trái - LEFT OUTER JOIN.

Liên kết ngoài trái tương tự như phép toán *=.

```
USE pubs
SELECT a.au_fname, a.au_lname, p.pub_name
FROM authors a LEFT OUTER JOIN publishers p
   ON a.city = p.city
ORDER BY p.pub_name ASC, a.au_lname ASC, a.au_fname ASC
```

+ Liên kết ngoài phảii – RIGHT OUTER JOIN.

Liên kết ngoài phải tương tự như phép toán =*.

```
USE pubs
SELECT a.au_fname, a.au_lname, p.pub_name
FROM authors a RIGHT OUTER JOIN publishers p
   ON a.city = p.city
ORDER BY p.pub_name ASC, a.au_lname ASC, a.au_fname ASC
```

+ Liên kết ngoài 2 phía – FULL OUTER JOIN.

Là phép liên kết trái hoặc phải.

```
USE pubs
SELECT a.au_fname, a.au_lname, p.pub_name
FROM authors a FULL OUTER JOIN publishers p
   ON a.city = p.city
ORDER BY p.pub_name ASC, a.au_lname ASC, a.au_fname ASC
```

Kết quả như sau:

```
au_fname             au_lname                  pub_name
---------------    ------------------------  -----------------
Reginald             Blotchet-Halls               NULL
Michel               DeFrance                     NULL
Innes                del Castillo                 NULL
Ann                  Dull                         NULL
Marjorie             Green                        NULL
Morningstar          Greene                       NULL
Burt                 Gringlesby                   NULL
Sheryl               Hunter                       NULL
Livia                Karsen                       NULL
Charlene             Locksley                     NULL
Stearns              MacFeather                   NULL
Heather              McBadden                     NULL
Michael              O'Leary                      NULL
Sylvia               Panteley                     NULL
Albert               Ringer                       NULL
Anne                 Ringer                       NULL
Meander              Smith                        NULL
Dean                 Straight                     NULL
Dirk                 Stringer                     NULL
Johnson              White                        NULL
Akiko                Yokomoto                     NULL
Abraham              Bennet                 Algodata Infosystems
Cheryl               Carson                 Algodata Infosystems
NULL                 NULL                   Binnet & Hardley
NULL                 NULL                   Five Lakes Publishing
NULL                 NULL                   GGG&G
NULL                 NULL                   Lucerne Publishing
NULL                 NULL                   New Moon Books
NULL                 NULL                   Ramona Publishers
NULL                 NULL                   Scootney Books

(30 row(s) affected)
```

+ Giá trị Null và phép Join.

Giá trị Null không xác định trong phép so sánh của mệnh đề Where (chỉ sử dụng với các phép so sánh Is Null hoặc Is Not Null), trong phép Join ta có thể xác định giống nhau giữa 2 giá trị Null. Xét ví dụ sau:

Giả sử có 2 bảng dữ liệu có giá trị như sau:

```
table1                          table2
a           b                   c           d
-------     ------              -------     ------
      1        one                NULL        two
   NULL      three                   4       four
      4      join4
```

Thực hiện phép Join như sau:

```
SELECT *
FROM table1 t1 JOIN table2 t2
   ON t1.a = t2.c
ORDER BY t1.a
```

Kết quả thực hiện:

```
a           b      c           d
----------- ------ ----------- ------
4           join4  4           four

(1 row(s) affected)
```

TRUY VẤN TỔNG HỢP.

Việc sử dụng các hàm tính toán như SUM, AVG,… thường được thực hiện theo các mệnh đề WHERE, GROUP BY, HAVING. Khi xác định điều kiện có sử dụng các hàm tính toán thì phải sử dụng mệnh đề HAVING mà không được sử dụng trong mệnh đề WHERE.

Các hàm tính toán có thể tóm tắt như sau:

SUM([ALL | DISTINCT])	Tính tổng tất cả hoặc những hàng khác nhau.
AVG([ALL | DISTINCT])	Tính trung bình tất cả hoặc những hàng khác nhau.
COUNT([ALL | DISTINCT])	Đếm số hàng tất cả hoặc những hàng khác nhau.
COUNT(*)	Đếm các hàng được lựa chọn.
MAX()	Tính giá trị lớn nhất.

MIN() Tính giá trị nhỏ nhất.

Các hàm SUM, AVG chỉ làm việc với dữ liệu dạng số, các hàm SUM, AVG, COUNT, MAX, MIN bỏ qua giá trị Null, hàm COUNT(*) đếm cả hàng có giá trị Null.

Sử dụng hàm tính toán.

+ Tính tổng toàn bộ.

```
USE pubs
SELECT SUM(ytd_sales)
FROM titles
```

+ Tính tổng, trung bình có điều kiện.

```
USE pubs
SELECT AVG(advance), SUM(ytd_sales)
FROM titles
WHERE type = 'business'
```

Mệnh đề Group By.

Group by được thực hiện nhóm các hàng theo giá trị cột xác định, các hàm tính toán sẽ được thực hiện theo nhóm nói trên.

```
USE Northwind
SELECT OrdD.ProductID AS ProdID,
       SUM(OrdD.Quantity) AS AmountSold
FROM [Order Details] AS OrdD JOIN Products as Prd
     ON OrdD.ProductID = Prd.ProductID
     AND Prd.CategoryID = 2
GROUP BY OrdD.ProductID
```

Kết quả thực hiện như sau:

```
ProdID      AmountSold
----------- -----------
3           328
4           453
5           298
6           301
8           372
15          122
44          601
61          603
63          445
65          745
66          239
77          791

(12 row(s) affected)
```

Mệnh đề Having.

Having được sử dụng cùng với các hàm tính toán xác định điều kiện lọc các hàng, thường được kết hợp cùng mệnh đề Group By để thực hiện các hàm tính toán theo nhóm.

+ Having với hàm SUM.

```
USE pubs
SELECT pub_id, total = SUM(ytd_sales)
FROM titles
GROUP BY pub_id
HAVING SUM(ytd_sales) > 40000
```

+ Having với hàm Count.

```
USE pubs
SELECT pub_id, total = SUM(ytd_sales)
FROM titles
GROUP BY pub_id
HAVING COUNT(*) > 5
```

+ Having với mệnh đề Where.

```
SELECT pub_id, SUM(advance) AS AmountAdvanced,
       AVG(price) AS AveragePrice
FROM pubs.dbo.titles
WHERE pub_id > '0800'
  AND price >= $5
GROUP BY pub_id
HAVING SUM(advance) > $15000
   AND AVG(price) < $20
ORDER BY pub_id DESC
```

+ Having thay cho mệnh đề Where.

```
SELECT titles.pub_id, AVG(titles.price)
FROM titles INNER JOIN publishers
   ON titles.pub_id = publishers.pub_id
GROUP BY titles.pub_id
HAVING publishers.state = 'CA'
```

TRUY VẤN LỒNG NHAU.

Phần này sẽ xem xét các câu lệnh truy vấn lồng nhau, trong câu lệnh truy vấn Select có câu lệnh truy vấn Select khác trong điều kiện xác định của lệnh Select ngoài. Thông thường các câu lệnh dạng này đi cùng các từ lhóa IN, NOT IN, EXITST, NOT EXIST, ANY, ALL.

Truy vấn lồng nhau với phép bằng.

```
USE pubs
SELECT title, price
FROM titles
WHERE price =
   (SELECT price
   FROM titles
   WHERE title = 'Straight Talk About Computers')
```

Đầu tiên câu lệnh sẽ xác định hàng trong lệnh Select trong, lệnh truy vấn này phải đưa ra kết quả duy nhất.

Truy vấn với từ khóa IN.

Kiểm tra nằm trong tập các giá trị truy vấn được.

```
USE pubs
SELECT distinct pub_name
FROM publishers
WHERE pub_id IN
   (SELECT pub_id
   FROM titles
   WHERE type = 'business')
```

Hằng số nằm trong khoảng:

```
USE pubs
SELECT DISTINCT au_lname, au_fname
FROM authors
WHERE 100 IN
   (SELECT royaltyper
   FROM titleauthor
   WHERE titleauthor.au_id = authors.au_id)
```

Truy vấn với từ khóa Exist.

Kiểm tra tồn tại hàng dữ liệu truy vấn được.

```
USE pubs
SELECT DISTINCT pub_name
FROM publishers
WHERE EXISTS
   (SELECT *
   FROM titles
   WHERE pub_id = publishers.pub_id
   AND type = 'business')
```

Truy vấn với hàm All.

Kiểm tra với tất cả các hàng.

```
USE pubs
SELECT t1.type
FROM titles t1
GROUP BY t1.type
HAVING MAX(t1.advance) >= ALL
   (SELECT 2 * AVG(t2.advance)
   FROM titles t2
   WHERE t1.type = t2.type)
```

Truy vấn với hàm Any.

Kiểm tra thỏa mãn với bất kỳ hàng nào.

```
USE pubs
SELECT title
FROM titles
WHERE advance > ANY
   (SELECT advance
   FROM publishers INNER JOIN titles
   ON titles.pub_id = publishers.pub_id
      AND pub_name = 'Algodata Infosystems')
```

Truy vấn với hàm Some.

Kiểm tra với ít nhất một hàng.

```
USE pubs
SELECT t1.type
FROM titles t1
GROUP BY t1.type
HAVING MAX(t1.advance) >= SOME
   (SELECT 2 * AVG(t2.advance)
   FROM titles t2
   WHERE t1.type = t2.type)
```

Nhiều lệnh Select lồng nhau.

```
USE pubs
SELECT au_lname, au_fname
FROM authors
WHERE au_id IN
   (SELECT au_id
   FROM titleauthor
   WHERE title_id IN
      (SELECT title_id
      FROM titles
      WHERE type = 'popular_comp'))
```

UPDATE, DELETE, INSERT VỚI LỆNH TRUY VẤN LỒNG NHAU.

Việc thực hiện các lệnh thao tác với dữ liệu có thể kết hợp điều kiện truy vấn lồng nhau để xác định phạm vi dữ liệu được thao tác.

Kết hợp với lenệh Select.

```
UPDATE titles
SET price = price * 2
WHERE pub_id IN
   (SELECT pub_id
   FROM publishers
   WHERE pub_name = 'New Moon Books')
```

Kết hợp với lệnh Join.

```
UPDATE titles
SET price = price * 2
FROM titles INNER JOIN publishers ON titles.pub_id =
publishers.pub_id
   AND pub_name = 'New Moon Books'
```

Xóa dữ liệu kết hợp với lệnh Select.

```
DELETE sales
WHERE title_id IN
   (SELECT title_id
   FROM titles
   WHERE type = 'business')
```

Xóa dữ liệu với phép Join.

```
DELETE sales
FROM sales INNER JOIN titles ON sales.title_id =
titles.title_id
   AND type = 'business'
```

LỆNH READTEXT – ĐỌC TEXT, IMAGE.

Lệnh ReadText được thực hiện đọc dữ liệu kiểu Text, Image và chuyển vào một biến.

```
USE pubs
GO
DECLARE @ptrval varbinary(16)
SELECT @ptrval = TEXTPTR(pr_info)
   FROM pub_info pr INNER JOIN publishers p
      ON pr.pub_id = p.pub_id
      AND p.pub_name = 'New Moon Books'
READTEXT pub_info.pr_info @ptrval 1 25
GO
```

Ví dụ trên thực hiện đọc dữ liệu từ cột pr_info bắt đầu từ vị trí 1, độ dài 25 byte.

THAO TÁC DỮ LIỆU NGOÀI.

Nội dung phần này sẽ giới thiệu câu lệnh, kỹ thuật truy vấn dữ liệu của hệ quản trị CSDL khác hoặc Instance khác.

Lệnh OpenRowSet.

Lệnh OpenRowSet sử dụng truy nhập dữ liệu xa với nguồn dữ liệu là OLE DB, kết nối kiểu này có thể thực hiện các lệnh Insert, Update, Delete, Select với bảng dữ liệu. Quyền thực hiện trong câu lệnh thực hiện theo user kết nối trong câu lệnh.

+ OPENROWSET với lệnh SELECT và Microsoft OLE DB Provider for SQL Server.

```
USE pubs
GO
SELECT a.*
FROM OPENROWSET('SQLOLEDB','seattle1';'sa';'MyPass',
   'SELECT * FROM pubs.dbo.authors ORDER BY au_lname,
au_fname') AS a
GO
```

Ví dụ trên thực hiện kết nối đến Instance có tên seattle1, user có tên sa, mật khẩu MyPass.

+ OPENROWSET với OLE DB Provider for ODBC.

```
USE pubs
GO
SELECT a.*
FROM OPENROWSET('MSDASQL',
   'DRIVER={SQL
Server};SERVER=seattle1;UID=sa;PWD=MyPass',
   pubs.dbo.authors) AS a
ORDER BY a.au_lname, a.au_fname
GO
```

+ Microsoft OLE DB Provider for Jet. Lệnh dạng này được này được thực hiện kết nối đến Access.

```
USE pubs
GO
SELECT a.*
FROM OPENROWSET('Microsoft.Jet.OLEDB.4.0',
```

```
'c:\MSOffice\Access\Samples\northwind.mdb';'admin';'myp
wd', Orders)
   AS a
GO
```

+ OPENROWSET với INNER JOIN một bảng khác.

```
USE pubs
GO
SELECT c.*, o.*
FROM Northwind.dbo.Customers AS c INNER JOIN
   OPENROWSET('Microsoft.Jet.OLEDB.4.0',

'c:\MSOffice\Access\Samples\northwind.mdb';'admin';'myp
wd', Orders)
   AS o
   ON c.CustomerID = o.CustomerID
GO
```

Lệnh OpenDataSource.

Lệnh OpenDataSource thực hiện mở dữ liệu ngoài Instance, không cần đến linked_server.

+ Kết nối đến Instance khác.

```
SELECT   *
FROM      OPENDATASOURCE(
         'SQLOLEDB',
         'Data Source=ServerName;User
ID=MyUID;Password=MyPass'
    ).Northwind.dbo.Categories
```

+ Kết nối đến Excel.

```
SELECT *
FROM OpenDataSource( 'Microsoft.Jet.OLEDB.4.0',
  'Data Source="c:\Finance\account.xls";User
ID=Admin;Password=;Extended properties=Excel
5.0')...xactions
```

Lệnh OpenQuery.

Lệnh OpenQuery thực hiện thao tác với dữ liệu ngoài thông qua LinkedServer.

```
EXEC sp_addlinkedserver 'OracleSvr',
   'Oracle 7.3',
   'MSDAORA',
   'ORCLDB'
GO
SELECT *
FROM OPENQUERY(OracleSvr, 'SELECT name, id FROM
joe.titles')
GO
```

MỘT SỐ HÀM CƠ BẢN.

Hàm hệ thống.

DB_ID	Trả về ID của CSDL khi biết tên.
DB_NAME	Trả về tên CSDL khi biết ID.
HOST_ID	Trả về ID của máy chủ.
HOST_NAME	Trả về tên máy chủ
SUSER_ID	Trả về ID User của Server khi biết tên
SUSER_NAME	Trả về tên User của Server khi biết ID.
USER_ID	Trả về ID User khi biết tên
USER_NAME	Trả về tên User khi biết ID

Hàm thao tác với chuỗi.

+ SUBSTRING - Lấy chuỗi nhỏ trong chuỗi.

SUBSTRING **(** *expression* **,** *start* **,** *length* **)**

Sử dụng với chuỗi ký tự:

```
USE pubs
SELECT au_lname, SUBSTRING(au_fname, 1, 1)
FROM authors
ORDER BY au_lname
```

Sử dụng với text, ntext, image:

```
USE pubs
SELECT pub_id, SUBSTRING(logo, 1, 10) AS logo,
   SUBSTRING(pr_info, 1, 10) AS pr_info
FROM pub_info
WHERE pub_id = '1756'
```

+ CHARINDEX – Trả về vị trí bắt đầu một mẫu trong chuỗi.

CHARINDEX **(** *expression1* **,** *expression2* [**,** *start_location*] **)** – Tìm vị trí xuất hiện chuỗi expression1 trong expression2.

Ví dụ tìm chuỗi ‘wonderful’ trong cột notes của bảng titles:

```
USE pubs
GO
SELECT CHARINDEX('wonderful', notes)
FROM titles
WHERE title_id = 'TC3218'
GO
```

+ PATINDEX – Trả về vị trí xuất hiện của mẫu trong chuỗi.

PATINDEX **(** '*%pattern%*' **,** *expression* **)**

Ví dụ trìm vị trí xuất hiện mẫu ‘%wonderful%’:

```
USE pubs
GO
SELECT PATINDEX('%wonderful%', notes)
FROM titles
WHERE title_id = 'TC3218'
GO
```

Ví dụ trìm vị trí xuất hiện mẫu ‘%won_erful%’:

```
USE pubs
GO
SELECT PATINDEX('%won_erful%', notes)
FROM titles
WHERE title_id = 'TC3218'
GO
```

+ STR – Chuyển dữ liệu kiểu số sáng chuỗi.

STR **(** *float_expression* [**,** *length* [**,** *decimal*]] **)**

Ví dụ chuyển số sang chuỗi có độ dài 6, làm tròn sau dấu phảy 1 số.

```
SELECT STR(123.45, 6, 1)
GO
```

Kết quả là chuỗi ‘123.5’

Ví dụ sử dụng với hàm Floor lấy giá trị nguyên nhỏ hơn của một số thực:

```
SELECT STR (FLOOR (123.45), 8, 3)
GO
```

Kết quả là ‘123.000’

+ STUFF – Chèn một chuỗi vào một chuỗi khác.

Hàm Stuff thực hiện xóa chuỗi nhỏ trong một chuỗi sau đó thực hiện chèn một chuỗi mới vào vị trí bắt đầu.

STUFF **(** *character_expression* **,** *start* **,** *length* **,** *character_expression* **)**

Ví dụ:

```
SELECT STUFF('abcdef', 2, 3, 'ijklmn')
GO
```

Kết quả thực hiện:

```
aijklmnef
```

+ SOUNDEX – Trả về hàm phát âm.

Hàm Soundex sử dụng so sánh phát âm giữa 2 chuỗi, ví dụ sau sẽ cho 2 mã Soundex như nhau:

```
SELECT SOUNDEX ('Smith'), SOUNDEX ('Smythe')
```

Kết quả thực hiện:

```
S530  S530
```

+ Defference – So sánh giá trị hàm Soundex giữa 2 chuỗi: Giá trị trả về từ 0 đến 4, 4 là giá trị giống nhau nhất. Ví dụ sau so sánh giữa 2 chuỗi:

```
SELECT DIFFERENCE('Smithers', 'Smythers')
GO
```

Kết quả thực hiện: 4

```
SELECT DIFFERENCE('Anothers', 'Brothers')
GO
```

Kết quả thực hiện: 2

+ UNICODE – Lấy mã unicode ký tự đầu tiên trong chuỗi.

+ NCHAR – Chuyển mã unicode thành ký tự.

Các hàm DateTime.

+ GETDATE: Trả về ngày, giờ hiện tại.

+ DATEPART: Trả về giá trị ngay hoặc tháng hoặc năm của một biểu thức ngày.

DATEPART **(** *datepart* **,** *date* **)**

Giá trị datepart theo bảng sau:

Datepart	Dạng rút gọn
year	**yy, yyyy**
quarter	**qq, q**
month	**mm, m**
dayofyear	**dy, y**
day	**dd, d**
week	**wk, ww**
weekday	**dw**
hour	**hh**
minute	**mi, n**
second	**ss, s**
millisecond	**ms**

```
SELECT DATEPART(m, 0), DATEPART(d, 0), DATEPART(yy, 0)
```

+ SET DATFIRST: Đặt ngày đầu tiên trong tuần.

+ SET DATEFORMAT: Đặt định dạng kiểu DateTime để nhập dữ liệu.

```
SET DATEFORMAT mdy
GO
DECLARE @datevar datetime
SET @datevar = '12/31/98'
SELECT @datevar
GO
```

+ DAY, MONTH, YEAR: Lấy giá trị ngày, tháng, năm.

+ ISDATE: Kiểm tra xem dữ liệu có hợp lệ DateTime không.

+ DATEDIFF: Xác định độ lệch giữa 2 giá trị DateTime.

DATEDIFF **(** *datepart* **,** *startdate* **,** *enddate* **)**

Ví dụ: Xác định số ngày đã phát hành sách.

```
USE pubs
GO
SELECT DATEDIFF(day, pubdate, getdate()) AS no_of_days
FROM titles
GO
```

+ DATEADD – Xác định giá DateTime mới khi thay đổi một khoảng thời gian.

DATEADD **(** *datepart* **,** *number***,** *date* **)**

```
USE pubs
GO
SELECT DATEADD(day, 21, pubdate) AS timeframe
FROM titles
GO
```

Các hàm chuyển đổi.

+ CONVERT

+ CAST

TRANSACTION – PHIÊN GIAO DỊCH.

Transaction là một đơn vị công việc trong nó bao gồm nhiều việc nhỏ, các việc này được thực hiện thành công thì Transaction thành công, dữ liệu thay đổi trong quá trình thực hiện của Transaction sẽ được cập nhật. Nếu trong quá trình có phát sinh lỗi thì Transaction sẽ lặp lại (Roll Back hoặc Cancel), dữ liệu không được cập nhật. Một phiên giao dịch có 4 đặc tính ACID (Atomicity, Consistency, Isolation, Durability).

Atomicity – Nguyên tố: Một phiên giao dịch là một đơn vị công việc nhỏ nhất, tất cả dữ liệu thay đổi trong phiên giao dịch được thực hiện hoặc tất cả không được thực hiện.

Consistency- Nhất quán: Giao dịch sẽ không thực hiện nếu có một thao tác xung khắc về mặt logic hoặc quan hệ. Tính nhất quán rất quan trọng với mô hình ứng dụng client/server, với mô hình dạng này tại một thời điểm có thể có nhiều giao dịch thực hiện đồng thời, nếu một giao dịch nào đó không nhất quán thì tất cả các giao dịch khác sẽ thực hiện sai, dẫn đến sự vi phạm toàn vẹn dữ liệu.

Isolation – Tách biệt: Tại một thời điểm có nhiều phiên giao dịch đồng thời, các phiên giao dịch chỉ tác động với nhau khi dữ liệu được cập nhật (kết thúc phiên). Giả sử có 2 phiên giao dịch có tác động

Durability - Bền vững: Sau khi giao dịch hoàn tất, dữ liệu ở trạng thái bền vững.

Một phiên giao dịch được xác định bắt đầu, kết thúc:

Bắt đầu phiên giao dịch.

Phiên giao dịch có 3 loại: explicit transaction, implicit transaction, autocommit transaction.

Explicit transaction: Là kiểu phiên giao dịch rõ, được bắt đầu bằng lệnh BEGIN TRANSACTION, đối với phiên giao dịch phân tán thì được bắt đầu bằng lệnh

BEGIN DISTRIBUTED TRAN

[*transaction_name* | *@tran_name_variable*]

Đặt tên giao dịch:

```
DECLARE @TranName VARCHAR(20)
SELECT @TranName = 'MyTransaction'

BEGIN TRANSACTION @TranName
GO
USE pubs
GO
UPDATE roysched
SET royalty = royalty * 1.10
WHERE title_id LIKE 'Pc%'
GO

COMMIT TRANSACTION MyTransaction
GO
```

Đánh dấu trong giao dịch:

```
BEGIN TRANSACTION RoyaltyUpdate
   WITH MARK 'Update royalty values'
GO
USE pubs
GO
UPDATE roysched
   SET royalty = royalty * 1.10
   WHERE title_id LIKE 'Pc%'
GO
COMMIT TRANSACTION RoyaltyUpdate
GO
```

Autocommit transaction: Mỗi câu lệnh tự cập nhật dư liệu khi nó kết thúc, không cần câu lệnh điều khiển phiên giao dịch.

Implicit transaction: Là phiên giao dịch ẩn, đặt chế độ này thông qua hàm API hoặc lệnh SET IMPLICIT_TRANSACTIONS ON. Khi phiên giao dịch kết thúc, câu lệnh T-SQL tiếp theo sẽ khởi động phiên giao dịch mới.

SET IMPLICIT_TRANSACTIONS { ON | OFF }

Sử dụng kết hợp với Implicit transaction:

```
USE pubs
GO

CREATE table t1 (a int)
GO
INSERT INTO t1 VALUES (1)
GO

PRINT 'Use explicit transaction'
BEGIN TRAN
INSERT INTO t1 VALUES (2)
SELECT 'Tran count in transaction'= @@TRANCOUNT
COMMIT TRAN
SELECT 'Tran count outside transaction'= @@TRANCOUNT
GO

PRINT 'Setting IMPLICIT_TRANSACTIONS ON'
GO
SET IMPLICIT_TRANSACTIONS ON
GO

PRINT 'Use implicit transactions'
GO
-- No BEGIN TRAN needed here.
INSERT INTO t1 VALUES (4)
SELECT 'Tran count in transaction'= @@TRANCOUNT
COMMIT TRAN
SELECT 'Tran count outside transaction'= @@TRANCOUNT
GO

PRINT 'Use explicit transactions with
IMPLICIT_TRANSACTIONS ON'
GO
BEGIN TRAN
INSERT INTO t1 VALUES (5)
SELECT 'Tran count in transaction'= @@TRANCOUNT
COMMIT TRAN
SELECT 'Tran count outside transaction'= @@TRANCOUNT
GO
```

```
SELECT * FROM t1
GO

-- Need to commit this tran too!
DROP TABLE t1
COMMIT TRAN
GO
```

Kết thúc phiên giao dịch.

Sử dụng lệnh Commit trong phiên giao dịch.

COMMIT [TRAN [SACTION] [*transaction_name* | **@***tran_name_variable*]]

+ Commit một phiên giao dịch.

```
BEGIN TRANSACTION
USE pubs
GO
UPDATE titles
SET advance = advance * 1.25
WHERE ytd_sales > 8000
GO
COMMIT
GO
```

+ Commit nhiều phiên giao dịch lồng nhau.

```
CREATE TABLE TestTran (Cola INT PRIMARY KEY, Colb CHAR(3))
GO
BEGIN TRANSACTION OuterTran -- @@TRANCOUNT set to 1.
GO
INSERT INTO TestTran VALUES (1, 'aaa')
GO
BEGIN TRANSACTION Inner1 -- @@TRANCOUNT set to 2.
GO
INSERT INTO TestTran VALUES (2, 'bbb')
GO
BEGIN TRANSACTION Inner2 -- @@TRANCOUNT set to 3.
GO
```

```
INSERT INTO TestTran VALUES (3, 'ccc')
GO
COMMIT TRANSACTION Inner2 -- Decrements @@TRANCOUNT to 2.
-- Nothing committed.
GO
COMMIT TRANSACTION Inner1 -- Decrements @@TRANCOUNT to 1.
-- Nothing committed.
GO
COMMIT TRANSACTION OuterTran -- Decrements @@TRANCOUNT to 0.
-- Commits outer transaction OuterTran.
GO
```

Hủy bỏ và quay lại phiên giao dịch.

Sử dụng lệnh RollBack Transaction hủy bỏ những thực hiện và quay lại phiên giao dịch.

ROLLBACK [TRAN [SACTION]
[*transaction_name* | *@tran_name_variable*
| *savepoint_name* | *@savepoint_variable*]]

LOCK – KHÓA.

Khi 2 hay nhiều người cùng truy nhập đồng thời một CSDL, SQL Server sử dụng khoá để xác định hoạt động cho một người và không xác didnhj cho người khác. Khoá là việc ngăn không cho những người đọc dữ liệu mà không bị người khác sử đổi.

Hầu hết SQL Server đều khoá tự động, bạn có thể thiết tiết CSDL một cách có hiệu quả hơn bằng việc tìm hiểu về khoá và chọn khoá cho ứng dụng của bạn.

Tìm hiểu về khoá.

Khoá gồm các loại sau:

Kiểu khoá	Mô tả
Shared	Là khoá không làm thay đổi, ghi dữ liệu, dùng cho lệnh Select
Update	Khoá hoặc cho phép sửa đổi dữ liệu
Exclusive	Khoá với các thao tác Update, Insert, Delete

Một số phạm vi khoá như sau:

Tên	Mô tả
Page	Trang dữ liệu 2K hoặc trang chỉ mục Index, thường được dùng
Extent	Nhóm các trang có kích thước 8k, chỉ dùng với trường hợp xác định
Table	Cả bảng dữ liệu, gồm dữ liệu và index
Intent	Là kiểu đặc biệt để đặt kiểu khoá của trang hiện tại trên bảng

Bảng xác định hiệu lực của các kiểu khoá

	Shared	Update	Exclusive
Shared	Yes	Yes	No
Update	Yes	No	No
Exclusive	No	No	No

Ví dụ: Khi đặt chế độ khoá là Exclusive thì những phiên giao dịch khác không thể yêu cầu bất cứ loại khoá nào đến khi hoá Exclusive bị bỏ.

Xem thông tin về khoá.

Để xem thông tin về khoá đạng sử dụng trong SQL Server ta làm như sau:

- Chọn đối tượng cần xem khoá
- Thực hiện thủ tục sp_lock

Chọn kiểu khoá.

Khoá đươc đặt trong các câu lệnh như: SELECT, INSERT, UPDATE, và DELETE , sau đây là bảng mô tả các kiểu khoá đối với phương thức nói trên

Tên	Mô tả
NOLOCK	Được sử dụng với câ lệnh Select, người đoc có thể đọc dữ liệu khi dữ liệu gốc khi chưa được ghi dữ liệu mới trong giao dịch đang sử dụng
HOLDLOCK	Khoa Shared được giữ đến khi phiên giao dịch được hoàn tất khi khoá chưa được giải phóng
UPDLOCK	Dùng để cập nhật dữ liệu của kiểu khoá Shared trong quá trình đọc bảng dữ liệu và được giữ đến khi kết thúc lệnh của phiên giao dịch. Khoa này dùng khi cập nhật dữ liệu, ngăn không cho người khác đọc đến khi phiên giao dịch cập nhật được hoàn tất
TABLOCK	Dùng khoá Shared trên một bảng dữ liệu, cho phép những người khác đọc dữ liệu nhưng ngăn không cho cập nhật
PAGELOCK	sử dụng kiểu khoá Shared phạm vi trang dữ liệu (Page), đây là

	loại khoá ngầm định
TABLOCKX	Dùng kiểu khoá Exlusive trên một bảng dữ liệu, ngăn người khác đọc và cập nhật dữ liệu từ bảng và giữ đến khi kết thúc lệnh, phiên giao dịch

- Cách đặt khoá như sau: Dùng lệnh SET
- Ví dụ 1:

```
USE pubs
GO
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE
GO
BEGIN TRANSACTION
SELECT au_lname FROM authors WITH (NOLOCK)
GO
```

- Vớ d? 2:

```
Select * from authors(UPDLOCK)
```

Đặt mức khoá.

Dùng đặt, điều khiển khoá trong các giao dịch của SQL Server

- Cú pháp:

```
SET TRANSACTION ISOLATION LEVEL
  { READ COMMITTED
    | READ UNCOMMITTED
    | REPEATABLE READ
    | SERIALIZABLE
  }
```

Trong đó:

- Read Committed: Dùng kiểu khoá Shared trong quá trình đọc dữ liệu
- Read Uncommitted: Không đặt khoá Shared và khoá Exclusive, có thể đọc dữ liệu gốc khi đang có phiên giao dịch sửa đổi dữ liệu
- RepeateTable Read: Khoá tất cả dữ liệu đang được sử dụng trong truy vấn, ngăn những người khác sửa dữ liệu nhưng người khác có thể chèn thêm dữ liệu mới vào bảng (hàng mới)
- Serializable: Đặt khoá trong một tập dữ liệu (khoá phạm vi) ngăn không cho người khác có thể sửa, thêm hàng mới vào tập dữ liệu đến khi giao dịch kết thúc, tương ự như HoldLock trong lệnh Select

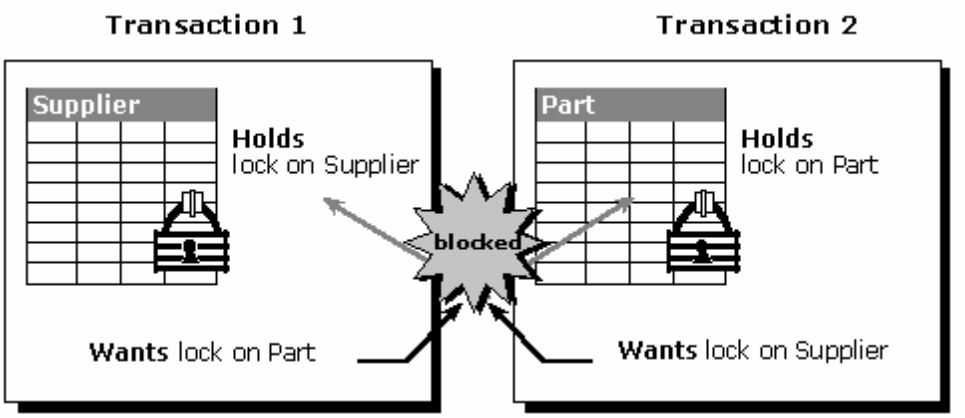
- Ví dụ:

```
SET TRANSACTION ISOLATION LEVEL REPEATABLE READ
```

```
GO
BEGIN TRANSACTION
SELECT * FROM publishers
SELECT * FROM authors
...
COMMIT TRANSACTION
```

Khoá chết (DeadLock).

Trong hệ quản trị CSDL quan hệ nói rieneg và các hệ quản trị khác nói chung, việc xuất hiện nhiều luồng dữ liệu đồng thời trong CSDL là thường xuyên xảy ra, một giao dịch có thể lấy dữ liệu từ nhiều nguồn dữ liệu khác nhau, hai giao dịch trong cùng CSDL có thể cùng chung một nguồn dữ liệu nào đó nên việc các giao dịch này đặt các mức khoá khác nhau cho các nguồn dữ liệu mà no năm giữ là không thể tránh khỏi, ví dụ tên sơ đồ sau mô tả sự giao chéo về nguồn dữ liệu trong giao dịch



Trong giao dịch 1 và 2 đều đặt các bảng dữ liệu ở mức khoá Exlusive, như vậy giao dịch 1 chỉ thực hiện được khi giao dịch 2 thực hiện xong hoặc quay lại trạng thái ban đầu, ngược lại giao dịch 2 cũng chừo giao dịch 1 thực hiện xong hoặc quay lại trạng thai ban đầu. Cứ như vây thì cả 2 giao dịch sẽ không bao giơ kết thúc được phiên giao dịch của mình. Phần chung của khoá nói trên gọi là khoá chết, và được khoá theo khối (block).

GRAND – GÁN QUYỀN.

Lệnh Grand thực hiện gán quyền cho user hoặc role của SQL Server. Người thực hiện Grand phải có quyền được thực hiện phân quyền cho user. Có 2 hình thức gán quyền: gán quyền thực hiện câu lệnh, gán quyền thao tác với đối tượng.

Gán quyền thao tác câu lệnh.

GRANT { ALL | *statement* [**,**...*n*] }
TO *security_account* [**,**...*n*]

Các câu lệnh:

- CREATE DATABASE
- CREATE DEFAULT
- CREATE FUNCTION
- CREATE PROCEDURE
- CREATE RULE
- CREATE TABLE
- CREATE VIEW
- BACKUP DATABASE
- BACKUP LOG

SQL Server ngầm định một số nhóm có quyền thực hiện câu lệnh như sau:

	dbcreator	**processadmin**	**securityadmin**	**serveradmin**	**bulkadmin**
ALTER DATABASE	X				
CREATE DATABASE	X				
BULK INSERT					X
DBCC				X (1)	
DENY			X (2)		
GRANT			X (2)		
KILL		X			
RECONFIGURE				X	
RESTORE	X				
REVOKE			X (2)		
SHUTDOWN				X	

	db_ owner	db_ datareader	db_ datawriter	db_ ddladmin	db_ backup operator	db_ security admin
ALTER DATABASE	X			X		
ALTER FUNCTION	X			X		
ALTER PROCEDURE	X			X		
ALTER TABLE	X (1)			X		
ALTER TRIGGER	X			X		
ALTER VIEW	X (1)			X		
BACKUP	X				X	
CHECKPOINT	X				X	
CREATE DEFAULT	X			X		
CREATE FUNCTION	X			X		
CREATE INDEX	X (1)			X		
CREATE PROCEDURE	X			X		
CREATE RULE	X			X		
CREATE TABLE	X			X		
CREATE TRIGGER	X (1)			X		
CREATE VIEW	X			X		
DBCC	X				X (2)	
DELETE	X (1)		X			
DENY	X					X
DENY on object	X					
DROP	X (1)			X		
EXECUTE	X (1)					
GRANT	X					X
GRANT on object	X (1)					
INSERT	X (1)		X			
READTEXT	X (1)	X				
REFERENCES	X (1)			X		
RESTORE	X					
REVOKE	X					X
REVOKE on object	X (1)					

SELECT	X (1)	X				
SETUSER	X					
TRUNCATE TABLE	X (1)			X		
UPDATE	X (1)		X			
UPDATE STATISTICS	X (1)					
UPDATETEXT	X (1)		X			
WRITETEXT	X (1)		X			

Các user được gán quyền có thể là user của SQL Server hoặc user của Windows NT.

Ví dụ gán quyền thao tác câu lệnh cho 3 user (trong đó có 2 user của SQL Server và 1 user của Windows NT):

```
GRANT CREATE DATABASE, CREATE TABLE
TO Mary, John, [Corporate\BobJ]
```

Ví dụ gán quyền thao tác cho role và user:

```
USE pubs
GO

GRANT SELECT
ON authors
TO public
GO

GRANT INSERT, UPDATE, DELETE
ON authors
TO Mary, John, Tom
GO
```

Gán quyền thao tác đối tượng.

Là việc gán quyền cho các user hoặc role có quyền thao tác với các đối tượng của SQL Server.

Ví dụ gán quyền thao tác cho Role:

```
GRANT CREATE TABLE TO Accounting
```

Ví dụ gán quyền để gán quyền thao tác cho user khác: Ví dụ Jean là dbo của bảng Plan_data, Jean thực hiện gán quyền với chức năng GRAND_OPTION cho role accounting, Jill thuộc role nói trên và Jill gán quyền được Select cho Jack, Jack không là thành viên của Accounting.

```
/* User Jean */
GRANT SELECT ON Plan_Data TO Accounting WITH GRANT
OPTION

/* User Jill */
GRANT SELECT ON Plan_Data TO Jack AS Accounting
```

Thủ cục sp_grandlogin.

Là thủ tục thực hiện gán quyền truy nhập cho user của Windows NT hoặc nhóm user của Windows NT.

sp_grantlogin [**@loginame** =] '*login*'

Ví dụ gán quyền truy nhập SQL Server cho BobJ.

```
EXEC sp_grantlogin 'Corporate\BobJ'
```

Thủ tục sp_grandaccess.

Gán quyền khai thác cho user của SQL Server hoặc Windows NT.

sp_grantdbaccess [**@loginame** =] '*login*'
[,[**@name_in_db** =] '*name_in_db*' [**OUTPUT**]]

Ví dụ gán quyền khai thác cho user của Windows và lấy theo tên mới.

```
EXEC sp_grantdbaccess 'Corporate\GeorgeW', 'Georgie'
```

REVOKE – TƯỚC QUYỀN.

Revoke là câu lệnh tước quyền khai thác của user.

Tước quyền được thực hiện câu lệnh.

REVOKE { ALL | *statement* [**,***...n*] }
FROM *security_account* [**,***...n*]

Ví dụ tước quyền khai thác với 2 user:

```
REVOKE CREATE TABLE FROM Joe, [Corporate\BobJ]
```

Ví dụ tước quyền khai thác 2 câu lệnh với các user:

```
REVOKE CREATE TABLE, CREATE DEFAULT
FROM Mary, John
```

Tước quyền khai thác của user với đối tượng.

Ví dụ tước quyền thực hiện lệnh Select trong role Budget_data đối với Mary:

```
REVOKE SELECT ON Budget_Data TO Mary
```

DENY – TỪ CHỐI QUYỀN.

Là câu lệnh từ chối quyền đối với user, user chỉ thực hiện được quyền khi có chỉ định rõ ràng.

Ví dụ từ chối quyền thực hiện lệnh với các user:

```
DENY CREATE DATABASE, CREATE TABLE
TO Mary, John, [Corporate\BobJ]
```

Ví dụ gán quyền khai thác cho role, sau đó thực hiện từ chối thực hiện của các user trong role:

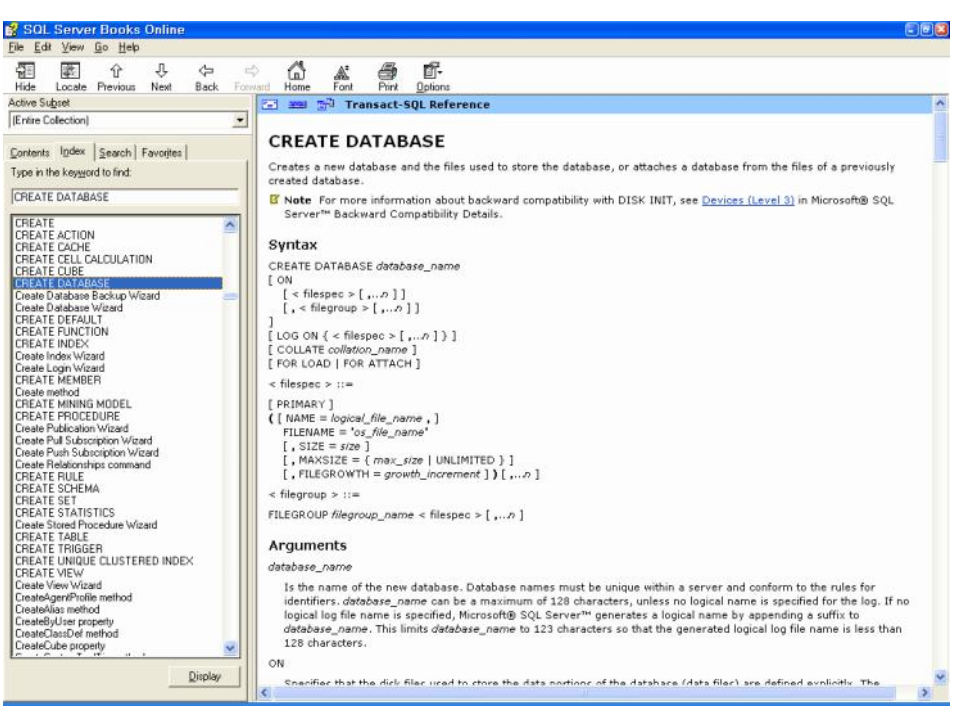
```
USE pubs
GO

GRANT SELECT
ON authors
TO public
GO

DENY SELECT, INSERT, UPDATE, DELETE
ON authors
TO Mary, John, Tom
```

Ví dụ từ chối quyền của role:

```
DENY CREATE TABLE TO Accounting
```

TRỢ GIÚP.

Trong quá trình thực hiện soạn lệnh T-SQL bạn có thể thực hiện tra cứu lệnh trong Book Online.



Phần 3. PHÁT TRIỂN ỨNG DỤNG VỚI SQL SERVER

Trong phần này ta sẽ xem xét kỹ thuật phát triển ứng dụng với SQL Server từ các ngôn ngữ lập trình (Visual Basic, C++, VBScript,…). Các ứng dụng khai thác CSDL của SQL Server thực hiện các bước sau:

+ Kết nối từ ứng dụng đến SQL Server.

+ Xây dựng cơ sở dữ liệu.

+ Thực hiện các lệnh khai thác hoặc thủ tục của SQL Server.

+ Khai thác dữ liệu thông qua công cụ có sẵn.

+ Ngắt kết nối.

GIỚI THIỆU.

Thiết kế ứng dụng là việc thực hiện tạo giao diện (API – Application Program Interface) giao tiếp với SQL Server, việc thực hiện kết nối thực hiện thông qua các công cụ ADO, URL, OLE DB, ODBC, Embedded SQL for C, DB-Library. Khi sử dụng các công cụ kết nối dữ liệu thao tác dưới dạng bảng hoặc dạng tài liệu XML.

+ Dữ liệu dưới dạng bảng được thực hiện thông qua các công cụ kết nối ADO, OLE DB, ODBC, Embedded SQL for C, DB-Library.

+ Dữ liệu thực hiện thông qua tài liệu XML thông qua các công cụ ADO, URL, OLE DB.

KẾT NỐI VỚI SQL SERVER BẰNG ADO.

ADO viết tắt của cụm từ ActiveX Data Object là công cụ giao tiếp với dữ liệu của nhiều hệ quản trị CSDL khác nhau, SQL Server là một ví dụ cho việc giao tiếp.

ADO sử dụng với CSDL quan hệ hoặc sử dụng với CSDL đa chiều, khi đó gọi là ADO MD (ADO Multi Dimention). ADO sử dụng kết nối kiểu OLE DB hoặc các thư viện kết nối COM (Component Object Model).

OLE DB sử dụng 2 phương thức Microsoft OLE DB Provider for SQL Server (SQLOLEDB) và Microsoft OLE DB Provider for ODBC (MSDASQL).

ADO có thể thực hiện từ các ngôn ngữ lập trình Visual Basic, ASP, C++.

Cấu trúc ứng dụng sử dụng ADO.

ADO gồm các thành phần cơ bản sau: Application, ADO, OLE DB Provider, Data Source.

Thành phần	Chức năng
Application	Gọi các đối tượng, thành phần, phương thức và các thuộc tính của ADO. Thông qua các thành phần này ứng dụng sẽ gửi các câu lệnh SQL và nhận kết quả xử lý.
ADO	Quản lý việc trao đổi dữ liệu giữa ứng dụng và OLE DB
OLE DB provider	Xử lý các lệnh gọi từ ứng dụng qua ADO, kết nối với Data Source. Processes all ADO calls from the application, connects to a data source, passes SQL statements from the application to the data source, and returns results to the application.
Data source	Contains the information used by a provider to access a specific instance of data in a DBMS.

Khi thực hiện lập trình ứng dụng với SQL Server sử dụng ADO, người lập trình phải thực hiện các thao tác sau:

+ Kết nối đến nguồn dữ liệu (data source).

+ Gửi câu lệnh SQL đến nguồn dữ liệu.

+ Xử lý kết quả nhận được từ câu lệnh đã gửi.

+ Xử lý các lỗi và thông báo.

+ Ngắt kết nối đến nguồn dữ liệu.

Đối với một số ứng dụng phức tạp sử dụng ADO có thể sử dụng một số thao tác sau:

+ Sử dụng con trỏ (cursor) để điều khiển vị trí trong tập kết quả.

+ Thực hiện thủ tục lưu trữ trên Server.

+ Thực hiện hàm tự định nghĩa trên Server.

+ Quản lý các phép truy vấn mà có nhiều tập kết quả.

+ Yêu cầu kết thúc hoặc lặp lại một phiên giao dịch.

+ Quản lý các thao tác với kiểu dữ liệu lớn (text, image).

+ Thực hiện các thao tác với XML sử dụng phép truy vấn XPath.

Kết nối đến SQL Server.

Để kết nối đến SQL Server, các công việc cơ bản cần thực hiện như sau:

+ Cấu hình kết nối.

+ Thiết lập hoặc ngắt kết nối đến nguồn dữ liệu.

+ Xác định OLE DB provider.

+ Thực hiện truy vấn.

+ Quản lý các phiên làm việc trên kết nối.

Khi sử dụng SQLOLEDB ta phải thực hiện đặt các thuộc tính sau cho kết nối:

+ **Initial Catalog:** Xác định CSDL.

+ **Data Source:** Xác định tên Server.

+ **Integrated Security**: Xác định chế độ xác thực, nếu là **SSPI chế độ xác thực là** Windows Authentication, hoặc xác định **User ID, Password** của chế độ xác thực SQL Server Authentication.

Ví dụ thực hiện kết nối đến SQL Server đặt từng thuộc tính tiêng biệt từ Visual Basic:

```
' Initialize variables.
Dim cn As New ADODB.Connection
. . .
Dim ServerName As String, DatabaseName As String, _
   UserName As String, Password As String

' Put text box values into connection variables.
ServerName = txtServerName.Text
DatabaseName = txtDatabaseName.Text
```

```
UserName = txtUserName.Text
Password = txtPassword.Text

' Specify the OLE DB provider.
cn.Provider = "sqloledb"

' Set SQLOLEDB connection properties.
cn.Properties("Data Source").Value = ServerName
cn.Properties("Initial Catalog").Value = DatabaseName

' Decision code for login authorization type:
' Windows NT or SQL Server authentication.
If optWinNTAuth.Value = True Then
    cn.Properties("Integrated Security").Value = "SSPI"
Else
    cn.Properties("User ID").Value = UserName
    cn.Properties("Password").Value = Password
End If

' Open the database.
cn.Open
```

Ví dụ kết nối đến SQL Server sử dụng chuỗi kết nối:

```
' Initialize variables.
Dim cn As New ADODB.Connection
Dim provStr As String

' Specify the OLE DB provider.
cn.Provider = "sqloledb"

' Specify connection string on Open method.
ProvStr =
"Server=MyServer;Database=northwind;Trusted_Connection=
yes"
cn.Open provStr
```

Ví dụ kết nối sử dụng ODBC:

```
Dim cn As New ADODB.Connection

cn.ConnectionTimeout = 100
' DSN connection. You can use variables for the
parameters.
cn.Open "MyDataSource", "sa", "MyPassword"
' Alternative syntax follows:
' cn.Open "DSN=DataSourceName;UID=sa;PWD=Password;"

cn.Close
```

Ví dụ kết nối xác định Driver của SQL Server:

```
Dim cn As New ADODB.Connection

' Connection to SQL Server without using ODBC data
source.
cn.Open "Driver={SQL
Server};Server=Server1;Uid=SA;Pwd=;Database=northwind"

cn.Close
```

Thực hiện truy vấn.

Thực hiện truy vấn sử dụng đối tượng Command.

cmd.Execute(*NumRecords, Parameters, Options***)**

Đối tượng Command có thể thực hiện nhiều kiểu câu lệnh (Select, Update, Insert, Delete, Create, Drop), đối với lệnh Select kết quả thực hiện là một recordset.

Set *rs* = cmd.Execute(*NumRecords, Parameters, Options***)**

Kiểu lệnh thực hiện trong Command được xác định theo option của lệnh, gồm một số kiểu sau:

Tên kiểu	Mô tả
adCmdFile	Tên file chứa đối tượng recordset

adCmdStoreProc	Stored procedure
adCmdTable	Tên bảng
adCmdTableDirect	Tên bảng mà các cột được truy vấn
adCmdText	Câu lệnh SQL
adCmdUnknown	Chưa xác định
adCmdUnspecified	Chưa xác định tham số cho lệnh

Thực hiện truy vấn thông qua đối tượng connection.

```
Dim cn As New ADODB.Connection
. . .
Dim rs As New ADODB.Recordset

cmd1 = txtQuery.Text
Set rs = cn.Execute(cmd1)
```

Thực hiện truy vấn có sử dụng tham số.

Khi thực hiện các thủ tục có tham số truyền vào các ứng dụng phải truyền tham số, trong phần này sẽ giới thiệu một ví dụ sử dụng đối tượng parameter.

Tạo thủ tục:

```
USE NORTHWIND
GO
drop proc myADOParaProc
GO
CREATE PROC myADOParaProc
@categoryid int(4)
AS
SELECT * FROM products WHERE categoryid = @categoryid
GO
```

Sử dụng đối tượng parameter truyền tham số là số nguyên xác định categoryID:

```
Dim cn As New ADODB.Connection
Dim cmd As New ADODB.Command
Dim rs As New ADODB.Recordset
Dim prm As ADODB.Parameter
```

```
Dim fld As ADODB.Field
Dim provStr As String

' Connect using the SQLOLEDB provider.
cn.Provider = "sqloledb"

' Specify connection string on Open method.
provStr =
"Server=MyServer;Database=northwind;Trusted_Connection=yes"
cn.Open provStr

' Set up a command object for the stored procedure.
Set cmd.ActiveConnection = cn
cmd.CommandText = "myADOParaProc"
cmd.CommandType = adCmdStoredProc
cmd.CommandTimeout = 15

' Set up a new parameter for the stored procedure.
Set prm = Cmd.CreateParameter("CategoryID", adInteger,
adParamInput, 4, 7)
Cmd.Parameters.Append prm

' Create a recordset by executing the command.
Set rs = cmd.Execute
Set Flds = rs.Fields

' Print the values for all rows in the result set.
While (Not rs.EOF)
   For Each fld in Flds
      Debug.Print fld.Value
   Next
   Debug.Print ""
    rs.MoveNext
Wend

' Close recordset and connection.
rs.Close

cn.Close
```

Đối tượng Recordset.

Sử dụng đối tượng Recordset lưu trữ kết quả của lệnh Select.

```
Dim cn As New ADODB.Connection
Dim rs As ADODB.Recordset
. . .
cmd1 = txtQuery.Text
Set rs = New ADODB.Recordset
rs.Open cmd1, cn
rs.MoveFirst
. . .
' Code to loop through result set(s)
```

Đối tượng Field.

Sử dụng đối tượng field là các cột của Recordset, thông qua nó ta có thể láy giá trị, thuộc tính của cột.

```
Dim rs As New ADODB.Recordset
Dim fld As ADODB.Field
Dim cn As ADODB.Connection
Dim cmdText As String

cn.Provider = "sqloledb"
cn.Properties("Data Source").Value = "MyServerName"
cn.Properties("Initial Catalog").Value = "northwind"
cn.Properties("Integrated Security").Value = "SSPI"
cn.Open

cmdText = "select * from authors"

rs.Open cmdText, cn
Set Flds = rs.Fields
Dim TotalCount As Integer
TotalCount = Flds.Count

For Each fld In Flds
    Debug.Print fld.Name
    Debug.Print fld.Type
    Debug.Print fld.Value
Next
rs.Close
```

Sử dụng con trỏ.

Khi sử dụng đối tượng Recordset của ADO, ta có thể sử dụng nhiều kiểu con trỏ khác nhau xác định kiểu khóa, điều khiển vị trí,...

```
Dim rs As New ADODB.Recordset
. . .
rs.Open "SELECT * FROM titles", , adOpenDynamic,
adLockOptimistic

rs.Close
```

COn trỏ nói trên gồm những thuộc tính cơ bản sau: **CursorType**, **CursorLocation**, **LockType**, **CacheSize**.

Thuộc tính	Mô tả
CursorType	- **adOpenForwardOnly:** Ngầm định Xác định kiểu con trỏ được sử dụng: - **adOpenForwardOnly:** Chỉ đọc, chỉ có thể cập nhật dữ liệu trên hàng dữ liệu hiện thời. - **adOpenStatic:** Trạng thái tĩnh, khi mở kiểu này hệ thống sẽ cung cấp một ảnh dữ liệu (snapshot), dữ liệu thay đổi trên bảng cơ sở sẽ không được thể hiện trên snapshot dạng này. - **adOpenKeyset:** Theo vị trí tùy chọn, khi di chuyển hàng cập nhật con trỏ kiểu này sẽ chiếu đến hàng dữ liệu cơ sở, hàng dữ liệu được khó và bạn có thể cập nhật, lấy dữ liệu từ hàng cơ sở**.** - **adOpenDynamic:** Động, con trỏ kiểu này gần giống keyset cursor , nhưng con trỏ kiểu này phản ảnh những thay đổi trên bảng cơ sở.
CursorLocation	-**adUseServer:** Ngầm định. - **adUseClient:** Nếu đặt là ta chỉ có thể mở ở trạng thái tĩnh.
LockType	-**adLockReadOnly:** Ngầm định. Xác định kiểu khóa trong quá trình cập nhật dữ liệu (**adLockPessimistic**, **adLockOptimistic**, **adLockBatchOptimistic**).
CacheSize	Ngầm định: 1 Xác định số hàng đặt trong bộ đệm hoặc đọc trong một thời điểm.

Các phương thức dịch chuyển hàng dữ liệu.

Khi sử dụng đối tượng Recordset bạn có thể dịch chuyển vị trí của hàng dữ liệu bằng các phương thức **MoveFirst**, **MoveLast**, **MoveNext**, **MovePrevious.** Đánh dấu vị trí theo phương thức **Bookmark,** phương thức **clone** để tạo một bản sao recordset.

Quản lý phiên làm việc.

Trong phần câu lệnh T-SQL ta đã xem xét việc điều khiển một hpiên làm việc (transaction), tuy nhiên ta có thể sử dụng đối tượng connection của ADO để điều khiển trực tiếp phiên làm việc như trong kịch bản lệnh nói trên bằng việc sử dụng các phương thức **BeginTrans**, **CommitTrans**, **RollbackTrans**. Xét ví dụ sau:

```
Dim cn As New ADODB.Connection
Dim rs As New ADODB.Recordset

. . .
' Open connection.
cn.Open

' Open titles table.
rs.Open "SELECT * FROM titles", Cn, adOpenDynamic,
adLockPessimistic
. . .
' Begin the transaction.
rs.MoveFirst
cn.BeginTrans

' User loops through the recordset making changes.
. . .
' Ask if the user wants to commit all the changes made.
If MsgBox("Save all changes?", vbYesNo) = vbYes Then
   cn.CommitTrans
Else
   cn.RollbackTrans
End If
```

Thực hiện các lệnh DDL.

Để thực hiện các lệnh DDL như CREATE TABLE, DROP TABLE, ALTER TABLE. Bạn có thể sử dụng đối tượng command của ADO, xét ví dụ sau:

```
Dim Cn As New ADODB.Connection
```

```
Dim Cmd As New ADODB.Command

' If the ADOTestTable does not exist, go to AdoError.
On Error GoTo AdoError

' Connect using the SQLOLEDB provider.
cn.Provider = "sqloledb"
cn.Properties("Data Source").Value = "MyServerName"
cn.Properties("Initial Catalog").Value = "northwind"
cn.Properties("Integrated Security").Value = "SSPI"
cn.Open

' Set up command object.
Set Cmd.ActiveConnection = Cn
Cmd.CommandText = "DROP TABLE ADOTestTable"
Cmd.CommandType = adCmdText
Cmd.Execute

Done:
    Cmd.CommandText = "SET NOCOUNT ON"
    Cmd.Execute
    Cmd.CommandText = "CREATE TABLE ADOTestTable (id
int, name char(100))"
    Cmd.Execute
    Cmd.CommandText = "INSERT INTO ADOTestTable
values(1, 'Jane Doe')"
    Cmd.Execute
    Cn.Close
Exit Sub

AdoError:
      Dim errLoop As Error
      Dim strError As String

      ' Enumerate Errors collection and display
properties of
      ' each Error object.
      Set Errs1 = Cn.Errors
      For Each errLoop In Errs1
            Debug.Print errLoop.SQLState
            Debug.Print errLoop.NativeError
            Debug.Print errLoop.Description
```

```
      Next

      GoTo Done

End Sub
```

Quản lý dữ liệu kiểu lớn – Text, image.

Dữ liệu kiểu text, ntext, image là kiểu dữ liệu phức tạp, việc quản lý, khai thác không được thực hiện thông thường, ADO hỗ trợ các phương thức riêng để thực hiện.

Thay vì độc, cập nhật dữ liệu trực tiếp thì dữ liệu kiểu này được thao tác theo đoạn (chunk) bằng cách sử dụng các phương thức **AppendChunk**, **GetChunk**.

Trước khi thực hiện bạn phải đặt tham số bằng cách thực hiện lệnh sau:

```
EXEC sp_dboption 'pubs', 'Select into/bulkcopy', 'True'
```

Xét ví dụ sau trên CSDL Pubs:

- Copy bảng pub_info sang bảng mới

```
USE pubs
SELECT * INTO pub_info_x
   FROM pub_info
GO
```

- Thực hiện chèn dữ liệu vào bảng:

```
Public Sub AppendChunkX()

   Dim cn As ADODB.Connection
   Dim rstPubInfo As ADODB.Recordset
   Dim strCn As String
   Dim strPubID As String
   Dim strPRInfo As String
   Dim lngOffset As Long
   Dim lngLogoSize As Long
   Dim varLogo As Variant
   Dim varChunk As Variant

   Const conChunkSize = 100
```

```
   ' Open a connection.
   Set cn = New ADODB.Connection
   strCn = "Server=srv;Database=pubs;UID=sa;Pwd=;"

   cn.Provider = "sqloledb"
   cn.Open strCn

   'Open the pub_info_x table.
   Set rstPubInfo = New ADODB.Recordset
   rstPubInfo.CursorType = adOpenDynamic
   rstPubInfo.LockType = adLockOptimistic
   rstPubInfo.Open "pub_info_x", cn, , , adCmdTable

   'Prompt for a logo to copy.
   strMsg = "Available logos are : " & vbCr & vbCr

   Do While Not rstPubInfo.EOF
      strMsg = strMsg & rstPubInfo!pub_id & vbCr & _
        Left(rstPubInfo!pr_info,
         InStr(rstPubInfo!pr_info, ",") - 1) & vbCr &
vbCr
      rstPubInfo.MoveNext
   Loop

   strMsg = strMsg & "Enter the ID of a logo to copy:"
   strPubID = InputBox(strMsg)

   ' Copy the logo to a variable in chunks.
   rstPubInfo.Filter = "pub_id = '" & strPubID & "'"
   lngLogoSize = rstPubInfo!logo.ActualSize
   Do While lngOffset < lngLogoSize
      varChunk = rstPubInfo!logo.GetChunk(conChunkSize)
      varLogo = varLogo & varChunk
      lngOffset = lngOffset + conChunkSize
   Loop

   ' Get data from the user.
   strPubID = Trim(InputBox("Enter a new pub ID:"))
   strPRInfo = Trim(InputBox("Enter descriptive
text:"))
```

```
   ' Add a new record, copying the logo in chunks.
   rstPubInfo.AddNew
   rstPubInfo!pub_id = strPubID
   rstPubInfo!pr_info = strPRInfo
   lngOffset = 0   ' Reset offset.

   Do While lngOffset < lngLogoSize
      varChunk = LeftB(RightB(varLogo, lngLogoSize - _
        lngOffset),conChunkSize)
      rstPubInfo!logo.AppendChunk varChunk
      lngOffset = lngOffset + conChunkSize
   Loop

   rstPubInfo.Update

   ' Show the newly added data.
   MsgBox "New record: " & rstPubInfo!pub_id & vbCr & _
     "Description: " & rstPubInfo!pr_info & vbCr & _
     "Logo size: " & rstPubInfo!logo.ActualSize

   rstPubInfo.Close
   cn.Close

End Sub
```

Kết nối từ ASP.

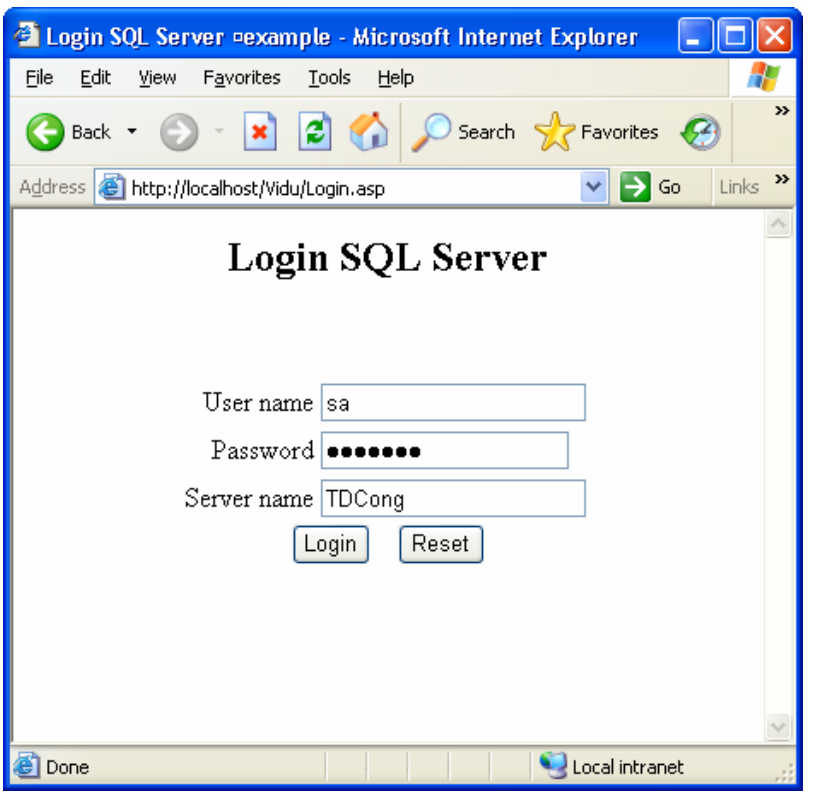
Trong ác ví dụ saiu đây thực hiện làm việc với SQL Server từ ASP, sử dụng ngôn ngữ lập trình VBScript, để làm được ví dụ này bạn đọc phải có kiến thức về thiết kế Web site (HTML).

Thiết kế form kết nối:

```
<%@LANGUAGE="VBSCRIPT" CODEPAGE="65001"%>
<html>
<head>
<title>Login SQL Server example</title>
<meta   http-equiv="Content-Type"   content="text/html;
charset=utf-8">
<style type="text/css">
<!--
```

```
.style1 {
    font-size: 24px;
    font-weight: bold;
}
-->
</style>
</head>

<body>
<p align="center" class="style1">Login SQL Server</p>
<p> </p>
<form name="frmlogin">
<table width="100%" border="0">
  <tr>
    <td    width="40%"><div    align="right">User    name
</div></td>
    <td width="60%">
      <input name="txtUser" type="text" id="txtUser">
    </td>
  </tr>
  <tr>
    <td><div align="right">Password </div></td>
    <td><input    name="txtPassword"    type="password"
id="txtPassword"></td>
  </tr>
  <tr>
    <td><div align="right">Server name </div></td>
    <td><input       name="txtServer"       type="text"
id="txtServer"></td>
  </tr>
</table>
  <div align="center">
    <input  name="cmdLogin"  type="button"  id="cmdLogin"
value="Login">
     
   <input type="reset" name="cmdReset" value="Reset">
  </div>
</form>
</body>
</html>
<Script language=vbscript>
sub cmdLogin_Onclick
```

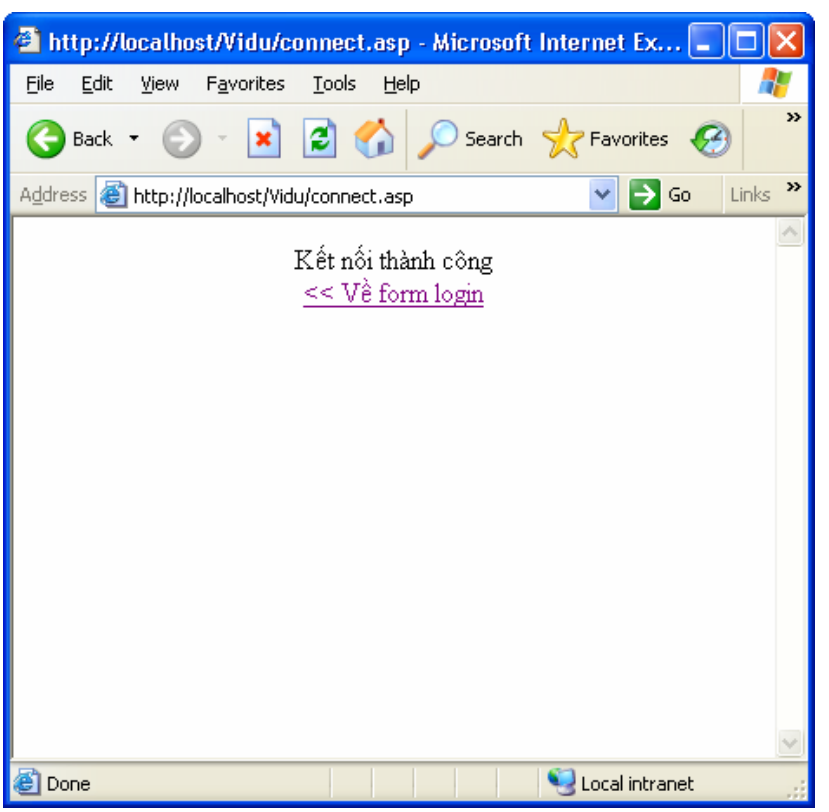
```
    frmlogin.method="post"
    frmlogin.action="connect.asp"
    frmlogin.submit
end sub
</script>
```



Tập tin connect.asp: Là tập tin được gọi từ form login.asp, thực hiện nhận tham số của form login.asp, kết nối đến SQL Server.

```
<%@LANGUAGE="VBSCRIPT" CODEPAGE="1252"%>
<%
dim username, password, servername, txt
    username=request.Form("txtUser")
    password = request.Form("txtPassword")
    servername=request.Form("txtServer")
    txt= "Provider=SQLOLEDB; "
    txt=txt & " Data Source=" & servername & ";"
    txt=txt & " Initial Catalog=pubs; "
    txt=txt & " User ID=" & username & ";"
    txt=txt & " PWD=" & password
```

```
    Set cn=Server.CreateObject("ADODB.Connection")
    cn.Open txt
    %>
<div    align="center"><span    class="style1">K&#7871;t
n&#7889;i th&agrave;nh c&ocirc;ng</span>
<%
    cn.close
%>
<br>
<a href="Login.asp<< V&#7873; form login</a>
</div>
```



Liệt kê danh sách.

Để liệt kê danh sách (có thể lấy dữ liệu bằng cách truy vấn trực tiếp hoặc thông qua khung nhìn - view) trước hết phải tạo một recordset lưu trữ kết quả truy vấn, từ recordset ta có thể lấy dữ liệu và đặt vào vị trí tương ứng cần thiết.

+ Khai báo Recordset:

```
    Set rs=Server.CreateObject("ADODB.Recordset")
    rs.ActiveConnection =cn
```

```
    rs.Source ="Select * from Authors"
    rs.Open
```

+ *Lấy giá trị*:

```
    Rs.fields(“au_id”)
```

+ *Đóng Recordset*:

```
    Rs.close
```

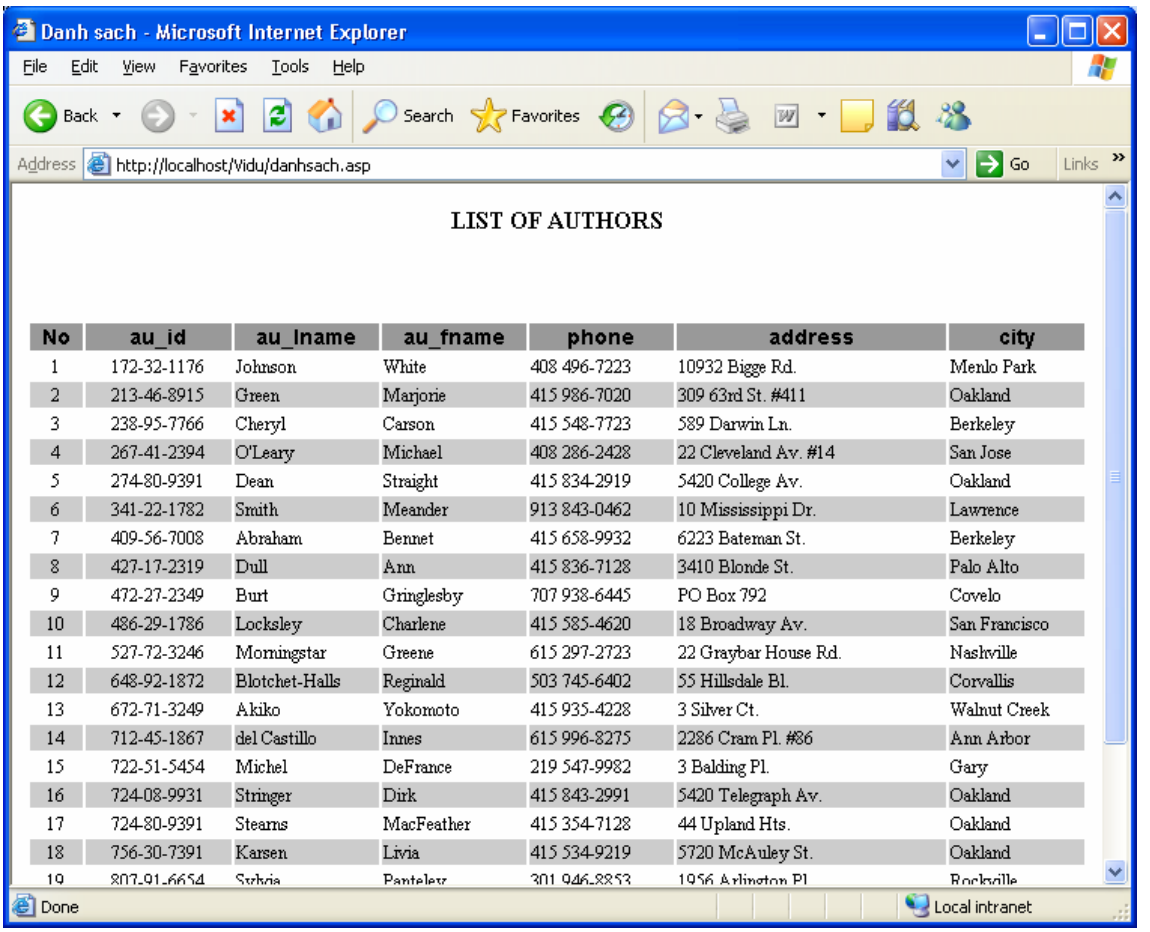
+ *Ví dụ liệt kê danh sách bằng cách truy vấn trực tiếp*:

```
<%@LANGUAGE="VBSCRIPT" CODEPAGE="65001"%>
<html>
<head>
<title>Danh sach</title>
<meta   http-equiv="Content-Type"   content="text/html;
charset=utf-8">
<style type="text/css">
<!--
.style5 {
    font-family: Arial, Helvetica, sans-serif;
    font-weight: bold;
    font-size: 14px;
}
.style6 {font-family: Arial, Helvetica, sans-serif}
.style7 {
    font-size: 14px;
    font-weight: bold;
}
.style9  {font-family:  Arial,  Helvetica,  sans-serif;
font-size: 14px; }
.style12 {font-size: 12px}
-->
</style>
</head>
<%
dim username, password, servername, txt
    username="sa"
```

```
    password = ""
    servername="TDCong"
    txt= "Provider=SQLOLEDB; "
    txt=txt & " Data Source=" & servername & ";"
    txt=txt & " Initial Catalog=pubs; "
    txt=txt & " User ID=" & username & ";"
    txt=txt & " PWD=" & password
    Set cn=Server.CreateObject("ADODB.Connection")
    cn.Open txt

    Set rs=Server.CreateObject("ADODB.Recordset")
    rs.ActiveConnection =cn
    rs.Source ="Select * from Authors"
    rs.Open
%>
<body>
<p align="center"><strong>LIST OF AUTHORS
</strong></p>
<p> </p>
<table width="100%"  border="0.2">
  <tr bgcolor="#999999">
    <td          width="5%"><div          align="center"
class="style5">No</div></td>
    <td  width="14%"><div  align="center"  class="style6
style7">au_id</div></td>
    <td          width="14%"><div          align="center"
class="style9"><strong>au_lname</strong></div></td>
    <td          width="14%"><div          align="center"
class="style9"><strong>au_fname</strong></div></td>
    <td          width="14%"><div          align="center"
class="style9"><strong>phone</strong></div></td>
    <td          width="26%"><div          align="center"
class="style9"><strong>address</strong></div></td>
    <td          width="13%"><div          align="center"
class="style9"><strong>city</strong></div></td>
  </tr>
  <%
    i=0
  do while not rs.eof and not rs.bof
     i=i+1
    if i mod 2<>0 then
  %>
```

```
  <tr bgcolor="#FFFFFF">
    <td><div                          align="center"><span
class="style12"><%=i%></span></div></td>
    <td><div                          align="center"><span
class="style12"><%=rs.fields("au_id")%></span></div></t
d>
    <td><span
class="style12"><%=rs.fields("au_fname")%></span></td>
    <td><span
class="style12"><%=rs.fields("au_lname")%></span></td>
    <td><span
class="style12"><%=rs.fields("Phone")%></span></td>
    <td><span
class="style12"><%=rs.fields("Address")%></span></td>
    <td><span
class="style12"><%=rs.fields("City")%></span></td>
  </tr>
  <%
   else
  %>
  <tr bgcolor="#CCCCCC">
    <td><div                          align="center"><span
class="style12"><%=i%></span></div></td>
    <td><div                          align="center"><span
class="style12"><%=rs.fields("au_id")%></span></div></t
d>
    <td><span
class="style12"><%=rs.fields("au_lname")%></span></td>
    <td><span
class="style12"><%=rs.fields("au_fname")%></span></td>
    <td><span
class="style12"><%=rs.fields("Phone")%></span></td>
    <td><span
class="style12"><%=rs.fields("Address")%></span></td>
    <td><span
class="style12"><%=rs.fields("City")%></span></td>
  </tr>
  <%
     end if
     rs.movenext
  loop
  %>
```

```
</table>
</body>
<%
     rs.close
%>
</html>
```



No	au_id	au_lname	au_fname	phone	address	city
1	172-32-1176	Johnson	White	408 496-7223	10932 Bigge Rd.	Menlo Park
2	213-46-8915	Green	Marjorie	415 986-7020	309 63rd St. #411	Oakland
3	238-95-7766	Cheryl	Carson	415 548-7723	589 Darwin Ln.	Berkeley
4	267-41-2394	O'Leary	Michael	408 286-2428	22 Cleveland Av. #14	San Jose
5	274-80-9391	Dean	Straight	415 834-2919	5420 College Av.	Oakland
6	341-22-1782	Smith	Meander	913 843-0462	10 Mississippi Dr.	Lawrence
7	409-56-7008	Abraham	Bennet	415 658-9932	6223 Bateman St.	Berkeley
8	427-17-2319	Dull	Ann	415 836-7128	3410 Blonde St.	Palo Alto
9	472-27-2349	Burt	Gringlesby	707 938-6445	PO Box 792	Covelo
10	486-29-1786	Locksley	Charlene	415 585-4620	18 Broadway Av.	San Francisco
11	527-72-3246	Morningstar	Greene	615 297-2723	22 Graybar House Rd.	Nashville
12	648-92-1872	Blotchet-Halls	Reginald	503 745-6402	55 Hillsdale Bl.	Corvallis
13	672-71-3249	Akiko	Yokomoto	415 935-4228	3 Silver Ct.	Walnut Creek
14	712-45-1867	del Castillo	Innes	615 996-8275	2286 Cram Pl. #86	Ann Arbor
15	722-51-5454	Michel	DeFrance	219 547-9982	3 Balding Pl.	Gary
16	724-08-9931	Stringer	Dirk	415 843-2991	5420 Telegraph Av.	Oakland
17	724-80-9391	Stearns	MacFeather	415 354-7128	44 Upland Hts.	Oakland
18	756-30-7391	Karsen	Livia	415 534-9219	5720 McAuley St.	Oakland
19	807-91-6654	Sylvia	Panteley	301 946-8853	1956 Arlington Pl	Rockville

KẾT NỐI VỚI SQL SERVER BẰNG SQL-DMO.

SQL DMO viết tắt của cụm từ SQL Distributed Management Objects, sử dụng thư viện liên kết động (dll) để kết nối đến SQL Server.

SQL DMO thực hiện liên kết nhúng (OLE Automation), các đối tượng SQL Server được thực hiện nhúng các đối tượng của SQL Server vào ứng dụng, khai thác các đối tượng thông qua thuộc tính, sự kiện và các phương thức làm việc của nó.

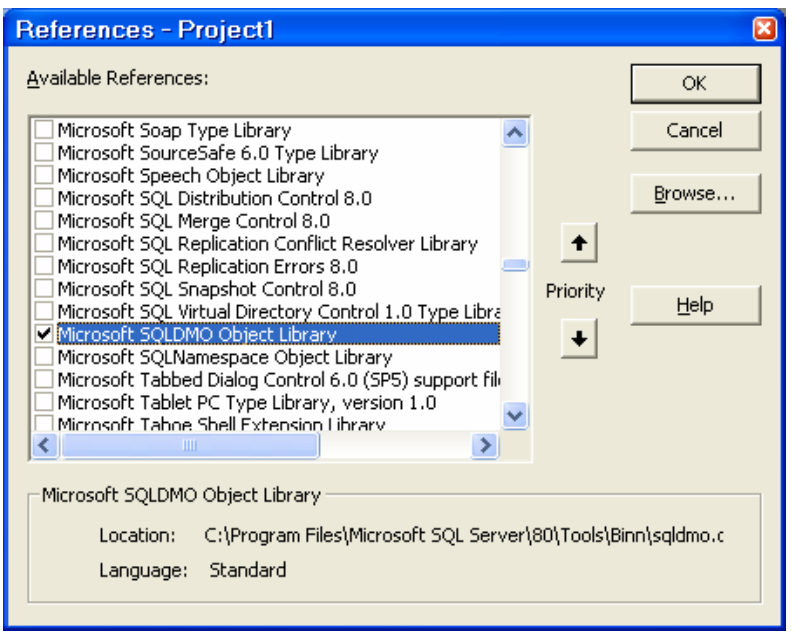
SQL DMO hỗ trợ phát triển ứng dụng từ ngôn ngữ lập trình Visual Basic, C++, khi đóng gói các thư viện liên kết động sẽ được đóng gói cùng, cài dặt ứng dụng thư viện sẽ được cài đặt trong Windows, nên khi chạy ứng dụng bạn không cần thiết lập môi trường Client Connectivity.

Các tập tin cơ bản cho SQL DMO: sqldmo.dll, sqldmo80.hlp, sqldmo.rll, sqldmo.h (C++), sqldmoid.h (C++), sqldmo.sql. Trong phần này sẽ giới thiệu kỹ thuật thiết kế ứng dụng từ Visual Basic 6.0.

Khai báo thư viện trong project.

- Vào menu Project -> References

- Chọn Microsoft SQL DMO Object Library -> Ok



Khai báo đối tượng.

Sau khi thực hiện khai báo thư viện trong project, ta có thể khai báo biến kiểu đối tượng (object) hoặc kiểu đối tượng của SQL DMO.

Ví dụ khai báo biến kiểu SQL Server:

```
Dim oSQLServer As SQLDMO.SQLServer
```

Kết nối đến SQL Server.

Để kết nối đến SQL Server ta sử dụng phương thức kết nối của đối tượng SQL Server, có 3 tham số Servername, LoginName, Pasword.

```
Dim oSQLServer As SQLDMO.SQLServer
    Set oSQLServer = New SQLDMO.SQLServer
    oSQLServer.Connect    "ServerName",    "LoginName",
"Pasword"
```

Thực hiện lại kết nối:

Trong nhiều trường hợp bạn muốn ngắt kết nối hiện tại và thực hiện lại kết nối để lấy trạng thái SQL Server hiện thời (tương tự động tác làm tươi – Reresh).

```
    oSQLServer.DisConnect

    oSQLServer.ReConnect
```

Làm việc với các đối tượng.

SQL DMO tạo đối tượng kế thừa từ những đối tượng con của nó, ví dụ SQL Server kế thừa từ các đối tượng Database <- Table <- Column,…

Xác định biến với CSDL:

```
Dim oDatabase as new SQLDMO.Database
Set oDatabase = oSQLServer.Databases("Northwind")
```

Lấy danh sách tên các CSDL vào hộp thoại:

```
Dim nDatabase as Integer
For nDatabase = 1 to oSQLServer.Databases.Count
    Combo1.AddItem oSQLServer.Databases(nDatabase).Name
Next nDatabase
```

Các đối tượng đều được kế thà từ các đối tượng con, các đối tượng con tạo thành một tập hợp, tập hợp nói trên có thể thực hiện các phương thức Add, Remove,…với từng đối tượng.

Ví dụ remove bảng khỏi CSDL:

```
oServer.Databases("Northwind").Tables.Remove("Orders",
"anne")
```

Thực hiện lệnh SQL:

Các đối tượng (SQL Server, Database) có thể thực hiện các lệnh SQL thông qua các phương thức ExecuteImmediate và ExecuteWithResults.

Ví dụ thực hiện lệnh thao tác:

```
oSQLServer.ExecuteImmediate “Create Database Example”
```

Ví dụ thực hiện lệnh truy vấn:

```
Dim rs As QueryResults
    Set  rs  =  oDatabase.ExecuteWithResults("Select  *
from Authors")
```

Ví dụ lấy dữ liệu từ một truy vấn:

```
    For i = 1 To rs.Rows
        For j = 1 To rs.Columns
            MsgBox rs.GetColumnString(i, j)
        Next j
    Next i
```

Các phương thức thực hiện kết nối có thể hỗ trợ theo từng ngôn ngữ lập trình, hỗ trợ nhiều trong việc lập trình từ Visual Basic, ASP, C, C++. Bạn có thể tham khảo các ví dụ trong thư mục Sample của SQL Server. Các ví dụ sẽ đề cập nhiều ngôn ngữ lập trình, nhiều sự kiện khác nhau.

Bài giảng: HỆ CƠ SỞ DỮ LIỆU

Nội dung môn học

- *Chương 1*: Giới thiệu về CSDL
- *Chương 2*: Mô hình liên kết – thực thể
- *Chương 3*: Mô hình dữ liệu quan hệ
- *Chương 4*: Chuẩn hóa
- *Chương 5*: Chuyển đổi sơ đồ LKTT thành QH
- *Chương 6*: Thiết kế vật lý CSDL
- *Chương 7*: Ngôn ngữ SQL
- *Chương 8*: Thủ tục lưu trữ và trigger
- *Chương 9*: Bảo mật và quyền của người dùng

Tài liệu học tập

- ***Sách, giáo trình chính:***

Dương Tuấn Anh, Nguyễn Trung Trực - Hệ cơ sở dữ liệu – NXB đại học quốc gia TP Hồ Chí Minh

- ***Sách tham khảo:***

Trần Đắc Phiến - Giáo trình cơ sở dữ liệu - Trường ĐH Công nghiệp TP Hồ Chí Minh

Mục đích của môn học

- Cung cấp các kiến thức căn về cơ sở dữ liệu cho sinh viên
- Cung cấp các kiến thức về ngôn ngữ SQL
- Cung cấp các kiến thức về lập trình với SQL- Server 2000

Yêu cầu môn học

- Phần mềm SQL SERVER 2007
- Hiện diện trên 80%
- Nắm được nội dung lý thuyết
- Hoàn thành các bài tập thực hành
- Điểm giữa kỳ lớn hơn hoặc bằng 5

Cơ sở dữ liệu

- *Cơ sở dữ liệu quan hệ?*
  - Là một CSDL trong đó dữ liệu được tổ chức trong các bảng có mối quan hệ với nhau
- *Bảng?*
  - Là đối tượng được sử dụng tổ chức lưu trữ dữ liệu

Minh họa

Bảng KHOA

MAKHOA	TENKHOA	DIENTHOAI
DHT01	Khoa Toán cơ - Tin học	054822407
DHT02	Khoa Công nghệ thông tin	054826767
DHT03	Khoa Vật lý	
DHT04	Khoa Hoá học	
...	...	

Bảng LOP

MALOP	TENLOP	KHOA	HEDAOTAO	NAMNHAPHOC	SISO	MAKHOA
C24101	Toán K24	24	Chính quy	2000	5	DHT01
C24102	Tin K24	24	Chính quy	2000	8	DHT02
C24103	Lý K24	24	Chính quy	2000	7	DHT03
C24301	Sinh K24	24	Chính quy	2000	5	DHT05

Bảng SINHVIEN

MASV	HODEM	TEN	NGAYSINH	GIOTINH	NOISINH	MALOP
0241010001	Ngô Thị Nhật	Anh	Nov 27 1982	0	Quảng Ninh, Quảng Bình	C24101
0241010002	Nguyễn Thị Ngọc	Anh	Mar 21 1983	0	Tân Kỳ, Nghệ An	C24101
0241010003	Ngô Việt	Bắc	May 11 1982	1	Yên Khánh, Ninh Bình	C24101
0241010004	Nguyễn Đình	Bình	Oct 6 1982	1	Huế	C24101
0241010005	Hồ Đăng	Chiến	Jan 20 1982	1	Phong Điền, TTHuế	C24101
0241020001	Nguyễn Tuấn	Anh	Jul 15 1979	1	Do Linh, Quảng Trị	C24102
0241020002	Trần Thị Kim	Anh	Nov 4 1982	0	Phong Điền, TTHuế	C24102
0241020003	Võ Đức	Ân	May 24 1982	1	Huế	C24102
0241020004	Nguyễn Công	Bình	Jun 6 1979	1	Thăng Bình, Quảng Nam	C24102
0241020005	Nguyễn Thanh	Bình	Apr 24 1982	1	Huế	C24102
...	...	...	...	...	...	...

Hình 1.1: *Các bảng trong một cơ sở dữ liệu*

Bảng

- Tên
- Cột, dòng
- Dữ liệu của bảng

Khóa của bảng

- Khái niệm
- Khóa chính
- Khóa ngoại

Ví dụ về khóa chính-primary key

MAMONHOC	TENMONHOC	SODVHT
HO-001	Hoá đại cương	3
TI-001	Tin học đại cương	4
TI-002	Ngôn ngữ C	5
TI-003	Lý thuyết hệ điều hành	4
TI-004	Cấu trúc dữ liệu và giải thuật	4
TO-001	Đại số tuyến tính	4
TO-002	Giải tích 1	4
TO-003	Bài tập Đại số	2
TO-004	Bài tập Giải tích 1	2
VL-001	Vật lý đại cương	3

Ví dụ về khóa ngoại-FK

MAKHOA	TENKHOA	DIENTHOAI
DHT01	Khoa Toán cơ - Tin học	054822407
DHT02	Khoa Công nghệ thông tin	054826767
DHT03	Khoa Vật lý	054823462
...	...	...

MALOP	TENLOP	KHOA	HEDAOTAO	NAMNHAPHOC	SISO	MAKHOA
C24101	Toán K24	24	Chính quy	2000	5	DHT01
C25101	Toán K25	25	Chính quy	2001	5	DHT01
C25102	Tin K25	25	Chính quy	2001	6	DHT02
C24102	Tin K24	24	Chính quy	2000	8	DHT02
...	...	...	...	...	...	...

Giới thiệu

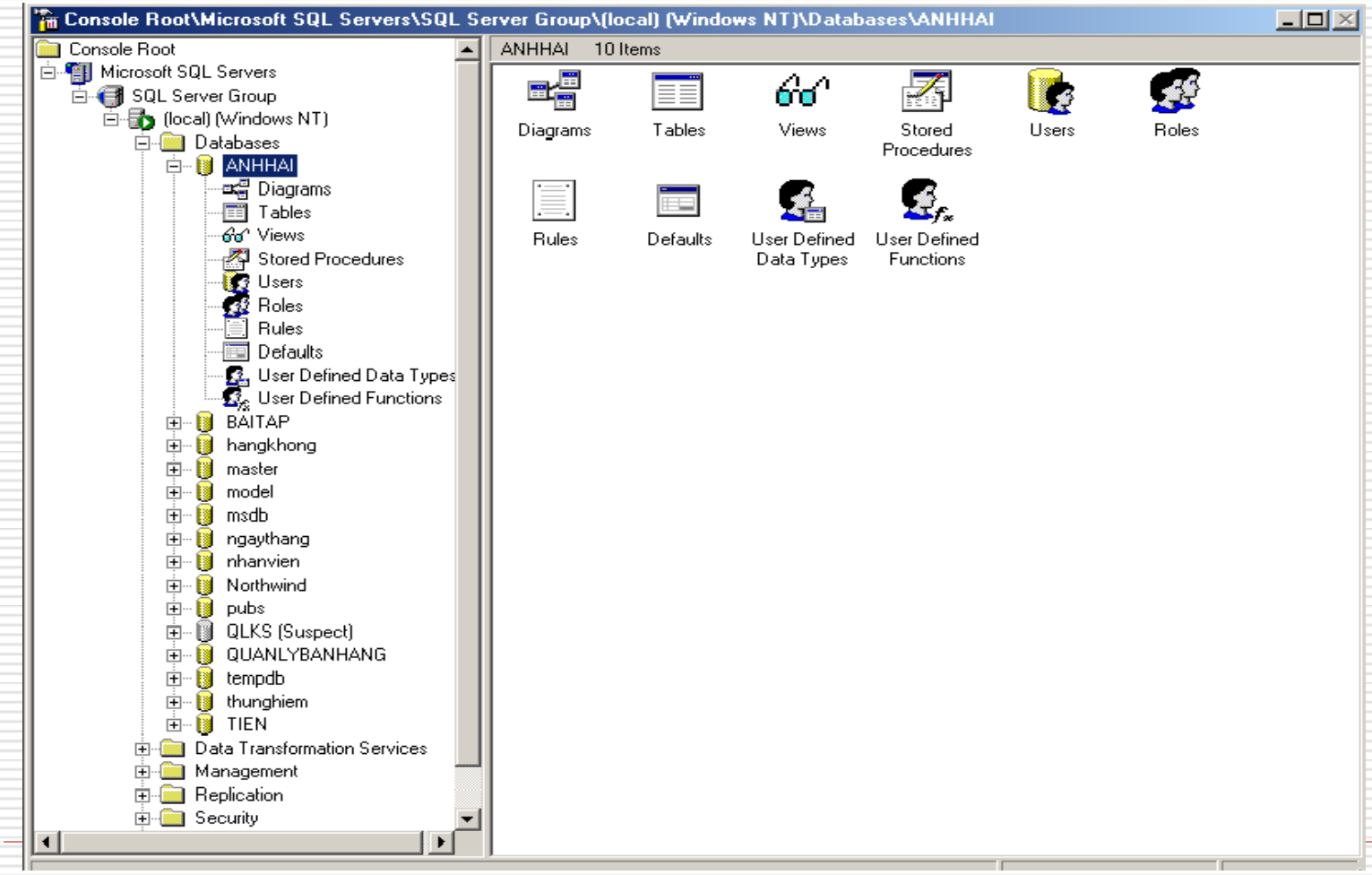
- Các hệ quản trị cơ sở dữ liệu
  - MS Access
  - Oracle
  - Foxpro
  - SQL Server
- Mục đích
  - Quản lý dữ liệu hiệu quả

Giới thiệu sơ lược
SQL SERVER 2000

Các thành phần trong SS 2000

- **Relational Database Engine**
- **Replication**
- **Data Transformation Service**
- **Analysis Service**
- **Meta Data Service**
- **SQL Server Tools**
  - **Enterprise Manager**
  - **Query Analyzer**

Tìm hiểu **Enterprise Manager**



Tìm hiểu **Query Analyzer**

- Là cửa sổ để viết lệnh
- Ngôn ngữ truy vấn SQL (**Structured query language)**

SQL- Giới thiệu

- SQL được xem là ngôn ngữ chuẩn trong cơ sở dữ liệu.
- SQL dùng để:
  - Định nghĩa dữ liệu
  - Truy xuất-thao tác dữ liệu
  - Điều khiển quyền truy cập
  - Đảm bảo toàn vẹn dữ liệu

Sơ lược về SQL

- Bao gồm khoảng 40 câu lệnh
- *Các lệnh thao tác dữ liệu:*
  - SELECT
  - INSERT
  - UPDATE
  - DELETE
  - TRUNCATE

Sơ lược về SQL

- *Định nghĩa dữ liệu*:
  - CREATE TABLE
  - DROP TABLE
  - ALTER TABLE
  - CREATE VIEW
  - ALTER VIEW
  - DROP VIEW
  - CREATE PROCEDURE
  - ALTER PROCEDURE
  - DROP PROCEDURE

NGÔN NGỮ THAO TÁC DỮ LIỆU

SELECT [ALL | DISTINCT][TOP n] danhsáchchọn
[INTO tênbảngmới]
FROM danhsáchbảng/khungnhìn
[WHERE điềukiện]
[GROUP BY danhsáchcột]
[HAVING điềukiện]
[ORDER BY cộtsắpxếp]
[COMPUTE danhsáchhàmgộp [BY danhsáchcột]]

Bảng dùng để minh họa-LOP

MALOP	TENLOP	KHOA	HEDAOTAO	NAMNHAPHOC	MAKHOA
C24101	Toán K24	24	Chính quy	2000	DHTO1
C24102	Tin K24	24	Chính quy	2000	DHTO2
C24103	Lý K24	24	Chính quy	2000	DHTO3
C24301	Sinh K24	24	Chính quy	2000	DHTO5
C25101	Toán K25	25	Chính quy	2001	DHTO1
C25102	Tin K25	25	Chính quy	2001	DHTO2
C25103	Lý K25	25	Chính quy	2001	DHTO3
C25301	Sinh K25	25	Chính quy	2001	DHTO5
C26101	Toán K26	26	Chính quy	2002	DHTO1
C26102	Tin K26	26	Chính quy	2002	DHTO2

Mệnh đề From

- Được dùng để chỉ các bảng và khung nhìn cần truy xuất DL.

Ví dụ: SELECT * FROM Khoa

- Bí danh được chỉ định ngay sau tên bảng.

Ví dụ: SELECT * FROM Khoa a

Danh sách chọn trong SELECT

- Được sử dụng để chỉ định các trường, các biểu thức cần hiển thị trong các cột của kết quả truy vấn.
- DSC được sử dụng trong các trường hợp sau:
  - Chọn tất cả các cột trong bảng
  - Chỉ định các cột hiển thị
  - Thay đổi tiêu đề cột- AS
    - 'Mã lớp'=malop, tenlop 'Tên lớp', khoa As Khóa

Cấu trúc CASE trong SELECT

- Được sử dụng trong DSC nhằm thay đổi kết quả của truy vấn tùy thuộc vào các trường hợp khác nhau.
- Cú pháp:
  - CASE biểu_thức
    WHEN biểu_thức_kiểm_tra THEN kết_quả
    [...]
    [ELSE kết_quả_của_else]
    END.

Cấu trúc CASE trong SELECT

- CASE
  WHEN điều_kiện THEN kết_quả
  [...]
  [ELSE kết_quả_của_else]
  END.

Sử dụng cấu trúc CASE

- SELECT masv, hodem,ten
  CASE gioitinh
  WHEN 1 THEN ‘Nam’
  ELSE ‘Nữ’
  END AS gioitinh
  FROM sinhvien

Sử dụng cấu trúc CASE

```
SELECT masv, hodem,ten
    CASE
    WHEN gioitinh=1 THEN ‘Nam’
     ELSE ‘Nữ’
     END AS gioitinh
  FROM sinhvien
```

Hằng và biểu thức trong DSC

- Mỗi một biểu thức trong DSC trở thành một cột trong kết quả truy vấn.

Ví dụ: SELECT tenmonhoc, sodvht*15 AS sotiet
FROM monhoc

Ví dụ: SELECT tenmonhoc,'Số tiết:', sodvht*15
FROM monhoc

DISTINCT-TOP

- DISTINCT dùng để loại bỏ các kết quả trùng nhau
- TOP n dùng để giới hạn kết quả trả về với số bản ghi là n.
  - Top n
  - Top n percent

Chỉ định điều kiện truy vấn DL

- Mệnh đề WHERE trong câu lệnh SELECT được sử dụng nhằm xác định các điều kiện truy xuất dữ liệu.
- Sau WHERE là một biểu thức logic và chỉ những dòng nào thỏa điều kiện thì mới xuất hiện trong kết quả truy vấn.

Mệnh đề WHERE thường sử dụng

- Các toán tử kết hợp điều kiện (AND,OR)
- Các toán tử so sánh
- Kiểm tra giới hạn của dữ liệu
  - BETWEEN hoặc NOT BETWEEN
- Danh sách
- Kiểm tra khuôn dạng dữ liệu
- Các giá trị NULL

Mệnh đề WHERE thường sử dụng

- Các toán tử so sánh:
  - =,>,<,>=,<=,<>,!>,!<
- Kiểm tra giới hạn dữ liệu
  - giá_trị BETWEEN a AND b
  - giá_trị NOT BETWEEN a AND b
- Danh sách (IN và NOT IN)
  - Sau IN hoặc NOT IN có thể là một danh sách các giá trị hoặc một câu lệnh khác.

Mệnh đề WHERE thường sử dụng

- Toán tử LIKE và các ký tự đại diện
  - Từ khóa LIKE và NOT LIKE sử dụng trong câu lệnh SELECT nhằm mô tả khuôn dạng của dữ liệu
  - Kết hợp với các ký tự đại diện:
    - % chuỗi ký tự bất kỳ không hoặc nhiều ký tự
    - _ ký tự đơn bất kỳ
    - [] ký tự đơn bất kỳ trong giới hạn được chỉ định
    - [^] ký tự đơn bất kỳ không nằm trong giới hạn được chỉ định

Mệnh đề WHERE thường sử dụng

- Giá trị NULL
  - Nếu không có dữ liệu được nhập cho cột và không có mặc định cho cột hay kiểu dữ liệu cho cột đó.
  - Người sử dụng trực tiếp đưa giá trị NULL
  - Một cột có kiểu dữ liệu là kiểu số sẽ chứa giá trị NULL nếu giá trị được chỉ định gây tràn số.

Tạo mới bảng bằng KQ lệnh SELECT

- SELECT ....INTO có tác dụng tạo bảng mới có cấu trúc và dữ liệu là kết quả của truy vấn.
- Sắp xếp kết quả truy vấn
  - Dùng ORDER BY để sắp xếp kết quả truy vấn. DESC giảm dần ASC tăng dần.
  - Nếu sau ORDER BY có nhiều cột thì ưu tiên từ trái qua phải.

Phép hợp

- Phép hợp được sử dụng trong trường hợp ta cần gộp kết quả của hai hay nhiều truy vấn thành một tập kết quả duy nhất.

Phép hợp

- Cú pháp:

```
   Câu_lệnh_1
UNION [ALL] Câu_lệnh_2
[UNION [ALL] Câu_lệnh_3] ...
[UNION [ALL] Câu_lệnh_n]
[ORDER BY cột_sắp_xếp]
[COMPUTE danh_sách_hàm_gộp [BY
   danh_sách_cột]]
```

Phép hợp

Trong đó: Câu lệnh 1 có dạng:

SELECT danh_sách_cột
[INTO tên_bảng_mới]
[FROM danh_sách_bảng|khung_nhìn]
[WHERE điều_kiện]
[GROUP BY danh_sách_cột]
[HAVING điều_kiện]

Phép hợp

Và Câu_lệnh_i (i = 2,..,n) có dạng

SELECT danh_sách_cột
[FROM danh_sách_bảng|khung_nhìn]
[WHERE điều_kiện]
[GROUP BY danh_sách_cột]
[HAVING điều_kiện]

Tìm hiểu qua ví dụ

Table 1

A	B	C
a	1	10
b	2	20
c	3	30
d	4	40
a	5	50
b	6	60

Table 2

D	E
a	1
b	2
d	3
e	4

A	B
a	1
a	5
b	2
b	6
c	3
d	3
d	4
e	4

SELECT A,B FROM Table1
UNION
SELECT D,E FROM Table2

Chú ý khi dùng UNION

- Danh sách cột phải có cùng số lượng
- Các cột tương ứng phải cùng kiểu dữ liệu
- Các cột tương ứng trong bản thân từng truy vấn có thứ tự
- Khi các kiểu dữ liệu khác nhau được kết hợp sẽ chọn kiểu dữ liệu cao hơn
- Tiêu đề cột trong kết quả của phép hợp sẽ là tiêu đề truy vấn đầu tiên

Phép nối

- Khi cần thực hiện một yêu cầu truy vấn dữ liệu từ hai hay nhiều bảng, ta phải sử dụng đến phép nối.

Tìm hiểu phép nối

MAKHOA	TENKHOA	DIENTHOAI
DHTO1	Khoa Toán cơ - Tin học	O54822407
DHTO2	Khoa Công nghệ thông tin	O54826767
DHTO3	Khoa Vật lý	O54823462
DHTO4	Khoa Hoá học	O54823951
DHTO5	Khoa Sinh học	O54822934
DHTO6	Khoa Địa lý - Địa chất	O54823837

Tìm hiểu phép nối

MALOP	TENLOP	KHOA	HEDAOTAO	NAMNHAPHOC	MAKHOA
C24101	Toán K24	24	Chính quy	2000	DHT01
C24102	Tin K24	24	Chính quy	2000	DHT02
C24103	Lý K24	24	Chính quy	2000	DHT03
C24301	Sinh K24	24	Chính quy	2000	DHT05
C25101	Toán K25	25	Chính quy	2001	DHT01
C25102	Tin K25	25	Chính quy	2001	DHT02
C25103	Lý K25	25	Chính quy	2001	DHT03
C25301	Sinh K25	25	Chính quy	2001	DHT05
C26101	Toán K26	26	Chính quy	2002	DHT01
C26102	Tin K26	26	Chính quy	2002	DHT02

Tìm hiểu phép nối

MAKHOA	TENKHOA	DIENTHOAI
DHT01	Khoa Toán cơ - Tin học	054822407
DHT02	Khoa Công nghệ thông tin	054826767
DHT03	Khoa Vật lý	054823462
DHT04	Khoa Hoá học	054823951
DHT05	Khoa Sinh học	054822934
DHT06	Khoa Địa lý - Địa chất	054823837

MALOP	TENLOP	KHOA	HEDAOTAO	NAMNHAPHOC	MAKHOA
C24101	Toán K24	24	Chính quy	2000	DHT01
C24102	Tin K24	24	Chính quy	2000	DHT02
C24103	Lý K24	24	Chính quy	2000	DHT03
C24301	Sinh K24	24	Chính quy	2000	DHT05
C25101	Toán K25	25	Chính quy	2001	DHT01
C25102	Tin K25	25	Chính quy	2001	DHT02
C25103	Lý K25	25	Chính quy	2001	DHT03
C25301	Sinh K25	25	Chính quy	2001	DHT05
C26101	Toán K26	26	Chính quy	2002	DHT01
C26102	Tin K26	26	Chính quy	2002	DHT02

MALOP	TENLOP
C24102	Tin K24
C25102	Tin K25
C26102	Tin K26

Biểu diễn phép nối

```
SELECT malop,tenlop
FROM khoa,lop
WHERE khoa.makhoa = lop.makhoa AND
   tenkhoa='Khoa Công nghệ Thông tin'
```

Sử dụng phép nối

Các yếu tố cần xác định khi thực hiện phép nối:

- Những cột nào cần hiển thị trong kết quả truy vấn
- Những bảng nào có tham gia vào truy vấn.
- Điều kiện để thực hiện phép nối giữa các bảng dữ liệu là gì?

Sử dụng phép nối trong SQL2

Cú pháp phép nối trong:

tên_bảng_1 [INNER] JOIN tên_bảng_2
ON điều_kiện_nối

Ví dụ:

```
SELECT hodem,ten,ngaysinh
FROM sinhvien INNER JOIN lop
        ON sinhvien.malop=lop.malop
WHERE tenlop='Tin K24'
```

Sử dụng phép nối trong SQL2

- Phép nối ngoài trái (LEFT OUTER JOIN)
- Phép nối ngoài phải (RIGHT OUTER JOIN)
- Phép nối ngoài đầy đủ (FULL OUTER JOIN)

Cú pháp:

tên_bảng_1 LEFT|RIGHT|FULL [OUTER]
JOIN tên_bảng_2 ON điều_kiện_nối

Sử dụng phép nối trong SQL2

Bảng DONVI

MADV	TENDV
1	Doi ngoai
2	Hanh chinh
3	Ke toan
4	Kinh doanh

Bảng NHANVIEN

HOTEN	MADV
Thanh	1
Hoa	2
Nam	2
Vinh	1
Hung	5
Phuong	NULL

Phép nối ngoài trái

```
SELECT *
FROM nhanvien LEFT OUTER JOIN donvi
    ON nhanvien.madv=donvi.madv
```

HOTEN	MADV	MADV	TENDV
Thanh	1	1	Doi ngoai
Hoa	2	2	Hanh chinh
Nam	2	2	Hanh chinh
Vinh	1	1	Doi ngoai
Hung	5	NULL	NULL
Phuong	NULL	NULL	NULL

Phép nối ngoài phải

```
SELECT *
FROM nhanvien RIGHT OUTER JOIN
    donvi ON nhanvien.madv=donvi.madv
```

HOTEN	MADV	MADV	TENDV
Thanh	1	1	Doi ngoai
Vinh	1	1	Doi ngoai
Hoa	2	2	Hanh chinh
Nam	2	2	Hanh chinh
NULL	NULL	3	Ke toan
NULL	NULL	4	Kinh doanh

Phép nối ngoài đầy đủ

```
SELECT *
FROM nhanvien FULL OUTER JOIN donvi
ON nhanvien.madv=donvi.madv
```

HOTEN	MADV	MADV	TENDV
Thanh	1	1	Doi ngoai
Hoa	2	2	Hanh chinh
Nam	2	2	Hanh chinh
Vinh	1	1	Doi ngoai
Hung	5	NULL	NULL
Phuong	NULL	NULL	NULL
NULL	NULL	4	Kinh doanh
NULL	NULL	3	Ke toan

Thống kê dữ liệu với GROUP BY

- Nhằm phân hoạch các dòng dữ liệu trong bảng thành các nhóm dữ liệu.
- Các hàm gộp:

SUM(biểu_thức)

AVG(biểu_thức)

COUNT(biểu_thức)

COUNT(*) Đếm số các dòng được chọn.

MAX(biểu_thức) MIN(biểu_thức)

Thống kê trên toàn bộ dữ liệu

- Ví dụ:

```
SELECT AVG(diemlan1)
FROM diemthi
```

- Ví dụ:

```
SELECT MAX(YEAR(GETDATE())- YEAR(ngaysinh)),
    MIN(YEAR(GETDATE())-YEAR(ngaysinh)),
    AVG(YEAR(GETDATE())-YEAR(ngaysinh))
FROM sinhvien
WHERE noisinh='Huế'
```

Thống kê dữ liệu trên các nhóm

Ví dụ:

```
SELECT lop.malop,tenlop,COUNT(masv)
    AS siso
FROM lop,sinhvien
WHERE lop.malop=sinhvien.malop
GROUP BY lop.malop,tenlop
```

Thống kê dữ liệu trên các nhóm

MALOP	TENLOP	SISO
C24101	Toán K24	5
C24102	Tin K24	8
C24103	Lý K24	7
C24301	Sinh K24	5
C25101	Toán K25	5
C25102	Tin K25	6
C25103	Lý K25	6
C25301	Sinh K25	8
C26101	Toán K26	5
C26102	Tin K26	5

Thống kê dữ liệu trên các nhóm

```
SELECT sinhvien.masv,hodem,ten,
          sum(diemlan1*sodvht)/sum(sodvht)
FROM sinhvien,diemthi,monhoc
WHERE sinhvien.masv=diemthi.masv AND
      diemthi.mamonhoc=monhoc.mamonhoc
GROUP BY sinhvien.masv,hodem,ten
```

Thống kê dữ liệu trên các nhóm

Chú ý:

```
SELECT lop.malop, tenlop,COUNT(masv)
FROM lop,sinhvien
WHERE lop.malop=sinhvien.malop
GROUP BY lop.malop
```

Chỉ định điều kiện đối với hàm gộp

```
SELECT sinhvien.masv,hodem,ten,
SUM(diemlan1*sodvht)/sum(sodvht)
FROM sinhvien,diemthi,monhoc
WHERE sinhvien.masv=diemthi.masv AND
diemthi.mamonhoc=monhoc.mamonhoc
GROUP BY sinhvien.masv,hodem,ten
HAVING
   sum(diemlan1*sodvht)/sum(sodvht)>=5
```

Thống kê dữ liệu với COMPUTE

```
SELECT khoa.makhoa,tenkhoa,COUNT(malop)
    AS solop
FROM khoa,lop
WHERE khoa.makhoa=lop.makhoa
GROUP BY khoa.makhoa,tenkhoa
```

MAKHOA	TENKHOA	SOLOP
DHTO1	Khoa Toán cơ - Tin học	3
DHTO2	Khoa Công nghệ thông tin	3
DHTO3	Khoa Vật lý	2
DHTO5	Khoa Sinh học	2

Thống kê dữ liệu với COMPUTE

Cú pháp:

COMPUTE hàm_gộp(tên_cột) [,…, hàm_gộp (tên_cột)] BY danh_sách_cột

Trong đó:

- Hàm gộp: SUM, AVG, MIN, MAX,COUNT.
- danh_sách_cột là DS cột sử dụng để phân nhóm dữ liệu

Thống kê dữ liệu với COMPUTE

```
SELECT
    khoa.makhoa,tenkhoa,malop,tenlop
FROM khoa,lop
WHERE khoa.makhoa=lop.makhoa
ORDER BY khoa.makhoa
COMPUTE COUNT(malop) BY
    khoa.makhoa
```

Thống kê dữ liệu với COMPUTE

MAKHOA	TENKHOA	MALOP	TENLOP
DHT01	Khoa Toán cơ - Tin học	C24101	Toán K24
DHT01	Khoa Toán cơ - Tin học	C25101	Toán K25
DHT01	Khoa Toán cơ - Tin học	C26101	Toán K26
		CNT	
		3	

MAKHOA	TENKHOA	MALOP	TENLOP
DHT02	Khoa Công nghệ thông tin	C26102	Tin K26
DHT02	Khoa Công nghệ thông tin	C25102	Tin K25
DHT02	Khoa Công nghệ thông tin	C24102	Tin K24
		CNT	
		3	

MAKHOA	TENKHOA	MALOP	TENLOP
DHT03	Khoa Vật lý	C24103	Lý K24
DHT03	Khoa Vật lý	C25103	Lý K25
		CNT	
		2	

MAKHOA	TENKHOA	MALOP	TENLOP
DHT05	Khoa Sinh học	C25301	Sinh K25
DHT05	Khoa Sinh học	C24301	Sinh K24
		CNT	
		2	

Thống kê dữ liệu với COMPUTE

Sử dụng COMPUTE ... BY thì cũng phải sử dụng mệnh đề ORDER BY.

Các cột liệt kê trong COMPUTE … BY và ORDER BY giống nhau hoàn toàn, và có cùng thứ tự.

Thống kê dữ liệu với COMPUTE

Nếu ORDER BY a, b, c thì:

COMPUTE F(X) BY a, b, c

COMPUTE F(X) BY a, b

COMPUTE F(X) BY a

COMPUTE F(X) BY b, c

COMPUTE F(X) BY a, c

COMPUTE F(X) BY c

Thống kê dữ liệu với COMPUTE

Trường hợp không có BY thì không cần sử dụng Order By, khi đó phạm vi tính toán của hàm gộp là trên toàn bộ dữ liệu.

```
SELECT malop,tenlop,hedaotao
FROM lop
ORDER BY makhoa
COMPUTE COUNT(malop)
```

Thống kê dữ liệu với COMPUTE

MALOP	TENLOP	HEDAOTAO
C24101	Toán K24	Chính quy
C25101	Toán K25	Chính quy
C26101	Toán K26	Chính quy
C26102	Tin K26	Chính quy
C25102	Tin K25	Chính quy
C24102	Tin K24	Chính quy
C24103	Lý K24	Chính quy
C25103	Lý K25	Chính quy
C25301	Sinh K25	Chính quy
C24301	Sinh K24	Chính quy

CNT
10

Truy vấn con

Là một câu lệnh SELECT được lồng vào bên trong một câu lệnh SELECT, INSERT, UPDATE, DELETE hoặc bên trong một truy vấn con khác.

Cú pháp:

```
(SELECT [ALL | DISTINCT] danh_sách_chọn
 FROM danh_sách_bảng
 [WHERE điều_kiện]
 [GROUP BY danh_sách_cột]
 [HAVING điều_kiện])
```

Phép so sánh đối với kết quả truy vấn con

WHERE biểu_thức phép_toán_số_học [ANY|ALL] (truy_vấn_con)

Ví dụ:

```
SELECT *
FROM monhoc
WHERE sodvht>=(SELECT sodvht
               FROM monhoc
               WHERE mamonhoc='TI-001')
```

Phép so sánh đối với kết quả truy vấn con

```
SELECT hodem,ten
FROM sinhvien JOIN lop ON
    sinhvien.malop=lop.malop
WHERE tenlop='Tin K25' AND
      ngaysinh<ALL(SELECT ngaysinh
                   FROM sinhvien JOIN lop
                   ON sinhvien.malop=lop.malop
                   WHERE lop.tenlop='Toán K25')
```

Phép so sánh đối với kết quả truy vấn con

```
SELECT hodem,ten
FROM sinhvien JOIN lop on
    sinhvien.malop=lop.malop
WHERE tenlop='Tin K25' AND
year(ngaysinh)= ANY(SELECT year(ngaysinh)
                    FROM sinhvien JOIN lop
                    ON sinhvien.malop=lop.malop
                    WHERE lop.tenlop='Toán K25')
```

Sử dụng truy vấn con với toán tử IN

Cú pháp:

WHERE biểu_thức [NOT] IN
(truy_vấn_con)

Sử dụng truy vấn con với toán tử IN

```
SELECT hodem,ten
FROM sinhvien JOIN lop on
    sinhvien.malop=lop.malop
WHERE tenlop='Tin K25' AND
year(ngaysinh) IN(SELECT year(ngaysinh)
                  FROM sinhvien JOIN lop
                 ON sinhvien.malop=lop.malop
                 WHERE lop.tenlop='Toán K25')
```

Sử dụng lượng từ EXISTS với truy vấn con

```
SELECT hodem,ten
FROM sinhvien
WHERE NOT EXISTS(SELECT masv
      FROM diemthi
     WHERE diemthi.masv=sinhvien.masv)
```

Sinh viên tìm các ví dụ để
làm rõ nghĩa câu lệnh exists.

Bổ sung, cập nhật và xóa DL

- Lệnh INSERT
- Lệnh UPDATE
- Lệnh DELETE

Bổ sung từng dòng dữ liệu với lệnh INSERT

Cú pháp:

INSERT INTO tên_bảng[(danh_sách_cột)]
VALUES(danh_sách_trị)

Ví dụ:

INSERT INTO khoa VALUES('DHT10','Khoa Luật','054821135')

Bổ sung từng dòng dữ liệu với lệnh INSERT

```
INSERT INTO
    sinhvien(masv,hodem,ten,gioitinh,malop)
VALUES('0241020008','Nguyễn
    Công','Chính',1,'C24102')
```

Câu lệnh còn được viết như sau:

```
INSERT INTO sinhvien
VALUES('0241020008','Nguyễn Công','Chính',
      NULL,1,NULL,'C24102')
```

Bổ sung nhiều dòng dữ liệu từ bảng khác

Cú pháp:

INSERT INTO tên_bảng[(danh_sách_cột)]
câu_lệnh_SELECT

Ví dụ

Cập nhật dữ liệu

Cú pháp:

UPDATE tên_bảng

SET tên_cột = biểu_thức

[, ..., tên_cột_k = biểu_thức_k]

[FROM danh_sách_bảng]

[WHERE điều_kiện]

Cập nhật dữ liệu

Ví dụ:

```
UPDATE monhoc
SET sodvht = 3
WHERE sodvht = 2
```

- Cấu trúc Case trong

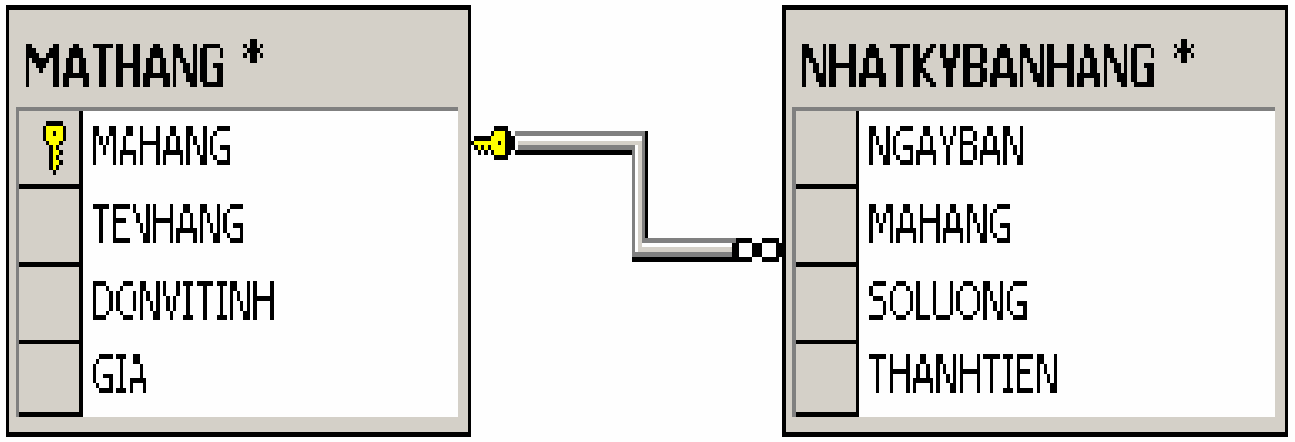
Cập nhật dữ liệu

Ví dụ:

```
UPDATE nhatkyphong
SET tienphong=songay*CASE WHEN
   loaiphong='A' THEN 100
WHEN loaiphong='B' THEN 70
ELSE 50
END
```

Cập nhật dữ liệu

- Điều kiện cập nhật dữ liệu liên quan đến nhiều bảng



Xóa dữ liệu

DELETE FROM Tên_bảng

FROM danh_sách_bảng

WHERE điều_kiện

Ví dụ: DELETE FROM SINHVIEN

WHERE noisinh LIKE '%Huế%'

Xóa dữ liệu

- Sử dụng truy vấn con trong Delete

```
DELETE FROM sinhvien
WHERE malop NOT IN (SELECT
    DISTINCT malop FROM sinhvien)
```

- Xóa toàn bộ dữ liệu trong bảng

```
DELETE FROM Điểm_thi
TRUNCATE TABLE Tên_bảng
```

Ngôn ngữ định nghĩa dữ liệu

Về cơ bản bao gồm các lệnh:

- CREATE định nghĩa và tạo ĐT mới
- ALTER thay đổi định nghĩa của bảng
- DROP xóa đối tượng CSDL

Ngôn ngữ định nghĩa dữ liệu

- Tạo bảng mới:
  - Cấu trúc bảng gồm những cột nào...
  - Khóa chính của bảng là cột nào...
  - Các ràng buộc về khuôn dạng dữ liệu

Ngôn ngữ định nghĩa dữ liệu

```
CREATE TABLE tên_bảng
(
   tên_cột thuộc_tính_cột các_ràng_buộc
   [,... ,tên_cột_n thuộc_tính_cột_n
    các_ràng_buộc_cột_n]
   [,các_ràng_buộc_trên_bảng]
)
```

Ngôn ngữ định nghĩa dữ liệu

```
CREATE TABLE nhanvien
(
manv NVARCHAR(10) NOT NULL,
hoten NVARCHAR(50) NOT NULL,
ngaysinh DATETIME NULL,
dienthoai NVARCHAR(10) NULL,
hsluong DECIMAL(3,2) DEFAULT (1.92)
)
```

Ngôn ngữ định nghĩa dữ liệu

```
INSERT INTO nhanvien
VALUES('NV01','Le Van
   A','2/4/75','886963',2.14)
INSERT INTO nhanvien(manv,hoten)
VALUES('NV02','Mai Thi B')
INSERT INTO
   nhanvien(manv,hoten,dienthoai)
VALUES('NV03','Tran Thi C','849290')
```

Ngôn ngữ định nghĩa dữ liệu

- Ràng buộc CHECK:

[CONSTRAINT tên_ràng_buộc]

CHECK (điều_kiện)

Ngôn ngữ định nghĩa dữ liệu

```
CREATE TABLE diemtotnghiep
(
hoten NVARCHAR(30) NOT NULL,
ngaysinh DATETIME,
diemvan DECIMAL(4,2)
CONSTRAINT chk_diemvan
CHECK(diemvan>=0 AND diemvan<=10),
diemtoan DECIMAL(4,2)
CONSTRAINT chk_diemtoan
CHECK(diemtoan>=0 AND diemtoan<=10),
)
```

Ngôn ngữ định nghĩa dữ liệu

```
INSERT INTO
    diemtotnghiep(hoten,diemvan,diemtoan)
VALUES('Le Thanh Hoang',9.5,2.5)
INSERT INTO diemtotnghiep(hoten,diemvan)
VALUES('Hoang Thi Mai',2.5)
```

Không hợp lệ:

```
INSERT INTO
    diemtotnghiep(hoten,diemvan,diemtoan)
VALUES('Tran Van Hanh',6,10.5)
```

Ngôn ngữ định nghĩa dữ liệu

```
CREATE TABLE lop
(
malop NVARCHAR(10) NOT NULL ,
tenlop NVARCHAR(30) NOT NULL ,
khoa SMALLINT NULL ,
hedaotao NVARCHAR(25) NULL,
namnhaphoc INT NULL ,
makhoa NVARCHAR(5),
CONSTRAINT chk_lop
CHECK (namnhaphoc<=YEAR(GETDATE()) AND
hedaotao IN ('chính quy','tại chức'))
)
```

Ngôn ngữ định nghĩa dữ liệu

- Ràng buộc Primary Key

  [CONSTRAINT tên_ràng_buộc]

  PRIMARY KEY [(danh_sách_cột)]

Ngôn ngữ định nghĩa dữ liệu

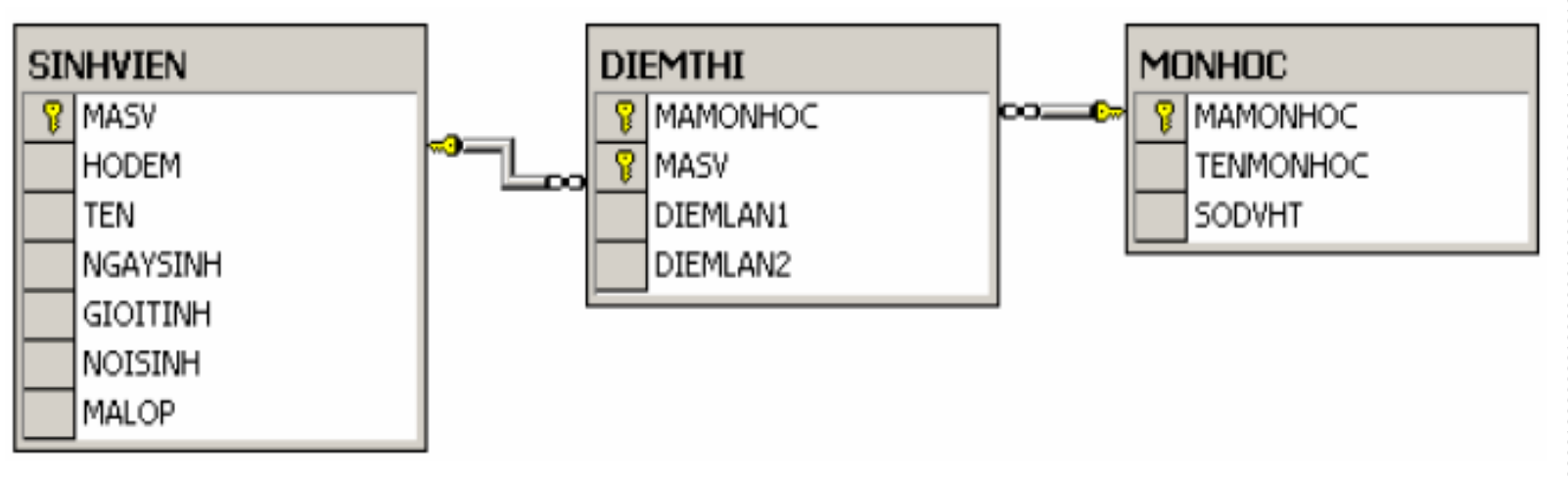
```
CREATE TABLE sinhvien
(
masv NVARCHAR(10)
CONSTRAINT pk_sinhvien_masv PRIMARY
    KEY,
hodem NVARCHAR(25) NOT NULL ,
ten NVARCHAR(10) NOT NULL ,
ngaysinh DATETIME,
gioitinh BIT,
noisinh NVARCHAR(255),
malop NVARCHAR(10)
)
```

Ngôn ngữ định nghĩa dữ liệu

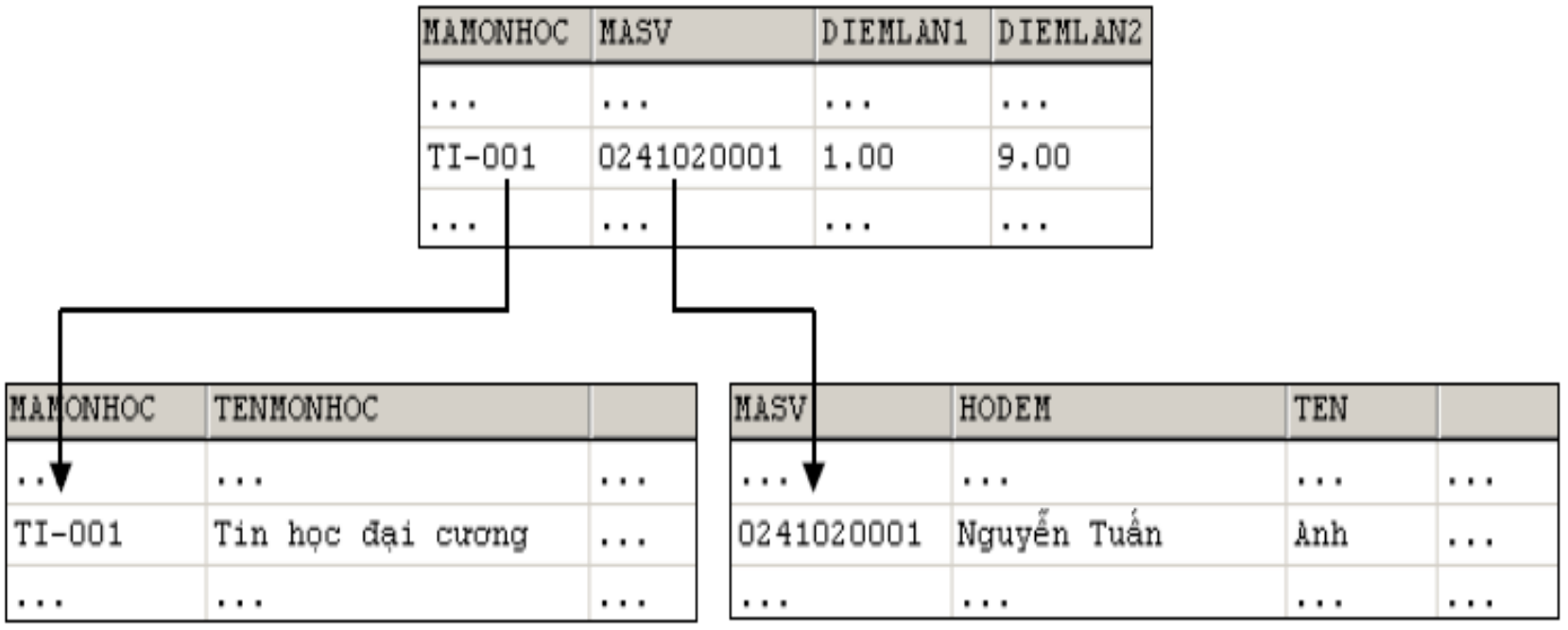
```
CREATE TABLE diemthi
(
mamonhoc NVARCHAR(10) NOT NULL ,
masv NVARCHAR(10) NOT NULL ,
diemlan1 NUMERIC(4, 2),
diemlan2 NUMERIC(4, 2),
CONSTRAINT pk_diemthi PRIMARY
    KEY(mamonhoc,masv)
)
```

Ngôn ngữ định nghĩa dữ liệu

- Ràng buộc FOREIGN KEY



Ngôn ngữ định nghĩa dữ liệu



Ngôn ngữ định nghĩa dữ liệu

[CONSTRAINT tên_ràng_buộc]

FOREIGN KEY [(danh_sách_cột)]

REFERENCES

tên_bảng_tham_chiếu(danh_sách_cột_tham_chiếu)

[ON DELETE CASCADE | NO ACTION | SET NULL | SET DEFAULT]

[ON UPDATE CASCADE | NO ACTION | SET NULL | SET DEFAULT]

Ngôn ngữ định nghĩa dữ liệu

```
CREATE TABLE diemthi
(
masv NVARCHAR(10) NOT NULL ,
diemlan2 NUMERIC(4, 2),
CONSTRAINT pk_diemthi PRIMARY
    KEY(mamonhoc,masv),
CONSTRAINT fk_diemthi_mamonhoc
FOREIGN KEY(mamonhoc)
REFERENCES monhoc(mamonhoc)
ON DELETE CASCADE
ON UPDATE CASCADE
CONSTRAINT fk_diemthi_masv ….
```

Ngôn ngữ định nghĩa dữ liệu

- Sửa đổi định nghĩa bảng
  - Bổ sung một cột vào bảng.
  - Xoá một cột khỏi bảng.
  - Thay đổi định nghĩa của một cột trong bảng.
  - Xoá bỏ hoặc bổ sung các ràng buộc cho bảng

Ngôn ngữ định nghĩa dữ liệu

ALTER TABLE tên_bảng

ADD định_nghĩa_cột |

ALTER COLUMN tên_cột kiểu_dữ_liêu [NULL | NOT NULL] |

DROP COLUMN tên_cột |

ADD CONSTRAINT tên_ràng_buộc định_nghĩa_ràng_buộc |

DROP CONSTRAINT tên_ràng_buộc

Ngôn ngữ định nghĩa dữ liệu

Giả sử có hai bảng NHANVIEN và DONVI:

```
CREATE TABLE donvi
  (
     madv INT NOT NULL PRIMARY KEY,
     tendv NVARCHAR(30) NOT NULL
  )
```

Ngôn ngữ định nghĩa dữ liệu

```
CREATE TABLE nhanvien
(
manv NVARCHAR(10) NOT NULL,
hoten NVARCHAR(30) NOT NULL,
ngaysinh DATETIME,
diachi CHAR(30) NOT NULL
)
```

Ngôn ngữ định nghĩa dữ liệu

Bổ sung ràng buộc:

```
ALTER TABLE nhanvien
ADD
dienthoai NVARCHAR(6)
CONSTRAINT chk_nhanvien_dienthoai
CHECK (dienthoai LIKE '[0-9][0-9][0-9][0-9][0-9][0-9]')
```

Ngôn ngữ định nghĩa dữ liệu

ALTER TABLE nhanvien

ADD madv INT NULL

Sửa đổi lại để có giá trị NULL

ALTER TABLE nhanvien

ALTER COLUMN diachi NVARCHAR(100) NULL

Xóa cột:

- ALTER TABLE nhanvien

DROP COLUMN ngaysinh

Ngôn ngữ định nghĩa dữ liệu

```
ALTER TABLE nhanvien
ADD
CONSTRAINT pk_nhanvien PRIMARY KEY(manv)
```

Bổ sung khóa ngoại:

```
ALTER TABLE nhanvien
ADD
CONSTRAINT fk_nhanvien_madv
FOREIGN KEY(madv) REFERENCES donvi(madv)
ON DELETE CASCADE
ON UPDATE CASCADE
```

Ngôn ngữ định nghĩa dữ liệu

Xoá bỏ ràng buộc kiểm tra số điện thoại của nhân viên

ALTER TABLE nhanvien

DROP CONSTRAINT CHK_NHANVIEN_DIENTHOAI

- Xóa bảng:

  DROP TABLE tên_bảng

Lưu ý

- Nếu bổ sung thêm một cột vào bảng và trong bảng đã có ít nhất một bản ghi thì cột mới cần bổ sung phải cho phép chấp nhận giá trị NULL hoặc phải có giá trị mặc định.
- Muốn xoá một cột đang được ràng buộc bởi một ràng buộc hoặc đang được tham chiếu bởi một khoá ngoài, ta phải xoá ràng buộc hoặc khoá ngoài trước sao cho trên cột không còn bất kỳ một ràng buộc và không còn được tham chiếu bởi bất kỳ khoá ngoài nào.
- Nếu bổ sung thêm ràng buộc cho một bảng đã có dữ liệu và ràng buộc cần bổ sung không được thoả mãn bởi các bản ghi đã có trong bảng thì câu lệnh ALTER TABLE không thực hiện được.

Ngôn ngữ định nghĩa dữ liệu

- Khung nhìn:

MASV	HODEM	TEN	NGAYSINH
0241010001	Ngô Thị Nhật	Anh	Nov 27 1982 ...
0241010002	Nguyễn Thị Ngọc	Anh	Mar 21 1983 ...
0241010003	Ngô Việt	Bắc	May 11 1982 ...
0241010004	Nguyễn Đình	Bình	Oct 6 1982 ...
0241010005	Hồ Đăng	Chiến	Jan 20 1982 ...
0241020001	Nguyễn Tuấn	Anh	Jul 15 1979 ...
0241020002	Trần Thị Kim	Anh	Nov 4 1982 ...
...	...	...	...

Table SINHVIEN

MALOP	TENLOP	
C24101	Toán K24	...
C24102	Tin K24	...
C24103	Lý K24	...
..	...	....

Table LOP

MASV	HODEM	TEN	TUOI	TENLOP
0241010001	Ngô Thị Nhật	Anh	22	Toán K24
0241010002	Nguyễn Thị Ngọc	Anh	21	Toán K24
0241010003	Ngô Việt	Bắc	22	Toán K24
0241010004	Nguyễn Đình	Bình	22	Toán K24
0241010005	Hồ Đăng	Chiến	22	Toán K24
0241020001	Nguyễn Tuấn	Anh	25	Tin K24
0241020002	Trần Thị Kim	Anh	22	Tin K24

Ngôn ngữ định nghĩa dữ liệu

CREATE VIEW
tên_khung_nhìn[(danh_sách_tên_cột)]
AS câu_lệnh_SELECT

Ngôn ngữ định nghĩa dữ liệu

```
CREATE VIEW dssv AS
    SELECT masv,hodem, ten,
    DATEDIFF(YY,ngaysinh,GETDATE()) AS
        tuoi,tenlop
FROM sinhvien,lop
WHERE sinhvien.malop=lop.malop
```

* Trong khung nhìn, dữ liệu "chỉ đọc"

Lưu ý khi tạo khung nhìn:

- Tên khung nhìn và tên cột tuân theo quy tắc định danh
- Không thể quy định ràng buộc và chỉ mục cho khung nhìn
- Câu lệnh SELECT:

- mệnh đề COMPUTE BY không được sử dụng
- Biểu thức phải đặt tên tiêu đề
- Không được trùng tiêu đề cột

Sửa đổi khung nhìn

- ALTER VIEW *tên_khung_nhìn* [(*danh_sách_tên_cột*)]

AS *Câu_lệnh_SELECT*

Định nghĩa khung nhìn như sau:

```
CREATE VIEW viewlop
AS
SELECT malop,tenlop,tenkhoa
FROM lop INNER JOIN khoa ON
lop.makhoa=khoa.makhoa
WHERE tenkhoa='Khoa Vật lý’
```

và có thể định nghĩa lại khung nhìn trên bằng câu lệnh:

- ALTER VIEW view_lop

AS

SELECT malop,tenlop,hedaotao

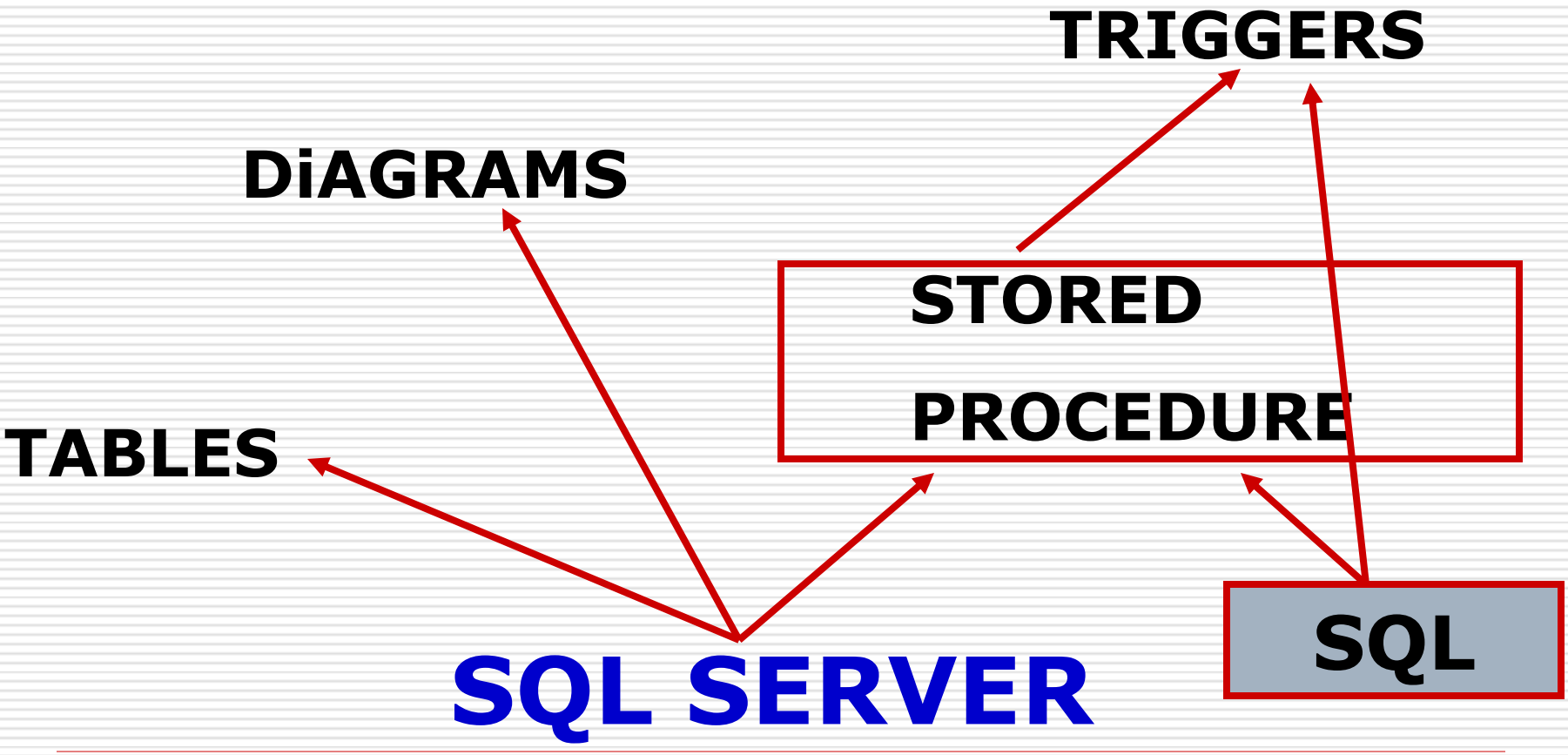
FROM lop INNER JOIN khoa ON lop.makhoa=khoa.makhoa

WHERE tenkhoa='Khoa Công nghệ thông tin'

Xoá khung nhìn

- DROP VIEW *tên_khung_nhìn*

Chương 7: LẬP TRÌNH TRONG CSDL



Lập trình trong CSDL

- Khái niệm
- Biến
  - Biến cục bộ

    - Được khai báo trong phần thân của một bó lệnh hoặc một thủ tục.

    - Bắt đầu bằng @.

Lập trình trong CSDL

- Biến cục bộ
  - Khai báo biến cục bộ:

  DECLARE @TENBIEN KDL

  Ví dụ:
  - Gán giá trị cho biến:

  Dùng SET, dùng SELECT

  Chú ý:
  - Xem giá trị hiện hành của biến

  Print @tênbiến

Lập trình trong CSDL

- Phạm vi: trong một lô, trong thủ tục (SP)
- Ví dụ 1:

```
DECLARE @EmpIDVar INT
SET @EmpIDVar=3
SELECT * FROM [Orders]
WHERE ….
```

Lập trình trong CSDL

- Ví dụ 2:

```
DECLARE @MyVariable INT
SET @MyVariable = 1
GO -- điểm kết thúc của một lô
-- @MyVariable
SELECT * FROM [Orders]
WHERE …..
```

Lập trình trong CSDL

```
DECLARE @FirstNameVariable NVARCHAR(20),
                @RegionVariable NVARCHAR(30)
   SET @FirstNameVariable = N'Anne'
   SET @RegionVariable = N'WA'
SELECT LastName, FirstName, Title
FROM Employees
WHERE FirstName = @FirstNameVariable
            OR Region = @RegionVariable
GO –kết thúc một lô
```

Lập trình trong CSDL

Ví dụ

```
DECLARE @EmpIDVariable INT
--Gán giá trị biến bằng câu lệnh Select
SELECT @EmpIDVariable =
   MAX(EmployeeID)
FROM Employees
GO
```

Lập trình trong CSDL

- Biến hệ thống
  - Ý nghĩa sử dụng

Một số biến hệ thống thường dùng:

- @@VERSION - Select @@VERSION
- @@TRANCOUNT

```
IF (@@TRANCOUNT > 0)
BEGIN
 RAISERROR('Task cannot be executed within a transaction.', 10, 1)
RETURN
END
```

Lập trình trong CSDL

- @@ROWCOUNT

Trả về số dòng bị ảnh hưởng đối với câu lệnh thực thi gần nhất.

Ví dụ 1:

```
UPDATE Employees SET LastName = 'Brooke'
WHERE LastName = 'Brook'
IF (@@ROWCOUNT = 0)
BEGIN
    PRINT 'Warning: No rows were updated'
    RETURN
END
```

Lập trình trong CSDL

Ví dụ 2:

```
UPDATE Customers
SET Phone =’030’ + Phone
WHERE Country= ’Germany’
PRINT @@ROWCOUNT
```

Lập trình trong CSDL

- Các hàm thường dùng:
- GetDate() lấy ngày hiện hành
- Month(Date); Year(Date)
- DateAdd(Datepart, Number, Date) trả về một Datetime là k/quả cộng thêm vào Date một giá trị số được chỉ trong phần Datepart.

Lập trình trong CSDL

- Datediff(đơn vị, ngày 1, ngày 2)

  Hàm trả về khoảng thời gian giữa hai giá trị kiểu này được chỉ định tuỳ thuộc vào tham số *datepart*

  Đơn vị: dd, mm, yy

- Datepart (datepart, date) Trả về một số nguyên biểu diễn Datepart của ngày được chỉ định.

Lập trình trong CSDL

- LOWER (character_expression) chuyển sang chữ thường
- UPPER (character_expression) Chuyển sang chữ hoa
- CAST (biểu thức AS kdl) chuyển đổi thành kiểu dữ liệu mong muốn
- STR(số) chuyển số thành chuỗi

Lập trình trong CSDL

VD1: SELECT 'The price is ' + CAST(price AS varchar(12))
FROM titles
WHERE price > 10.00 GO

VD2: SELECT SUBSTRING(title, 1, 30) AS Title, ytd_sales
FROM titles
WHERE CONVERT(char(20), ytd_sales) LIKE '3%' GO

Lập trình trong CSDL

Cấu trúc điều khiển:

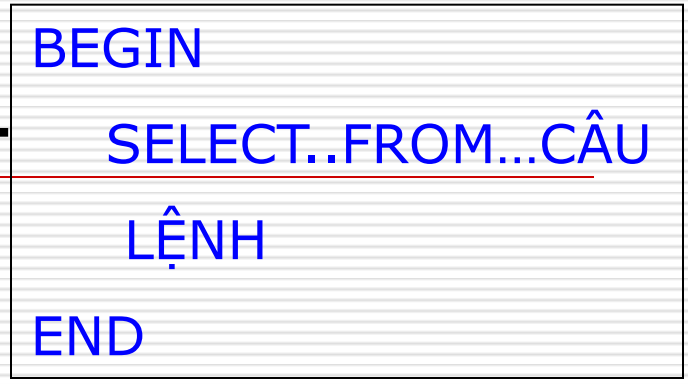
- Khối Begin…End
- Cấu trúc rẽ nhánh If..Else

  Cú pháp:

  - If <biểu thức logic>

    Câu lệnh1| khối lệnh

    Else Câu lệnh 2| khối lệnh



Lập trình trong CSDL

Ví dụ:

```
IF (SELECT COUNT(*) FROM Customers
    WHERE Country='Germany') > 0
BEGIN
Print ‘Có khách hàng này trong CSDL’
END
```

Lập trình trong CSDL

- Kết hợp từ khóa Exists:

If Exists(…)

If Exists (câu lệnh select)

câu lệnh-khối lệnh

Else câu lệnh- khối lệnh

Ví dụ:

Lập trình trong CSDL

- Cấu trúc lặp While

  Cú pháp: While biểu thức logic

  BEGIN

  Câu lệnh

  END

Ví dụ:

Lập trình trong CSDL

- Câu lệnh Return: quá trình xử lý KT
- Câu lệnh WAITFOR

Là một chỉ thị tạm dừng một thời gian trước khi xử lý các câu lệnh tiếp theo.

WAITFOR { DELAY 'time' | TIME 'time' }

- Câu lệnh RAISERROR phát sinh lỗi của người dùng.

Thủ tục lưu trữ

Thủ tục lưu trữ (store procedure)

Khái niệm: là thủ tục được lưu trữ đi theo cơ sở dữ liệu.

Được dùng để xử lý các công việc bên trong ứng dụng như sửa, xóa…

Thủ tục lưu trữ

Thủ tục lưu trữ chứa những dòng lệnh, các biến và các cấu trúc điều khiển bên trong nó.

Thủ tục nội tại cũng tương tự như các hàm trong các ngôn ngữ lập trình khác.

Thủ tục lưu trữ

- Tên, tham số nhận giá trị vào, tham số nhận giá trị trả ra.
- Được phép gọi một thủ tục đã có.
- *Đặc biệt* nó còn được gọi thực hiện bên trong một ngôn ngữ lập trình khác.

Thủ tục lưu trữ

- *Lợi ích khi sử dụng thủ tục nội tại*
  - Tốc độ xử lý nhanh- cùng lưu trữ bên trong một cơ sở dữ liệu
  - Có thể sử dụng để phân quyền cho người sử dụng.

Thủ tục lưu trữ

Cú pháp:

```
CREATE PROCEDURE  tên_thủ_tục
    [(danh_sách_tham_số)]
[WITH
    RECOMPILE|ENCRYPTION|RECOMPILE,ENC
    RYPTION]
AS Các_câu_lệnh_của_thủ_tục
```

Thủ tục lưu trữ

- Tham số bên trong thủ tục lưu trữ
  - Một thủ tục có thể có một hoặc nhiều tham số
  - Tham số chỉ có phạm vi cục bộ
  - Tên tham số phải gợi nhớ và duy nhất
- Tham số đầu vào:

Cú pháp: Create Procedure tenthutuc
@tenthamso kdl

As

Declare @biencucbo

Lệnh

Thủ tục lưu trữ

Trong đó: tên tham số của thủ tục phải là duy nhất

Kiểu dữ liệu: kdl của tham số qui định loại dữ liệu mà tham số truyền vào

Giá trị: là giá trị mặc định được gán vào khi tham số không nhận được giá trị truyền vào

Thủ tục lưu trữ

- *Tham số đầu ra*: là những tham số mà giá trị của nó được tính toán bên trong thủ tục lưu trữ và các giá trị đó vẫn được giữ nguyên sau khi thoát thủ tục.

Cú pháp: create procedure tenthutuc

@tenthamso kdl output

As

Declare @biencucbo

Lệnh

Thủ tục lưu trữ

Bài toán:

1. Bổ sung thêm môn học Cơ sở dữ liệu có mã TI-005 và số đơn vị học trình là 5 vào bảng MONHOC.
2. Lên danh sách nhập điểm thi môn Cơ sở dữ liệu cho các sinh viên học lớp có mã C24102 (tức là bổ sung thêm vào bảng DIEMTHI các bản ghi với cột MAMON nhận giá trị TI-005, cột MASV nhận giá trị lần lượt là mã các sinh viên học lớp có mã C24102 và các cột điểm là NULL).

Thủ tục lưu trữ

MONHOC

MAMON	TENMON	SDVHT
TI-005		

TI-005	Cơ sở dữ liệu	5

DIEMTHI

MAMON	MASV	DIEM	
		NULL	

Thủ tục lưu trữ

```
INSERT INTO MONHOC
   VALUES('TI-005','Cơ sở dữ liệu',5)
INSERT INTO
    DIEMTHI(MAMONHOC,MASV)
   SELECT ‘TI-005’, MASV
   FROM SINHVIEN
   WHERE MALOP= ‘C24102'
```

Thủ tục lưu trữ

```
CREATE PROC sp_LenDanhSachDiem(
    @mamonhoc VARCHAR(10), @tenmonhoc VARCHAR(50),
    @sodvht SMALLINT, @malop VARCHAR(10))
AS
BEGIN
    INSERT INTO MONHOC
    VALUES(@mamonhoc,@tenmonhoc,@sodvht)
    INSERT INTO diemthi(mamonhoc,masv)
    SELECT @mamonhoc, masv FROM sinhvien
    WHERE malop=@malop
END
```

Thủ tục lưu trữ

Lời gọi thủ tục:

- Tên_thủ_tục [danh_sách_các_đối_số]

Chạy thủ tục:

```
EXECUTE tên_thủ_tục [danh_sách_các_đối_số]
```

Lời gọi thủ tục:

```
sp_LenDanhSachDiem @malop='C24102',
        @tenmonhoc='Cơ sở dữ liệu',
        @mamonhoc='TI-005',
        @sodvht =5
```

Thủ tục lưu trữ

- Giữ lại tham số

@tên_tham_số kiểu_dữ_liệu OUTPUT

hoặc:

@tên_tham_số kiểu_dữ_liệu OUT

Thủ tục lưu trữ

Ví dụ:

```
CREATE PROCEDURE sp_Conghaiso(
@a INT,
@b INT,
@c INT OUTPUT)
AS
SELECT @c=@a+@b
```

và thực hiện lời gọi thủ tục trong một tập các câu lệnh như sau:

```
DECLARE @tong INT
SELECT @tong=0
EXECUTE sp_Conghaiso 100,200,@tong OUTPUT
SELECT @tong
```

Tham số với giá trị mặc định

- Tham số với giá trị mặc định được khai báo theo cú pháp như sau:

@tên_tham_số kiểu_dữ_liệu = giá_trị_mặc_định

Sửa đổi, xoá thủ tục

- Sửa: ALTER PROCEDURE *tên_thủ_tục* [(*danh_sách_tham_số*)]

  AS

  Các_câu_lệnh_Của_thủ_tục
- Xoá:

DROP PROCEDURE tên_thủ_tục

Hàm

```
CREATE FUNCTION tên_hàm
   ([danh_sách_tham_số])
RETURNS (kiểu_trả_về_của_hàm)
AS
BEGIN
các_câu_lệnh_của_hàm
END
```

Ví dụ:

```
CREATE FUNCTION thu(@ngay DATETIME)
RETURNS NVARCHAR(10)
AS
BEGIN
DECLARE @st NVARCHAR(10)
SELECT @st=CASE DATEPART(DW,@ngay)
WHEN 1 THEN 'Chu nhật'
WHEN 2 THEN 'Thứ hai'
WHEN 3 THEN 'Thứ ba'
WHEN 4 THEN 'Thứ tư'
WHEN 5 THEN 'Thứ năm'
WHEN 6 THEN 'Thứ sáu'
ELSE 'Thứ bảy'
END
RETURN (@st) /* Trị trả về của hàm */
END
```

```
SELECT masv,hodem,ten,
dbo.thu(ngaysinh),ngaysinh
FROM sinhvien
WHERE malop='C24102'
```

Hàm trả về bảng dữ liệu với 1 biến

- CREATE FUNCTION tên_hàm ([danh_sách_tham_số])

RETURNS TABLE

AS

RETURN (câu_lệnh_select)

Vd:

- CREATE FUNCTION func_XemSV(@khoa SMALLINT)

```
RETURNS TABLE
AS
RETURN(SELECT masv,hodem,ten,ngaysinh
FROM sinhvien INNER JOIN lop
ON sinhvien.malop=lop.malop
WHERE khoa=@khoa)
```

- Xem ds sinh viên khoá 25:

```
SELECT * FROM dbo.func_XemSV(25)
```

Trả về bảng với nhiều biến

```
CREATE FUNCTION tên_hàm
    ([danh_sách_tham_số])
RETURNS @biến bảngTABLE định nghĩa bảng
AS
BEGIN
Các câu lệnh trong thân hàm
RETURN
END
```

```
CREATE FUNCTION Func_Tongsv(@khoa SMALLINT)
RETURNS @bangthongke TABLE
(
makhoa NVARCHAR(5),
tenkhoa NVARCHAR(50),
tongsosv INT
)
AS
BEGIN
IF @khoa=0 //cho tất cả các khoá
    INSERT INTO @bangthongke
        SELECT khoa.makhoa,tenkhoa,COUNT(masv)
        FROM …..
ELSE
    INSERT INTO @bangthongke
    SELECT khoa.makhoa,tenkhoa,COUNT(masv)
     FROM
    WHERE khoa=@khoa
RETURN /*Trả kết quả về cho hàm*/
END
```

Trigger

- Khái niệm
  - *Trigger* là một loại *stored procedure* nó được định nghĩa để thực thi khi có một câu lệnh Update, Insert, hoặc Delete được phát sinh trên bảng.
  - *Trigger* là một công cụ mạnh mà nó có thể dùng để ràng buộc các qui tắc quản lý một cách tự động khi dữ liệu bị hiệu chỉnh.

Trigger

- Trigger cũng có thể nới rộng tính toàn vẹn kiểm soát logic của SQL Server.
- Trigger tự động thực thi, không thể gọi một trigger thi hành một cách trực tiếp được.

Trigger

- *Hạn chế*: trigger không ngăn ngừa thay đổi cấu trúc. Chỉ quan tâm đến các dòng dữ liệu.

Cơ cấu thực thi:

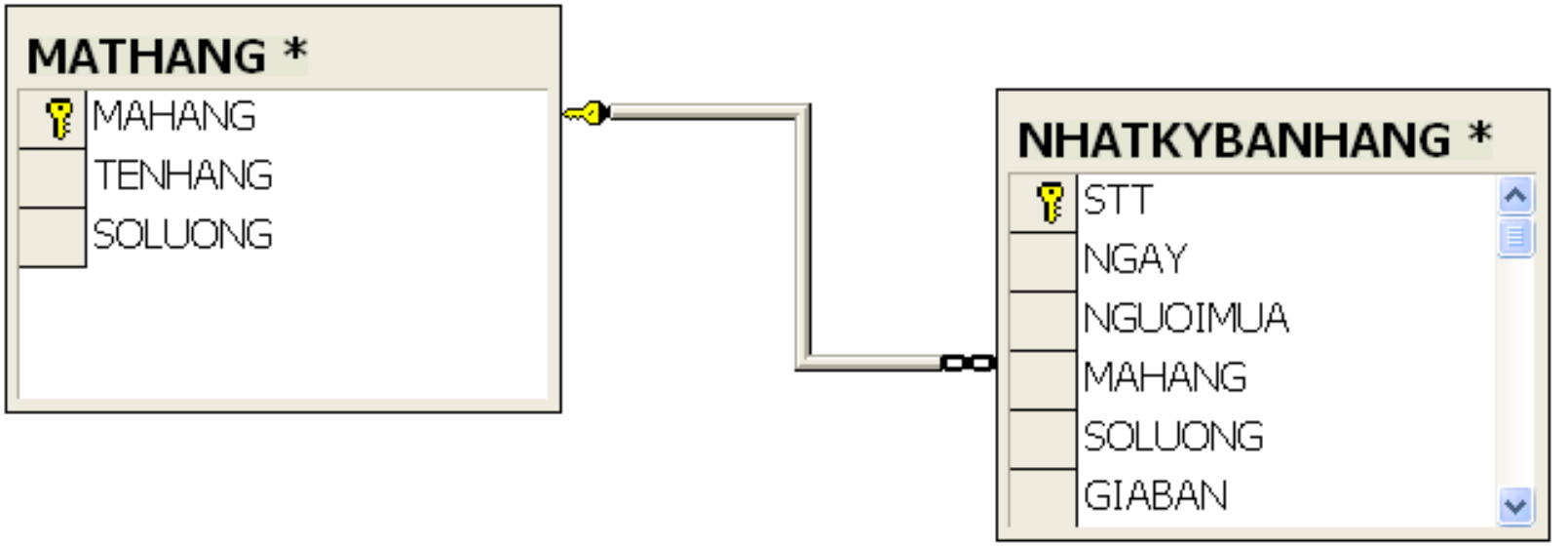
- Insert hoặc Update dữ liệu trigger sẽ được kích hoạt, trigger sẽ lưu dòng dữ liệu mới hoặc dòng dữ liệu đã hiệu chỉnh bảng Inserted vào trong bộ nhớ Cash.
- Khi xóa dữ liệu của bảng trigger sẽ được kích hoạt, trigger sẽ lưu trữ dòng dữ liệu bị xóa vào bảng Deleted.

Trigger

- Tạo trigger:

```
CREATE TRIGGER tên_trigger
ON tên_bảng
FOR
   {[INSERT][,][UPDATE][,][DELETE]}
AS
   các_câu_lệnh_của_trigger ….
```

Trigger


MATHANG *
MAHANG
TENHANG
SOLUONG
NHATKYBANHANG *
STT
NGAY
NGUOIMUA
MAHANG
SOLUONG
GIABAN

Trigger

```
CREATE TRIGGER trg_nhatkybanhang_insert
ON nhatkybanhang
FOR INSERT
AS
UPDATE mathang
SET mathang.soluong= mathang.soluong-
    inserted.soluong //tự động giảm số lượng tại bảng mahang
FROM mathang INNER JOIN inserted
ON mathang.mahang = inserted.mahang
```

Trigger

```
INSERT INTO nhatkybanhang
  (ngay,nguoimua,mahang,soluong,giaban)
VALUES ('5/5/2004','Tran Ngoc
   Thanh','H1',10,5200)
```

Kết thúc

CƠ SỞ DỮ LIỆU

Chương 1: Giới thiệu về HCSDL

Các chức năng của hệ QTCSDL

là phần mềm cung cấp một môi trường thuận lợi và hiệu quả để tạo lập, lưu trữ và tìm kiếm thông tin.

Chương 1: Giới thiệu về HCSDL

- Hệ QTCSDL cùng với cơ sở dữ liệu được gọi chung là Hệ *cơ sở dữ liệu*.
- **Quản trị CSDL (Database Management System)**

Chương 1: Giới thiệu về HCSDL

Các chức năng cơ bản của hệ quản trị CSDL :

a) Cung cấp cách tạo lập cơ sở dữ liệu.

Người dùng sử dụng *ngôn ngữ định nghĩa dữ liệu* để khai báo kiểu và cấu trúc của dữ liệu, khai báo các ràng buộc trên dữ liệu được lưu trữ trong CSDL.

Chương 1: Giới thiệu về HCSDL

b) Cung cấp các cập nhật dữ liệu, tìm kiếm và kết xuất thông tin. Người dùng sử dụng *ngôn ngữ thao tác dữ liệu* để diễn tả yêu cầu cập nhật (nhập, sửa, xóa dữ liệu) hay tìm kiếm, kết xuất thông tin (xuất dữ liệu chứa trong CSDL theo những điều kiện tìm kiếm cụ thể).

Chương 1: Giới thiệu về HCSDL

c) Cung cấp công cụ kiểm soát, điều khiển việc truy cập vào CSDL :

Hệ quản trị CSDL đảm bảo :

- Phát hiện và ngăn chặn truy cập trái phép
- Duy trì tính nhất quán của dữ liệu.
- Tổ chức, điều khiển các truy cập cùng lúc.
- Khôi phục CDSL khi gặp sự cố.
- Quản lí các mô tả dữ liệu.

Chương 1: Giới thiệu về HCSDL

Hoạt động của một hệ CSDL:

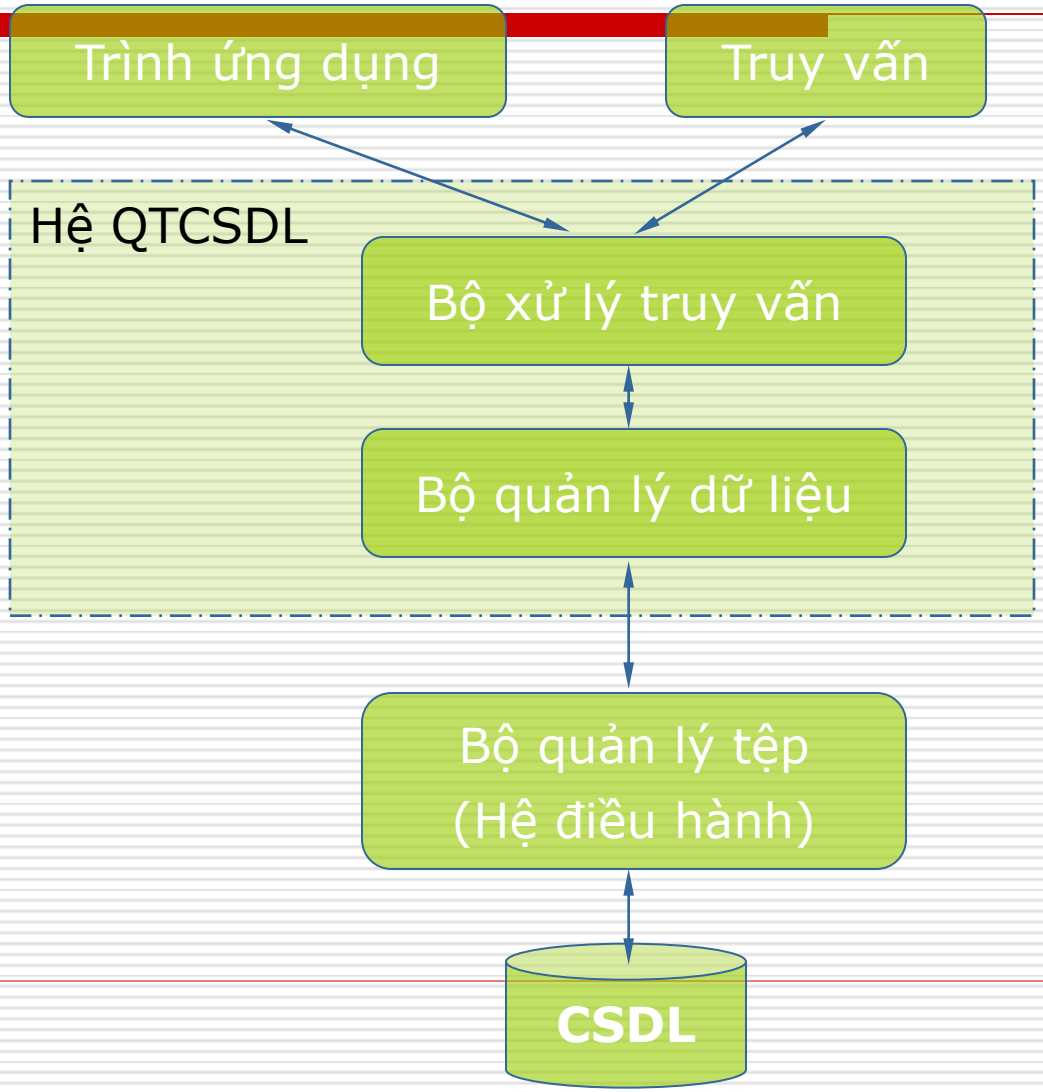
Một hệ QTCSDL gồm 2 thành phần chính :

- Bộ xử lí truy vấn (xử lí các yêu cầu).
- Bộ quản lí dữ liệu.

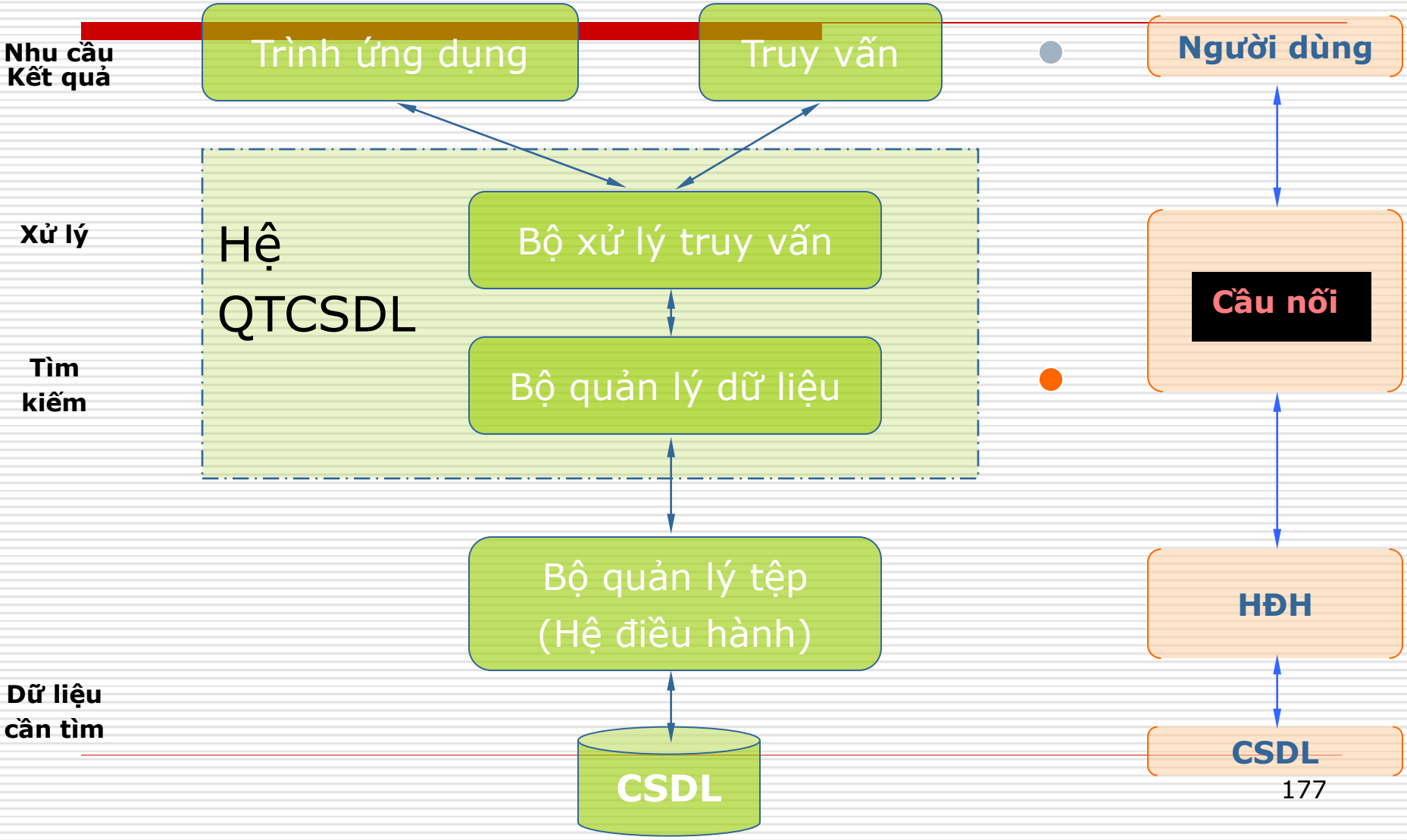
Chương 1: Giới thiệu về HCSDL

Hệ QTCSDL đóng vai trò cầu nối giữa người dùng (các chương trình ứng dụng, các truy vấn) với cơ sở dữ liệu (được quản lý bởi hệ điều hành).

Chương 1: Giới thiệu về HCSDL



Chương 1: Giới thiệu về HCSDL



Chương 1: Giới thiệu về HCSDL

Vai trò của con người khi làm việc với các hệ cơ sở dữ liệu

a) Người quản trị cơ sở dữ liệu (administrator).

- Administrator (admin) được hiểu là một người (hoặc một nhóm người) được trao quyền quản lý CSDL.

Chương 1: Giới thiệu về HCSDL

- Admin có vai trò thiết kế, cài đặt, cấp phát phần mềm, phần cứng, phân quyền người dùng … nhằm đảm bảo hệ thống hoạt động ổn định, đáp ứng các yêu cầu của các trình ứng dụng, của người dùng.

Chương 1: Giới thiệu về HCSDL

b) Người lập trình ứng dụng.

- Người lập trình ứng dụng có nhiệm vụ viết các chương trình dựa trên hệ quản trị CSDL nhằm đáp ứng nhu cầu khai thác của các nhóm người dùng cụ thể.

Chương 1: Giới thiệu về HCSDL

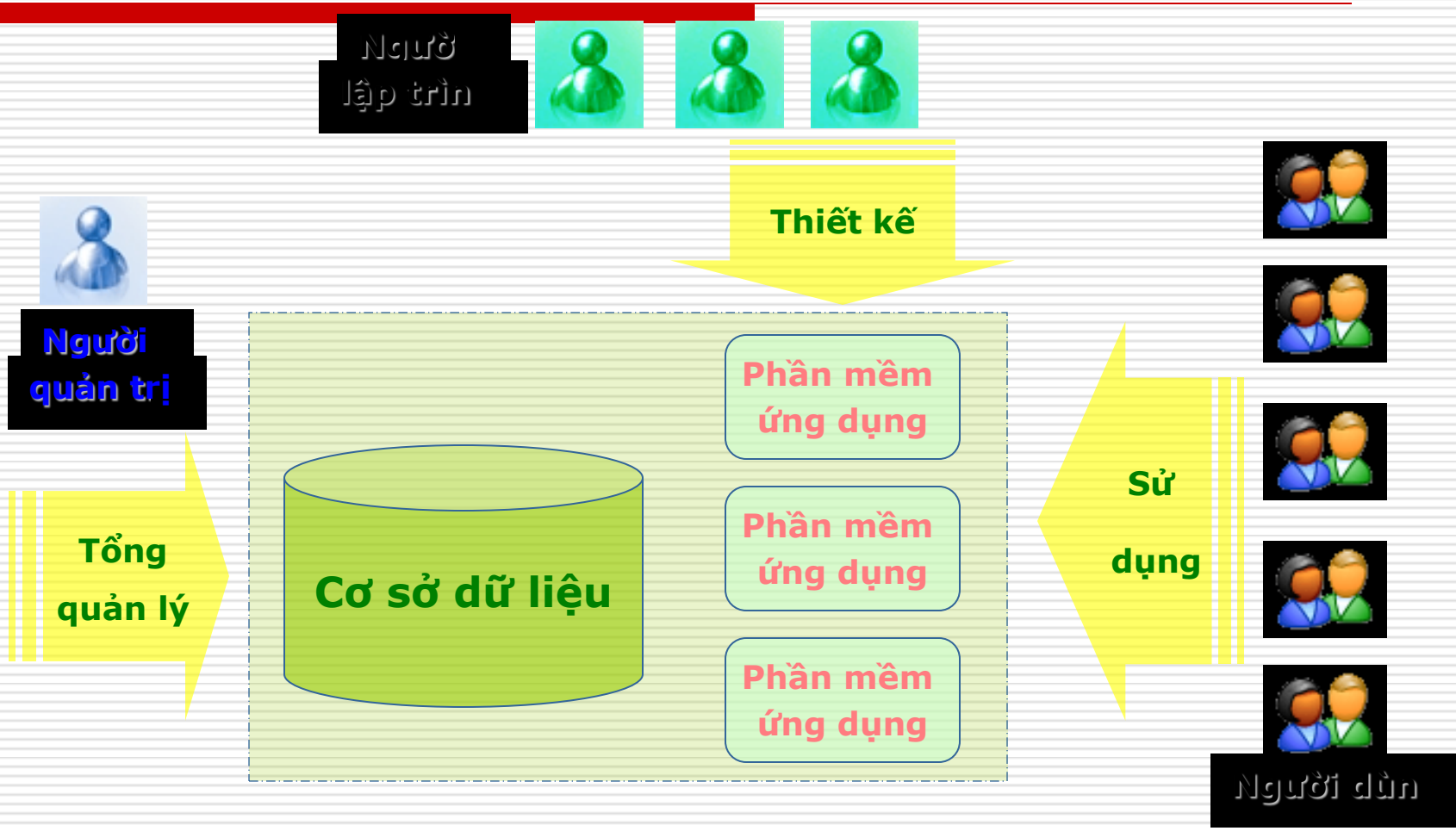
c) Người dùng. (User)

- Người dùng (còn gọi là người dùng đầu cuối) là các khách hàng có nhu cầu khai thác thông tin từ CSDL thông qua việc sử dụng các chương trình đã được viết trước.

Chương 1: Giới thiệu về HCSDL

- Người dùng được phân thành từng nhóm (user group) với các quyền hạn nhất định để truy cập và khai thác CSDL.

Chương 1: Giới thiệu về HCSDL



Chương 1: Giới thiệu về HCSDL

1) Vì sao hệ quản trị CSDL lại phải có khả năng kiểm soát và điều khiển các truy cập đến CSDL? Hãy nêu ví dụ để minh họa cho giải thích.
2) Khi làm việc với các hệ CSDL, anh (chị) muốn giữ vai trò gì (người quản trị CSDL, người lập trình ứng dụng hay người dùng)? Vì sao?
3) Hãy trình bày sơ lược về hoạt động của một hệ quản trị CSDL.

Chương 1: Giới thiệu về HCSDL

- Tiếp cận cơ sở dữ liệu theo cách mới đã giải quyết được một số vấn đề của phương pháp cũ.
  - Cấu trúc logic và cấu trúc vật lý- tiếp cận này tạo cho người quản trị CSDL thay đổi cấu trúc vật lý hay nơi lưu trữ mà không ảnh hưởng tới ứng dụng.
  - Vấn đề dư thừa dữ liệu
  - Sự khai thác dữ liệu của NSD

Chương 2: Mô hình liên kết thực thể

- Giới thiệu: các nhà phân tích xây dựng lược đồ CSDL từ mô hình liên kết thực thể. Mô hình này lại được xây dựng từ bài toán thực tế.

Chương 2: Mô hình liên kết thực thể

- Xét phân tích yêu cầu của Công Ty:

  Công ty có nhiều nhân viên:
  - Một nhân viên chỉ được làm trong một phòng ban
  - Một nhân viên có thể tham gia nhiều dự án

  Một nhân viên có nhiều thân nhân:
  - Một thân nhân có tên tuổi địa chỉ và đặc biệt là quan hệ với nhân viên.

Chương 2: Mô hình liên kết thực thể

Công ty có nhiều phòng ban:

- Một phòng có duy nhất một tên, một mã số và một trưởng phòng.
- Cần lưu lại thời gian nhận chức của trưởng phòng.

Công ty thực hiện nhiều dự án:

- Một dự án có một tên, mã số duy nhất và địa điểm triển khai.
- Một phòng có thể giám sát nhiều dự án

Chương 2: Mô hình liên kết thực thể

Yêu cầu:

- Muốn lưu số ngày tham gia dự án của nhân viên trong 1 tuần.
- Lưu trưởng phòng của nhân viên đó

Chương 2: Mô hình liên kết thực thể

- Thực thể: là đối tượng hoặc khái niệm trong thế giới thực. Có nhiều thuộc tính.→những gì có thể phân biệt được.
  - VD: nhân viên Nguyễn Văn A, dự án X.
- Thuộc tính: dùng để mô tả thực thể

Tên của một phòng, giới tính của một nhân viên

Mỗi thuộc tính của một thực thể xác định có một giá trị cụ thể, nó là một giá trị được lưu giữ.

Mỗi thuộc tính có một kiểu dữ liệu xác định

Chương 2: Mô hình liên kết thực thể

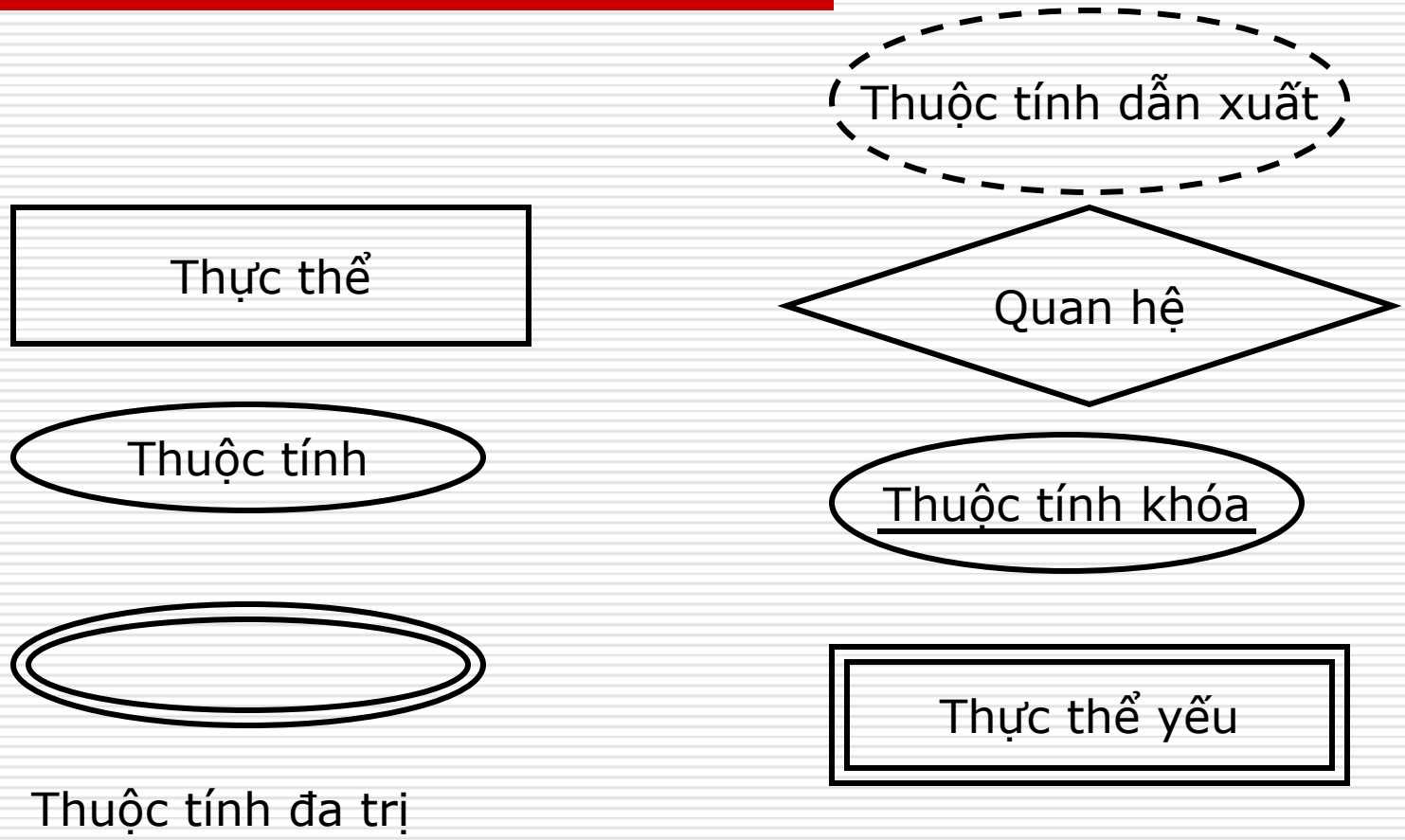
Các loại thuộc tính

- Thuộc tính đơn (nguyên tử-GT)
- Thuộc tính gộp (Họ, tên)
- Thuộc tính đơn trị (MSNV)
- Thuộc tính đa trị (Sở thích)
- Thuộc tính cơ sở
- Thuộc tính dẫn xuất

Chương 2: Mô hình liên kết thực thể

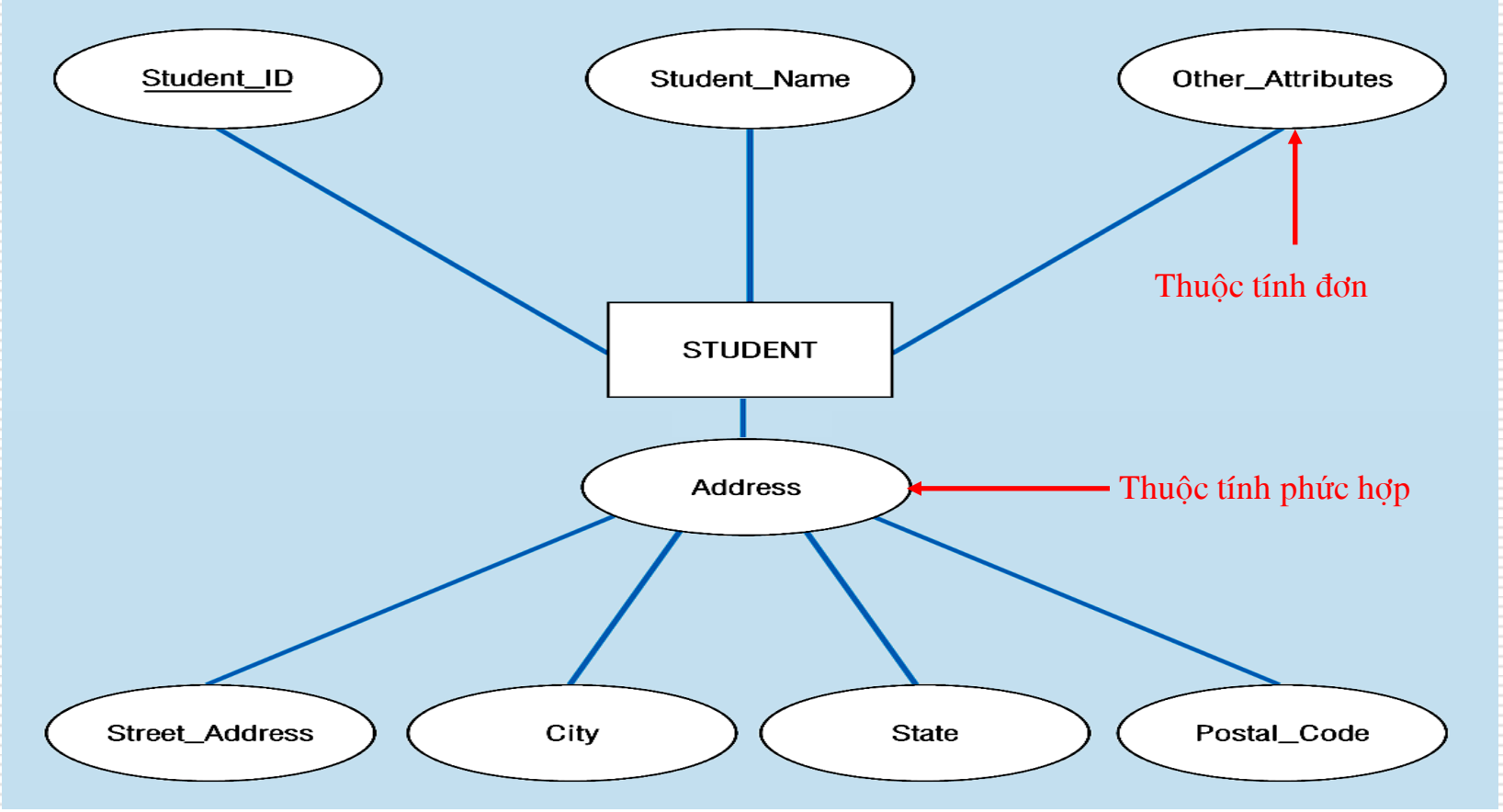
- Tập thực thể: là một tập các thực thể cùng kiểu
- Quan hệ (liên kết) là quan hệ giữa một hoặc nhiều tập thực thể.

Chương 2: Mô hình liên kết thực thể

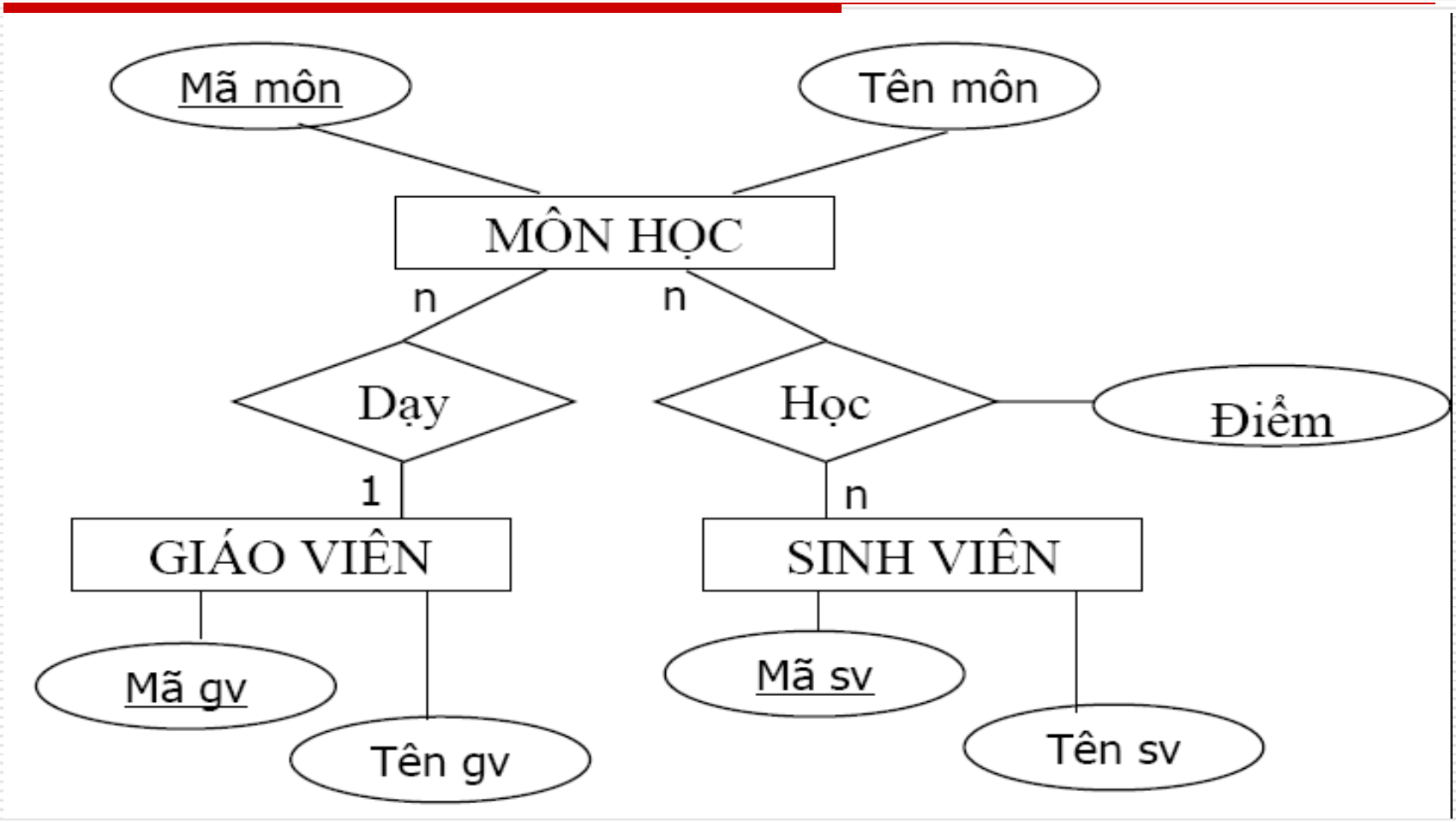


- Sơ đồ ER:

Chương 2: Mô hình liên kết thực thể



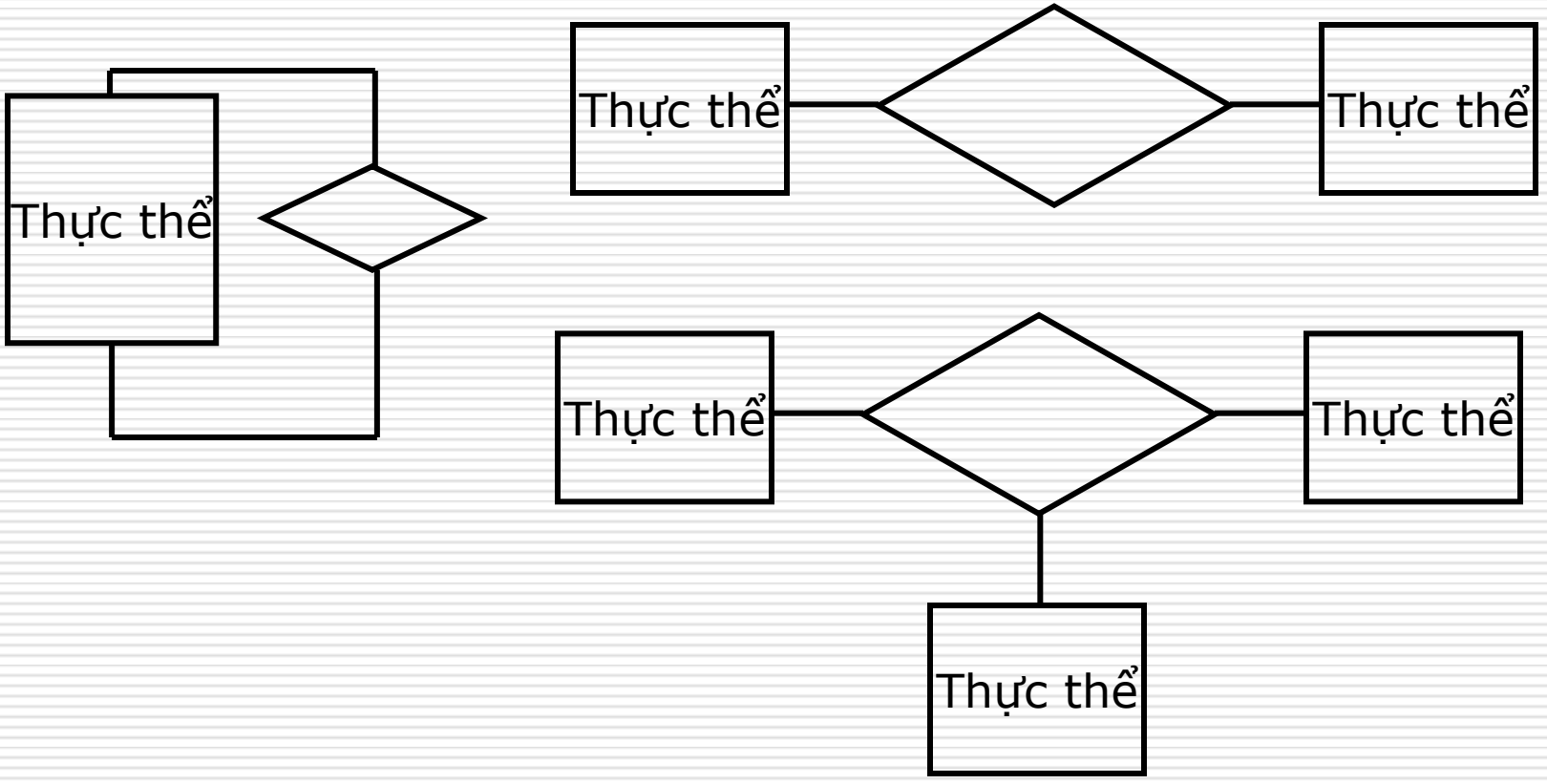
Chương 2: Mô hình liên kết thực thể



Chương 2: Mô hình liên kết thực thể

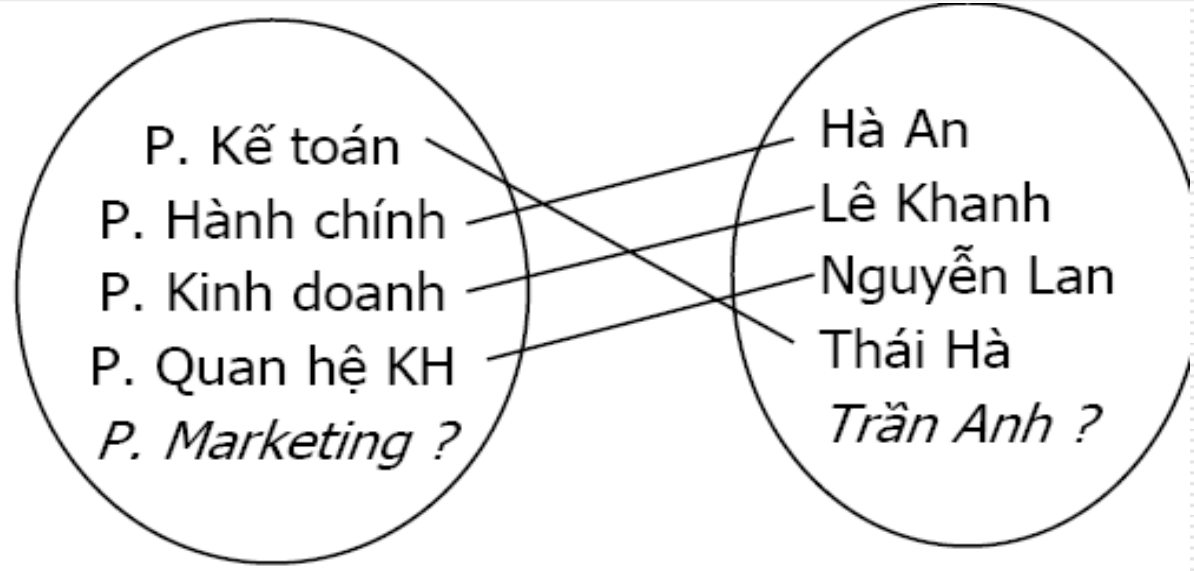
- Ngôi bậc quan hệ: là số các tập thực thể tham gia vào một quan hệ
- Bậc của quan hệ có thể là: đơn phân, nhị phân, tam phân,..., n phân.

Chương 2: Mô hình liên kết thực thể



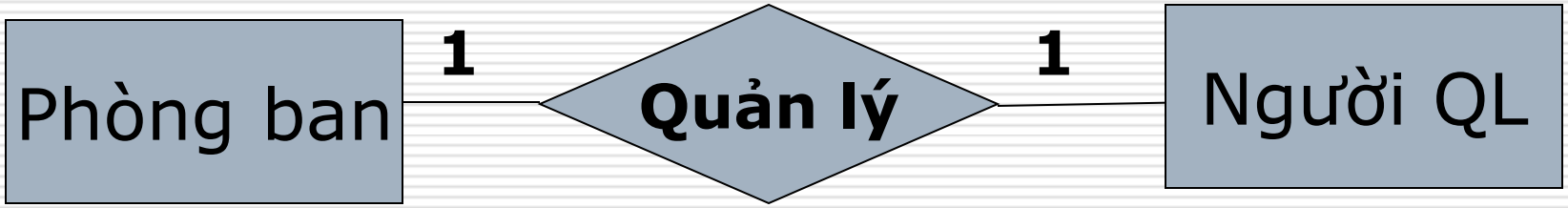
Chương 2: Mô hình liên kết thực thể

- Quan hệ 1-1: Mỗi tập thực thể trong E1 chỉ quan hệ với nhiều nhất một thực thể trong tập E2.



Chương 2: Mô hình liên kết thực thể

- Một người quản lý chỉ được quản lý một phòng ban.
- Một phòng ban chỉ có thể được quản lý bởi một người.

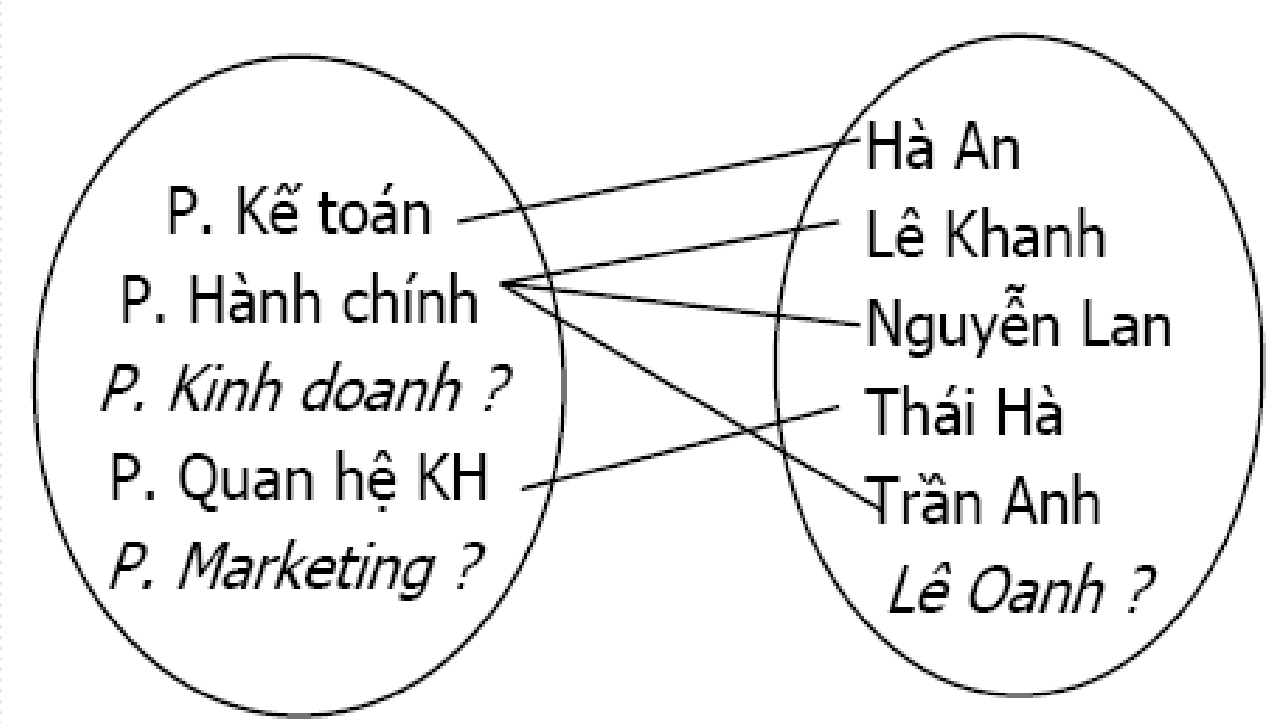


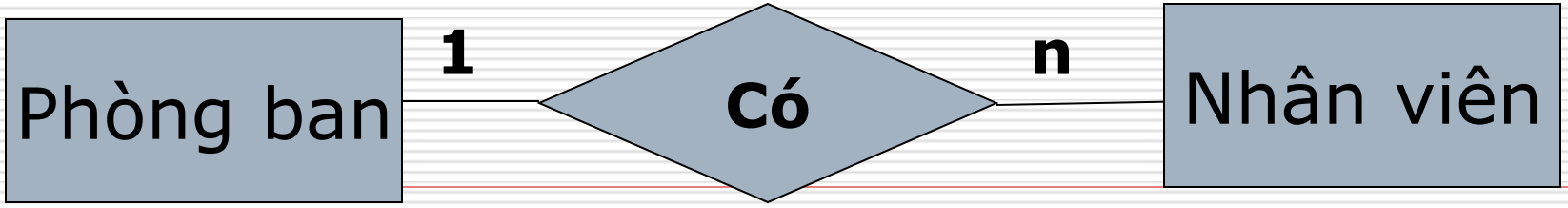
Chương 2: Mô hình liên kết thực thể

- Quan hệ 1-n/n-1: Một thực thể trong E1 quan hệ với một hoặc nhiều thực thể trong tập E2.

Một phòng ban có nhiều nhân viên nhưng một nhân viên chỉ làm việc cho một phòng ban.

Chương 2: Mô hình liên kết thực thể



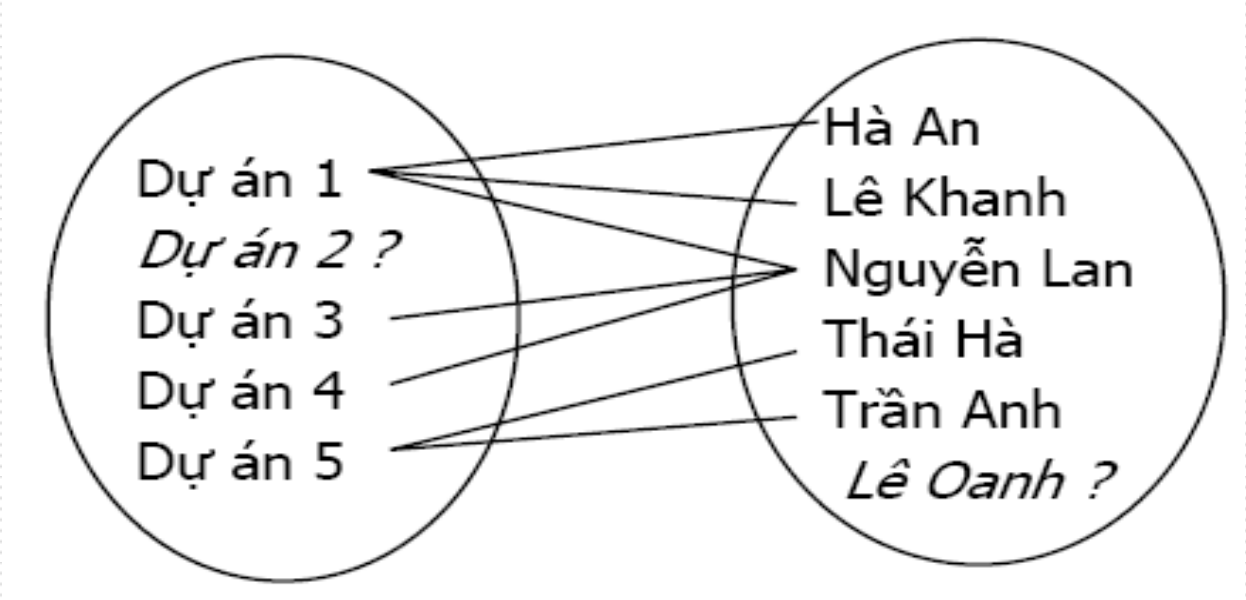


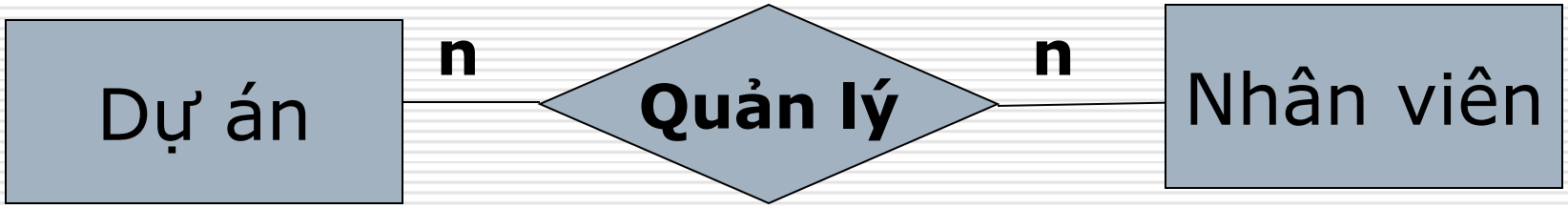
Chương 2: Mô hình liên kết thực thể

Quan hệ n-n: một thực thể liên kết với nhiều thực thể trong tập thực thể khác và ngược lại.

Một nhân viên có thể tham gia nhiều dự án và một dự án cần nhiều nhân viên tham gia.

Chương 2: Mô hình liên kết thực thể





Chương 3: Mô hình dữ liệu quan hệ

- *Quan hệ* (*relation*) là một bảng dữ liệu hai chiều bao gồm nhiều hàng (mẩu tin) và nhiều cột (thuộc tính hoặc vùng tin).
  - Mỗi hàng là duy nhất: không thể có hai hàng có cùng các giá trị ở tất cả vùng tin.
  - Thứ tự của các hàng là không quan trọng.
  - Thứ tự của các cột là không quan trọng.
  - Không phải mọi bảng đều là quan hệ. Quan hệ là một bảng không chứa các hàng giống hệt nhau.

Chương 3: Mô hình dữ liệu quan hệ

Quan hệ: Nhân viên

MNV	Tên nhân viên	Địa chỉ
S1	Nguyễn Trung Tiến	Quảng Phú
S2	Trần Thị Yến	Chánh Lộ
S3	Nguyễn Văn An	Nghĩa Điền

Chương 3: Mô hình dữ liệu quan hệ

- Quan hệ chứa dữ liệu của một tập thực thể hoặc một tập liên kết

Chương 3: Mô hình dữ liệu quan hệ

- Khóa
  - *Khóa quan hệ* là một tập nhỏ nhất các thuộc tính dùng để xác định duy nhất một hàng.
  - Một khóa chỉ có một thuộc tính được gọi là *khóa đơn* (*simple key*).
  - Một khóa có nhiều thuộc tính được gọi là *khóa phức hợp* (*composite key*).
  - Khóa thường được sử dụng làm *chỉ mục* (*index*) của bảng dữ liệu để làm tăng tốc độ xử lý câu truy vấn.

Chương 3: Mô hình dữ liệu quan hệ

- Khóa
  - Một quan hệ phải có ít nhất một khóa và có thể có nhiều khóa.
  - Các thuộc tính thuộc một khóa được gọi là *thuộc tính khóa* (*prime attribute*), các thuộc tính còn lại trong lược đồ quan hệ được gọi là các *thuộc tính không khóa* (*nonprime attribute*).
  - Các thuộc tính khóa được gạch dưới.
  - Các thuộc tính khóa không được có giá trị rỗng (*null value*).

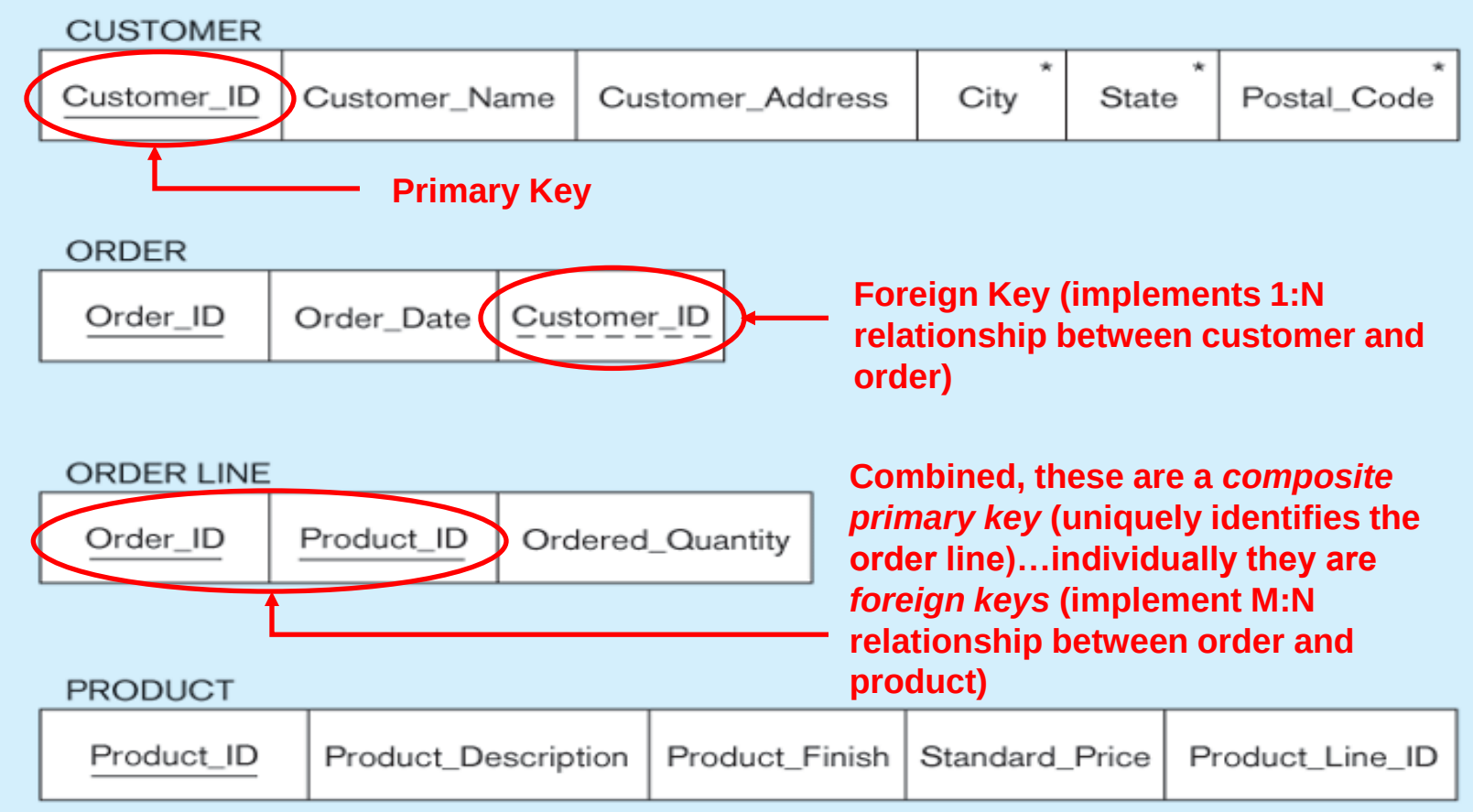
Chương 3: Mô hình dữ liệu quan hệ

- Khóa
  - Tất cả các khóa của một quan hệ được gọi là *khóa dự tuyển* (*candidate key*).
  - Một trong các khóa dự tuyển được chọn làm khóa tiêu biểu, khóa này được gọi là *khóa chính* (*primary key*).
  - Một quan hệ chỉ có một khóa chính và có thể có nhiều khóa dự tuyển.
  - Trong một quan hệ, một hoặc nhiều thuộc tính được gọi là *khóa ngoại* (*foreign key*) nếu chúng là khóa chính của một quan hệ khác.

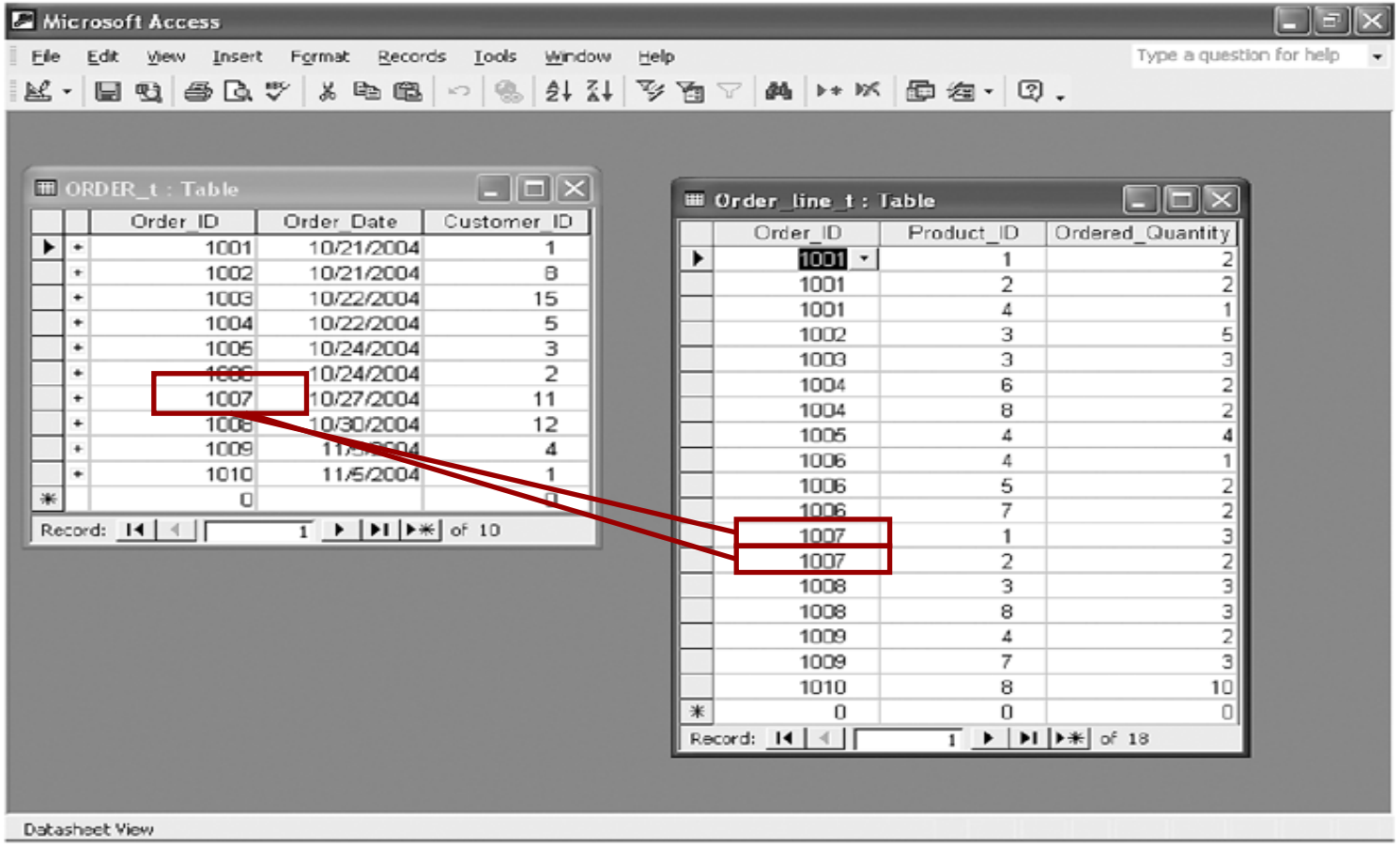
Chương 3: Mô hình dữ liệu quan hệ

- Cơ sở dữ liệu quan hệ (*relational database*) bao gồm các bảng (quan hệ) biểu diễn các thực thể và các khóa chính / khóa ngoại biểu diễn các mối liên kết.

Chương 3: Mô hình dữ liệu quan hệ



Chương 3: Mô hình dữ liệu quan hệ



Chương 3: Mô hình dữ liệu quan hệ

- Mỗi quan hệ (bảng) tương ứng với một kiểu thực thể hoặc với một kiểu mối liên kết nhiều - nhiều.
- Mỗi hàng tương ứng với một thể hiện thực thể hoặc với một thể hiện mối liên kết nhiều - nhiều.
- Mỗi cột tương ứng với một thuộc tính.
- Từ *quan hệ* (*relation*) trong cơ sở dữ liệu quan hệ không có cùng nghĩa với từ *mối quan hệ* (*relationship*) trong mô hình *ER*.

Chương 3: Mô hình dữ liệu quan hệ

- Lược đồ cơ sở dữ liệu
  - *database schema*
  - *Lược đồ cơ sở dữ liệu* là một tập hợp các lược đồ quan hệ.
  - Trong một lược đồ cơ sở dữ liệu, các tên lược đồ quan hệ là duy nhất.
  - **Lược đồ cơ sở dữ liệu:**
  - ***Emp (Empnum, Name, Sal, Tax, Mgrnum, Deptnum)***
  - ***Dept (Deptnum, Name, Area, Mgrnum)***
  - ***Supplier (Snum, Name, City)***
  - ***Supply (Snum, Pnum, Deptnum, Quan)***

Chương 3: Mô hình dữ liệu quan hệ

- Ràng buộc toàn vẹn
  - *integrity constraint*
  - *Ràng buộc toàn vẹn* là một qui tắc mà tất cả các dữ liệu trong CSDL phải thỏa mãn qui tắc này.
- Ràng buộc miền trị
  - *domain constraint*
  - Các giá trị cho phép của một thuộc tính.
- Toàn vẹn thực thể
  - *entity integrity*
  - Thuộc tính khóa chính không có giá trị rỗng (*null value*).

Chương 3: Mô hình dữ liệu quan hệ

Attribute	*Domain Name*	*Description*	*Domain*
Customer_ID	Customer_IDs	Set of all possible customer IDs	character: size 5
Customer_Name	Customer_Names	Set of all possible customer names	character: size 25
Customer_Address	Customer_Addresses	Set of all possible customer addresses	character: size 30
City	Cities	Set of all possible cities	character: size 20
State	States	Set of all possible states	character: size 2
Postal_Code	Postal_Codes	Set of all possible postal zip codes	character: size 10
Order_ID	Order_IDs	Set of all possible order IDs	character: size 5
Order_Date	Order_Dates	Set of all possible order dates	date format mm/dd/yy
Product_ID	Product_IDs	Set of all possible product IDs	character: size 5
Product_Description	Product_Descriptions	Set of all possible product descriptions	character size 25
Product_Finish	Product_Finishes	Set of all possible product finishes	character: size 15
Standard_Price	Unit_Prices	Set of all possible unit prices	monetary: 6 digits
Product_Line_ID	Product_Line_IDs	Set of all possible product line IDs	integer: 3 digits
Ordered_Quantity	Quantities	Set of all possible ordered quantities	integer: 3 digits

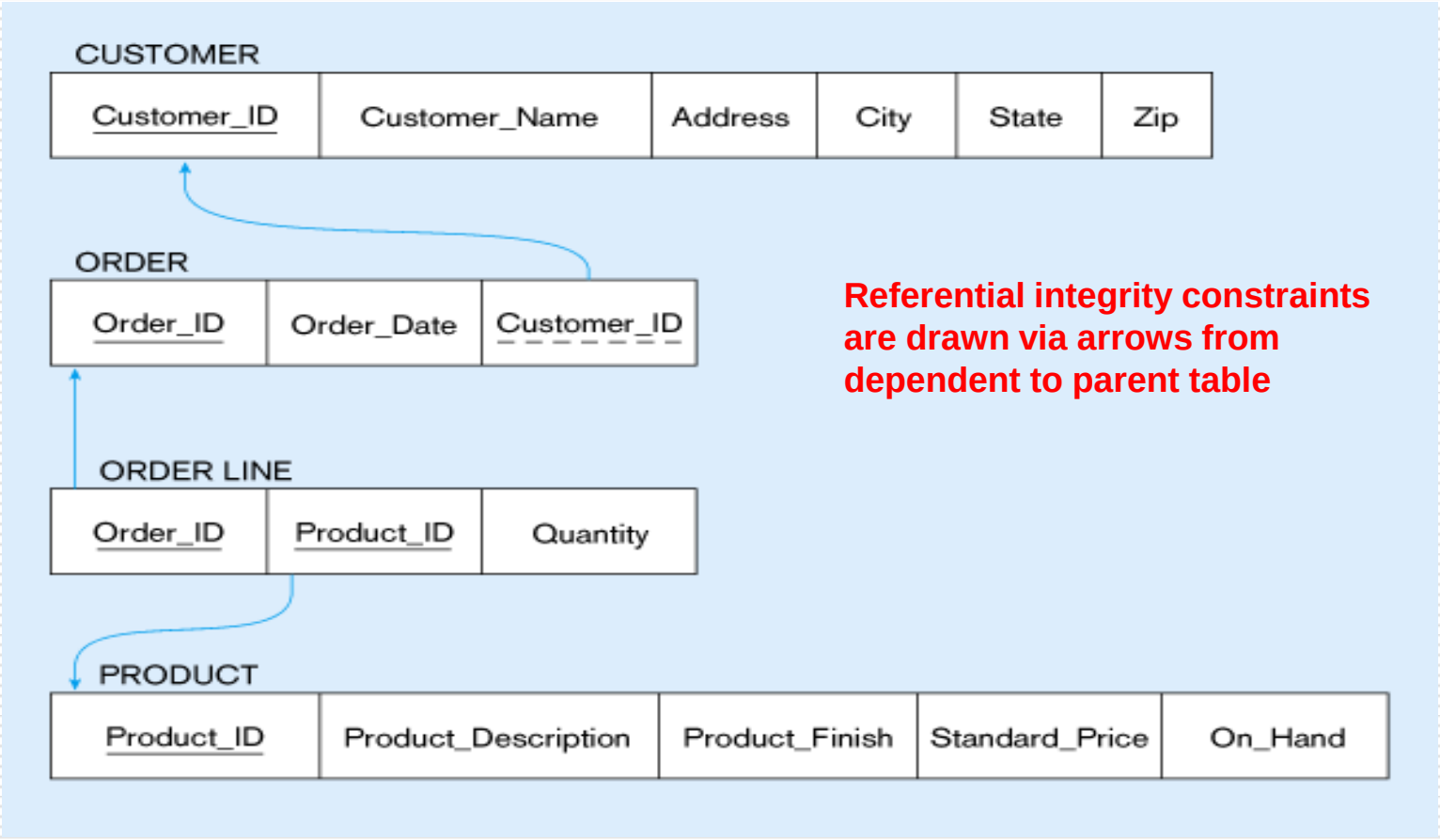
Chương 3: Mô hình dữ liệu quan hệ

- Ràng buộc toàn vẹn tham chiếu
  - *referential integrity constraint*
  - *Ràng buộc toàn vẹn tham chiếu* là một qui tắc mà tất cả các giá trị của khóa ngoại (nếu khác *null*) trong quan hệ bên phía *nhiều* phải có trong các giá trị của khóa chính trong quan hệ bên phía *một*.

Chương 3: Mô hình dữ liệu quan hệ

- Ràng buộc toàn vẹn tham chiếu
  - Qui tắc xóa các hàng dữ liệu
    - Hạn chế (*restrict*): không cho phép xóa các hàng bên phía cha (*parent*) nếu tồn tại các hàng liên quan bên phía phụ thuộc (*dependent*).
    - Tầng (*cascade*): tự động xóa các hàng bên phía phụ thuộc tương ứng với các hàng bên phía cha.
    - Gán null (*set-to-null*): gán *null* cho khóa ngoại của các hàng bên phía phụ thuộc tương ứng với các hàng bên phía cha. Không áp dụng cho các thực thể yếu.

Chương 3: Mô hình dữ liệu quan hệ

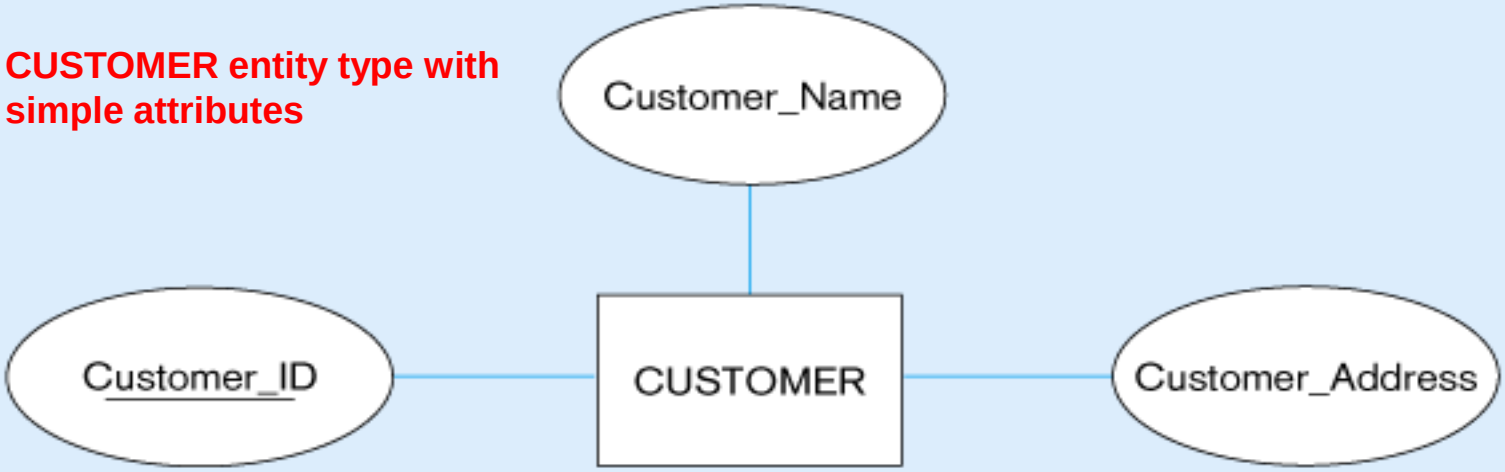


Chương 3: Mô hình dữ liệu quan hệ

- Biến đổi mô hình ER thành quan hệ
- Qui tắc 1: Biến đổi một kiểu thực thể thành một quan hệ.
  - Đối với kiểu thực thể thông thường (*regular entity type*): khóa của quan hệ là khóa của kiểu thực thể.
  - Thuộc tính của quan hệ là thuộc tính của kiểu thực thể.
  - Quan hệ chỉ chứa các thuộc tính thành phần của thuộc tính phức hợp.
  - Quan hệ không chứa các thuộc tính đa trị.

Chương 3: Mô hình dữ liệu quan hệ

CUSTOMER entity type with simple attributes



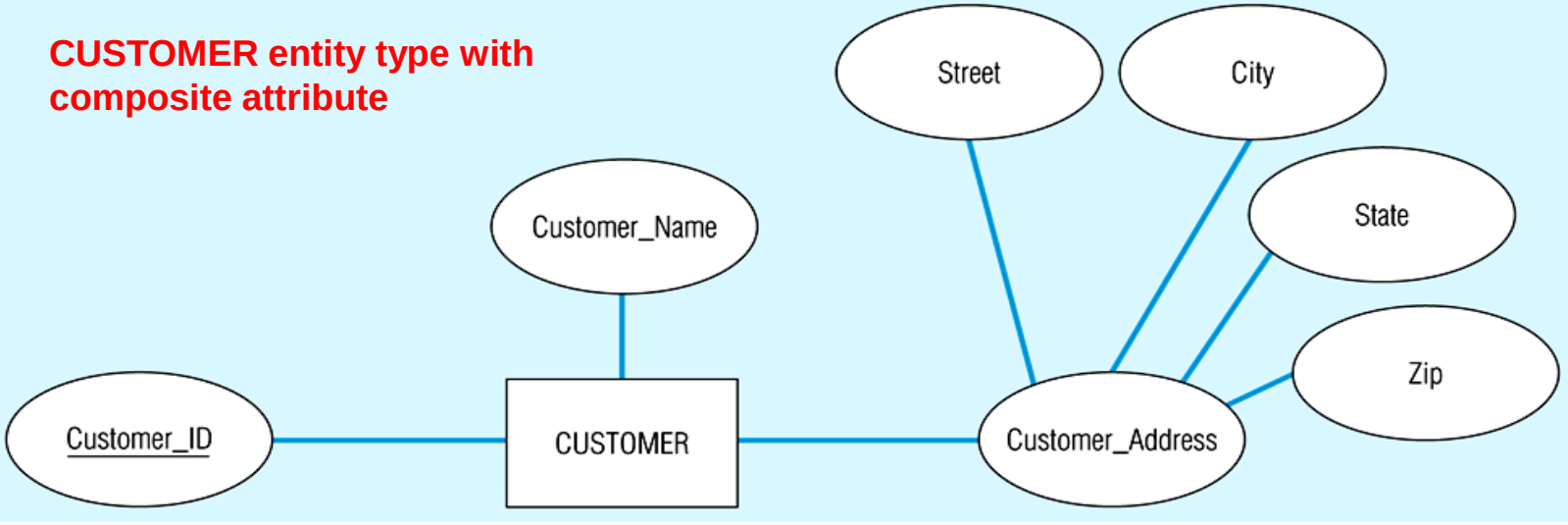
CUSTOMER relation

CUSTOMER

Customer_ID	Customer_Name	Customer_Address

Chương 3: Mô hình dữ liệu quan hệ

CUSTOMER entity type with composite attribute



CUSTOMER relation with address detail

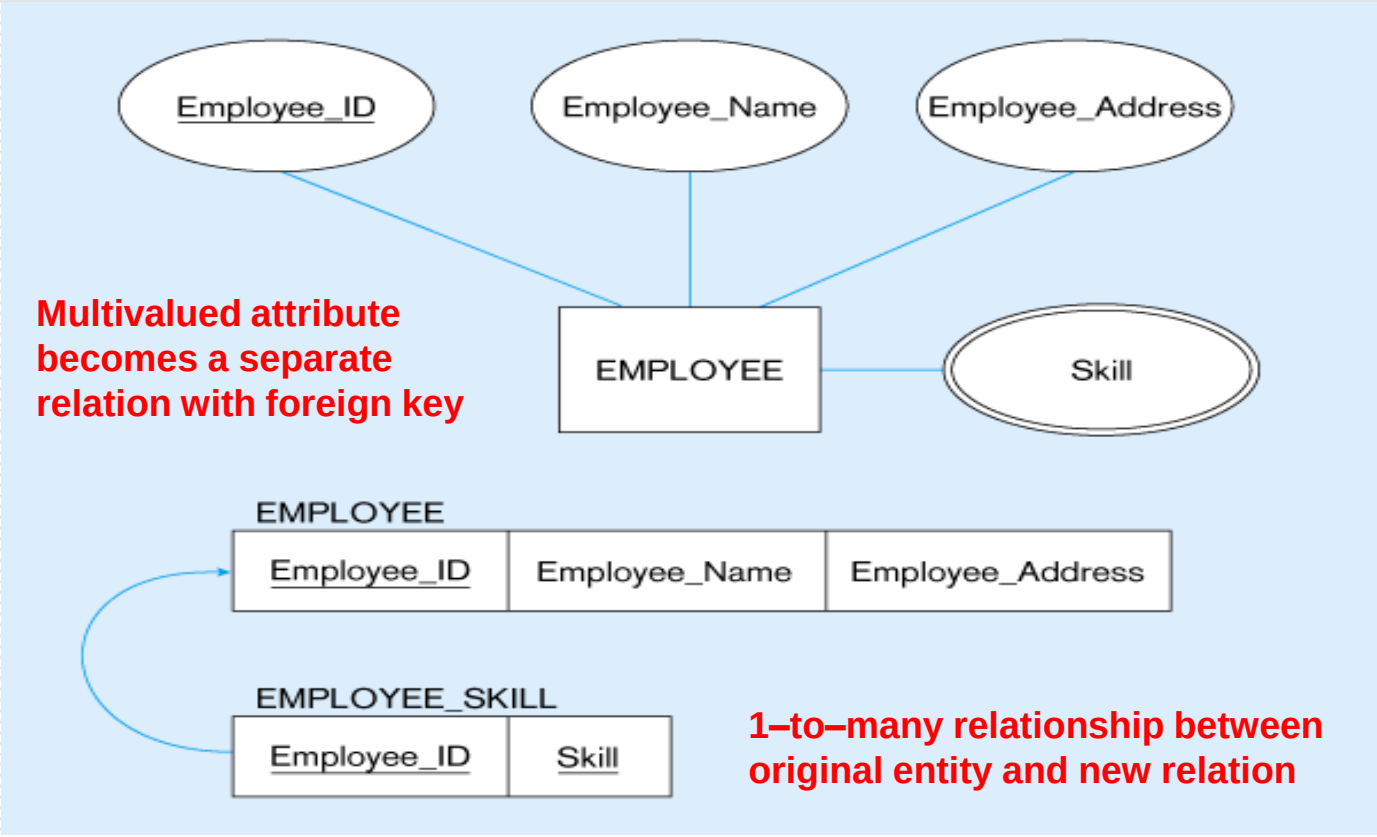
CUSTOMER

Customer_ID	Customer_Name	Street	City	State	Zip

Chương 3: Mô hình dữ liệu quan hệ

- Qui tắc 2: Biến đổi thuộc tính đa trị thành một quan hệ.
  - Quan hệ chứa khóa của kiểu thực thể và thuộc tính đa trị.
  - Khóa của quan hệ gồm khóa của kiểu thực thể và thuộc tính đa trị.

Chương 3: Mô hình dữ liệu quan hệ

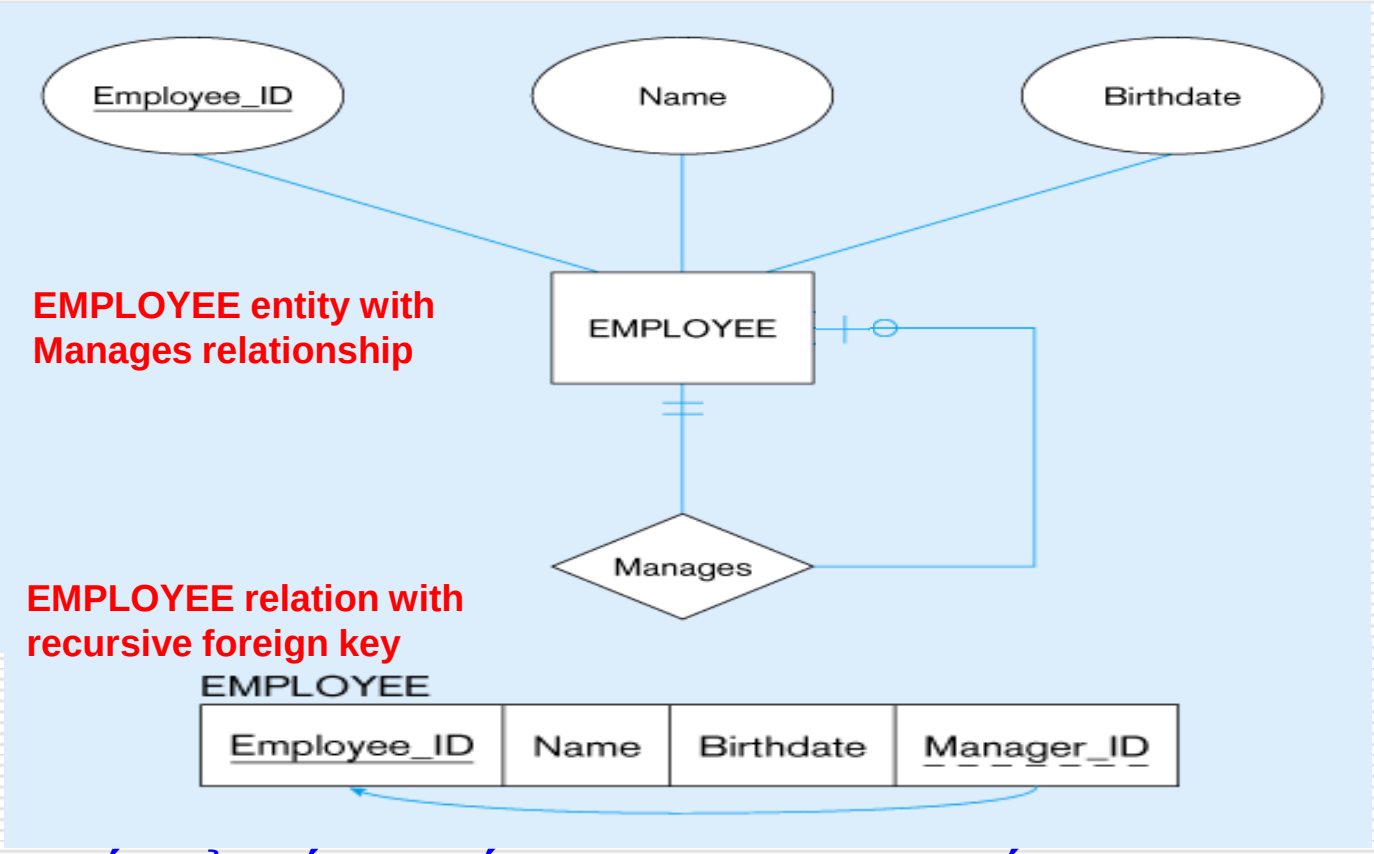


Biến đổi thuộc tính đa trị

Chương 3: Mô hình dữ liệu quan hệ

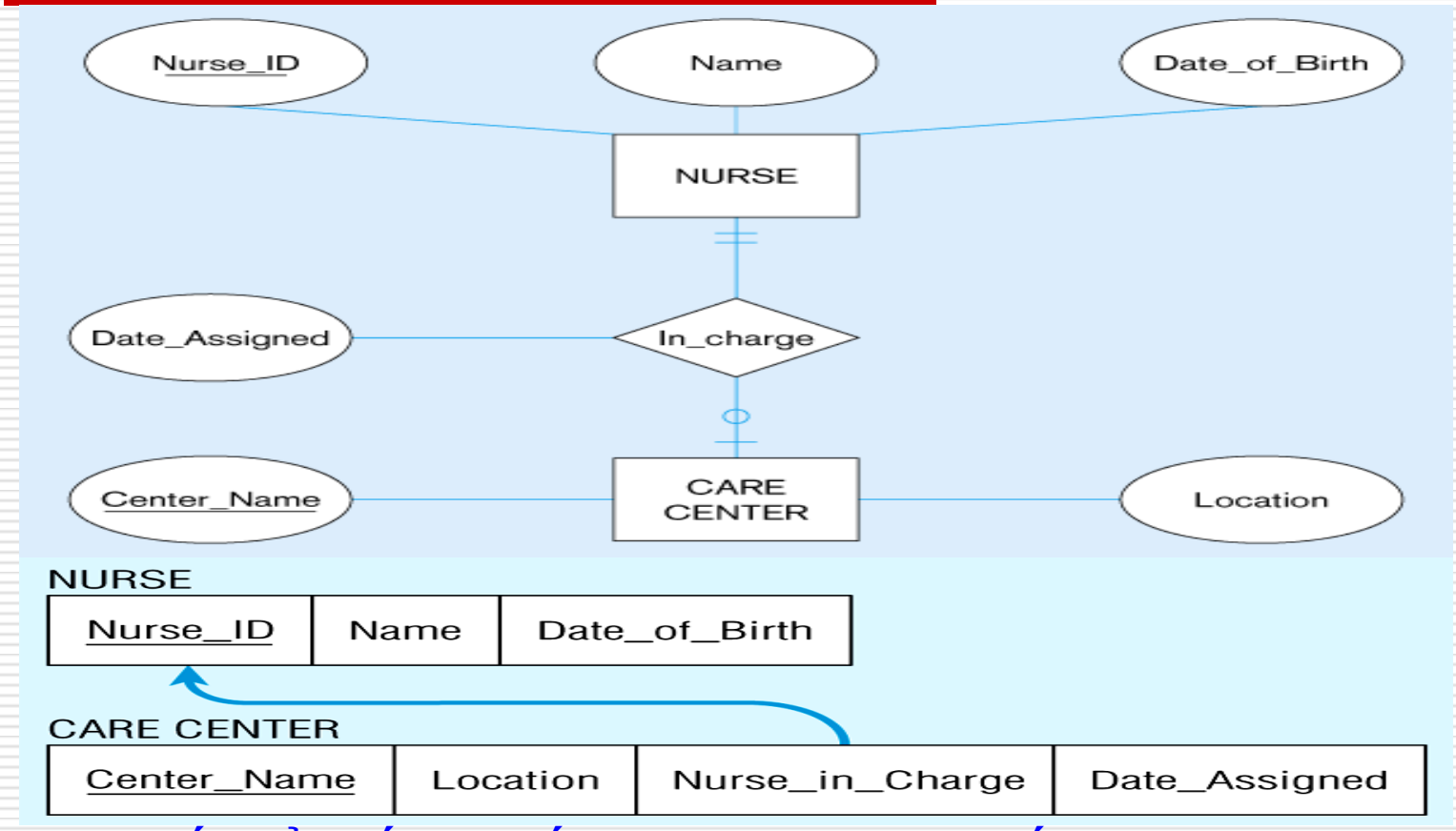
- Qui tắc 3: Biểu diễn mối liên kết 1-ngôi hoặc 2-ngôi có lượng số một-một.
  - Đặt khóa của kiểu thực thể bên phía bắt buộc và các thuộc tính của mối liên kết vào quan hệ của kiểu thực thể bên phía tùy chọn.

Chương 3: Mô hình dữ liệu quan hệ



Biến đổi mối liên kết một ngôi có lượng số một - một

Chương 3: Mô hình dữ liệu quan hệ

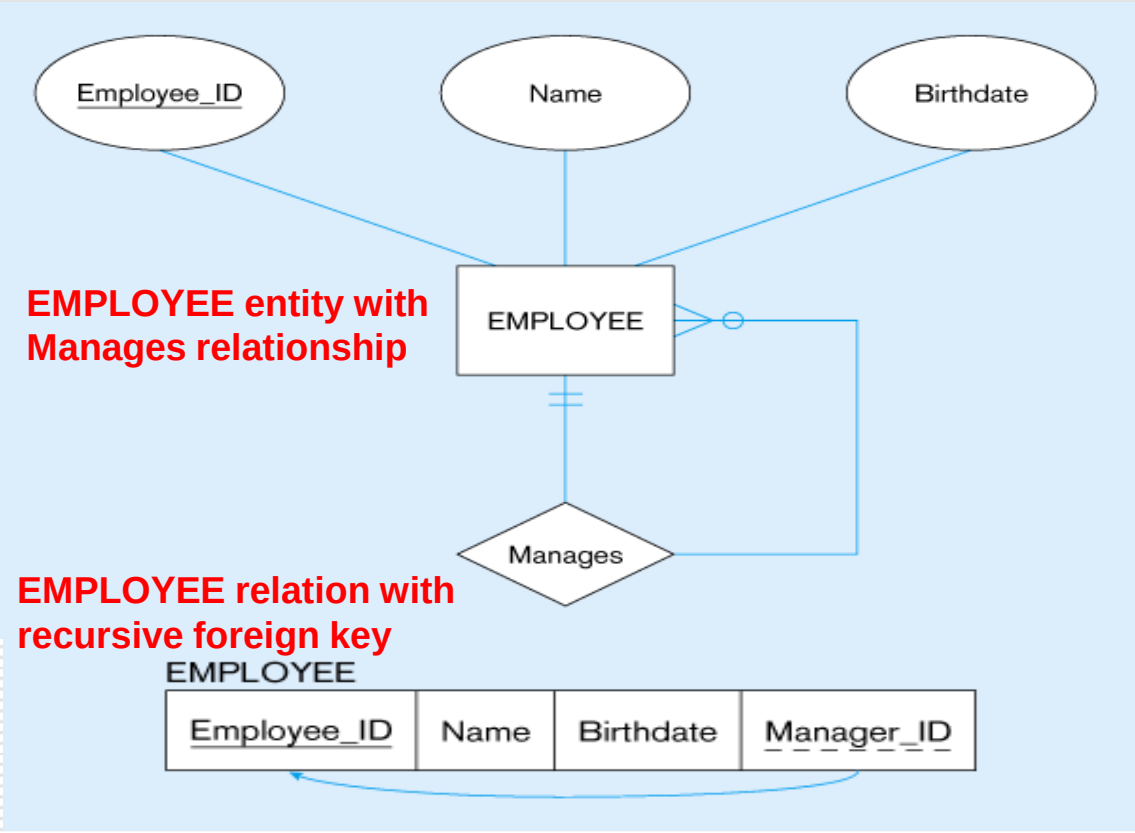


Biến đổi mối liên kết hai ngôi có lượng số một - một

Chương 3: Mô hình dữ liệu quan hệ

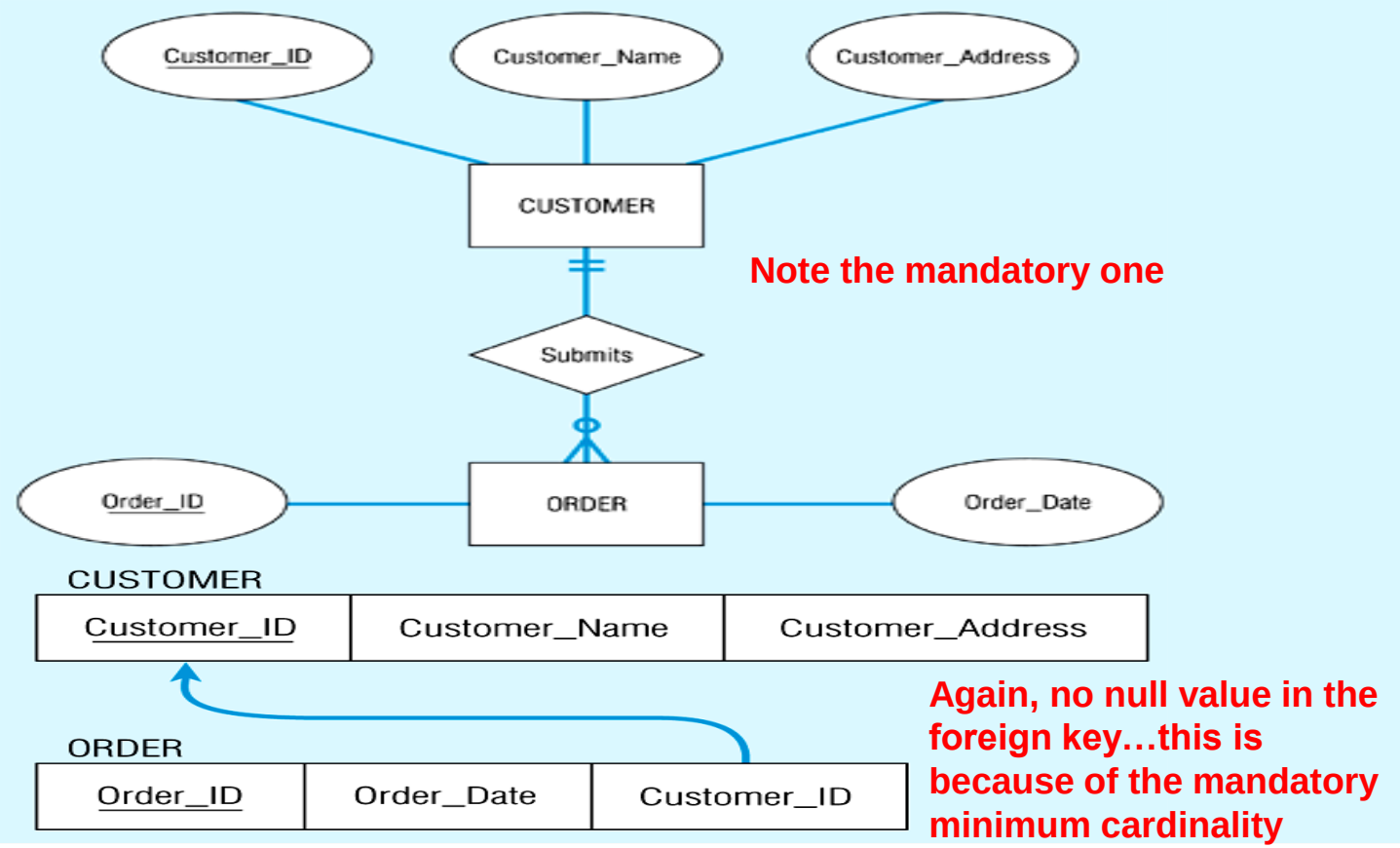
- Qui tắc 4: Biểu diễn mối liên kết 1-ngôi hoặc 2-ngôi có lượng số một-nhiều.
  - Đặt khóa của kiểu thực thể bên phía *một* và các thuộc tính của mối liên kết vào quan hệ của kiểu thực thể bên phía *nhiều*.

Chương 3: Mô hình dữ liệu quan hệ



Biến đổi mối liên kết một ngôi có lượng số một - nhiều

Chương 3: Mô hình dữ liệu quan hệ

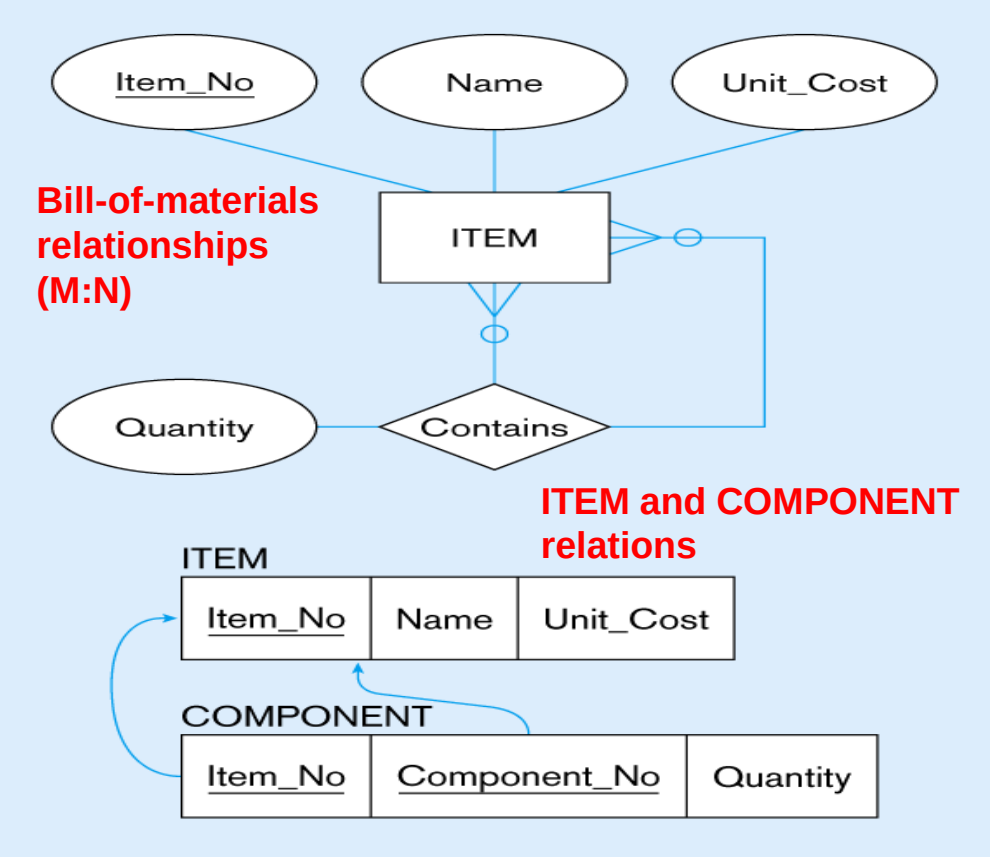


Biến đổi mối liên kết hai ngôi có lượng số một - nhiều

Chương 3: Mô hình dữ liệu quan hệ

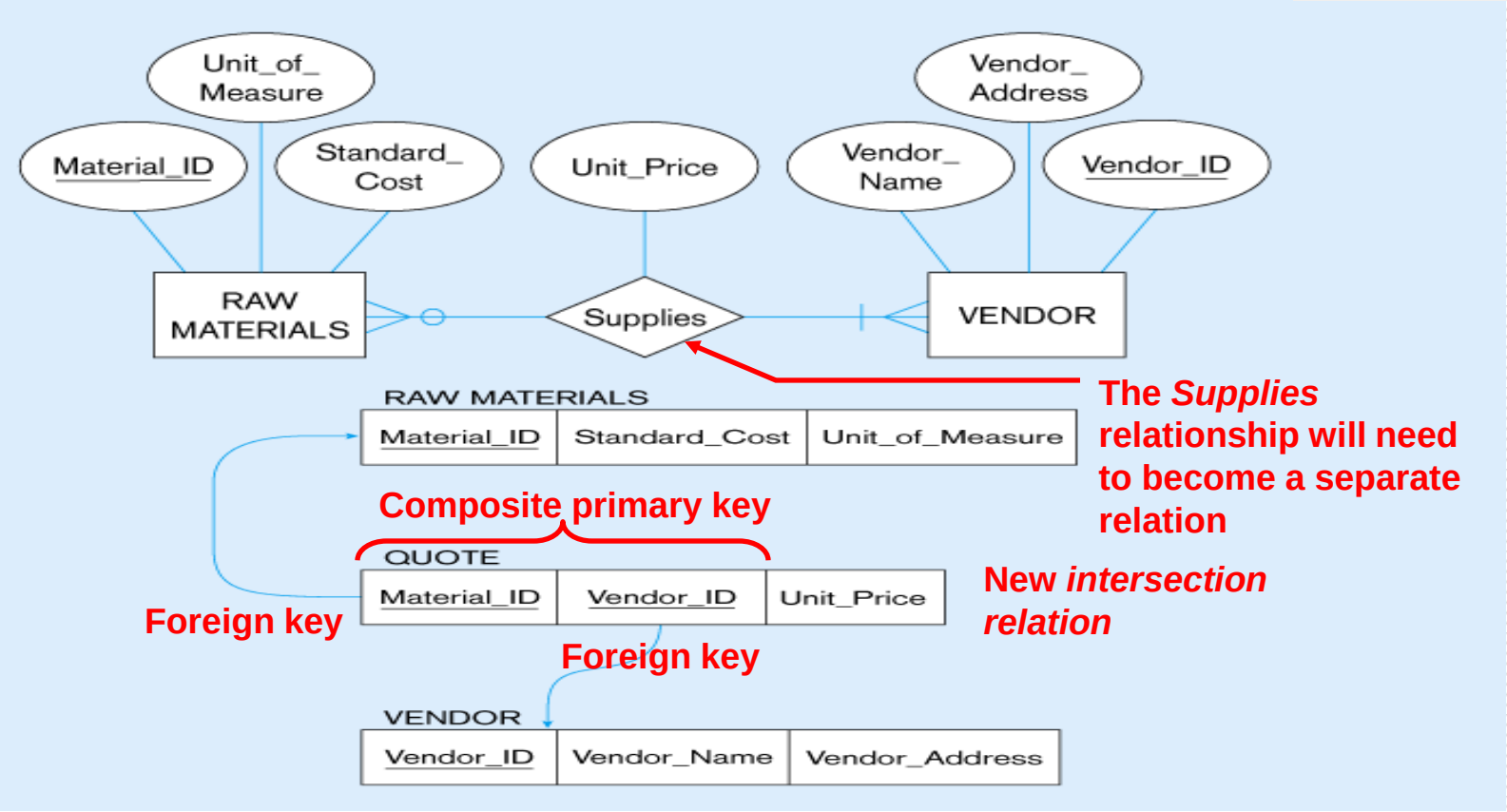
- Qui tắc 5: Biến đổi mối liên kết 1-ngôi hoặc 2-ngôi có lượng số nhiều-nhiều thành một quan hệ.
  - Quan hệ chứa các khóa của các kiểu thực thể tham gia vào mối liên kết.
  - Khóa của quan hệ gồm cả hai khóa của hai kiểu thực thể.
  - Thuộc tính của quan hệ là thuộc tính của mối liên kết.

Chương 3: Mô hình dữ liệu quan hệ



Biến đổi mối liên kết một ngôi có lượng số nhiều - nhiều

Chương 3: Mô hình dữ liệu quan hệ

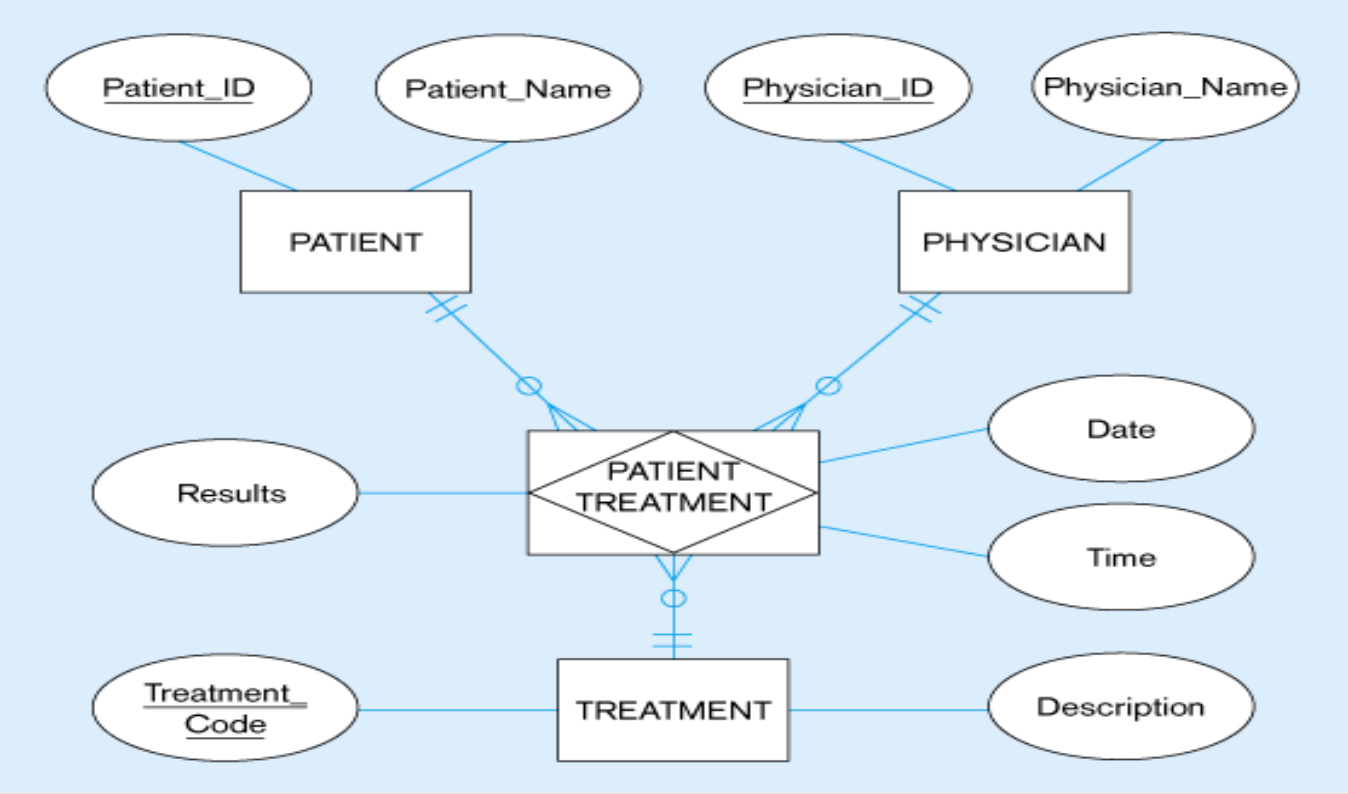


Biến đổi mối liên kết hai ngôi có lượng số nhiều - nhiều

Chương 3: Mô hình dữ liệu quan hệ

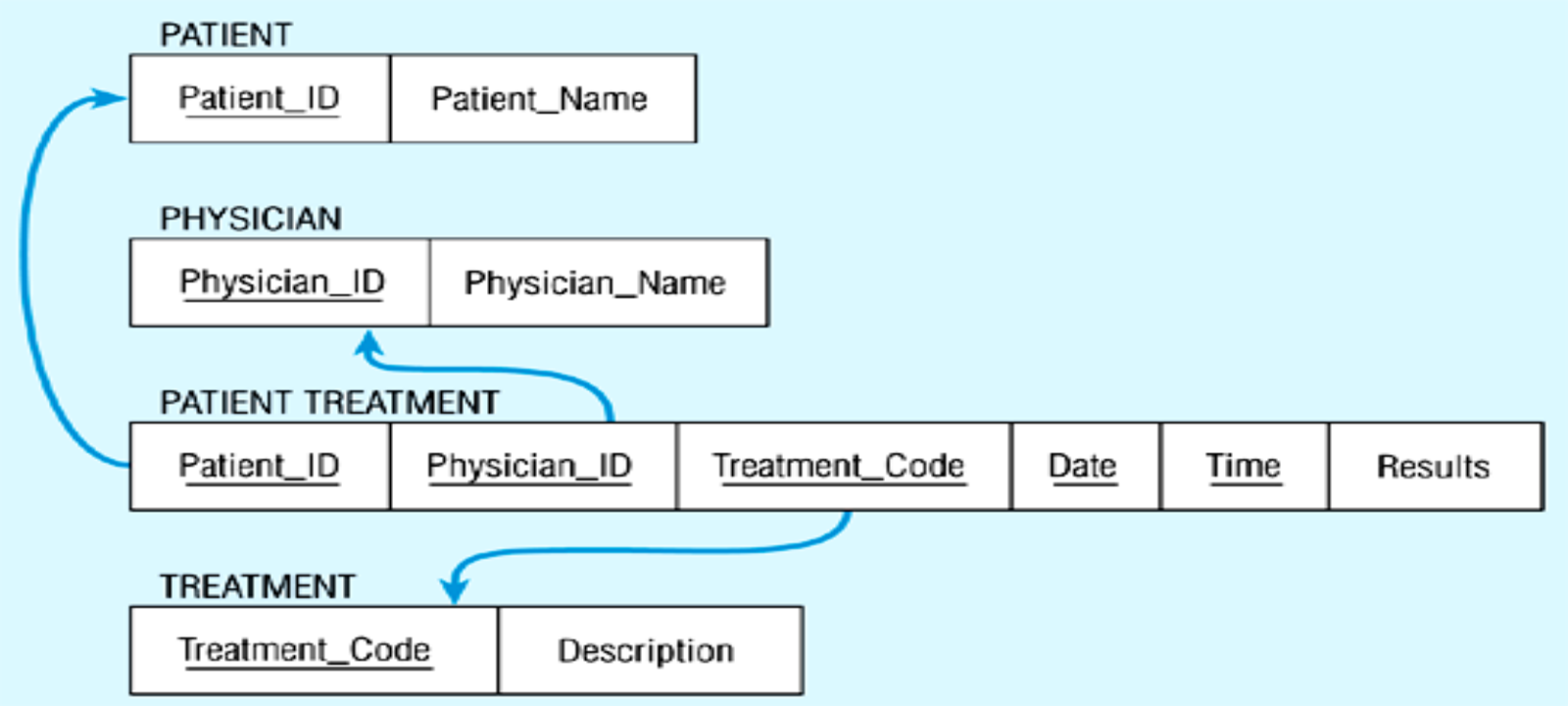
- Qui tắc 6: Biến đổi mối liên kết 3-ngôi thành một quan hệ.
  - Quan hệ chứa ba khóa của ba kiểu thực thể tham gia vào mối liên kết.
  - Mối liên kết có bao nhiêu kiểu thực thể bên phía *một* thì quan hệ có bấy nhiêu khóa: đối với một kiểu thực thể bên phía *một* thì khóa của quan hệ gồm cả hai khóa của hai kiểu thực thể còn lại. Nếu không có kiểu thực thể bên phía *một* thì khóa của quan hệ bao gồm cả ba khóa của ba kiểu thực thể.
  - Thuộc tính của quan hệ là thuộc tính của mối liên kết.

Chương 3: Mô hình dữ liệu quan hệ



Biến đổi mối liên kết ba ngôi

Chương 3: Mô hình dữ liệu quan hệ



Biến đổi mối liên kết ba ngôi

Chương 3: Mô hình dữ liệu quan hệ

Đại số quan hệ:

- Toán hạng là các quan hệ và kết quả thu được là một quan hệ khác.
- Với các phép toán hợp, hiệu, tích, giao
- Các toán hạng phải là các quan hệ cùng ngôi
- Không phụ thuộc vào tên thuộc tính
- Phụ thuộc vào thứ tự thuộc tính.

Chương 3: Mô hình dữ liệu quan hệ

- Phép hợp: R1 ∪ R2

Là tập các bộ thuộc R1 hoặc R2 hoặc thuộc cả hai.

- Phép giao R1 ∩ R2

Là tập các bộ thuộc R1 và R2.

- Phép hiệu R1 - R2

Là tập các bộ thuộc R1 nhưng không thuộc R2.

Chương 3: Mô hình dữ liệu quan hệ

R1

A	B	C
a	b	c
d	a	f
c	b	d

R2

D	E	F
b	g	a
d	a	f

R1 ∪ R2

a	b	c
d	a	f
c	b	d
b	g	a

R1 ∩ R2

d	a	f

R1 - R2

a	b	c
c	b	d

Chương 3: Mô hình dữ liệu quan hệ

- Tích Descartes
- R1 là quan hệ k1 ngôi
- R2 là quan hệ k2 ngôi
- R1 x R2 là tập các bộ, mỗi bộ có k1 + k2 thành phần.

Chương 3: Mô hình dữ liệu quan hệ

R1

A	B	C
a	b	c
d	a	f
c	b	d

R2

D	E	F
b	g	a
d	a	f

R1 x R2

A	B	C	D	E	F
a	b	c	b	g	a
a	b	c	d	a	f
d	a	f	b	g	a
d	a	f	d	a	f
c	b	d	b	g	a
c	b	d	d	a	f

Chương 3: Mô hình dữ liệu quan hệ

Phép chiếu:

- Đầu vào: một quan hệ R
- Đầu ra: quan hệ R chỉ gồm một số thuộc tính được chiếu.
- Quan hệ thu được loại bỏ các bộ giống nhau.
- Ký hiệu: Π x1, x2, .., xn(R)
- x1, x2,.. là các thuộc tính của R được chiếu.
- R là quan hệ mà phép chiếu sẽ thực hiện trên đó.

Chương 3: Mô hình dữ liệu quan hệ

SV

MSV	Hoten	Ngaysinh	Diachi
A01	Lê Na	1/4/1980	7 Giải phóng, HBT, HN
A02	Trần Hà	2/4/1980	8 Tràng Tiền, HK, HN
A03	Hà Ly	3/4/1981	9 Thái hà, ĐĐ, HN

$$\Pi_{\textbf{Hoten, Diachi}}(SV)$$

Hoten	Diachi
Lê Na	7 Giải phóng, HBT, HN
Trần Hà	8 Tràng Tiền, HK, HN
Hà Ly	9 Thái hà, ĐĐ, HN

Chương 3: Mô hình dữ liệu quan hệ

- Đầu vào: (phép chọn) là một quan hệ R
- Đầu ra: tập các bộ thuộc R mà thỏa mãn điều kiện.
- Ký hiệu: $\sigma_C(R)$
- C là công thức điều kiện (biểu thức logic), gồm:
- Toán hạng là hằng hoặc tên thuộc tính
- Toán tử so sánh số học <, =, >, ≤, ≠, ≥, và
- Toán tử logic $\wedge$ (and), $\vee$ (or), ¬ (not)
- R là quan hệ mà phép chọn thực hiện trên đó.

Chương 3: Mô hình dữ liệu quan hệ

SV

MSV	Hoten	Ngaysinh	Diachi
A01	Lê Na	1/4/1980	7 Giải phóng, HBT, HN
A02	Trần Hà	2/4/1980	8 Tràng Tiền, HK, HN
A03	Hà Ly	3/4/1981	9 Thái hà, ĐĐ, HN

$$\sigma_{\text{Ngaysinh} \geq 1/1/1981}(\text{SV})$$

MSV	Hoten	Ngaysinh	Diachi
A03	Hà Ly	3/4/1981	9 Thái hà, ĐĐ, HN

Chương 3: Mô hình dữ liệu quan hệ

- Phép nối: $R1 \underset{i\,\theta\,j}{\bowtie} R2 = \sigma_c(R1 \times R2)$

θ là một toán tử so sánh số học <, =, >, ≤, ǂ, ≥

- Nối θ của R1 và R2 là những bộ thuộc tích Descartes của R1xR2 sao cho thành phần thứ i của R1 có quan hệ θ với thành phần thứ j của R2.

Chương 3: Mô hình dữ liệu quan hệ

R1

A	B	C
1	2	3
4	5	6
7	8	9

R2

D	E
3	1
6	2

R1 $\bowtie_{B < D}$ R2

A	B	C	D	B
1	2	3	3	1
1	2	3	6	2
4	5	6	6	2

Chương 4: Chuẩn hóa dữ liệu

- Chuyển Slide Phụ thuộc hàm

Giảng viên: Ths. Nguyễn Thị Kim Phụng - Đại học CNTT

HỆ QUẢN TRỊ CSDL ORACLE

CHƯƠNG 1

KHOA HỆ THỐNG THÔNG TIN

Chương 1: Giới thiệu các công cụ SQL*Plus, iSQLPlus, OEM và Ngôn ngữ truy vấn SQL

1. Giới thiệu Oracle, các phiên bản.
2. Công cụ SQL*Plus
3. Công cụ iSQLPlus
4. Công cụ OEM
5. Ngôn ngữ SQL (Các lệnh định nghĩa dữ liệu, thao tác dữ liệu, truy vấn dữ liệu, điều khiển dữ liệu, phân quyền users, roles).
6. Các lệnh giao tác
7. Sequences, Views, Indexes, Synnonym
8. Oracle data dictionary

1. Giới thiệu Oracle (1)

- Oracle là tên của một hãng phần mềm, một hệ quản trị cơ sở dữ liệu phổ biến trên thế giới . Hãng Oracle ra đời đầu những năm 70 của thế kỷ 20 tại Mỹ.
- Tập hợp các sản phẩm phần mềm phục vụ cho mục đích xây dựng và quản lý hệ thống thông tin, các ứng dụng giao tiếp cơ sở dữ liệu bên dưới.
- Là hệ quản trị cơ sở dữ liệu quan hệ (RDBMS) mang tính mềm dẻo, linh động, thích ứng cao với các quy mô xử lý giao dịch, an toàn hệ thống. Cung cấp các công cụ xây dựng và quản lý cơ sở dữ liệu.
- Tích hợp Web: kết nối ứng dụng với công nghệ Web được tích hợp trong Oracle WebServer.

1. Giới thiệu Oracle (1) – Các phiên bản

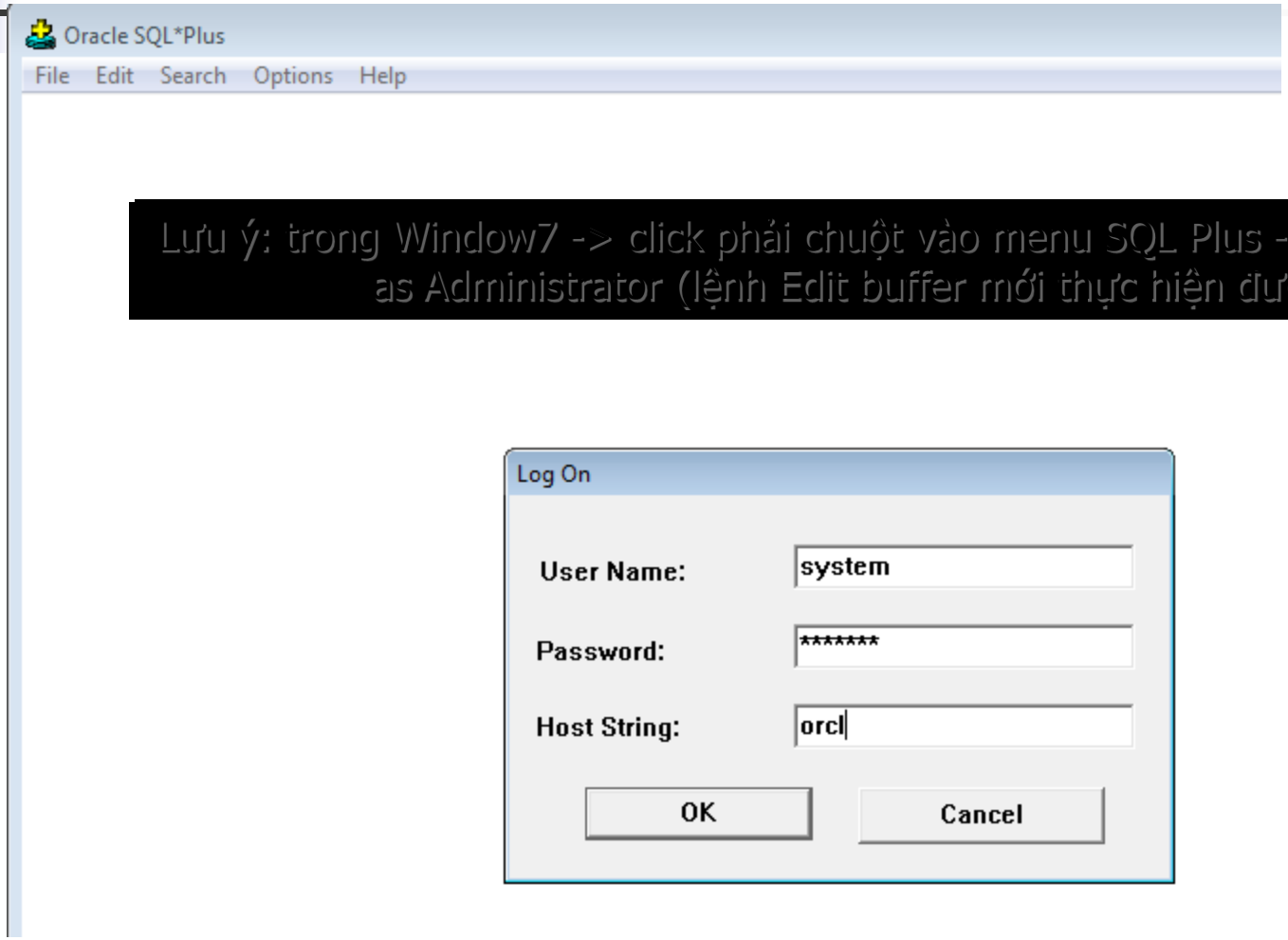
- Phiên bản 1 phát hành năm 1977.
- Phiên bản 2 phát hành năm 1979.
- Phiên bản 3 phát hành năm 1983.
- Phiên bản 4 phát hành năm 1984.
- Phiên bản 5 phát hành năm 1985 (SQLNet: hệ thống khách/chủ (client/server)).
- Phiên bản 6 phát hành năm 1988 (Sequence, thao tác ghi trễ).
- Oracle7 được phát hành năm 1992 (SQL*DBA).
- Năm 1999 Oracle giới thiệu Oracle8i (i:internet).
- Năm 2001-2002: 2 phiên bản Oracle9i (Release 1&2).
- Năm 2004-2005: 2 phiên bản Oracle10g (g:Grid) (Release 1&2).
- Năm 2007-2009: phiên bản Oracle11g (Release 1&2).

- http://www.oracle.com/technology/software/products/database/index.html

1. Giới thiệu Oracle (2) – Các sản phẩm

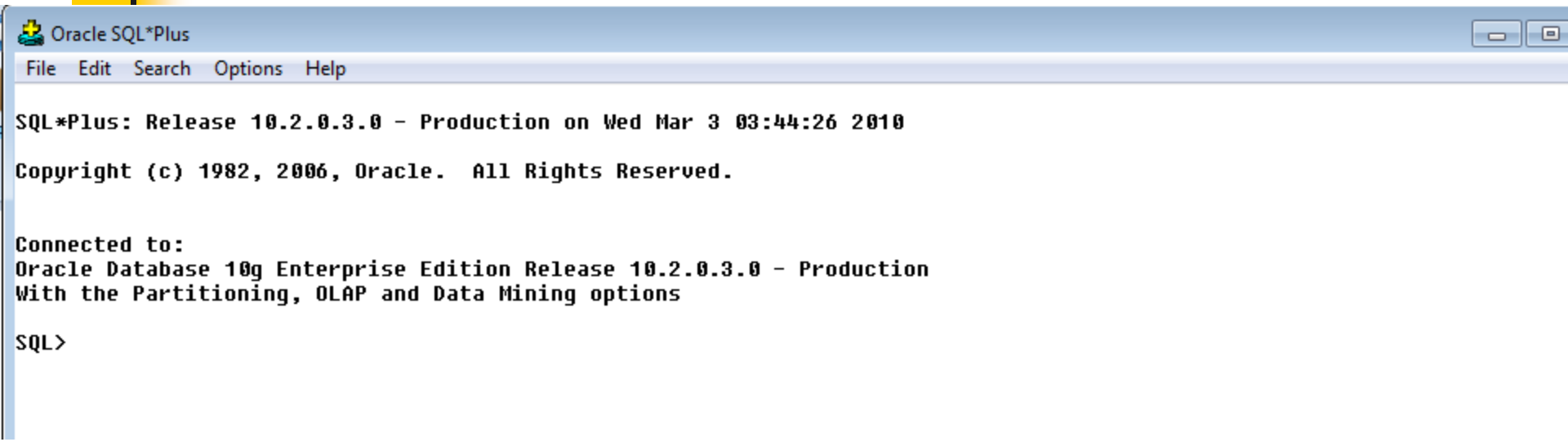
- Database Server (Server quản lý cơ sở dữ liệu)
- Công cụ thao tác cơ sở dữ liệu: SQL*Plus
- Công cụ phát triển ứng dụng: Oracle Developer Suite (Form, Report,..),Oracle JDeveloper,...
- Phân tích dữ liệu: Oracle Discoverer, Oracle Express, Oracle Warehouse Builder,...
- Oracle Application Server (OAS)
- Ứng dụng đóng gói: Oracle Human Resource, Oracle Financial Applications,…
- Oracle Email, Oracle Calendar, Oracle Web Conferencing,...

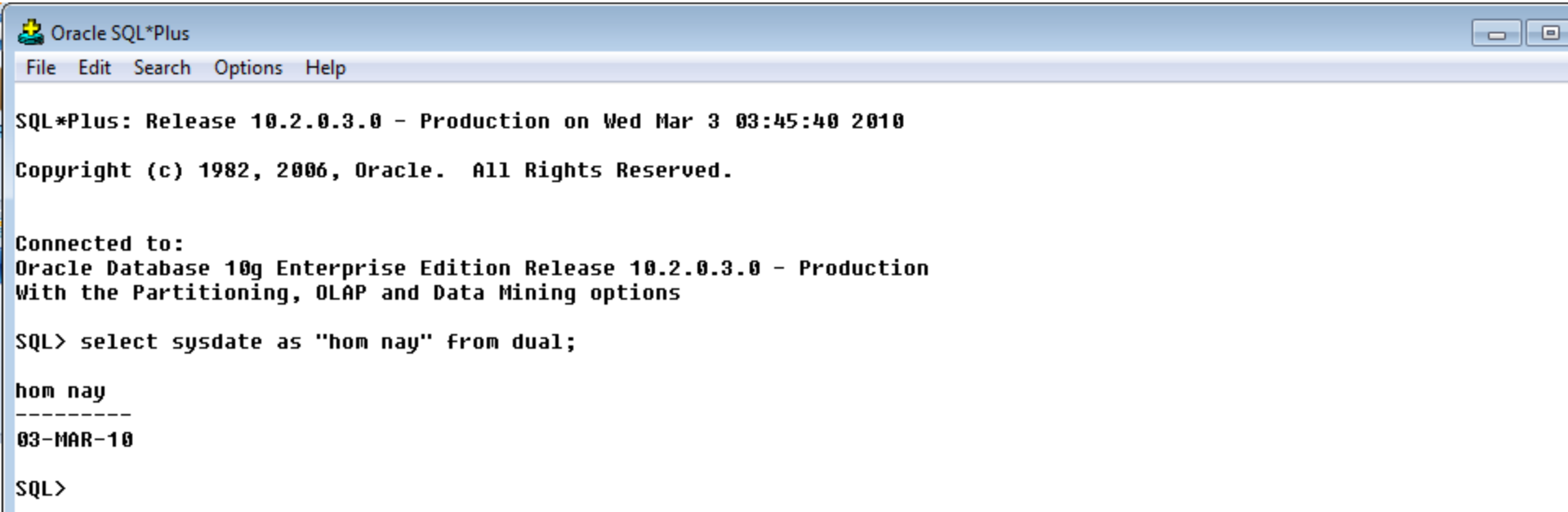
2. Công cụ SQL*Plus (1) – Giao diện



Lưu ý: trong Window7 -> click phải chuột vào menu SQL Plus -> run as Administrator (lệnh Edit buffer mới thực hiện được)

2. Công cụ SQL*Plus (2) – Giao diện





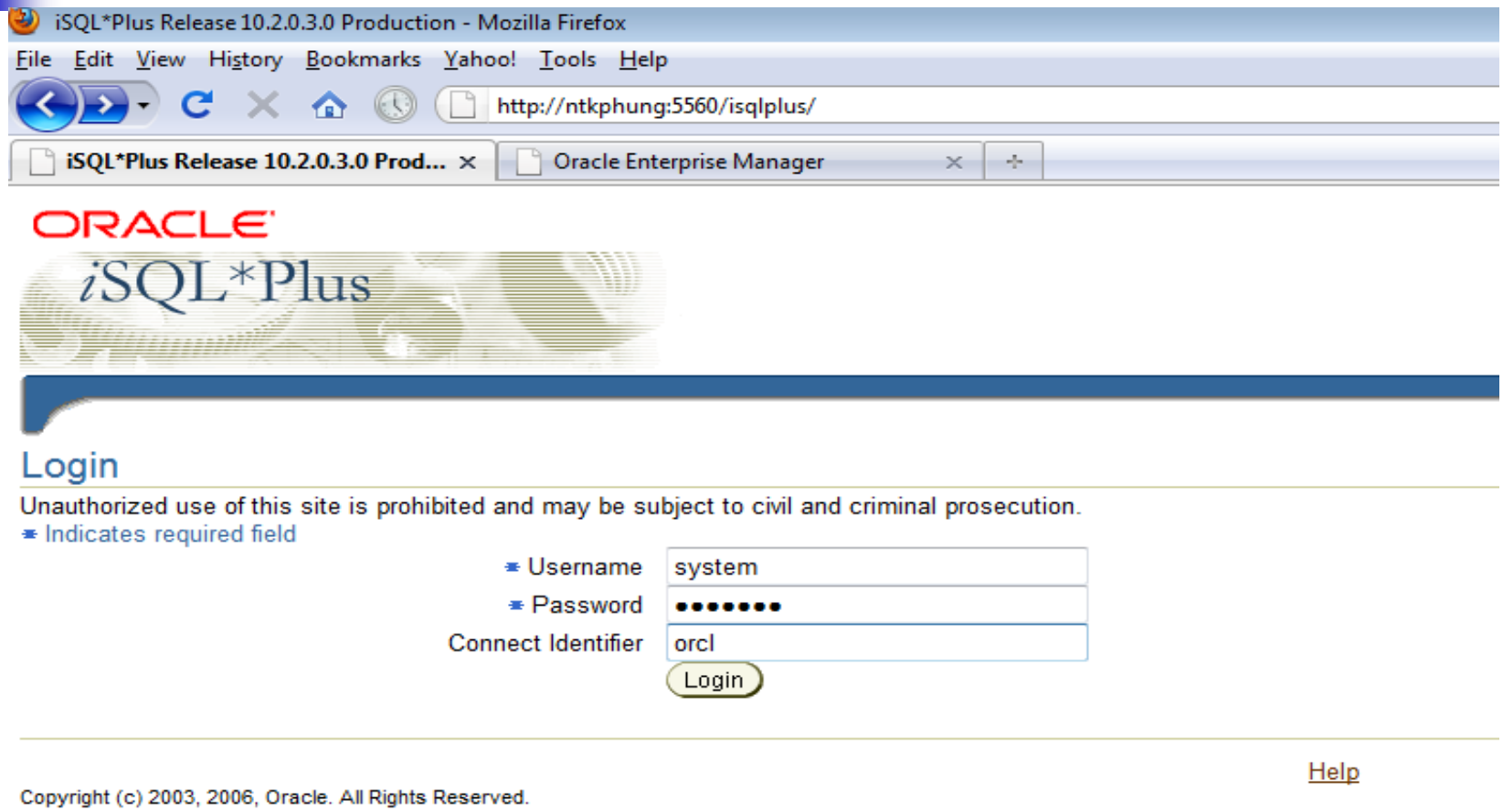
2. Công cụ SQL*Plus (3)

- Ngôn ngữ SQL: ngôn ngữ chuẩn để truy vấn và thao tác trên CSDL quan hệ, dùng trong Oralce khi cần truy xuất CSDL.
- Ngôn ngữ PL/SQL: ngôn ngữ thủ tục của Oracle dùng để xây dựng các ứng dụng, kết hợp SQL để truy xuất dữ liệu.
- Công cụ SQL*Plus: sản phẩm của Oracle, là môi trường để thực hiện các lệnh SQL và PL/SQL.
- SQL*Plus còn có các lệnh riêng để điều khiển cách xử lý của SQL*Plus, định dạng dữ liệu xuất.

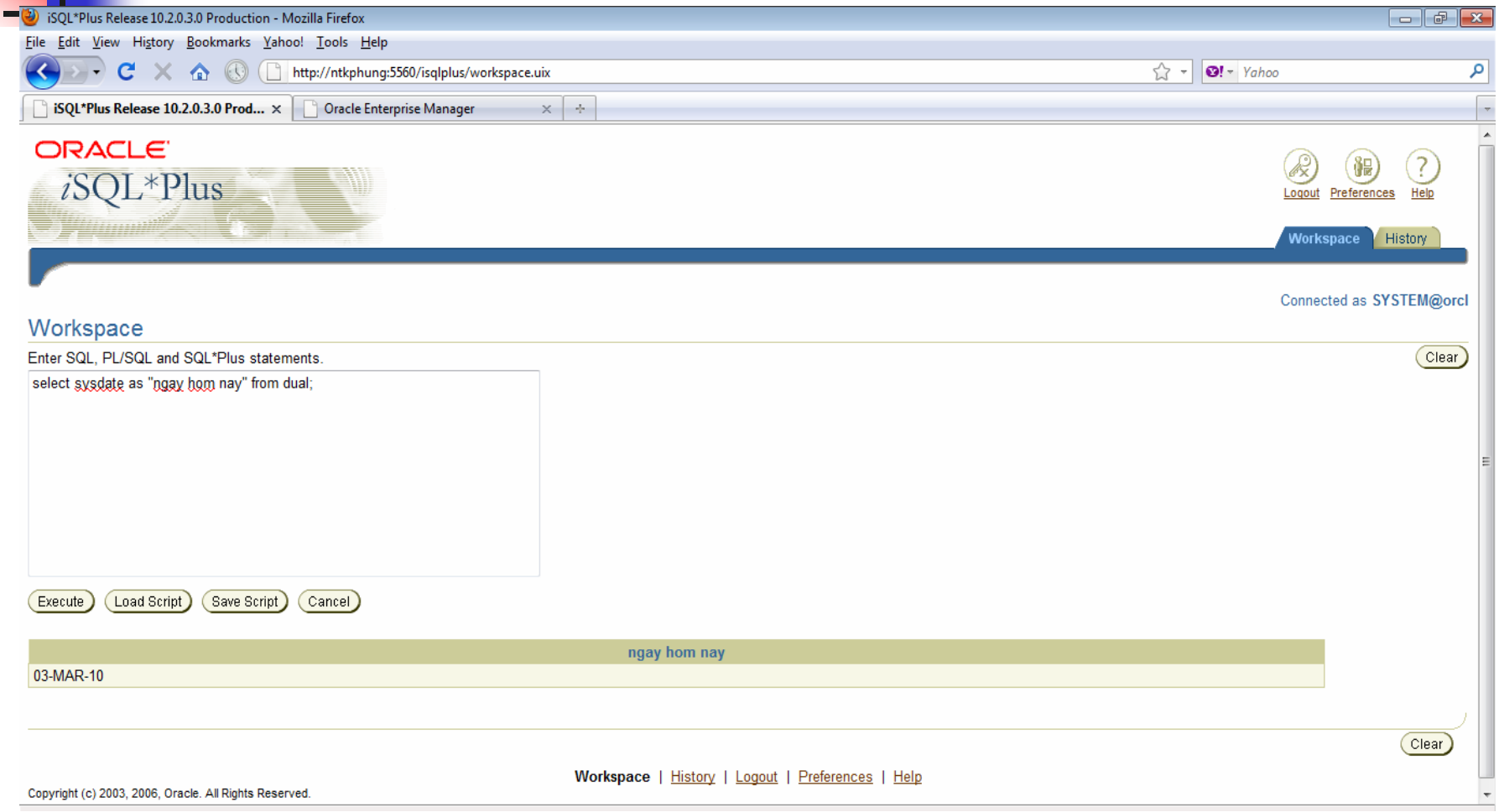
2. Công cụ SQL*Plus (4) – Các lệnh thường dùng

Lệnh	Mục đích
CLEAR BUFFER	Xóa tất cả dòng lệnh từ SQL Buffer (file "afdeit.buf")
DEL	Xóa dòng lệnh hiện hành trong Buffer
LIST	Liệt kê tất cả các hàng trong SQL Buffer
LIST n	Liệt kê hàng thứ n trong SQL Buffer
LIST m n	Liệt kê các hàng trong phạm vi từ m đến n
RUN (chạy trong Buffer (file "afdeit.buf"))	Hiển thị câu SQL và thực thi lệnh SQL hiện hành
/	Chỉ thực thi lệnh SQL hiện hành trong Buffer, ko in ra
SAVE filename	Lưu nội dung hiện hành của SQL Buffer vào filename
GET filename	Xuất nội dung của filename
START filename	**Thực thi lệnh trong filename**
EDIT filename	Soạn thảo nội dung filename
EXIT	Thoát khỏi môi trường SQL*Plus
@filename	**Thực thi lệnh trong filename**
CONNECT user/password@service_name	Dùng Username và password connect đến DB Server

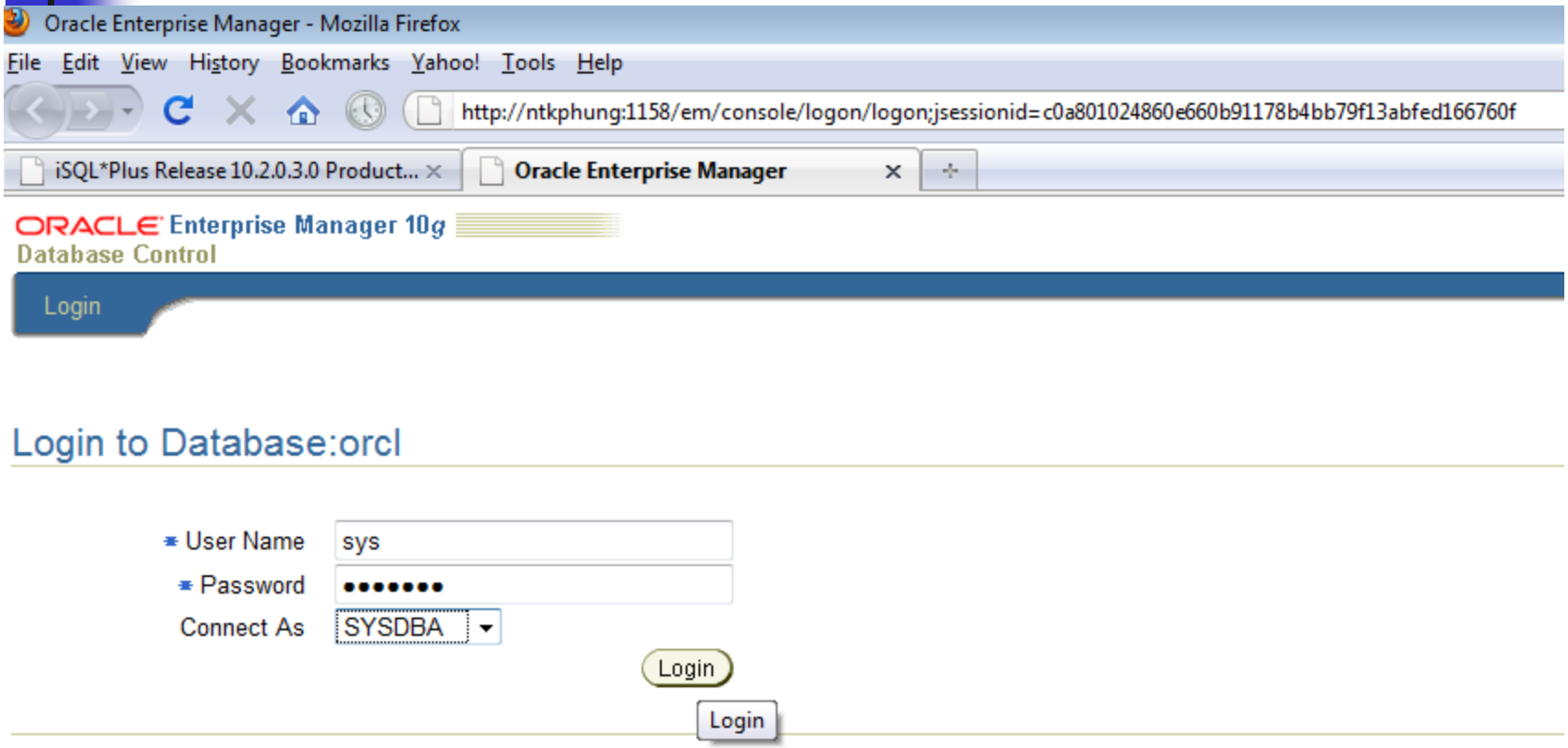
3. Công cụ iSQLPlus (1) – Thực thi lệnh SQL,PL/SQL trên web



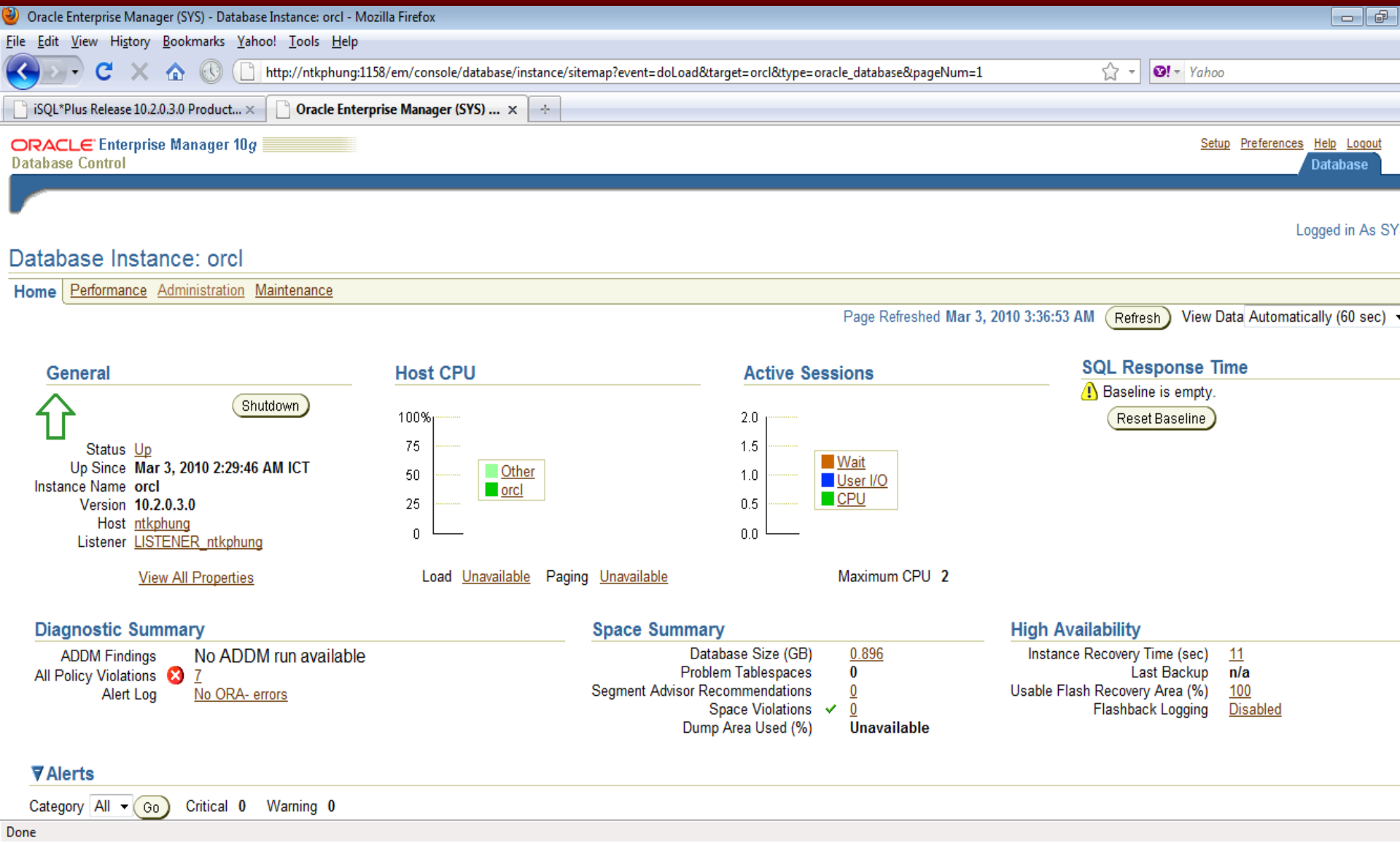
3. Công cụ iSQLPlus (2) – Thực thi lệnh SQL,PL/SQL trên web



4. Công cụ Oracle Enterprise Manager - (OEM) quản trị Oracle qua web (1)



4. Công cụ Oracle Enterprise Manager - (OEM) quản trị Oracle qua web (2)



SỬ DỤNG BIẾN THAY THẾ &, &&

* Các loại biến trong SQL*Plus: 2 kiểu biến

Biến thay thế &: dấu & đặt trước biến. Biến được nhập giá trị lúc thực thi câu SQL. Kết quả câu SQL tùy thuộc vào giá trị nhập cho biến.

- Ví dụ:
```
SELECT MaNV, HoNV, TenNV
    FROM NhanVien
    WHERE MaPhong=&DEPT_NUMBER;
```

Khi chạy lệnh SQL trong môi trường SQL*Plus sẽ hiện ra dòng chữ -> nhập giá trị vào (VD:5).

```
SQL>Enter value for DEPT_NUMBER:5
```

- Lưu ý: biến kiểu chuỗi, kiểu ngày đặt trong cặp dấu ' '

SỬ DỤNG BIẾN THAY THẾ &, &&

- Biến thay thế &&: dấu && đặt trước biến. Giá trị nhập vào được lưu trữ cho những lần sau.

- Ví dụ:
```
SELECT MaNV, HoNV, TenNV
FROM NhanVien
WHERE MaPhong=&&DEPTNO;
```

Khi chạy lệnh SQL trong môi trường SQL*Plus sẽ hiện ra dòng chữ -> nhập giá trị vào (VD:5).

```
SQL>Enter value for DEPTNO:5
```

Ghi chú: Lần sau chạy câu lệnh, giá trị này được nhớ.

5. Ngôn ngữ SQL (1) – Giới thiệu

- Là ngôn ngữ chuẩn để truy vấn và thao tác trên CSDL quan hệ
- Là ngôn ngữ phi thủ tục
- Khởi nguồn của SQL là SEQUEL - *Structured English Query Language,* năm 1974*)*
- *Các chuẩn SQL*
  - SQL89 (SQL1)
  - SQL92 (SQL2)
  - SQL99 (SQL3)

5. Ngôn ngữ SQL (2) – Phân loại

- Ngôn ngữ định nghĩa dữ liệu (DDL): tạo table (bảng), view (khung nhìn), sửa cấu trúc table và thêm, xóa RBTV, xóa table, xóa view, đổi tên table
- Ngôn ngữ thao tác dữ liệu a, sửa dữ truy vấn dữ liệu.
- Ngôn ngữ điều khiển dữ liệu: tạo quyền hạn, xóa quyền, làm cho hiệu lực/mất hiệu lực quyền, tạo người dùng, đổi mật khẩu, xóa người dùng, cấp quyề thu hồi quyền sử dụng trên cơ sở dữ liệu.
- Ngoài ra còn có các lệnh điều khiển giao tác.
- Lệnh thao tác trên các thành phần CSDL khác: Synonym, Index và Sequence

5. Ngôn ngữ SQL (3) – Ngôn ngữ định nghĩa dữ liệu

- Ngôn ngữ định nghĩa dữ liệu (DDL– Data Definition Language)
- m:
  - nh tạo table, tạo view (CREATE...)
  - nh sửa cấu trúc table, thêm, xóa ràng buộc toàn vẹn trên table (ALTER...)
  - nh xóa table, xóa view (DROP...)
  - Đổi tên table (RENAME…)

5. Ngôn ngữ SQL (4) – Ngôn ngữ thao tác dữ liệu

- Ngôn ngữ thao tác dữ liệu (DML – Data Manipulation Language)
- Bao gồm:
  - Lệnh thêm dữ u (INSERT...)
  - Lệnh sửa dữ u (UPDATE...)
  - Lệnh xóa dữ u (DELETE...)
  - Truy vấn dữ u (SELECT...)

5. Ngôn ngữ SQL (5) – Ngôn ngữ điều khiển dữ liệu

- Ngôn ngữ điều khiển dữ liệu (DCL – Data Control Language). Bao gồm:
  - Lệnh tạo quyền hạn (Create Role…)
  - Lệnh thiết lập, đổi hay bỏ mật khẩu của role (Alter Role…)
  - Lệnh xóa quyền hạn (Drop Role…)
  - Lệnh tạo người dùng, đổi mật khẩu và xóa người dùng (Create User…, Alter User…, Drop User…)
  - Lệnh cấp quyền cho người sử ng cơ sở dữ u (GRANT...)
  - Lệnh thu hồi quyền hạn của người sử dụng cơ sở dữ liệu (REVOKE...)

5. Lệnh điều khiển giao tác + các đối tượng khác

- Lệnh điều khiển giao tác bao gồm:
  - Lệnh COMMIT, lệnh ROLLBACK, lệnh SAVEPOINT, lệnh AUTOCOMMIT
- Các đối tượng khác:
  - SYNONYM: tạo một Synonym (Create Synonym…), xóa Synnonym (Drop Synonym…)
  - INDEX: tạo chỉ mục cho table (Create Index…) , bảo đảm giá trị duy nhất trong cột, thường là giá trị Primary key.
  - SEQUENCE: tạo giá trị SEQUENCE cho cột (Create Sequence…).

Ngôn ngữ định nghĩa dữ liệu
Tạo table, view (1)

- Cú pháp

CREATE TABLE <tên_table>
(
tên_cột1 kiểu_dữ_liệu [not null],
tên_cột2 kiểu_dữ_liệu [not null],
…
tên_cộtn kiểu_dữ_liệu [not null],
khai báo khóa chính, khóa ngoại, ràng buộc toàn vẹn
)

Ngôn ngữ định nghĩa dữ liệu
Tạo table, view (2)

- Cú pháp

  CREATE [OR REPLACE] [FORCE] VIEW <tên_view> [tên-cột1, cột2,…]

  AS

  SELECT …

  [Điều kiện] [Ràng buộc]

- Ghi chú: tùy chọn Replace sẽ xóa view và tạo view mới nếu view đã tồn tại rồi.

Ngôn ngữ định nghĩa dữ liệu

Tạo table, view - Kiểu dữ liệu (3)

Loại dữ liệu	Mô tả
VARCHAR2(n)	Dữ liệu kiểu ký tự, n<=4000
CHAR(n)	Dữ liệu kiểu ký tự, kích thước cố đinh, n<=2000
NUMBER	Kiểu số nguyên, số ký số tối đa là 38 ký số
NUMBER(p)	Kiểu số nguyên, với số ký số tối đa là p
NUMBER(p,s)	Kiểu số thực, tối đa p ký số, s số thập phân. p≤38, -84 ≤ s ≤ 127.Ví dụ: số 7456123, khai báo kiểu number (7, -2) = 7456100
DATE	Kiểu ngày, lưu ngày từ 1/1/4712 BC -> 31/12/9999
LONG	Kiểu ký tự
RAW	Chuỗi nhị phân dài tối đa 2000 bytes
LONG RAW	Chuỗi nhị phân dài tối đa 2GB
BLOB	(Bynary Large Object) có độ dài ≤ 4GB
CLOB	(Character Large Object) có độ dài ≤ 4GB
BFILE	Chứa con trỏ chỉ đến một tập tin nhị phân ở ngoài DB

Ngôn ngữ định nghĩa dữ liệu
Tạo table, view (4)

Cho lược đồ CSDL "quản lý đề án cty" như sau

NHANVIEN (MaNV, HoNV, TenLot, TenNV, Phai, Luong, Phong, NgaySinh, DiaChi, Ma_NQL)

PHONGBAN (MaPHG, TenPHG, TrPHG, NG_NhanChuc)

DEAN (MaDA, TenDA, DDIEM_DA, Phong)

PHANCONG (MaNV, MaDA, ThoiGian)

DIADIEM_PHG (MaPHG, DIADIEM)

THANNHAN (MaNV, TenTN, Phai, NGSinh,QuanHe)

NHANVIEN

MANV	HOTEN	NTNS	PHAI	MA_NQL	MaPH	LUONG
001	Vuong Ngoc Quyen	22/10/1957	Nu		QL	3.000.000
002	Nguyen Thanh Tung	09/01/1955	Nam	001	NC	2.500.000
003	Le Thi Nhan	18/12/1960	Nu	001	DH	2.500.000
004	Dinh Ba Tien	09/01/1968	Nam	002	NC	2.200.000
005	Bui Thuy Vu	19/07/1972	Nam	003	DH	2.200.000
006	Nguyen Manh Hung	15/09/1973	Nam	002	NC	2.000.000
007	Tran Thanh Tam	31/07/1975	Nu	002	NC	2.200.000
008	Tran Hong Minh	04/07/1976	Nu	004	NC	1.800.000

DEAN

MADA	TENDA	PHONG	NamThucHien
TH001	Tin hoc hoa 1	NC	2002
TH002	Tin hoc hoa 2	NC	2003
DT001	Dao tao 1	DH	2004
DT002	Dao tao 2	DH	2004

PHONGBAN

MAPH	TENPH	TRPH
QL	Quan Ly	001
DH	Dieu Hanh	003
NC	Nghien Cuu	002

PHANCONG

MANV	MADA	THOIGIAN
001	TH001	30,0
001	TH002	12,5
002	TH001	10,0
002	TH002	10,0
002	DT001	10,0
002	DT002	10,0
003	TH001	37,5
004	DT001	22,5
004	DT002	10,0
006	DT001	30,5
007	TH001	20,0
007	TH002	10,0
008	DT002	12,5

Ngôn ngữ định nghĩa dữ liệu
Tạo table, view (5)

- Ví dụ: câu lệnh để tạ t table nhân viên

```
CREATE TABLE NHANVIEN
(  MANV varchar2(10) NOT NULL,
   HONV varchar2(50) NOT NULL, TENLOT varchar2(50) NOT NULL,
   TENNV varchar2(50) NOT NULL, NGAYSINH date,
   PHAI varchar2(3) NOTNULL,  DIACHI varchar2(100),
   MA_NQL varchar2(10),
   PHONG varchar2(10),
   LUONG number,
   CONSTRAINT PK_NV   PRIMARY KEY (MANV),
   CONSTRAINT FK_NV_PB   FOREIGN KEY (PHONG) REFERENCES
   PHONGBAN (MAPHG)      )
```

Ngôn ngữ định nghĩa dữ liệu
Tạo table, view (6)

- Ví dụ: câu lệnh để tạ t view chứa họ tên nhân viên phòng số 5 và tên phòng ban họ trực thuộc

```
CREATE VIEW TrucThuoc
AS SELECT MANV, HONV, TENNV, TENPHG  FROM
NhanVien nv, PhongBan p WHERE nv.PHONG =
p.MAPHG AND p.nv.PHONG = 5
```

Ngôn ngữ định nghĩa dữ liệu
Sửa cấu trúc table (7)

- Thêm thuộc tính

  ALTER TABLE tên_table ADD tên_cột kiểu_dữ_liệu

  - Ví dụ: o table nhân viên

    ALTER TABLE NHANVIEN ADD GHI_CHU varchar2(20)

- Sửa kiểu dữ liệu thuộc tính

  ALTER TABLE tên_table MODIFY tên_cột kiểu_dữ_liệu_mới

Ngôn ngữ định nghĩa dữ liệu
Sửa cấu trúc table (8)

- Ví dụ: t Ghi chú
  ALTER TABLE NHANVIEN **MODIFY** GHI_CHU varchar2(30)

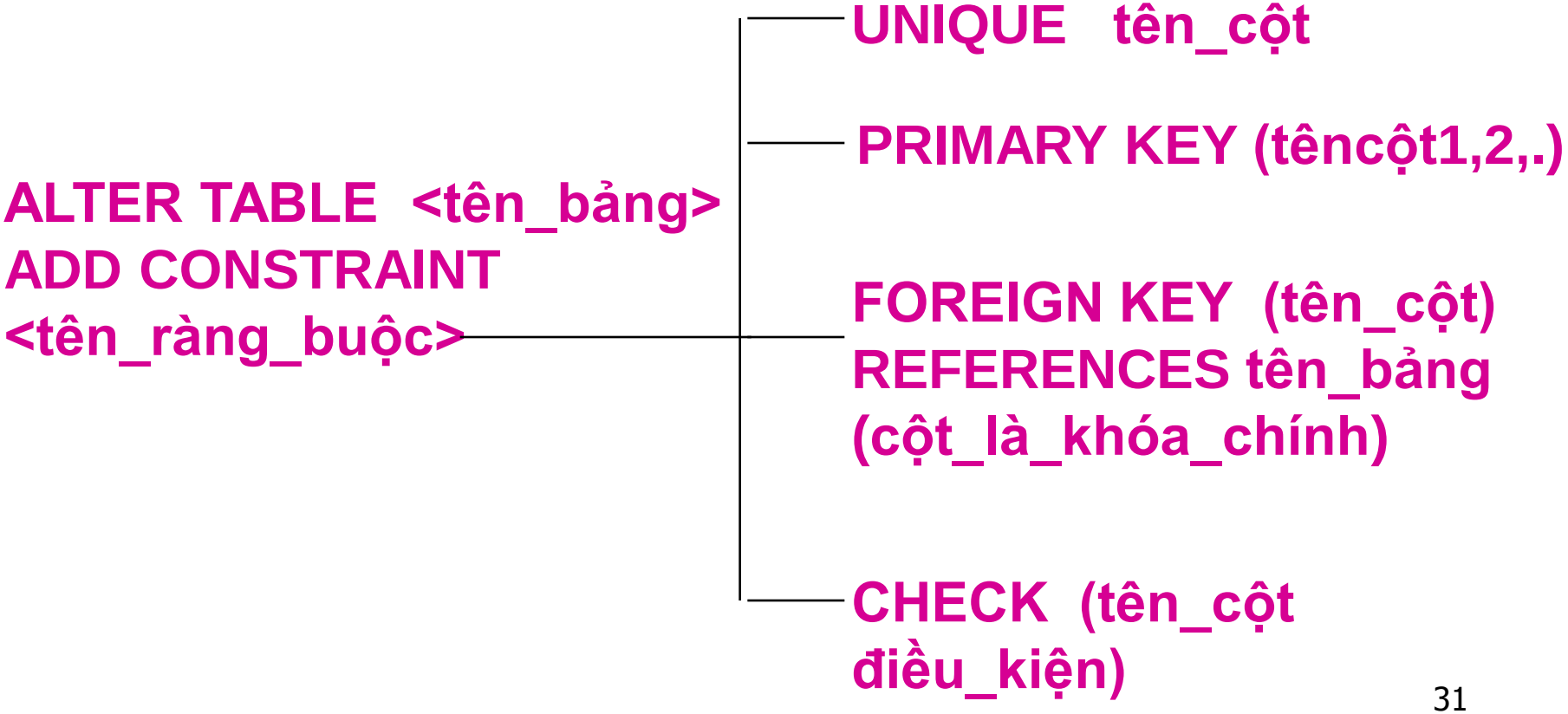
- Xóa thuộc tính

  ALTER TABLE tên_bảng DROP COLUMN tên_cột

  - Ví dụ: a cột Ghi_chú từ bảng nhân viên
    ALTER TABLE NHANVIEN DROP COLUMN GHI_CHU

Ngôn ngữ định nghĩa dữ liệu
Sửa cấu trúc table (9)

- Thêm ràng buộc toàn vẹn



Ngôn ngữ định nghĩa dữ liệu
Sửa cấu trúc bảng (10)

- Ví dụ
  - ALTER TABLE NHANVIEN ADD CONSTRAINT PK_NV PRIMARY KEY (MANV)
  - ALTER TABLE NHANVIEN ADD CONSTRAINT FK_NV_PB FOREIGN KEY (PHONG) REFERENCES PHONGBAN(MAPHG)
  - ALTER TABLE NHANVIEN ADD CONSTRAINT CHK CHECK (PHAI IN ('Nam') OR ('Nu'))
  - ALTER TABLE NHANKHAU ADD CONSTRAINT UQ_NK UNIQUE (CMND)

Ngôn ngữ định nghĩa dữ liệu
Sửa cấu trúc bảng (11)

- Xóa ràng buộc toàn vẹn

  ALTER TABLE tên_bảng DROP CONSTRAINT tên_ràng_buộc

  - Ví dụ

    ALTER TABLE NHANVIEN DROP CONSTRAINT FK_NV_PB

Ngôn ngữ định nghĩa dữ liệu
Xóa table, xóa view (12)

- Cú pháp xóa table

  DROP TABLE tên_table
- Cú pháp xóa view

  DROP VIEW tên_view

  - Ví dụ: xóa bảng (table) nhân viên
    DROP TABLE NHANVIEN
  - Ví dụ: xóa khung nhìn (view) TrucThuoc
    DROP VIEW TrucThuoc

Ngôn ngữ thao tác dữ liệu
Thêm dữ liệu vào bảng (1)

- Cú pháp

  INSERT INTO tên_bảng VALUES (giá_trị_1, giá_trị_2,…, giá_trị_n)

  INSERT INTO tên_bảng (cột1, cột2) VALUES (giá_trị_1, giá_trị_2)

  - Ví dụ

    INSERT INTO NHANVIEN VALUES ('001', 'Vuong', 'Ngoc', 'Quyen', **To_Date('**31/01/1977'**,'dd/mm/yyyy')**, '450 Trung Vuong, Ha Hoi', 'QL')

Ngôn ngữ thao tác dữ liệu
Sửa dữ liệu của bảng (2)

- Cú pháp

  UPDATE tên_bảng SET cột_1 = giá_trị_1,
  cột_2 = giá_trị_2 [WHERE điều_kiện]

  - Ví dụ: Sửa họ nhân viên có mã số '001' thành 'Nguyen'

    UPDATE NHANVIEN SET HONV = 'Nguyen'
    WHERE MANV='001'

Ngôn ngữ thao tác dữ liệu
Sửa dữ liệu của bảng (3)

- Cú pháp
  - Ví dụ: Sửa họ tên của nhân viên có mã số '001' thành 'Nguyen Thanh Tung' và ngày sinh mới là 1/1/1978

```
UPDATE NHANVIEN SET HONV = 'Nguyen',
TENLOT = 'Thanh', TENNV = Tung',
NGAYSINH=To_date('1/1/1978', 'dd-mm-yyyy') WHERE MANV='001'
```

Ngôn ngữ thao tác dữ liệu
Xóa dữ liệu trong bảng (4)

- Cú pháp

  DELETE FROM tên_bảng [WHERE điều_kiện]

  - Ví dụ: xóa nhân viên có mã số '001'

    DELETE FROM NHANVIEN

    WHERE MANV='001'
  - Ví dụ: xóa toàn bộ nhân viên

    DELETE FROM NHANVIEN

Ngôn ngữ thao tác dữ liệu
Truy vấn dữ liệu – lệnh SELECT (5)

- Câu truy vấn tổng quát

```
SELECT [DISTINCT] tên_cột | hàm
FROM bảng
[WHERE điều_kiện]
[GROUP BY cột]
[HAVING điều_kiện]
[ORDER BY cột ASC | DESC]
```

Ngôn ngữ thao tác dữ liệu
Toán tử truy vấn (6)

- Toán tử so sánh
  - =
  - >
  - <
  - >=
  - <=
  - <>
- Toán tử logic: AND, OR, NOT
- Phép toán: +, - ,* , /

Ngôn ngữ thao tác dữ liệu
Toán tử truy vấn (7)

- Các toán tử so sánh khác
  - BETWEEN - định nghĩa một đoạn giá trị liên tục
  - IS NULL - kiểm tra giá trị thuộc tính có null hay không
  - LIKE – kiểm tra chuỗi ký tự tương tự
  - IN – kiểm tra giá trị thuộc tính có thuộc tập hợp các giá trị đã định nghĩa hay không
  - EXISTS – mang giá trị TRUE nếu mệnh đề so sánh trả về ít nhất một bộ (record), FALSE nếu ngược lại

Ngôn ngữ thao tác dữ liệu
Toán tử truy vấn (8)

- Toán tử BETWEEN
  - Ví dụ: Tìm nhân viên sinh vào khoảng 1965 và 1977. Tìm nhân viên có lương không nằm trong khoảng 100000 đến 300000
  - SELECT * FROM NHANVIEN WHERE To_number(To_char(NGAYSINH,'yyyy')) BETWEEN 1965 AND 1977
  - SELECT n.MANV, n.TENNV, p.TENPHG
    FROM NHANVIEN n, PHONGBAN p
    WHERE n.PHONG=p.MAPHG
    AND n.LUONG NOT BETWEEN 100000 AND 300000

Ngôn ngữ thao tác dữ liệu
Toán tử truy vấn (9)

- **Toán tử IS NULL**
  - **Ví dụ: tìm những nhân viên có người quản lý, tìm những học viên chưa đóng tiền học phí.**

a. SELECT * FROM NHANVIEN

WHERE MA_NQL IS NOT NULL

b. SELECT h.MAHV,h.HOTEN,h.DIACHI

FROM HOCVIEN h, BIENLAI b

WHERE h.MAHV=b.MAHV AND b.TIENNOP IS NULL

Ngôn ngữ thao tác dữ liệu
Toán tử truy vấn (10)

- Toán tử LIKE
  - So sánh chuỗi tương đối
  - Cú pháp: s LIKE p, p có thể chứa % hoặc _
  - % : thay thế một chuỗi ký tự bất kỳ
  - _ : thay thế một ký tự bất kỳ
  - Ví dụ SELECT * FROM NHANVIEN

    WHERE HONV LIKE 'Nguyen%'

Ngôn ngữ thao tác dữ liệu
Toán tử truy vấn (11)

- Toán tử IN
  - **Ví dụ**

a. SELECT * FROM NHANVIEN

WHERE PHONG IN ('NC','QL','DH')

b. SELECT MANV, TENNV, DIACHI

FROM NHANVIEN

WHERE MANV NOT IN (SELECT MANV FROM PHANCONG)

Ngôn ngữ truy vấn dữ liệu
Toán tử truy vấn (12)

- Toán tử EXISTS
  - Ví dụ: tìm nhân viên làm việc cho tất cả các đề án

```
SELECT * FROM NHANVIEN n  WHERE NOT
  EXISTS

 (SELECT * FROM DEAN d  WHERE NOT
  EXISTS

   (SELECT * FROM PHANCONG p WHERE
 n.MANV=p.MANV AND d.MADA=p.MADA))
```

Ngôn ngữ truy vấn dữ liệu
Mệnh đề GROUP BY (13)

- Mệnh đề GROUP BY
  - Chia các dòng thành các nhóm nhỏ dựa trên tập thuộc tính chia nhóm.
  - Tất cả các thành viên của nhóm đều thỏa các thuộc tính này.
  - Thực hiện các phép toán trên nhóm như: Count (thực hiện phép đếm), Sum (tính tổng), Min(lấy giá trị nhỏ nhất), Max(lấy giá trị lớn nhất), AVG (lấy giá trị trung bình).

Ngôn ngữ truy vấn dữ liệu
Mệnh đề GROUP BY (14)

Quan hệ NV

Q	S
a	10
a	2
b	9
b	5
c	10
c	8
c	6
c	4
c	10
d	16
d	18
d	50

nhóm

Chia các dòng thành các nhóm dựa trên tập thuộc tính chia nhóm

Q	Count(S)
a	2
b	2
c	5
d	3

Tương tự cho các hàm SUM, MIN, MAX, AVG

Các thuộc tính GROUP BY: Q

Câu SQL:

```
Select Q, count(S)
From NV
Group by Q
```

Ngôn ngữ truy vấn dữ liệu
Mệnh đề GROUP BY (15)

- Các hàm SQL cơ bản
  - COUNT: Đếm số bộ dữ liệu của thuộc tính
  - MIN: Tính giá trị nhỏ nhất
  - MAX: Tính giá trị lớn nhất
  - AVG: Tính giá trị trung bình
  - SUM: Tính tổng giá trị các bộ dữ liệu

Ngôn ngữ truy vấn dữ liệu
Mệnh đề GROUP BY (16)

- Ví dụ: tìm tổng lương, lương lớn nhất, lương ít nhất và lương trung bình của các nhân viên

```
SELECT SUM(LUONG), MAX(LUONG), MIN(LUONG), AVG(LUONG) FROM NHANVIEN ;
```

- Ví dụ: tìm tổng lương, lương lớn nhất, lương ít nhất và lương trung bình của các nhân viên phòng "Nghiên cứu"

```
SELECT SUM(LUONG), MAX(LUONG), MIN(LUONG), AVG(LUONG) from NHANVIEN , PHONGBAN WHERE MAPHG=PHONG AND TENPHG='Nghien cuu';
```

- Ví dụ: cho biết số lượng nhân viên

```
SELECT COUNT(*) FROM NHANVIEN;
```

Ngôn ngữ truy vấn dữ liệu
Mệnh đề GROUP BY (17)

- Ví dụ

```
SELECT n.MANV, n.TENNV, n.PHONG,
MIN(p.THOIGIAN)  thap_nhat,
MAX (p.THOIGIAN) cao_nhat, AVG(p.THOIGIAN)
trung_binh,
SUM (p.THOIGIAN) tong_so_gio
FROM NHANVIEN n, PHANCONG p
WHERE n.MANV=p.MANV
GROUP BY n.MANV, n.TENNV, n.PHONG
```

Ngôn ngữ truy vấn dữ liệu
Mệnh đề HAVING (18)

- Mệnh đề HAVING
  - Lọc kết quả theo điều kiện, sau khi đã gom nhóm
  - Điều kiện của HAVING là điều kiện các hàm tính toán trên nhóm (Count, Sum, Min, Max, AVG) và các thuộc tính trong danh sách GROUP BY.

Ngôn ngữ truy vấn dữ liệu
Mệnh đề HAVING (19)

- **Ví dụ:**

nh của phòng trên 2000000

```
SELECT p.TENPHG, COUNT(*) so_luong_nv,
AVG(n.LUONG) luong_tb
FROM NHANVIEN n, PHONGBAN p
WHERE n.PHONG = p.MAPHG
GROUP BY p.TENPHG
HAVING AVG(n.LUONG) > 2000000
```

Ngôn ngữ truy vấn dữ liệu
Mệnh đề HAVING (20)

- Ví dụ
  - Liệt kê nhân viên có số giờ làm việc nhiều nhất trong công ty.

```
SELECT n.MANV, n.TENNV, SUM (p.THOIGIAN)
FROM NHANVIEN n, PHANCONG p
WHERE n.MANV = p.MANV
GROUP BY n.MANV, n.HOTEN
HAVING SUM (p.THOIGIAN)  >= ALL (SELECT SUM(THOIGIAN)
FROM PHANCONG GROUP BY (MANV)
```

Ngôn ngữ điều khiển dữ liệu (DCL)
Giới thiệu System và Object Privileges (1)

- Ngôn ngữ điều khiển dữ liệu Data Control Language (DCL) được dùng để cấp phát, thu hồi quyền trên các đối tượng như: tables, views, sequences, synonyms, procedures đến các user và role.

- Lệnh DCL bao gồm 2 lệnh:
  - GRANT :Use to **grant privileges on tables, view, procedure** to other users or roles.
  - REVOKE :Use to take back privileges granted to other users and roles.

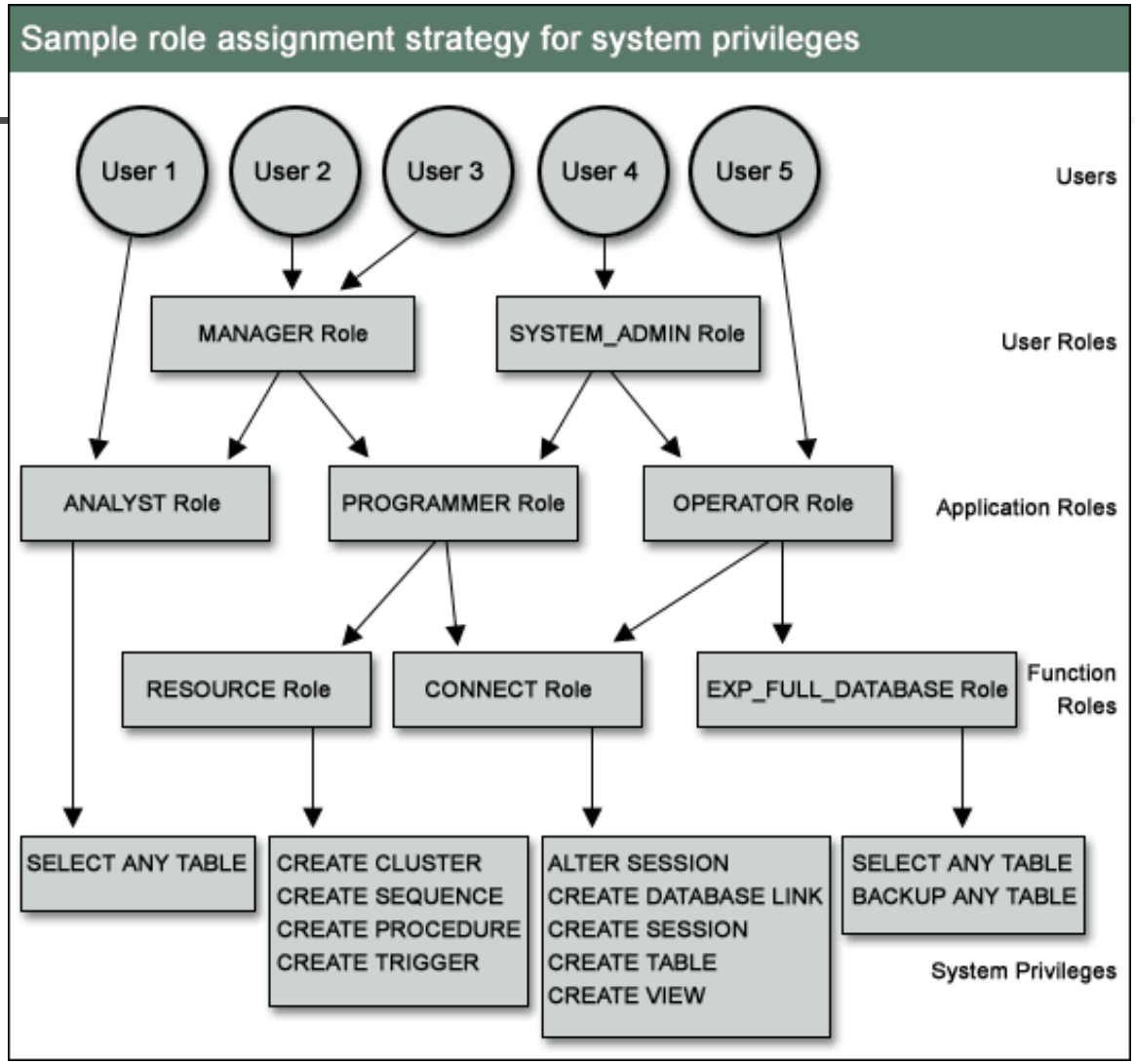
- Phân loại quyền: có 2 loại
  - **System Privileges**
  - **Object privileges**

- System Privileges thường được cấp bởi DBA đến các users. Ví dụ một số quyền hệ thống: CREATE SESSION, CREATE TABLE, CREATE USER,…. System privileges grants cover many areas of access in a broad brush, với những quyền tương tự như select any table. Ví dụ:

```
grant create any cluster to customer_role;
grant select any table to fred;
grant create tablespace to dba_role;
```

- Object privileges là các quyền trên các object như: tables, views, synonyms, procedure. Những quyền này được cấp bởi người sở hữu object.

Ngôn ngữ điều khiển dữ liệu



Ngôn ngữ điều khiển dữ liệu
Object Privileges (2)

Object Privileges are

ALTER	Change the table definition with the ALTER TABLE statement.
DELETE	Remove rows from the table with the DELETE statement. **Note:** You must grant the SELECT privilege on the table along with the DELETE privilege.
INDEX	Create an index on the table with the CREATE INDEX statement.
INSERT	Add new rows to the table with the INSERT statement.
REFERENCES	Create a constraint that refers to the table. You cannot grant this privilege to a role.
SELECT	Query the table with the SELECT statement.
UPDATE	Change data in the table with the UPDATE statement.
	Note: You must grant the SELECT privilege on the table along with the UPDATE privilege.

Ngôn ngữ điều khiển dữ liệu
Giới thiệu Object Privileges (3)

- **Một User muốn tạo được schema riêng cho mình cần phải có 2 quyền hệ thống thích hợp là CONNECT đến cơ sở dữ liệu và quyền RESOURCE (tài nguyên).**

- **Object Privileges:** có 8 quyền có thể gán cho User và Role, 8 quyền bao gồm: SELECT, INSERT, UPDATE, DELETE, ALTER, INDEX, REFERENCES, ALL.
- **Role:** là tên của một tập hợp các quyền hệ thống nhằm quản lý các quyền cho các ứng dụng cơ sở dữ liệu hoặc nhóm người dùng (User Group).
- Một User có thể truy xuất đến nhiều Role và ngược lại.
- Có vài Role được định nghĩa trước như DBA chứa tất cả các quyền của hệ thống.

Ngôn ngữ điều khiển dữ liệu
Tạo Role (nhóm quyền) (4)

- **Cú pháp**

  CREATE ROLE <role-name> NOT IDENTIFIED | IDENTIFIED BY <password>

  Trong đó:

  - NOT IDENTIFIED: không đặt password cho role.
  - IDENTIFIED BY password: user phải nhập password để làm cho role có hiệu lực.

  Lưu ý: ký tự đầu tiên của password không được là số hoặc ký tự đặc biệt

  - Ví dụ: tạo một role QuanTriSV với password là QT987654

  CREATE ROLE QuanTriSV IDENTIFIED BY QT987654

Ngôn ngữ điều khiển dữ liệu

Thiết lập/thay đổi/bỏ mật khẩu cho Role + xóa Role (5)

- **Thiết lập/thay đổi/bỏ mật khẩu cho quyền**

  Cú pháp: ALTER ROLE <role-name> NOT IDENTIFIED | IDENTIFIED BY <password>

  Ví dụ: đổi password mới cho role QuanTriSV là QT123456789

  ALTER ROLE QuanTriSV IDENTIFIED BY QT123456789

- **Xóa quyền**

  Cú pháp: DROP ROLE <role-name>

  Ví dụ: xóa quyền QuanTriSV

  DROP ROLE QuanTriSV

- **Gán quyền select/update/delete,.. đến public**

  grant select on TênUser.TênTable to public

Ngôn ngữ điều khiển dữ liệu
Tạo người dùng (6)

- **Cú pháp**

  CREATE USER <user-name> NOT IDENTIFIED | IDENTIFIED BY <password> default tablespace USERS

  Trong đó:

  - NOT IDENTIFIED: không đặt password cho user.
  - IDENTIFIED BY password: user phải nhập password mới có thể đăng nhập hệ thống.

  - Ví dụ: tạo một user Phuong với password là P987654

  CREATE USER Phuong IDENTIFIED BY P987654

  CREATE USER Phuong1 IDENTIFIED BY P987654 **default tablespace TEMP**

  CREATE USER Phuong1 IDENTIFIED BY P987654 **default tablespace USERS**

Ngôn ngữ điều khiển dữ liệu

Thiết lập/thay đổi/bỏ mật khẩu cho người dùng + xóa người dùng (7)

- **Thiết lập/thay đổi/bỏ mật khẩu cho người dùng**

  Cú pháp: ALTER USER <User-name> NOT IDENTIFIED | IDENTIFIED BY <password>

  Ví dụ: thay đổi password mới cho user Phuong là P123456789

  ALTER USER Phuong IDENTIFIED BY P123456789

- **Xóa người dùng**

  Cú pháp: DROP USER <User-name>

  Ví dụ: xóa người dùng Phuong

  DROP USER Phuong

Ngôn ngữ điều khiển dữ liệu
Cấp phát quyền (8)

- **Cú pháp cấp phát quyền cho người dùng**

  GRANT privil1, privil2,.../ALL ON <Tênđốitượng>
  TO User1, User2,... [WITH GRANT OPTION]

- **Cú pháp cấp phát quyền cho Role**

  GRANT privil1, privil2,.../ALL ON <Tênđốitượng>
  TO Role1, Role2,... [WITH GRANT OPTION]

Trong đó:

- **<Tênđốitượng>** : có thể là tên của một Table, View, Sequence, Synonym, Procedure, Function, Package.
- **privil1, privil1,..**là 1 trong 8 quyền hệ thống nêu trên để cấp quyền trên table hay view.

Ngôn ngữ điều khiển dữ liệu
Cấp phát quyền (9)

- **Ban quyền truy xuất đến tất cả User, Role bằng lệnh**

  GRANT privil1, privil2,.../ALL ON <Tênđốitượng>

  TO PUBLIC

  Ví dụ: grant select on TênUser.TênTable to public
- Ví dụ 1: cấp phát quyền cho user Phuong

  GRANT INSERT, UPDATE ON TênUserTạoTableNhanVien.NHANVIEN TO Phuong
- Ví dụ 2: cấp phát quyền cho role QuanTriSV

  GRANT ALL ON TênUserTạoTableNhanVien.NHANVIEN TO QuanTriSV WITH GRANT ADMIN

Ngôn ngữ điều khiển dữ liệu
Cấp phát quyền (10)

- **Gán quyền Role cho User**
  (User được cấp phát quyền Role)
  GRANT tên-Role TO tên-User [WITH GRANT OPTION]

- Ví dụ : cấp phát role QuanTriSV cho user Phuong
  GRANT QuanTriSV TO Phuong

- Ví dụ 2: cấp phát role QuanTriSV cho user Phuong
  GRANT QuanTriSV TO Phuong WITH GRANT ADMIN

Ngôn ngữ điều khiển dữ liệu
Thu hồi quyền (11)

- **Cú pháp rút lại (hủy bỏ) các quyền đã cấp phát**

  REVOKE privil1, privil2,…/Role1, Role2,… ON <Tên-table>/<Tên-view>

  FROM User1, User2,…/Role1, Role2,…

- Ví dụ
  - REVOKE UPDATE, DELETE ON NHANVIEN FROM Phuong
  - REVOKE ALL ON NHANVIEN FROM QuanTriSV

Ngôn ngữ điều khiển dữ liệu
VD đoạn PL/SQL cấp quyền (12)

-- This will grant read only access on all your objects to another schema

```
begin
    for x in (select object_name from
user_objects)
        loop
        execute immediate 'grant select on
'|| x.object_name || ' to &schema';
        end loop;
end;
```

Ngôn ngữ điều khiển dữ liệu
Tuy vấn xem thông tin quyền (13)

- To see which table privileges are granted by you to other users.
  SELECT * FROM USER_TAB_PRIVS_MADE
- To see which table privileges are granted to you by other users
  SELECT * FROM USER_TAB_PRIVS_RECD;
- To see which column level privileges are granted by you to other users.
  SELECT * FROM USER_COL_PRIVS_MADE
- To see which column level privileges are granted to you by other users
  SELECT * FROM USER_COL_PRIVS_RECD;
- To see which privileges are granted to roles
  SELECT * FROM USER_ROLE_PRIVS;

6. CÁC LỆNH GIAO TÁC (1) – TRANSACTION

GIAO TÁC : một Transaction là một giao tác trên CSDL bao gồm chuỗi các thay đổi (hành động) trên một hay nhiều table. Điều khiển Transaction bằng các lệnh sau.

- **Lệnh COMMIT** (hoàn tất giao tác)

  Cú pháp: COMMIT
- **Lệnh ROLLBACK** (phục hồi ngược lại chuỗi hành động đã thực hiện trước đó).

  Cú pháp: ROLLBACK [TO SAVEPOINT *name*]
- **Lệnh SavePoint**

  Cú pháp: SAVEPOINT tên-SavePoint
- **Chế độ AutoCommit** (AUTOCOMMIT ON/OFF)

6. CÁC LỆNH GIAO TÁC (2) – TRANSACTION

- **Ví dụ: minh họa cách dùng các lệnh SavePoint, RollBack, Commit**

```
begin
insert into student values(1,'Nguyen Van A');
savepoint A;
insert into student values(2,'Nguyen Van B');
savepoint B;
insert into student values(3,'Nguyen Van C');
savepoint C;
insert into student values(4,'Nguyen Van D');
savepoint A;  // savepoint A trước đó ko đc hiểu nữa
insert into student values(5,'Nguyen Van E');
savepoint D;
end;
```

- select * from student; -> kết quả 5 sinh viên
- rollback to savepoint A; //thải hồi lệnh ngược đến savepoint A
- select * from student; -> kết quả 4 sinh viên

6. CÁC LỆNH GIAO TÁC (3) – TRANSACTION

Giá trị nào được thêm vào bảng NUMBERS?
Test trường hợp khác sử dụng lệnh drop table, lệnh DROP có được Rollback??

```
BEGIN
    FOR IX IN 9..12 LOOP
        IF IX = 9 THEN
            INSERT INTO NUMBERS VALUES (IX);
        ELSIF IX = 11 THEN
            DELETE FROM NUMBERS;
        END IF;
        IF IX = 11 THEN
            ROLLBACK;
        ELSE
            COMMIT;
        END IF;
    END LOOP;
    COMMIT;
END;
```

7. CÁC ĐỐI TƯỢNG KHÁC – Synonym (1)

- **Synonym:**

  Là tên đặt cho một đối tượng cụ thể nào đó. Thường dùng synonym để tạo ra những đối tượng dùng chung.

- **Ví dụ:**

  Một người dùng muốn sử dụng một bảng được sở hữu bởi người dùng khác thay vì gọi "tên_người_dùng.tên_bảng" thì user đó tạo ra một synonym cho bảng đó với một tên dễ nhớ nào đó, tên thật của đối tượng được che dấu đi.

- Các thao tác trên Synonym: tạo Synonym, xóa Synonym.

7. CÁC ĐỐI TƯỢNG KHÁC – Synonym (2)

- **Tạo Synonym:**

  Cú pháp:

  CREATE SYNONYM tênSynonym **FOR** tênUser**.**tênTable|tênView

  Ví dụ:

  CREATE SYNONYM nv **FOR** phuong.NHANVIEN;

- **Xóa Synonym:**

  Cú pháp:

  DROP SYNONYM tên-synonym;

  Ví dụ:

  DROP SYNONYM nv**;**

7. CÁC ĐỐI TƯỢNG KHÁC – Index (1)

- **Index (tạo chỉ mục): sử dụng Oracle Index nhằm**

- Tăng tốc độ xây dựng lại các dòng theo một khóa đặc biệt.
- Bảo đảm giá trị duy nhất trong cột, thường là giá trị primary key.

- **Tạo Index**

Cú pháp: CREATE [UNIQUE] INDEX index_name ON table (column1 [, <column2>] , …)

Ví dụ:

CREATE UNIQUE INDEX i_cmnd ON NHAN_KHAU(SO_CMND);

7. CÁC ĐỐI TƯỢNG KHÁC – Index (2)

- **Phân loại index:**

- UNIQUE: Bảo đảm giá trị trong các cột là duy nhất.
  (no two rows of a table have duplicate values in the key column (or columns))

- NONUNIQUE: (là default) kết quả truy vấn có thể nhanh nhất (do not impose this restriction on the column values)

* Nếu là Single colum thì chỉ một cột tồn tại index.
* Nếu là Concatenated Index: Trên 16 cột có thể chỉ ra trong index (giới hạn 30).

7. CÁC ĐỐI TƯỢNG KHÁC – Index (3)

(concatenated index also is called Composite Index - index that you create on multiple columns in a table)

VENDOR_PARTS

VEND ID	PART NO	UNIT COST
1012	10–440	.25
1012	10–441	.39
1012	457	4.95
1010	10–440	.27
1010	457	5.10
1220	08–300	1.33
1012	08–300	1.19
1292	457	5.28

Concatenated Index
(index with multiple columns)

```
CREATE INDEX VP_INDEX
ON
VENDOR_PARTS(VEND_ID,
PART_NO);
```

- **Xóa index**

Cú pháp: DROP INDEX index_name;

Ví dụ: DROP INDEX i_cmnd;

7. CÁC ĐỐI TƯỢNG KHÁC – Sequence (1)

- **Sequence (giá trị được tạo tự động)**

  Sequence là đối tượng tạo ra một dãy số liên tiếp một cách tự động, thường hay sử dụng trong câu lệnh INSERT để nhập dữ liệu cho bảng (ví dụ mã khách hàng tăng tự động).

- **Tạo Sequence**

  Cú pháp:

```
CREATE SEQUENCE sequence_name
[INCREMENT BY n]
[START WITH n]
[MAXVALUE n | NOMAXVALUE]
[MINVALUE n | NOMINVALUE]
[CYCLE | NOCYCLE]
[CACHE n | NOCACHE]
[ORDER | NOORDER]
```

7. CÁC ĐỐI TƯỢNG KHÁC – Sequence (2)

- **Ví dụ**

```
CREATE SEQUENCE s_nv
MINVALUE 1
MAXVALUE 9999999
INCREMENT BY 1
START WITH 100
NOCACHE
NOORDER
NOCYCLE;
```

- **Sử dụng Sequence**

Giá trị hiện hành và kế tiếp của dãy sequence được lưu trong hai cột:

- Giá trị hiện hành của sequence: tên_sequence.CURRVAL
- Giá trị kế tiếp của sequence: tên_sequence.NEXTVAL

7. CÁC ĐỐI TƯỢNG KHÁC – Sequence (3)

Hai cột trên trong Oracle được gọi là **pseudo column (cột ảo)**, người dùng chỉ được truy xuất, không được cập nhật dữ liệu trong các cột này.

Oracle có các cột ảo như: **ROWID (mã dòng), ROWNUM (số thứ tự dòng), SYSDATE (ngày hiện hành của hệ thống),...**

- **Ví dụ**

Truy cập giá trị tiếp theo của sequence s_nv bằng lệnh:

SELECT s_nv.nextval from DUAL; (bảng DUALlà tạm trong Oracle)

Truy vấn giá trị hiện tại của sequence s_nv (nếu ko gọi s_nv.nextval một lần khi login vào session của mình trước, xem lỗi ở slide kế tiếp):

SELECT s_nv.currval from DUAL;

Sử dụng sequence trong một câu insert:

INSERT INTO NHANVIEN (MA_NV, HO_TEN) VALUES (s_nv.nextval, 'Nguyen van A'); /* s_nv là sequence đã tạo trước. */

7. CÁC ĐỐI TƯỢNG KHÁC

– Lỗi truy xuất CURRVAL nếu chưa khởi tạo giá trị Sequence (4)

- **ORA-08002:** sequence *string*.CURRVAL is not yet defined in this session
- **Cause:** sequence CURRVAL has been selected before sequence NEXTVAL
  Action: select NEXTVAL from the sequence before selecting CURRVAL

- Link xem các loại lỗi (Oracle Error Code Collections): **http://www.ora-code.com/**

7. CÁC ĐỐI TƯỢNG KHÁC – Sequence (5)

- **Sửa Sequence (tạo lại các thông số)**

  ALTER SEQUENCE tên-sequence
  [INCREMENT BY n]
  [START WITH n]
  [MAXVALUE n | NOMAXVALUE]
  [MINVALUE n | NOMINVALUE]
  [CYCLE | NOCYCLE]
  [CACHE n | NOCACHE]
  [ORDER | NOORDER]

- **Xóa Sequence**

  Cú pháp **DROP** SEQUENCE tên-sequence

  Ví dụ: **DROP** SEQUENCE s_nv

8. TỪ ĐIỂN DỮ LIỆU – Oracle data dictionary

MỘT SỐ VIEW TỪ ĐIỂN THƯỜNG DÙNG

- USER_TABLES : xem các table người dùng đã tạo
- USER_VIEWS : xem các view người dùng đã tạo
- USER_INDEXES : xem các index người dùng tạo
- USER_SEQUENCES : xem các sequence người dùng tạo
- USER_FUNCTIONS : xem các function người dùng tạo
- USER_PROCEDURES : xem các procedure người dùng tạo
- USER_TRIGGERS : xem các trigger người dùng đã tạo
- user_sys_privs: xem quyền của user hiện hành
- .....

MỘT SỐ HÀM THƯỜNG DÙNG TRONG ORACLE

- HỌC VIÊN XEM FILE WORD ĐI KÈM

Quản trị CSDL và Phần mềm ứng dụng

Bộ môn CNTT
Khoa Tin học Thương Mại

Mục tiêu môn học

- Trang bị kiến thức cơ bản về CSDL.
- Giới thiệu phương pháp thiết kế, xây dựng CSDL quan hệ, ngôn ngữ SQL.
- Trang bị những hiểu biết cơ bản về thao tác với CSDL thông qua một phần mềm ứng dụng quản trị CSDL quan hệ

Yêu cầu môn học

- Nghe giảng
  - Giờ lý thuyết : 30 tiết
- Thảo luận + Thực hành
  - Giờ thảo luận + thực hành: 6 tiết
- Đọc tài liệu tham khảo

Tài liệu tham khảo

- [1] *Giáo trình quản trị CSDL và phần mềm ứng dụng*, Trường Đại học Thương mại.
- [2]*Nhập môn CSDL quan hệ.* Lê Tiến Vương, NXB Thống kê, 2002.
- [3] *Access 2002 Bible.* Cary N.Prague & Michael R.Irwin, New York. NY, 2002
- [4] *Principles of Database Systems.* Ullman, J.D, Computer Sience Press, Rockville, Md. 1982.
- [5] *Nguyên lý của các hệ CSDL.* Nguyễn Kim Anh, NXB Đại học Quốc gia Hà Nội, 2004.

Nội dung lý thuyết

Chương 1: **Tổng quan về CSDL**

Chương 2: **Thiết kế CSDL quan hệ**

Chương 3: **Ngôn ngữ SQL**

Chương 4: **Phần mềm ứng dụng quản trị CSDL**

Chương I: Tổng quan về CSDL

- **1.CÁC KHÁI NIỆM CƠ BẢN**
  - 1.1.CSDL
  - 1.2.Những người sử dụng CSDL
  - 1.3.Hệ quản trị CSDL
  - 1.4.Hệ CSDL
- **2.KIẾN TRÚC CỦA MỘT HỆ CSDL**
  - 2.1.Các mức trừu tượng
  - 2.2.Lược đồ CSDL
  - 2.3.Tính độc lập dữ liệu
- **3.CÁC MÔ HÌNH DỮ LIỆU**
  - 3.1. Mô hình thực thể liên kết (ER)
    - 3.1.1.Thực thể và liên kết
    - 3.1.2.Sơ đồ thực thể liên kết
  - 3.2. Mô hình dữ liệu quan hệ
    - 3.2.1. Các khái niệm trong mô hình quan hệ
    - 3.2.2. Biến đổi sơ đồ ER sang lược đồ quan hệ

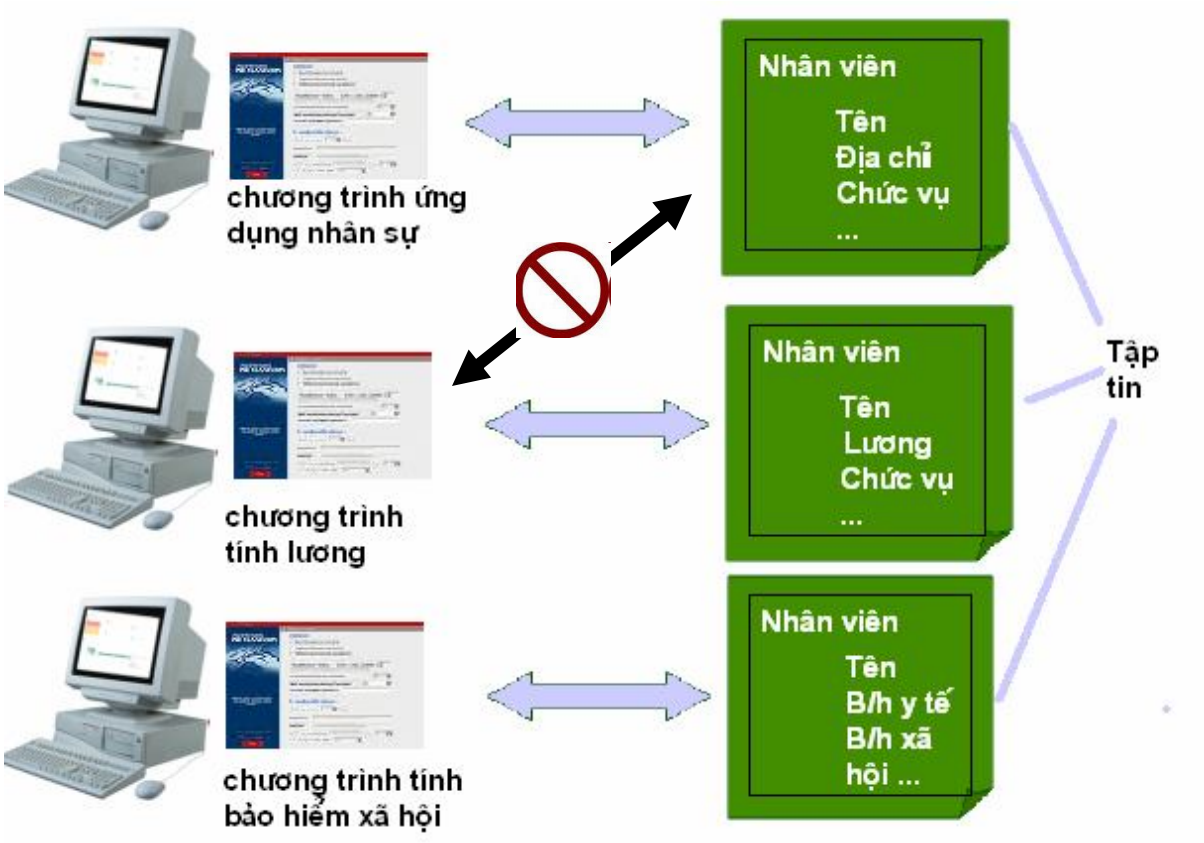
1.1. CSDL

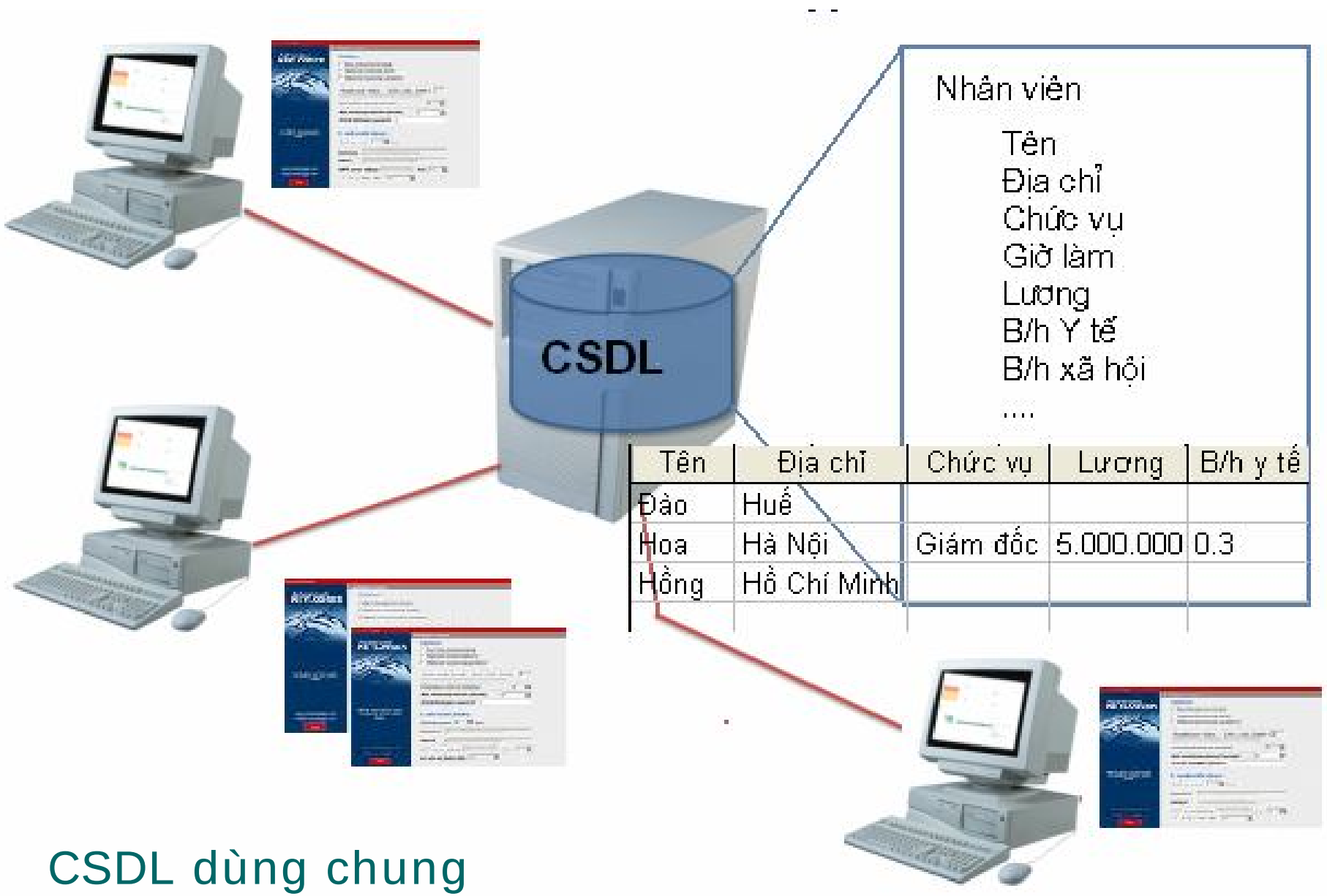
- Định nghĩa:
  - Dữ liệu: phản ánh một sự vật hiện tượng trong thế giới khách quan được lưu trữ trong máy tính dưới dạng có cấu trúc (bản ghi) hoặc không có cấu trúc (hình ảnh, âm thanh)
  - CSDL: tập hợp dữ liệu tương quan có tổ chức được lưu trữ trên các phương tiện lưu trữ như đĩa từ, băng từ v..v nhằm thỏa mã các yêu cầu khai thác thông tin (đồng thời) của nhiều người sử dụng và của nhiều chương trình ứng dụng.

Đặc tính của CSDL

- **Chia sẻ** – tức CSDL cho phép nhiều người dùng, nhiều ứng dụng.
- **Bền vững** – tức dữ liệu được đặt trên thiết bị lưu trữ ổn định, cho phép sử dụng lại nhiều lần

Trước khi có CSDL





CSDL dùng chung

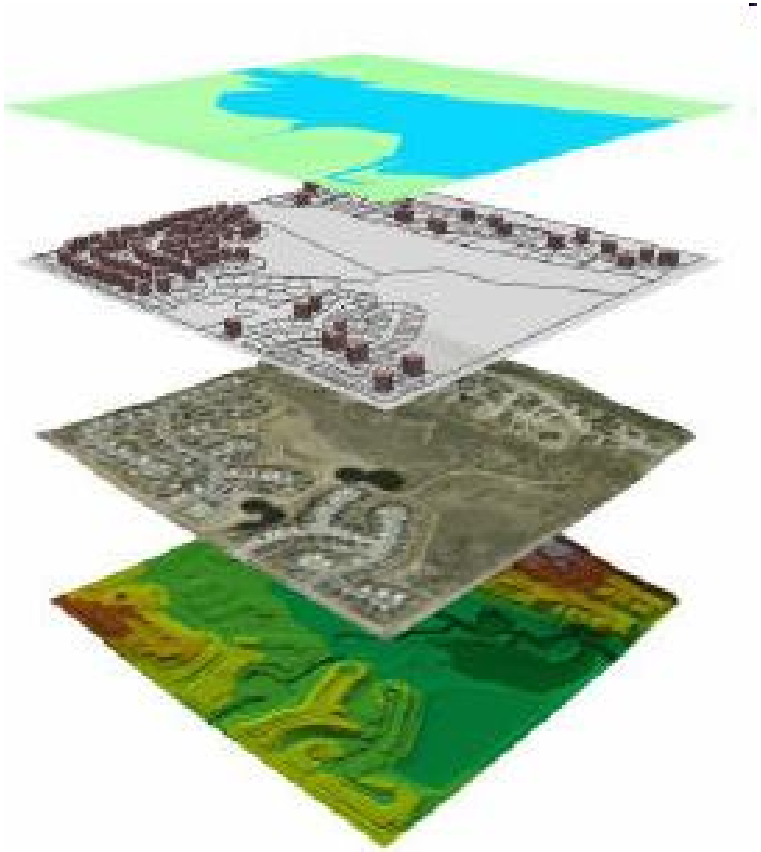
CSDL trong các ứng dụng

- Thông dụng
  - Csdl trong các ứng dụng quản lý nguồn nhân lực, dịch vụ công cộng (điện, nước...)
- Thương mại điện tử
  - Csdl khách hàng, sản phẩm hàng hoá... của các cửa hàng trực tuyến (B2C) sàn giao dịch trực tuyến(B2B)
- Chuyên biệt
  - Csdl lưu trữ dữ liệu của các ứng dụng chuyên biệt như hệ thống thông tin địa lý (GIS_ Geographic Information System)

- CSDL lưu trữ dưới dạng bảng (hay quan hệ) là hình thức lưu trữ nhiều nhất và được ứng dụng rộng rãi nhất hiện nay.

Ngày	Số HĐ	Sản phẩm & dịch vụ	ĐVT	Giá vốn	Giá bán	Số lượng	Thàn
20/10/2007	HD2	CANH CHUA CÁ LÓC	cái	35,000	65,000	1	
20/10/2007	HD2	NƯỚC ÉP TRÁI CÂY	ly	8,000	14,000	3	
20/10/2007	HD2	MỰC NƯỚNG MUỐI ỚT	dĩa	30,000	45,000	1	
20/10/2007	HD2	COM ĐÙI GÀ	dĩa	15,000	25,000	4	
20/10/2007	HD3	MỰC NƯỚNG MUỐI ỚT	dĩa	30,000	45,000	2	
20/10/2007	HD3	CANH CHUA CÁ LÓC	cái	35,000	65,000	1	
20/10/2007	HD3	BIA TIGER	lon	7,000	14,000	5	
20/10/2007	HD4	BIA TIGER	lon	7,000	14,000	10	
20/10/2007	HD4	CANH CHUA CÁ LÓC	cái	35,000	65,000	1	
20/10/2007	HD4	MỰC NƯỚNG MUỐI ỚT	dĩa	30,000	45,000	1	
20/10/2007	HD4	COM ĐÙI GÀ	dĩa	15,000	25,000	3	
20/10/2007	HD5	BIA 333	lon	6,000	12,000	24	
20/10/2007	HD5	MỰC NƯỚNG MUỐI ỚT	dĩa	30,000	45,000	4	
20/10/2007	HD5	CANH CHUA CÁ LÓC	cái	35,000	65,000	5	
20/10/2007	HD6	MỰC NƯỚNG MUỐI ỚT	dĩa	30,000	45,000	1	
20/10/2007	HD6	CANH CHUA CÁ LÓC	cái	35,000	65,000	2	
20/10/2007	HD6	BIA TIGER	lon	7,000	14,000	2	
20/10/2007	HD7	NƯỚC ÉP TRÁI CÂY	ly	8,000	14,000	1	

- Csdl thông tin địa lý lưu trữ
  - Dữ liệu bản đồ (hình ảnh bản đồ)
  - Dữ liệu thuộc tính (mô tả đặc tính, đặc điểm và các hiện tượng xảy ra tại một vị trí địa lý cụ thể)
- Mỗi lớp trong dữ liệu bản đồ lưu trữ một bản đồ liên quan đến 1 chức năng cụ thể



Tiêu chuẩn của CSDL

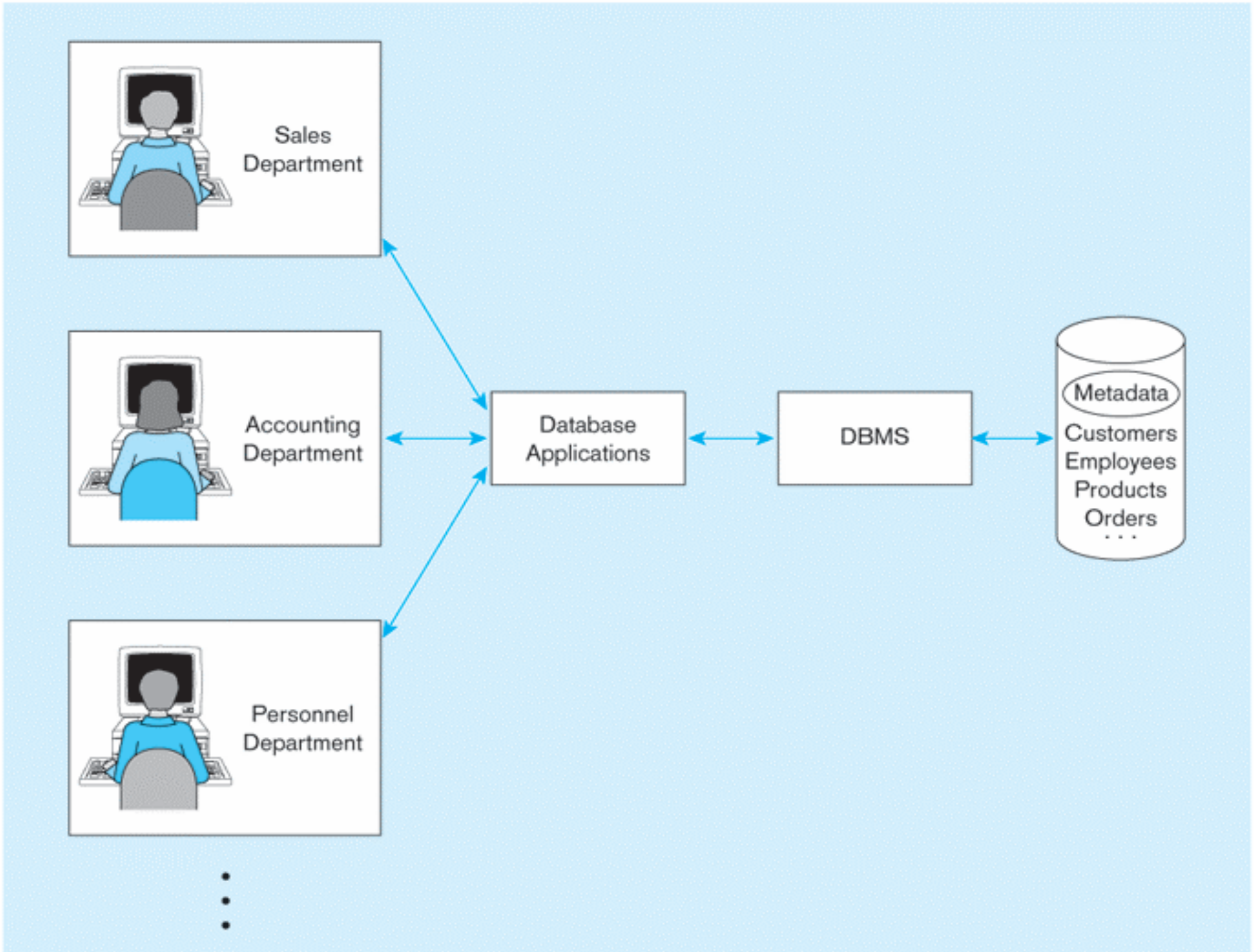
- Giảm việc dư thừa: Mỗi một ứng dụng không cần phải có các tập tin dữ liệu của riêng nó
- Tránh được sự không nhất quán dữ liệu
- Dữ liệu được chia sẻ
- Áp dụng các chuẩn nghiêm ngặt
- Áp dụng các biện pháp an toàn bảo mật
- Các ràng buộc phải được duy trì

1.2. Người sử dụng CSDL

- Người dùng cuối: Khai thác CSDL thông qua các ứng dụng hoặc dựa trên ***phần mềm quản trị CSDL***.
- Người lập trình ứng dụng: Là người viết các chương trình ứng dụng cho phép người sử dụng cuối sử dụng CSDL.
- Người quản trị CSDL (Database Administrator): Là người thu thập dữ liệu, *thiết kế và bảo trì CSDL, thiết lập các cơ chế đảm bảo an toàn cho CSDL (sao lưu, phục hồi dữ liệu).*

1.3. Hệ quản trị CSDL

- Khái niệm
  - Hệ quản trị CSDL là một phần mềm cho phép tạo lập CSDL và điều khiển mọi truy nhập đối với CSDL đó.
- Đặc điểm
  - Quản lý dữ liệu lâu dài
  - Hỗ trợ truy nhập dữ liệu lớn một cách hiệu quả


Sales Department
Accounting Department
Personnel Department
Database Applications
DBMS
Metadata
Customers
Employees
Products
Orders

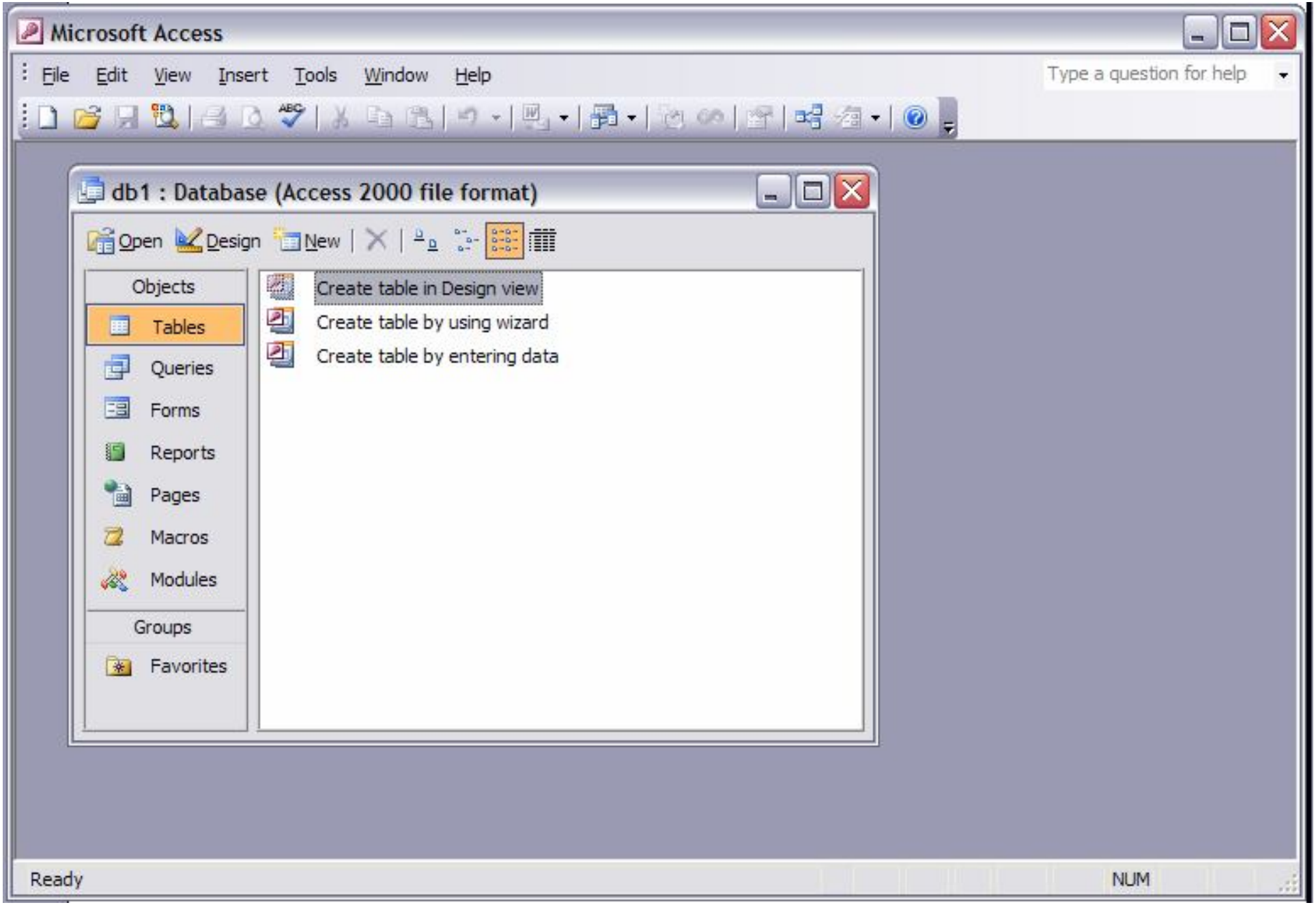
Chức năng của hệ QT CSDL

- *Hỗ trợ ít nhất một* ***cách tổ chức dữ liệu (mô hình dữ liệu)***
- Lưu trữ, truy xuất và cập nhật dữ liệu
- Quản lý giao dịch (*transaction management*)
- Điều khiển tương tranh (*concurrency control*)
- Chép lưu và phục hồi dữ liệu.
- Bảo mật dữ liệu
- Duy trì tính toàn vẹn/nhất quán dữ liệu.
- Cung cấp các tiện ích
- Hỗ trợ truyền thông dữ liệu

Phân loại hệ quản trị CSDL

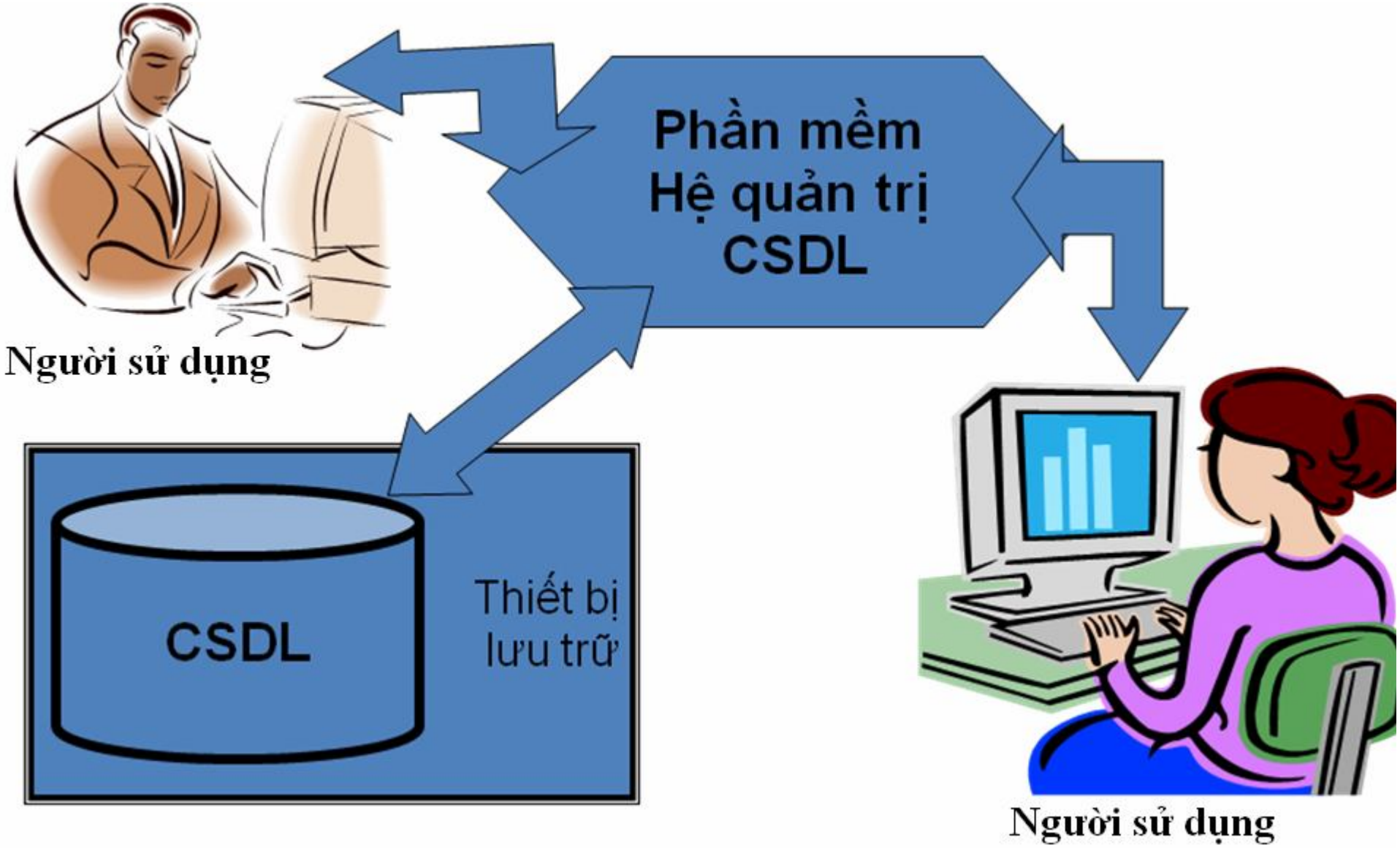
Dựa trên cách thức tổ chức dữ liệu

- Hệ QTCSDL phân cấp (IMS của IBM)
- Hệ QTCSDL mạng (IDMS của Cullinet Software)
- Hệ QTCSDL quan hệ
  - Cho máy tính cá nhân:Microsoft Access
  - Cho máy chủ: Microsoft SQL Server, MySQL, Oracle
- Hệ QTCSDL đối tượng (Ozone)

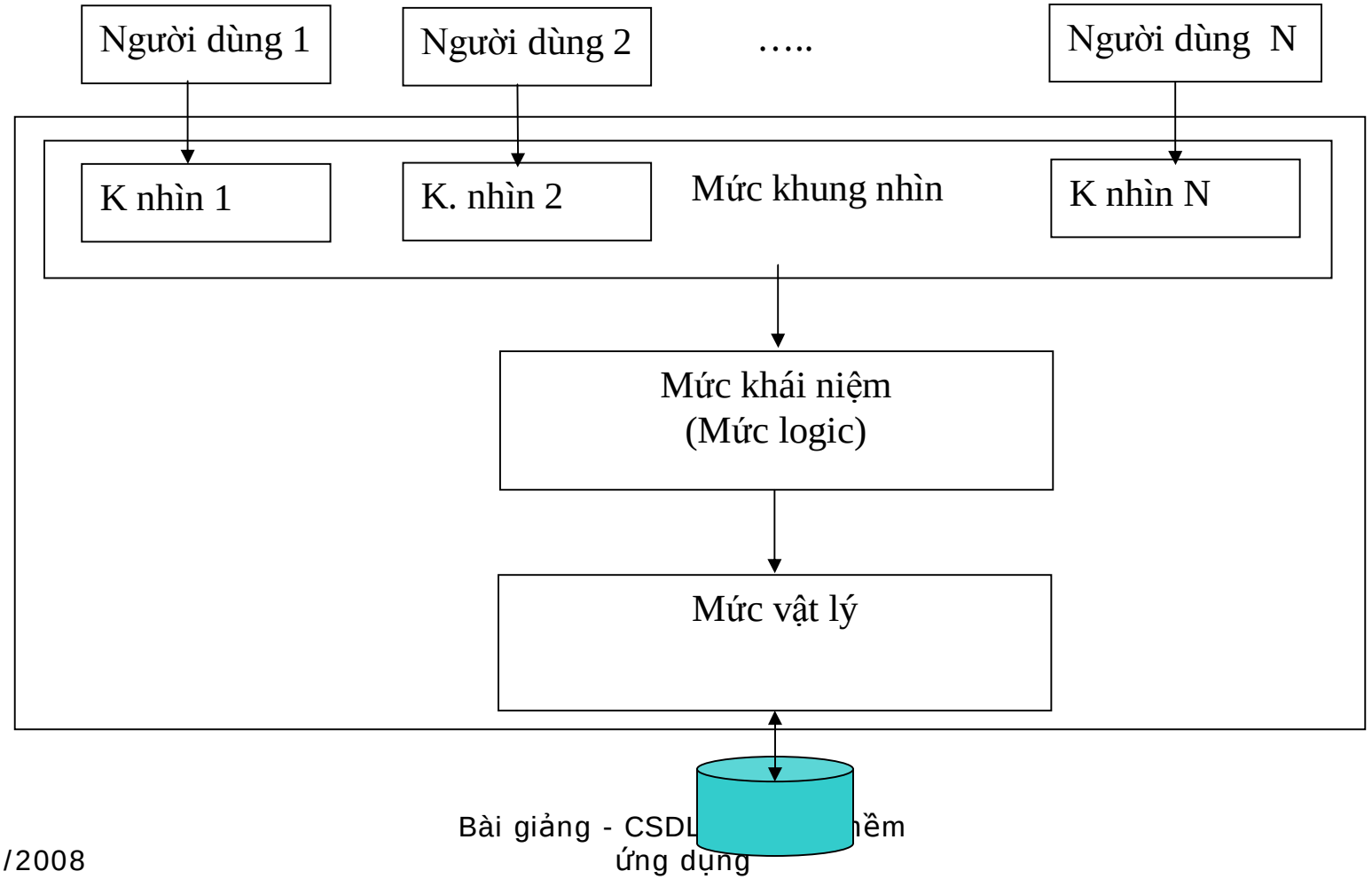

Microsoft Access
File Edit View Insert Tools Window Help
Type a question for help
db1 : Database (Access 2000 file format)
Open Design New
Objects
Tables
Queries
Forms
Reports
Pages
Macros
Modules
Groups
Favorites
Create table in Design view
Create table by using wizard
Create table by entering data
Ready
NUM

1.4. Hệ CSDL

- Khái niệm
  - Hệ CSDL là hệ thống gồm 4 thành phần
    - Người sử dụng
    - Hệ QTCSDL
    - CSDL
    - Phần cứng: Máy tính, ổ đĩa, băng từ dùng để lưu trữ dữ liệu


Phần mềm
Hệ quản trị
CSDL
Người sử dụng
CSDL
Thiết bị
lưu trữ
Người sử dụng

2. Kiến trúc hệ CSDL



2.1. Các mức trừu tượng

- Mức khung nhìn:
  - Mô tả chỉ một phần của toàn bộ CSDL. Hệ thống có thể cung cấp nhiều khung nhìn đối với cùng một CSDL.
- Mức khái niệm (logic):
  - Mô tả những dữ liệu nào được lưu trữ trong CSDL và các mối quan hệ nào tồn tại giữa các dữ liệu này.
- Mức vật lý:
  - Mô tả dữ liệu được lưu trữ như thế nào. Tại mức vật lý, các cấu trúc dữ liệu mức thấp phức tạp được mô tả chi tiết.

2.2. Lược đồ CSDL

- Một thiết kế tổng thể của CSDL được gọi là lược đồ CSDL
  - **Lược đồ khái niệm là bộ khung của CSDL mức khái niệm.**
  - Lược đồ vật lý là bộ khung của CSDL mức vật lý.
  - Lược đồ khung nhìn được gọi là lược đồ con.

2.3.Tính độc lập dữ liệu

- Khái niệm
  - Khả năng thay đổi một định nghĩa lược đồ trong một mức mà không ảnh hưởng đến định nghĩa lược đồ mức cao hơn tiếp theo được gọi là tính độc lập dữ liệu.
- Phân loại
  - Độc lập dữ liệu mức vật lý là khả năng thay đổi lược đồ mức vật lý mà không dẫn đến các chương trình ứng dụng phải viết lại. Các thay đổi tại mức vật lý đôi khi là cần thiết để tăng hiệu năng hệ thống.
  - Độc lập dữ liệu ở mức logic là khả năng thay đổi lược đồ mức logic mà không dẫn đến các chương trình ứng phải viết lại. Các thay đổi tại mức logic là cần thiết bất kể khi nào cấu trúc logic của CSDL bị sửa đổi.

Độc lập dữ liệu mức logic là khó đạt được hơn so với độc lập dữ liệu mức vật lý do các chương trình ứng dụng phụ thuộc nhiều vào cấu trúc logic của dữ liệu mà họ đang truy cập.

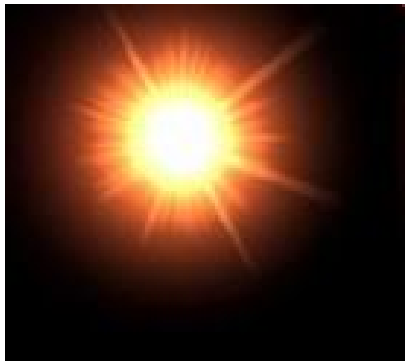
3. Mô hình dữ liệu

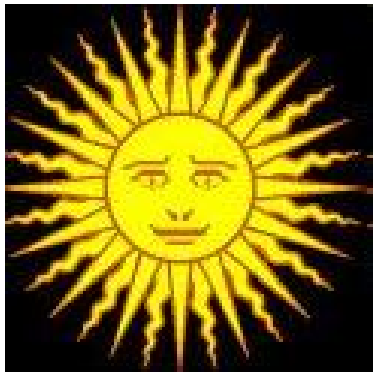
- Khái niệm
  - Mô hình dữ liệu là một mô tả của việc tổ chức dữ liệu trong CSDL: dữ liệu, ràng buộc được định nghĩa cho dữ liệu cùng quan hệ giữa các dữ liệu.
- Phân loại
  - Mô hình dữ liệu dựa trên đối tượng
    - Ví dụ: Mô hình thực thể liên kết ER (Entity-Relationship model)
  - Mô hình dữ liệu dựa trên bản ghi
    - Ví dụ: Mô hình dữ liệu phân cấp, mô hình dữ liệu mạng, mô hình dữ liệu quan hệ.

3.1. Mô hình thực thể liên kết

- Mô hình thực thể liên kết dựa trên cơ sở sự nhận thức của thế giới thực bao gồm một tập các đối tượng cơ sở được gọi là các **thực thể** và một tập các **liên kết** giữa các đối tượng này.

Tại sao dùng ER ?





- Mô hình thực thể liên kết là cách tiếp cận chính để mô hình hóa dữ liệu theo khái niệm (conceptual data modeling).
- Mô hình ER là công cụ giao tiếp giữa người sử dụng cuối cùng và người thiết kế CSDL để xây dựng CSDL trong giai đoạn phân tích.
- Mô hình ER được dùng để xây dựng mô hình dữ liệu theo khái niệm (conceptual data model) nhằm biểu diễn cấu trúc và các ràng buộc của CSDL.

3.1.1. Thực thể và liên kết

- Một thực thể là một đối tượng có thể được định nghĩa và dữ liệu về nó có thể được lưu trữ.
  - Đặc tính của thực thể gọi là **thuộc tính** của thực thể.
- Một liên kết là sự kết hợp giữa một số thực thể.
  - Liên kết 1- 1
  - Liên kết 1- nhiều
  - Liên kết nhiều- nhiều
  - Liên kết là-một

Biểu diễn tập thực thể ở mức sơ đồ

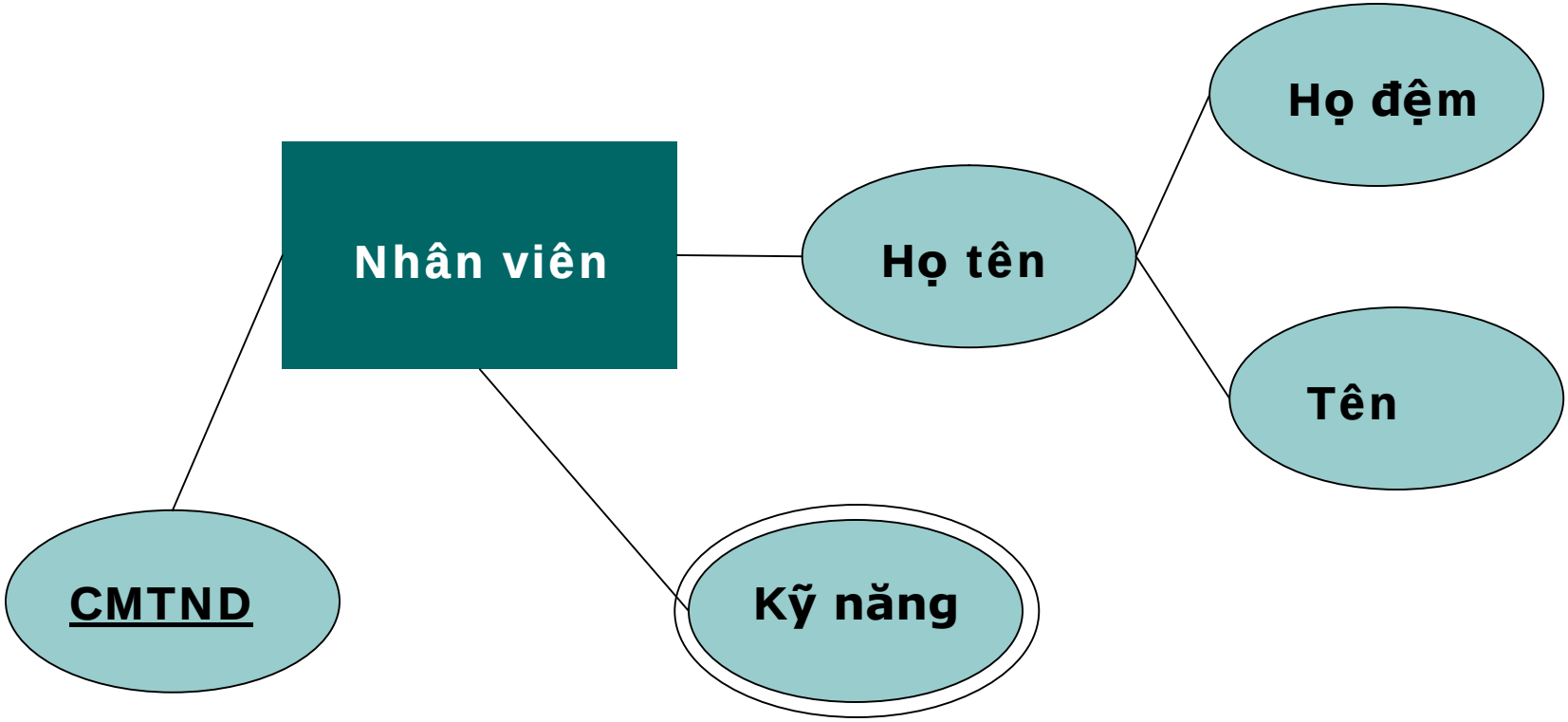
- Nhóm bao gồm tập thực thể *giống nhau (ít nhất cần có một tập các thuộc tính chung)* được gọi là **một tập thực thể (kiểu thực thể)**
- Biểu diễn

Các dạng thuộc tính

- ***Thuộc tính định danh / khóa*** :Xác định thực thể một cách duy nhất
  - Mô tả được gạch chân để phân biệt với các thuộc tính khác
  - Giá trị khác rỗng và thành phần không chứa các thông tin thay đổi
  - Khóa đơn, khóa phức, khóa dự tuyển, khóa chính
- ***Thuộc tính mô tả:*** Cung cấp thông tin làm rõ thêm về thực thể
  - Chỉ nên xuất hiện trong một kiểu thực thể tránh dư thừa dữ liệu

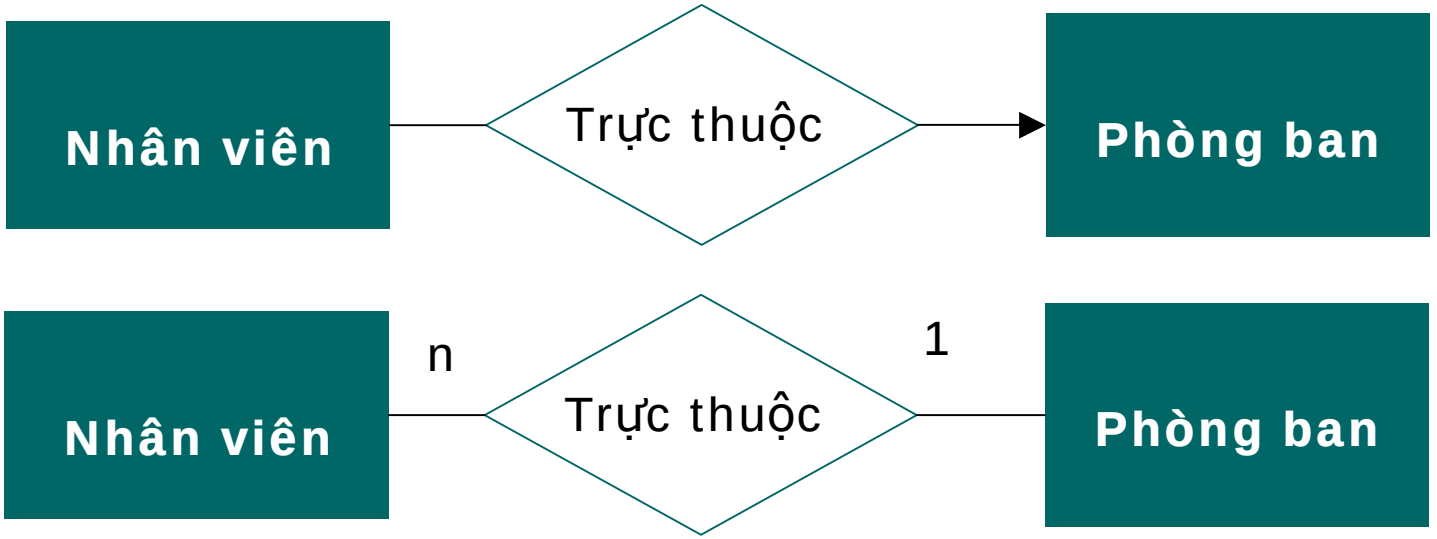
Các dạng thuộc tính (t)

- ***Thuộc tính phức hợp:*** Hợp thành của một số thành phần thông tin-thuộc tính của thực thể.
  - Phân rã thành thuộc tính đơn (không thể bị phân rã) tùy thuộc nhu cầu xử lý.
- ***Thuộc tính đa trị***: Nhận nhiều hơn một giá trị


Họ đệm
Nhân viên
Họ tên
Tên
CMTND
Kỹ năng

Biểu diễn tập liên kết mức sơ đồ

- Nhóm bao gồm các liên kết cùng kiểu được gọi là một ***tập liên kết (kiểu mối liên kết / mối liên kết)***
- Biểu diễn

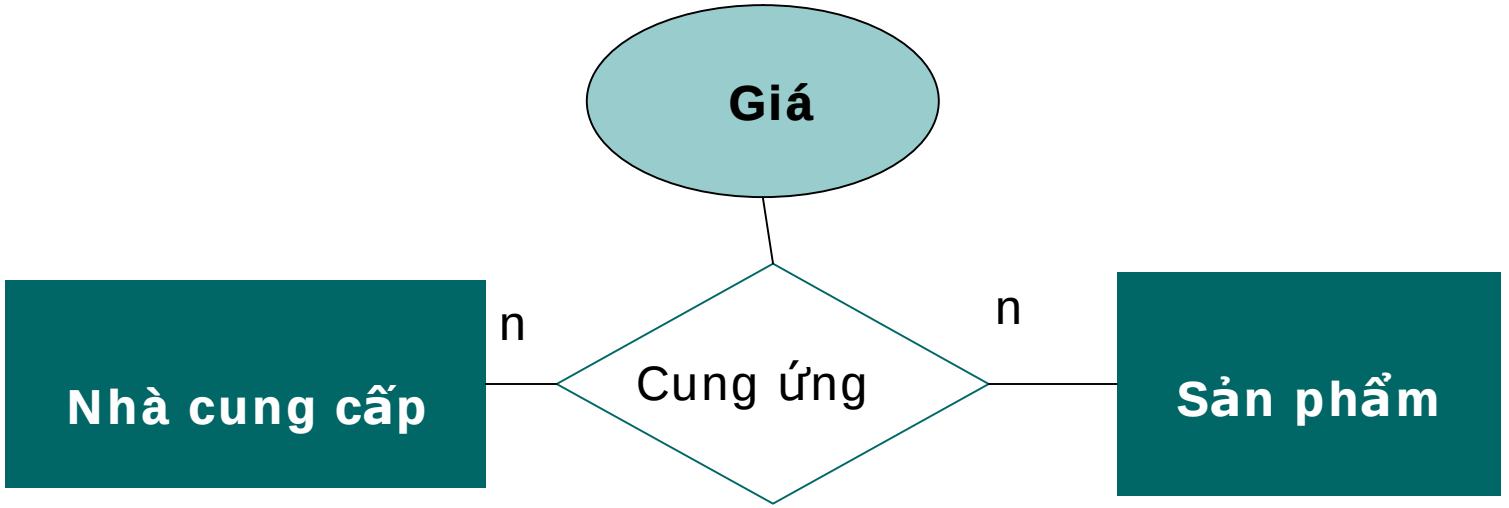


Các kiểu mối liên kết giữa hai tập thực thể (E1, E2)

- **một - một** (one-to-one): một thực thể a trong E1 liên kết với một thực thể b trong E2 và ngược lại
- **nhiều - nhiều** (many-to-many): một thực thể a trong E1 liên kết với nhiều thực thể b trong E2 và ngược lại.
- **một - nhiều** (one-to-many): một thực thể a trong E1 liên kết với nhiều thực thể b trong E2; một thực thể b trong E2 liên kết với một thực thể a trong E1.
- **là một:** Tập thực thể E2 là sự tổng quát hóa của tập thực thể E1, hay E1 là một kiểu đặc biệt của E2

Thuộc tính của mối liên kết

- Mối liên kết có thể có nhiều thuộc tính dùng để mô tả các đặc tính của sự liên kết giữa các thực thể



3.1.2. Sơ đồ thực thể liên kết

- Sơ đồ tóm tắt thông tin cần lưu trữ trong một csdl được mô tả bằng mô hình thực thể liên kết được gọi là sơ đồ thực thể liên kết hay sơ đồ ER.
- Sơ đồ ER của một cơ sở dữ liệu thuộc một ứng dụng được xây dựng trên các đặc tả của dữ liệu cho ứng dụng đó.

Ví dụ

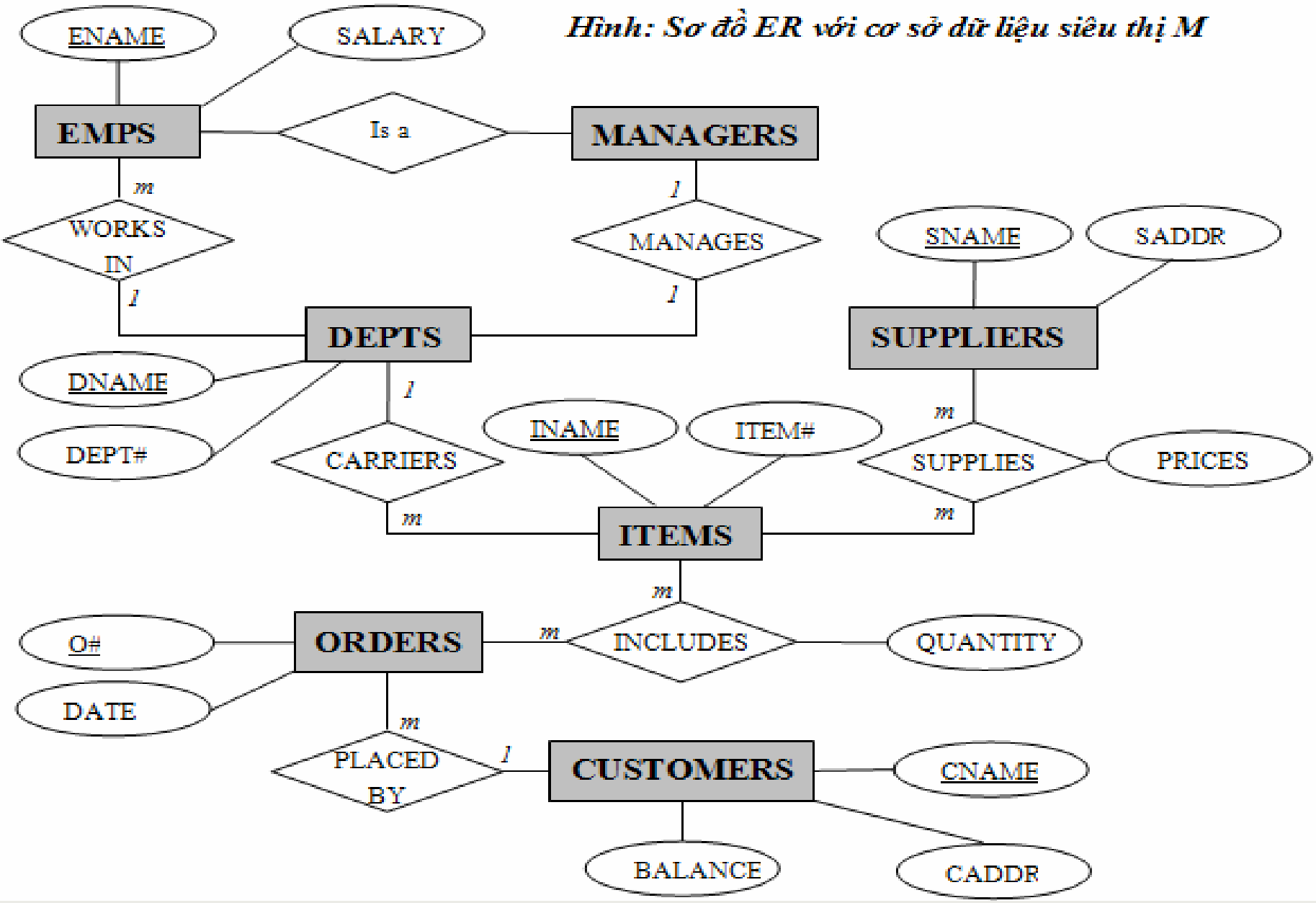
- Đặc tả dữ liệu cho ứng dụng quản lý của siêu thị M[5]:
  - Siêu thị được tổ chức thành các **phòng**, mỗi phòng có một **người quản lý** và một số **nhân viên**
  - Mỗi phòng có 1 người quản lý, và mỗi người quản lý cũng là một nhân viên trong siêu thị
  - Mỗi **mặt hàng** được bán bởi chỉ một phòng và mỗi phòng sẽ bán một số mặt hàng nhất định.

Đặc tả dữ liệu cho ứng dụng quản lý của siêu thị M[5] (t)

- Mỗi **khách hàng** sẽ đặt mua các mặt hàng từ siêu thị M thông qua các **hóa đơn.** Mỗi hóa đơn bao gồm danh sách các mặt hàng và số lượng tương ứng cho mỗi khách hàng cụ thể.
- Mỗi mặt hàng có thể do nhiều **nhà cung cấp** khác nhau cung cấp và mỗi nhà cung cấp lại cung cấp nhiều mặt hàng khác nhau Mỗi mặt hàng có một giá do nhà cung cấp đưa ra cho siêu thị.

Các bước xây dựng sơ đồ ER

- B1: Xác định các kiểu thực thể liên kết chính
- B2: Xác định các thuộc tính cho các kiểu thực thể này
- B3: Xác định các thuộc tính khóa và thuộc tính kết nối
- B4: Xác định các mối liên kết
- B5: Xây dựng lược đồ



Hình: Sơ đồ ER với cơ sở dữ liệu siêu thị M

3.2. Mô hình quan hệ (Relation Model)

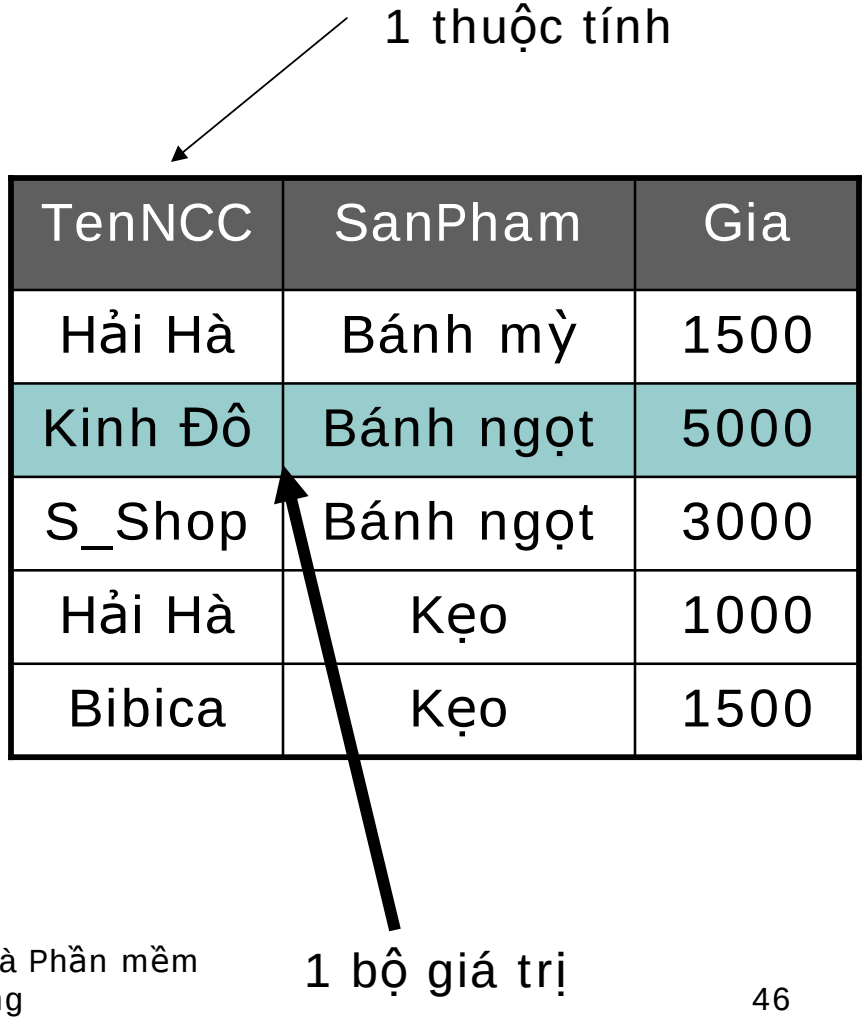
- Biểu diễn mọi dữ liệu dưới dạng các bảng, bảng được định dạng gồm các hàng và cột.
- Mô tả việc tổ chức dữ liệu trong hệ quản trị cơ sở dữ liệu.

3.2.1. Các khái niệm trong mô hình quan hệ

- **Quan hệ**
- **Thuộc tính**
- **Bộ giá trị**
- **Khóa**
- **Lược đồ quan hệ**
- **Thể hiện của quan hệ**

Quan hệ (Relation)

- Một quan hệ là một bảng
  - Mỗi hàng (bản ghi) biểu diễn một bộ giá trị của quan hệ. Số các bộ được gọi là lực lượng của quan hệ.
  - Mỗi cột (trường) biểu diễn một thuộc tính/ thành phần của các bộ. Số các thành phần được gọi là bậc của quan hệ.



Thuộc tính (Property)

- Ý nghĩa:
  - Thuộc tính là một tính chất riêng biệt của một đối tượng cần được lưu trữ trong CSDL để phục vụ cho việc khai thác dữ liệu về đối tượng.
- Ví dụ:
  - Các thuộc tính cho đối tượng nhà cung cấp hàng hóa là TenNCC, ĐiaChi, ...

Thuộc tính (Property)

- Thuộc tính được đặc trưng bởi: **Tên gọi, kiểu dữ liệu, miền giá trị**
  - **Tên gọi:** Thuộc tính được đặt tên một cách gợi nhớ nhưng không quá dài hoặc quá ngắn. VD: SNAME, TEN_NCC, TenNCC, TênNCC
  - **Kiểu:** Mỗi thuộc tính đều phải thuộc một kiểu kiểu dữ liệu nhất định. Ví dụ: TenNCC kiểu xâu kí tự Char, NgaySinh kiểu ngày tháng Date.
  - **Miền giá trị (MGT):** Là tập tất cả các giá trị mà thuộc tính có thể có. Ví dụ: SanPham có các giá trị thuộc tập {Bánh mỳ, Bánh ngọt, Kẹo}. Giá trị đặc biệt: giá trị rỗng/NULL.

Bộ giá trị (Tupe)

- Ý nghĩa
  - Một bộ là các thông tin của một đối tượng thuộc quan hệ.
- Đặc điểm
  - Hai bộ bất kỳ trong quan hệ là khác nhau ở ít nhất giá trị của một thuộc tính.
  - Một bộ không thể được xác định nhờ vị trí của nó trong quan hệ ➔ bộ nhận diện nhờ khóa

Khóa (Key)

- Khóa của quan hệ là một hay một tập hợp các thuộc tính mà giá trị của nó xác định duy nhất một bộ trong một quan hệ (khóa nội).
  - Khóa đơn là khóa chỉ có một thuộc tính.Khóa kép là khóa có hai thuộc tính trở lên
  - Khóa tối thiểu là khóa chứa tối thiểu các thuộc tính để có thể tạo thành 1 khóa cho quan hệ. Khóa chính (primary key) là một khóa tối thiểu tốt nhất cho quan hệ.Các khóa còn lại gọi là khóa dự bị
  - Một/ một tập thuộc tính của quan hệ mà giá trị của nó khớp với khóa chính của một quan hệ khác thì nó được gọi là khóa ngoại (foreign key) của quan hệ đó.

Khóa (Key)

TenNCC	SanPham	Gia
Hải Hà	Bánh mỳ	1000
Kinh Đô	Bánh ngọt	5000
S_Shop	Bánh ngọt	3000
Hải Hà	Kẹo	1500
Kinh Đô	Bánh quy	5000
Bibica	Kẹo	1500

- Khóa {TenNCC, SanPham}➔ {TenNCC, SanPham, Gia} cũng là khóa
- {TenNCC, SanPham}: Khóa nội, khóa tối thiểu, khóa chính.

TenNCC	SanPham	Gia
Hải Hà	Bánh mì	1000
Kinh Đô	Bánh ngọt	5000
S_Shop	Bánh ngọt	3000
Hải Hà	Kẹo	1500
Kinh Đô	Bánh quy	5000
Bibica	Kẹo	1500

Khóa ngoại (Khóa kép)

Khóa đơn

TenNCC	Diachi	SoluongNV
Hải Hà	Hà Nội	100
Kinh Đô	Hà Nội	500
S_Shop	Hồ Chí Minh	30
Bibica	Đà Nẵng	150

Lược đồ quan hệ

Quan hệ

TenNCC	SanPham	Gia
Hải Hà	Bánh mỳ	1500
Kinh Đô	Bánh ngọt	5000
S_Shop	Bánh ngọt	3000
Hải Hà	Kẹo	1000
Bibica	Kẹo	1500

➔Lược đồ:CUNG_UNG (TenNCC,SanPham, Gia)

- Sự trừu tượng hóa của quan hệ ở mức độ cấu trúc của một bảng hai chiều.
  - 1 quan hệ i có tương ứng 1 lược đồ quan hệ Ri
  - Lược đồ CSDL C = {Ri}

Lược đồ quan hệ

- Mô tả:
  - Lược đồ quan hệ được mô tả thông qua một ***tân từ***
  - *Ví dụ:*
    - *Lược đồ CUNG_UNG*(TenNCC, SanPham, Gia)
    - *Tân từ* : Mỗi sự cung ứng sẽ được thực hiện bởi một nhà cung cấp, tương ứng với một sản phẩm. Mỗi sản phẩm được cung ứng bởi một nhà cung cấp sẽ có một giá.

Thể hiện của quan hệ

- Thể hiện (tình trạng) của quan hệ R, ký hiệu là TR, là tập hợp các bộ giá trị của quan hệ R vào một thời điểm. Tại những thời điểm khác nhau thì quan hệ sẽ có những thể hiện khác nhau

3.2.2. Biến đổi sơ đồ ER sang lược đồ CSDL quan hệ

<u>Bước 1:</u> Biến đổi các tập thực thể

- **Một tập thực thể** E được biến **thành một quan hệ R** mà lược đồ của R bao gồm các thuộc tính của E. Mỗi bộ của R là một thực thể trong E
  - Ví dụ: tập thực thể CUSTOMERS ➔ CUSTOMERS(CNAME, CADDR, BALANCE)
- Nếu các thực thể trong E ***được xác định*** thông qua một liên kết với một tập thực thể F nào đó thì lược đồ R tương ứng có thuộc tính là các thuộc tính trong khóa chính của F.
  - Ví dụ: tập thực thể MANAGERS chỉ có một thuộc tính ENAME là khóa của EMPS.

Bước 2: Biến đổi liên kết

- Một liên kết giữa các tập thực thể E1, E2, .., Ek được biến thành một lược đồ R trong đó thuộc tính của R là tập tất cả các thuộc tính trong khóa của mỗi E1, E2, ..., Ek
  - Ví dụ: Ta có quan hệ WORKS IN (ENAME, DNAME) trong đó ENAME là nhân viên trong phòng DNAME, còn quan hệ MANAGES (ENAME, DNAME) thì ENAME là quản lý của phòng DNAME
- *Chú ý khi lấy các tập thuộc tính của R' cần:*
  - Đặt lại tên nếu có 2 thuộc tính của hai tập nào đó trùng tên
  - Xác định miền giá trị và không làm mất các thực thể.

Các quan hệ biến đổi từ tập thực thể cùng tên

- EMPS(ENAME, SALARY);
- MANAGERS(ENAME);
- DEPTS(DNAME, DEPT#);
- SUPPLIERS(SNAME, SADDR);
- ITEMS(INAME, ITEM#);
- ORDERS(O#, DATE);
- CUSTOMERS(CNAME, CADDR, BALANCE);

Các quan hệ biến đổi từ các liên kết

- WORKS_IN(ENAME, DNAME);
- MANAGES(ENAME, DNAME);
- CARRIES(INAME, DNAME);
- PLACED_BY(O#, CNAME);
- SUPLLIES(SNAME, INAME, PRICES);
- INCLUDES(O#, INAME, QUANTITY).

Bước 3: Tìm khóa cho các lược đồ

- Ngyên tắc
  - Nếu một lược đồ quan hệ được biến đổi từ một tập các thực thể, khóa của lược đồ quan hệ chính là khóa của tập thực thể.
  - Nếu một lược đồ quan hệ được biến đổi từ một liên kết nhiều – nhiều thì khóa của quan hệ là tất cả các thuộc tính (khoá) của quan hệ
  - Nếu một lược đồ quan hệ được biến đổi từ một liên kết một – một giữa hai tập thực thể E và F thì khóa của lược đồ quan hệ có thể là khoá của E hoặc là khoá của F.
  - Nếu một lược đồ quan hệ được biến đổi từ một liên kết nhiều – một từ E1, E2, …, Ek-1 đến Ek thì khóa thường là hợp các khóa của E1, E2, .., Ek-1.

Khóa của các quan hệ biến đổi từ tập thực thể cùng tên

- EMPS(**ENAME**, SALARY);
- MANAGERS(**ENAME**);
- DEPTS(**DNAME**, DEPT#);
- SUPPLIERS(**SNAME**, SADDR);
- ITEMS(**INAME**, ITEM#);
- ORDERS(**O#**, DATE);
- CUSTOMERS(**CNAME**, CADDR, BALANCE);

Khóa của các quan hệ biến đổi từ các liên kết

- WORKS_IN(ENAME, DNAME);
- MANAGES(ENAME, DNAME);
- CARRIES(INAME, DNAME);
- PLACED_BY(O#, CNAME);
- SUPLLIES(**SNAME**, **INAME**, PRICES);
- INCLUDES(**O#**, **INAME**, QUANTITY).

Bước 4: Giản ước lược đồ CSDL

- Nguyên tắc
  - Khi hai lược đồ quan hệ có khoá chung, có thể tổ hợp các thuộc tính của hai lược đồ này và thay thế hai lược đồ này bằng một lược đồ quan hệ mới với tập thuộc tính là hợp của hai tập thuộc tính của hai lược đồ quan hệ ban đầu.
    - Tiết kiệm không gian bộ nhớ
    - Truy vấn thuận tiện

Lược đồ CSDL sau khi giản lược

- EMPS(**ENAME**, SALARY, DNAME)
- DEPTS(**DNAME**, *DEPT#*, MGR)
- ITEMS(**INAME**, *ITEM#*, DNAME)
- CUSTOMERS(**CNAME**, CADDR, BALANCE)
- SUPPLIERS(**SNAME**, SADDR)
- ORDERS(**O#**, DATE, CNAME)
- SUPPLIES(**SNAME**, **INAME**, PRICE)
- INCLUDES(**O#**, **INAME**, QUANTITY)

- NHAN_VIEN (**TenNV**, Luong, TenPhong)
- PHONG_BAN(**TenPhong**, MaPhong, NguoiQuanLy)
- MAT_HANG(**TenHang**, MaHang, TenPhong)
- KHACH_HANG(**TenKH**, DiaChi, SoDuTK)
- NHA_CUNG_CAP(**TenNCC**, DiaChi)
- DON_DAT_HANG(**MaDDH**, NgayLap, TenKH)
- CUNG_UNG(**TenNCC**, **TenHang**, Gia)
- CHI_TIET_DON_HANG(**MaDDH**, **TenHang**, SoLuong)

Chỉ mục

- Database Management System (DBMS): Hệ quản trị CSDL.
- Data Definition Language (DDL): Ngôn ngữ định nghĩa dữ liệu
- Data Manipulation Language (DML): Ngông ngữ thao tác dữ liệu
- Data Control Language (DCL): Ngôn ngữ điều khiển dữ liệu
- Entity Relationship(ER): Mô hình dữ liệu thực thể liên kết
- Hierachical Model: Mô hình dữ liệu phân cấp
- Network Model: Mô hình lưới
- Relational Model: Mô hình quan hệ

Quản trị Cơ sở dữ liệu và Phần mềm ứng dụng

Bộ môn CNTT
Khoa Tin học Thương mại

Chương II: *Thiết kế CSDL quan hệ*

1. Giới thiệu chung
   - 1.1. Thiết kế CSDL QH và các cách tiếp cận
   - 1.2. Phụ thuộc hàm
2. Chuẩn hóa lược đồ quan hệ
   - 2.1. Các dạng chuẩn
   - 2.2. Tách lược đồ quan hệ theo chuẩn
3. Ràng buộc toàn vẹn trong CSDL quan hệ
   - 3.1. Khái niệm ràng buộc toàn vẹn
   - 3.2. Ràng buộc toàn vẹn trên thuộc tính
   - 3.3. Ràng buộc toàn vẹn trên quan hệ

1. Giới thiệu chung

1.1. Thiết kế CSDL QH và các cách tiếp cận

- Thiết kế cơ sở dữ liệu quan hệ ⇔ xây dựng lược đồ CSDL QH gồm một tập các lược đồ quan hệ thỏa mãn hai yêu cầu:
  - Lưu trữ thông tin không dư thừa
  - Tìm kiếm thông tin dễ dàng
- Ví dụ
  - Lược đồ quan hệ
    - CUNG_UNG(**MaNCC**, TenNCC, DiaChi, **SanPham**, Gia)

Dư thừa dữ liệu

- Quan hệ CUNG_UNG_0

MaNCC	TenNCC	DiaChi	SanPham	Gia
1	Hải Hà	Hà Nội	Kẹo mềm	100
1	Hải Hà	Hà Nội	Kẹo cứng	150
1	Hải Hà	Hà Nội	Bánh	200
2	Kinh đô	Hồ Chí Minh	Kẹo	120
2	Kinh đô	Hồ Chí Minh	Bánh	150

- Một nhà cung cấp cung cấp nhiều mặt hàng.
- Lặp các thông tin về nhà cung cấp ứng với mỗi một mặt hàng khác nhau của cùng nhà cung cấp đó.

Không nhất quán

- Quan hệ CUNG_UNG_0

MaNCC	TenNCC	DiaChi	SanPham	Gia
1	Hải Hà	Đà Nẵng	Kẹo mềm	100
1	Hải Hà	Hà Nội	Kẹo cứng	150
1	Hải Hà	Hà Nội	Bánh	200
2	Kinh đô	Hồ Chí Minh	Kẹo	120
2	Kinh đô	Hồ Chí Minh	Bánh	150

- Dị thường khi cập nhật thông tin về nhà cung cấp như thay đổi địa chỉ.

Dị thường khi thêm bộ

- Quan hệ CUNG UNG 0

MaNCC	TenNCC	DiaChi	SanPham	Gia
1	Hải Hà	Hà nội	Kẹo mềm	100
1	Hải Hà	Hà Nội	Kẹo cứng	150
1	Hải Hà	Hà Nội	Bánh	200
2	Kinh đô	Hồ Chí Minh	Kẹo	120
2	Kinh đô	Hồ Chí Minh	Bánh	150
3	Bibica	Đà nẵng	NULL	NULL

- Dị thường khi thêm mới thông tin về nhà cung cấp nhưng nhà cung cấp chưa cung cấp mặt hàng nào.

Dị thường khi xóa bộ

- Quan hệ CUNG_UNG_0

MaNCC	TenNCC	DiaChi	SanPham	Gia
1	Hải Hà	Hà nội	Kẹo mềm	100
1	Hải Hà	Hà Nội	Kẹo cứng	150
1	Hải Hà	Hà Nội	Bánh	200
2	Kinh đô	Hồ Chí Minh	Kẹo	120

- Tồn tại nhà cung cấp chỉ cung cấp một mặt hàng.
- Dị thường khi xóa thông tin về sự cung cấp ➔ xóa luôn thông tin về nhà cung cấp.

Tìm kiếm thông tin

MaNCC	TenNCC	DiaChi
1	Hải Hà	Hà Nội
2	Kinh đô	Hồ Chí Minh

CUNG_UNG_11

MaNCC	SanPham	Gia
1	Kẹo mềm	100
1	Kẹo cứng	150
1	Bánh	200
2	Kẹo	120
2	Bánh	200

CUNG_UNG_12

- Quan hệ CUNG_UNG_0 tách thành 2 quan hệ CUNG_UNG_11 và CUNG_UNG_12
  - Lưu trữ thông tin không dư thừa ???
  - Tìm kiếm thông tin dễ dàng ???

Các cách tiếp cận

- **Từ trên xuống *(Topdown)*:**
  - Xây dựng sơ đồ thực thể liên kết ER từ các đặc tả
  - Chuyển đổi sơ đồ ER thành lược đồ CSDL quan hệ.
  - Chuẩn hóa lược đồ CSDL quan hệ (nếu cần)
- **Từ dưới lên *(Bottom Up)*:**
  - Xây dựng lược đồ quan hệ ban đầu từ các đặc tả.
  - Chuẩn hóa lược đồ quan hệ.

1.2. Phụ thuộc hàm

a. Khái niệm

- Cho quan hệ R, thuộc tính B của quan hệ R được gọi là phụ thuộc hàm vào thuộc tính A của quan hệ R nếu với mỗi giá trị của A xác định duy nhất một giá trị của B. A được gọi là xác định hàm của B.
- Ký hiệu: A➔B

a. Khái niệm (t)

- *Tập các phụ thuộc hàm F của 1 lược đồ quan hệ* R là một tập gồm các phụ thuộc hàm xác định trên R.
  - Ví dụ: Tập phụ thuộc hàm F={A➔B, B➔C} của R(A,B,C)
- *Trong quan hệ R, ký hiệu A, B, C dành cho các thuộc tính đơn, X, Y, Z dành cho tập các thuộc tính.*

Ví dụ

- Tập tất cả các thuộc tính của quan hệ phải phụ thuộc hàm vào khóa.
  - MaNCC➔ TenNCC
  - MaNCC➔ SoNV
  - MaNCC➔ DiaChi
  - MaNCC: Khóa

MaNCC	TenNCC	SoNV	DiaChi
S1	Hải Hà	20	Hà Nội
S2	Kinh Đô	10	Hà Nội
S3	Bibica	30	HCM

- F={ MaNCC➔ TenNCC, MaNCC➔ SoNV, MaNCC➔ DiaChi}

Ví dụ

- Một tập thuộc tính là xác định hàm của các thuộc tính khác thì chưa chắc là một khóa.
  - TenNCC➔ DiaChi
  - TenNCC không phải là khóa

MaNCC	TenNCC	SoNV	DiaChi
S1	Hải Hà	20	Hà Nội
S2	Kinh Đô	10	Hà Nội
S3	Bibica	30	HCM
S4	Hải Hà	10	Hà Nội

b. Hệ tiên đề Amstrong

- Giả thiết
  - Lược đồ quan hệ R.
  - X,Y,Z: tập các thuộc tính thuộc R.
  - XY=X∪Y
- Hệ 3 tiên đề với các phụ thuộc hàm:
  - Phản xạ:XY➔X; XY➔Y
  - Tăng trưởng: X➔Y thì XZ➔YZ
  - Bắc cầu:X➔Y, Y➔Z thì X➔Z

Luật suy ra từ hệ tiên đề

- Luật hợp
  - Nếu X➔Y, X➔Z thì X➔YZ
- Luật tựa bắc cầu
  - Nếu X➔Y, WY➔Z thì XW➔Z
- Luật tách
  - Nếu X➔Y, Z thuộc Y thì X➔Z

c. Phụ thuộc hàm đầy đủ và phụ thuộc bắc cầu

- **Phụ thuộc hàm đầy đủ**
  - Y phụ thuộc hàm đầy đủ vào X nếu Y phụ thuộc hàm vào X nhưng không phụ thuộc hàm vào bất kỳ một tập con thực sự nào của X.
  - Ví dụ
    - Lược đồ R(A, B, C, D)
    - F={AB➔C; AB➔D; B➔D}
    - C phụ thuộc hàm đầy đủ vào {A,B}
    - D không phụ thuộc hàm đầy đủ vào {A,B}

Phụ thuộc bắc cầu

- Phụ thuộc hàm $X \rightarrow A$, A được gọi là **phụ thuộc bắc cầu** vào X nếu tồn tại Y để cho $X \rightarrow Y$, $Y \rightarrow A$, $Y \not\rightarrow X$ và $A \notin XY$
- Ví dụ:
  - F = {A➔B, B➔C}
  - A➔C: C phụ thuộc bắc cầu vào A

d. Bao đóng và phủ của tập các phụ thuộc hàm

- *Cho tập các phụ thuộc hàm F xác định trên R.*
  - **Bao đóng F^+** *của tập các phụ thuộc hàm F* là tập tất cả các phụ thuộc hàm được *suy diễn logic* từ F.
  - **Phủ G** của tập các phụ thuộc hàm F (G≈F) là tập các phụ thuộc hàm xác định trên R sao cho $G^+ = F^+$.

X^+ ?

- Bao đóng X^+ của thuộc tính X đối với tập phụ thuộc hàm F là tất cả các thuộc tính A mà phụ thuộc hàm X➔A có thể được suy diễn logic từ F nhờ hệ tiên đề Amstrong.
- Một phụ thuộc hàm X➔Y thuộc F^+ nếu Y thuộc X^+: Kiểm tra X➔Y có thuộc F^+

Ý nghĩa của phụ thuộc hàm

- Chỉ ra các phụ thuộc dữ liệu/ràng buộc có thể xảy ra giữa tập thuộc tính của một lược đồ quan hệ.
  - ➔ Giúp xác định khóa tối thiểu, khóa chính của quan hệ.
  - ➔ Giúp chuẩn hóa lược đồ quan hệ

2. Chuẩn hóa lược đồ quan hệ

- Khái niệm
  - Là quá trình phân tách các lược đồ quan hệ thành các lược đồ quan hệ nhỏ hơn theo một số **tiêu chuẩn** nhằm loại bỏ việc lưu trữ dư thừa dữ liệu.
  - Phép tách thành các lược đồ quan hệ đơn giản hơn, nhỏ hơn phải đảm bảo **không làm mất mát thông tin**.

2.1. Các dạng chuẩn

- Dạng chuẩn 1
- Dạng chuẩn 2
- Dạng chuẩn 3
- Dạng chuẩn Boye-Codd
- *Chuẩn 4 và các dạng chuẩn khác*

a. Dạng chuẩn 1(1NF)

- **Định nghĩa**
  - Một lược đồ quan hệ R ở dạng chuẩn 1 nếu và chỉ nếu toàn bộ các miền giá trị của các thuộc tính trong R đều chỉ chứa các giá trị nguyên tố.
  - Một quan hệ xác định trên lược đồ quan hệ ở dạng chuẩn 1 được gọi là quan hệ ở dạng chuẩn 1.

Dạng chuẩn 1 (t)

- Một quan hệ thuộc dạng chuẩn 1 là một quan hệ trong đó mỗi miền giá trị của một thuộc tính chỉ chứa những giá trị nguyên tố (không phân chia được nữa).
- Một quan hệ thuộc dạng chuẩn 1 nếu mỗi một ô trong bảng chỉ chứa duy nhất một giá trị

Ví dụ

- Quan hệ CUNG_UNG_0 **chưa thuộc dạng chuẩn 1**

MaNCC	TenNCC	DiaChi	SanPham	Gia
1	Hải Hà	Hà Nội	Kẹo mềm Kẹo cứng Bánh	100 150 200
2	Kinh Đô	Hồ Chí Minh	Kẹo Bánh	120 200

Ví dụ(t)

- Quan hệ CUNG_UNG_1 **đã thuộc dạng chuẩn 1**

MaNCC	TenNCC	DiaChi	SanPham	Gia
1	Hải Hà	Hà Nội	Kẹo mềm	100
1	Hải Hà	Hà Nội	Kẹo cứng	150
1	Hải Hà	Hà Nội	Bánh	200
2	Kinh đô	Hồ Chí Minh	Kẹo	120
2	Kinh đô	Hồ Chí Minh	Bánh	200

b. Dạng chuẩn 2 (2NF)

- **Định nghĩa**
  - Một lược đồ quan hệ R được gọi là ở dạng chuẩn 2 nếu nó đã ở dạng chuẩn 1 và mọi thuộc tính không khóa đều phụ thuộc hàm đầy đủ vào khóa chính.
  - Một quan hệ xác định trên lược đồ quan hệ ở dạng chuẩn 2 được nói là quan hệ ở dạng chuẩn 2.

Ví dụ

- Quan hệ CUNG_UNG_1 **chưa thuộc dạng chuẩn 2**.

MaNCC	TenNCC	DiaChi	SanPham	Gia
1	Hải Hà	Hà Nội	Kẹo mềm	100
1	Hải Hà	Hà Nội	Kẹo cứng	150
1	Hải Hà	Hà Nội	Bánh	200
2	Kinh đô	Hồ Chí Minh	Kẹo	120
2	Kinh đô	Hồ Chí Minh	Bánh	200

Lặp →

Lặp →

Ví dụ(t)

- R(M, T, D, S, G) = MTDSG= Lược đồ của quan hệ CUNG_UNG_1
- Phụ thuộc hàm
  - M➔ TD, MS➔G
- MS: Khóa tối thiểu
  - M, S: Thuộc tính khóa
  - T, D, G: Thuộc tính không khóa
- **T, D không phụ thuộc hàm đầy đủ vào MS**

➔ Lược đồ quan hệ CUNG_UNG_1 không thuộc dạng chuẩn 2.

Ví dụ

MaNCC	TenNCC	DiaChi
1	Hải Hà	Hà Nội
2	Kinh đô	Hồ Chí Minh

CUNG_UNG_11

MaNCC	SanPham	Gia
1	Kẹo mềm	100
1	Kẹo cứng	150
1	Bánh	200
2	Kẹo	120
2	Bánh	200

CUNG_UNG_12

- Ví dụ 2 thành quan hệ CUNG_UNG_11 và CUNG_UNG_12 tách từ quan hệ CUNG_UNG_1 **đã thuộc dạng chuẩn 2**.
  - TenNCC, DiaChi phụ thuộc hàm đầy đủ vào MaNCC
  - Gia phụ thuộc hàm đầy đủ vào {MaNCC, SanPham}

c. Dạng chuẩn 3

- **Định nghĩa**
  - Một lược đồ quan hệ R được gọi là ở dạng chuẩn 3 nếu nó đã ở dạng chuẩn 2 và mọi thuộc tính không khóa của R đều chỉ phụ thuộc hàm duy nhất vào khóa chính.
  - Một quan hệ xác định trên lược đồ quan hệ ở dạng chuẩn ba được nói là quan hệ ở dạng chuẩn 3.

Ví dụ

MaNCC	TenNCC	DiaChi
1	Hải Hà	Hà Nội
2	Kinh đô	Hồ Chí Minh

CUNG_UNG_11

MaNCC	SanPham	Gia
1	Kẹo mềm	100
1	Kẹo cứng	150
1	Bánh	200
2	Kẹo	120
2	Bánh	200

CUNG_UNG_12

- Ví dụ 2 quan hệ CUNG_UNG_11 và CUNG_UNG_12 **đã thuộc dạng chuẩn 3**.
- MTDSG tách thành 1 lược đồ con MTD và MSG:
  - MTD: T, D phụ thuộc chỉ vào khóa M
  - MSG: G phụ thuộc hàm chỉ vào MS

Ví dụ

- Ví dụ
  - Lược đồ quan hệ:
    - R(Store, Item, Departement, Manager) ⇔ R(S,I,D,M)
  - Tập các phụ thuộc hàm:
    - F={SI➔D, SD➔M}
  - Khóa tối thiểu: SI
  - Lược đồ thuộc chuẩn 2: D, M phụ thuộc hàm đầy đủ vào SI
  - Lược đồ **không thuộc chuẩn 3**: M phụ thuộc bắc cầu vào SI
    - SI➔D; SI➔SD;SD➔M;SI➔M(*)

d. Dạng chuẩn Boye-Codd (BCNF)

- Chuẩn 3 không đáp ứng được những lược đồ quan hệ
  - Có nhiều hơn một khóa tối thiểu
  - Các khóa tối thiểu là khóa kép
  - Các khóa tối thiểu giao nhau
- **Định nghĩa**
  - Một lược đồ quan hệ R thuộc dạng chuẩn Boye-Codd khi và chỉ khi mọi xác định hàm đều là một khóa.

Ví dụ

- Lược đồ quan hệ: R(CITY, STREET, ZIP)
- Phụ thuộc hàm:
  - CITY, STREET ➔ ZIP, ZIP ➔ CITY
- Khóa tối thiểu:
  - {CITY, STREET}, {STREET, ZIP}
- Thuộc tính khóa:
  - CITY, STREET, ZIP ➔ không có thuộc tính không khóa, thuộc dạng chuẩn 3.
- *Xác định hàm* ZIP không phải là một khóa **➔ lược đồ không thuộc chuẩn Boye-Codd**

Ví dụ

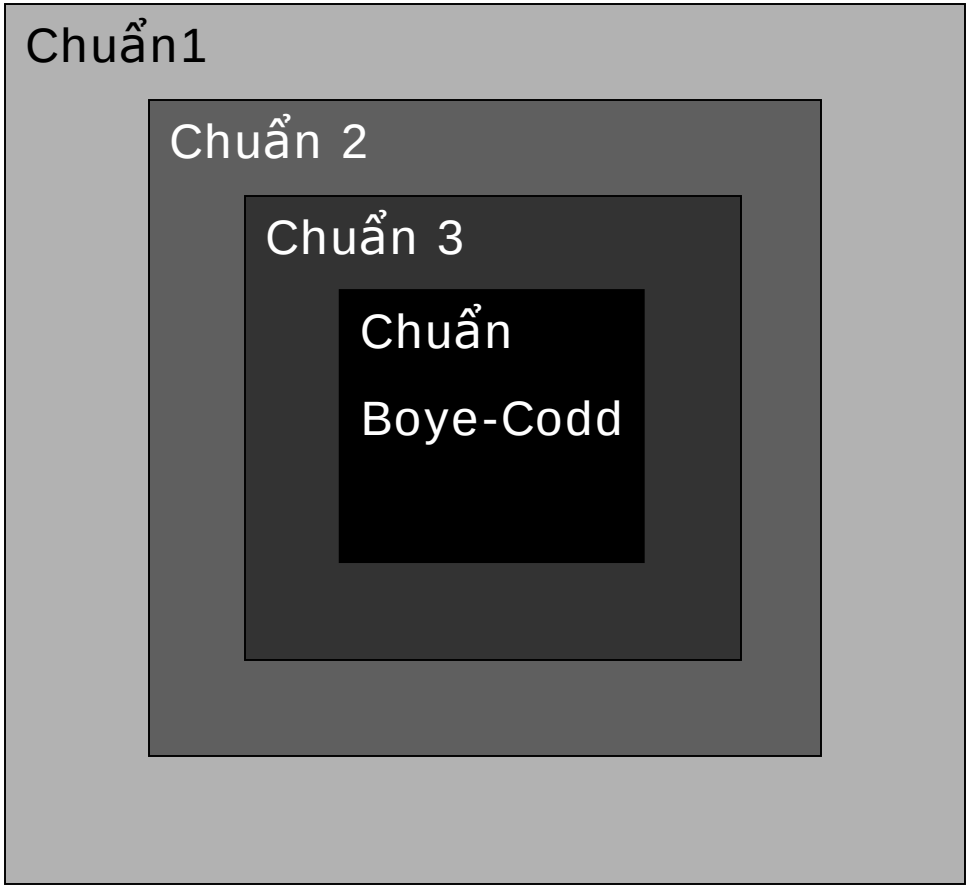
- 2 lược đồ con R1(CITY, ZIP) và R2(STREET, ZIP) **thuộc dạng chuẩn Boye-Codd**.

CITY	STREET	ZIP
Hà Nội	Láng	0423
Hà Nội	Phạm Hùng	0425
HCM	Võ Thị Sáu	0811

CITY	ZIP
Hà Nội	0423
Hà Nội	0425
HCM	0811

STREET	ZIP
Láng	0423
Phạm Hùng	0425
Võ Thị Sáu	0811

Quan hệ giữa các dạng chuẩn



2.2. Tách lược đồ quan hệ theo chuẩn

2.2.1. Tách bảo toàn thông tin và tách bảo toàn tập phụ thuộc hàm

2.2.2. Các thuật toán tách lược đồ quan hệ

2.2.3. Tách lược đồ quan hệ theo từng bước

2.2.1.Tách bảo toàn thông tin và tách bảo toàn tập phụ thuộc hàm

a. Phép tách bảo toàn thông tin

- Khái niệm
  - Phép tách bảo toàn thông tin là phép tách lược đồ quan hệ sao cho khi **kết nối tự nhiên** các quan hệ xác định trên các lược đồ con, kết quả cho lại quan hệ ban đầu.

Phép kết nối tự nhiên

- Phép ghép các cặp bộ của hai quan hệ trên các thuộc tính bằng nhau của hai quan hệ, một trong hai thuộc tính của phép so sánh "=" được loại bỏ sau khi thực hiện phép ghép.

- R(A, B, C)

a1	b1	1
a2	b2	2
a3	b3	3

- S(D, E)

1	e1
2	e2
3	e3

- R kết nối tự nhiên với s với C=D

a1	b1	1	e1
a2	b2	2	e2
a3	b3	3	e3

Ví dụ

- Lược đồ quan hệ MTDSG = (MaNCC, TenNCC, DiaChi, SanPham, Gia) ⇔(M, T, D, S, G);
- Tập phụ thuộc hàm F={M➔TD, MS➔G}
- Quan hệ xác định trên lược đồ MTDSG:

MaNCC	TenNCC	DiaChi	SanPham	Gia
1	Hải Hà	Hà Nội	Kẹo mềm	10
1	Hải Hà	Hà Nội	Kẹo cứng	15
1	Hải Hà	Hà Nội	Bánh ngọt	40
1	Hải Hà	Hà Nội	Bánh mỳ	16
2	Kinh Đô	Hồ Chí Minh	Kẹo	12
2	Kinh Đô	Hồ Chí Minh	Bánh quy	45
3	Kinh Đô	Đà Nẵng	Bánh mỳ	10
3	Kinh Đô	Đà Nẵng	Bánh quy	30

Ví dụ(t)

MaNCC	TenNCC	DiaChi
1	Hải Hà	Hà Nội
2	Kinh Đô	Hồ Chí Minh
3	Kinh Đô	Đà Nẵng

MaNCC	SanPham	Gia
1	Kẹo mềm	10
1	Kẹo cứng	15
1	Bánh ngọt	40
1	Bánh mỳ	16
2	Kẹo	12
2	Bánh quy	45
3	Bánh mỳ	10
3	Bánh quy	30

- Phép tách bảo toàn thông tin
  - Lược đồ con 1: MTD=(M, T, D)
  - Lược đồ con 2: MSG=(M, S, G)

Ví dụ(t)

MaNCC	TenNCC	DiaChi
1	Hải Hà	Hà Nội
2	Kinh Đô	Hồ Chí Minh
3	Kinh Đô	Đà Nẵng

TenNCC	SanPham	Gia
Hải Hà	Kẹo mềm	10
Hải Hà	Kẹo cứng	15
Hải Hà	Bánh ngọt	40
Hải Hà	Bánh mỳ	16
Kinh Đô	Kẹo	12
Kinh Đô	Bánh quy	45
Kinh Đô	Bánh mỳ	10
Kinh Đô	Bánh quy	30

- Phép tách mất mát thông tin
  - Lược đồ con 1: MTD=(M, T, D)
  - Lược đồ con 2: MSG=(M, S, G)

Thuật toán kiểm tra

- *Kiểm tra phép tách không mất mát thông tin của một lược đồ quan hệ thành nhiều lược đồ quan hệ con.*
  - *Vào*
    - Lược đồ quan hệ: $R=(A_1, A_2,... A_n)$
    - Tập phụ thuộc hàm F trên R
    - Phép tách $\rho=(R_1, R_2,... R_k)$
  - *Ra*
    - Một khẳng định rằng phép tách ρ có mất mát thông tin hay không?

Thuật toán kiểm tra (t)

- *Phương pháp*
  - Bước 1: Xây dựng một bảng k hàng, n cột
    - Hàng i tương ứng với R_i
    - Cột j tương ứng với thuộc tính A_j
    - Giá trị tại hàng i, cột j
      - a_j: Nếu A_j thuộc R_i
      - b_{ij}: Nếu A_j không thuộc R_i

Ví dụ (bước 1)

- Lược đồ quan hệ MTDSG
- Tập phụ thuộc hàm F={M➔TD, MS➔G}
- Tách thành hai lược đồ MTD, MSG

	M	T	D	S	G
R_1= MTD	a1	a2	a3	b14	b15
R_2= MSG	a1	b22	b23	a4	a5

Phương pháp

- Bước 2: Lặp
  - Áp dụng các phụ thuộc hàm cho bảng vừa được xây dựng: X➔Y: Nếu tồn tại hai hàng có cùng giá trị trên X thì làm bằng nhau các giá trị trên Y.
    - Nếu có giá trị một hàng thuộc Y là a_j thì các giá trị khác thuộc Y gán bằng a_j.
    - Nếu không gán bằng một trong các giá trị bij.
- Dừng khi các giá trị trong bảng không thể thay đổi được nữa

Ví dụ (bước 2)

	M	T	D	S	G
R_1 = MTD	**a1**	**a2**	**a3**	b14	b15
R_2 = MSG	**a1**	**b22**	**b23**	a4	a5

- Áp dụng phụ thuộc hàm M➔TD
  - Thay $b_{22} = a_2$
  - Thay $b_{23} = a_3$

Phương pháp

- Bước 3: Kiểm tra
  - Nếu trong bảng có một hàng gồm toàn ký hiệu a_1, a_2, a_3,... a_n thì phép tách là không mất mát thông tin.
  - Ngược lại phép tách là mất mát thông tin.

Ví dụ

	M	T	D	S	G
R_1 = MTD	a1	a2	a3	b14	b15
R_2 = MSG	**a1**	**a2**	**a3**	**a4**	**a5**

- Phép tách trên là không mất mát thông tin.

Định lý kiểm tra

- *Kiểm tra phép tách không mất mát thông tin của một lược đồ quan hệ thành hai lược đồ quan hệ con.*
  - Cho
    - lược đồ quan hệ R
    - Tập phụ thuộc hàm F trên R
    - $\rho = (R_1, R_2)$
  - Phép tách là không mất mát thông tin nếu $R_1 \cap R_2 \rightarrow$ R1\R2 hoặc $R1 \cap R2 \rightarrow$ R2\R1.

Ví dụ

- Lược đồ MTDSG tách thành hai lược đồ con MTD và MSG:
  - MTD∩MSG = M
  - MTD\MSG = TD
  - M➔TD thuộc F^+: Phép tách là bảo toàn thông tin.
- Lược đồ MTDSG tách thành hai lược đồ con MTD và TSG
  - MTD TSG = T
  - MTD∩SG = MD
  - TSG\MTD = SG
  - T➔MD và T➔SG không thuộc F^+: Phép tách là mất mát thông tin.

b. Phép tách bảo toàn tập phụ thuộc hàm

- Khái niệm
  - Phép tách lược đồ quan hệ R thành các lược đồ quan hệ con R_i là bảo toàn các tập phụ thuộc hàm nếu hợp của các phụ thuộc hàm là hình chiếu của F trên R_i suy diễn logic được tất cả các phụ thuộc hàm trong F.
  - Hình chiếu của F lên R_i là các phụ thuộc hàm X➔Y thỏa mãn:
    - X, Y thuộc R_i
    - X➔Y thuộc F^+

Ví dụ

- Cho
  - Lược đồ R = ACBCD
  - Tập phụ thuộc hàm F = { A➔B, C➔D}
  - Phép tách ρ = (R_1,R_2): R_1= AB, R_2 = CD
- Phép tách bảo toàn tập phụ thuộc hàm
  - Hình chiếu của F lên R_1: A➔B, AB➔A, AB➔B ...
  - Hình chiếu của F lên R_2: C➔D, CD➔C, CD➔D ...
  - Các phụ thuộc hàm trong F có thể được suy diễn từ các hình chiếu này
- Phép tách không bảo toàn thông tin
  - AB ∩ CD = Φ
  - AB\CD = AB
  - CD\AB = CD

Ví dụ

- Cho
  - Lược đồ R = CSZ
  - Tập phụ thuộc hàm F={CS➔Z, Z➔C}
  - Phép tách ρ = (R_1,R_2): R_1= CZ, R_2 = SZ
- Phép tách không bảo toàn tập phụ thuộc hàm
  - Hình chiếu của F lên R_1: Z➔C, CZ➔C, CZ➔Z,...
  - Tập phụ thuộc hàm trong R_2: SZ➔S, SZ➔Z,...
  - Phụ thuộc hàm CS➔Z không thể được suy ra từ các hình chiếu này.
- Phép tách bảo toàn thông tin
  - CZ∩SZ ➔ CZ\SZ hay Z➔C

Ví dụ

- Cho
  - Lược đồ R = MTDSG
  - Tập phụ thuộc hàm F={M➔TD, MS➔G}
  - Phép tách ρ = (R_1 ,R_2): R_1= MTD, R_2 = MSG
- Phép tách bảo toàn tập phụ thuộc hàm
  - Hình chiếu của F lên R_1: M➔TD, …
  - Hình chiếu của F lên R_2: MS➔G,…
  - Các phụ thuộc hàm trong F có thể được suy diễn logic từ các hình chiếu này.
- Phép tách bảo toàn thông tin
  - Đã chứng minh

2.2.2. Các thuật toán tách lược đồ quan hệ

a. *Thuật toán tìm một khóa tối thiểu của lược đồ quan hệ dựa vào tập phụ thuộc hàm*

b. *Thuật toán tách không mất mát thông tin và bảo toàn tập phụ thuộc hàm về dạng chuẩn 3*

c. *Thuật toán tách không mất mát thông tin về dạng chuẩn Boye-Codd*

a. *Thuật toán tìm một khóa tối thiểu của lược đồ quan hệ dựa vào tập phụ thuộc hàm*

- Khóa của lược đồ quan hệ
  - Giá trị của tập thuộc tính khóa trên mỗi bộ của quan hệ là duy nhất
  - Mọi thuộc tính của quan hệ phải phụ thuộc hàm vào khóa.
- Thuật toán tìm một khóa tối thiểu
  - Loại bỏ dần từng thuộc tính thuộc khóa của quan hệ cho tới khi tập thuộc tính nhỏ nhất còn lại vẫn thỏa mãn là một khóa➔ khóa tối thiểu của quan hệ.

b. *Thuật toán tách không mất mát thông tin và bảo toàn tập phụ thuộc hàm về dạng chuẩn 3*

- *Vào:*
  - Lược đồ quan hệ R
  - Tập phụ thuộc hàm F (phủ tối thiểu)
- *Ra:*
  - Tập sơ đồ con, trong đó mỗi sơ đồ con
    - Thuộc dạng chuẩn 3
    - Phụ thuộc hàm là hình chiếu của F lên nó

b. *Thuật toán tách không mất mát thông tin và bảo toàn tập phụ thuộc hàm về dạng chuẩn 3 (t)*

- *Phương pháp:*
  - Nếu tồn tại một thuộc tính thuộc R không có mặt ở vế trái hay vế phải của bất kỳ phụ thuộc hàm nào thì tách thuộc tính này ra khỏi R.
  - Nếu tồn một tại phụ thuộc hàm liên quan tới mọi thuộc tính của R thì kết quả là R.
  - Nếu các phụ thuộc hàm dạng:
    - X➔A : lược đồ con ở dạng XA.
    - X➔A_1, …X➔A_n: lược đồ con ở dạng X$A_{1…}A_n$

Ví dụ

- Cho lược đồ R = ABCDEG
- Tập phụ thuộc hàm F = {AB➔C, C➔A, BC➔D, ACD➔B, D➔EG, BE➔C, CG➔BD, CE➔AG}
- Tập phụ thuộc hàm tối thiểu F' = {AB➔C, C➔A, BC➔D, D➔E, D➔G, BE➔C, CG➔B, CE➔G}
- Phép tách ρ = (ABC, CA, BCD, DEG, BEC, CGB, CEG) = (ABC, BCD, DEG, BEC, CGB, CEG) gồm các lược đồ con đã ở dạng chuẩn 3 bảo toàn tập phụ thuộc hàm và bảo toàn thông tin.

c. *Thuật toán tách không mất mát thông tin về dạng chuẩn Boye-Codd*

- *Vào :*
  - Lược đồ quan hệ R
  - Tập phụ thuộc hàm F
- *Ra:*
  - Tập sơ đồ con, trong đó mỗi sơ đồ con
    - Thuộc dạng chuẩn Boye-Codd
    - Phụ thuộc hàm là hình chiếu của F lên nó

c. *Thuật toán tách không mất mát thông tin về dạng chuẩn Boye-Codd (t)*

- *Phương pháp:*
  - ρ = (R)
  - Lặp
    - S là một sơ đồ quan hệ trong ρ không ở dạng chuẩn Boye-Codd. Xét một phụ thuộc hàm X➔A của S. Nếu X không chứa khóa của S, A không thuộc X thì S được tách thành:
      - S_1 = A U{X}
      - S_2 = S\{A}
  - Dừng cho tới khi mọi sơ đồ con trong ρ đã thuộc dạng chuẩn Boye-Codd

2.2.3. Tách lược đồ quan hệ theo từng bước

- Quy tắc
  - *Quy tắc 1:* Loại bỏ các thuộc tính phụ thuộc chỉ một phần vào khóa chính.

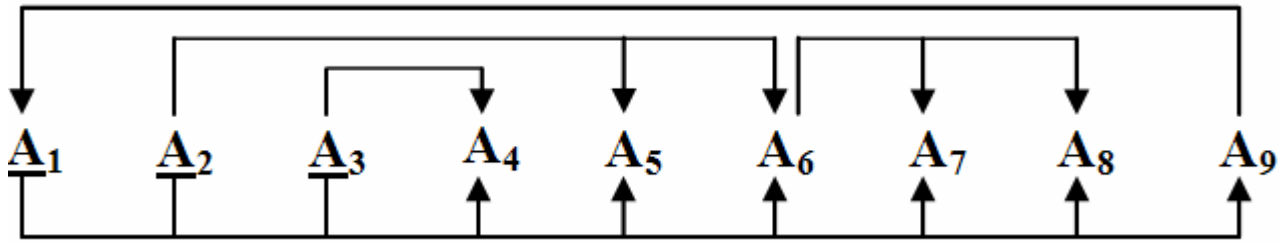
  ➔ Chuẩn hóa về 2NF
  - *Quy tắc 2:* Loại bỏ các thuộc tính không khóa không phụ thuộc vào khóa chính.

  ➔Chuẩn hóa về 3NF
  - Quy tắc 3: Loại bỏ các thuộc tính là giao của các khóa tối thiểu.

  ➔Chuẩn hóa về BCNF

Ví dụ

- Cho quan hệ R(A_1, A_2, A_3, A_4, A_5, A_6, A_7, A_8, A_9) trong đó $A_1A_2A_3$ là khóa với sơ đồ phụ thuộc hàm:



- Quan hệ R ở dạng chuẩn nào? Tại sao? Tách R thành các quan hệ ở dạng chuẩn BCNF.

3. Ràng buộc toàn vẹn trong CSDL quan hệ (Integrity Constraint)

3.1. Khái niệm ràng buộc toàn vẹn

- *Ràng buộc toàn vẹn là điều kiện bất biến không được vi phạm trong một CSDL.*
  - Các điều kiện mà mọi thể hiện của quan hệ phải thỏa mãn
  - RBTV xuất phát từ các quy tắc quản lý được áp đặt lên thế giới thực
- Ví dụ:
  - RBTV1: Mỗi bộ của quan hệ DON_DAT_HANG phải ứng với một đơn đặt hàng với mã đơn đặt hàng (MaDDH) duy nhất.
  - RBTV2: Mọi chi tiết về đơn đặt hàng phải có mã hàng (MaHang) thuộc về danh mục hàng.
  - RBTV3: Mã đơn đặt hàng (MaDDH) khác NULL.
  - RBTV4: Tổng các trị giá của các mặt hàng trong CHI_TIET_DON_HANG có cùng MaDDH phải bằng TongGT ghi trong DON_DAT_HANG
- Mục đích của RBTV
  - Đảm bảo tính nhất quán của dữ liệu(RBTV2, RBTV4)
  - Đảm bảo ngữ nghĩa nghĩa thực tế của dữ liệu(RBTV1, RBTV3)

Khái niệm ràng buộc toàn vẹn

- RBTV được xác định và mô tả trong quá trình thiết kế csdl. RBTV có 3 yếu tố chính:
  - Nội dung
  - Bối cảnh
  - Tầm ảnh hưởng
- RBTV được khai báo thông qua ngôn ngữ định nghĩa và thao tác dữ liệu và được hỗ trợ bởi hqt csdl.
  - Mệnh đề check, arssertion, triger
- RBTV được kiểm tra và xử lý bởi hqt csdl.

Nội dung của RBTV

- Được phát biểu
  - Ngôn ngữ tự nhiên: Đơn giản dễ hiểu
  - Ngôn ngữ hình thức: khó hiểu
    - Đại số quan hệ, phép tính quan hệ, mã giả
    - Biểu thức toán học
- Ví dụ:
  - RBTV5:
    - Mỗi nhân viên có một mã số dùng để phân biệt với những nhân viên khác

      $\forall t_1, t_2 \in NHAN\_VIEN(t_1 \neq t_2 \Rightarrow t_1.MaNV \neq t_2.MaNV)$
  - RBTV6:
    - Mỗi nhân viên làm việc trong một phòng ban

      $NHAN\_VIEN[MaPhong] \in PHONG\_BAN[MaPhong]$
  - RBTV7:
    - Mỗi phòng phải có ít nhất một nhân viên

      $\forall s \in PHONG\_BAN(\exists t \in NHAN\_VIEN(t.MaPhong = s.MaPhong))$

Bối cảnh của RBTV

- Là những quan hệ mà RBTV có hiệu lực: Một hoặc nhiều quan hệ
- Ví dụ
  - RBTV5 có bối cảnh là quan hệ NHAN_VIEN
  - RBTV6, RBTV7 có bối cảnh là quan hệ NHAN_VIEN, PHONG_BAN

Tầm ảnh hưởng

- RBTV có thể bị vi phạm khi thực hiện các thao tác cập nhật trên bối cảnh: Thêm, xóa, sửa
- Bảng tầm ảnh hưởng dùng để xác định thời điểm cần kiểm tra RBTV

Xây dựng bảng tầm ảnh hưởng cho từng RBTV

- Nội dung mỗi ô
  - +: Cần phải kiểm tra RBTV
  - - : Không cần phải kiểm tra RBTV

Tên RBTV	Thêm	Xóa	Sửa
Quan hệ 1	+	+	-
...	...	...	...
Quan hệ k	+	-	-

Các quan hệ bối cảnh

Ví dụ

RBTV5	Thêm	Xóa	Sửa
NHAN_VIEN	+	-	-

RBTV6	Thêm	Xóa	Sửa
NHAN_VIEN	+	-	+
PHONG_BAN	-	+	-

RBTV7	Thêm	Xóa	Sửa
NHAN_VIEN	-	-	+
PHONG_BAN	+	-	-

Xây dựng bảng tầm ảnh hưởng tổng hợp

- Xây dựng trên cơ sở bảng tầm ảnh hưởng của các RBTV
- Để xác định thời điểm kiểm tra RBTV khi một thao tác cập nhật trên quan hệ nào đó được thực hiện

	Tên RBTV 1				Tên RBTV r		
	T	X	S	...	T	X	S
Quan hệ 1	+	+	-		+	-	+
...	...	...	...	...	...	...	...
Quan hệ n	-	-	+		+	+	+

Ví dụ

- Xây dựng bảng tầm ảnh hưởng cho các ràng buộc toàn vẹn RBTV1…, RBTV7 cho lược đồ CSDL quan hệ siêu thị M ?

Phân loại RBTV

- Dựa trên yếu tố bối cảnh của RBTV
  - RBTV có bối cảnh là một quan hệ/RBTV trên thuộc tính
    - RBTV miền giá trị
    - RBTV liên thuộc tính
    - RBTV liên bộ
  - RBTV có bối cảnh là nhiều quan hệ/RBTV trên quan hệ
    - RBTV tham chiếu
    - RBTV thuộc tính tổng hợp
    - RBTV liên thuộc tính
    - RBTV liên bộ

3.2. Ràng buộc toàn vẹn trên thuộc tính

a.Ràng buộc toàn vẹn về miền giá trị của một thuộc tính

- RBTV đặc tả tập giá trị có thể kết hợp với một thuộc tính.
- Ví dụ: quan hệ NHAN_VIEN
  - RBTV8: Tuổi của nhân viên trong công ty phải lớn hơn 18 và nhỏ hơn 65.

RBTV8 $\forall t \in NHAN\_VIEN(18 \leq t.Tuoi \leq 65)$	T	X	S
NHAN_VIEN	+	-	+

b. RBTV liên thuộc tính

- RBTV có liên quan tới nhiều thuộc tính của một quan hệ. Thông thường đó là các phụ thuộc tính toán, hoặc một suy diễn từ giá trị của một hay nhiều thuộc tính trong cùng một bộ giá trị.
- Ví dụ
  - Quan hệ NHAN_VIEN
  - RBTV9: Nếu nhân viên có giới tính là nữ tuổi của nhân viên trong công ty phải lớn hơn 18 và nhỏ hơn 55.

$$\forall t \in NHAN\_VIEN\,(18 \le t.Tuoi \le 55(t.GioiTinh = Nu))$$

RBTV9	T	X	S
NHAN_VIEN	+	-	+

c. RBTV liên bộ, liên thuộc tính

- RBTV có liên quan tới nhiều bộ và có thể tới nhiều thuộc tính của (các) bộ giá trị trong một quan hệ.
- Ví dụ: RBTV5

3.3. Ràng buộc toàn vẹn trên quan hệ

a. RBTV tham chiếu/RBTV về phụ thuộc tồn tại (phụ thuộc về khóa ngoại)

- Bộ giá trị của quan hệ này được thêm vào một cách hợp lệ nếu tồn tại một bản ghi tương ứng trong một quan hệ khác.
- Đảm bảo rằng giá trị xuất hiện trong một quan hệ đối với một tập các thuộc tính đã cho cũng xuất hiện đối với một tập các thuộc tính nhất định trong một quan hệ khác.

a. Ràng buộc toàn vẹn về phụ thuộc tồn tại (phụ thuộc về khóa ngoại)

- Ví dụ:
  - RBTV1 : Mỗi bộ của CHI_TIET_DON_HANG phải có một đơn hàng với MaDDH tương ứng

$$\forall t \in CHI\_TIET\_DON\_HANG(\exists u \in DON\_HANG(t.MaDDH = u.MaDDH))$$

RVTV4	T	X	S
CHI_TIET_DON_HANG	+	-	-
DON_HANG	-	+	-

Ví dụ

- RBTV2 : Mỗi bộ của CHI_TIET_DON_HANG phải có MaHang thuộc về danh mục hàng trong quan hệ MAT_HANG
- Biểu diễn hình thức ? Bảng xác định tầm ảnh hưởng?

b. *Ràng buộc toàn vẹn tổng hợp*

- Khi có sự hiện diện của 1 thuộc tính mang tính chất tổng hợp (tức là giá trị của thuộc tính có thể được tính toán từ giá trị của các thuộc tính khác trên một hay nhiều bộ giá trị của các quan hệ trong CSDL)
- Ví dụ:
  - DON_DAT_HANG(**MaDDH**, NgayLap, TenKH, TongGT)
  - CHI_TIET_DON_HANG(**MaDDH**, **TenSanPham**, SoLuong, Giatri).
  - RBTV4: Tổng các trị giá của các mặt hàng trong CHI_TIET_DON_HANG có cùng MaDDH phải bằng TongGT ghi trong DON_DAT_HANG .

$$\forall t \in DON\_DAT\_HANG(t.TongGT = \sum u.GiaTri(u \in CHI\_TIET\_DON\_HANG \wedge u.MaDDH = t.MaDDH))$$

RVTV4	T	X	S
CHI_TIET_DON_HANG	+	+	+
DON_DAT_HANG	+	-	+

c. RBTV liên thuộc tính

- Mối quan hệ giữa các thuộc tính trong nhiều lược đồ quan hệ
- Ví dụ
  - Quan hệ:
    - NHAN_VIEN (**TenNV**, Luong, NgaySinh TenPhong)
    - PHONG_BAN(**TenPhong**, MaPhong, NguoiQuanLy, NgayNhanChuc)
  - RBTV 10:
    - Ngày nhận chức của trưởng phòng phải lớn hơn ngày sinh
    - Biểu diễn hình thức. Bảng xác định tầm ảnh hưởng?

d.Ràng buộc toàn vẹn liên bộ

- Mối quan hệ giữa các bộ trên nhiều lược đồ quan hệ
- Ví dụ
  - Quan hệ NHAN_VIEN, PHONG_BAN
  - RBTV11:
    - Lương của nhân viên không được cao hơn lương trưởng phòng
    - Biểu diễn hình thức? Bảng xác định tầm ảnh hưởng?

Chỉ mục

- Functional dependency : Phụ thuộc hàm
- Functional determinant: Xác định hàm
- Normal Form: Dạng chuẩn

Phụ lục

- Thuật toán tìm phủ tối thiểu của một tập phụ thuộc hàm
  - Thuật toán 4.2 (128, Nguyên lý của các hệ csdl)

……..o0o……..

Cơ sở dữ liệu nâng cao

CHƯƠNG 1
TỔNG QUAN VỀ HỆ QUẢN TRỊ CƠ SỞ DỮ LIỆU QUAN HỆ

Trong chương này chúng ta hệ thống lại các khái niệm cơ bản của cơ sở dữ liệu quan hệ. Mục đích là định nghĩa các thuật ngữ, đưa ra bộ khung cơ sở cho các phần sau. Việc chọn mô hình cơ sở dữ liệu quan hệ làm hệ thống cơ sở có nhiều lý do: thứ nhất là cơ sở toán học vững chắc của mô hình quan hệ làm nó trở thành mô hình lý tưởng trong việc giải quyết các vấn đề lý thuyết, hai là phần lớn các vấn đề trình bày trong các chương sau có thể mô tả dễ dàng bởi mô hình quan hệ, ba là thị trường hệ quản trị cơ sở dữ liệu quan hệ rất phát triển và vẫn đang mở rộng và cuối cùng là phần lớn các hệ cơ sở dữ liệu sau này cũng thuộc loại quan hệ.

Mô hình quan hệ có ba đặc trưng cơ bản:

1. Cấu trúc dữ liệu đơn giản. Chúng là các quan hệ (relation), được biểu diễn như các bảng 2 chiều với phần tử là các bản ghi (record, data item). Nó cung cấp đặc tính độc lập ở mức độ cao so với dạng biểu thị vật lý của dữ liệu.
2. Mô hình quan hệ cung cấp một nền tảng vững chắc, bảo đảm được tính nhất quán dữ liệu (data consistency). Thiết kế cơ sở dữ liệu được hỗ trợ bởi quá trình chuẩn hoá, giúp loại bỏ các bất thường dữ liệu. Các trạng thái nhất quán của một cơ sở dữ liệu có thể được định nghĩa và duy trì một cách thống nhất qua các quy tắc toàn vẹn cơ sở dữ liệu (integrity rules).
3. Mô hình quan hệ cho phép các thao tác trên quan hệ theo kiểu tập hợp. Đặc tính này đã dẫn đến việc phát triển các ngôn ngữ phi thủ tục mạnh mẽ với nền tảng là lý thuyết tập hợp (đại số quan hệ) hoặc lôgic (phép tính quan hệ).

1. Khái niệm cơ sở dữ liệu

1.1. Cơ sở dữ liệu

Dữ liệu được lưu trữ trên các thiết bị lưu trữ theo một cấu trúc nào đó để có thể phục vụ cho nhiều người sử dụng với nhiều mục đích khác nhau gọi là *cơ sở dữ liệu.*

1.2. Hệ quản trị cơ sở dữ liệu

Phần mềm cho phép một hoặc nhiều người tạo lập, lưu trữ, cập nhật và khai thác cơ sở dữ liệu gọi là *hệ quản trị cơ sở dữ liệu* (DataBase Management Systems - DBMS).

Vai trò chính của hệ quản trị cơ sở dữ liệu là cho phép người dùng thao tác với dữ liệu thông qua các thuật ngữ trìu tượng, khác với việc máy tính lưu trữ dữ liệu. Theo nghĩa này hệ quản trị cơ sở dữ liệu có nhiệm vụ như là một bộ thông dịch (interpreter) với ngôn ngữ bậc cao nhằm giúp người dùng sử dụng hệ thống mà không cần quan tâm đến cách biểu diễn dữ liệu trong máy hoặc các thuật toán chi tiết. Ví dụ người dùng không cần biết hệ quản trị cơ sở dữ liệu *Access* tổ chức dữ liệu theo kiểu hàm băm, kiểu file chỉ mục hay kiểu cây cân bằng, và cũng không cần biết thuật toán thực hiện lệnh sắp xếp là Quick Sort, thuật toán nổi bọt hay sắp xếp nhị phân ...

Một cơ sở dữ liệu gồm một hoặc nhiều tập tin được thiết kế theo một cấu trúc nhất định và có quan hệ chặt chẽ với nhau. Cơ sở dữ liệu được dùng chung cho

nhiều người và nhiều mục đích khác nhau, vì vậy sẽ tiết kiệm được tài nguyên, giảm thiểu sự trùng lặp thông tin, bảo đảm tính nhất quán thông tin.

1.3. Các đặc trưng của phương pháp cơ sở dữ liệu

+ *Chia sẻ dữ liệu.* Mục đích chính của cách tiếp cận cơ sở dữ liệu là dữ liệu được chia sẻ bởi nhiều người dùng hợp pháp.

+ *Giảm thiểu dư thừa dữ liệu.* Dữ liệu dùng chung cho nhiều bộ phận, thay vì được lưu trữ phân tán trùng lặp nay được lưu trữ tập trung một chỗ theo một cấu trúc thống nhất.

+ *Tính tương thích dữ liệu.* Việc loại bỏ sự dư thừa dữ liệu kéo theo hệ quả là sự tương thích dữ liệu. Ví dụ khi địa chỉ nhân viên thay đổi thì tất cả các bộ phận đều được cập nhật địa chỉ mới.

+ *Tính toàn vẹn dữ liệu* (data integrity). Mỗi cơ sở dữ liệu cần đảm bảo một số loại ràng buộc toàn vẹn (integrity constraints). Đặc biệt khi người dùng thực hiện các thao tác như chèn, xoá hay sửa đổi dữ liệu thì các ràng buộc đó cần phải được kiểm tra một cách chặt chẽ.

+ *Bảo mật dữ liệu (data security).* Khi có nhiều người cùng chia sẻ dữ liệu, việc bảo đảm an toàn dữ liệu và bảo mật thông tin là tối quan trọng. Cần phải có cơ chế bảo mật như mật khẩu (*password*) và phân quyền truy cập dữ liệu (*data access rights*).

+ *Tính đồng bộ dữ liệu* (synchronization). Thông thường cơ sở dữ liệu được nhiều người dùng truy cập đồng thời, gây nên sự cạnh tranh dữ liệu. Vì vậy cần có cơ chế bảo vệ chống sự không tương thích bởi các thao tác cùng lúc lên dữ liệu.

+ *Tính độc lập dữ liệu.* Sự tách biệt cấu trúc mô tả dữ liệu khỏi chương trình ứng dụng sử dụng dữ liệu gọi là độc lập dữ liệu. Điều này cho phép phát triển tổ chức dữ liệu mà không cần sửa đổi chương trình ứng dụng. Sự độc lập dữ liệu là một trong mục tiêu chính của cách tiếp cận cơ sở dữ liệu.

Tương tự như một phần mềm, *vòng đời* của cơ sở dữ liệu gồm có các giai đoạn chính sau:

- Lập kế hoạch cơ sở dữ liệu (database planning).
- Khảo sát, phân tích (study and analysis).
- Thiết kế cơ sở dữ liệu (database design).
- Cài đặt cơ sở dữ liệu (database implementation).
- Bảo trì cơ sở dữ liệu (post-implementation).

1.4. Lược đồ dữ liệu và thể hiện dữ liệu

Khi thiết kế cơ sở dữ liệu ta tạo ra cấu trúc cơ sở dữ liệu, cái đó gọi là *lược đồ dữ liệu.* Ví dụ lược đồ dữ liệu hồ sơ nhân sự gồm các thành phần sau:

Họ tên, ngày sinh, hệ số lương

Các thành phần của lược đồ dữ liệu gọi là *thuộc tính* hoặc *trường*.

Khi sử dụng cơ sở dữ liệu thì ta làm việc với dữ liệu thật sự, đó là sự *thể hiện dữ liệu*. Ví dụ đối với lược đồ trên ta có thể có các thể hiện dữ liệu sau:

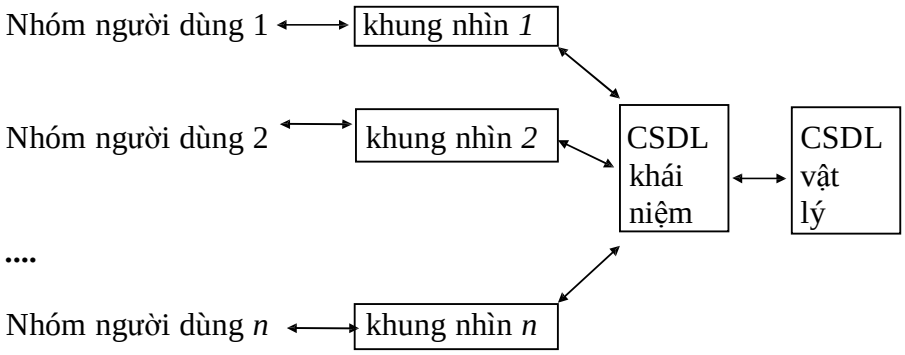
Nguyễn Văn A, 20/08/1970, 3.40

Trần Thị B, 15/05/1962, 4.12

Mỗi thể hiện dữ liệu gọi là *bộ* hay *bản ghi*.

1.5. Các mức trìu tượng dữ liệu

Giữa máy tính thao tác với các bit, người thiết kế cơ sở dữ liệu và người dùng có cách nhìn khác nhau đối với dữ liệu, đó chính là các *mức trìu tượng dữ liệu*. Sơ đồ chuẩn về các mức trìu tượng như sau:



a. Cơ sở dữ liệu vật lý: nằm cố định trong các thiết bị lưu trữ như đĩa và băng từ. Bản thân cơ sở dữ liệu vật lý cũng có nhiều mức trìu tượng khác nhau: từ mức bản ghi và file trong ngôn ngữ lập trình như PASCAL, qua mức bản ghi lôgic được hỗ trợ bởi hệ điều hành, đến mức các bit và địa chỉ vật lý trong các thiết bị lưu trữ.

b. Cơ sở dữ liệu khái niệm (schema): là sự trìu tượng thế giới thực đối với một đối tượng nào đó. Hệ quản trị cơ sở dữ liệu cung cấp ngôn ngữ thiết kế dữ liệu để thiết kế sơ đồ khái niệm. Đây là ngôn ngữ bậc cao cho phép mô tả cơ sở dữ liệu khái niệm bằng ngôn ngữ "mô hình dữ liệu". Một ví dụ điển hình là đồ thị có hướng trong mô hình mạng, trong đó các nút biểu diễn các đơn thể và các cung biểu diễn quan hệ.

c. Khung nhìn hoặc *lược đồ con (subschema)*: là mô hình trìu tượng một phần của cơ sở dữ liệu khái niệm. Có những hệ trang bị công cụ gọi là ngôn ngữ thiết kế khung nhìn cho phép khai báo khung nhìn, và công cụ gọi là ngôn ngữ thao tác dữ liệu khung nhìn để diễn tả câu hỏi và các thao tác đối với khung nhìn.

Theo một nghĩa nào đó, khung nhìn là cơ sở dữ liệu khái niệm và cùng mức trìu tượng như cơ sở dữ liệu khái niệm. Mặt khác khung nhìn có thể trìu tượng hơn theo nghĩa dữ liệu của nó được suy ra từ cơ sở dữ liệu khái niệm.

d. Độc lập dữ liệu

Độc lập dữ liệu dữ liệu giữa các mức trìu tượng có ý nghĩa rất quan trọng đối với cơ sở dữ liệu.

- *Độc lập dữ liệu mức vật lý*: Sự thay đổi lược đồ dữ liệu vật lý không làm thay đổi lược đồ dữ liệu mức khái niệm và mức khung nhìn.

Ta cần hiểu rằng sự thay đổi tổ chức vật lý dữ liệu có thể làm ảnh hưởng đến hiệu quả chương trình ứng dụng. Sự độc lập dữ liệu mức vật lý đảm bảo không phải viết lại chương trình chỉ vì lý do thay đổi cách tổ chức dữ liệu. ý nghĩa của tính độc lập dữ liệu mức vật lý là nó cho phép ta tinh chỉnh cơ sở dữ liệu mức vật lý để tăng hiệu quả sử dụng trong khi các chương trình ứng dụng vẫn chạy bình thường như không có vấn đề gì xảy ra.

- *Độc lập dữ liệu lôgic*: Sự thay đổi lược đồ dữ liệu khái niệm không làm thay đổi khung nhìn.

Trong quá trình sử dụng cơ sở dữ liệu có thể ta phải sửa đổi hiệu chỉnh lược đồ khái niệm, chẳng hạn thêm thông tin về thực thể mà cơ sở dữ liệu mô tả.

1.6. Ngôn ngữ dữ liệu

Mỗi hệ quản trị cơ sở dữ liệu cần phải có ngôn ngữ riêng của mình. Có hai loại ngôn ngữ cơ sở dữ liệu.

a. Ngôn ngữ mô tả dữ liệu (Data Definition Language - DDL). Gồm các lệnh cho phép khai báo, hiệu chỉnh cấu trúc cơ sở dữ liệu, mô tả các mối quan hệ của dữ liệu cũng như các quy tắc áp đặt lên dữ liệu. Ngôn ngữ mô tả dữ liệu được xây dựng dựa trên loại mô hình dữ liệu (mô hình quan hệ, mô hình mạng, mô hình phân cấp, mô hình hướng đối tượng ...) mà hệ quản trị cơ sở dữ liệu tương ứng được thiết kế.

b. Ngôn ngữ thao tác dữ liệu (Data Manipulation Language - DML). Là bộ lệnh cho phép người dùng thực hiện các công việc:

- Cập nhật dữ liệu như thêm, sửa, xoá.
- Truy vấn, tổng hợp dữ liệu.
- Các hàm tính toán.
- Bảo mật dữ liệu.

* *Giao tiếp ngôn ngữ chủ* : là khả năng cho phép chương trình viết trong các ngôn ngữ bậc cao như COBOL, C , ... có thể truy cập xử lý dữ liệu trong cơ sở dữ liệu.

* *Ngôn ngữ truy vấn SQL* : là ngôn ngữ có cú pháp tiếng Anh, giúp người dùng có thể thao tác dữ liệu dễ dàng mà không cần lập trình. Đây là loại ngôn ngữ thế hệ thứ 4 với đặc trưng *phi thủ tục*.

2. Khái niệm cơ sở dữ liệu quan hệ

2.1. Miền

Miền là tập hợp các giá trị. Người ta thường dùng chữ hoa để ký hiệu miền.

◊ *Ví dụ*. Các tập hợp sau là các miền:

- Tập các số nguyên.
- Tập các xâu ký tự độ dài không quá 30 ký tự.
- A = {0,1}.

2.2. Tích Đề-các

Tích Đề-các của các miền $D_1, D_2,..., D_n$, ký hiệu là

$$D_1 \times D_2 \times ... \times D_n,$$

là tập hợp tất cả *n-bộ* $(v_1, v_2, ..., v_n)$ thoả mãn

$$v_1 \in D_1, v_2 \in D_2, ..., v_n \in D_n.$$

Tức là

$$D_1 \times D_2 \times ... \times D_n = \{(v_1, v_2,..., v_n) \mid v_1 \in D_1, v_2 \in D_2, ..., v_n \in D_n \}.$$

◊ *Ví dụ.* Cho $D_1 = \{0,1\}$, $D_2 = \{a,b,c\}$. Khi đó tích đề-các

$$D_1 \times D_2 = \{(0,a),(0,b),(0,c),(1,a),(1,b),(1,c)\}$$

2.3. Quan hệ

Quan hệ là tập con của tích đề-các của 1 hoặc nhiều miền. Quan hệ có thể có hữu hạn hoặc vô hạn số phần tử. Trong giáo trình này ta giả thiết rằng quan hệ có hữu hạn phần tử.

◊ *Ví dụ*

Cho $D_1 = \{0,1\}$, $D_2 = \{a,b,c\}$. Tập **r** $= \{(0,a),(1,b),(1,c)\} \subset D_1 \times D_2$, vậy **r** là quan hệ trên D_1 và D_2.

Ta nói quan hệ **r** có *bậc n* nếu **r** là tập con của tích đề-các của *n* miền.

Mỗi phần tử của quan hệ gọi là *bộ*. Mỗi bộ của quan hệ bậc *n*, còn gọi là *n-bộ*, có *n thành phần*. Mỗi thành phần của bộ là *nguyên tố*, có nghĩa không thể phân tách được thành các thành phần nhỏ hơn.

Để trực quan ta có thể coi quan hệ như một *bảng* trong đó mỗi *hàng* là một bộ và mỗi *cột* ứng với một thành phần.

Dữ liệu được tổ chức dưới dạng các quan hệ có liên quan với nhau gọi là *cơ sở dữ liệu quan hệ.*

Mỗi cột của quan hệ được gán một tên gọi là *thuộc tính*.

Tập hợp tất cả các tên thuộc tính của quan hệ gọi là *lược đồ quan hệ*.

Tập hợp các lược đồ quan hệ của một cơ sở dữ liệu gọi là *lược đồ cơ sở dữ liệu quan hệ.*

· ***Các tính chất của quan hệ***

- Các giá trị trên cột phải cùng một miền giá trị và đơn trị.

Giá trị trên giao của cột và hàng đơn trị, không chấp nhận nhiều giá trị.

- Mỗi hàng là duy nhất.

Không cho phép hai hàng hoàn toàn giống nhau.

◊ *Ký hiệu*

Lược đồ quan hệ R có các thuộc tính $A_1, A_2, ..., A_n$ ký hiệu là

$$R = (A_1, A_2, ..., A_n)$$

Quan hệ **r** với lược đồ $R = (A_1, A_2, ..., A_n)$ có thể viết

r(R) hoặc **r**(A_1, A_2, ..., A_n)

Cho **r** là quan hệ với lược đồ R = (A_1, A_2, ..., A_n), *t* là một bộ của **r**, A ⊂ {A_1, A_2, ..., A_n}. Khi đó *t*(A) ký hiệu bộ các thành phần của *t* ứng với các thuộc tính trong tập A. Nếu A là 1 thuộc tính thì *t*(A) chính là giá trị thành phần ứng với thuộc tính A.

◊ *Ví dụ*

Đây là ví dụ sẽ dùng làm cơ sở dữ liệu mẫu để mô hình hoá một công ty. Các thực thể được mô hình hoá là:

- Các *nhân viên*, ký hiệu EMP, viết tắt từ *employee*.
- Các *dự án*, ký hiệu PROJ, viết tắt từ *project*.

Đối với mỗi nhân viên chúng ta muốn theo dõi các thông tin sau:

- *Mã số nhân viên*, ký hiệu *ENO*, viết tắt từ *employee number*.
- *Tên nhân viên*, ký hiệu *ENAME*, viết tắt từ *employee name*.
- *Chức vụ*, ký hiệu *TITLE*.
- *Lương*, ký hiệu *SAL*, viết tắt từ *salary*.
- *Mã số dự án*, ký hiệu *PNO*, viết tắt từ *project number*.
- *Nhiệm vụ dự án*, ký hiệu *RESP*, viết tắt từ *responsibility*.
- *Thời gian làm việc trong dự án*, ký hiệu *DUR*, viết tắt từ *duration*.

Đối với mỗi dự án chúng ta muốn theo dõi các thông tin sau:

- *Mã số dự án*, ký hiệu *PNO*, viết tắt từ *project number*.
- *Tên dự án*, ký hiệu *PNAME*, viết tắt từ *project name*.
- *Kinh phí dự án*, ký hiệu *BUDGET*.

Các lược đồ quan hệ (relation scheme) cho cơ sở dữ liệu này có thể định nghĩa như sau:

EMP(*ENO, ENAME, TITLE, SAL, PNO, RESP, DUR*)

PROJ(*PNO, PNAME, BUDGET*)

Lược đồ EMP có 7 thuộc tính (attribute): *ENO, ENAME, TITLE, SAL, PNO, RESP, DUR*.

Giá trị của *ENO* lấy tự miền chứa các mã số nhân viên, giả sử là D_1, Giá trị của *ENAME* lấy tự miền chứa các tên nhân viên hợp lệ, giả sử là D_2, ...

Đây là một thể hiện cơ sở dữ liệu mẫu của chúng ta gồm hai bảng như sau:

EMP

ENO	*ENAME*	*TITLE*	*SAL*	*PNO*	*RESP*	*DUR*
E1	J.Doe	Elect.Eng.	40000	P1	Manager	12
E2	M.Smith	Syst.Anal.	34000	P1	Analyst	24
E2	M.Smith	Syst.Anal.	34000	P2	Analyst	6
E3	A.Lee	Mech.Eng.	27000	P3	Consultant	10
E3	A.Lee	Mech.Eng.	27000	P4	Engineer	48
E4	J.Miller	Programmer	24000	P2	Programmer	18
E5	B.Casey	Syst.Anal.	34000	P2	Manager	24

E6	L.Chu	Elect.Eng.	40000	P4	Manager	48
E7	R.David	Mech.Eng.	27000	P3	Engineer	36
E8	J.Jones	Syst.Anal.	34000	P3	Manager	40

PROJ

PNO	*PNAME*	*BUDGET*
P1	Instrumentation	150000
P2	Database Development	135000
P3	CAD/CAM	250000
P4	Maintenance	310000

Các cột của bảng tương ứng các thuộc tính của quan hệ. Các thông tin nhập theo hàng tương ứng với các bộ hay bản ghi. Dòng trên cùng biểu diễn lược đồ quan hệ của bảng.

Một giá trị của thuộc tính, chẳng hạn lương của nhân viên, thời gian tham gia dự án,... tại thời điểm nào đó có thể chưa được xác định. Khi đó có nhiều cách diễn giải khác nhau như "chưa được biết" hay "chưa áp dụng được". "Giá trị đặc biệt" thường được gọi là ***null***. Giá trị ***null*** phải khác các giá trị trong miền thuộc tính, và cũng cần phân biệt nó với giá trị *zero* (với thuộc tính kiểu số) hay giá trị *rỗng* (với thuộc tính kiểu ký tự).

2.4. Khoá

a) Siêu khoá

Cho lược đồ quan hệ $R=(A_1, A_2, ..., A_n)$ và tập con $S \subset \{ A_1, A_2, ..., A_n \}$. Tập S gọi là *siêu khoá* (superkey) của lược đồ R nếu các thuộc tính của S xác định duy nhất các bộ của mỗi quan hệ của lược đồ R, tức là với mọi quan hệ **r** của lược đồ R phải thoả mãn:

$$\forall\, t_1, t_2 \in \mathbf{r}: t_1 \neq t_2 \Rightarrow \exists\, A \in S: t_1(A) \neq t_2(A)$$

Lưu ý rằng theo định nghĩa, mỗi bộ là duy nhất nên đối với mỗi lược đồ quan hệ, *tập hợp tất cả thuộc tính là siêu khoá*. Siêu khoá là cơ sở để phân biệt 2 bộ khác nhau trong 1 quan hệ. Một lược đồ có thể có nhiều siêu khoá. Tính chất của siêu khoá là quy luật được xác định trong quá trình phân tích thiết kế cơ sở dữ liệu.

◊ *Ghi chú*

(1) Tập tất cả thuộc tính R là siêu khoá (tầm thường).

(2) $S \subset T \subset R$ & S là siêu khoá $\Rightarrow$ T là siêu khoá.

b) Khoá

Tập K các thuộc tính của lược đồ R là *khoá (key)* nếu K là siêu khoá *cực tiểu*, tức là mọi tập con thực sự của K không phải là siêu khoá.

· *Mệnh đề*: Mọi lược đồ quan hệ luôn có khóa.

Chứng minh. Mệnh đề suy ra từ sự tồn tại phần tử cực tiểu trong tập có quan hệ thứ tự.

◊ *Ví dụ*

Lược đồ PROJ có khoá là *PNO*.

Lược đồ EMP có khoá là (*ENO, PNO*).

Các thuộc tính thuộc khoá nào đó gọi là *thuộc tính khoá* hay *thuộc tính nguyên tố*.

Thuộc tính không phải thuộc tính khoá gọi là *thuộc tính không khoá*.

Mỗi quan hệ có ít nhất một khoá. Trường hợp có nhiều khoá thì gọi các khoá đó là *khoá dự tuyển* (candidate key), trong đó có một khoá là *khoá chính* (primary key).

c) Khoá ngoại

Cho lược đồ R và lược đồ Q. Tập con H các thuộc tính của R gọi là *khoá ngoại* của R *tham chiếu* đến lược đồ Q, nếu Q có khoá K gồm các thuộc tính (có thể dưới tên khác) của H thoả mãn:

Với mọi quan hệ ***r*** và ***q*** là các quan hệ của 1 cơ sở dữ liệu ứng với lược đồ R và Q ta có:

$$\forall x \in \mathbf{r}\ \exists y \in \mathbf{q}: x(H) = y(K)$$

◊ *Ví dụ*

Lược đồ EMP có khoá ngoại *PNO* tham chiếu đến khoá *PNO* của lược đồ PROJ.

Ở dạng bảng cơ sở dữ liệu mẫu của chúng ta gồm 2 bảng như sau

EMP

ENO	*ENAME*	*TITLE*	*SAL*	*PNO*	*RESP*	*DUR*
				P1		

PROJ

PNO	*PNAME*	*BUDGET*
P1		

3. Quy tắc toàn vẹn

Qui tắc toàn vẹn (*integrity rule*) là các ràng buộc đảm bảo trạng thái nhất quán của cơ sở dữ liệu. Chúng thường được diễn tả như là các ràng buộc toàn vẹn.

Có các loại qui tắc toàn vẹn sau: *Toàn vẹn thực thể* (Entity integrity), *Miền giá trị* (Domains integrity), *Toàn vẹn tham chiếu* (Referential integrity), *Thao tác bẫy* (Triggering operations).

3.1. Qui tắc toàn vẹn thực thể

Qui tắc *toàn vẹn thực thể* yêu cầu thực thể phải có khoá chính, các thuộc tính khoá phải có giá trị duy nhất và khác *null*. Qui tắc này không cho phép hai bản ghi trùng khoá.

◊ *Ví dụ*. Xét cơ sở dữ liệu

EMP(*ENO, ENAME, TITLE, SAL, PNO, RESP, DUR*)
PROJ(*PNO, PNAME, BUDGET*)

Qui tắc toàn vẹn thực thể ràng buộc:

- Trong lược đồ PROJ thuộc tính khoá *PNO* không thể nhận giá trị *null* và có giá trị không trùng nhau.

- Trong lược đồ EMP cặp thuộc tính khoá (*EMP, PNO*) không thể nhận giá trị *null* và có giá trị không trùng nhau.

3.2. Qui tắc miền giá trị

Đây là loại ràng buộc lên các giá trị hợp lệ của thuộc tính. *Miền giá trị* là tập hợp tất cả các loại dữ liệu và phạm vi giá trị được thuộc tính thừa nhận. Định nghĩa miền giá trị xác định các tham số đặc trưng của thuộc tính: kiểu dữ liệu (data type), độ dài (length), khuôn dạng (format), phạm vi (range), giá trị cho phép (allowable values), ý nghĩa (meaning), tính duy nhất (uniqueness), chấp nhận giá trị *null* (null support).

◊ *Ví dụ*. Xét quan hệ

PROJ(*PNO, PNAME, BUDGET*)

Các thuộc tính *PNAME* và *BUDGET* có ràng buộc miền giá trị như sau

Tên thuộc tính	:	*PNAME*	*BUDGET*
Ý nghĩa	:	Tên dự án	Kinh phí dự án
Kiểu dữ liệu	:	Ký tự (Character)	Số (numeric)
Độ dài	:	20	10
Format	:		9,999,999
Phạm vi	:		> 1000 & <10,000,000
Giá trị cho phép	:		
Duy nhất	:	Có	Không
Null support	:	Non-null	Null

3.3. Qui tắc toàn vẹn tham chiếu

Toàn vẹn tham chiếu là ràng buộc đảm bảo tính hợp lệ của sự tham chiếu của một đối tượng trong cơ sở dữ liệu (gọi là đối tượng tham chiếu) đến đối tượng khác (gọi là đối tượng được tham chiếu) trong cơ sở dữ liệu đó. Các thuộc tính tương ứng gọi *thuộc tính cặp ghép* của ràng buộc tham chiếu.

◊ *Ví dụ*. Xét các quan hệ

EMP(*ENO, ENAME, TITLE, SAL, PNO, RESP, DUR*)

PROJ(*PNO, PNAME, BUDGET*)

Với mỗi bộ **e** ∈ EMP phải tồn tại bộ **p**∈ PROJ sao cho

$$\mathbf{e}(PNO) = \mathbf{p}(PNO)$$

Thuộc tính *PNO* của quan hệ EMP là khoá ngoại tham chiếu đến khoá chính *PNO* của quan hệ PROJ: EMP.PNO→PROJ.PNO

Quan hệ EMP là quan hệ tham chiếu và quan hệ PROJ là quan hệ được tham chiếu, với thuộc tính cặp ghép là (*EMP.PNO*, *PROJ.PNO*).

Qui tắc toàn vẹn tham chiếu được xét đến trong khi cập nhật quan hệ tham chiếu hoặc quan hệ được tham chiếu. Ta xét các qui tắc con sau.

· ***Qui tắc chèn***: Không thể chèn hàng mới vào quan hệ tham chiếu nếu quan hệ được tham chiếu chưa có dữ liệu thuộc tính cặp ghép tương ứng.

◊ *Ví dụ*. Xét các quan hệ EMP và PROJ. Giả sử ta muốn chèn bản ghi

e = (E1, J.Doe, Elect.Eng., 40000, P5, Manager, 20)

trong đó P5 là dự án Elect.Commerce (thương mại điện tử) với kinh phí 500000 USD.

Khi đó, nếu quan hệ PROJ chưa có bản ghi

p = (P5, Elect.Commerce, 500000)

thì qui tắc chèn đảm bảo không thể chèn bản ghi **e** vào quan hệ EMP được.

Muốn chèn bản ghi **e** vào quan hệ EMP, trước hết ta phải chèn bản ghi **p** vào quan hệ PROJ.

· ***Qui tắc xoá***: Không thể xoá hàng của quan hệ được tham chiếu nếu hàng đó có dữ liệu thuộc tính cặp ghép tương ứng trong quan hệ tham chiếu.

◊ *Ví dụ*. Xét các quan hệ EMP và PROJ.

EMP

ENO	*ENAME*	*TITLE*	*SAL*	*PNO*	*RESP*	*DUR*
E1	J.Doe	Elect.Eng.	40000	P1	Manager	12
E2	M.Smith	Syst.Anal.	34000	P1	Analyst	24
E2	M.Smith	Syst.Anal.	34000	P2	Analyst	6
...	...	...	...	...	...	...

PROJ

PNO	*PNAME*	*BUDGET*

P1	Instrumentation	150000
P2	Database Development	135000
P3	CAD/CAM	250000
P4	Maintenance	310000

Giả sử dự án P1 đã kết thúc và ta muốn xoá nó khỏi quan hệ PROJ. Tuy nhiên các bản ghi một và hai của EMP tham chiếu đến dự án P1, vì thế ta không thể xoá dự án P1 được.

Tuy nhiên, qui tắc toàn vẹn tham chiếu cho phép các phương án lựa chọn sau:

1) *Restrict*: Không cho xoá.
2) *Nullify*: Gán giá trị *null* cho thuộc tính cặp ghép của các bản ghi của quan hệ tham chiếu tham chiếu đến bản ghi bị xoá.
3) *Cascade*: Xoá tất cả các bản ghi của quan hệ tham chiếu tham chiếu đến bản ghi bị xoá.

3.4. Qui tắc thao tác bẫy

Qui tắc thao tác bẫy là qui tắc yêu cầu tính hợp pháp của dữ liệu trong các tác nghiệp cập nhật như xoá, chèn và sửa.

Thao tác bẫy có thể liên quan đến các thuộc tính của một quan hệ hoặc nhiều quan hệ. Các ràng buộc phức tạp thường được phát biểu dạng thao tác bẫy.

Một thao tác bẫy thường có các thành phần sau:

1) *Qui tắc người dùng*: là yêu cầu ngắn gọn của ràng buộc.
2) *Sự kiện*: là các thao tác xử lý dữ liệu (chèn, sửa hoặc xoá) kích hoạt thao tác bẫy.
3) *Tên quan hệ*: tên các quan hệ liên quan.
4) *Điều kiện*: là các lý do dẫn đến việc kích hoạt thao tác bẫy.
5) *Hành động*: là công việc thực thi khi thao tác bẫy được kích hoạt.

◊ *Ví dụ*. Cho quan hệ

NHANVIEN(*Manv, HoTen, NgaySinh, NgayBC, ...*).

Hiển nhiên là *NgayBC* (ngày vào biên chế) không được sớm hơn *NgaySinh*. Ta có thể đảm bảo điều kiện này bằng thao tác bẫy sau:

Qui tắc người dùng: *NgayBC* không sớm hơn *NgaySinh.*
Sự kiện: Chèn, Sửa.
Tên quan hệ: NHANVIEN
Điều kiện: *NgayBC < NgaySinh.*
Hành động: Phủ nhận thao tác cập nhật.

◊ *Ví dụ*. Xét hai quan hệ

KHACH(*Makhach, TenKhach, TaiKhoan, SoDu*)
THANHTOAN(*MaKhach, SoTien*)

Ta thấy rằng *SoTien* của THANHTOAN không thể vượt quá *SoDu* của KHACH. Ta có thể đảm bảo điều kiện này bằng thao tác bẫy sau:

Qui tắc người dùng: *SoTien* không lớn hơn *SoDu*.
Sự kiện: Chèn, Sửa.
Tên quan hệ: THANHTOAN, KHACH
Điều kiện: THANHTOAN.*SoTien* > KHACH.*SoDu*
Hành động: Phủ nhận thao tác cập nhật.

4. Các ngôn ngữ dữ liệu quan hệ

Các ngôn ngữ thao tác dữ liệu được phát triển cho mô hình quan hệ (thường gọi là *ngôn ngữ vấn tin*, query language) được chia làm hai nhóm căn bản: Các ngôn ngữ dựa trên đại số quan hệ (relational algebra) và các ngôn ngữ dựa trên phép tính quan hệ (relational calculus). Khác biệt giữa chúng là cách thức người sử dụng đưa ra câu vấn tin. Đại số quan hệ thuộc loại thủ tục (procedural), trong đó người dùng cần phải đặc tả, nhờ một số toán tử, bằng cách nào đạt được kết quả. Ngược lại phép tính quan hệ thuộc loại phi thủ tục (nonprocedural), người dùng chỉ cần đặc tả các mối liên hệ cần phải đảm bảo trong kết quả. Cả hai ngôn ngữ được Codd đưa ra năm 1970 và ông đã chứng minh rằng chúng tương đương về khả năng diễn tả.

4.1. Đại số quan hệ

Đại số quan hệ có một tập các phép toán trên các quan hệ. Chúng có nguồn gốc từ lý thuyết tập hợp (mỗi quan hệ thực chất là một tập hợp). Mỗi toán tử nhận một hoặc hai quan hệ làm toán hạng và cho ra một quan hệ mới (quan hệ kết quả), đến lượt nó quan hệ kết quả có thể dùng làm toán hạng cho một toán tử khác. Những phép toán này cho phép vấn tin và cập nhật cơ sở dữ liệu quan hệ.

Có năm phép toán đại số cơ bản và năm phép toán khác có thể định nghĩa theo các phép toán cơ bản.

Các phép toán cơ bản là: *phép chọn, phép chiếu, phép hợp, phép hiệu* và *tích Descartes*. Hai phép toán đầu thuộc loại một ngôi, và ba phép toán sau thuộc loại hai ngôi.

Các phép toán bổ sung có thể định nghĩa bởi các phép toán cơ bản là: *phép giao, phép nối, phép nối tự nhiên, phép bán nối*, và *phép thương*.

Trong thực hành đại số quan hệ được mở rộng để có thể nhóm hoặc sắp xếp kết quả và có thể thực hiện các phép gộp phần (aggregation) hoặc các phép tính số học. Một số phép toán khác như nối ngoài (outer join) cũng được hỗ trợ.

Các toán hạng của một số phép toán hai ngôi phải *ứng hợp* (*union compartible*), tức là chúng cùng bậc (cùng số thuộc tính) và các thuộc tính tương ứng có cùng miền giá trị.

a. Phép chiếu

Cho quan hệ **r** với lược đồ quan hệ $R=(A_1, ..., A_n)$. Cho S là lược đồ con của R, $S \subset R$, S có m thuộc tính ($m < n$). Chiếu của **r** lên lược đồ S, ký hiệu $\pi_S(\mathbf{r})$, được định nghĩa là quan hệ với lược đồ S gồm các m-bộ u sao cho tồn tại n-bộ $v \in \mathbf{r}$ thoả mãn

$$v(S) = u$$

tức là

$$\pi_S(\mathbf{r}) = \{\text{m-bộ } u : \exists\, v \in \mathbf{r}, v(S) = u \}$$

◊ *Ví dụ*

Cho quan hệ **r** với các thuộc tính A,B,C. Sau đây là ví dụ cụ thể về chiếu của **r** lên hai thuộc tính A và C.

r

A	**B**	**C**
a	b	c
d	a	f
c	b	d
c	f	d

⇒

$\pi_{A,C}(\mathbf{r})$

A	**C**
a	c
d	f
c	d

◊ *Ví dụ*: Xét quan hệ PROJ

PROJ

PNO	*PNAME*	*BUDGET*
P1	Instrumentation	150000
P2	Database Development	135000
P3	CAD/CAM	250000
P4	Maintenance	310000

Chiếu của PROJ lên các thuộc tính PNO và BUDGET có kết quả như sau

$\pi_{PNO,BUDGET}$(PROJ)

PNO	*BUDGET*
P1	150000
P2	135000
P3	250000
P4	310000

b. Phép chọn

Cho quan hệ **r** với lược đồ quan hệ $R=(A_1,..., A_n)$ và F là biểu thức lôgic bao gồm

(i) Các toán hạng là hằng hoặc (số hiệu) thành phần,

(ii) Các phép toán so sánh : <, =, >, ≠ , ≤, ≥

(iii) Các phép toán logic : &, ∧ (và), ∨ (hoặc) , ¬ (phủ định).

Phép chọn của **r** theo biểu thức F, ký hiệu $\sigma_F(\mathbf{r})$, là quan hệ với lược đồ R gồm tất cả các bộ t trong **r** sao cho khi thay các thành phần của t vào biểu thức F thì ta có giá trị đúng. Tức là

$$\sigma_F(\mathbf{r}) = \{ t \in \mathbf{r} : F(t) = true \}$$

◊ *Ví dụ:* Cho quan hệ **r** với các thuộc tính A,B,C. Sau đây là ví dụ cụ thể về chọn của **r** theo biểu thức B=b

r

A	**B**	**C**
a	b	c

⇒

$\sigma_{B=b}(\mathbf{r})$

A	**B**	**C**
a	b	c

d	a	f		c	b	d
c	b	d			
c	f	d			

◊ *Ví dụ*: Xét quan hệ EMP

EMP

ENO	*ENAME*	*TITLE*	*SAL*	*PNO*	*RESP*	*DUR*
E1	J.Doe	Elect.Eng.	40000	P1	Manager	12
E2	M.Smith	Syst.Anal.	34000	P1	Analyst	24
E2	M.Smith	Syst.Anal.	34000	P2	Analyst	6
E3	A.Lee	Mech.Eng.	27000	P3	Consultant	10
E3	A.Lee	Mech.Eng.	27000	P4	Engineer	48
E4	J.Miller	Programmer	24000	P2	Programmer	18
E5	B.Casey	Syst.Anal.	34000	P2	Manager	24
E6	L.Chu	Elect.Eng.	40000	P4	Manager	48
E7	R.David	Mech.Eng.	27000	P3	Engineer	36
E8	J.Jones	Syst.Anal.	34000	P3	Manager	40

Các bộ của các kỹ sư điện (electrical engineer) được biểu diễn bằng phép chọn như sau

$\sigma_{TITLE = \text{'Elect. Eng.'}}(EMP)$

ENO	*ENAME*	*TITLE*	*SAL*	*PNO*	*RESP*	*DUR*
E1	J.Doe	Elect.Eng.	40000	P1	Manager	12
E6	L.Chu	Elect.Eng.	40000	P4	Manager	48

c. Phép hợp

Cho quan hệ **r**, **s** với lược đồ quan hệ R=(A_1,..., A_n).

Hợp của **r** và **s**, ký hiệu **r** ∪ **s** , là quan hệ với lược đồ R gồm tất cả các bộ thuộc **r** hoặc thuộc **s**.

◊ *Ví dụ*

Cho quan hệ **r** và **s** với các thuộc tính A,B,C . Sau đây là ví dụ cụ thể về hợp của **r** và **s**.

r

A	**B**	**C**
a	b	c
d	a	f
c	b	d

s

A	**B**	**C**
b	g	a
d	a	f

⇒

r ∪ **s**

A	**B**	**C**
a	b	c
d	a	f
c	b	d
b	g	a

d. Phép hiệu

Cho quan hệ **r**, **s** với lược đồ quan hệ R=(A_1,..., A_n).

Hiệu của **r** và **s**, ký hiệu **r** − **s**, là quan hệ với lược đồ R gồm tất cả các bộ thuộc **r** nhưng không thuộc **s**.

◊ *Ví dụ*

Cho quan hệ **r** và **s** với các thuộc tính A,B,C . Sau đây là ví dụ cụ thể về hiệu của **r** và **s**.

r

A	B	C
a	b	c
d	a	f
c	b	d

s

A	B	C
b	g	a
d	a	f

⇒ **r − s**

A	B	C
a	b	c
c	b	d

e. Tích Đề-các

Cho quan hệ **r** với lược đồ quan hệ $R=(A_1, ..., A_n)$ và quan hệ **s** với lược đồ quan hệ $S=(B_1,..., B_m)$.

Tích Đề các của **r** và **s**, ký hiệu **r** × **s**, là quan hệ với lược đồ $=(A_1, ..., A_n, B_1,..., B_m)$ gồm tất cả các (n+m)-bộ, trong đó n thành phần đầu là bộ thuộc **r** và m thành phần sau là bộ thuộc **s**.

◊ *Ví dụ*

Cho quan hệ **r** với các thuộc tính A,B,C và **s** với các thuộc tính D,E,F. Sau đây là ví dụ cụ thể về tích Đề-các của **r** và **s**.

r

A	B	C
a	b	c
d	a	f
c	b	d

s

D	E	F
b	g	a
d	a	f

⇒ **r × s**

A	B	C	D	E	F
a	b	c	b	g	a
a	b	c	d	a	f
d	a	f	b	g	a
d	a	f	d	a	f
c	b	d	b	g	a
c	b	d	d	a	f

f. Phép giao

Cho quan hệ **r**, **s** với lược đồ quan hệ $R=(A_1,..., A_n)$.

Giao của **r** và **s**, ký hiệu **r** ∩ **s**, là quan hệ với lược đồ R gồm tất cả các bộ thuộc **r** và thuộc **s**.

◆ *Công thức*. Phép giao suy ra từ phép hiệu

$$\mathbf{r} \cap \mathbf{s} = \mathbf{r} - (\mathbf{r} - \mathbf{s}) = \mathbf{s} - (\mathbf{s} - \mathbf{r})$$

◊ *Ví dụ*

Cho quan hệ **r** và **s** với các thuộc tính A,B,C . Sau đây là ví dụ cụ thể về giao của **r** và **s**.

r

A	B	C
a	b	c
d	a	f
c	b	d

s

A	B	C
b	g	a
d	a	f

⇒ **r ∩ s**

A	B	C
d	a	f

g. Phép nối

Cho quan hệ **r** với lược đồ quan hệ $R=(A_1, ..., A_n)$ và quan hệ **s** với lược đồ quan hệ $S=(B_1, ..., B_m)$.

Cho biểu thức lôgic F. Phép *nối* của quan hệ **r** và quan hệ **s** theo điều kiện nối F, ký hiệu

$$\mathbf{r} \underset{F}{><} \mathbf{s}$$

là quan hệ với lược đồ $=(A_1, ..., A_n, B_1, ..., B_m)$ gồm tất cả các (n+m)-bộ (t,u) thoả $t \in \mathbf{r}$, $u \in \mathbf{s}$ và khi thế các giá trị của *t* và u vào F thì ta được giá trị đúng.

Ở đây F là biểu thức lôgic gồm các toán hạng dạng $A\theta B$, trong đó A là thuộc tính của **r** và B là thuộc tính **s** và θ là phép toán so sánh.

◆ *Công thức.* Ta có thể biểu diễn

$$\mathbf{r} \underset{F}{><} \mathbf{s} = \sigma_F(\mathbf{r} \times \mathbf{s})$$

◆ *Phép đẳng nối*

Nếu các toán tử so sánh trong biểu thức F đều là phép bằng (=), thì phép nối theo F được gọi là phép *đẳng nối* (*equijoin*).

◊ *Ví dụ*

Cho quan hệ **r** với các thuộc tính A,B,C và **s** với các thuộc tính D,E. Sau đây là ví dụ cụ thể về phép nối và đẳng nối của **r** và **s**.

r

A	B	C
1	2	3
4	5	6
7	8	9

s

D	E
3	1
6	2

$\Rightarrow$ $\mathbf{r} \underset{B<D}{><} \mathbf{s}$

A	B	C	D	E
1	2	3	3	1
1	2	3	6	2
4	5	6	6	2

r

A	B	C
1	2	3
4	5	6
7	8	9

s

D	E
3	1
6	2

$\Rightarrow$ $\mathbf{r} \underset{C=D}{><} \mathbf{s}$

A	B	C	D	E
1	2	3	3	1
4	5	6	6	2

◊ *Ví dụ.* Xét các quan hệ

EMP(*ENO, ENAME, TITLE*)

ENO	*ENAME*	*TITLE*
E1	J.Doe	Elect.Eng.
E2	M.Smith	Syst.Anal.
E3	A.Lee	Mech.Eng.
E4	J.Miller	Programmer
E5	B.Casey	Syst.Anal.
E6	L.Chu	Elect.Eng.
E7	R.David	Mech.Eng.
E8	J.Jones	Syst.Anal.

PAY(*TITLE, SAL*)

TITLE	*SAL*
Elect.Eng.	40000
Syst.Anal.	34000
Mech.Eng.	27000
Programmer	24000
Operator	15000

với ràng buộc toàn vẹn tham chiếu EMP.*TITLE*→PAY.*TITLE*.

Phép nối EMP với PAY theo EMP.*TITLE*=PAY.*TITLE* có kết quả sau

EMP >< PAY
EMP.*TITLE*=PAY.*TITLE*

ENO	*ENAME*	*EMP.TITLE*	*PAY.TITLE*	*SAL*
E1	J.Doe	Elect.Eng.	Elect.Eng.	40000
E2	M.Smith	Syst.Anal.	Syst.Anal.	34000
E3	A.Lee	Mech.Eng.	Mech.Eng.	27000
E4	J.Miller	Programmer	Programmer	24000
E5	B.Casey	Syst.Anal.	Syst.Anal.	34000
E6	L.Chu	Elect.Eng.	Elect.Eng.	40000
E7	R.David	Mech.Eng.	Mech.Eng.	27000
E8	J.Jones	Syst.Anal.	Syst.Anal.	34000

h. Phép nối tự nhiên

Nối tự nhiên giữa 2 quan hệ **r** và **s**, ký hiệu là **r** >< **s**, là phép đẳng nối trên các thuộc tính cụ thể có cùng miền giá trị. Tuy nhiên khác với đẳng nối là các thuộc tính dùng để nối tự nhiên chỉ xuất hiện một lần trong bảng kết quả.

◆ *Cách tính*.

Cho các quan hệ **r** và **s** với các cột được đặt tên theo thuộc tính. *Nối tự nhiên* **r** >< **s** được tính như sau:

(i) Tính tích Đề-các **r** × **s**.

(ii) Với mỗi thuộc tính A có cả trong **r** và **s**, chọn các bộ trong **r** × **s** có R.A = S.A, trong đó R.A (S.A) là tên cột của R × S tương ứng với cột A của R (S).

(iii) Với mỗi thuộc tính A như trên ta loại bỏ cột S.A.

Một cách hình thức, giả sử A_1, A_2, ... , A_k là tên các thuộc tính dùng chung cho cả R và S. Gọi i_1, i_2, ... , i_m là danh sách các thành phần của R × S , trừ các thuộc tính $S.A_1$, $S.A_2$, ..., $S.A_k$. Khi đó ta có thể biểu diễn nối tự nhiên bằng công thức sau.

◆ *Công thức*

$$\mathbf{r} >< \mathbf{s} = \pi_{i_1, i_2, ..., i_m} \sigma_{R.A_1=S.A_1 \wedge ... \wedge R.A_k=S.A_k} (\mathbf{r} \times \mathbf{s})$$

◊ *Ví dụ*

Cho quan hệ **r** với các thuộc tính A,B,C và **s** với các thuộc tính B,C,D. Sau đây là ví dụ cụ thể về phép nối tự nhiên của **r** và **s**.

r			**s**				**r** >< **s**			
A	**B**	**C**	**B**	**C**	**D**	⇒	**A**	**B**	**C**	**D**
a	b	c	b	c	d		a	b	c	d
d	b	c	b	c	e		a	b	c	e
b	b	f	a	d	b		d	b	c	d

c	a	d		d	b	c	e
				c	a	d	b

◊ *Ví dụ.* Các quan hệ

EMP(*ENO, ENAME, TITLE*)

PAY(*TITLE, SAL*)

cho ở ví dụ trước, có thuộc tính chung là *TITLE*.

Nối tự nhiên của hai quan hệ cho ở bảng sau

EMP >< PAY

ENO	*ENAME*	*TITLE*	*SAL*
E1	J.Doe	Elect.Eng.	40000
E2	M.Smith	Syst.Anal.	34000
E3	A.Lee	Mech.Eng.	27000
E4	J.Miller	Programmer	24000
E5	B.Casey	Syst.Anal.	34000
E6	L.Chu	Elect.Eng.	40000
E7	R.David	Mech.Eng.	27000
E8	J.Jones	Syst.Anal.	34000

i. Phép bán nối

Cho quan hệ **r** với lược đồ quan hệ $R=(A_1,..., A_n)$ và quan hệ **s** với lược đồ quan hệ $S=(B_1, ..., B_m)$.

Ký hiệu θ là toán tử so sánh (<, =, >, ≠ , ≤, ≥). Cho F là biểu thức lôgic gồm các toán hạng dạng AθB, trong đó A là thuộc tính của **r** và B là thuộc tính **s**.

Phép *bán nối* của quan hệ **r** và quan hệ **s** theo điều kiện nối F, ký hiệu

$$\mathbf{r} \underset{F}{><} \mathbf{s}$$

là quan hệ với lược đồ R gồm các bộ của **r** có tham gia vào nối của **r** và **s** theo F**.**

◆ *Công thức.* Ta có thể biểu diễn

$$\mathbf{r} \underset{F}{><} \mathbf{s} = \pi_R(\mathbf{r} \underset{F}{><} \mathbf{s})$$

Ưu điểm của phép bán nối là giảm số lượng các bộ cần xử lý để thực hiện nối. Trong hệ cơ sở dữ liệu phân tán điều này có ý nghĩa rất quan trọng vì nó làm giảm số lượng dữ liệu cần truyền giữa các vị trí để ước lượng câu vấn tin.

◆*Bán nối tự nhiên*

$$\mathbf{r} >< \mathbf{s} = \pi_R(\mathbf{r} >< \mathbf{s})$$

◊ *Ví dụ.* Cho quan hệ **r** với các thuộc tính A,B,C và **s** với các thuộc tính D,E. Sau đây là ví dụ cụ thể về phép bán nối của **r** và **s**.

r			**s**			$\mathbf{r} \underset{B<D}{><} \mathbf{s}$		
A	**B**	**C**	**D**	**E**	⇒	**A**	**B**	**C**
1	2	3	3	1		1	2	3

4	5	6		6	2		4	5	6
7	8	9							

r **s** **r** $\overset{A=E}{><}$ **s**

A	**B**	**C**		**D**	**E**	⇒	**A**	**B**	**C**
1	2	3		3	1		1	2	3
4	5	6		6	2				
7	8	9							

◊ *Ví dụ*. Xét các quan hệ

EMP(*ENO, ENAME, TITLE*)

PAY(*TITLE, SAL*)

ở ví dụ trước. Phép bán nối của hai quan hệ theo EMP.*TITLE*=PAY.*TITLE* có kết quả sau

EMP >< PAY

EMP.*TITLE*=PAY.*TITLE*

ENO	*ENAME*	*TITLE*
E1	J.Doe	Elect.Eng.
E2	M.Smith	Syst.Anal.
E3	A.Lee	Mech.Eng.
E4	J.Miller	Programmer
E5	B.Casey	Syst.Anal.
E6	L.Chu	Elect.Eng.
E7	R.David	Mech.Eng.
E8	J.Jones	Syst.Anal.

PAY >< EMP

EMP.*TITLE*=PAY.*TITLE*

TITLE	*SAL*
Elect.Eng.	40000
Syst.Anal.	34000
Mech.Eng.	27000
Programmer	24000

j. Phép chia (thương)

Cho quan hệ **r** với lược đồ quan hệ $R=(A_1, ..., A_n)$ và quan hệ **s** với lược đồ quan hệ $S=(A_1, ..., A_m)$, trong đó $m < n$.

Ta định nghĩa *phép chia* **r** ÷ **s** là quan hệ với lược đồ $(A_{m+1}, ..., A_n)$ gồm tất cả (n-m)-bộ *v* sao cho với mọi m-bộ *u* thuộc **s**, bộ (*u*,*v*) thuộc **r**.

Để lập công thức cho phép chia ta ký hiệu

$$\mathbf{t} = \pi_{A_{m+1},...,A_n}(r)$$

Khi đó

$$(\mathbf{s} \times \mathbf{t}) - \mathbf{r}$$

là tập hợp các n-bộ không thuộc **r**, tạo ra bằng cách lấy các phần tử của **s** kết hợp với n-m thành phần của các phần tử thuộc **r**. Đặt

$$\mathbf{q} = \pi_{A_{m+1},\ldots,A_n}\left((s \times t) - r\right)$$

Như vậy **q** là tập tất cả các (n−m)-bộ *v* gồm n−m thành phần cuối của các phần tử của **r** và tồn tại phần tử $u \in \mathbf{s}$ sao cho bộ (*u*,*v*) không thuộc **r**. Suy ra

◆ *Công thức*

$$\mathbf{r} \div \mathbf{s} = \mathbf{t} - \pi_{A_{m+1},\ldots,A_n}\left((s \times t) - r\right)$$

◊ *Ví dụ*

Cho quan hệ **r** với các thuộc tính A,B,C,D và **s** với các thuộc tính C,D. Sau đây là ví dụ cụ thể về phép chia của **r** cho **s**.

r				**s**			**r ÷ s**	
A	**B**	**C**	**D**	**C**	**D**	⇒	**A**	**B**
a	b	c	d	c	d		a	b
a	b	e	f	e	f		e	d
b	c	e	f					
e	d	c	d					
e	d	e	f					
a	b	d	e					

◊ *Ví dụ*. Xét các quan hệ sau:

ASG'

ENO	*PNO*	*PNAME*	*BUDGET*
E1	P1	Instrumentation	150000
E2	P1	Instrumentation	150000
E2	P2	Database Deve.	135000
E3	P3	CAD/CAM	250000
E3	P4	Maintenance	310000
E4	P2	Database Deve.	135000
E5	P2	Database Deve.	135000
E6	P4	Maintenance	310000
E7	P3	CAD/CAM	250000
E8	P3	CAD/CAM	250000

PROJ'

PNO	*PNAME*	*BUDGET*
P3	CAD/CAM	250000
P4	Maintenance	310000

Để tìm mã số nhân viên của những nhân viên được phân vào tất cả dự án trong PROJ', ta phải thực hiện phép chia ASG' cho PROJ' và được kết quả là

ASG' ÷ PROJ'

ENO
E3

· ***Các chương trình đại số quan hệ***

Vì tất cả các phép toán đại số đều nhận quan hệ làm đối biến và sinh ra các quan hệ kết quả, chúng ta có thể lồng ghép các phép toán này bằng các dấu ngoặc đơn và sinh ra các *chương trình đại số quan hệ*.

Dưới đây là một số ví dụ minh hoạ sử dụng các quan hệ

EMP(*ENO, ENAME, TITLE*)
PAY(*TITLE, SAL*)
PROJ(*PNO, PNAME, BUDGET*)
ASG(*ENO, PNO, RESP, DUR*)

◊ *Ví dụ*. Sau đây là chương trình tìm tên tất cả nhân viên đang làm việc cho dự án CAD/CAM

$$\pi_{ENAME}(((\sigma_{PNAME='CAD/CAM'}(PROJ)) >< ASG) >< EMP)$$

Thứ tự thực hiện như sau: thực hiện phép chọn trên PROJ, sau đó nối tự nhiên với ASG, theo sau là nối tự nhiên với EMP, và cuối cùng là chiếu lên ENAME.

Một chương trình tương đương nhưng kích thước quan hệ trung gian nhỏ hơn là

$$\pi_{ENAME}(EMP >< (\pi_{ENO}(ASG >< (\sigma_{PNAME='CAD/CAM'}(PROJ))))$$

◊ *Ví dụ*. Sau đây là chương trình tăng lương tất cả các lập trình viên (programmer) lên 25000 USD.

$$(PAY - (\sigma_{TITLE='Programmer'}(PAY)) \cup (<'Programmer',25000>)$$

4.2. Đại số quan hệ

Trong các ngôn ngữ dựa trên phép tính quan hệ, thay vì xác định xem phải làm thế nào để thu được kết quả, chúng ta sẽ xác định xem kết quả là gì bằng cách đưa ra mối liên hệ được giả sử là đúng đối với kết quả.

Ngôn ngữ phép tính quan hệ được phân làm 2 nhóm: *phép tính quan hệ bộ (tuple relational calculus)* và *phép tính quan hệ miền (domain relational calculus)*. Sự khác biệt giữa chúng là ở các biến nguyên thuỷ được dùng khi xác định các câu vấn tin.

Ngôn ngữ phép tính quan hệ có cơ sở lý thuyết vững chắc bởi vì chúng xây dựng trên logic vị từ bậc nhất. Ngữ nghĩa được gán cho các công thức bằng cách diễn giải chúng như các phán đoán trên cơ sở dữ liệu. Một cơ sở dữ liệu quan hệ có thể xem như tập các bộ hoặc tập các miền. Phép tính quan hệ bộ diễn giải các biến trong công thức như một bộ của quan hệ, còn phép tính quan hệ miền diễn giải biến như giá trị của miền.

a) Phép tính quan hệ bộ (Codd 1970)

Biến nguyên thuỷ dùng trong phép tính quan hệ bộ là *biến bộ* (tuple variable), biểu thị một bộ của quan hệ. Nói cách khác biến này biến thiên trên các bộ của quan hệ.

Trong phép tính quan hệ bộ, câu vấn tin được đặc tả là

$$\{t \mid F(t)\}$$

trong đó t là biến bộ và F là công thức chỉnh dạng. Công thức nguyên tử có hai dạng:

(1) *Biểu thức kiểu phần tử biến bộ.*

Nếu t là một biến bộ biến thiên trên các bộ của một quan hệ R, biểu thức "*bộ t thuộc quan hệ R*" là công thức nguyên tử, và được viết là $R.t$ hoặc $R(t)$.

(2) *Điều kiện.*

Loại công thức nguyên tử này có thể định nghĩa như sau:

(i) $s[A]\ \theta\ t[B]$, trong đó s và t là các biến bộ và A và B là các thành phần tương ứng của s và t , θ là một trong các toán tử so sánh $<, >, =, \leq, \geq$ và $\neq$.

(ii) $s[A]\ \theta\ c$, trong đó s , A và θ định nghĩa giống như trên và c là hằng.

Hiện có nhiều ngôn ngữ dựa trên phép tính quan hệ bộ, và ngôn ngữ thông dụng nhất là SQL và QUEL. SQL hiện là chuẩn quốc tế (duy nhất) với các phiên bản chuẩn hoá đã được đưa ra năm 1986 (SQL1), năm 1992 (SQL2) và năm 1998 (SQL3).

· ***Ngôn ngữ truy vấn SQL***

SQL thuộc loại *ngôn ngữ thế hệ thứ tư (4GL)* được nghiên cứu nhiều năm và trở thành tiêu chuẩn quốc tế về kiểm soát dữ liệu. SQL kế thừa *tính phi thủ tục* của 4GL: Xử lý đồng thời hàng loạt câu lệnh. Người dùng chỉ cần nêu ra yêu cầu về dữ liệu mà không cần biết máy tính xử lý bên trong như thế nào. Người dùng có thể truy xuất nhanh chóng với những CSDL lớn, yêu cầu những xử lý phức tạp tinh vi mà không cần lập trình.

SQL là *ngôn ngữ có cấu trúc*. Trong câu lệnh của SQL có một số mệnh đề tuân theo những cú pháp riêng của nó. Có 4 loại lệnh trong SQL :

- Các lệnh truy vấn dữ liệu.
- Các lệnh định nghĩa dữ liệu (DDL).
- Các lệnh xử lý cập nhật dữ liệu (DML).
- Các lệnh kiểm soát dữ liệu.

◆ Các lệnh truy vấn dữ liệu, gọi là câu vấn tin có cú pháp tổng quát như sau

```
SELECT [DISTINCT] <biểu thức 1> AS <tên 1>] [,...] | *
FROM  <bảng 1> [<bí danh 1>] [,...]
[INTO <dbf đích>]
[WHERE  <điều kiện nối >  [AND | OR <điều kiện lọc>]]
[GROUP BY  <cột nhóm 1> [,...]
   [HAVING <điều kiện nhóm>]]
[ORDER BY <biểu thức sắp xếp 1> [ASC | DESC] [,...]]
[UNION | INTERSECT | MINUS  < câu truy vấn khác>]
```

Dưới đây là một số ví dụ minh họa sử dụng các quan hệ

EMP(*ENO, ENAME, TITLE*)
PAY(*TITLE, SAL*)
PROJ(*PNO, PNAME, BUDGET*)
ASG(*ENO, PNO, RESP, DUR*)

◊ *Ví dụ*. Tìm tên tất cả nhân viên đang làm việc cho dự án CAD/CAM

```
Select   EMP.ENAME
From     EMP, ASG, PROJ
Where   (EMP.ENO = ASG.ENO)
         AND (ASG.PNO = PROJ.PNO)
         AND (PROJ.PNAME = "CAD/CAM")
```

◊ *Ví dụ*. Tìm tên tất cả nhân viên đang quản lý dự án (Manager)

```
SELECT      ENAME
FROM        EMP, ASG
WHERE       (EMP.ENO = ASG.ENO)
        AND    (RESP = "Manager")
```

◊ *Ví dụ*. Tìm tên tất cả nhân viên đang làm việc trong dự án P3 và P4. (bài tập)

◆ Các lệnh cập nhật dữ liệu gồm có lệnh UPDATE (hiệu chỉnh), INSERT (thêm) và DELETE (xoá).

◊ *Ví dụ.* Tăng lương các lập trình viên (programmer) lên 25000 USD.

```
UPDATE  PAY
SET     SAL = 25000
WHERE   PAY.TITLE = "Programmer"
```

◊ *Ví dụ.* Thêm nhân viên mới vào EMP

```
INSERT INTO EMP
   VALUE ('E10', 'John Smith', 'Programmer')
```

◊ *Ví dụ.* Xoá dự án 'P1'

```
DELETE FROM PROJ
   WHERE  PNO = 'P1'
```

b) Phép tính quan hệ miền (Lacroix, Pirotte 1977)

Biến nguyên thuỷ dùng trong phép tính quan hệ miền là *biến miền* (*domain variable*), xác định một thành phần của bộ biến thiên trong tập giá trị của miền. Nói cách khác, miền xác định của biến miền bao gồm các miền trên đó quan hệ được định nghĩa. Câu vấn tin có dạng sau:

$$x_1, \ldots, x_n \mid F(x_1, \ldots, x_n)$$

trong đó F là công thức chỉnh dạng còn $x_1, \ldots, x_n$ là các biến tự do.

Thành công của ngôn ngữ phép tính quan hệ miền chủ yếu do QBE (Zloof, 1977) đem lại. Đây là ứng dụng kiểu trực quan của phép tính miền. QBE (Query by example) được thiết kế dành cho kiểu làm việc tương tác từ thiết bị đầu cuối trực quan và thân thiện.

Khái niệm cơ bản là *example*: người sử dụng đưa ra các câu vấn tin bằng cách cung cấp một example có thể có của câu trả lời. Hành động gõ tên quan hệ sẽ kích hoạt việc hiển thị các lược đồ của chúng lên màn hình. Sau đó bằng cách cung cấp các từ khoá trong các cột (miền), người dùng đặc tả câu vấn tin.

Chẳng hạn các thuộc tính của quan hệ chiếu được cho bằng từ P (Project).

Theo mặc định tất cả các câu vấn tin đều là kiểu truy xuất. Câu vấn tin cập nhật đòi hỏi phải có đặc tả U dưới tên quan hệ cần cập nhật.

◊ *Ví dụ.* Tìm tên tất cả nhân viên đang làm việc cho dự án CAD/CAM

EMP	*ENO*	*ENAME*	*TITLE*
	E2	P	

ASG	*ENO*	*PNO*	*RESP*	*DUR*
	E2	P3		

PROJ	*PNO*	*PNAME*	*BUDGET*

P3	CAD/CAM	

◊ *Ví dụ.* Tăng lương các lập trình viên (programmer) lên 25000 USD.

PAY	TITLE	SAL
	Programmer	U.25000

5. Thiết kế cơ sở dữ liệu quan hệ

5.1. Dư thừa dữ liệu

Khi thiết kế cơ sở dữ liệu quan hệ ta thường đứng trước vấn đề lựa chọn giữa các lược đồ quan hệ: lược đồ nào tốt hơn ? Tại sao ? Mục này sẽ nghiên cứu một số tiêu chuẩn đánh giá lược đồ quan hệ và các thuật toán giúp chúng ta xây dựng được lược đồ cơ sở dữ liệu quan hệ có cấu trúc tốt.

Có thể nói tổng quát một lược đồ quan hệ có *cấu trúc tốt* là lược đồ không chứa đựng sự *dư thừa dữ liệu*, tức là là sự trùng lặp thông tin trong cơ sở dữ liệu.

5.1.1. Sự dư thừa dữ liệu

Dư thừa dữ liệu là sự trùng lặp thông tin trong cơ sở dữ liệu.

◊ *Ví dụ*

Xét quan hệ EMP(*ENO*, *ENAME, TITLE, SAL*, *PNO*, *RESP*, *DUR*). Nếu một nhân viên tham gia trong nhiều dự án, thì các dữ liệu như *ENAME, TITLE, SAL* phải lặp lại nhiều lần và kéo theo dư thừa dữ liệu.

Ngoài việc gây lãng phí dung lượng lưu trữ, sự dư thừa dữ liệu có thể gây ra những hậu quả nghiêm trọng đối với dữ liệu khi người dùng cập nhật dữ liệu làm cho dữ liệu không tương thích, bất định hoặc mất mát. Các sự cố như vậy gọi là những *dị thường*.

5.1.2. Các dị thường cập nhật dữ liệu

Ta sẽ minh hoạ các dị thường bằng các lược đồ

EMP(*ENO*, *ENAME, TITLE, SAL*, *PNO*, *RESP, DUR*)
PROJ(*PNO*, *PNAME, BUDGET*)

a. Dị thường do dữ liệu lặp: Một số thông tin có thể được lặp lại một cách vô ích.

◊ *Ví dụ*: Trong quan hệ EMP tên (*ENAME*), chức vụ (*TITLE*), và lương (*SAL*) của nhân viên được lặp lại trong mỗi dự án mà họ tham gia. Điều này rõ ràng là làm lãng phí chỗ lưu trữ và đối nghịch với các nguyên lý của cơ sở dữ liệu.

b. Dị thường chèn bộ: Không thể chèn bộ mới vào quan hệ, nếu không có đầy đủ dữ liệu.

◊ *Ví dụ*: Xét quan hệ EMP. Giả sử một nhân viên mới được nhận vào công ty và chưa được phân công vào dự án nào cả. Khi đó chúng ta không thể nhập các thông tin về tên, chức vụ, lương của nhân viên này vào quan hệ, vì khoá của EMP là (*ENO, PNO*).

c. Dị thường xoá bộ: Trường hợp này ngược với dị thường chèn bộ. Việc xoá bộ có thể kéo theo mất thông tin.

◊ *Ví dụ*: Xét quan hệ EMP. Giả sử một nhân viên làm việc trong một dự án duy nhất. Khi dự án chấm dứt, chúng ta không thể xoá thông tin về dự án đó trong EMP được, vì nếu làm thế ta sẽ mất luôn thông tin về nhân viên đó.

d. Dị thường sửa bộ: Việc sửa đổi dữ liệu dư thừa có thể dẫn đến sự không tương thích dữ liệu.

◊ *Ví dụ*: Xét quan hệ EMP. Giả sử một nhân viên làm việc trong nhiều dự án. Khi có sự thay đổi về lương, rất nhiều bộ phải cập nhật sự thay đổi này. Điều đó gây lãng phí thời gian công sức và là nguy cơ gây ra sự không thống nhất dữ liệu.

Trong các ví dụ trên ta thấy tác hại của sự dư thừa dữ liệu và sự cần thiết phải loại bỏ chúng khỏi các lược đồ quan hệ. Quá trình từng bước thay thế một lược đồ quan hệ bằng các tập lược đồ quan hệ đơn giản và chuẩn tắc hơn gọi là ***chuẩn hoá***. Mục đích của chuẩn hoá là loại bỏ các dị thường (hoặc các khía cạnh không mong muốn khác) để có những quan hệ tốt hơn.

Cơ sở lý thuyết của việc thiết kế lược đồ cơ sở dữ liệu quan hệ tốt là khái niệm *phụ thuộc dữ liệu*. Phụ thuộc dữ liệu biểu diễn các quan hệ nhân quả giữa các thuộc tính trong quan hệ. Ví dụ trong bảng EMP, thuộc tính SAL phụ thuộc vào thuộc tính ENO, vì mỗi nhân viên chỉ có một lương duy nhất.

Cũng dựa trên khái niệm phụ thuộc dữ liệu người ta định nghĩa các *dạng chuẩn* của lược đồ dữ liệu quan hệ. Mỗi dạng chuẩn đáp ứng một yêu cầu nhất định đối với lược đồ quan hệ.

Quá trình biến đổi một lược đồ thành lược đồ *tương đương* (bảo toàn thông tin và phụ thuộc dữ liệu) thoả mãn dạng chuẩn gọi là quá trình *chuẩn hoá lược đồ quan hệ*.

Khái niệm phụ thuộc dữ liệu sẽ được nghiên cứu chi tiết ở phần sau.

5.2. Cấu trúc phụ thuộc dữ liệu

Có ba dạng phụ thuộc dữ liệu, *phụ thuộc hàm* (*functional dependancy*- FD) , *phụ thuộc đa trị* (*multivalued dependancy* - MVD) và *phụ thuộc chiếu nối* (*projection-join dependancy* - PJD)

a. Phụ thuộc hàm

Cho lược đồ quan hệ R=(A_1, A_2, ..., A_n) và X, Y là các tập con của {A_1, A_2, ..., A_n}. Ta nói rằng X *xác định hàm* Y hay Y *phụ thuộc hàm* X, ký hiệu $X \rightarrow Y$, nếu mọi quan hệ bất kỳ **r** của lược đồ R thoả mãn:

$$\forall u, v \in \mathbf{r} : u(X) = v(X) \Rightarrow u(Y) = v(Y)$$

Cần nhấn mạnh rằng tính chất phụ thuộc hàm phải thoả với mọi quan hệ **r** của lược đồ R. Ta không thể chỉ xét một quan hệ đặc biệt (quan hệ rỗng chẳng hạn) rồi quy nạp cho toàn lược đồ. Nhưng ta có thể phủ nhận phụ thuộc hàm qua một quan hệ cụ thể nào đó.

Phụ thuộc hàm $X \rightarrow Y$ gọi là phụ thuộc hàm *tầm thường* nếu $Y \subset X$ (hiển nhiên là nếu $Y \subset X$ thì theo định nghĩa ta có $X \rightarrow Y$).

Phụ thuộc hàm $X \rightarrow Y$ gọi là phụ thuộc hàm *nguyên tố* nếu không có tập con thực sự $Z \subset X$ thoả $Z \rightarrow Y$.

Tập thuộc tính $K \subset R$ gọi là *khoá* nếu nó xác định hàm tất cả các thuộc tính và $K \rightarrow R$ là phụ thuộc hàm nguyên tố.

◊ *Ví dụ*: Xét quan hệ PROJ. Ta có thể chấp nhận rằng mỗi dự án có tên và kinh phí xác định. Vậy có thể khẳng định

$$PNO \rightarrow (PNAME, BUDGET)$$

Trong quan hệ EMP ta có

$$(ENO, PNO) \rightarrow (ENAME, TITLE, SAL, RESP, DUR)$$
$$ENO \rightarrow (ENAME, TITLE, SAL)$$

Hoàn toàn hợp lý khi chúng ta khẳng định rằng lương của mỗi chức vụ là cố định, do đó sẽ tồn tại phụ thuộc hàm

$$TITLE \rightarrow SAL$$

b. Phụ thuộc đa trị

Cho lược đồ quan hệ $R=(A_1, A_2, ..., A_n)$ và X, Y là các tập con của $\{A_1, A_2, ..., A_n\}$. Ta nói rằng X *xác định đa trị* Y hay Y *phụ thuộc đa trị* vào X, ký hiệu $X \rightarrow\rightarrow Y$, nếu mọi quan hệ bất kỳ **r** của lược đồ R thoả mãn:

Ứng với mỗi giá trị của miền giá trị các thuộc tính trong X, có một tập giá trị các thuộc tính trong Y liên quan và tập này độc lập với các thuộc tính trong $Z=R\backslash(X \cup Y)$, tức là:

$$\forall x \in D(X)\ \forall y, y' \in D(Y)\ \forall z, z' \in D(Z): (x,y,z), (x,y',z') \in \mathbf{r} \Rightarrow (x,y,z'), (x,y',z) \in \mathbf{r}$$

với D(X), D(Y) và D(Z) là miền giá trị của X, Y và Z.

· Chú ý rằng *phụ thuộc hàm là trường hợp riêng của phụ thuộc đa trị*, tức là

$$X \rightarrow Y \quad \Rightarrow \quad X \rightarrow\rightarrow Y$$

Thật vậy, nếu

$$(x,y,z), (x, y', z') \in \mathbf{r} \text{ và } X \rightarrow Y$$

thì y=y', và kéo theo

$$(x, y, z'), (x, y', z) \in \mathbf{r}$$

◊ *Ví dụ*

Trở lại ví dụ đang xét. Giả sử ta muốn duy trì thông tin về tập nhân viên và về tập dự án có liên quan đến công ty cũng như về chi nhánh (*PLACE*) thực hiện dự án. Yêu cầu này có thể được thực hiện bằng cách định nghĩa quan hệ

SKILL**(***ENO, PNO, PLACE***)**

Ta giả sử (có thể không thực tế cho lắm) (1) mỗi nhân viên đều có thể làm việc cho mọi dự án, (2) mỗi nhân viên đều có thể làm việc tại mọi chi nhánh và (3) mỗi dự án đều có thể được thực hiện tại bất kỳ chi nhánh nào. Một quan hệ mẫu thoả các điều kiện này cho ở bảng sau:

SKILL

ENO	*PNO*	*PLACE*
E1	P1	Toronto
E1	P1	New York
E1	P1	London
E1	P2	Toronto
E1	P2	New York
E1	P2	London
E2	P1	Toronto
E2	P1	New York
E2	P1	London
E2	P2	Toronto
E2	P2	New York
E2	P2	London

Chú ý rằng không có phụ thuộc hàm không tầm thường nào trong quan hệ SKILL; tất cả thuộc tính là thuộc tính khoá. Quan hệ SKILL có hai phụ thuộc đa trị

$$ENO \rightarrow\rightarrow PNO$$
$$ENO \rightarrow\rightarrow PLACE$$

c. Phụ thuộc chiếu-nối

Cho lược đồ quan hệ $R=(A_1, A_2, ..., A_n)$ và $R_1, R_2, ..., R_k$ là các tập con của $\{A_1, A_2, ..., A_n\}$. Ta nói rằng $\{R_1, R_2, ..., R_k\}$ xác định một *phụ thuộc chiếu-nối* của R, nếu mọi quan hệ **r** của R là nối tự nhiên của các chiếu của nó lên $R_1, R_2, ..., R_k$, tức là

$$\mathbf{r} = \pi_{R1}(\mathbf{r}) >< \pi_{R2}(\mathbf{r}) >< \ ... >< \pi_{Rk}(\mathbf{r})$$

· Chú ý rằng *phụ thuộc đa trị là trường hợp riêng của phụ thuộc chiếu-nối*, tức là

$$X \rightarrow\rightarrow Y \quad \Rightarrow \quad \{X \cup Y, X \cup Z\}$$ xác định một phụ thuộc chiếu-nối,

trong đó $Z=R\backslash(X \cup Y)$.

Thật vậy, cho quan hệ **r** trên lược đồ R thỏa phụ thuộc đa trị X→→Y. Hiển nhiên $\pi_{XY}(\mathbf{r}) >< \pi_{XZ}(\mathbf{r}) \supset \mathbf{r}$. Ta chỉ cần chứng minh $\pi_{XY}(\mathbf{r}) >< \pi_{XZ}(\mathbf{r}) \subset \mathbf{r}$. Cho $(x,y) \in \pi_{XY}(\mathbf{r})$ và $(x,z) \in \pi_{XZ}(\mathbf{r})$. Khi đó tồn tại z' và y' thỏa $(x,y,z') \in \mathbf{r}$ và $(x,y',z) \in \mathbf{r}$ (theo định nghĩa phép chiếu), kéo theo $(x,y,z) \in \mathbf{r}$ (vì X→→Y). Từ đó suy ra $\pi_{XY}(\mathbf{r}) >< \pi_{XZ}(\mathbf{r}) \subset \mathbf{r}$.

◊ *Ví dụ*

Xét quan hệ SKILL(*ENO, PNO, PLACE*) ở trên. Do *ENO*→→*PNO*, nên {(*ENO,PNO*), (*ENO,PLACE*)} xác định phụ thuộc chiếu nối.
Ta có

$\pi_{ENO,PNO}$(SKILL)

ENO	*PNO*
E1	P1
E1	P2
E2	P1
E2	P2

$\pi_{ENO,PLACE}$(SKILL)

ENO	*PLACE*
E1	Toronto
E1	New York
E1	London
E2	Toronto
E2	New York
E2	London

Suy ra

$$\text{SKILL} = \pi_{ENO,PNO}(\text{SKILL}) >< \pi_{ENO,PLACE}(\text{SKILL})$$

5.3. Phụ thuộc đa trị

Trong phần này chúng ta sẽ nghiên cứu sâu hơn về phụ thuộc đa trị, mối quan hệ giữa phụ thuộc đa trị và phụ thuộc hàm.

5.3.1. Định nghĩa

Ta nhắc lại định nghĩa phụ thuộc đa trị.

Cho lược đồ quan hệ $R(A_1, A_2, ..., A_n)$ và X, Y là các tập con của $\{A_1, A_2, ..., A_n\}$. Ta nói rằng X *xác định đa trị* Y hay Y *phụ thuộc đa trị* vào X, ký hiệu X→→Y, nếu mọi quan hệ bất kỳ **r** của lược đồ R thoả mãn:

Ứng với mỗi giá trị của miền giá trị các thuộc tính trong X, có một tập giá trị các thuộc tính trong Y liên quan và tập này độc lập với các thuộc tính trong Z=R\(X∪Y), tức là:

$$\forall x \in D(X) \forall y,y' \in D(Y)\ \forall z, z' \in D(Z): (x,y,z), (x,y',z') \in \mathbf{r} \Rightarrow (x,y,z'), (x,y',z) \in \mathbf{r}$$

◊ *Ví dụ*

Xét lược đồ CTHRSG=(*C,T,H,R,S,G*), trong đó C (Course) là môn học, T (Teacher) là giáo viên, H (Hour) là tiết học, R (Room) là phòng học, S (Student) là sinh viên và G (Grade) là điểm số. Một quan hệ mẫu cho ở bảng sau

C	*T*	*H*	*R*	*S*	*G*
CS101	Trần	M9	222	Hùng	9
CS101	Trần	W9	333	Hùng	9
CS101	Trần	F9	222	Hùng	9
CS101	Trần	M9	222	Dũng	7
CS101	Trần	W9	333	Dũng	7
CS101	Trần	F9	222	Dũng	7

Trong ví dụ đơn giản này ta thấy *môn học* có nhiều giờ học, trong các phòng học khác nhau, trong tuần. Mỗi *sinh viên* có một bản ghi cho mỗi giờ học và điểm của sinh viên cũng được lặp lại tương ứng.

Như vậy ta suy ra phụ thuộc đa trị $C \rightarrow\rightarrow H,R$, tức là sẽ có tập *giờ-phòng* ứng với mỗi môn học, độc lập với các thuộc tính khác. Ví dụ, cho hai bản ghi

t = (CS101, Trần, M9, 222, Hùng, 9)
s = (CS101, Trần, W9, 333, Dũng, 7)

chúng ta chờ đợi rằng có thể hoán chuyển (M9, 222) của *t* với (W9, 333) của *s* để nhận được các bộ sau

u = (CS101, Trần, M9, 222, Dũng, 7)
v = (CS101, Trần, W9, 333, Hùng, 9)

Và ta thấy *u, v* cũng có mặt trong quan hệ trên.

Cần nhấn mạnh rằng, $C \rightarrow\rightarrow H,R$ đúng bởi vì với mỗi môn học *c*, nếu tồn tại các bộ

(c, h1, r1, t1, s1, g1)

và

(c, h2, r2, t2, s2, g2)

thì cũng sẽ tồn tại

(c, h1, r1, t2, s2, g2) và (c, h2, r2, t1, s1, g1).

Lưu ý rằng $C \rightarrow\rightarrow H$ và $C \rightarrow\rightarrow R$ không đúng, bởi vì, nếu ngược lại, từ *t* và *s* suy ra bản ghi

(CS101, Trần, M9, 333, Dũng, 7)

phải có trong quan hệ trên.

Tồn tại nhiều phụ thuộc đa trị khác như $C \rightarrow\rightarrow S,G$ và $H,R \rightarrow\rightarrow S,G$ (bài tập).

5.3.2. Các tiên đề phụ thuộc đa trị và phụ thuộc hàm

Ta sẽ trình bày tập hợp đầy đủ các tiên đề phụ thuộc hàm và phụ thuộc đa trị trên tập thuộc tính U. Các tiên đề Armstrong được nhắc lại vì tính hệ thống.

(A1) ***Quy tắc phản xạ phụ thuộc hàm***:

$$Y \subset X \subset U \Rightarrow X \rightarrow Y$$

(A2) ***Quy tắc tăng trưởng phụ thuộc hàm*:**

$$X \rightarrow Y \ \& \ Z \subset U \Rightarrow X \cup Z \rightarrow Y \cup Z$$

(A3) ***Quy tắc bắc cầu phụ thuộc hàm***:

$$X \rightarrow Y \ \& \ Y \rightarrow Z \Rightarrow X \rightarrow Z$$

(M1) ***Quy tắc bù phụ thuộc đa trị***:

$$X \rightarrow\rightarrow Y \Rightarrow X \rightarrow\rightarrow (U \setminus (X \cup Y))$$

(M2) ***Quy tắc tăng trưởng phụ thuộc đa trị***:

$$X \rightarrow\rightarrow Y \ \& \ V \subset W \Rightarrow X \cup W \rightarrow\rightarrow Y \cup V$$

(M3) ***Quy tắc bắc cầu phụ thuộc đa trị***:

$$X \rightarrow\rightarrow Y \ \& \ Y \rightarrow\rightarrow Z \Rightarrow X \rightarrow\rightarrow (Z \setminus Y)$$

(M4) ***Quy tắc phụ thuộc hàm-đa trị***:

$$X \rightarrow Y \Rightarrow X \rightarrow\rightarrow Y$$

(M5) ***Quy tắc phụ thuộc đa trị-hàm***:

$$X \rightarrow\rightarrow Y \ \& \ Z \subset Y \ \& \ W \rightarrow Z \ \& \ W \cap Y = \varnothing \Rightarrow X \rightarrow Z$$

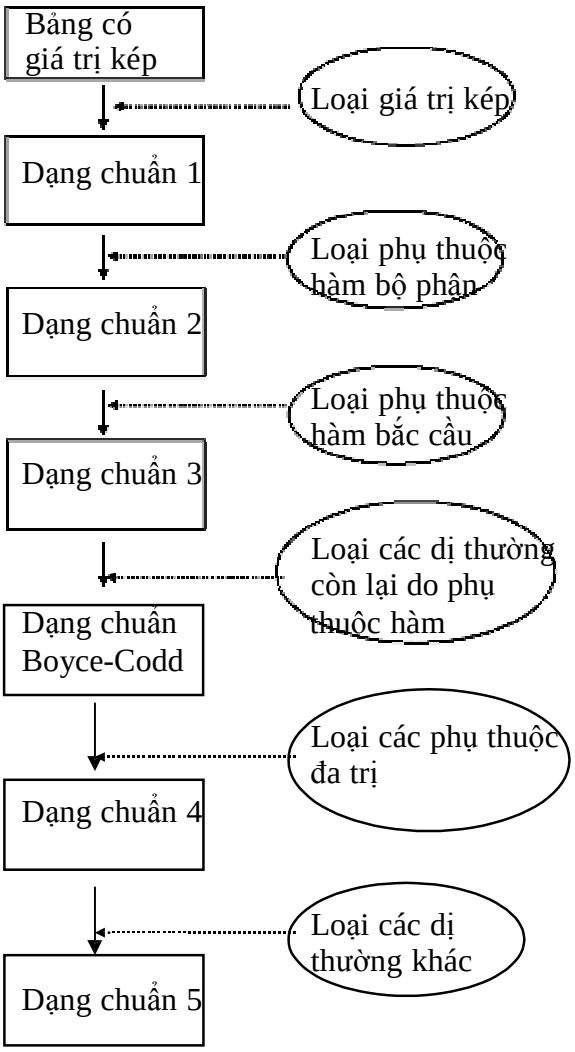
· ***Định lý*:** Các tiên đề A1-A3 và M1-M5 là đúng và đủ cho phụ thuộc hàm và phụ thuộc đa trị. Tức là, nếu D là tập hợp các phụ thuộc hàm và phụ thuộc đa trị trên tập thuộc tính U, và D^+ là tập hợp các phụ thuộc hàm và phụ thuộc đa trị suy diễn lôgic từ D (theo nghĩa mỗi quan hệ thoả D thì cũng thoả D^+), thì D^+ chính là tập hợp các phụ thuộc hàm và phụ thuộc đa trị suy ra từ D bằng các tiên đề trên.

Chứng minh. (công nhận)

5.4. Chuẩn hoá lược đồ quan hệ

Chúng ta đã chỉ ra rằng sự dư thừa dữ liệu là nguyên nhân của các dị thường khi cập nhật dữ liệu dẫn đến sự không tương thích dữ liệu và các hậu quả nghiêm trọng khác. Một lược đồ cơ sở dữ liệu được cho là *tốt* là phải loại bỏ được sự dư thừa dữ liệu. Tuy nhiên ta cần đưa ra định nghĩa chính xác thế nào là lược đồ cơ sở dữ liệu tốt cùng với quá trình thiết kế chúng. Quá trình biến đổi một lược đồ cơ sở dữ liệu thành *lược đồ tương đương*, tức phải bảo toàn thông tin và bảo toàn phụ thuộc dữ liệu, thoả mãn những tiêu chuẩn nhất định gọi là quá trình *chuẩn hoá lược đồ quan hệ*.

Chuẩn hoá lược đồ quan hệ thường được thực hiện qua các giai đoạn tương ứng với các dạng chuẩn (xem sơ đồ dưới). *Dạng chuẩn* là trạng thái quan hệ được xác định bằng cách áp dụng các quy tắc đối với phụ thuộc hàm của quan hệ.



5.4.1. Dạng chuẩn thứ nhất (1NF)

Quan hệ gọi là ở *dạng chuẩn thứ nhất* hay *quan hệ chuẩn hoá* nếu miền giá trị của mỗi thuộc tính chỉ chứa những giá trị *nguyên tử*, tức là không phân chia được nữa. Như vậy mỗi giá trị trong quan hệ cũng là nguyên tử.

Dạng chuẩn 1 chỉ có ý nghĩa ở mức thể hiện của lược đồ quan hệ, vì chỉ liên quan đến giá trị các thuộc tính của các bộ trong một quan hệ được định nghĩa trên lược đồ quan hệ đó.

5.4.2. Dạng chuẩn thứ 2 (2NF)

Thuộc tính A gọi là *phụ thuộc đầy đủ* vào tập thuộc tính X, nếu $X \rightarrow A$ là phụ thuộc hàm nguyên tố.

Giả sử K là khoá của lược đồ R. Khi đó mọi thuộc tính không khoá A của R đều phụ thuộc hàm vào khoá K: $K \rightarrow A$. Nếu A không phụ thuộc đầy đủ vào K thì tồn tại tập con thực sự H của K xác định hàm A, tức $H \rightarrow A$. Khi đó phụ thuộc hàm $H \rightarrow A$ gọi là *phụ thuộc hàm bộ phận*.

Một lược đồ quan hệ gọi là *ở dạng chuẩn thứ 2* nếu nó ở dạng chuẩn thứ 1 và không có phụ thuộc hàm bộ phận, tức là mọi thuộc tính không khoá đều phụ thuộc đầy đủ vào các khoá của lược đồ.

◊ *Ví dụ*

- Xét các quan hệ sau:

EMP(*ENO, ENAME, TITLE, SAL, PNO, RESP, DUR*)
PROJ(*PNO, PNAME, BUDGET*)

Lược đồ của EMP có khoá là (*ENO, PNO*).

Phụ thuộc hàm *ENO→*(*ENAME, TITLE*) là phụ thuộc hàm bộ phận vì vế phải là tập con thực sự của khoá. Vậy EMP không ở dạng chuẩn thứ 2.

Lược đồ của PROJ không có phụ thuộc hàm bộ phận, vậy nó ở dạng chuẩn 2.

- Xét quan hệ KHO_HANG(*Kho, Hang, QuayHang, NhanVien*). Lược đồ của quan hệ này có hai phụ thuộc hàm sau:

Kho,Hang→QuayHang: Mỗi mặt hàng ở mỗi kho chỉ được bán ở 1 quầy hàng;

Kho,QuayHang→NhanVien: Mỗi quầy hàng của mỗi kho chỉ có 1 nhân viên phụ trách.

Khoá của lược đồ này là (*Kho, Hang*).

Vậy lược đồ này ở dạng chuẩn thứ 2 vì không có phụ thuộc hàm bộ phận.

5.4.3. Dạng chuẩn thứ 3 (3NF)

Phụ thuộc hàm X→A gọi là *phụ thuộc hàm bắc cầu*, nếu nó là phụ thuộc hàm nguyên tố, A là thuộc tính không khoá, A∉X, và X chứa thuộc tính không khoá.

Khi đó với mọi khoá K ta có các phụ thuộc hàm không tầm thường K→X & X→A. Mặt khác không thể có X→K, vì X chứa các thuộc tính không khoá và không chứa khoá (vì X→A là nguyên tố).

Nói một cách khác phụ thuộc hàm bắc cầu là sự phụ thuộc không tầm thường giữa các thuộc tính không khoá.

Một lược đồ quan hệ gọi là *ở dạng chuẩn thứ 3* nếu nó ở dạng chuẩn thứ 2 và không có phụ thuộc hàm bắc cầu.

◊ *Ví dụ*

- Lược đồ của quan hệ

EMP(*ENO, ENAME, TITLE, SAL, PNO, RESP, DUR*)

có khoá là (*ENO, PNO*).

Phụ thuộc hàm *TITLE→SAL* là phụ thuộc hàm bắc cầu. Vậy EMP không ở dạng chuẩn thứ 3.

- Lược đồ của quan hệ

PROJ(*PNO, PNAME, BUDGET*)

không có phụ thuộc hàm bắc cầu, vậy nó ở dạng chuẩn 3.

- Xét quan hệ KHO_HANG(*Kho, Hang, QuayHang, NhanVien*). Ta có hai phụ thuộc hàm sau:

Kho,Hang→QuayHang: Mỗi mặt hàng ở mỗi kho chỉ được bàn ở 1 quầy hàng;

Kho,QuayHang→NhanVien: Mỗi quầy hàng của mỗi kho chỉ có 1 nhân viên phụ trách.

Khoá của lược đồ này là (*Kho, Hang*).

Phụ thuộc hàm thứ hai là phụ thuộc hàm bắc cầu, vì thế lược đồ không ở dạng chuẩn thứ 3, mặc dù nó ở dạng chuẩn thứ 2.

5.4.4. Dạng chuẩn Boyce-Codd (BCNF)

Một lược đồ quan hệ gọi là ở *dạng chuẩn Boyce-Codd* nếu mọi phụ thuộc hàm không tầm thường đều có vế trái là siêu khoá

◊ *Ví dụ*:

- Lược đồ của quan hệ

PROJ(*PNO, PNAME, BUDGET*)

chỉ có phụ thuộc hàm duy nhất *PNO→*(*PNAME, BUDGET*), vậy nó ở dạng chuẩn Boyce-Codd.

- Xét lược đồ

LOPHOC(*Lop,MonHoc,GiaoVien*) với 2 phụ thuộc hàm sau:

$$GiaoVien \rightarrow MonHoc \quad \text{và} \quad (Lop,MonHoc) \rightarrow GiaoVien$$

Lược đồ có 2 khoá

$$K_1 = (Lop, MonHoc) \text{ và } K_2 = (Lop, GiaoVien),$$

nên tất cả thuộc tính đều là thuộc tính khoá. Như vậy lược đồ ở dạng chuẩn thứ 3. Tuy nhiên lược đồ không ở dạng chuẩn Boyce-Codd vì phụ thuộc hàm

$$GiaoVien \rightarrow MonHoc$$

không thoả yêu cầu vế trái phải là siêu khoá.

Sự dị thường khi thêm bộ hay sửa bộ thể hiện ở chỗ nếu một giáo viên dạy nhiều lớp (cùng một môn học) thì thông tin về giáo viên đó lặp lại nhiều lần gây dư thừa dữ liệu.

Sự dị thường khi xoá bộ thể hiện ở chỗ nếu giáo viên T chỉ dạy lớp C nào đó, thì thông tin về giáo viên T (môn học mà giáo viên đó dạy) sẽ bị mất nếu ta xoá bản ghi tương ứng (chẳng hạn vì giáo viên T thôi không dạy lớp C nữa).

5.4.5. Dạng chuẩn thứ 4 (4NF)

Một quan hệ R được gọi là *ở dạng chuẩn thứ 4*, nếu với mỗi phụ thuộc đa trị X→→Y trong R, X cũng xác định hàm tất cả thuộc tính của R.

Như vậy, *nếu quan hệ ở dạng chuẩn BCNF và các phụ thuộc đa trị cũng là phụ thuộc hàm thì quan hệ này ở dạng chuẩn 4*.

◊ *Ví dụ*: Xét quan hệ

SKILL(*ENO, PNO, PLACE*)

ENO	*PNO*	*PLACE*
E1	P1	Toronto
E1	P1	New York
E1	P1	London
E1	P2	Toronto
E1	P2	New York
E1	P2	London
E2	P1	Toronto
E2	P1	New York
E2	P1	London
E2	P2	Toronto
E2	P2	New York
E2	P2	London

Chú ý rằng không có phụ thuộc hàm nào trong quan hệ SKILL; tất cả thuộc tính là thuộc tính khoá. Quan hệ SKILL có hai phụ thuộc đa trị

ENO→→PNO
ENO→→PLACE

Vì quan hệ không có phụ thuộc hàm nên nó ở dạng BCNF. Tuy nhiên nó không ở dạng chuẩn 4, vì *ENO* không phải là khoá.

Để đạt dạng chuẩn 4, cần phân rã SKILL thành hai quan hệ

EP(*ENO, PNO*) và EL(*ENO, PLACE*)

5.4.6. Dạng chuẩn thứ 5 (5NF)

Một quan hệ R được gọi là *ở dạng chuẩn thứ 5*, còn gọi là *dạng chuẩn chiếu-nối* PJNF, nếu mỗi phụ thuộc chiếu nối được xác định bởi các khoá của R.

◊ *Ví dụ*

Với quan hệ PROJ(*PNO, PNAME, BUDGET*) ta có phụ thuộc chiếu-nối

{(*PNO, PNAME*), (*PNO, BUDGET*)}

và mỗi thành phần đều có khoá chính PNO. Vì vậy PROJ ở dạng chuẩn 5.

CHƯƠNG 2

CƠ SỞ DỮ LIỆU PHÂN TÁN

1. Hệ quản trị cơ sở dữ liệu phân tán

1.1. Khái niệm hệ quản trị cơ sở dữ liệu phân tán

Công nghệ các *hệ quản trị cơ sở dữ liệu phân tán* (*distributed database management system - distributed DBMS*) là sự hợp nhất của hai hướng tiếp cận đối với quá trình xử lý dữ liệu: *Công nghệ cơ sở dữ liệu* và *công nghệ mạng máy tính*. Xử lý dữ liệu chuyển từ hệ thống xử lý file cổ điển sang dạng cơ sở dữ liệu, quản lý tập trung. Điều này dẫn đến tính *độc lập dữ liệu*, nghĩa là các ứng dụng được "miễn nhiễm" đối với những thay đổi về tổ chức lôgic hoặc vật lý của dữ liệu và ngược lại.

Một trong những động lực chủ yếu thúc đẩy sử dụng cơ sở dữ liệu là nhu cầu tích hợp các dữ liệu hoạt tác của một xí nghiệp và cho phép truy xuất tập trung. Công nghệ mạng máy tính đặc trưng ở chỗ phi tập trung hoá thiết bị. Tuy nhiên điểm mấu chốt của ý tưởng cơ sở dữ liệu phân tán là *tích hợp* (*integration*), chứ không phải *tập trung hoá* (*centralization*). Cần hiểu rằng có thể tích hợp mà không cần tập trung. Và đây chính là mục tiêu của công nghệ cơ sở dữ liệu phân tán.

Tại sao chúng ta phải thực hiện phân tán ? Câu trả lời kinh điển cho câu hỏi này là việc xử lý phân tán nhằm thích ứng tốt hơn với việc phân bố ngày càng rộng rãi các công ty, xí nghiệp. Nhiều ứng dụng hiện tại của công nghiệp máy tính được phân tán. Thương mại điện tử, các ứng dụng đa phương tiện, giáo dục từ xa, chữa bệnh từ xa, điều khiển sản xuất từ xa, ... là các ví dụ minh hoạ.

Tuy nhiên từ góc độ tổng quát hơn, xử lý phân tán là một biến thể của qui tắc "*chia để trị*" nhằm giải quyết tốt hơn các bài toán lớn và phức tạp. Từ quan điểm kinh tế, cách tiếp cận này có ưu điểm cơ bản là việc tính toán phân tán tận dụng sức mạnh của nhiều bộ phận xử lý một cách tối ưu.

- ***Hệ cơ sở dữ liệu phân tán là gì ?***

Chúng ta có thể định nghĩa một *cơ sở dữ liệu phân tán* là tập hợp nhiều cơ sở dữ liệu có liên quan lôgic và được phân bổ trên một mạng máy tính.

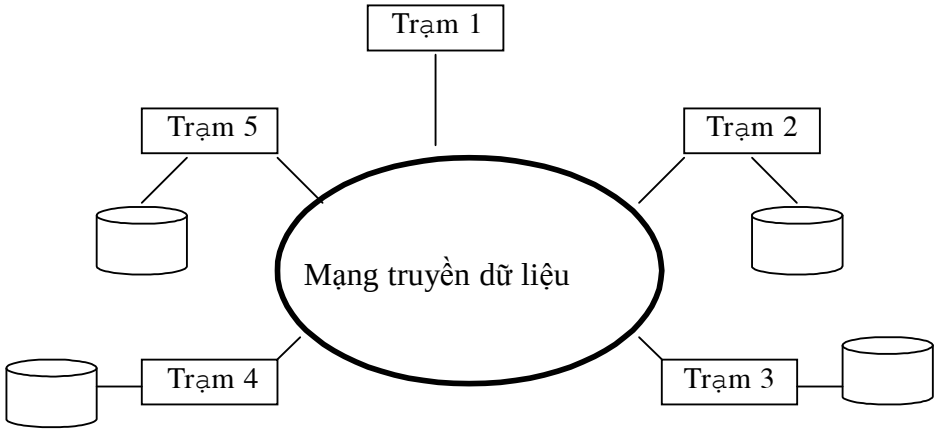
Hệ quản trị cơ sở dữ liệu phân tán (*distributed database management system - distributed DBMS*) là hệ thống phần mềm cho phép quản lý các hệ cơ sở dữ liệu phân tán và làm cho việc phân tán trở nên *vô hình* đối với người sử dụng.

Lưu ý rằng một hệ cơ sở dữ liệu phân tán không phải là tập hợp các tập tin lưu trữ riêng rẽ tại các nút của mạng máy tính. Để tạo ra một hệ cơ sở dữ liệu phân tán, các tập tin không chỉ liên đới lôgic mà còn phải có cấu trúc chung và được truy xuất qua một giao diện chung. Cũng cần phân biệt một hệ cơ sở dữ liệu phân tán với các *dữ liệu bán cấu trúc* (*semi-structrured data*) được lưu trên Internet (chẳng hạn như các trang Web).

Định nghĩa trên cũng loại bỏ các hệ thống đa bộ xử lý dùng chung bộ nhớ trong (shared memory) hoặc dùng chung bộ nhớ thứ cấp (shared disk).

Ngoài ra một hệ cơ sở dữ liệu phân tán không phải là hệ thống mà trong đó, mặc dù có sự hiện diện của mạng máy tính, cơ sở dữ liệu chỉ nằm tại một nút của mạng.

Môi trường của một hệ CSDL phân tán có thể biểu diễn bằng sơ đồ sau



• ***Các đặc trưng của cơ sở dữ liệu phân tán***

Đặc tính *vô hình* là sự tách biệt về ngữ nghĩa ở mức độ cao của hệ thống với các vấn đề cài đặt ở cấp độ thấp. Ưu điểm của hệ cơ sở dữ liệu vô hình là không cho người dùng "nhìn thấy" các chi tiết cài đặt, hỗ trợ phát triển các ứng dụng phức tạp.

Độc lập dữ liệu là dạng vô hình cơ bản cần có trong một hệ cơ sở dữ liệu. Sự độc lập dữ liệu liên quan đến khả năng "miễn nhiễm" của các ứng dụng đối với những thay đổi trong định nghĩa và tổ chức dữ liệu, và ngược lại.

Vô hình kết mạng. Trong môi trường phân tán, hệ thống mạng là một loại tài nguyên quan trọng cần quản lý. Thông thường, người dùng cần được tách khỏi mọi chi tiết hoạt động của mạng, thậm chí người ta mong muốn che dấu sự tồn tại của mạng, nếu được. Khi đó đối với người dùng sẽ không có sự khác biệt giữa các ứng dụng chạy trên cơ sở dữ liệu tập trung và các ứng dụng chạy trên cơ sở dữ liệu phân tán. Kiểu vô hình này gọi là *vô hình kết mạng* (*network transparency*) hoặc *vô hình phân bổ* (*distribution transparency*).

Vô hình nhân bản. Vì những lý do về hiệu năng (performance), độ tin cậy (reliability) và tính sẵn sàng (availability), người ta mong muốn có thể nhân dữ liệu thành nhiều bản (nhân bản) trên các máy mạng. Việc nhân bản giúp tăng hiệu năng vì những yêu cầu sử dụng có xung đột và nằm rải rác có thể đáp ứng kịp thời. Thí dụ, dữ liệu thường được một người truy xuất có thể được đặt tại máy của người đó và trên máy của những người khác có cùng nhu cầu truy xuất, như thế sẽ làm tăng khu vực truy xuất. Ngoài ra nếu một máy phải ngưng hoạt động, một bản sao khác của dữ liệu vẫn có sẵn trên máy khác của mạng. Tuy nhiên việc nhân bản sẽ gây khó khăn khi cập nhật cơ sở dữ liệu. Vì vậy việc nhân bản và qui mô nhân bản do các ứng dụng quyết định.

Vô hình phân mảnh. Phân hoạch dữ liệu cho các vị trí khác nhau là yêu cầu tất yếu của hệ phân tán. Quá trình này gọi là *quá trình phân mảnh* (*fragmentation*). Có hai kiểu phân mảnh. *Phân mảnh ngang* (*horizontal fragmentation*), trong đó mỗi quan hệ được phân hoạch thành tập các quan hệ con, mỗi quan hệ con này chứa một tập con các bộ của quan hệ ban đầu. *Phân mảnh dọc* (*vertical fragmentation*), trong đó mỗi quan hệ được phân hoạch thành tập các quan hệ

con, mỗi quan hệ con này được định nghĩa trên một tập con các thuộc tính của quan hệ ban đầu).

Khi các đối tượng cơ sở dữ liệu bị phân mảnh, chiến lược xử lý vấn tin là dựa trên các mảnh chứ không phải quan hệ. Như vậy *câu vấn tin toàn cục* (*global query*) phải được dịch thành *câu vấn tin theo mảnh* (*fragment query*).

1.2. Mô hình kiến trúc hệ quản trị cơ sở dữ liệu phân tán

Kiến trúc của một hệ thống xác định cấu trúc của nó. Tức là các thành phần của hệ thống được xác định, chức năng mỗi thành phần được mô tả, các mối tương liên (interrelationship) và tương tác (interaction) giữa các thành phần được định nghĩa.

Chúng ta hãy xem xét một số cách kết hợp nhiều cơ sở dữ liệu lại để dùng chung cho nhiều hệ quản trị cơ sở dữ liệu. Ta phân loại hệ thống theo các đặc điểm (1) *tính tự trị* (*autonomy*) của các hệ thống cục bộ, (2) *tính phân tán* (*distribution*) của chúng, (3) *tính đa chủng* (*heterogeneity*) của chúng.

1.2.1. Tính tự trị

Tính tự trị (*autonomy*) muốn nói đến sự *phân bổ quyền điều khiển*, chứ không phải phân bố dữ liệu. Tính tự trị chỉ ra mức độ hoạt tác độc lập của từng hệ quản trị cơ sở dữ liệu. Tính tự trị biểu hiện qua một số yếu tố sau:

- Các hệ thống thành viên có trao đổi thông tin với nhau không.
- Các hệ thống thực hiện các giao dịch một cách độc lập hay không.
- Các hệ thống có được sửa đổi hay không.

Từ đó người ta xây dựng các yêu cầu đối với một hệ thống tự trị. Chẳng hạn hệ thống tự trị phải thoả mãn:

(1) Các hoạt động cục bộ của từng hệ quản trị cơ sở dữ liệu không bị ảnh hưởng bởi sự tham gia của chúng vào trong phức hệ cơ sở dữ liệu (multidatabase system).

(2) Phương thức xử lý và tối ưu hoá vấn tin trong từng hệ quản trị cơ sở dữ liệu không bị ảnh hưởng bởi việc thực hiện các câu truy vấn toàn cục truy xuất nhiều cơ sở dữ liệu.

(3) Tính nhất quán và hoạt động của hệ thống không bị ảnh hưởng khi từng hệ quản trị cơ sở dữ liệu riêng rẽ tham gia hoặc tách ra khỏi liên minh cơ sở dữ liệu.

Ở bình diện khác, tính tự trị thể hiện ở các khía cạnh sau:

(1) *Tự trị thiết kế* (*design autonomy*): Mỗi hệ quản trị cơ sở dữ liệu tự do sử dụng các mô hình dữ liệu và các kỹ thuật quản lý giao dịch thích hợp.

(2) *Tự trị truyền thông* (*communication autonomy*): Mỗi hệ quản trị cơ sở dữ liệu tự do quyết định loại thông tin cung cấp cho hệ quản trị cơ sở dữ liệu khác hoặc cho các phần mềm điều khiển hoạt động toàn cục.

(3) *Tự trị thực thi* (*execution autonomy*): Mỗi hệ quản trị cơ sở dữ liệu có thể thực hiện các giao dịch theo phương thức của mình.

Tính tự trị có thể chia làm ba cấp độ sau:

(**0**) *Tích hợp mật thiết* (*tight integration*): Chỉ tồn tại một hình ảnh duy nhất về toàn bộ hệ thống cơ sở dữ liệu cho người dùng muốn dùng chung thông tin trong

nhiều cơ sở dữ liệu. Một trong các bộ quản lý dữ liệu (data manager) nắm quyền kiểm soát việc xử lý yêu cầu của người dùng, ngay cả khi yêu cầu đó phải được nhiều bộ quản lý dữ liệu tham gia xử lý.

(**1**) *Hệ thống bán tự trị* (*semiautonomous system*): Bao gồm các hệ quản trị cơ sở dữ liệu có thể hoạt tác độc lập, nhưng quyết định tham gia vào liên minh nhằm chia sẻ dữ liệu cục bộ của chúng. Mỗi hệ quản trị cơ sở dữ liệu phải xác định những phần cơ sở dữ liệu nào của riêng chúng mà các hệ quản trị cơ sở dữ liệu khác được truy xuất. Chúng không phải là hệ thống tự trị hoàn toàn mà cần sửa đổi lại để có thể trao đổi thông tin với những hệ thống khác.

(**2**) *Hệ thống cô lập*: Các hệ quản trị cơ sở dữ liệu hoạt động cô lập, không có giao tiếp chia sẻ dữ liệu với nhau.

1.2.2. Tính phân tán

Tính phân tán (*distribution*) chỉ khả năng *phân bố dữ liệu* ở những vị trí khác nhau. Có hai loại kiến trúc phân tán:

(**1**) *Phân tán khách/chủ* (*client/server*): Đây là hình thức phân tán chức năng. Thành phần chủ (server) chịu trách nhiệm quản trị dữ liệu; thành phần khách (client) chịu trách nhiệm cung cấp môi trường ứng dụng, kể cả giao diện người dùng.

Nhiệm vụ truyền thông được chia sẻ giữa chủ và khách.

(**2**) *Phân tán ngang hàng* (*peer-to-peer*): Còn gọi là *phân tán hoàn toàn*. Không có sự phân biệt giữa máy chủ và khách, mỗi máy đều có đầy đủ chức năng của một hệ quản trị cơ sở dữ liệu và có thể trao đổi thông tin với máy khác để thực hiện vấn tin và giao dịch.

1.2.3. Tính đa chủng

Tính đa chủng (*heterogeneous*) thể hiện dưới nhiều hình thái khác nhau trong các hệ phân tán, từ khác biệt về phần cứng, các giao thức kết nối mạng đến sự khác biệt của các bộ quản lý dữ liệu.

Tính đa chủng liên quan đến sự khác biệt các mô hình dữ liệu (data model), ngôn ngữ vấn tin (query language) và nghi thức quản lý giao dịch (transaction management protocol).

1.2.4. Các kiểu kiến trúc

Ta tổ hợp các mức độ tự trị, phân tán và đa chủng để nghiên cứu các mô hình kiến trúc khác nhau.

Ký hiệu

A0 ... hệ thống tự trị tích hợp
A1 ... hệ thống bán tự trị
A2 ... hệ thống cô lập
D0 ... hệ thống không phân tán (tập trung)
D1 ... hệ thống phân tán khách/chủ
D2 ... hệ thống phân tán ngang hàng
H0 ... hệ thống đồng chủng

H1 ... hệ thống đa chủng

• Mô hình (A0,D0,H0): *Hệ thống tích hợp, không phân tán, đồng chủng*. Được gọi là *hệ thống phức hợp* (composite system), bao gồm nhiều hệ quản trị cơ sở dữ liệu được tích hợp về mặt lôgic. Kiến trúc này phù hợp với những hệ thống đa bộ xử lý và mọi tài nguyên dùng chung.

• Mô hình (A0,D0,H1): *Hệ thống tích hợp, không phân tán, đa chủng*. Nó có nhiều hệ quản trị cơ sở dữ liệu đa chủng, nhưng cung cấp một hình ảnh tích hợp cho người dùng. Chẳng hạn trên cơ sở dữ liệu mạng cùng hiện diện cả cơ sở dữ liệu phân cấp và cơ sở dữ liệu quan hệ.

• Mô hình (A0,D1,H0): *Hệ thống tích hợp, phân tán khách/chủ, đồng chủng*. Hệ thống cung cấp một hình ảnh tích hợp cho người dùng.

• Mô hình (A0,D2,H0): *Hệ thống tích hợp, phân tán hoàn toàn, đồng chủng*. Hệ thống không phân biệt khách chủ. Mỗi vị trí đều trang bị đầy đủ chức năng.

• Mô hình (A1,D0,H0): *Hệ thống bán tự trị, không phân tán, đồng chủng*, được gọi dưới cái tên *hệ quản trị cơ sở dữ liệu liên bang* (*federated DBMS*).

Các hệ thống thành viên đều có quyền tự trị nhất định trong các hoạt động của chúng. Chúng tự do hiệp đồng với những hệ thống khác khi thực hiện các yêu cầu của người dùng truy xuất đến nhiều cơ sở dữ liệu.

◊ *Ví dụ*. Nhiều bản cài đặt một hệ quản trị cơ sở dữ liệu "*mở*" trên cùng một máy. "Mở" ở đây có nghĩa là hệ quản trị cơ sở dữ liệu có khả năng tham gia vào liên bang.

• Mô hình (A1,D0,H1): *Hệ thống bán tự trị, không phân tán, đa chủng*, được gọi dưới cái tên *hệ quản trị cơ sở dữ liệu liên bang đa chủng* (heterogeneous federated DBMS).

◊ *Ví dụ*. Một hệ quản trị cơ sở dữ liệu quan hệ lo quản lý dữ liệu có cấu trúc, một hệ quản trị cơ sở dữ liệu đồ hoạ lo quản lý hình ảnh tĩnh, và một máy chủ cung cấp các hình video. Nếu chúng ta muốn cung cấp một hình ảnh tích hợp cho người dùng thì cần phải "che dấu" tính tự trị và đa chủng của các hệ thống thành viên và thiết lập giao diện chung.

• Mô hình (A1,D1,H1): *Hệ thống bán tự trị, phân tán khách/chủ, đa chủng*, được gọi dưới cái tên *hệ quản trị cơ sở dữ liệu liên bang đa chủng phân tán* (heterogeneous federated distributed DBMS). Dữ liệu được phân tán trên các máy khác nhau.

• Kiến trúc (A2,D0,H0): *Hệ thống cô lập, không phân tán, đồng chủng*, được gọi dưới cái tên *hệ quản trị cơ sở dữ liệu phức hệ* (multidatabase system-MDBS). Các thành viên không có khái niệm hiệp đồng. Đây thực chất là tập các cơ sở dữ liệu tự trị và được kết nối lại.

• Mô hình (A2,D0,H1): *Hệ thống cô lập, không phân tán, đa chủng*. Hệ thống này được dùng để xây dựng các ứng dụng truy xuất dữ liệu từ nhiều hệ thống lưu trữ khác nhau với các đặc tính khác nhau. Một số hệ thống có thể không phải là hệ quản trị cơ sở dữ liệu.

• Mô hình (A2,D1,H1) và (A2,D2,H1): Ta xét chung hai trường hợp này do tính tương tự của những vấn đề do chúng sinh ra. Cả hai đều biểu diễn cho trường hợp các cơ sở dữ liệu thành viên tạo ra phức hệ phân tán trên một số vị trí – chúng được gọi là các *phức hệ cơ sở dữ liệu phân tán* (distributed MDBS). Khác biệt chính giữa hai kiến trúc này là ở phân tán khách/chủ, phần lớn các công việc tương tác được trao cho *hệ thống trung gian* (middleware system), tạo ra *kiến trúc ba tầng* (three layer architecture).

1.3. Kiến trúc hệ quản trị cơ sở dữ liệu phân tán

Chúng ta sẽ xem xét chi tiết ba kiến trúc hệ thống trong số các kiến trúc giới thiệu ở phần trước. Đó là các hệ thống khách/chủ (Ax,D1,Hy), các hệ phân tán (A0,D2,H0) và các phức hệ (A2,Dx,Hy).

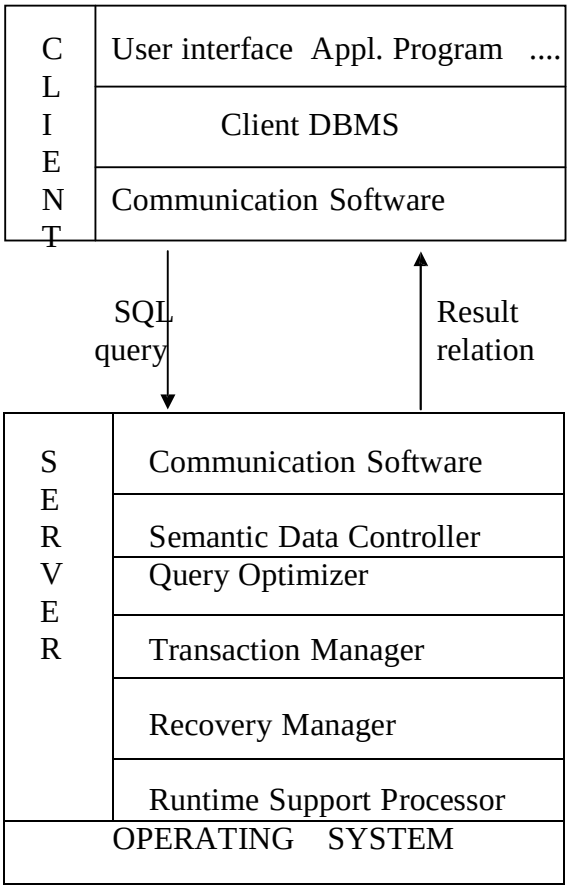
1.3.1. Các hệ khách/chủ

Các hệ quản trị cơ sở dữ liệu *khách/chủ* xuất hiện đầu những năm 90 và có ảnh hưởng lớn đến công nghệ DBMS và phương thức xử lý tính toán. ý tưởng tổng quát hết sức đơn giản và rõ ràng: phân biệt các chức năng cần được cung cấp và chia những chức năng này thành hai lớp: *chức năng chủ* (server function) và *chức năng khách* (client function). Nó cung cấp một *kiến trúc hai tầng* (*two-level architecture*), tạo điều kiện dễ dàng cho việc quản lý mức độ phức tạp của các hệ quản trị cơ sở dữ liệu hiện đại và độ phức tạp của việc phân tán dữ liệu.

Máy chủ thực hiện phần lớn công việc quản lý dữ liệu: xử lý tối ưu hoá vấn tin, quản lý giao dịch, quản lý thiết bị lưu trữ.

Máy khách quản lý các ứng dụng, giao diện, hệ quản trị cơ sở dữ liệu của khách, chịu trách nhiệm quản lý dữ liệu được gửi cho khách và có thể cả quản lý các khoá chốt giao dịch.

Kiến trúc này được mô tả trong hình sau.



kiến trúc tham chiếu khách/chủ

Kiến trúc này thông dụng trong các hệ cơ sở dữ liệu quan hệ, ở đó việc giao tiếp giữa khách và chủ nằm ở mức câu lệnh SQL. Khách hàng chuyển câu vấn tin cho máy chủ mà không cần biết nó thực hiện và tối ưu hoá như thế nào. Máy chủ thực hiện hầu hết công việc và gửi quan hệ kết quả về cho khách.

Có một số kiến trúc khách/chủ khác nhau:

• *Kiến trúc nhiều khách, một chủ.*

Từ góc độ quản lý, loại này không khác nhiều so với cơ sở dữ liệu tập trung, vì CSDL được lưu trên một máy chủ duy nhất và cũng có phần mềm quản lý.

Có một số khác biệt quan trọng ở cách thực hiện các giao dịch và quản lý bộ nhớ cache.

• *Kiến trúc nhiều khách, nhiều chủ*:

Có hai chiến lược quản lý:

- Mỗi máy khách tự quản lý kết nối của nó với các máy chủ khác nhau. Lối tiếp cận này làm đơn giản chương trình ở máy chủ nhưng đặt gánh nặng lên máy khách cùng với các trách nhiệm khác.
- Mỗi máy khách chỉ quan hệ với máy chủ đại diện của mình và giao tiếp với các máy chủ khác thông qua đại diện khi cần.

1.3.2. Các hệ phân tán ngang hàng

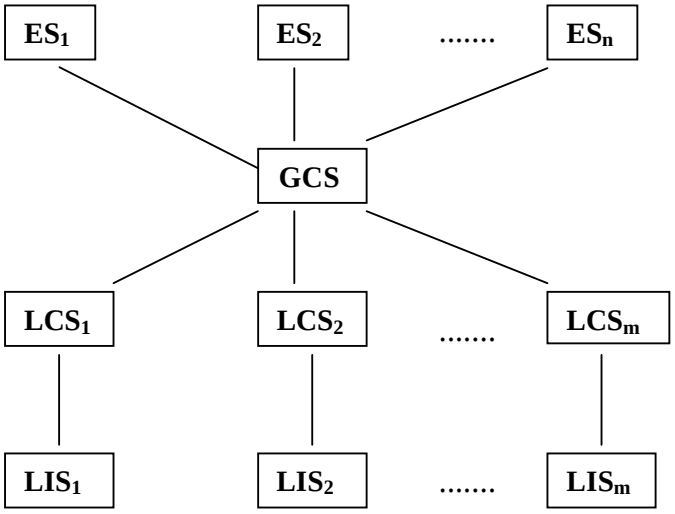
Trong kiến trúc này tổ chức dữ liệu vật lý trên mỗi máy có thể rất khác nhau. Điều này dẫn đến việc định nghĩa cấu trúc dữ liệu riêng cho mỗi vị trí, gọi là *lược đồ nội tại cục bộ* LIS (local internal schema). Cấu trúc lôgic của dữ liệu ở mọi vị trí được mô tả bằng *lược đồ khái niệm toàn cục* GCS (global conceptual schema).

Để mô tả tổ chức lôgic của dữ liệu tại mỗi vị trí cần phải có *tầng thứ ba* trong kiến trúc gọi là *lược đồ khái niệm cục bộ* LCS (local conceptual schema).

Lược đồ khái niệm toàn cục lúc này là *hợp* của các lược đồ khái niệm cục bộ.

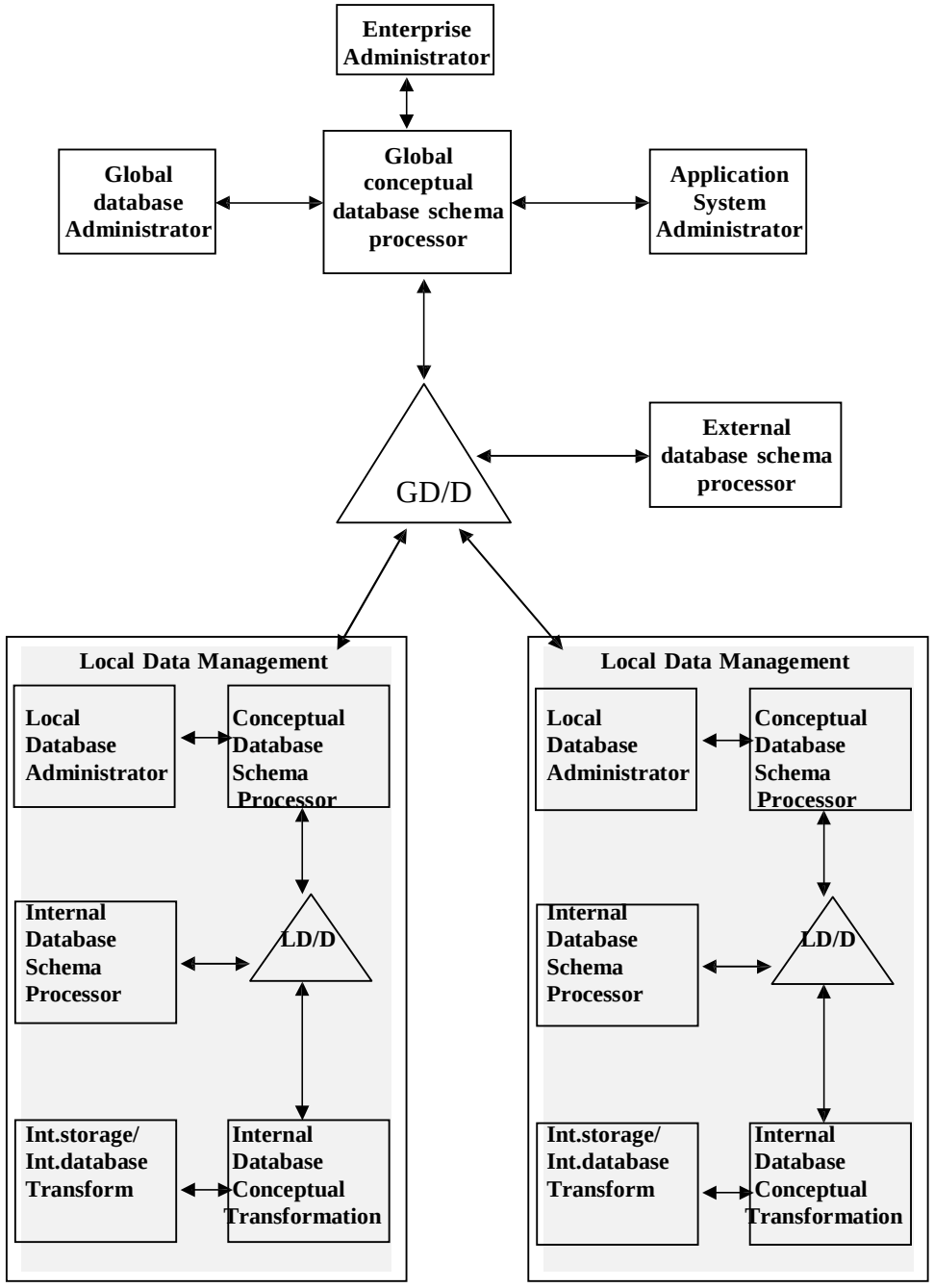
Các ứng dụng và việc truy xuất cơ sở dữ liệu được hỗ trợ qua các *lược đồ ngoại giới* ES (external schema), được định nghĩa là một tầng nằm trên tầng lược đồ khái niệm toàn cục.

Kiến trúc này được mô tả trong hình sau.



kiến trúc tham chiếu CSDL phân tán ngang hàng

Các chức năng của hệ phân tán ngang hàng được biểu diễn bằng sơ đồ sau:



sơ đồ chức năng của hệ quản trị CSDL phân tán ngang hàng

Trong hệ thống này các *từ điển/thư mục dữ liệu*, viết tắt là D/D, có vai trò trung tâm, vừa xử lý lược đồ dữ liệu vừa cung cấp các ánh xạ giữa chúng ở các cấp độ toàn cục và cục bộ.

• *Thư mục/từ điển toàn cục* GD/D (global directory/dictionary) chứa các định nghĩa lược đồ toàn cục và thực hiện các ánh xạ toàn cục.

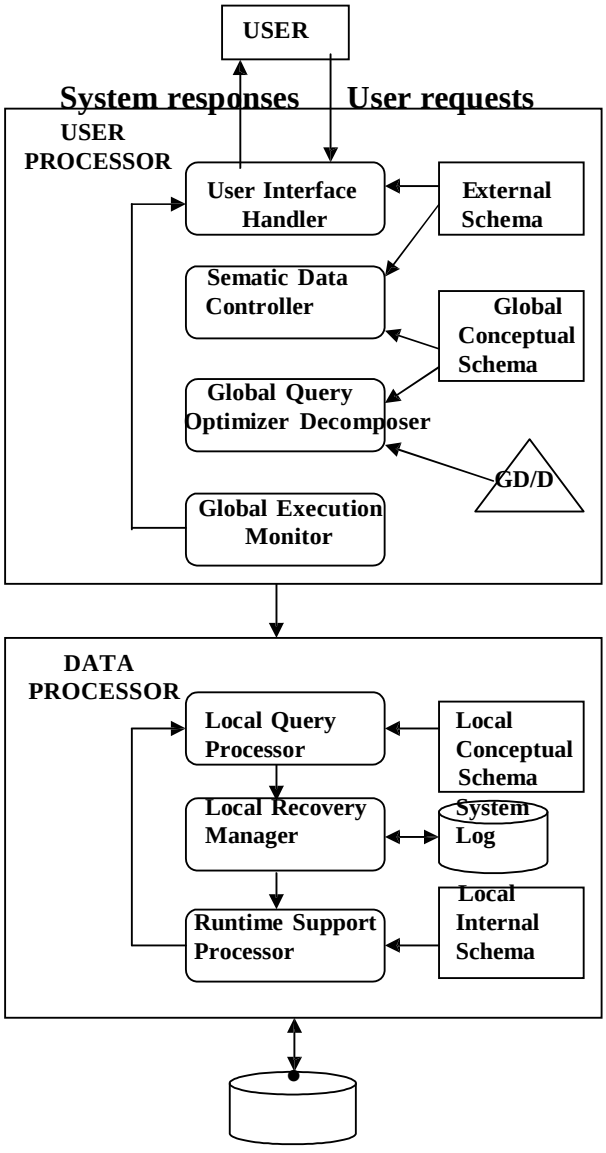
• *Thư mục/từ điển cục bộ* LD/D (local directory/dictionary) chứa các định nghĩa lược đồ cục bộ và thực hiện các ánh xạ cục bộ. Nó cũng có thể chứa các số liệu thống kê về các ứng dụng, các thông tin kiểm soát truy xuất,...

Các thành phần quản lý cơ sở dữ liệu cục bộ được tích hợp nhờ các chức năng của hệ quản trị cơ sở dữ liệu toàn cục.

Trong sơ đồ này, lược đồ khái niệm cục bộ là ánh xạ của lược đồ khái niệm toàn cục vào mỗi vị trí. Hơn nữa, những cơ sở dữ liệu loại này thường được thiết kế theo kiểu từ trên xuống, vì thế tất cả định nghĩa khung nhìn đều có phạm vi toàn cục.

Mỗi vị trí cũng có một quản trị viên cơ sở dữ liệu thể hiện mong muốn có được khả năng điều khiển cục bộ đối với hoạt động quản trị cơ sở dữ liệu.

Các thành phần của hệ quản trị cơ sở dữ liệu phân tán được trình bày trong hình sau:



Hệ thống có hai thành phần chính: Một thành phần lo xử lý mọi tương tác với người dùng gọi là *bộ phận phục vụ người dùng* (user processor), còn thành phần thứ hai lo việc lưu trữ dữ liệu, gọi là *bộ phận xử lý dữ liệu* (data processor).

- *Bộ phận phục vụ người dùng* (user processor): bao gồm bốn phần:
  (1) *Bộ phận giao tiếp* (user interface handler) chịu trách nhiệm diễn dịch các yêu cầu của người dùng (user requests) và định dạng dữ liệu kết quả để chuyển cho người dùng.
  (2) *Bộ phận kiểm soát dữ liệu ngữ nghĩa* (sematic data controller): sử dụng các *ràng buộc toàn vẹn* (integrity constraints) và thông tin quyền hạn (authorization), được định nghĩa như thành phần của lược đồ khái niệm toàn cục, để kiểm tra xem các câu vấn tin có thể xử lý được hay không.
  (3) *Bộ phận phân rã và tối ưu hoá vấn tin* (global query optimizer and decomposer) xác định chiến lược hoạt động nhằm giảm thiểu chi phí, phiên dịch các câu vấn tin toàn cục thành các câu vấn tin cục bộ bằng cách sử dụng các lược đồ khái niệm toàn cục, lược đồ khái niệm cục bộ và các thư mục toàn cục. Bộ phận tối ưu vấn tin toàn cục, ngoài những nhiệm vụ khác, còn chịu trách nhiệm tạo ra chiến lược thực thi tốt nhất cho các phép nối phân tán.
  (4) *Bộ phận theo dõi hoạt động phân tán* (distributed execution monitor) điều phối việc thực hiện phân tán các yêu cầu người dùng và cũng được gọi là *bộ quản lý giao dịch phân tán* (distributed transaction manager). Khi thực hiện các vấn tin phân tán, các bộ phận tại các vị trí có thể giao tiếp với nhau.

- *Bộ phận xử lý dữ liệu* (data processor): bao gồm ba phần.
  (1) *Bộ phận xử lý câu vấn tin cục bộ* (local query processor): hoạt động như *bộ chọn đường truy xuất* (access path selector), chịu trách nhiệm chọn ra một đường truy xuất thích hợp nhất để truy xuất các mục dữ liệu.
  (2) *Bộ phận khôi phục cục bộ* (local recovery manager): bảo đảm cho các cơ sở dữ liệu cục bộ vẫn duy trì được tính nhất quán ngay cả khi có sự cố xảy ra.
  (3) *Bộ phận hỗ trợ thực thi* (run-time support processor): truy xuất cơ sở dữ liệu tuỳ vào các lệnh trong *lịch biểu* (schedule) do bộ phận tối ưu vấn tin sinh ra. Nó chính là giao diện với hệ điều hành và chứa *bộ quản lý vùng đệm cơ sở dữ liệu* (database buffer manager), chịu trách nhiệm quản lý vùng đệm và việc truy xuất dữ liệu.

Lưu ý rằng việc sử dụng thuật ngữ *Bộ phận phục vụ người dùng* và *Bộ phận xử lý dữ liệu* không phải là sự phân chia chức năng giống như các hệ khách/chủ. Sự phân chia này chỉ thể hiện khía cạnh tổ chức và không bắt buộc phải đặt trên các máy khác nhau. Trong các hệ thống ngang hàng, người ta mong muốn có cả môdun phục vụ người dùng và môđun xử lý dữ liệu trên cùng một máy.

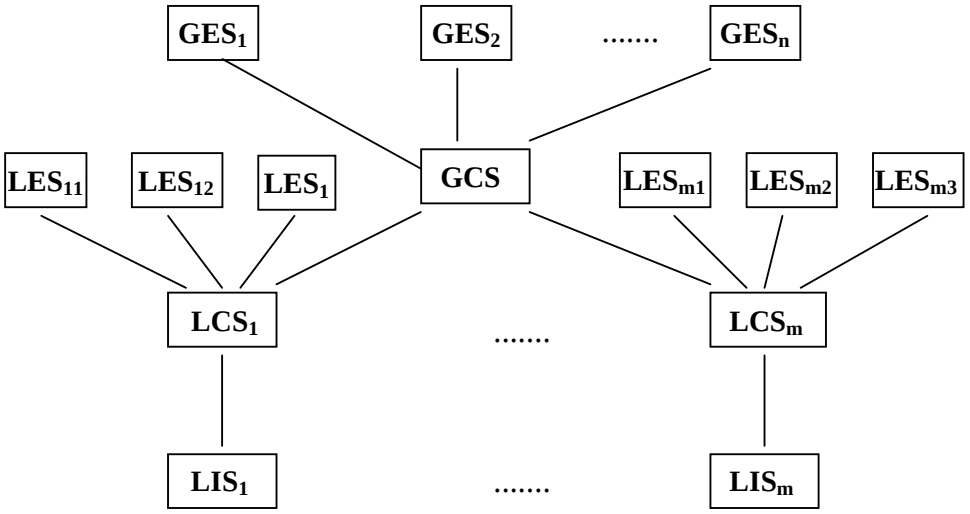
1.3.3. Các phức hệ cơ sở dữ liệu

Hệ quản trị phức hệ cơ sở dữ liệu phân tán khác biệt với hệ quản trị cơ sở dữ liệu phân tán về

- mức độ tự trị: phản ánh trong các mô hình kiến trúc
- định nghĩa lược đồ khái niệm toàn cục: trong hệ phức hợp lược đồ khái niệm toàn cục chỉ là một tập con bao gồm một số cơ sở dữ liệu cục bộ mà mỗi hệ quản trị CSDL muốn dùng chung, còn trong hệ phân tán cơ sở dữ liệu toàn cục là hợp các cơ sở dữ liệu cục bộ.

a) Các mô hình sử dụng lược đồ khái niệm toàn cục

Trong phức hệ cơ sở dữ liệu, lược đồ khái niệm toàn cục GCS được định nghĩa bằng cách tích hợp các lược đồ ngoài của các cơ sở dữ liệu tự trị hoặc các thành phần của lược đồ khái niệm cục bộ của chúng, xem hình sau



kiến trúc phức hệ CSDL với một lược đồ khái niệm toàn cục

Người dùng của hệ quản trị cơ sở dữ liệu cục bộ sẽ định nghĩa khung nhìn riêng (LES) của họ trên cơ sở dữ liệu cục bộ và không cần thay đổi các ứng dụng hiện có nếu họ không truy xuất dữ liệu của cơ sở dữ liệu khác. Đây chính là khía cạnh tự trị của kiến trúc này.

Thiết kế lược đồ khái niệm toàn cục trong phức hệ cơ sở dữ liệu bao gồm việc tích hợp các lược đồ khái niệm cục bộ (ánh xạ đi từ dưới lên, từ lược đồ khái niệm cục bộ lên lược đồ khái niệm toàn cục, ngược lại với hệ phân tán) hoặc tích hợp các lược đồ ngoài cục bộ (ánh xạ đi theo chiều từ trên xuống).

Một khi đã thiết kế xong GCS, các khung nhìn trên lược đồ có thể định nghĩa cho người dùng cần truy xuất ở phạm vi toàn cục. Các lược đồ ngoại giới toàn cục GES và lược đồ khái niệm toàn cục GCS không nhất thiết sử dụng cùng một mô hình và cùng ngôn ngữ, chúng không cần xác định xem hệ thống *đồng chủng* hay *đa chủng*.

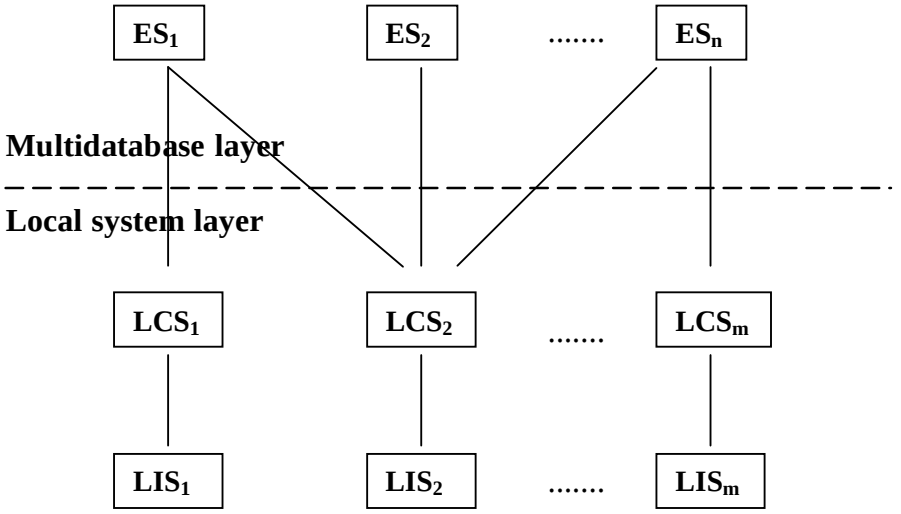
Trong trường hợp đa chủng, ta có hai lựa chọn cài đặt: *đơn ngôn* (unilingal) hoặc *đa ngôn* (multilingal).

Trong phức hệ cơ sở dữ liệu đơn ngôn, khi truy xuất toàn cục người dùng sử dụng chung một lược đồ ngoại giới toàn cục và một ngôn ngữ xử lý toàn cục.

Trong phức hệ cơ sở dữ liệu đa ngôn, khi truy xuất toàn cục mỗi người dùng sử dụng lược đồ ngoại giới toàn cục được định nghĩa bằng ngôn ngữ của hệ quản trị cơ sở dữ liệu cục bộ của mình và các câu vấn tin toàn cục cũng được tạo bằng ngôn ngữ của hệ quản trị cơ sở dữ liệu cục bộ.

b) Các mô hình không có lược đồ khái niệm toàn cục

Kiến trúc của phức hệ cơ sở dữ liệu không có lược đồ khái niệm toàn cục được trình bày trong hình sau



kiến trúc phức hệ CSDL không có lược đồ khái niệm toàn cục

Kiến trúc này có hai tầng: *tầng hệ thống cục bộ* và *tầng phức hệ CSDL* phía trên.

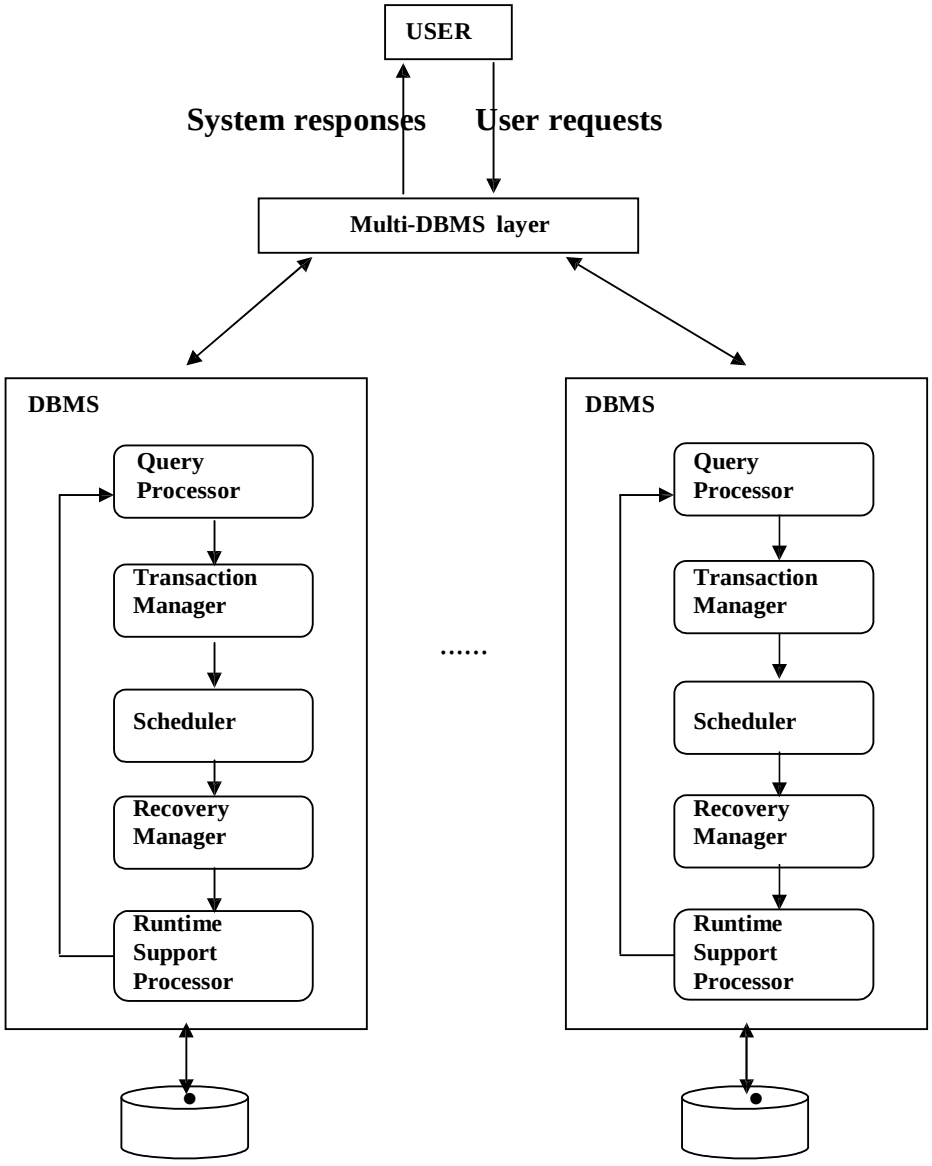
Tầng hệ thống cục bộ bao gồm một số hệ quản trị cơ sở dữ liệu, với chức năng là giới thiệu cho tầng phức hệ cơ sở dữ liệu các thành phần của cơ sở dữ liệu cục bộ có thể dùng chung với những cơ sở dữ liệu khác. Dữ liệu dùng chung này được trình bày qua lược đồ khái niệm cục bộ thực sự hoặc qua lược đồ ngoại giới cục bộ. Nếu có vấn đề đa chủng, mỗi lược đồ LCS_i có thể sử dụng mô hình dữ liệu khác nhau.

Tầng phức hệ cơ sở dữ liệu gồm các lược đồ ngoại giới, trong đó mỗi khung nhìn được định nghĩa trên một hay nhiều lược đồ khái niệm cục bộ. Vì vậy trách nhiệm cung cấp quyền truy xuất đến nhiều cơ sở dữ liệu (có thể đa chủng) được trao cho ánh xạ giữa lược đồ ngoại giới và lược đồ khái niệm cục bộ. Đây là điểm khác biệt cơ bản so với kiến trúc sử dụng lược đồ khái niệm toàn cục, trong đó quan hệ giữa lược đồ ngoại giới và lược đồ khái niệm cục bộ được thực hiện bởi các ánh xạ thông qua lược đồ khái niệm toàn cục.

Kiến trúc cơ sở dữ liệu liên bang cũng không sử dụng lược đồ khái niệm toàn cục. Trong hệ thống này mỗi hệ quản trị cơ sở dữ liệu cục bộ định nghĩa một

lược đồ xuất (export schema), trong đó định nghĩa dữ liệu muốn chia sẻ với các DBMS khác, mỗi ứng dụng truy xuất đến cơ sở dữ liệu toàn cục qua định nghĩa của *lược đồ nhập* (import schema), thực chất là hình ảnh ngoại giới toàn cục.

Các thành phần của hệ quản trị phức hợp khác biệt nhiều so với hệ quản trị cơ sở dữ liệu phân tán. ở đây có một tầng phần mềm chạy bên trên những hệ quản trị cơ sở dữ liệu riêng biệt và cung cấp cho người dùng những tiện ích để truy xuất nhiều cơ sở dữ liệu khác nhau. Tuỳ thuộc vào sự tồn tại hay không lược đồ khái niệm toàn cục và vấn đề đa chủng mà nội dung phần mềm sẽ thay đổi cho phù hợp.



các thành phần của phức hệ CSDL

1.4. Tổ chức thư mục toàn cục

Các vấn đề *tổ chức thư mục toàn cục* chỉ được đề cập đến trong các hệ phân tán và phức hệ có sử dụng lược đồ khái niệm toàn cục.

Thư mục toàn cục cũng là cơ sở dữ liệu chứa dữ liệu về các dữ liệu thực sự lưu trữ trong cơ sở dữ liệu (còn gọi là *meta dữ liệu* hay *siêu dữ liệu*). Vì thế những kỹ thuật thiết kế cơ sở dữ liệu phân tán cũng áp dụng cho việc quản lý thư mục. Như vậy thư mục có thể *toàn cục* đối với toàn bộ cơ sở dữ liệu hoặc *cục bộ* đối với từng vị trí. Nghĩa là có thể có một thư mục duy nhất chứa các thông tin về tất cả dữ liệu trong cơ sở dữ liệu, hoặc có một số thư mục, mỗi thư mục chứa thông tin được lưu ở các vị trí khác nhau. Trong trường hợp sau chúng ta có thể xây dựng hệ phân cấp thư mục để dễ dàng khi tìm kiếm hoặc cài đặt một chiến lược tìm kiếm phân tán cho phép trao đổi giữa các vị trí lưu trữ thư mục.

Vấn đề thứ hai liên quan đến vị trí chứa thư mục. Thư mục có thể được duy trì tập trung tại một vị trí hoặc phân tán đến một số vị trí. Giữ thư mục tại một vị trí làm cho việc quản lý được dễ dàng, nhưng có thể làm tăng tải trọng tại đó, gây ùn tắc lưu lượng thông báo ở vị trí đó. Ngược lại, phân tán thư mục trên nhiều vị trí làm giảm tải trọng tại một điểm nhưng sẽ làm cho vấn đề quản lý thư mục thêm phức tạp. Trong các phức hệ cơ sở dữ liệu, sự lựa chọn sẽ phụ thuộc vào vấn đề hệ thống có phân tán hay không. Nếu có, thư mục sẽ được phân tán, ngược lại nó được quản lý tập trung.

Vấn đề thứ ba là nhân bản thư mục. Có thể có một bản thư mục duy nhất hoặc nhiều bản. Có nhiều bản thư mục sẽ làm tăng độ tin cậy của hệ thống do khả năng truy xuất được một bản thư mục sẽ cao hơn. Hơn nữa thời gian trễ khi truy xuất thư mục sẽ giảm đi do ít xảy ra tranh chấp và khoảng cách đến các bản sao sẽ ngắn hơn. Tuy nhiên, việc duy trì cập nhật các bản sao cũng phức tạp và tốn kém. Vì vậy sự lựa chọn sẽ phụ thuộc vào môi trường hệ thống và phải cân bằng các yếu tố như thời gian đáp ứng, kích thước thư mục, khả năng của máy ở mỗi vị trí, yêu cầu khả tín, mức độ thay đổi của thư mục.

2. Thiết kế cơ sở dữ liệu phân tán

Thiết kế một hệ thống máy tính phân tán cần phải chọn *vị trí đặt dữ liệu* và *chương trình* trên một mạng máy tính, rất có thể phải kể luôn cả việc thiết kế mạng. Đối với hệ quản trị cơ sở dữ liệu phân tán cần phải thực hiện hai điều: *phân tán cơ sở dữ liệu* và *phân tán các chương trình ứng dụng* chạy trên hệ đó. ở đây chúng ta chỉ tập trung vào việc phân tán dữ liệu.

Việc tổ chức các hệ phân tán có thể được nghiên cứu dựa theo ba trục không gian

- *Mức độ chia sẻ dữ liệu* (level of sharing)
- *Kiểu mẫu truy xuất* (behavior of access pattern)
- *Mức độ hiểu biết về kiểu mẫu truy xuất*

(1) Theo mức độ chia sẻ có ba khả năng xảy ra:

- *Không chia sẻ dữ liệu*: mỗi ứng dụng và dữ liệu của nó thực thi tại một vị trí, không có trao đổi hoặc giao tiếp với những chương trình khác hoặc truy xuất dữ liệu ở những vị trí khác. Hình thức này đặc trưng cho các kết nối mạng ở thời kỳ sơ khai.
- *Chia sẻ dữ liệu*: tất cả chương trình đều được nhân bản cho mỗi vị trí, nhưng không nhân bản dữ liệu. Theo đấy các yêu cầu của người dùng được xử lý tại mỗi vị trí và dữ liệu cần thiết được chuyển đi trên mạng.
- *Chia sẻ dữ liệu – chương trình*: cả chương trình và dữ liệu được dùng chung. Nghĩa là chương trình nằm tại một vị trí có thể yêu cầu dịch vụ từ một chương trình nằm ở vị trí thứ hai, và đến lượt nó, chương trình này có thể truy xuất dữ liệu nằm tại vị trí thứ ba.

Ở đây cần phân biệt giữa *Chia sẻ dữ liệu* và *Chia sẻ dữ liệu - chương trình*, đặc biệt đối với hệ phân tán đa chủng. Trong môi trường đa chủng rất khó khăn, có khi không thể được, cho thực thi một chương trình trên một phần cứng khác và trong hệ điều hành khác.

(2) Theo kiểu mẫu truy xuất có hai kiểu lựa chọn:

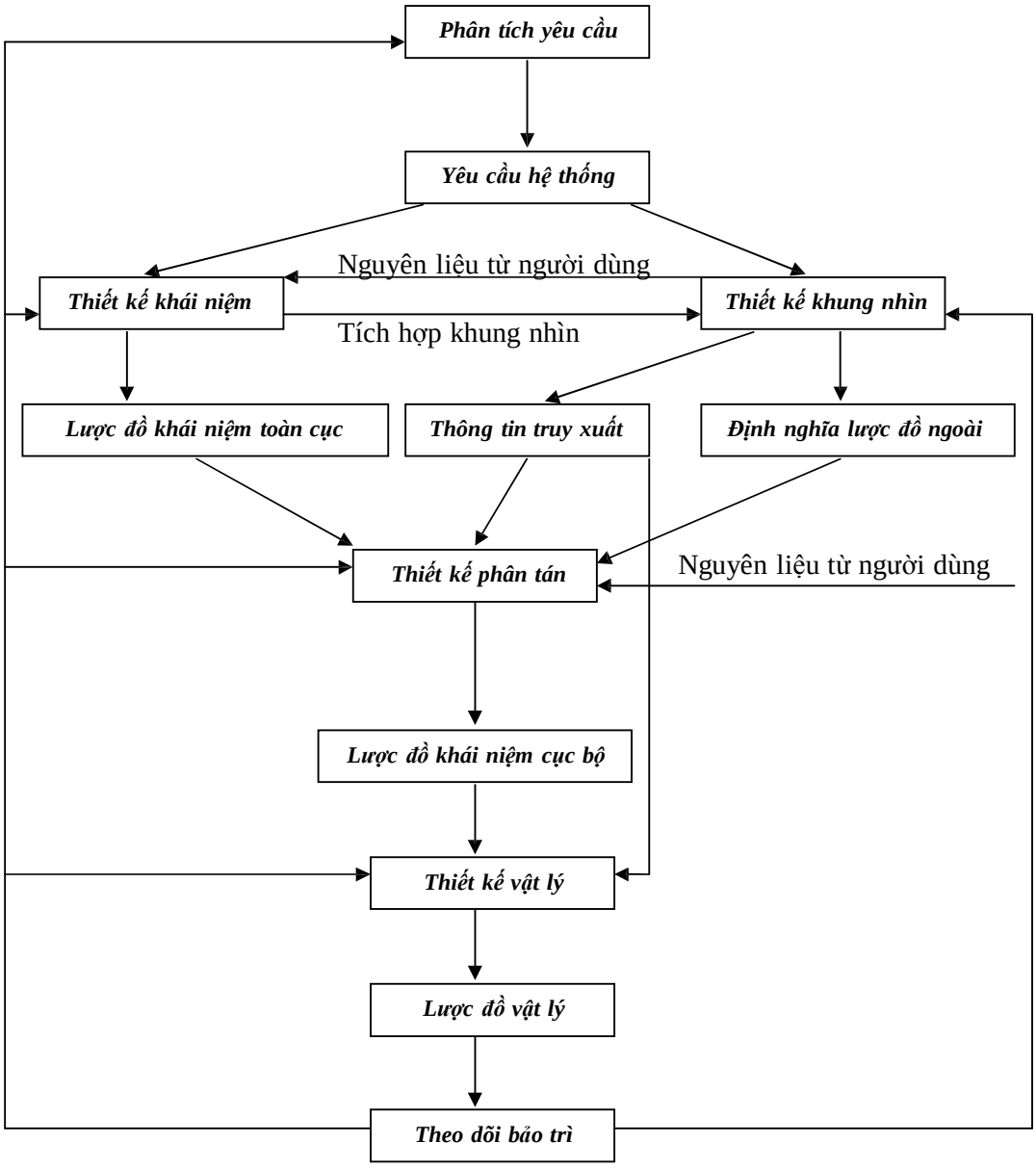
- Loại *tĩnh*, không thay đổi theo thời gian
- Loại *động*, thay đổi theo thời gian.

2.1. Các chiến lược thiết kế

Hai chiến lược chính trong việc thiết kế cơ sở dữ liệu phân tán là *tiếp cận từ trên xuống* và *tiếp cận từ dưới lên*. Trong thực tế rất hiếm các ứng dụng đơn giản để chỉ sử dụng một cách tiếp cận, vì vậy trong phần lớn thiết kế cả hai cách tiếp cần đều được áp dụng bổ sung nhau.

2.1.1. Quá trình thiết kế từ trên xuống

Sơ đồ quá trình thiết kế từ trên xuống biểu diễn trong hình sau:



Việc *phân tích yêu cầu* nhằm định nghĩa môi trường hệ thống và thu nhập các nhu cầu xử lý của tất cả người dùng, đồng thời cũng xác định *yêu cầu hệ thống.*

Hồ sơ ghi chép các yêu cầu là nguyên liệu cho hai hoạt động song song: *Thiết kế khung nhìn* (view design) và *Thiết kế khái niệm* (conceptual design).

Thiết kế khung nhìn định nghĩa các giao diện cho *người dùng đầu cuối* (end-user).

Thiết kế khái niệm là quá trình xem xét tổng thể đối tượng – xí nghiệp, nhằm xác định các loại thực thể và mối liên hệ giữa chúng với nhau. Ta có thể chia quá trình này thành 2 nhóm bao gồm các hoạt động liên quan tới nhau: *Phân tích thực*

thể (entity analysis) và *Phân tích chức năng* (functional analysis). Phân tích thực thể có liên quan đến việc xác định các thực thể, các thuộc tính và các mối liên hệ giữa chúng. Phân tích chức năng đề cập đến việc xác định các chức năng cơ bản có liên quan đến xí nghiệp cần được mô hình hoá. Kết quả của hai quá trình này cần được đối chiếu qua lại, giúp chúng ta biết được chức năng nào sẽ hoạt tác trên những thực thể nào.

Có sự liên hệ khăng khít giữa thiết kế khái niệm và thiết kế khung nhìn. Theo nghĩa nào đó thiết kế khái niệm được coi như là sự tích hợp các khung nhìn. Tuy nhiên mô hình khái niệm cần phải hỗ trợ không chỉ những ứng dụng hiện có mà còn cả những ứng dụng trong tương lai. Tích hợp khung nhìn nhằm đảm bảo các yêu cầu về thực thể và các mối liên hệ giữa các khung nhìn đều phải được bao quát trong lược đồ khái niệm.

Trong các hoạt động thiết kế khái niệm và thiết kế khung nhìn, người thiết kế cần phải đặc tả các thực thể dữ liệu và phải xác định các ứng dụng chạy trên cơ sở dữ liệu cũng như các thông tin thống kê về những ứng dụng này. Thông tin thống kê bao gồm đặc tả về tần số ứng dụng, khối lượng thông tin khác nhau,...

Lược đồ khái niệm toàn cục GCS và thông tin về kiểu mẫu truy xuất thu được trong thiết kế khung nhìn sẽ là nguyên liệu (input) cho bước *thiết kế phân tán*. Mục tiêu của giai đoạn này là thiết kế các lược đồ khái niệm cục bộ LCS bằng cách phân tán các thực thể cho các vị trí của hệ thống phân tán.

Ta chia quan hệ thành nhiều quan hệ nhỏ hơn gọi là các *mảnh* (*fragment*) và phân tán các mảnh này. Hoạt động thiết kế phân tán gồm hai bước: *Phân mảnh* (*fragmentation*) và *cấp phát* (*allocation*). Ta sẽ thảo luận về vấn đề này trong các phần sau.

Thiết kế vật lý là ánh xạ lược đồ khái niệm cục bộ sang các thiết bị lưu trữ vật lý có sẵn tại các vị trí tương ứng. Nguyên liệu cho quá trình này là lược đồ khái niệm cục bộ và thông tin về kiểu mẫu truy xuất các mảnh.

Hoạt động phát triển và thiết kế luôn là quá trình liên tục, đòi hỏi theo dõi hiệu chỉnh thường xuyên. Vì thế chúng ta đưa vấn đề quan sát và theo dõi như một hoạt động chính trong quá trình này. Cần chú ý rằng chúng ta không chỉ theo dõi vấn đề cài đặt cơ sở dữ liệu, mà còn quan sát theo dõi tính thích hợp của các khung nhìn của người dùng. Kết quả này có tác dụng phản hồi, tạo cơ sở cho việc tái thiết kế về sau.

2.1.2. Quá trình thiết kế từ dưới lên

Thiết kế từ trên xuống thích hợp cho những cơ sở dữ liệu được thiết kế từ đầu. Tuy nhiên trong thực tế cũng có khi đã có sẵn một số cơ sở dữ liệu, và chúng ta phải tích hợp chúng thành một cơ sở dữ liệu chung. Tiếp cận từ dưới lên sẽ thích hợp cho tình huống này. Khởi điểm của thiết kế từ dưới lên là các lược đồ khái niệm cục bộ, sẽ phải được tích hợp thành lược đồ khái niệm toàn cục.

2.2. Các vấn đề thiết kế phân tán

Trong mục này chúng ta sẽ trả lời các câu hỏi sau:

- Tại sao cần phân mảnh
- Làm thế nào để thực hiện phân mảnh
- Phân mảnh nên thực hiện đến mức độ nào
- Cách thức kiểm tra tính đúng đắn của phân mảnh
- Cách thức cấp phát dữ liệu
- Những thông tin nào cần thiết cho phân mảnh và cấp phát

2.2.1. Các lý do phân mảnh

Trước tiên, khung nhìn của các ứng dụng thường chỉ là tập con của quan hệ. Vì thế đơn vị truy xuất không phải toàn bộ quan hệ mà chỉ là tập con của quan hệ. Kết quả là xem tập con của quan hệ là đơn vị phân tán là thích hợp.

Thứ hai là, nếu các ứng dụng có các khung nhìn được định nghĩa trên một quan hệ cho trước lại nằm tại những vị trí khác nhau thì chỉ có hai cách chọn lựa với đơn vị phân tán là toàn bộ quan hệ, khi không có phân mảnh: Hoặc quan hệ không được nhân bản mà được lưu ở một vị trí, hoặc quan hệ được nhân bản cho tất cả hoặc một số vị trí có chạy ứng dụng. Chọn lựa đầu gây ra một số lượng lớn truy xuất không cần thiết đến dữ liệu ở xa. Còn chọn lựa sau có thể dẫn đến nhân bản không cần thiết, gây khó khăn khi cập nhật và lãng phí không gian lưu trữ.

Cuối cùng việc phân rã quan hệ thành nhiều mảnh, mỗi mảnh xử lý như một đơn vị, sẽ cho phép thực hiện nhiều giao dịch đồng thời. Việc phân mảnh quan hệ cho phép thực hiện song song câu vấn tin, bằng cách chia nó thành một tập câu vấn tin con hoạt tác trên các mảnh. Vì thế việc phân mảnh sẽ làm tăng mức độ hoạt động đồng thời và kéo theo tăng lưu lượng hoạt động của hệ thống. Kiểu hoạt động này gọi là *đồng thời nội vấn tin (intraquery concurency)*, sẽ được phân tích trong các phần sau.

2.2.2. Các kiểu phân mảnh

Có hai kiểu phân mảnh: *phân mảnh theo chiều dọc* và *phân mảnh theo chiều ngang*.

◊ *Ví dụ*

Chúng ta sử dụng lược đồ cơ sở dữ liệu đã phát triển trong chương trước. Ta thêm vào lược đồ PROJ thuộc tính LOC (vị trí) để chỉ nơi thực hiện dự án. Sau đây là một thể hiện cơ sở dữ liệu sẽ được dùng:

EMP

ENO	*ENAME*	*TITLE*
E1	J.Doe	Elect.Eng.
E2	M.Smith	Syst.Anal.
E3	A.Lee	Mech.Eng.
E4	J.Miller	Programmer
E5	B.Casey	Syst.Anal.
E6	L.Chu	Elect.Eng.
E7	R.David	Mech.Eng.
E8	J.Jones	Syst.Anal.

ASG

ENO	*PNO*	*RESP*	*DUR*
E1	P1	Manager	12
E2	P1	Analyst	24
E2	P2	Analyst	6
E3	P3	Consultant	10
E3	P4	Engineer	48
E4	P2	Programmer	18
E5	P2	Manager	24
E6	P4	Manager	48
E7	P3	Engineer	36
E8	P3	Manager	40

PROJ

PNO	*PNAME*	*BUDGET*	*LOC*
P1	Instrumentation	150000	Montreal
P2	Database Develop.	135000	New York
P3	CAD/CAM	250000	New York
P4	Maintenance	310000	Paris

PAY

TITLE	*SAL*
Elect.Eng.	40000
Syst.Anal.	34000
Mech.Eng.	27000
Programmer	24000

Trong hình sau trình bày quan hệ PROJ được tách ngang thành 2 quan hệ $PROJ_1$ chứa các thông tin về dự án có kinh phí dưới 200000 USD, và $PROJ_2$ chứa các thông tin về dự án có kinh phí lớn hơn 200000 USD.

$PROJ_1$

PNO	*PNAME*	*BUDGET*	*LOC*
P1	Instrumentation	150000	Montreal
P2	Database Develop.	135000	New York

$PROJ_2$

PNO	*PNAME*	*BUDGET*	*LOC*
P3	CAD/CAM	250000	New York
P4	Maintenance	310000	Paris

Còn trong hình sau trình bày quan hệ PROJ được tách dọc thành 2 quan hệ $PROJ_1$ chứa các thông tin về kinh phí dự án, và $PROJ_2$ chứa các thông tin về tên và vị trí dự án.

$PROJ_1$

PNO	*BUDGET*
P1	150000
P2	135000
P3	250000
P4	310000

$PROJ_2$

PNO	*PNAME*	*LOC*
P1	Instrumentation	Montreal
P2	Database Develop.	New York
P3	CAD/CAm	New York
P4	Maintenamce	Paris

Việc phân mảnh có thể lồng ghép, vừa phân mảnh ngang vừa phân mảnh dọc, thành *phân mảnh tổng hợp* (*hybrid fragmentation*).

2.2.3. Các qui tắc phân mảnh đúng đắn

Có ba qui tắc trong khi phân mảnh đảm bảo cơ sở dữ liệu sẽ không thay đổi ngữ nghĩa khi phân mảnh.

1) *Tính đầy đủ* (*completeness*): Cho quan hệ **r** bất kỳ. Giả sử **r** được phân rã thành các mảnh. Khi đó tính đầy đủ yêu cầu mỗi mục dữ liệu trong **r** cũng phải được lưu trữ trong một hoặc vài mảnh nào đó.

2) *Tính tái thiết* (*reconstruction*): Cho quan hệ **r** bất kỳ. Giả sử **r** được phân rã thành các mảnh $\mathbf{r_1}$, ..., $\mathbf{r_n}$. Khi đó tính tái thiết yêu cầu "hợp" các phân mảnh của quan hệ **r** trả lại đầy đủ dữ liệu ban đầu của quan hệ **r**. Khái niệm "hợp" ở đây là toán tử quan hệ Δ sao cho

$$\mathbf{r} = \Delta_{i=1}^{n} r_i$$

Toán tử Δ thay đổi tuỳ theo từng loại phân mảnh.

Khả năng tái thiết một quan hệ từ các mảnh của nó cũng phải đảm bảo rằng các ràng buộc định nghĩa theo phụ thuộc dữ liệu sẽ được bảo toàn.

3) *Tính tách biệt* (*disjointness*): Cho quan hệ **r** bất kỳ. Giả sử **r** được phân rã thành các mảnh $\mathbf{r_1}$, ..., $\mathbf{r_n}$. Khi đó tính tách biệt yêu cầu một mục dữ liệu *d* nào đó một khi đã xuất hiện trong mảnh $\mathbf{r_i}$ thì sẽ không xuất hiện trong mảnh $\mathbf{r_k}$ khác. Tiêu chuẩn này đảm bảo các mảnh ngang sẽ tách biệt nhau. Còn trong phân mảnh dọc thì các thuộc tính khoá chính phải được lặp lại trong mỗi mảnh, vì vậy tính tách biệt chỉ áp dụng với các thuộc tính không khoá.

2.2.4. Các kiểu cấp phát

Giả sử cơ sở dữ liệu đã được phân mảnh thích hợp và cần phải quyết định cấp phát các mảnh cho các vị trí trên mạng. Khi dữ liệu được cấp phát, nó có thể được nhân bản hoặc chỉ duy trì một bản duy nhất.

Một cơ sở dữ liệu không nhân bản, gọi là *cơ sở dữ liệu phân hoạch*, có chứa các mảnh được cấp phát cho các vị trí, trong đó chỉ tồn tại một bản duy nhất cho mỗi mảnh trên mạng.

Kiểu *cơ sở dữ liệu nhân bản* có hai dạng:

- *Cơ sở dữ liệu nhân bản hoàn toàn*, trong đó toàn bộ cơ sở dữ liệu đều có bản sao ở mỗi vị trí.
- *Cơ sở dữ liệu nhân bản một phần*, trong đó các mảnh được phân tán đến các vị trí, mỗi mảnh có thể có nhiều bản sao nằm ở các vị trí khác nhau. Số lượng các bản sao của các mảnh có thể là tham số (input) cho các thuật toán cấp phát (allocation algorithm) hoặc là biến quyết định (dicision variable) mà giá trị của nó được xác định bằng thuật toán này.

Lý do nhân bản là nhằm đảm bảo được độ tin cậy và hiệu quả cho các câu vấn tin chỉ đọc. Nếu có nhiều bản sao của một mục dữ liệu thì chúng ta vẫn có cơ hội truy xuất được dữ liệu đó ngay cả khi hệ thống có sự cố. Hơn nữa các câu vấn tin chỉ đọc truy xuất đến cùng một mục dữ liệu có thể cho thực hiện song song vì các bản sao có mặt tại nhiều vị trí.

Ngược lại, trong cơ sở dữ liệu nhân bản các câu vấn tin cập nhật có thể gây nhiều rắc rối vì phải đảm bảo các bản sao phải được cập nhật chính xác.

Vì vậy quyết định nhân bản cần được cân nhắc và phụ thuộc vào tỉ lệ giữa các câu vấn tin chỉ đọc và câu vấn tin cập nhật. Quyết định này hầu như ảnh hưởng đến tất cả các thuật toán của hệ quản trị cơ sở dữ liệu phân tán và các chức năng kiểm soát khác.

2.3. Phương pháp phân mảnh

Trong phần này chúng ta sẽ bàn đến các chiến lược và thuật toán phân mảnh. Có hai chiến lược phân mảnh cơ bản: *phân mảnh ngang* (*horizontal fragmentation*) và *phân mảnh dọc* (*vertical fragmentation*). Ngoài ra còn có khả năng phân mảnh hỗn hợp.

2.3.1. Phân mảnh ngang

Phân mảnh ngang chia quan hệ theo các bộ. Mỗi mảnh là một tập con của quan hệ. Có hai loại phân mảnh ngang: *phân mảnh nguyên thuỷ* (*primary horizontal fragmentation*), thực hiện dựa trên các vị từ định nghĩa trên chính

quan hệ đó, và *phân mảnh dẫn xuất* (*derived horizontal fragmentation*), dựa trên các vị từ định nghĩa trên quan hệ khác.

Trước khi thực hiện phân mảnh, chúng ta cần thu thập thông tin cần thiết.

a) Yêu cầu thông tin

• *Thông tin về cơ sở dữ liệu*

Thông tin này bao gồm lược đồ khái niệm toàn cục, các liên kết giữa các quan hệ, đặc biệt là phép nối.

Trong mô hình quan hệ, các mối liên hệ được biểu thị bằng các quan hệ. Tuy nhiên trong các mô hình khác, như mô hình thực thể-quan hệ, các mối liên hệ được biểu diễn tường minh. Với mục đích thiết kế phân tán, các mối liên hệ cũng được mô hình hoá trong bộ khung quan hệ. Theo cách này chúng ta sẽ vẽ các đường nối (L) có hướng giữa các quan hệ (R, S) ràng buộc nhau qua phép đẳng nối dạng

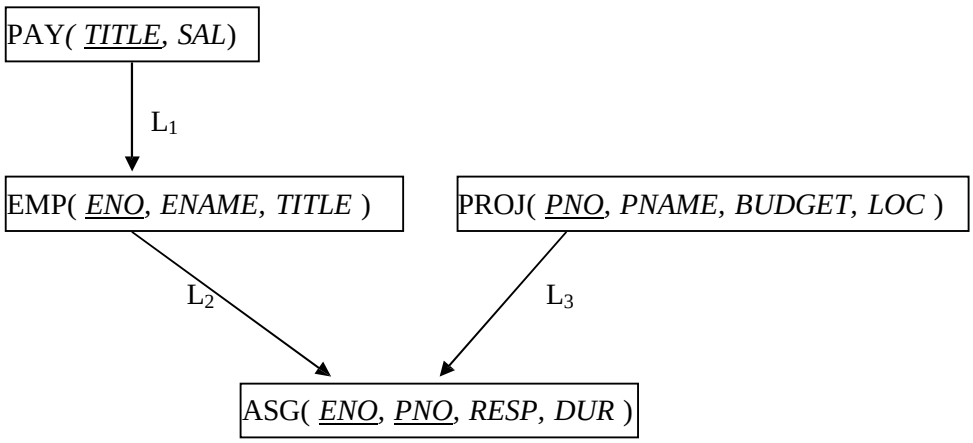
$$R \xrightarrow{L} S$$

trong đó R gọi là quan hệ chủ, S gọi là quan hệ thành viên. Người ta dùng hàm *owner* và *member* để phân biệt các quan hệ này:

$$owner(L) = R \quad \text{và} \quad member(L) = S$$

◊ *Ví dụ*

Hình sau trình bày một cách biểu diễn các đường nối giữa các quan hệ PAY, EMP, PROJ và ASG.



ở đây có ba đường nối L_1, L_2, L_3. Ta có

$owner(L_1) = PAY$ và $member(L_1) = EMP$

$owner(L_2) = EMP$ và $member(L_2) = ASG$

$owner(L_3) = PROJ$ và $member(L_3) = ASG$

Thông tin định lượng về cơ sở dữ liệu, tức là *lực lượng* (*cardinality*) của mỗi quan hệ R, ký hiệu *card*(R).

• *Thông tin về ứng dụng*

Yêu cầu thông tin định tính lẫn định lượng về các ứng dụng. Thông tin định tính định hướng cho hoạt động phân mảnh, còn thông tin định lượng sử dụng cho các mô hình cấp phát.

Những thông tin định tính cơ bản gồm các vị từ được dùng trong các câu vấn tin. Nếu không thể phân tích được hết tất cả các ứng dụng để xác định những vị từ này thì ít nhất cũng phải nghiên cứu được các ứng dụng quan trọng nhất. Một hướng dẫn quan trọng, gọi là *qui tắc 80/20*, là "*20% câu vấn tin sẽ chiếm đến 80% truy xuất dữ liệu*".

Cho quan hệ $R(A_1, ..., A_n)$, trongđó A_i là một thuộc tính được định nghĩa trên miền giá trị D_i. Một *vị từ đơn giản* p được định nghĩa trên R có dạng:

$$p: A_i \, \theta \; Value$$

Trong đó $\theta \in \{ =, <, \leq, \neq, >, \geq \}$ và *Value* là giá trị được chọn từ miền D_i.

◊ *Ví dụ*

Xét bảng quan hệ PROJ. Các biểu thức

$$PNAME = \text{"Maintenance"} \quad \text{và} \quad BUDGET \leq 200000$$

là các vị từ đơn giản.

Các câu vấn tin thường chứa nhiều vị từ phức tạp, là tổ hợp các vị từ đơn giản. Một tổ hợp cần đặc biệt chú ý được gọi là *tiểu hạng* (minterm predicate), là *hội* (conjuction) của các vị từ đơn giản. Bởi vì ta luôn có thể biến đổi một biểu thức bool thành *dạng chuẩn hội* (conjuctive normal form), việc sử dụng tiểu hạng trong thuật toán thiết kế không làm mất tính tổng quát.

Cho một tập các vị từ đơn giản $Pr = \{p_1, ..., p_m\}$ trên quan hệ R. Tập các tiểu hạng $M=\{m_1, ..., m_z\}$ gồm các vị từ dạng

$$m_j = \bigwedge_{p_k \in Pr} p_k^* \text{ , với } 1 \leq k \leq m, \; 1 \leq j \leq z$$

trong đó $p_k^* = p_k$ hoặc $p_k^* = \neg p_k$, $\forall k=1,...,m$. Như vậy mỗi vị từ đơn giản có thể xuất hiện trong vị từ tiểu hạng dạng khẳng định hoặc phủ định.

◊ *Ví dụ*

Xét bảng quan hệ PAY. Sau đây là các vị từ đơn giản định nghĩa trên PAY.

p_1 : *TITLE* = "Elect. Eng."
p_2 : *TITLE* = "Syst. Anal."
p_3 : *TITLE* = "Mech. Eng."
p_4 : *TITLE* = "Programmer"
p_5 : *SAL* ≤ 30000
p_6 : *SAL* > 30000

Từ các vị từ đơn giản trên có thể định nghĩa các vị từ tiểu hạng sau

m_1 : *TITLE* = "Elect. Eng." ∧ *SAL* ≤ 30000
m_2 : *TITLE* = "Elect. Eng." ∧ *SAL* > 30000
m_3 : ¬(*TITLE* = "Elect. Eng.") ∧ *SAL* ≤ 30000
m_4 : ¬(*TITLE* = "Elect. Eng.") ∧ *SAL* > 30000
m_5 : *TITLE* = "Programmer" ∧ *SAL* ≤ 30000
m_6 : *TITLE* = "Programmer" ∧ *SAL* > 30000

Theo những thông tin định lượng về các ứng dụng, chúng ta cần biết hai tập dữ liệu.

(1) *Độ tuyển tiểu hạng* (*minterm selectivity*): Số lượng các bộ quan hệ sẽ được truy xuất bởi câu vấn tin được đặc tả theo một vị từ tiểu hạng đã cho. Ta ký hiệu độ tuyển của vị từ tiểu hạng *m* là *sel(m).*

(2) *Tần số truy xuất* (*access frequency*): tần số ứng dụng truy xuất dữ liệu. Cho

$$Q = \{ q_1, q_2, ..., q_k \}$$

là tập các câu vấn tin, ký hiệu $acc(q_i)$ biểu thị tần số truy xuất của q_i trong một khoảng thời gian đã cho. Ta cũng ký hiệu tần số truy xuất của một vị từ tiểu hạng *m* là *acc(m).*

b) Phân mảnh ngang nguyên thuỷ

Phân mảnh ngang nguyên thuỷ được định nghĩa bằng một phép toán chọn trên các quan hệ chủ của một lược đồ CSDL.

Cho quan hệ **r**, các mảnh ngang của **r** là các quan hệ con r_i, i = 1, ..., k, với

$$\mathbf{r_i} = \sigma_{F_i}(r), \text{ i = 1, ..., k}$$

trong đó F_i , i = 1, ..., k, là công thức chọn để có mảnh $\mathbf{r_i}$.

◊ *Ví dụ*

Phân rã quan hệ PROJ thành các mảnh ngang $PROJ_1$ và $PROJ_2$ trong ví dụ trên có thể được định nghĩa như sau:

$$PROJ_1 = \sigma_{BUDGET \le 200000}(PROJ)$$
$$PROJ_2 = \sigma_{BUDGET > 200000}(PROJ)$$

Ta cũng có thể định nghĩa các mảnh ngang sau đây

$$PRJ_1 = \sigma_{LOC="Montreal"}(PROJ)$$
$$PRJ_2 = \sigma_{LOC="New\ York"}(PROJ)$$
$$PRJ_3 = \sigma_{LOC="Paris"}(PROJ)$$

PRJ_1

PNO	*PNAME*	*BUDGET*	*LOC*
P1	Instrumentation	150000	Montreal

PRJ_2

PNO	*PNAME*	*BUDGET*	*LOC*
P2	Database Develop.	135000	New York
P3	CAD/CAM	250000	New York

PRJ_3

PNO	*PNAME*	*BUDGET*	*LOC*
P4	Maintenamce	310000	Paris

Bây giờ ta có thể định nghĩa một mảnh ngang chặt chẽ hơn. Một mảnh ngang $\mathbf{r_i}$ của quan hệ **r** có chứa tất cả các bộ của **r** thoả vị từ tiểu hạng m_i. Như vậy, cho tập M các vị từ tiểu hạng, số mảnh ngang bằng số các vị từ tiểu hạng trong tập M. Tập các mảnh ngang này gọi là tập các *mảnh tiểu hạng* (minterm fragment).

Theo như đã phân tích, việc định nghĩa các mảnh ngang phụ thuộc vào các vị từ tiểu hạng. Vì thế bước đầu tiên của mọi thuật toán phân mảnh là xác định tập các vị từ đơn giản sẽ cấu thành các vị từ tiểu hạng.

Các vị từ đơn giản cần có các tính chất *đầy đủ* và *cực tiểu*.

• *Tính đầy đủ*. Tập các vị từ đơn giản *Pr* gọi là *đầy đủ* nếu và chỉ nếu xác suất mỗi ứng dụng truy xuất đến một bộ bất kỳ thuộc về một mảnh tiểu hạng nào đó được định nghĩa theo *Pr* đều bằng nhau.

◊ *Ví dụ*. Xét phân mảnh PRJ_1, PRJ_2, PRJ_3 ở ví dụ trước. Nếu ứng dụng duy nhất truy xuất PROJ muốn truy xuất các bộ theo vị trí, thì tập vị từ (ứng dụng)

$$Pr = \{LOC=\text{"Montreal"}, LOC=\text{"New York"}, LOC=\text{"Paris"}\}$$

là đầy đủ bởi vì mỗi bộ của mỗi mảnh PRJ_i đều có xác suất truy xuất như nhau.

Tuy nhiên, nếu có ứng dụng thứ hai chỉ truy xuất các bộ dự án có ngân sách nhỏ hơn 200000 USD, thì tập vị từ *Pr* không còn đầy đủ nữa. Một số bản ghi trong mỗi mảnh $PROJ_i$ được ứng dụng thứ hai truy xuất với xác suất cao hơn.

Để cho tập vị từ đầy đủ ta phải thêm các vị từ $BUDGET \leq 200000$, $BUDGET > 200000$ vào *Pr*, tức là

$$Pr = \{LOC=\text{"Montreal"}, LOC=\text{"New York"}, LOC=\text{"Paris"}, BUDGET \leq 200000, BUDGET > 200000\}$$

Lý do cần phải đảm bảo tính đầy đủ là vì các mảnh thu được theo tập vị từ đầy đủ sẽ nhất quán về mặt logic do tất cả chúng đều thoả vị từ tiểu hạng. Chúng cũng đồng nhất về mặt thống kê theo cách mà các ứng dụng truy xuất chúng. Vì vậy chúng ta sẽ sử dụng tập vị từ đầy đủ làm cơ sở cho phân mảnh ngang nguyên thủy.

• *Tính cực tiểu*

Cho tập các vị từ đơn giản *Pr*. Ta nói một vị từ là *có liên đới* (*relevant*) trong việc xác định phân mảnh, nếu nó ảnh hưởng đến việc mảnh f nào đó bị phân thành các mảnh con f_1 và f_2 thì phải có ít nhất một ứng dụng truy xuất đến f_1 và f_2 theo cách khác nhau.

Tập *Pr* gọi là *cực tiểu* nếu mọi vị từ trong nó là *có liên đới* và không có vị từ tương đương.

◊ *Ví dụ*. Tập vị từ *Pr* trong ví dụ trước là đầy đủ và cực tiểu. Tuy nhiên nếu chúng ta thêm vị từ

$$PNAME=\text{"Instrumentation"}$$

vào *Pr*, thì *Pr* không còn là cực tiểu nữa bởi vì vị từ trên không có liên đới ứng với *Pr*. Không có ứng dụng truy xuất khác nhau đến các mảnh ảnh hưởng bởi vị từ trên .

Bây giờ chúng ta sẽ trình bày một thuật toán lặp sinh ra một tập các vị từ đầy đủ và cực tiểu. Ta sẽ sử dụng qui tắc cơ bản về tính đầy đủ và cực tiểu gọi tắt là qui tắc 1.

♦ *Qui tắc 1*. Một quan hệ hoặc một mảnh được phân hoạch thành ít nhất hai phần và chúng được truy xuất khác nhau bởi một ứng dụng.

◊ Ký hiệu *f(p)* là mảnh sinh bởi vị từ *p* và *F* là tập các mảnh, f_k là mảnh sinh bởi vị từ p_k.

• ***Thuật toán: COM_MIN***

Đầu vào: Quan hệ **r** và tập các vị từ đơn giản *Pr*.
Đầu ra: Tập các vị từ *Pr'* đầy đủ và cực tiểu.
Khai báo: *F* là tập các mảnh tiểu hạng.
begin
Tìm vị từ $p \in Pr$ sao cho *p* phân hoạch **r** theo qui tắc 1
$Pr' \leftarrow p$
$Pr \leftarrow Pr - \{p\}$
$F \leftarrow f\{= f(p)\}$
repeat
begin
Tìm vị từ $p \in Pr$ sao cho *p* phân hoạch một mảnh f_k nào đó của *F* theo qui tắc 1 :
$Pr' \leftarrow Pr' \cup \{p\}$
$Pr \leftarrow Pr - \{p\}$
$F \leftarrow F \cup \{f\}$
if tồn tại $p_k \in Pr'$ không liên đới hoặc có vị từ tương đương **then**
begin
$Pr' \leftarrow Pr' - \{p_k\}$
$F \leftarrow F - \{f_k\}$
end
end
until *Pr'* đầy đủ
end. {COM_MIN}

Bước thứ hai trong quá trình thiết kế phân mảnh ngang nguyên thuỷ là suy dẫn ra các tập vị từ tiểu hạng có thể được định nghĩa trên các vị từ trong *Pr'*. Các vị từ tiểu hạng này xác định các mảnh "ứng cử viên" cho bước cấp phát.

Việc xác định các vị từ tiểu hạng không khó. Khó khăn chính là tập này rất lớn (thực sự chúng tỉ lệ hàm mũ theo số lượng vị từ đơn giản), cần phải giảm số lượng.

Bước thứ ba là loại bỏ một số mảnh vô nghĩa. Điều này được thực hiện bằng cách xác định những vị từ mâu thuẫn với tập các *phép kéo theo*, kí hiệu là I.

Chẳng hạn, nếu $Pr' = \{p_1, p_2\}$, trong đó

$$p_1 : att = value1 \quad \& \quad p_2 : att = value2$$

và miền giá trị của thuộc tính *att* là {*value1*,*value2*}. Như vậy sẽ phát sinh hai phép kéo theo:

$$i_1 : p_1 \Rightarrow \neg p_2 \qquad \text{và} \qquad i_2 : \neg p_1 \Rightarrow p_2$$

Bốn vị từ tiểu hạng của *Pr'* được định nghĩa như sau:

$$m_1 : (att=value1) \wedge (att=value2)\ m_2 : (att=value1) \wedge \neg(att=value2)\ m_3 : \neg(att=value1) \wedge (att=value2)\ m_4 : \neg(att=value1) \wedge \neg(att=value2)$$

Trong trường hợp này các vị từ tiểu hạng m_1 và m_4 mâu thuẫn với các phép kéo theo i_1 và i_2 , và vì thế chúng bị loại khỏi tập các vị từ tiểu hạng M.

Thuật toán phân mảnh ngang nguyên thuỷ được trình bày như sau:

- ***Thuật toán: PHORIZONTAL***

Đầu vào: Quan hệ **r** và tập các vị từ đơn giản *Pr*.
Đầu ra: Tập các vị từ tiểu hạng M.
begin
Đặt *Pr'*:=COM_MIN(**r**, *Pr*);
Xác định tập M các vị từ tiểu hạng;
Xác định tập I các phép kéo theo giữa các vị từ trong *Pr'*;
Loại vị từ tiểu hạng mâu thuẫn:
for mỗi vị từ tiểu hạng $m \in$ M do
if *m* mâu thuẫn với I **then**
M $\leftarrow$ M − {*m*} { loại *m* khỏi M}
End-if
End-for
End.

◊ *Ví dụ*

Xét thiết kế lược đồ CSDL: EMP, ASG, PROJ, PAY cho ở phần trên. Có hai quan hệ cần phân mảnh ngang nguyên thuỷ là PAY và PROJ.

Giả sử có một ứng dụng truy xuất PAY, ứng dụng này kiểm tra thông tin lương và xác định số lương sẽ tăng. Giả sử rằng các mẩu tin nhân viên được quản lý ở hai nơi. Một nơi xử lý các mẩu tin có lương ≤ 30000 USD, và nơi khác xử lý các mẩu tin của nhân viên có lương >30000 USD. Vì thế câu vấn tin được sử dụng ở cả hai nơi.

Tập vị từ đơn giản được sử dụng để phân hoạch PAY là

$$p_1 : \text{SAL} \leq 30000 \qquad \& \qquad p_2 : \text{SAL} > 30000$$

Từ đó ta có tập vị từ đơn giản khởi đầu là

$$Pr = \{p_1, p_2\}$$

Áp dụng thuật toán COM_MIN ta có tập khởi đầu *Pr*' = $\{p_1\}$. Đây là tập đầy đủ và cực tiểu vì p_2 không phân hoạch mảnh $f_1 = f(p_1)$ theo qui tắc 1. Chúng ta có thể tạo ra các vị từ tiểu hạng cho tập M:

$$m_1 : (\text{SAL} \leq 30000) = p_1 \quad \text{và} \quad m_2 : \neg(\text{SAL} \leq 30000) = (\text{SAL} > 30000) = p_2$$

Sau đó ta định nghĩa hai mảnh $F_{PAY} = \{\text{PAY}_1, \text{PAY}_2\}$ theo M:

PAY_1

TITLE	*SAL*
Mech.Eng.	27000
Programmer	24000

PAY_2

TITLE	*SAL*
Elect.Eng.	40000
Syst.Anal.	34000

Bây giờ ta xét quan hệ PROJ. Giả sử có hai ứng dụng.

Ứng dụng thứ nhất được đưa ra tại ba vị trí và cần tìm tên và ngân sách của các dự án khi biết vị trí. Theo ký pháp SQL, câu vấn tin được viết là

```
SELECT   PNAME, BUDGET
FROM     PROJ
WHERE    LOC = Value
```

Đối với ứng dụng này, các vị từ đơn giản có thể dùng là

$$p_1 : LOC = \text{"Montreal"}, \quad p_2 : LOC = \text{"New York"}, \quad p_3 : LOC = \text{"Paris"}$$

Ứng dụng thứ hai được đưa ra tại hai vị trí và liên quan tới việc điều hành dự án. Những dự án có ngân sách ≤ 200000 USD được quản lý tại một vị trí, còn những dự án có ngân sách > 200000 USD được quản lý tại vị trí thứ hai. Vì thế các vị từ đơn giản phải được sử dụng để phân mảnh theo ứng dụng thứ hai là

$$p_4 : BUDGET \leq 200000 \quad \text{và} \quad p_5 : BUDGET > 200000$$

Nếu kiểm tra bằng thuật toán COM_MIN, tập

$$\text{Pr'} = \{p_1, p_2, p_3, p_4, p_5\}$$

rõ ràng là đầy đủ và cực tiểu.

Dựa trên *Pr*' chúng ta có thể định nghĩa sáu vị từ tiểu hạng sau tạo nên M.

$$m_1 : (LOC = \text{"Montreal"}) \wedge (BUDGET \leq 200000)$$
$$m_2 : (LOC = \text{"Montreal"}) \wedge (BUDGET > 200000)$$

m_3 : (*LOC* = "New York") $\wedge$ (*BUDGET* $\leq$ 200000)
m_4 : (*LOC* = "New York") $\wedge$ (*BUDGET* > 200000)
m_5 : (*LOC* = "Paris") $\wedge$ (*BUDGET* $\leq$ 200000)
m_6 : (*LOC* = "Paris") $\wedge$ (*BUDGET* > 200000)

Đây không phải là tất cả các vị từ tiểu hạng có thể được tạo ra. Chẳng hạn có thể định nghĩa vị từ

$$p_1 \wedge p_2 \wedge p_3 \wedge p_4 \wedge p_5$$

Tuy nhiên, các phép kéo theo hiển nhiên là

i_1 : $p_1 \Rightarrow \neg p_2 \wedge \neg p_3$
i_2 : $p_2 \Rightarrow \neg p_1 \wedge \neg p_3$
i_3 : $p_3 \Rightarrow \neg p_1 \wedge \neg p_2$
i_4 : $p_4 \Rightarrow \neg p_5$
i_5 : $p_5 \Rightarrow \neg p_4$
i_6 : $\neg p_4 \Rightarrow p_5$
i_7 : $\neg p_5 \Rightarrow p_4$

cho phép loại bỏ những vị từ tiểu hạng khác và chúng ta còn lại m_1 đến m_6.
Kết quả của phân mảnh ngang nguyên thuỷ cho PROJ là tạo ra sáu mảnh

$$F_{PROJ} = \{PROJ_1, PROJ_2, PROJ_3, PROJ_4, PROJ_5, PROJ_6\}$$

của quan hệ PROJ tương ứng theo các vị từ tiểu hạng m_1 đến m_6 trong M. Chú ý rằng các mảnh $PROJ_2$ và $PROJ_5$ rỗng.

$PROJ_1$

PNO	PNAME	BUDGET	LOC
P1	Instrumentation	150000	Montreal

$PROJ_3$

PNO	PNAME	BUDGET	LOC
P2	Database Develop.	135000	New York

$PROJ_4$

PNO	PNAME	BUDGET	LOC
P3	CAD/CAM	250000	New York

$PROJ_6$

PNO	PNAME	BUDGET	LOC
P4	Maintenance	310000	Paris

c) Phân mảnh ngang dẫn xuất

Phân mảnh ngang dẫn xuất được định nghĩa trên một quan hệ thành viên của một đường nối dựa theo phép toán chọn trên quan hệ chủ nhân của đường nối đó. Ta cần lưu ý hai điểm sau. Trước tiên đường nối giữa quan hệ chủ và quan hệ thành viên được định nghĩa bằng một phép đẳng nối. Thứ hai, đẳng nối có thể

được cài đặt nhờ các phép *bán nối*. Điểm này rất quan trọng vì ta muốn phân hoạch quan hệ thành viên theo phân mảnh của quan hệ chủ, nhưng cũng muốn mảnh thu được chỉ định nghĩa trên các thuộc tính của quan hệ thành viên.

Muốn thực hiện phân mảnh ngang dẫn xuất, ta cần ba yếu tố đầu vào: tập các phân hoạch của quan hệ chủ, quan hệ thành viên, và tập các vị từ bán nối giữa quan hệ chủ và quan hệ thành viên.

Như thế, nếu cho trước đường nối L, trong đó *owner*(L)=S và *member*(L)=R, các mảnh ngang dẫn xuất của R được định nghĩa là

$$R_i = R >< S_i \text{ , } 1 \le i \le n$$

trong đó *n* là số lượng mảnh được định nghĩa trên R, và $S_i = \sigma_{F_i}(S)$ với F_i là công thức định nghĩa mảnh nguyên thuỷ S_i.

◊ *Ví dụ*. Xét đường nối L_1 trong ví dụ mục a). Ta có *owner*(L_1)=PAY và *member*(L_1)=EMP. Ta có thể nhóm các kỹ sư (engineer) thành hai nhóm tuỳ theo lương: nhóm có lương từ 30000 USD trở xuống và nhóm có lương từ 30000 USD trở lên. Hai mảnh EMP_1 và EMP_2 được định nghĩa như sau

$$EMP_1 = EMP >< PAY_1 \quad \text{và} \quad EMP_2 = EMP >< PAY_2$$

trong đó

$$PAY_1 = \sigma_{SAL \le 30000}(PAY) \quad \text{và} \quad PAY_2 = \sigma_{SAL > 30000}(PAY)$$

Kết quả được trình bày ở các bảng sau

EMP_1

ENO	*ENAME*	*TITLE*
E3	A.Lee	Mech.Eng.
E4	J.Miller	Programmer
E7	R.David	Mech.Eng.

EMP_2

ENO	*ENAME*	*TITLE*
E1	J.Doe	Elect.Eng.
E2	M.Smith	Syst.Anal.
E5	B.Casey	Syst.Anal.
E6	L.Chu	Elect.Eng.
E8	J.Jones	Syst.Anal.

Cũng có vấn đề phức tạp cần chú ý. Trong lược đồ CSDL chúng ta hay gặp nhiều đường nối đến quan hệ R (chẳng hạn có hai đường nối đến quan hệ ASG trong ví dụ trên). Như thế có thể có nhiều cách phân mảnh ngang dẫn xuất cho R. Quyết định chọn cách phân mảnh nào dựa trên 2 tiêu chuẩn.

(1) Phân mảnh nào có đặc tính nối tốt hơn.
(2) Phân mảnh nào được sử dụng trong nhiều ứng dụng hơn.

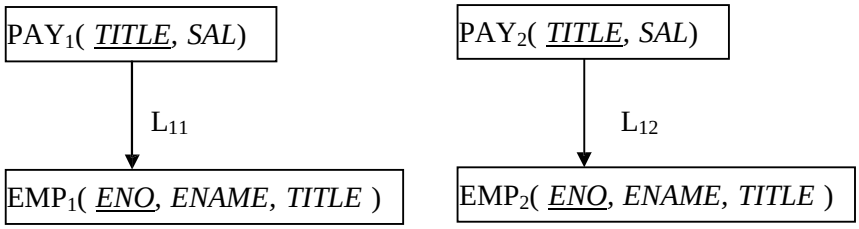
Chúng ta sẽ bàn về tiêu chuẩn thứ 2 trước. Tiêu chuẩn rất đơn giản nếu xét đến tần số truy xuất dữ liệu của các ứng dụng. Nếu được, ta nên ưu tiên những người dùng quan trọng để tác động của họ lên hiệu năng hệ thống là nhỏ nhất.

Tuy nhiên, việc áp dụng tiêu chuẩn thứ nhất không đơn giản. Chẳng hạn thử xét phân mảnh đã thảo luận trong ví dụ trước. Tác động (và mục tiêu) của phân mảnh này là nối của các quan hệ EMP và PAY để trả lời các câu vấn tin được hỗ

trợ (a) bằng cách thực hiện nó trên các quan hệ nhỏ hơn (tức các mảnh) và (b) bằng cách thực hiện các nối theo lối phân tán.

Điểm (a) là hiển nhiên. Các mảnh của EMP nhỏ hơn bản thân EMP. Vì thế khi nối một mảnh của PAY với một mảnh của EMP sẽ nhanh hơn là nối các quan hệ này.

Tuy nhiên điểm (b) lại quan trọng hơn và là trọng tâm của CSDL phân tán. Nếu ngoài việc thực hiện được nhiều câu vấn tin tại nhiều vị trí khác nhau, chúng ta có thể cho thực hiện song song một câu vấn tin, thời gian đáp ứng và lưu lượng hệ thống sẽ được cải thiện. Chẳng hạn xét đồ thị nối giữa các mảnh EMP và PAY lấy từ ví dụ trên. Mỗi mảnh chỉ có 1 đường nối đến hoặc đi khỏi.



Một đồ thị nối như thế gọi là *đồ thị đơn giản*. Thiết kế, trong đó mối liên hệ nối giữa các mảnh là đơn giản, có ưu điểm là quan hệ thành viên và quan hệ chủ có thể được cấp phát cho một vị trí, và các nối giữa các cặp mảnh khác nhau có thể tiến hành độc lập và song song.

Tuy nhiên, không phải lúc nào cũng thu được các đồ thị nối đơn giản. Khi đó thiết kế cần tạo ra *đồ thị nối phân hoạch*. Một đồ thị phân hoạch chứa 2 hoặc nhiều đồ thị con và không có đường nối giữa chúng. Các mảnh như thế không dễ phân tán để thực hiện song song như trong các đồ thị đơn giản, nhưng vẫn có thể cấp phát.

◊ *Ví dụ*. Xét tiếp ví dụ trên. Ta đã phân mảnh EMP theo phân mảnh của PAY. Bây giờ ta xét quan hệ ASG. Giả sử có hai ứng dụng sau:

(1) Ứng dụng thứ nhất tìm tên các kỹ sư có làm việc tại một nơi nào đó. Ứng dụng này chạy ở cả ba trạm và truy xuất thông tin về các kỹ sư làm việc trong các dự án tại chỗ với xác suất cao hơn các kỹ sư làm việc ở những vị trí khác.

(2) Ứng dụng thứ hai là tại mỗi trạm quản lý, nơi lưu các mẩu tin nhân viên, người dùng muốn truy xuất đến các dự án đang được các nhân viên này thực hiện và cần biết xem họ sẽ làm việc với dự án đó trong bao lâu.

Ứng dụng thứ nhất dẫn đến việc phân mảnh ASG theo các mảnh $PROJ_1$, $PROJ_3$, $PROJ_4$ và $PROJ_6$ của PROJ thu được trong ví dụ phân mảnh ngang nguyên thuỷ, trong đó

$$PROJ_1 = \sigma_{LOC=\text{"Montreal"} \wedge BUDGET \leq 200000}(PROJ)$$
$$PROJ_3 = \sigma_{LOC=\text{"New York"} \wedge BUDGET \leq 200000}(PROJ)$$
$$PROJ_4 = \sigma_{LOC=\text{"New York"} \wedge BUDGET > 200000}(PROJ)$$

$$PROJ_6 = \sigma_{LOC=\text{"Paris"} \wedge BUDGET > 200000}(PROJ)$$

Vì thế phân mảnh dẫn xuất của ASG theo $PROJ_1$, $PROJ_3$, $PROJ_4$ và $PROJ_6$ được định nghĩa như sau:

$$ASG_1 = ASG >< PROJ_1$$
$$ASG_2 = ASG >< PROJ_3$$
$$ASG_3 = ASG >< PROJ_4$$
$$ASG_4 = ASG >< PROJ_6$$

Thể hiện các mảnh này được trình bày ở các bảng sau:

ASG_1

ENO	*PNO*	*RESP*	*DUR*
E1	P1	Manager	12
E2	P1	Analyst	24

ASG_3

ENO	*PNO*	*RESP*	*DUR*
E3	P3	Consultant	10
E7	P3	Engineer	36
E8	P3	Manager	40

ASG_2

ENO	*PNO*	*RESP*	*DUR*
E2	P2	Analyst	6
E4	P2	Programmer	18
E5	P2	Manager	24

ASG_4

ENO	*PNO*	*RESP*	*DUR*
E3	P4	Engineer	48
E6	P4	Manager	48

Ứng dụng thứ hai là câu vấn tin được viết bằng SQL như sau:

```
SELECT   RESP, DUR
FROM     ASG, EMP_i
WHERE    ASG.ENO = EMP_i.ENO
```

trong đó *i* =1 hoặc *i* =2 tuỳ thuộc nơi đưa ra câu vấn tin. Phân mảnh dẫn xuất của ASG theo phân mảnh của EMP được định nghĩa dưới đây và cho ở bảng sau:

$$ASG_1 = ASG >< EMP_1$$
$$ASG_2 = ASG >< EMP_2$$

ASG_1

ENO	*PNO*	*RESP*	*DUR*
E3	P3	Consultant	10
E3	P4	Engineer	48
E4	P2	Programmer	18
E7	P3	Engineer	36

ASG_2

ENO	*PNO*	*RESP*	*DUR*
E1	P1	Manager	12
E2	P1	Analyst	24
E2	P2	Analyst	6
E5	P2	Manager	24
E6	P4	Manager	48
E8	P3	Manager	40

Ví dụ này minh hoạ 2 điều:

(1) Phân mảnh dẫn xuất có thể xảy ra dây chuyền, trong đó một quan hệ được phân mảnh như hệ quả của phân mảnh quan hệ khác, và đến lượt nó làm cho các quan hệ khác phân mảnh (chẳng hạn như dây chuyền PAY→EMP→ASG).

(2) Thông thường một quan hệ có nhiều cách phân mảnh (chẳng hạn như ASG). Chọn một lược đồ phân mảnh là bài toán quyết định và sẽ được phân tích trong phần cấp phát.

d) Kiểm định tính đúng đắn

Bây giờ ta cần kiểm tra các tiêu chuẩn đối với các thuật toán.

• *Tính đầy đủ*

Tính đầy đủ của phân mảnh ngang nguyên thuỷ dựa vào các vị từ chọn được dùng. Với điều kiện các vị từ chọn là đầy đủ, phân mảnh thu được cũng đảm bảo đầy đủ. Bởi vì cơ sở của thuật toán phân mảnh là tập các vị từ cực tiểu và đầy đủ *Pr'*, tính đầy đủ sẽ được đảm bảo.

Tính đầy đủ của phân mảnh ngang dẫn xuất có phức tạp hơn. Trước tiên chúng ta định nghĩa lại qui tắc đầy đủ.

Cho R là quan hệ thành viên của đường nối với quan hệ chủ S. Quan hệ S được phân mảnh thành $F_S = \{S_1, ..., S_k\}$. Gọi A là thuộc tính nối giữa R và S. Ký hiệu $\{R_1, ..., R_k\}$ là tập phân mảnh dẫn xuất của R theo F_S. Khi đó với mỗi bộ t của R_i thì phải có 1 bộ t' của S_i thoả t[A]=t'[A], với mọi i =1, ..., k.

Tính đầy đủ ở đây thực chất là đảm bảo *ràng buộc toàn vẹn tham chiếu*.

• *Tính tái thiết*

Tái thiết một quan hệ toàn cục từ các mảnh được thực hiện bằng toán tử hợp trong cả phân mảnh ngang nguyên thuỷ lẫn dẫn xuất.

$$R = \mathop{\mathrm{U}}_{R_i \in F} R_i$$

• *Tính tách rời*

Trong phân mảnh ngang nguyên thuỷ tính tách rời được đảm bảo nếu các vị từ tiểu hạng xác định phân mảnh có tính loại trừ tương hỗ.

Tuy nhiên phân mảnh dẫn xuất có hàm chứa các bán nối có phức tạp hơn. Tính tách rời được đảm bảo nếu đồ thị nối thuộc loại đơn giản. Nếu đồ thị nối không đơn giản thì phải xem xét các giá trị thực sự của phân mảnh.

◊ *Ví dụ*. Khi phân mảnh quan hệ PAY ở ví dụ trước, các vị từ tiểu hạng $M=\{m_1,m_2\}$ là

$$m_1 = SAL \leq 30000$$
$$m_2 = SAL > 30000$$

Vì m_1 và m_2 loại trừ tương hỗ, phân mảnh của PAY là tách biệt.

Tuy nhiên, với quan hệ EMP ta đòi hỏi

(i) Mỗi kỹ sư có một chức vụ duy nhất.
(ii) Mỗi chức vụ có một giá trị lương duy nhất đi kèm.

Vì hai qui tắc này suy ra từ ngữ nghĩa của CSDL, phân mảnh của EMP ứng với PAY cũng tách rời.

2.3.2. Phân mảnh dọc

Một phân mảnh dọc của quan hệ R là tập các mảnh R_1, ..., R_k, trong đó mỗi mảnh chứa tập con thuộc tính của R và các khoá của R. Mục đích của phân mảnh dọc là phân hoạch một quan hệ thành tập các quan hệ nhỏ hơn để nhiều ứng dụng chỉ cần chạy trên 1 mảnh. Phân mảnh tối ưu cho phép giảm tối đa thời gian thực thi các ứng dụng chạy trên các mảnh đó.

Phân mảnh dọc phức tạp hơn so với phân mảnh ngang, do số khả năng phân mảnh dọc rất lớn. Trong phân mảnh ngang, nếu tổng số vị từ đơn giản trong *Pr* là *n* thì có 2^n vị từ tiểu hạng có thể định nghĩa trên nó. Ngoài ra có thể loại bỏ một số lớn trong đó mâu thuẫn với phép kéo theo hiện có, như vậy sẽ làm giảm đi số mảnh dự tuyển. Trong trường hợp phân mảnh dọc, nếu quan hệ có *m* thuộc tính không khoá, thì số mảnh có thể bằng *B*(*m*), *số Bell* thứ *m*. Với *m* lớn $B(m) \approx m^m$, chẳng hạn với $B(10) \approx 115000$, $B(15) \approx 10^9$, $B(30) \approx 10^{23}$.

Những giá trị này cho thấy, nỗ lực tìm lời giải tối ưu cho bài toán phân hoạch dọc không hiệu quả; vì thế phải dùng đến các phương pháp *heuristic*. Chúng ta nêu ở đây hai loại heuristic cho phân mảnh dọc các quan hệ toàn cục.

(1) *Nhóm thuộc tính*: Bắt đầu bằng cách gán mỗi thuộc tính cho một mảnh, và tại mỗi bước nối số mảnh lại cho đến khi thoả tiêu chuẩn nào đó.

(2) *Tách mảnh*: Bắt đầu bằng một quan hệ và quyết định cách phân hoạch có lợi dựa trên hành vi truy xuất của các ứng dụng trên các thuộc tính.

Trong phần tiếp chúng ta chỉ thảo luận kỹ thuật tách mảnh, vì nó thích hợp phương pháp thiết kế từ trên xuống hơn và giải pháp tối ưu gần với quan hệ đầy đủ hơn là tập các mảnh chỉ có 1 thuộc tính. Hơn nữa kỹ thuật tách mảnh sinh ra các mảnh với các thuộc tính không khoá không chồng nhau.

Việc nhân bản khoá chính cho các mảnh là đặc trưng của phân mảnh dọc, cho phép tái thiết quan hệ toàn cục. Vì thế phương pháp tách mảnh chỉ đề cập đến các thuộc tính không khoá.

Mặc dù có những vấn đề phát sinh, nhân bản các thuộc tính khoá có ưu điểm nổi bật là duy trì tính toàn vẹn ngữ nghĩa.

Một chọn lựa khác đối với nhân bản các thuộc tính khoá là sử dụng một *mã định bộ* (TID - tuple identifier). Đây là giá trị duy nhất được hệ thống gán cho mỗi bộ của quan hệ.

a) Các yêu cầu thông tin của phân mảnh dọc

Những thông tin chính cần cho phân mảnh dọc có liên quan tới các ứng dụng. Vì phân mảnh dọc đặt vào một mảnh các thuộc tính thường được truy xuất chung với nhau, ta cần xác định một độ đo khái niệm "chung với nhau". Số đo này gọi là *ái lực* (*affinity*) của thuộc tính, chỉ mức độ liên đới giữa các thuộc tính.

Thông số quan trọng về dữ liệu có liên quan đến các ứng dụng là *tần số truy xuất* (*access frequency*) của chúng. Gọi $Q = \{q_1, ..., q_k\}$ là tập các vấn tin của người dùng sẽ chạy trên quan hệ $R(A_1, ..., A_n)$. Với mỗi câu vấn tin q_i và mỗi thuộc tính A_j ta định nghĩa *giá trị sử dụng thuộc tính* (*attribute usage value*), ký hiệu $use(q_i, A_j)$, như sau

$$use(q_i, A_j) = \begin{cases} 1, \text{ nếu } q_i \text{ tham chiếu thuộc tính } A_j \\ 0, \text{ nếu ngược lại} \end{cases}$$

Các vectơ $use(q_i, \bullet)$ cho mỗi ứng dụng sẽ được xác định nếu nhà thiết kế biết được các ứng dụng chạy trên cơ sở dữ liệu. Cần nhắc lại rằng *qui tắc 80/20* rất có ích cho công việc này.

◊ *Ví dụ.* Xét quan hệ PROJ của ví dụ phần trước. Giả sử các ứng dụng sau đây chạy trên quan hệ đó.

q_1 : Tìm ngân sách của dự án, cho biết mã số dự án.

```
SELECT   BUDGET
FROM     PROJ
WHERE    PNO = Value
```

q_2 : Tìm tên và ngân sách của tất cả dự án.

```
SELECT   PNAME, BUDGET
FROM     PROJ
```

q_3 : Tìm tên của các dự án được thực hiện ở thành phố đã cho

```
SELECT   PNAME
FROM     PROJ
WHERE    LOC = Value
```

q_4 : Tìm tổng ngân sách các dự án được thực hiện ở thành phố đã cho

```
SELECT   SUM(BUDGET)
FROM     PROJ
WHERE    LOC = Value
```

Dựa theo 4 ứng dụng này ta có thể xác định được giá trị sử dụng các thuộc tính. Để cho đơn giản ta ký hiệu

$$A_1 = PNO,\ A_2 = PNAME,\ A_3 = BUDGET \text{ và } A_4 = LOC.$$

Giá trị sử dụng cho ở ma trận sau (phần tử ô [i, j] chính là $use(q_i, A_j)$) :

	A_1	A_2	A_3	A_4
q_1	1	0	1	0
q_2	0	1	1	0
q_3	0	1	0	1
q_4	0	0	1	1

Giá trị sử dụng thuộc tính không đủ thông tin để làm cơ sở cho việc tách và phân mảnh. Lý do là chúng không biểu thị độ lớn của tần số ứng dụng. Số đo tần số có thể được chứa trong định nghĩa về số đo ái lực thuộc tính $aff(A_i, A_j)$, biểu thị cho *cầu nối* (*bond*) giữa hai thuộc tính của một quan hệ theo cách chúng được các ứng dụng truy xuất.

Cho quan hệ R(A_1, ..., A_n) và tập ứng dụng Q={q_1, ..., q_k}. Với mỗi cặp thuộc tính (A_i, A_j) ký hiệu

S_l là vị trí thứ l, l = 1, ..., L

$ref_l(q_k)$ là số lần truy xuất đến A_i, A_j cho mỗi lần thực hiện ứng dụng q_k tại vị trí S_l.

$acc_l(q_k)$ là số đo tần số thực hiện ứng dụng q_k tại vị trí S_l.

$Q(i, j) = \{k \mid use(q_k, A_i) = 1 \ \& \ use(q_k, A_j) = 1\}$

Số đo ái lực thuộc tính giữa hai thuộc tính A_i và A_j của quan hệ R(A_1 , ..., A_n) ứng với tập ứng dụng Q={q_1, ..., q_k} được xác định theo công thức

$$aff(A_i, A_j) = \sum_{k \in Q(i,j)} \sum_{l=1}^{L} ref_l(q_k).acc_l(q_k)$$

Kết quả là ma trận vuông cấp n gọi là *ma trận ái lực thuộc tính* (*AA - attribute affinity matrix*).

◊ *Ví dụ.* Ta tiếp tục ví dụ trên. Để đơn giản ta giả sử rằng có 3 vị trí S_1, S_2 và S_3 và 4 ứng dụng và $ref_l(q_k)$ =1 với mọi S_l và mọi q_k. Cho tần số sử dụng như sau

$$\begin{array}{lll} acc_1(q_1) = 15; & acc_2(q_1) = 20; & acc_3(q_1) = 10; \\ acc_1(q_2) = 5; & acc_2(q_2) = 0; & acc_3(q_2) = 0; \\ acc_1(q_3) = 25; & acc_2(q_3) = 25; & acc_3(q_3) = 25; \\ acc_1(q_4) = 3; & acc_2(q_4) = 0; & acc_3(q_4) = 0; \end{array}$$

Vì chỉ có ứng dụng q_1 truy xuất đến cả hai thuộc tính A_1 và A_3, nên ta có

$$aff(A_1, A_3) = acc_1(q_1) + acc_2(q_1) + acc_3(q_1) = 45$$

Tương tự ta tính được ma trận ái lực sau

$$AA = \begin{array}{c} \\ A_1 \\ A_2 \\ A_3 \\ A_4 \end{array} \begin{array}{c} \begin{array}{cccc} A_1 & A_2 & A_3 & A_4 \end{array} \\ \begin{pmatrix} 45 & 0 & 45 & 0 \\ 0 & 80 & 5 & 75 \\ 45 & 5 & 53 & 3 \\ 0 & 75 & 3 & 78 \end{pmatrix} \end{array}$$

b) Thuật toán tụ nhóm

Nhiệm vụ cơ bản trong việc xây dựng thuật toán phân mảnh dọc là tìm tiêu chí nào đó để nhóm các thuộc tính của quan hệ dựa trên ma trận ái lực. *Thuật toán năng lượng nối BEA* (*bond energy algorithm*) là thích hợp vì những lý do sau:

(1) BEA gom các thuộc tính có cùng độ lớn ái lực lại với nhau.

(2) Các kết quả tụ nhóm không bị ảnh hưởng bởi thứ tự đưa các thuộc tính vào thuật toán.

(3) Độ phức tập tính toán $O(n^2)$, với n là số thuộc tính, là chấp nhận được.

(4) Có thể xác định mối liên hệ giữa các nhóm thuộc tính.

Thuật toán năng lượng nối BEA hoán vị các hàng và cột ma trận ái lực để tạo thành *ma trận ái lực tụ CA* (*clustered affinity matrix*). Hoán vị được thực hiện để cực đại *số đo ái lực chung AM* (*global affinity measure*):

$$AM = \sum_{i=1}^{n}\sum_{j=1}^{n} aff(A_i, A_j)[aff(A_i, A_{j-1}) + aff(A_i, A_{j+1}) + aff(A_{i-1}, A_j) + aff(A_{i+1}, A_j)]$$

trong đó

$$aff(A_0, A_j) = aff(A_i, A_0) = aff(A_{n+1}, A_j) = aff(A_i, A_{n+1}) = 0 \quad \forall i,j$$

Vì ma trận ái lực *AA* đối xứng nên hàm mục tiêu *AM* có thể rút gọn như sau

$$AM = \sum_{i=1}^{n}\sum_{j=1}^{n} aff(A_i, A_j)[aff(A_i, A_{j-1}) + aff(A_i, A_{j+1})]$$

Quá trình sinh *ma trận ái lực tụ CA* được thực hiện qua 3 bước.

(1) *Khởi tạo*. Đặt cố định một trong các cột của *AA* vào *CA*, chẳng hạn cột 1. Gán số cột trong *CA* i := 1.

(2) *Thực hiện lặp*. Lấy lần lượt một trong $n-i$ cột còn lại trong *AA* đặt vào $i+1$ vị trí giữa i cột trong *CA* (kể cả hai đầu). Chọn nơi đặt sao cho nó làm tăng nhiều nhất số đo ái lực được mô tả ở trên. Tăng $i := i+1$.
Tiếp tục bước này cho đến khi $i = n$.

(3) *Sắp thứ tự hàng*. Một khi thứ tự các cột đã được xác định, các hàng cũng sắp thứ tự lại tương ứng.

Để bước (2) của thuật toán hoạt động chúng ta cần định nghĩa xem *đóng góp* (*contribution*) của thuộc tính vào số đo ái lực mang ý nghĩa gì.
Ta có

$$\begin{aligned} AM &= \sum_{i=1}^{n}\sum_{j=1}^{n} aff(A_i, A_j)[aff(A_i, A_{j-1}) + aff(A_i, A_{j+1})] \\ &= \sum_{i=1}^{n}\sum_{j=1}^{n} \left[aff(A_i, A_j)aff(A_i, A_{j-1}) + aff(A_i, A_j)aff(A_i, A_{j+1})\right] \end{aligned}$$

Ta định nghĩa *cầu nối* (*bond*) giữa hai thuộc tính A_x và A_y như sau

$$bond(A_x, A_y) = \sum_{z=1}^{n} aff(A_z, A_x)aff(A_z, A_y)$$

Như vậy *AM* có thể viết lại là

$$AM = \sum_{j=1}^{n} \left[bond(A_j, A_{j-1}) + bond(A_j, A_{j+1})\right]$$

Bây giờ xét n thuộc tính sau

$$\underbrace{A_1, A_2, \ldots, A_{i-1}}_{AM'} \underbrace{A_i A_j A_{j+1}, \ldots, A_n}_{AM''}$$

Số đo ái lực chung cho các thuộc tính này có thể viết như sau:

$$AM_{old} = AM' + AM'' + bond(A_{i-1}, A_i) + bond(A_i, A_j) + bond(A_j, A_i) + bond(A_j, A_{j+1})$$

$$= \sum_{l=1}^{i-1}[bond(A_l, A_{l-1}) + bond(A_l, A_{l+1})] + \sum_{l=j+1}^{n}[bond(A_l, A_{l-1}) + bond(A_l, A_{l+1})] + 2bond(A_i, A_j)$$

Bây giờ xét đến việc đặt thuộc tính mới A_k vào giữa thuộc tính A_i và A_j trong ma trận ái lực tụ:

$$\underbrace{A_1, A_2, \ldots, A_{i-1}}_{AM'} \underbrace{A_i A_k A_j A_{j+1}, \ldots, A_n}_{AM''}$$

Số đo ái lực chung mới có thể biểu diễn tương tự như sau:

$$AM_{new} = AM' + AM'' + bond(A_i, A_k) + bond(A_k, A_i) + bond(A_k, A_j) + bond(A_j, A_k)$$
$$= AM' + AM'' + 2bond(A_i, A_k) + 2bond(A_k, A_j)$$

Vì thế đóng góp thực (net contribution) cho số đo ái lực chung khi đặt thuộc tính A_k vào giữa thuộc tính A_i và A_j là

$$cont(A_i, A_k, A_j) = AM_{new} - AM_{old} = 2bond(A_i, A_k) + 2bond(A_k, A_j) - 2bond(A_i, A_j)$$

Sau đây là *thuật toán năng lượng nối* BEA

• **Thuật toán BEA**
Đầu vào: Ma trận ái lực *AA* cấp *n*.
Đầu ra: Ma trận ái lực tụ *CA*.
{*loc* là biến lưu vị trí nơi có giá trị đóng góp *cont* lớn nhất}
Begin
{khởi tạo}
CA(•,1) := *AA*(•,1); {đưa cột thứ nhất của *AA* vào cột thứ nhất của *CA*}
CA(•,2) := *AA*(•,2); {đưa cột thứ hai của *AA* vào cột thứ hai của *CA*}
index := 3;
while *index* ≤ *n* **do** {chọn vị trí tốt nhất cho cột AA_{index}}
begin
for *i:=1* to *index−1* **do**
begin
tính $cont(A_{i-1}, A_{index}, A_i)$
end {for}
tính $cont(A_{index-1}, A_{index}, A_{index+1})$

```
        loc := nơi có giá trị cont lớn nhất.
        for  j := index downto loc do
            CA(•, j) := CA(•, j−1)
            CA(•, loc) := AA(•, index)
        index := index + 1
      end;
  {sắp thứ tự các hàng theo thứ tự các cột}
End; {BEA}
```

◊ *Ví dụ*

Xét ma trận *AA* tính ở ví dụ trước

$$AA = \begin{array}{c} \\ A_1 \\ A_2 \\ A_3 \\ A_4 \end{array} \begin{array}{c} \begin{array}{cccc} A_1 & A_2 & A_3 & A_4 \end{array} \\ \begin{pmatrix} 45 & 0 & 45 & 0 \\ 0 & 80 & 5 & 75 \\ 45 & 5 & 53 & 3 \\ 0 & 75 & 3 & 78 \end{pmatrix} \end{array}$$

Ta tính phần đóng góp khi di chuyển A_4 vào giữa các thuộc tính A_1 và A_2 được cho bằng công thức

$$cont(A_1, A_4, A_2) = 2bond(A_1, A_4) + 2bond(A_4, A_2) - 2bond(A_1, A_2)$$
$$= 2*135 + 2*11865 - 2*225 = 23550$$

vì

$$bond(A_1, A_4) = 45*0 + 0*75 + 45*3 + 0*78 = 135$$
$$bond(A_4, A_2) = 0*0 + 75*80 + 3*5 + 78*75 = 11865$$
$$bond(A_1, A_2) = 45*0 + 0*80 + 45*5 + 0*75 = 225$$

Trường hợp thuộc tính A_k đặt vào tận cùng bên trái hoặc tận cùng bên phải ta có

$$bond(A_0, A_k) = bond(A_k, A_{k+1}) = 0$$

◊*Ví dụ*

Ta xét tiếp ví dụ trên. Bây giờ ta gom tụ các thuộc tính của quan hệ PROJ và dùng ma trận ái lực *AA*.

Theo bước khởi đầu ta đưa cột 1 và cột 2 của *AA* vào *CA*

$$\begin{array}{c} \\ A_1 \\ A_2 \\ A_3 \\ A_4 \end{array} \begin{array}{c} \begin{array}{cccc} A_1 & A_2 & A_3 & A_4 \end{array} \\ \begin{pmatrix} & & 45 & 0 \\ & & 5 & 75 \\ & & 53 & 3 \\ & & 3 & 78 \end{pmatrix} \end{array} \longrightarrow \begin{array}{c} \\ A_1 \\ A_2 \\ A_3 \\ A_4 \end{array} \begin{array}{c} \begin{array}{cc} A_1 & A_2 \end{array} \\ \begin{pmatrix} 45 & 0 \\ 0 & 80 \\ 45 & 5 \\ 0 & 75 \end{pmatrix} \end{array}$$

Tiếp theo ta di chuyển cột 3 của AA sang CA. Có 3 nơi có thể đặt cột 3:

- Bên trái cột 1 (0-3-1): ta có

  $bond(A_0,A_1) = bond(A_0,A_3) = 0$

$bond(A_3,A_1) = 45*45+5*0+53*45+3*0 = 4410$

suy ra

$cont(A_0,A_3,A_1) = 2\ bond(A_0,A_3) + 2bond(A_3,A_1) - 2\ bond(A_0,A_1) = 8820$

- Giữa cột 1 và 2 (1-3-2): ta có

$bond(A_1,A_3) = bond(A_3,A_1) = 4410$

$bond(A_3,A_2) = 890$

$bond(A_1,A_2) = 225$

suy ra

$cont(A_1,A_3,A_2) = 10150$

- Bên phải cột 2 (2-3-4): ta có

$bond(A_3,A_4) = bond(A_2,A_4) = 0$

$bond(A_1,A_4) = 890$

suy ra

$cont(A_2,A_3,A_4) = 1780$

Vì đóng góp của trường hợp thứ 2 là lớn nhất, ta chèn A_3 vào giữa A_1 và A_2, và có

$$\begin{array}{c} \\ A_1 \\ A_2 \\ A_3 \\ A_4 \end{array}\begin{array}{c} \begin{array}{cccc} A_1 & A_2 & A_3 & A_4 \end{array} \\ \begin{pmatrix} & & & 0 \\ & & & 75 \\ & & & 3 \\ & & & 78 \end{pmatrix} \end{array} \longrightarrow \begin{array}{c} \\ A_1 \\ A_2 \\ A_3 \\ A_4 \end{array}\begin{array}{c} \begin{array}{ccc} A_1 & A_3 & A_2 \end{array} \\ \begin{pmatrix} 45 & 45 & 0 \\ 0 & 5 & 80 \\ 45 & 53 & 5 \\ 0 & 3 & 75 \end{pmatrix} \end{array}$$

Tính toán tương tự cho A_4 chỉ ra rằng cần đặt nó bên phải A_2 .

$$\begin{array}{c} \\ A_1 \\ A_2 \\ A_3 \\ A_4 \end{array}\begin{array}{c} \begin{array}{cccc} A_1 & A_3 & A_2 & A_4 \end{array} \\ \begin{pmatrix} 45 & 45 & 0 & 0 \\ 0 & 5 & 80 & 75 \\ 45 & 53 & 5 & 3 \\ 0 & 3 & 75 & 78 \end{pmatrix} \end{array}$$

Cuối cùng ta hoán chuyển thứ tự các hàng và được ma trận kết quả

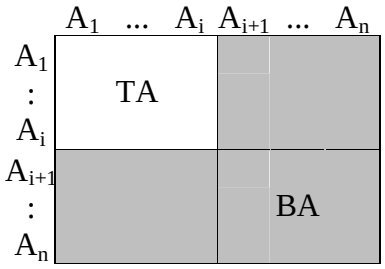
$$CA = \begin{array}{c} \\ A_1 \\ A_3 \\ A_2 \\ A_4 \end{array}\begin{array}{c} \begin{array}{cccc} A_1 & A_3 & A_2 & A_4 \end{array} \\ \begin{pmatrix} 45 & 45 & 0 & 0 \\ 45 & 53 & 5 & 3 \\ 0 & 5 & 80 & 75 \\ 0 & 3 & 75 & 78 \end{pmatrix} \end{array}$$

Ta thấy quá trình tạo ra hai tụ: một ở góc trên bên trái chứa các ái lực nhỏ, còn tụ kia ở góc dưới bên phải chứa các ái lực cao.

c) Thuật toán phân hoạch

Mục đích của hành động tách thuộc tính là tìm tập thuộc tính được truy xuất cùng nhau, hoặc với phần lớn các thuộc tính, bởi các tập ứng dụng riêng biệt. Thí dụ, nếu biết hai thuộc tính A_1 và A_2 chỉ được ứng dụng q_1 truy xuất và các thuộc tính A_3 và A_4 được các ứng dụng khác truy xuất, chẳng hạn q_2 và q_3, thì quyết định phân mảnh là đương nhiên. Vấn đề là tìm được thuật toán để xác định các nhóm này.

Xét ma trận thuộc tính tụ *CA*. Nếu một điểm nằm trên đường chéo được cố định, hai tập thuộc tính sẽ được xác định. Tập $\{A_1, ..., A_i\}$ ở góc trên trái và tập $\{A_{1+1}, ..., A_n\}$ ở góc dưới phải. Tập thứ nhất gọi là *đỉnh* (top) và ký hiệu *TA* , tập thứ hai gọi là *đáy* (bottom) và ký hiệu *BA*.



Bây giờ xét tập ứng dụng Q = $\{q_1, ..., q_k\}$ và định nghĩa các tập ứng dụng chỉ truy xuất TA, BA hoặc cả hai. Những tập này định nghĩa như sau

$$
\begin{aligned}
AQ(q_i) &= \{A_j \mid use(q_i, A_j) = 1\} \\
TQ &= \{q_j \mid AQ(q_i) \subseteq TA\} \\
BQ &= \{q_j \mid AQ(q_i) \subseteq BA\} \\
OQ &= Q - (TQ \cup BQ)
\end{aligned}
$$

Phương trình đầu định nghĩa tập thuộc tính được truy xuất bởi ứng dụng q_i ; TQ và BQ tương ứng là các tập ứng dụng chỉ truy xuất TA và BA, còn OQ là tập ứng dụng truy xuất cả hai tập TA và BA.

Ở đây nảy sinh bài toán tối ưu hoá. Nếu có *n* thuộc tính trong quan hệ, thì sẽ có $n-1$ vị trí khả hữu có thể là điểm phân chia trên đường chéo của ma trận thuộc tính tụ cho quan hệ đó. Vị trí tốt nhất để phân chia là vị trí sinh các tập TQ và BQ sao cho tổng các *truy xuất chỉ một mảnh* là lớn nhất, còn tổng các *truy xuất cả hai mảnh* là nhỏ nhất. Vì thế ta định nghĩa các phương trình chi phí như sau:

$$CQ = \sum_{q_i \in Q} \sum_{\forall S_j} ref_j(q_i) acc_j(q_i)$$

$$CTQ = \sum_{q_i \in TQ} \sum_{\forall S_j} ref_j(q_i) acc_j(q_i)$$

$$CBQ = \sum_{q_i \in BQ} \sum_{\forall S_j} ref_j(q_i) acc_j(q_i)$$

$$COQ = \sum_{q_i \in OQ} \sum_{\forall S_j} ref_j(q_i) acc_j(q_i)$$

Mỗi phương trình trên đếm tổng số truy xuất đến các thuộc tính bởi các ứng dụng trong các lớp tương ứng của chúng. Dựa trên số liệu này bài toán tối ưu hoá được định nghĩa là bài toán tìm điểm x ($1 \le x \le n$) sao cho biểu thức

$$z = CTQ*CBQ - COQ^2$$

lớn nhất.

Đặc trưng quan trọng của biểu thức này là nó định nghĩa hai mảnh sao cho giá trị của CTQ và CBQ càng gần bằng nhau càng tốt. Điều này cho phép cân bằng tải trọng xử lý khi các mảnh được phân tán đến các vị trí khác nhau. Thuật toán phân hoạch có độ phức tạp tuyến tính *O(n)*, theo số thuộc tính của quan hệ.

Để tăng thêm số phương án xét, tức hiệu quả thuật toán, chúng ta cần hiệu chỉnh thuật toán bằng thủ tục *xê dịch* (SHIFT) thuộc tính như sau. Sau mỗi bước lặp tìm vị trí tốt nhất cho một thứ tự thuộc tính, ta dịch cột tận trái sang thành cột tận phải, và hàng trên cùng xuống cuối cùng. Với thao tác xê dịch như thế này ta chỉ cần kểm tra $n-1$ vị trí trên đường chéo để tìm trị z lớn nhất. Độ phức tạp của thuật toán sẽ tăng thêm n lần, tức là $O(n^2)$.

Với giả thiết đã có thủ tục xê dịch SHIFT, ta có thuật toán phân hoạch PARTITION sau:

• **Thuật toán PARTITION**

Đầu vào: ma trận ái lực tụ *CA*
quan hệ *R*
ma trận sử dụng thuộc tính *ref*
ma trận tần số truy xuất *acc*

Đầu ra: tập các mảnh $F_R = \{R_1, R_2\}$. với $R_1 \cap R_2$ là khoá.

Begin
{xác định giá trị z cho cột thứ nhất}
{các chỉ mục trong phương trình chi phí chỉ ra điểm tách}
Tính CTQ_{n-1} ;
Tính CBQ_{n-1} ;
Tính COQ_{n-1} ;
gán *best* := $CTQ_{n-1} * CBQ_{n-1} - (COQ_{n-1})^2$;
do {xác định cách phân hoạch tốt nhất}
begin
for i := n–2 downto 1 do
begin
Tính CTQ_i ;
Tính CBQ_i ;

Tính COQ_i ;
gán $z := CTQ_i * CBQ_i - (COQ_i)^2$;
if $z > best$ *then*
begin
$best := z$;
ghi nhận điểm tách vào trong hành động xê dịch
end;
end;{for}
gọi SHIFT(CA);
end;
until không thể thực hiện SHIFT được nữa.
xây dựng lại ma trận theo vị trí xê dịch
$R_1 := \pi_{TA}(R) \cup K$ {K là tập thuộc tính khoá chính của R}
$R_2 := \pi_{BA}(R) \cup K$
$F := \{R_1, R_2\}$;
end.

◊ *Ví dụ*

Áp dụng thuật toán PARTITION cho ma trận *CA* từ quan hệ PROJ ở thí dụ trước. Kết quả là định nghĩa các mảnh $F_{PROJ} = \{PROJ_1, PROJ_2\}$, trong đó

$PROJ_1 = \{A_1, A_3\} = \{ PNO, BUDGET \}$
$PROJ_2 = \{A_1, A_2, A_4\} = \{ PNO, PNAME, LOC \}$

d) Kiểm tra tính đúng đắn

♦ *Tính đầy đủ*: được đảm bảo bằng thuật toán PARTITION vì mỗi thuộc tính của quan hệ toàn cục được đưa vào một trong các mảnh.

♦ *Tính tái thiết*: đảm bảo

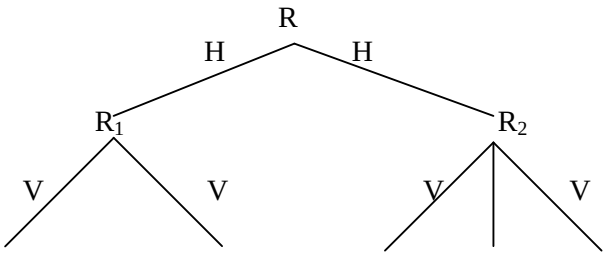
$$R = \ ><_K R_i \ \forall R_i \in F_R .$$

nếu mỗi R_i là đầy đủ và chứa thuộc tính khoá của R hoặc chứa *mã định bộ* (TID - tuple identifier) được gán bởi hệ thống.

♦ *Tính tách biệt*: đảm bảo tách biệt đối với các thuộc tính không khoá.

2.3.3. Phân mảnh hỗn hợp

Trong đa số trường hợp, phân mảnh ngang hoặc phân mảnh dọc đơn giản cho một lược đồ CSDL không đủ đáp ứng yêu cầu các ứng dụng. Khi đó phân mảnh dọc có thể được thực hiện sau một phân mảnh ngang hoặc ngược lại. Chiến lược này sinh ra một lối phân hoạch có cấu trúc cây và gọi là *phân mảnh hỗn hợp* (hybrid fragmentation).



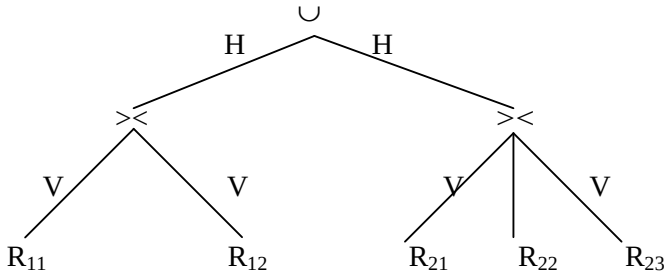
R_{11} R_{12} R_{21} R_{22} R_{23}

Phân mảnh hỗn hợp

Một ví dụ điển hình minh hoạ cho sự cần thiết phải có phân mảnh hỗn hợp là quan hệ PROJ. Trong một ví dụ trước ta đã phân hoạch nó thành sáu mảnh ngang dựa vào hai ứng dụng. Trong một ví dụ sau đó, chúng ta lại phân mảnh dọc PROJ thành hai mảnh. Như thế chúng ta có một tập các mảnh ngang, mỗi mảnh ngang lại được phân tiếp thành hai mảnh dọc.

Số mức lồng ghép có thể khá lớn, nhưng hữu hạn. Trong thực tế, do các quan hệ toàn cục đã chuẩn hoá và chi phí nối nhiều mảnh có thể quá cao, nên phần lớn người ta chỉ thực hiện tối đa hai mức lồng ghép.

Tính đúng đắn của phân mảnh hỗn hợp suy ra từ tính đúng đắn của phân mảnh ngang và phân mảnh dọc. Thí dụ để tái thiết quan hệ toàn cục, người ta bắt đầu tại các nút lá của cây phân hoạch và di chuyển lên trên bằng cách thực hiện các phép nối và hợp như ở hình sau



Tính tái thiết của phân mảnh hỗn hợp

2.4. Cấp phát

Cấp phát tài nguyên cho các nút của mạng máy tính là bài toán đã được nghiên cứu rộng rãi.

2.4.1. Bài toán cấp phát

Giả sử có tập các mảnh $FF = \{F_1, ..., F_n\}$ và mạng bao gồm các vị trí $SS = \{S_1,..., S_m\}$ trên đó có tập ứng dụng $QQ = \{q_1 ,..., q_l\}$ đang chạy.

Bài toán cấp phát (allocation problem) là tìm phân phối tối ưu của FF cho SS.

Một trong các điểm quan trọng là định nghĩa khái niệm *tối ưu* (*optimality*). Khái niệm tối ưu có thể định nghĩa ứng với hai số đo:

(1) *Chi phí nhỏ nhất*: Hàm chi phí gồm có chi phí lưu mỗi mảnh F_i tại vị trí S_j, chi phí vấn tin F_i tại vị trí S_j , chi phí cập nhật F_i tại tất cả mọi vị trí chứa nó và chi phí truyền dữ liệu. Bài toán cấp phát cố gắng tìm lược đồ cấp phát với hàm chi phí tổ hợp thấp nhất.

(2) *Hiệu năng*: Chiến lược cấp phát được thiết kế nhằm duy trì hiệu năng hữu lượng. Các chiến lược đã biết là hạ thấp thời gian đáp ứng và tăng tối đa lưu lượng hệ thống tại mỗi vị trí.

Một lược đồ cấp phát hoàn hảo là trả lời câu vấn tin trong thời gian ngắn nhất mà vẫn duy trì chi phí xử lý thấp nhất. Do tính phức tạp của bài toán, nên người ta mới chỉ giải quyết những trường hợp đơn giản.

Chúng ta xét một trường hợp đơn giản. Cố định mảnh $F \in FF$. Chúng ta đưa ra một số giả thiết và định nghĩa nhằm mô hình hoá bài toán cấp phát này.

(1) Giả sử có thể sửa đổi QQ để xác định được các *vấn tin cập nhật* (update query) và *vấn tin chỉ đọc* (retrieval-only query), và để định nghĩa các đại lượng sau đây cho mảnh F

$$T = \{t_1,...,t_m\}$$

với t_i là lưu lượng chỉ đọc được sinh ra tại S_i cho mảnh F và

$$U = \{u_1,...,u_m\}$$

với u_i là lưu lượng cập nhật được sinh ra tại S_i cho mảnh F .

(2) Giả sử chi phí truyền giữa hai vị trí S_i và S_j là cố định với mỗi đơn vị truyền. Ngoài ra cũng giả sử rằng chúng khác nhau khi cập nhật và khi truy xuất chỉ đọc. Ta định nghĩa

c_{ij} là chi phí truyền 1 đơn vị đối với các yêu cầu *chỉ đọc* giữa vị trí S_i và S_j ($i,j=1, ..., m$);

c'_{ij} là chi phí truyền 1 đơn vị đối với các yêu cầu *cập nhật* giữa vị trí S_i và S_j ($i, j=1, ..., m$);

và ký hiệu

$$C(T) = \{c_{12}, ..., c_{1m}, ..., c_{m-1,m}\}$$
$$C'(U) = \{c'_{12}, ..., c'_{1m}, ..., c'_{m-1,m}\}$$

(3) Gọi chi phí lưu trữ mảnh F tại vị trí S_i là d_i và ký hiệu

$$D = \{d_1, ..., d_m\}$$

(4) Giả thiết không có ràng buộc về khả năng lưu trữ cho các vị trí hoặc cho các đường truyền.

Khi đó bài toán cấp phát có thể được đặc tả là bài toán cực tiểu hoá chi phí, trong đó ta tìm tập $I \subseteq SS$ các vị trí lưu các bản sao của mảnh F. Ký hiệu x_i là *biến quyết định* (decision variable) chọn nơi đặt sao cho

$$x_i = \begin{cases} 1, \text{ nếu } F \text{ phân cho } S_i \\ 0, \text{ nếu ngược lại} \end{cases}$$

Bài toán cấp phát được phát biểu chính xác như sau

$$\min_{I \subset SS} \left[\sum_{i=1}^{m} \left(\sum_{S_j \in I} x_j u_j c_{ij} + t_i \min_{S_j \in I} c_{ij} \right) + \sum_{S_j \in I} x_j d_j \right]$$

Số hạng thứ nhất tương ứng với chi phí truyền các cập nhật từ các vị trí (S_i) đến mọi vị trí có giữ bản sao của mảnh F và với chi phí thực hiện các yêu cầu chỉ đọc tại vị trí đó (S_i) nhưng với chi phí truyền dữ liệu thấp nhất. Số hạng thứ hai tính tổng chi phí lưu tất cả bản sao của mảnh F.

Cách đặt vấn đề hết sức đơn giản và không phù hợp lắm với thực tế thiết kế cơ sở dữ liệu phân tán. Đây thực chất là mô hình *bài toán cấp phát tập tin* - FAP (*file allocation problem*) cho mạng máy tính. Để phân biệt với bài toán trên, ta gọi bài toán cấp phát mảnh trong thiết kế cơ sở dữ liệu phân tán là *bài toán cấp phát cơ sở dữ liệu* - DAP (*database allocation problem*). Trong thực tế, vì bài toán quá phức tạp nên ta chỉ nghiên cứu các *phương pháp heuristic* để tìm lời giải gần tối ưu.

2.4.2. Yêu cầu về thông tin

Ở giai đoạn cấp phát, ta cần thông tin định lượng về cơ sở dữ liệu, về các ứng dụng chạy trên đó, về cấu trúc mạng, khả năng xử lý và giới hạn lưu trữ của mỗi vị trí trên mạng.

a) Thông tin về cơ sở dữ liệu

Cho mảnh F_j và ứng dụng q_i. Độ tuyển của F_j ứng với ứng dụng q_i, ký hiệu $sel_i(F_j)$, là số lượng các bộ của F_j mà q_i truy xuất để xử lý.

Kích thước của mảnh F_j cũng được định nghĩa thông qua lực lượng $card(F_j)$ và độ dài $length(F_j)$ như sau

$$size(F_j) = card(F_j) * length(F_j)$$

b) Thông tin về ứng dụng

Phần lớn các thông tin về ứng dụng đều đã được biên dịch trong khi thực hiện phân mảnh. Hai số liệu quan trọng là

RR_{ij} số truy xuất chỉ đọc bởi câu vấn tin q_i thực hiện trên mảnh F_j,
UR_{ij} số truy xuất cập nhật bởi câu vấn tin q_i thực hiện trên mảnh F_j.

Ta cũng định nghĩa hai ma trận UM và RM với các phần tử tương ứng u_{ij} và r_{ij} như sau

$$u_{ij} = \begin{cases} 1, \text{ nếu } q_i \text{ cập nhật } F_j \\ 0, \text{ nếu ngược lại} \end{cases}$$

$$r_{ij} = \begin{cases} 1, \text{ nếu } q_i \text{ cần đọc } F_j \\ 0, \text{ nếu ngược lại} \end{cases}$$

Tiếp theo, vectơ $O = \{o(1),..., o(k)\}$ gồm các thành phần $o(i)$ đặc tả vị trí đưa ra câu vấn tin q_i (i = 1, .., k).

Cuối cùng để định nghĩa ràng buộc thời gian đáp ứng, thời gian đáp ứng tối đa được phép của mỗi ứng dụng cũng cần được đặc tả.

c) Thông tin về vị trí

Với mỗi vị trí, chúng ta cần biết về khả năng lưu trữ và xử lý của nó. Ta ký hiệu

USC_k là chi phí lưu 1 đơn vị dữ liệu tại S_k.
LPC_k là chi phí xử lý 1 đơn vị dữ liệu tại S_k.

d) Thông tin về mạng

Chi phí truyền dữ liệu được định nghĩa theo đơn vị là *bó dữ liệu* (frame). Ký hiệu

g_{ij} là chi phí truyền 1 bó dữ liệu giữa hai vị trí S_i và S_j.

2.4.3. Mô hình cấp phát

Chúng ta sẽ thảo luận một mô hình cấp phát có mục tiêu là giảm thiểu tổng chi phí xử lý và lưu trữ trong khi vẫn cố gắng đáp ứng được các đòi hỏi về thời gian đáp ứng. Mô hình có dạng sau

min (Tổng chi phí)

với ràng buộc

ràng buộc thời gian đáp ứng, ràng buộc lưu trữ, ràng buộc xử lý.

Ta sẽ khai triển các thành phần của mô hình này. Ký hiệu biến quyết định

$$x_{ij} = \begin{cases} 1, \text{ nếu } F_i \text{ phân cho } S_j \\ 0, \text{ nếu ngược lại} \end{cases}$$

- ***Tổng chi phí***

Hàm tổng chi phí có hai thành phần: phần xử lý vấn tin và phần lưu trữ. Vì vậy nó có thể biểu diễn là

$$TOC = \sum_{q_i \in QQ} QPC_i + \sum_{S_k \in SS} \sum_{F_j \in FF} STC_{jk}$$

với

QPC_i là chi phí xử lý câu vấn tin của ứng dụng q_i

STC_{jk} là chi phí lưu mảnh F_j tại vị trí S_k.

Ta xét chi phí lưu trữ trước. Nó được cho bởi

$$STC_{jk} = USC_k * size(F_j) * x_{jk}$$

và hai ký hiệu tổng là tìm các tổng chi phí lưu trữ tại tất cả vị trí cho tất cả các mảnh.

Chi phí xử lý vấn tin khó xác định hơn. Ta tách QPC_i thành hai thành phần chi phí xử lý *PC* và chi phí truyền *TC*

$$QPC_i = PC_i + TC_i$$

Thành phần xử lý *PC* gồm 3 hệ số chi phí: chi phí truy xuất *AC,* chi phí toàn vẹn *IE* và chi phí điều khiển đồng thời *CC*:

$$PC_i = AC_i + IE_i + CC_i$$

Mô tả chi tiết cho mỗi hệ số chi phí phụ thuộc thuật toán dùng để hoàn tất các tác vụ đó. Chi tiết về chi phí truy xuất AC như sau

$$AC_i = \sum_{S_k \in SS} \sum_{F_j \in FF} \left(u_{ij} * UR_{ij} + r_{ij} * RR_{ij} \right) * x_{jk} * LPC_k$$

Hai số hạng đầu trong công thức trên tính số truy xuất của vấn tin q_i đến mảnh F_j. Chú ý rằng

$$\left(u_{ij} * UR_{ij} + r_{ij} * RR_{ij}\right)$$

là tổng số các truy xuất cập nhật và đọc mảnh F_j. Giả thiết chi phí xử lý là giống nhau với mọi mảnh. Ký hiệu tổng cho biết tổng số các truy xuất cho tất cả các mảnh được q_i tham chiếu. Nhân với LPC_k cho ra chi phí của truy xuất này tại vị trí S_k. Ta dùng x_{jk} để tách giá trị chi phí cho các vị trí có lưu các mảnh.

Hệ số chi phí duy trì tính toàn vẹn cơ sở dữ liệu có thể mô tả giống thành phần xử lý ngoại trừ chi phí xử lý cục bộ một đơn vị cần được thay đổi nhằm phản ánh chi phí thực sự để duy trì tính toàn vẹn. Ta sẽ quay lại vấn đề này ở chương sau.

Hàm chi phí truyền dữ liệu có thể biểu diễn giống hàm chi phí truy xuất. Tuy nhiên tổng chi phí truyền dữ liệu cho cập nhật và cho yêu cầu chỉ đọc sẽ khác nhau hoàn toàn. Trong vấn tin cập nhật, ta cần cho tất cả mọi vị trí biết nơi có các bản sao, còn trong vấn tin chỉ đọc thì chỉ cần truy xuất đến 1 trong các bản sao là đủ. Ngoài ra vào lúc kết thúc yêu cầu vấn tin cập nhật thì không cần truyền dữ liệu ngược lại cho vị trí đưa ra vấn tin ngoài một thông báo xác nhận, còn trong vấn tin chỉ đọc có thể phải có nhiều thông báo truyền dữ liệu.

Thành phần cập nhật hàm truyền dữ liệu là

$$TCU_i = \sum_{S_k \in SS} \sum_{F_j \in FF} u_{ij} * x_{jk} * g_{o(i),k} + \sum_{S_k \in SS} \sum_{F_j \in FF} u_{ij} * x_{jk} * g_{k,o(i)}$$

Số hạng thứ nhất để gửi thông báo cập nhật từ vị trí gốc $o(i)$ của q_i đến tất cả bản sao cần cập nhật. Số hạng thứ hai dành cho thông báo xác nhận.

Thành phần chi phí chỉ đọc là

$$TCR_i = \sum_{F_j \in FF} \min_{S_k \in SS} \left(r_{ij} * x_{jk} * g_{o(i),k} + r_{ij} * x_{jk} * \frac{sel_i(F_j) * length(F_j)}{fsize} * g_{k,o(i)} \right)$$

Số hạng thứ nhất trong TCR biểu thị chi phí truyền yêu cầu chỉ đọc đến những vị trí có bản sao của mảnh cần truy xuất. Số hạng thứ hai biểu thị chi phí để truyền các kết quả từ những vị trí này đến vị trí yêu cầu. Phương trình này khẳng định rằng trong số các vị trí có bản sao của cùng 1 mảnh, chỉ vị trí có tổng chi phí truyền thấp nhất mới được chọn để thực hiện thao tác này.

Bây giờ hàm chi phí truyền dữ liệu của câu vấn tin q_i có thể được tính là

$$TC_i = TCU_i + TCR_i$$

- ***Ràng buộc***

Các hàm ràng buộc có thể được đặc tả tương tự. Tuy nhiên thay vì mô tả những hàm này kỹ lưỡng, chúng ta chỉ nêu những ý tổng quát.

Ràng buộc thời gian đáp ứng cần được đặc tả là

$$\textit{thời gian thực thi của } q_i \leq \textit{thời gian đáp ứng lớn nhất của } q_i\ , \forall q_i \in QQ.$$

Người ta thường đặc tả số đo chi phí của hàm theo thời gian vì nó đơn giản.

Ràng buộc lưu trữ là

$$\sum_{F_j \in FF} STC_{jk} \leq \textit{khả năng lưu trữ tại vị trí } S_k, \forall S_k \in SS$$

Ràng buộc xử lý là

$$\sum_{q_i \in QQ} \textit{tải trọng xử lý của } q_i \textit{ tại } S_k \leq \textit{khả năng xử lý tại vị trí } S_k, \forall S_k \in SS$$

Mô hình bài toán cấp phát có lời giải phi đa thức, vì thế người ta tìm các phương pháp *heuristic* cho lời giải gần tối ưu. Có sự tương ứng giữa bài toán cấp phát và bài toán chọn vị trí đặt thiết bị đã được khám phá trong các nghiên cứu về quá trình điều hành sản xuất.

BÀI TẬP

1. Cho quan hệ

EMP

ENO	ENAME	TITLE
E1	J.Doe	Elect.Eng.
E2	M.Smith	Syst.Anal.
E3	A.Lee	Mech.Eng.
E4	J.Miller	Programmer
E5	B.Casey	Syst.Anal.
E6	L.Chu	Elect.Eng.
E7	R.David	Mech.Eng.
E8	J.Jones	Syst.Anal.

Ký hiệu p_1 là vị từ TITLE < "Programmer" và p_2 là vị từ TITLE > "Programmer". Giả thiết chuỗi ký tự được sắp theo thứ tự từ điển.

a) Thực hiện phân mảnh ngang cho quan hệ EMP ứng với $\{p_1, p_2\}$.

b) Giải thích tại sao phân mảnh kết quả $\{EMP_1, EMP_2\}$ không đáp ứng quy tắc đúng đắn của phân mảnh.

c) Sửa lại các vị từ p_1 và p_2 để chúng phân hoạch EMP theo các quy tắc đúng đắn của phân mảnh.

Hướng dẫn. Sửa các vị từ bằng cách xây dựng các vị từ tiểu hạng và suy ra các phép kéo theo tương ứng với thực hiện phân mảnh ngang của EMP dựa trên các vị từ tiểu hạng này. Cuối cùng chứng tỏ kết quả có tính đầy đủ, tính tái thiết và tính tách biệt.

2. Xét quan hệ

ASG

ENO	PNO	RESP	DUR
E1	P1	Manager	12
E2	P1	Analyst	24
E2	P2	Analyst	6
E3	P3	Consultant	10
E3	P4	Engineer	48
E4	P2	Programmer	18
E5	P2	Manager	24
E6	P4	Manager	48
E7	P3	Engineer	36
E8	P3	Manager	40

Giả sử có hai ứng dụng truy xuất ASG. ứng dụng thứ nhất được đưa ra tại 5 vị trí và muốn tìm thời gian phân công các nhân viên khi biết mã số nhân viên. Giả sử rằng nhà quản lý (Manager), nhà tư vấn (Consultant), kỹ sư (Engineer) và lập trình viên (Programmer) được lưu tai 4 vị trí khác nhau. ứng dụng thứ hai được đưa ra tại 2 vị trí trong đó các nhân viên có thời gian phân công dưới 20 tháng được quản lý tại một vị trí còn nhứng người trên 20 tháng được quản lý tại vị trí thứ hai. Hãy phân mảnh ngang nguyên thủy cho ASG theo các thông tin trên.

3. Xét các quan hệ sau

EMP

ENO	ENAME	TITLE
E1	J.Doe	Elect.Eng.
E2	M.Smith	Syst.Anal.
E3	A.Lee	Mech.Eng.
E4	J.Miller	Programmer
E5	B.Casey	Syst.Anal.
E6	L.Chu	Elect.Eng.
E7	R.David	Mech.Eng.
E8	J.Jones	Syst.Anal.

PAY

TITLE	SAL
Elect.Eng.	40000
Syst.Anal.	34000
Mech.Eng.	27000
Programmer	24000

Giả sử EMP và PAY được phân mảnh ngang như sau:

$EMP_1 = \sigma_{TITLE="Elect.Eng."}(EMP)$
$EMP_2 = \sigma_{TITLE="Syst.Anal"}(EMP)$
$EMP_3 = \sigma_{TITLE="Mech.Eng."}(EMP)$
$EMP_4 = \sigma_{TITLE="Programmer."}(EMP)$
$PAY_1 = \sigma_{SAL >=30000}(PAY)$
$PAY_2 = \sigma_{SAL < 30000}(PAY)$

Vẽ đồ thị nối nửa của $E ><_{TITLE} PAY$. Đây là đồ thị đơn giản hay phân hoạch ? Nếu nó phân hoạch, hãy sửa lại phân mảnh của EMP hoặc PAY để có đồ thị nối nửa $E ><_{TITLE} PAY$ đơn giản.

4. Xét quan hệ PAY và EMP ở bài tập trên. Ký hiệu p_1 là vị từ SAL < 30000 và p_2 là vị từ SAL >= 30000. Thực hiện phân mảnh ngang cho PAY ứng với p_1 và p_2 để có được hai mảnh PAY_1 và PAY_2. Sử dụng phân mảnh của PAY thực hiện phân mảnh ngang dẫn xuất cho EMP. Chứng tỏ tính đầy đủ, tính tái thiết và tính tách biệt của phân mảnh EMP.

5. Xét các quan hệ EMP, ASG ở các bài tập trên. Giả sử định nghĩa khung nhìn sau

```
CREATE   VIEW    EMPVIEW(ENO, ENAME, PNO, RESP)
AS       SELECT  EMP.ENO, EMP.ENAME, ASG.PNO, ASG.RESP
         FROM    EMP, ASG
         WHERE   EMP.ENO = ASG.ENO
           AND   ASG.DUR = 24
```

được truy xuất bởi ứng dụng q_1 nằm tại các vị trí 1 và 2 với tần suất lần lượt là 10 và 20. Giả sử tiếp rằng có một vấn tin q_2 khác được định nghĩa là

```
SELECT   ENO, DUR
FROM     ASG
```

chạy tại các vị trí 2 và 3 với tần số tương ứng là 20 và 10. Dựa trên những thông tin này hãy xây dựng ma trận $use(q_i, A_j)$ cho các thuộc tính của cả hai quan hệ

EMP và ASG. Xây dựng ma trận ái lực chứa tất cả các thuộc tính của EMP và ASG. Cuối cùng, hãy biến đổi ma trận ái lực để có thể dùng nó khi tách các quan hệ này thành hai mảnh dọc bằng một heuristic hoặc thuật toán BEA.

6. Giả sử môi trường làm việc như ở bài tập 5. Giả sử 60% truy xuất của q_1 là cập nhật đến PNO và RESP của khung nhìn EMPVIEW, trong khi ASG.DUR không được cập nhật qua khung nhìn EMPVIEW. Ngoài ra, giả sử rằng tốc độ truyền dữ liệu giữa vị trí 1 và vị trí 2 chỉ bằng một nữa tốc độ truyền dữ liệu giữa vị trí 2 và vị trí 3. Dựa vào những thông tin trên hãy tìm một phân mảnh hợp lý của ASG và EMP và cách nhân bản và vị trí đặt dữ liệu tối ưu cho các mảnh, giả thiết rằng chi phí lưu trữ không đáng kể, nhưng các bản sao phải nhất quán.

Hướng dẫn. Xét phân mảnh ngang cho ASG dựa trên vị từ DUR=24 và phân mảnh ngang dẫn xuất tương ứng cho EMP. Cần xemét ma trận ái lực nhận được trong bài tập 5 cho EMP và ASG và xét xem nó có ý nghĩa gì không trong việc thực hiện phân mảnh dọc cho ASG.

7. Cho thí dụ về ma trận CA, trong đó điểm tách không duy nhất và phân hoạch nằm ngay giữa ma trận. Cho biết số các thao tác xê dịch cần thực hiện để có điểm tách duy nhất.

8. Gọi $Q = \{q_1, q_2, q_3, q_4, q_5\}$ là tập vấn tin, $A = \{A_1, A_2, A_3, A_4, A_5\}$ là tập thuộc tính và $S = \{S_1, S_2, S_3\}$ là tập vị trí. Ma trận giá trị sử dụng các thuộc tính và ma trận tần số truy xuất ứng dụng lần lượt là

	A_1	A_2	A_3	A_4	A_5
q_1	1	1	1	0	1
q_2	1	1	1	0	1
q_3	1	0	0	1	1
q_4	0	0	1	0	0
q_5	1	1	1	0	0

và

	S_1	S_2	S_3	
q_1	10	20	0	q_1
q_2	5	0	10	q_2
q_3	0	35	5	q_3
q_4	0	10	0	q_4
q_5	0	15	0	q_5

Giả thiết $ref_l(q_k) = 1$ cho mọi q_k và S_l, A_1 là thuộc tính khóa. Sử dụng các thuật toán năng lượng nối và phân hoạch dọc để có được một phân mảnh dọc cho tập thuộc tính trong A.

Bài giảng

Cơ sở dữ liệu quan hệ

Bộ môn Tin học Cơ bản, Khoa CNTT, ĐH Mỏ - Địa Chất

MỤC LỤC

CH□ƠNG 1. TỔNG QUAN VỀ CƠ SỞ DỮ LIỆU

Hệ quản trị cơ sở dữ liệu, đặc biệt là cơ sở dữ liệu quan hệ là một hệ thống phần mềm có vai trò quan trọng trong các hệ thống lập trình. Cũng giống như các loại phần mềm hệ thống chủ yếu khác như: trình biên dịch và hệ điều hành, các nguyên lý của hệ quản trị cơ sở dữ liệu đã được phát triển từ khá lâu. Những khái niệm này rất hữu ích, không những giúp cho việc sử dụng hiệu quả các hệ quản trị cơ sở dữ liệu mà còn hỗ trợ trong việc thiết kế và cài đặt chúng.

Bài 1. Các khái niệm cơ bản về cơ sở dữ liệu (2 tiết)

1. Các hệ thống tệp truyền thống

1.1. Bài toán:

Giả thiết bài toán quản lý thư viện có hai chức năng như sau:

- Chức năng 1 (CN1): In ra danh mục sách với tệp sách chứa các thông tin: tên sách, mã sách, tên tác giả, nhà xuất bản,...
- Chức năng 2 (CN2): Thống kê mỗi loại sách có trong thư viện với số lượng cụ thể.

Để giải quyết bài toán trên có thể có hai cách:

Cách 1: Viết CN2 độc lập CN1. Với CN2 tạo một tệp sách phân loại (tệp 2) chứa các thông tin: mã sách, tên sách, nhà xuất bản, số lượng,...

Cách này có một số nhược điểm như sau:

- Giữa tệp 1 và tệp 2 có một số trường trùng nhau dẫn tới dư thừa dữ liệu
- Dữ liệu lưu lặp ở cả hai tệp, khi dữ liệu thay đổi phải cập nhật ở cả hai tệp. Dữ liệu không nhất quán khi tiến hành các thao tác sửa đổi (*hiện tượng dị thường*)

Cách 2: Tận dụng tệp sách đã có (tệp 1). Sửa tệp 1 thành tệp mới có chứa thêm trường số lượng. Đặc điểm của cách 2:

- Tránh được dư thừa và lặp lại dữ liệu
- Phải sửa lại CN1 để sử dụng tệp mới (tệp 2) -> chương trình phụ thuộc vào dữ liệu

1.2. Phương án đề xuất theo hướng mới

Cả hai cách trên đều có những hạn chế nhất định, do đó cần đưa ra một phương án mới, hiệu quả hơn, khắc phục được những hạn chế trên đây, với mục đích xây dựng một hệ thống thoả mãn các yêu cầu sau:

- Tránh dư thừa dữ liệu
- Không phụ thuộc dữ liệu

- Các thao tác tra cứu, tìm kiếm, cập nhật nhanh chóng và hiệu quả

2. Khái niệm và phân loại các hệ quản trị cơ sở dữ liệu

2.1. Cơ sở dữ liệu là gì ?

Để dễ dàng cho việc giải thích các khái niệm, trước hết xem xét hệ thống bán vé máy bay bằng máy tính. Dữ liệu lưu trữ trong máy tính bao gồm thông tin về hành khách, chuyến bay, đường bay,v.v... Mọi thông tin về mối quan hệ này được biểu diễn trong máy thông qua việc đặt chỗ của khách hàng. Vậy làm thế nào để biểu diễn được dữ liệu đó và để đảm bảo cho khách hàng đi đúng chuyến.

Dữ liệu trên được lưu trữ trong máy theo một quy định nào đó và được gọi là *cơ sở dữ liệu* (viết tắt CSDL, tiếng Anh là *Database*)

Theo một định nghĩa khác: cơ sở dữ liệu là bộ lưu trữ các dữ liệu tác nghiệp của một xí nghiệp, được lưu trữ để phục vụ cho các ứng dụng.

Ví dụ 2.1: Xí nghiệp là một thư viện, dữ liệu tác nghiệp là: sách, độc giả, yêu cầu,v.v...

2.2. Hệ quản trị cơ sở dữ liệu

Phần chương trình để có thể xử lý, thay đổi cơ sở dữ liệu gọi là *Hệ quản trị cơ sở dữ liệu* (viết tắt HQTCSDL, tiếng Anh là *Database management system*). Theo định nghĩa này HQTCSDL có nhiệm vụ rất quan trọng như là một bộ *diễn dịch* (*interpreter*) với ngôn ngữ bậc cao nhằm giúp người sử dụng có thể dùng được hệ thống mà ít nhiều không cần quan tâm đến thuật toán chi tiết hoặc biểu diễn dữ liệu trong máy.

Theo một cách hiểu khác:

HQTCSDL là một phần mềm cho phép tạo lập CSDL và điều khiển hoặc truy nhập CSDL đó, đặc biệt HQTCSDL đảm bảo tính độc lập dữ liệu (là sự bất biến của các chương trình ứng dụng đối với các thay đổi về cấu trúc lưu trữ và chiến lược truy nhập).

Ví dụ 2.2: Một số Hệ QTCSDL thông dụng hiện nay: MS Access, SQL Server (của hãng Microsoft), Oracle (của hãng Oracle), DB2, FoxPro,v.v...

a) Hệ quản trị cơ sở dữ liệu hỗ trợ các tính năng sau:

- Định nghĩa dữ liệu (*Database definition*)
- Xây dựng dữ liệu (*Database construction*) : Chức năng định nghĩa và xây dựng dữ liệu hỗ trợ người dùng xây dựng các bộ dữ liệu riêng.
- Thao tác dữ liệu (*Database manipulation*): các thao tác cập nhật, tìm kiếm, sửa, xoá,...

- Quản trị dữ liệu (*Database administrator*): phân quyền sử dụng, bảo mật thông tin,...
- Bảo vệ dữ liệu (*Database protection*): thực hiện các thao tác sao chép, phục hồi, tránh mất mát dữ liệu.

b) Ngôn ngữ của hệ quản trị cơ sở dữ liệu bao gồm:

- Ngôn ngữ con định nghĩa dữ liệu (*Database Definition Language - DDL*): cung cấp các câu lệnh cho phép mô tả, định nghĩa các đối tượng của CSDL.
- Ngôn ngữ con thao tác dữ liệu (*Database Manipulation Language - DML*): dùng để thao tác, xử lý trên các đối tượng của CSDL như thêm, xoá, sửa, tìm kiếm,v.v...
- Ngôn ngữ con kiểm soát dữ liệu (*Database Control Language - DCL*): điều khiển tính đồng thời (tương tranh) đối với dữ liệu.

Hình 1- Hệ CSDL đa người dùng

c) Hệ quản trị cơ sở dữ liệu đa người dùng:

- Cơ sở dữ liệu ở dạng hợp nhất (hay là tập hợp toàn bộ dữ liệu tác nghiệp của xí nghiệp), có hai tính chất cơ bản sau:
  - Không dư thừa dữ liệu: cố gắng tối thiểu và kiểm soát sự dư thừa.
  - Sử dụng chung nguồn dữ liệu: chia sẻ nhiều người sử dụng.
- Người sử dụng hệ QTCSDL bao gồm:
  - Người phân tích hệ thống (*System analyst*)
  - Người thiết kế CSDL (*Database deginer*)
  - Người viết chương trình ứng dụng (*Application programer*): xây dựng các chương trình ứng dụng dựa trên các cơ sở dữ liệu đã có.
  - Người sử dụng cuối (*end - user*): là người truy nhập vào CSDL từ một thiết bị đầu cuối.
  - Người quản trị CSDL (*Database administrator*): thường là một người hoặc một nhóm người có nhiệm vụ điều khiển toàn bộ hệ CSDL.

2.3. Phân loại các hệ quản trị cơ sở dữ liệu

Hệ QTCDSL có thể được phân loại dựa theo nhiều tiêu chí khác nhau, và mỗi cách phân loại có thể có nihều kiểu hệ QTCSDL. Trong phần này mô tả 3 kiểu hệ QTCSDL khác nhau được phân loại theo quan điểm chung nhất.

a) Hệ QTCSDL loại 1:

Người dùng vừa thiết kế, sử dụng, và quản trị, hay còn gọi là hệ QTCSDL đơn người dùng (hệ CSDL cá nhân, nhỏ). Thường sử dụng để giải quyết những nhiệm vụ đơn lẻ với một người hoặc một vài người.

Hình 2- Mô hình hệ QTCSDL đơn người dùng

b) Hệ QTCSDL loại 2:

Người sử dụng cuối truy nhập CSDL thông qua các thiết bị truy nhập đầu cuối (*terminal*), còn gọi là hệ QTCSDL đa người dùng (hệ QTCSDL trung tâm).

Hình 3- Mô hình hệ QTCSDL đa người dùng

c) Hệ QTCSDL loại 3:

Những yêu cầu của người sử dụng cuối và trình ứng dụng được xử lý tại các máy trạm (Client), chỉ những yêu cầu nào cần tới CSDL mới được chuyển tới hệ QTCSDL nằm trên máy chủ (Server), đây là mô hình hệ QTCSDL Client / Server.

Hình 4 - Mô hình hệ QTCSDL Client / Server

Nhận xét: Cả 3 loại hệ QTCSDL trên đều đặt CSDL tại một nơi. Do đó hệ QTCSDL có tính chất tập trung.

Ví dụ 2.3:

- Một công ty muốn lưu trữ và duy trì thông tin về các nhân viên cung cấp (*supplier*) và các mặt hàng (*part*).
- Các thông tin lưu trữ cần thiết về nhân viên cung cấp bao gồm: số hiệu nhân viên, họ tên, ngày sinh, mức lương, và địa chỉ thành phố.
- Thông tin về mặt hàng bao gồm: số hiệu mặt hàng, tên mặt hàng, mầu sắc và giá.
- Việc chuyển hàng được mô tả thông qua số hiệu nhân viên, tên mặt hàng và số lượng.

NHAN_VIEN

s#	Ho_Ten	Thanh_Pho	Nam_Sinh	Luong
10	Lê Văn A	Ha Noi	1960	400
20	Hoàng Thị B	HCM	1970	500
30	Lê Văn Sơn	Hai Phong	1945	600

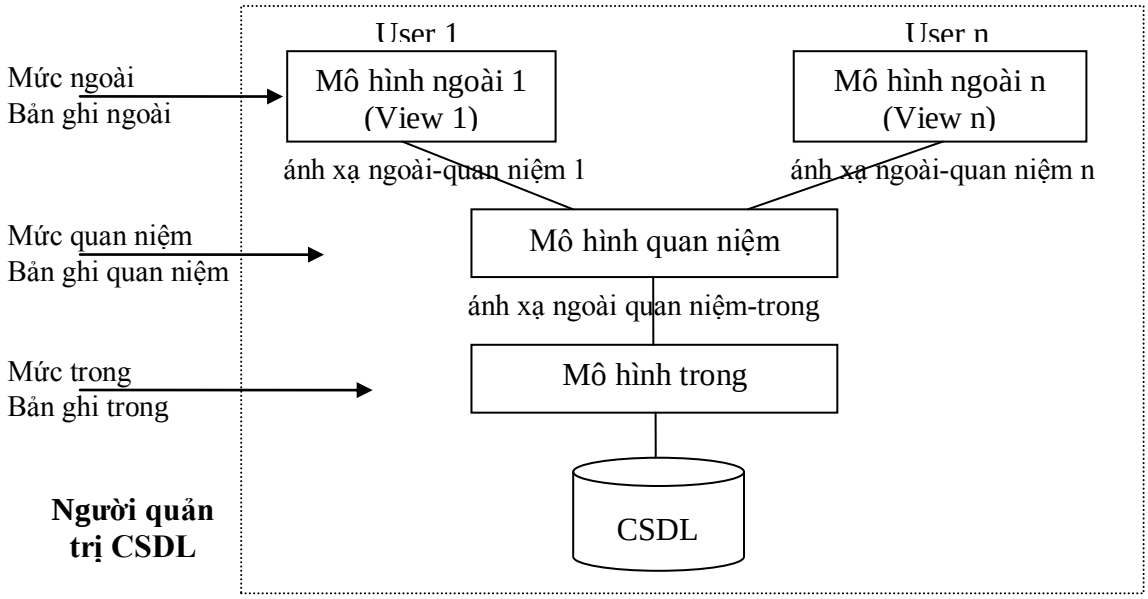
Hình 5 - Quan hệ NHÂN VIÊN

Bài 2. Kiến trúc một hệ cơ sở dữ liệu (1 tiết)

1. Mô hình kiến trúc 3 mức

Một CSDL được phân thành các mức khác nhau như trên hình. ở đây có thể xem như chỉ có một CSDL đơn giản và có một hệ phần mềm QTCSDL.

Người dùng truy nhập thông qua mô hình ngoài, qua ánh xạ ngoài quan niệm đi vào mô hình quan niệm, qua ánh xạ quan niệm trong vào mô hình trong, và truy nhập vào CSDL thực hiện các yêu cầu. Sau khi thực hiện các yêu cầu, ánh xạ ngược trở lại phía người dùng.



Hình 6 - Kiến trúc hệ QTCSDL

Đây là kiến trúc chuẩn 3 mức:

- **Mức ngoài:** là mức sát với người dùng. Mức này tương ứng với cách nhìn riêng của người dùng đối với hệ QTCSDL.
- **Mức trong (mức vật lý):** là mức sát với lưu trữ thực sự của các tệp dữ liệu theo một cấu trúc nào đó trên các thiết bị nhớ thứ cấp (như đĩa từ, băng từ...).
- **Mức khái niệm:** là mức trung gian. CSDL mức khái niệm là một sự biểu diễn trừu tượng của CSDL vật lý (hay có thể nói: CSDL mức vật lý là sự cài đặt cụ thể của CSDL mức khái niệm).

- **Các khung nhìn (view):** là cách nhìn hay quan niệm của từng người sử dụng đối với CSDL mức khái niệm. Sự khác nhau giữa khung nhìn và mức khái niệm thực chất là không lớn.

Tương ứng với 3 mức trên là 3 mô hình cụ thể của hệ QTCSDL bao gồm: mô hình ngoài, mô hình quan niệm, mô hình trong. Các mô hình này tương tác thông qua các ánh xạ.

(1) Mô hình ngoài: của một người dùng là tập hợp tất cả dữ liệu của người dùng được phép nhìn thấy và truy nhập vào, thường chỉ là một bộ phận của toàn bộ CSDL.

(2) Mô hình quan niệm: bao gồm tập toàn bộ dữ liệu của xí nghiệp và được biểu diễn ít nhiều trừu tượng.

(3) Mô hình trong: bao gồm tập toàn bộ dữ liệu của xí nghiệp được biểu diễn sát với lưu trữ thật sự trên các thiết bị nhớ.

Mô hình ngoài thường được biểu diễn thông qua bản ghi gọi là bản ghi ngoài, tương ứng với mô hình quan niệm và mô hình trong ta có bản ghi quan niệm và bản ghi trong.

Do có sự khác nhau giữa bản ghi ngoài và bản ghi quan niệm, để ánh xạ tới người dùng cần có ánh xạ ngoài-quan niệm. Tương ứng cần có ánh xạ quan niệm-trong giữa mô hình quan niệm và mô hình trong.

Thực chất các mô hình ngoài, mô hình quan niệm và mô hình trong là các sơ đồ kiểu bản ghi, tương ứng các ánh xạ là ánh xạ từ kiểu bản ghi này sang kiểu bản ghi khác.

Hệ QTCSDL trên được quản lý bởi người quản trị CSDL:

- Là người quyết định nội dung thông tin của CSDL, được hoàn thành thông qua viết sơ đồ quan niệm hay biểu diễn mô hình quan niệm.
- Là người quyết định cấu trúc lưu trữ và số lượng truy nhập đối với toàn bộ dữ liệu thông qua viết sơ đồ trong.
- Với sự giúp đỡ của người sử dụng, viết sơ đồ ngoài tương ứng với người sử dụng.
- Đưa ra các phương án sao lưu, phục hồi cho CSDL.
- Đưa ra các cánh kiểm soát thẩm quyền và kiểm tra tính đúng đắn của CSDL.

2. Ưu điểm của kiếm trúc 3 mức

Tại sao phân chia hệ QTCSDL thành 3 mức ? Việc phân chia như vậy có đạt được mục tiêu đặt ra không ?

Kiến trúc một hệ CSDL cần bảo đảm các mục tiêu sau:

(1) Tránh dư thừa dữ liệu

(2) Tính độc lập giữa dữ liệu và chương trình ứng dụng.

(3) Tra cứu, tìm kiếm, cập nhật dữ liệu nhanh chóng.

(4) Đảm bảo tính an toàn và toàn vẹn dữ liệu.

Chứng minh tính đúng đắn:

(1) Tránh dư thừa dữ liệu: Tất cả các mô hình ngoài (thực chất là ảo) đều truy nhập vào cùng CSDL (tính hợp nhất của dữ liệu) do đó đảm bảo không dư thừa dữ liệu.

(2) Tính độc lập:

Ba mức trong hệ QTCSDL (mức ngoài, mức quan niệm và mức trong) độc lập với nhau, khi xoá hay thêm logic diễn ra trên mô hình quan niệm tương ứng là sự thay đổi vật lý diễn ra ở mô hình trong (cấu trúc lưu trữ), do đó đảm bảo tính độc lập logic (tính bất biến của ứng dụng với sự thay đổi cơ sở ở mức logic).

Ba mức đảm bảo tính độc lập: khi mô hình trong thay đổi chỉ cần sửa ánh xạ trong quan niệm, các mô hình quan niệm và ứng dụng của người dùng không cần thay đổi (đảm bảo tính độc lập vật lý). Tương tự, khi mô hình quan niệm thay đổi chỉ cần sửa ánh xạ ngoài quan niệm.

(3) Tra cứu, tìm kiếm và cập nhật dữ liệu nhanh chóng: Sử dụng các thuật toán (ví dụ: đánh chỉ số cho CSDL,...) cho phép tra cứu, tìm kiếm và cập nhật nhanh dữ liệu.

(4) Tính an toàn và toàn vẹn dữ liệu: Mô hình ngoài chính là kĩ thuật đảm bảo tính an toàn, toàn vẹn dữ liệu. Mô hình quan niệm là mô hình ổn định vì mô hình ngoài phụ thuộc người dùng, mô hình trong phụ thuộc thiết bị thứ cấp.

3. Kết luận

Trong chương này chúng ta đã làm quen và tìm hiểu các khái niệm cơ bản liên quan đến CSDL, kiến trúc và các ưu điểm của hệ QTCSDL so với cách lưu trữ trước đây. Vậy khi nào cần và khi nào không cần tới hệ QTCSDL ?

3.1. Tại sao cần hệ QTCSDL

- Để đảm bảo tính tiêu chuẩn hoá: các hệ CSDL khác nhau, dữ liệu của các chương trình ứng khác nhau dựa trên một tiêu chuẩn chung.
- Cung cấp các công cụ định nghĩa và thao tác dữ liệu linh hoạt

- Tích hợp với nhiều trình ứng dụng khác nhau: các ngôn ngữ lập trình, các ứng dụng hỗ trợ phân tích thiết kế, v.v...

3.2. Khi nào không cần hệ QTCSDL

- Khi chúng ta giải quyết các vấn đề đơn giản mà các chương trình ứng dụng có thể thực hiện tốt không cần tới hệ CSDL.
- Khi hệ thống CSDL không đáp ứng được yêu cầu về hiệu năng như: tốc độ, tính bảo mật, định dạng dữ liêu cần lưu trữ, v.v..
- Khi không cần thiết đa người dùng cùng truy nhập vào một CSDL chung.

3.3. Tài liệu tham khảo

- Lê Tiến Vương, *Nhập môn Cơ sở dữ liệu quan hệ*, NXB Thống kê, Chương1
- P. O' Neil, *Database - Principles, Programming, Performance*, Chương 1.1, 1.2
- R. Elmasri, S.B. Navathe, Fundamentals of Database Systems, Chương 1

CH□ƠNG 2. MÔ HÌNH DỮ LIỆU

Trong chương này, chúng ta sẽ xem xét các mô hình chính được sử dụng trong các hệ thống cơ sở dữ liệu. Trong đó đặc biệt nhấn mạnh về mô hình thực thể liên kết, mô hình này được dùng chủ yếu làm công cụ thiết kế CSDL.

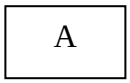
Bài 1. Sơ đồ thực thể liên kết (1 tiết)

1. Các khái niệm chung

1.1. Thực thể

Thực thể được định nghĩa là một đối tượng có thực hay trừu tượng mà ta muốn lưu trữ thông tin về nó.

Để biểu diễn (hay kí hiệu) thực thể: sử dụng hình chữ nhật bao quanh tên thực thể (tên thực thể biểu diễn là danh từ).



A: tên thực thể, là danh từ

1.2. Thuộc tính

Một thực thể được xây dựng bởi một tập thuộc tính đặc trưng cho thực thể.

Biểu diễn:

- a_1 ai : thuộc tính thứ i của thực thể A (là danh từ)
- .
- ai
- .
- an

Ví *dụ 1.2:* Thực thể sách gồm các thuộc tính: mã sách, tên sách, tác giả, nhà xuất bản

1.3. Liên kết thực thể

Liên kết thực thể chỉ mối quan hệ ràng buộc giữa các thực thể. Liên kết thực thể chia thành 3 loại như sau:

a) Kiểu liên kết một thực thể:

Là liên kết của thực thể đó với chính nó (hay còn gọi là liên kết đệ quy)

Biểu diễn:

b) Kiểu liên kết hai thực thể:

Liên kết hai thực thể là mối liên kết giữa hai thực thể khác nhau. Tên của các liên kết được biểu diễn bằng các động từ. Có 3 loại liên kết hai thực thể:

- Liên kết 1 -1: là liên kết thoả mãn điều kiện nếu xuất hiện một thực thể A thì xuất hiện một thực thể B hoặc ngược lại. Biểu diễn như sau:
- Liên kết 1 - nhiều: là liên kết thoả mãn điều kiện nếu xuất hiện một thực thể A thì xuất hiện nhiều thực thể B hoặc ngược lại, nếu xuất hiện một thực thể B thì xuất hiện nhiều thực thể A. Biểu diễn như sau:

Ví dụ 1.3.1: Xét quan hệ giữa nhân viên và phòng trong một công ty. Đây là quan hệ một nhiều (một nhân viên thuộc chỉ một phòng còn một phòng có thể có nhiều nhân viên).

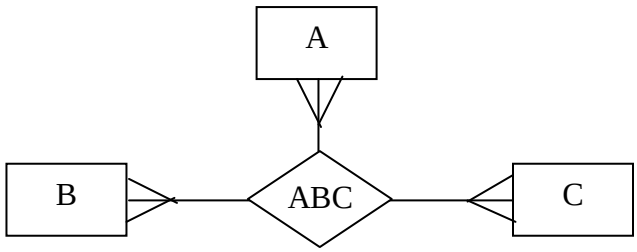
- Liên kết nhiều -nhiều: là liên kết thoả mãn điều kiện xuất hiện nhiều thực thể A thì xuất hiện nhiều thực thể B hoặc ngược lại. Biểu diễn như sau:

Ví *dụ 1.3.2:* Quan hệ giữa thực thể sách và thực thể độc giả là quan hệ nhiều - nhiều (giả thiết một độc giả có thể mượn nhiều loại sách cùng một lúc và một loại sách có nhiều bản sao có thể cho mượn).

c) Liên kết nhiều thực thể:

Là mối liên kết trong đó có nhiều hơn hai thực thể. Để biểu diễn liên kết nhiều thực thể và đơn giản hoá khi biểu diễn ta quy các liên kết nhiều thực thể này về các liên kết hai thực thể bằng cách đưa thêm vào thực thể trung gian (kí hiệu là: TG).

Biểu diễn liên kết nhiều - nhiều - nhiều như sau:



Khi thêm thực thể trung gian (TG), liên kết nhiều - nhiều - nhiều sẽ chuyển thành 3 liên kết 1 - nhiều (đây là liên kết hai thực thể). Biểu diễn như sau:

Theo định nghĩa liên kết thực thể ta có thể coi liên kết thực thể là một dạng thực thể đặc biệt và cũng được lưu trữ. Do đó CSDL phải lưu trữ cả hai đối tượng: thực thể và liên kết thực thể, đồng thời CSDL phải có khả năng biểu diễn hai đối tượng này.

Bài 2. Các mô hình dữ liệu (2 tiết)

Hiện nay có nhiều loại mô hình dữ liệu khác nhau. Trong bài này chúng ta sẽ chỉ nghiên cứu ba loại mô hình dữ liệu cơ bản thường được sử dụng bao gồm: mô hình quan hệ, mô hình mạng, và mô hình phân cấp.

1. Mô hình quan hệ (Relational model)

Mô hình này dựa trên cơ sở khái niệm lý thuyết tập hợp của các quan hệ, tức là tập các k - bộ với k cố định.

Hay nói một cách đơn giản hơn, trong mô hình quan hệ dữ liệu được biểu diễn dưới dạng các bảng, đặc biệt cả hai đối tượng: thực thể và liên kết thực thể được biểu diễn dưới dạng duy nhất là dạng bảng.

Như vậy sơ đồ thực thể liên kết được ánh xạ thành mô hình quan hệ:

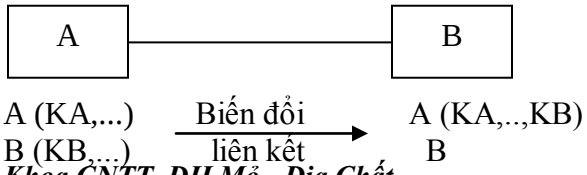
- Tên của thực thể được ánh xạ thành tên bảng, mỗi bảng biểu diễn một thực thể tương ứng.
- Thuộc tính của thực thể tương ứng với các cột trong bảng.

Ví *dụ 1.1:* Thực thể Nhân viên trong một công ty được biểu diễn dưới dạng bảng như sau:

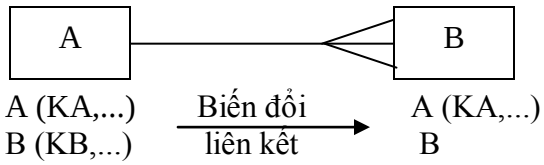
SS#	HoHo_Tenen	Thanhnh_Phoho	Namam_Sinhnh	Luongng
	Bùiïi Ngọcäc Anhnh	HàHµ Nộiéi	1960	600
11	NguyễnÔn Thanhnh HuyềnÒn	HàHµ Nộiéi	1965	500
12		Hải¶i Phòngng	1970	400
13	NguyễnÔn Quangng Trungng	HCMCM	1975	400

Hình 7 - Biểu diễn thực thể dưới dạng bảng

- Biểu diễn liên kết thực thể:
  - Trước tiên phải xác định khoá của thực thể, sau đó chuyển khoá này thành cột khoá của bảng tương ứng (*khái niệm khoá sẽ được giải thích rõ trong phần sau*).
  - Bước tiếp theo là biến đổi liên kết thực thể:
    Đối với liên kết 1 - 1: Lấy khoá của thực thể này đặt vào thực thể kia.



Đối với liên kết 1 - nhiều: Lấy khoá của bảng đầu liên kết một đặt vào bảng đầu

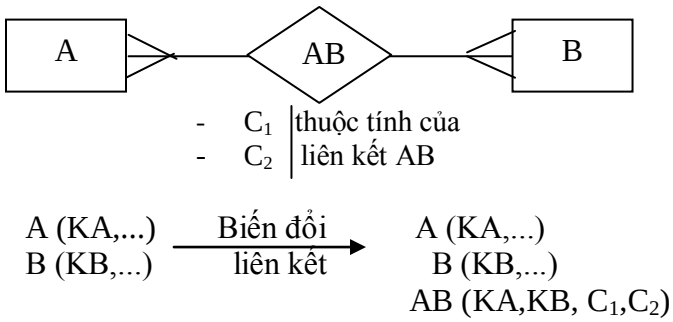


liên kết nhiều.

Đối với liên kết nhiều - nhiều: Thêm một bảng mới (bảng trung gian) gồm khoá của các bảng liên kết và thuộc tính liên kết

Ví dụ 1.2: Xét quan hệ giữa nhân viên và phòng trong một công ty. Đây là quan hệ một nhiều (một nhân viên thuộc chỉ một phòng còn một phòng có thể có nhiều nhân viên).

Mô hình quan hệ:



2. Mô hình mạng (Network model)

Dữ liệu được biểu diễn dưới dạng một đồ thị có hướng phức tạp (dạng đa danh sách).

Xây dựng một ánh xạ biến đổi sơ đồ thực thể liên kết sang mô hình mạng. Nó chỉ có khả năng biểu diễn trực tiếp đối với các liên kết 1 - 1 hay 1 - nhiều. Do đó để biểu diễn được các liên kết khác như liên kết nhiều - nhiều, cần quy các liên kết này về dạng 1-nhiều bằng cách đưa thêm một thực thể trung gian (TG) vào liên kết.

Chuyển thành liên kết 1- nhiều:

Với mô hình mạng:

- Thực thể chuyển thành kiểu bản ghi.
- Thuộc tính chuyển thành các trường của bản ghi.
- Liên kết giữa các thực thể: một liên kết 1 - nhiều tương ứng tập gán nhãn.
  - Kiểu bản ghi ở đầu 1 gọi là kiểu bản ghi chủ
  - Kiểu bản ghi ở đầu nhiều gọi là kiểu bản ghi thành viên

Biểu diễn:

Chiều mũi tên hướng tới đầu nhiều

Ví *dụ 2:* Xét quan hệ giữa nhân viên và phòng trong một công ty. Đây là quan hệ một nhiều (một nhân viên thuộc chỉ một phòng còn một phòng có thể có nhiều nhân viên).

3. Mô hình phân cấp (Hierachical model)

Mô hình dữ liệu là một cây, trong đó các nút biểu diễn các tập thực thể, giữa các nút con và nút cha được liên hệ theo một mối quan hệ xác định.

Mô hình phân cấp là trường hợp đặc biệt của mô hình mạng với một số điều kiện hạn chế sau:

- Không có chu trình.
- Mỗi một kiểu bản ghi chỉ là kiểu bản ghi thành viên nhiều nhất của một tập.

Với mô hình phân cấp khi biểu diễn không cần kí hiệu mũi tên. Trong cấu trúc cây, đầu một tương đương với nút gốc của cây, và đầu nhiều tương ứng với các nút lá của cây.

Ví *dụ 3:* Xét quan hệ giữa nhân viên và phòng trong một công ty như ví dụ trên.

4. Đánh giá và so sánh 3 loại mô hình dữ liệu

4.1. Cách biểu diễn dữ liệu

- Mô hình quan hệ: cả hai đối tượng thực thể và liên kết thực thể đều biểu diễn dưới dạng bảng
- Mô hình mạng: các đối tượng thực thể biểu diễn dưới dạng bản ghi, liên kết thực thể tương ứng là liên kết móc nối giữa các bản ghi dạng đồ thị có hướng
- Mô hình phân cấp: các đối tượng thực thể biểu diễn dưới dạng bản ghi, liên kết thực thể tương ứng là liên kết giữa các nút của cây

4.2. Cách thực hiện yêu cầu

Ví dụ 4: thực hiện các yêu cầu sau:

- Bổ sung thêm một quyển sách ('S03', 'Mạng máy tính', 'NT Hải') vào CSDL
- Xoá độc giả với mã độc giả là 'ĐG1'

Mô hình quan hệ:

- Với yêu cầu thứ nhất, hệ thống sẽ thêm một hàng trong bảng *Sách*.
- Với yêu cầu thứ hai để xoá độc giả có mã 'ĐG1', trước tiên sẽ xoá độc giả này trong bảng *Mượn* sau đó mới xoá trong bảng *Độc giả*.

Mô hình mạng và mô hình phân cấp:

- Các phép toán thao tác đối mô hình mạng và phân cấp phức tạp hơn đối với mô hình quan hệ (chỉ thao tác trên các bảng).

5. Kết luận và bài tập

Chương này đã trình bầy các khái niệm và sơ đồ thực thể liên kết, đặc biệt là ba loại mô hình dữ liệu: mô hình quan hệ, mô hình mạng, và mô hình phân cấp. Đây là các loại mô hình cơ bản để xây dựng các hệ QTCSDL hiện nay.

5.1. Tài liệu tham khảo

- Lê Tiến Vương, *Nhập môn Cơ sở dữ liệu quan hệ*, NXB Giáo dục, Chương1
- P. O' Neil, *Database - Principles, Programming, Performance*, Chương 1.3, 1.4
- R. Elmasri, S.B. Navathe, *Fundamentals of Database Systems*, Chương 3,4

5.2. Bài tập

Bài 1) Cho sơ đồ thực thể liên kết giữa công việc, nhân viên và phòng như dưới đây, hãy biến đổi sang 3 mô hình dữ liệu:
Bài 2) Cho sơ đồ thực thể liên kết giữa sách và độc giả như sau, hãy hoàn chỉnh sơ đồ sau đó biến đổi sang 3 mô hình dữ liệu.

5.3. Chữa bài tập

Bài 1) biến đổi sang 3 mô hình dữ liệu:

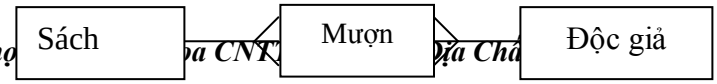
a) Mô hình quan hệ:

Công việc (Mã CV, Tên CV, Định mức)
Nhân Viên (Mã NV, Họ và tên, Ngày sinh, Mã CV, Mã phòng)
Phòng (Mã phòng, Tên phòng, Số phòng)

b) Mô hình mạng:

c) Mô hình phân cấp:

Bài 2) Chuyển sơ đồ quan hệ về dạng dưới đây, sau đó thực hiện tương tự:



CH□ƠNG 3. MÔ HÌNH DỮ LIỆU QUAN HỆ

Mô hình dữ liệu quan hệ là mô hình phổ biến và quan trọng, được E.Codd đưa ra vào năm 1970. Một trong những ưu điểm của mô hình dữ liệu quan hệ là có hỗ trợ các ngôn ngữ khai báo, khá đơn giản nhưng hiệu qủa và các phép toán trên dữ liệu. Các phép toán này có thể tổ hợp và phân tách dễ dàng nhờ vào một hệ thống kí hiệu đại số gọi là đại số quan hệ (relational algebra).

Bài 1. Các khái niệm cơ bản (1 tiết)

1. Thuộc tính và miền thuộc tính

- Theo khái niệm toán học, mô hình quan hệ được hiểu là quan hệ với ý nghĩa của lý thuyết tập hợp, tức là tập con tích Đề - Các của các miền. Với miền (*domain*) là một tập các giá trị. Ví dụ: tập các số nguyên là một miền, tập các xâu kí tự tạo thành tên người trong tiếng Anh có độ dài không quá 30 kí tự là một miền; tập 2 số {0,1} cũng là một miền v.v...

Gọi D_1, D_2, ... Dn, là n miền. Tích Đề - Các của n miền là D1xD2x...xDn là tập tất cả n - bộ (n - tuples) (v_1, v_2, ... vn) sao cho vi [redacted]i, với i = 1.. n

Ví dụ 1. 1: n =2, D_1 = {0,1}, D_2 = {a,b,c}, khi đó

D_1 x D_2 = {(0,a), (0,b), (0,c), (1,a), (1,b), (1,c)}

- Theo cách hiểu khác, miền thuộc tính là một tập các giá trị mà từ đó có thể rút ra những giá trị thực sự xuất hiện trong cột biểu diễn thuộc tính đó.

Ví dụ 1. 2: miền mã sách, kí hiệu DOM (Mã Sách)

DOM (Mã Sách) = {'S01', 'S02', ... , 'S99'}

miền tên tác giả, kí hiệu DOM (Tên TG)

DOM (Tên TG) = { char(40) }

2. Quan hệ

2.1. Định nghĩa 1

Cho n tập D_1, D_2, ... Dn, không nhất thiết phân biệt nhau, khi đó R được định nghĩa là 1 quan hệ trên n tập đó nếu R là một tập nào đó các bộ được sắp (d_1, d_2, ... dn) sao cho di ∈ Di

Ví dụ 2.1: quan hệ $r = \{ (d_1^i, d_2^i, ..., d_n^i) \mid i = 1..m \}$ là n bộ được sắp thứ i.

Đặt tên các cột là A_1, A_2, ... An:

A_1	.., Ai,..	An
d_1^1	.. , d_i^1 ,..	d_n^1
d_1^2	..., d_i^2 ,..	d_n^2
⋮	⋮	⋮
d_1^m	.., d_i^m ,..	d_n^m

Mỗi quan hệ có thể biểu diễn dưới dạng bảng, khi đó một dòng tương ứng với một bộ của quan hệ, một cột tương ứng với các giá trị của một thuộc tính.

Quan hệ	**Bảng**	**Tệp**
Bộ	Dòng	Bản ghi
Thuộc tính	Cột	Trường

Có hai đại lượng đo kích thước quan hệ:

- Số thuộc tính của quan hệ: gọi là bậc quan hệ
- Số bộ: gọi là lực lượng của quan hệ

2.2. Định nghĩa 2

Gọi R = {A_1,A_2,..., An} là tập hữu hạn của các thuộc tính, mỗi thuộc tính Ai với i = 1,..,n có miền giá trị tương ứng là DOM(A_1) . Khi đó r là quan hệ xác định trên n thuộc tính nếu r là tập con hoặc trùng với tích Đề Các của Di (i = 1..n)

r ⊆ D_1 x D_2 x ... x Dn

Khi đó kí hiệu là r(R) hoặc r(A_1,A_2,..., An).

Ví dụ 2.2: Hình cho thấy quan hệ NHAN_VIEN bao gồm các thuộc tính Ho_Ten, Thanh_Pho, Nam_Sinh và Luong là một quan hệ 4 ngôi.

NHAN_VIEN (Ho_Ten, Thanh_Pho, Nam_Sinh, Luong)

	Ho_Ten	Thanh_Pho	Nam_Sinh	Luong
t_1	Lê Văn A	Ha Noi	1960	400
t_2	Hoàng Thị B	HCM	1970	500
t_3	Lê Văn Sơn	Hai Phong	1945	600

Hình 8 - Quan hệ NHÂN VIÊN

t_1 = (Lê Văn A, Ha Noi, 1960, 400) là một bộ của quan hệ NHAN_VIEN

3. Khoá (Key)

3.1. Định nghĩa

Khoá của một quan hệ r xác định trên tập thuộc tính R = {A_1,A_2,..., An} là tập con K ⊆ {A_1,A_2,..., An} nếu với mọi t_1, t_2 ∈ r, t_1 ≠ t_2 đều tồn tại một thuộc tính A ∈ K sao cho giá trị của thuộc tính t_1 tại A khác giá trị t_2 tại A (t_1[A] ≠ t_2 [A]).

Nói một cách khác, không tồn tại hai bộ mà có giá trị bằng nhau trên mọi thuộc tính của K. Điều kiện này có thể viết (t_1[K] ≠ t_2 [K]). Do vậy mỗi giá trị của K là xác định duy nhất một bộ.

Ví dụ 3.1:

K_1 = {Mã_Sách} là khoá

K_2 = {Mã_ĐộcGiả} là khoá

K_3 = {Tên_Sách} không phải là khoá vì có thể có hai quyển sách có tên giống nhau nhưng nội dung khác nhau (không duy nhất).

3.2. Tính chất

- Nếu K là khoá của quan hệ r(A_1,A_2,..., An) thì K ⊆ K' ⊆ R, K' cũng là khoá của r, nghĩa là bất kì t_1, t_2 ∈ r từ t_1[K] ≠ t_2 [K] luôn có t_1[K'] ≠ t_2 [K'].
- K là khoá tối thiểu của nếu như K là khoá của r và với mọi K' ⊂ K, K' không là khoá của quan hệ r.
- Một quan hệ có thể có nhiều khoá tối thiểu, do đó không cần xác định hết khoá tối thiểu mà chỉ cần xác định 1, 2 khoá tối thiểu làm khoá chính, các khoá khác là dự bị.
- K được gọi là khoá ngoài của quan hệ r nếu như K không phải là khoá chính của quan hệ r nhưng nó lại là khoá chính của quan hệ khác.

Ví dụ 3.2:

Thực thể Sách có khoá chính là Mã_Sách.

Thực thể Độc giả có khoá chính là Mã_Độc_Giả.

Thực thể Mượn có khoá ngoài là Mã_Sách và Mã_Độc_Giả.

Bài 2. Các phép toán đại số quan hệ (3 tiết)

Trong bài này sẽ trình bày chi tiết đại số quan hệ như là cơ sở của một ngôn ngữ bậc cao để thao tác trên các quan hệ. Bao gồm các phép toán tập hợp như: hợp, trừ, giao, tích Đề Các và các phép toán quan hệ như: chọn, chiếu, kết nối, chia.

1. Định nghĩa

a) Định nghĩa 1:

Hai quan hệ r và s gọi là khả hợp nếu chúng được xác định trên cùng tập miền trị.

Ví dụ: quan hệ r xác định trên miền $\{D_1,D_2,...,Dn\}$ có bậc n

quan hệ s xác định trên miền $\{D'_1,D'_2,...,D'_m\}$ có bậc m

nếu n = m và {r} = {s} thì r và s được gọi là khả hợp

b) Định nghĩa 2:

Hai quan hệ r và s khả hợp nếu chúng được xác định trên cùng tập thuộc tính, các thuộc tính cùng tên có cùng miền trị và đối sánh với nhau.

2. Các phép toán đại số

2.1. Phép hợp (Union)

Hợp của hai quan hệ r và s khả hợp, kí hiệu là r ∪ s là tập các bộ thuộc r hoặc s hoặc thuộc cả hai quan hệ.

Biểu diễn hình thức cho phép hợp có dạng như sau:

Ví dụ 2.1:

r (A B C)	s (A B C)	r ∪ s = (A B C)
a_1 b_1 c_1	a_1 b_1 c_1	a_1 b_1 c_1
a_2 b_1 c_2	a_2 b_2 c_2	a_2 b_1 c_2
a_2 b_2 c_1		a_2 b_2 c_2
		a_2 b_2 c_1

2.2. Phép giao (Intersect)

Giao của hai quan hệ r và s khả hợp, kí hiệu là r ∩ s là tập các bộ thuộc cả hai quan hệ r và s. Biểu diễn hình thức cho phép giao có dạng như sau:

Ví dụ 2.2: Với r và s là hai quan hệ trong ví dụ trên, giao của chúng là:

r ∩ s =	(A	B	C)
	a_1	b_1	c_1

2.3. Phép trừ (Minus)

Hiệu của hai quan hệ r và s khả hợp, kí hiệu r - s.

Biểu diễn hình thức phép trừ có dạng như sau:

Ví dụ 2.3: Với r và s là hai quan hệ trong ví dụ trên, phép trừ của chúng là:

r - s = (A B C)

a_2 b_1 c_2

a_2 b_2 c_1

Chú ý: Phép giao của 2 quan hệ r [redacted] có thể biểu diễn quan phép trừ:

r [redacted] = r - (r-s)

2.4. Phép chọn (Selection)

Phép chọn trên một quan hệ thực chất là phép toán lọc ra một tập con các bộ của quan hệ thoả mãn điều kiện cho trước. Điều kiện là biểu thức chọn, là một tổ hợp logic của các toán hạng trong đó mỗi một toán hạng là một phép so sánh giữa hai thuộc tính hoặc giữa một thuộc tính và một giá trị hằng.

[redacted] Các phép toán logic bao gồm: [redacted]và), [redacted]hoặc), [redacted]không hay phủ định).

[redacted] Các phép toán so sánh trong biểu thức là: <, =, >, <=, >= và [redacted]

Hình thức hoá phép chọn được định nghĩa như sau:

[redacted]r) = { t [redacted] | F(t) = đúng} (2.4)

F(t) được hiểu là giá trị của các thuộc tính xuất hiện trong biểu thức F tại bộ t thoả các điều kiện của F.

Ví dụ 2.4: cho quan hệ như sau:

r (A B C D)

a_1 b_1 c_1 d_1

a_1 b_1 c_1 d_2

a_2 b_2 c_2 d_2

a_2 b_2 c_3 d_3

Các phép chọn:

[redacted]$_{=a1}$(r) = s(A B C D)

a_1 b_1 c_1 d_1

[redacted]$_{=a1}$ [redacted]$_{b = d2}$(r) = (A B C D)

a_1 b_1 c_1 d_2

a_1 b_1 c_1 d_2

2.5. Phép chiếu (Projection)

Phép chiếu trên một quan hệ là phép toán loại bỏ đi một số thuộc tính và chỉ giữ lại một số thuộc tính còn lại của quan hệ đó. Giả sử r là một quan hệ n - ngôi (gồm n thuộc tính) ta có:

Gọi X là tập con của thuộc tính R = {A_1,A_2,..., An}. Phép chiếu trên tập X của quan hệ r, kí hiệu là [redacted](r) được định nghĩa như sau:

với t[X] là giá trị của bộ t tại thuộc tính X.

Ví dụ 2.5:

R = {A,B,C,D}; X = {A,B}; Y = {A,C}

r (A B C D) [redacted](r) = p(A B) [redacted](r)=p(A C)

a_1	b_1	c_1	d_1	a_1	b_1	a_1	c_1
a_1	b_1	c_1	d_2	a_2	b_2	a_2	c_2
a_2	b_2	c_2	d_2			a_2	c_3
a_2	b_2	c_3	d_3				

So sánh : s = [redacted]$_{=a1}$(r) ; p = [redacted](r) và r:

- Bậc p = bậc của r , lực lượng của p [redacted]ượng của r
- s không khả hợp với r, bậc của l [redacted]ậc của r; lực lượng của l [redacted]ực lượng của r.

2.6. Tích Đề Các (Times)

Gọi r là quan hệ xác định trên tập thuộc tính (A_1,A_2,..., An) và s là quan hệ xác định trên tập thuộc tính (B_1,B_2,..., Bm). Tích Đề Các r x s của r và s là tập (n+m) bộ với n thành phần đầu có dạng một bộ thuộc r và m thành phần sau đó có dạng một bộ thuộc s.

Biểu diễn hình thức của quan hệ r(U) và quan hệ s(V) với U = (A_1,A_2,..., An) và V = (B_1,B_2,..., Bm) có dạng:

với u = (a_1,a_2,..., an) và s = (b_1,b_2,..., bm)

Ví dụ 2.6: r (A B) s (C D) r x s = p(A B C D)

a_1	1	1	d_1	a_1	1	1	d_1		
a_2	2	2	d_2	a_1	1	2	d_2		
				a_2	2	1	d_1		
				a_2	2	2	d_2		

2.7. Phép kết nối (Join)

Biểu thức kết nối được định nghĩa là tổ hợp logic các toán hạng, trong đó mỗi toán hạng là một phép so sánh giữa một thuộc tính của quan hệ r

Phép toán kết nối của hai qua n hệ r(U) và s(V) được biểu diễn như sau:

Trong đó F là điều kiện kết nối, F = (A B), với à một trong các phép so sánh {<, =, >, <=, >= , . Dĩ nhiên, ở đây cần giả thiết rằng mỗi giá trị của thuộc tính A (A) đều có thể so sánh được (qua phép với mỗi giá trị của thuộc tính B (B)

Ví dụ 2.7: phép kết nối giữa quan hệ r và s với điều kiện kết nối F = (B C) như sau:

r (A	B)	s (C	D)	r ⊳⊲ s = p'(A	B	C	D)
a_1	1	1	d_1	B C	a_1	1	1	d_1
a_2	2	2	d_2		a_2	2	1	d_1
					a_2	2	2	d_2

Nhận xét:

Tập kết quả của phép kết nối p' là tập con của tập kết quả phép tích Đề Các p:

p'

<=> p' = C (p)

<=> r ⊳⊲ s = C (r x s)

B C

Do đó có thể định nghĩa phép chiếu thông qua phép chọn và phép tích Đề Các như sau:

$$r \rhd\lhd s = \sigma_{B\ C}(r \times s)$$

Nếu điều kiện kết nối F = (A B) với A và B trong trường hợp phép so sánh à "=" thì phép kết nối được gọi là "*kết nối bằng*".

Ví dụ: với quan hệ r và s trong ví dụ trên ta có:

r (A	B)	s (C	D)	r ⊳⊲ s = p''(A	B	C	D)
a_1	1	1	d_1	B = C	a_1	1	1	d_1
a_2	2	2	d_2		a_2	2	2	d_2

Trường hợp kết nối bằng tại thuộc tính cùng tên của hai quan hệ và một trong hai thuộc tính đó được loại bỏ qua phép chiếu, thì phép kết nối được gọi là *"kết nối tự nhiên"*

và sử dụng kí hiệu "*". Khi đó phép kết nối tự nhiên của hai quan hệ r(AB) và s(CD) biểu diễn như sau:

r(AB)*s(BD) = {t[ABD] | t[AB] [redacted] và t[BD] [redacted]}

Ví dụ:

r (A	B)	s' (B	D)	r ⊳⊲ s = p'''(A	B	B	D)
a_1	1	1	d_1	a_1	1	1	d_1
a_2	2	2	d_2	a_2	2	2	d_2

($B = B$)

Hay kí hiệu r * s' = p'''(A B D)

a_1	1	d_1
a_2	2	d_2

Nhận xét: Phép kết nối tự nhiên có thể được biểu diễn thông qua phép chiếu và kết nối thông thường. Xét ví dụ dưới đây:

Cho hai quan hệ r (A B C) và s (B C D)

Kết nối tự nhiên r * s = p (A B C D)

Hay có thể biểu diễn r * s = [redacted]$_{CD}$ (r ⊳⊲ s)

($B = B, C=C$)

Có nghĩa là tập kết quả của phép kết nối tự nhiên thu được sau khi thực hiện phép chiếu trên các thuộc tính khác nhau đối với tập kết quả của phép kết nối tại các thuộc tính cùng tên.

2.8. Phép chia (Divide)

Gọi r là quan hệ n ngôi và s là quan hệ m ngôi (với n > m, và s [redacted]). Phép chia r[redacted] là tập của tất cả (n - m) bộ t sao cho với mọi bộ u [redacted] thì bộ t [redacted] [redacted].

Biểu diễn hình thức như sau:

r[redacted] = { t | [redacted] [redacted] => t [redacted] [redacted]} (2.8)

Ví dụ 2.8:

r(A	B	C	D)	S (C	D)	r[redacted] = (A	B)
a	b	c	d	c	d	a	b
a	b	e	f	e	f	e	d
a	c	e	f				

e d c d

e d e f

3. Các ví dụ về tìm kiếm bằng đại số quan hệ

Ví dụ có ba quan hệ:

S (S#, SNAME, STATUS, CITY) : các hãng cung ứng,

P (P#, PNAME, COLOR, WEIGHT, CITY) : các mặt hàng

SP(S#, P#, QTY) : các mặt hàng đã cung cấp

- Tìm số hiệu của những hãng đã cung cấp mặt hàng P2

  [redacted]([redacted]# = 'P2'(SP))

- Tìm số hiệu của những hãng đã cung ứng ít nhất là một mặt hàng nàu đỏ

  [redacted]([redacted]OLOR = 'RED'(P*SP))

hoặc [redacted](([redacted]OLOR = 'RED'(P))*SP)

4. Kết luận chung và bài tập:

4.1. Kết luận

Nói chung các phép tính của đại số quan hệ là khá đơn giản, mạnh và là một đại số có tính đầy đủ, không cần thủ tục. Tuy nhiên đây là ngôn ngữ khá gần ngôn ngữ lập trình thủ tục, dễ dàng mở rộng và chủ yếu là làm cơ sở cho việc thiết lập các ngôn ngữ con dữ liệu bậc cao hơn. Trong bài sau sẽ trình bày ngôn ngữ con dữ liệu SQL, đây ngôn ngữ con dữ liệu quan hệ được xác nhận là rất mạnh, phổ dụng và lại dễ sử dụng.

4.2. Tài liệu tham khảo

- Lê Tiến Vương, *Nhập môn Cơ sở dữ liệu quan hệ*, NXB Giáo dục, Chương1
- P. O' Neil, *Database - Principles, Programming, Performance*, Chương 3

4.3. Bài tập

Bài 1 (trang 175), Nhập môn Cơ sở dữ liệu quan hệ, Lê Tiến Vương, NXB Thống Kê.

CH□ƠNG 4. NGÔN NGỮ VẤN TIN CÓ CẤU TRÚC SQL (STRUCTURE QUERY LANGUAGE)

Hệ QTCSDL cung cấp cho người sử dụng công cụ để thực hiện các yêu cầu thông quan ngôn ngữ giao diện. Trong chương này sẽ trình bày ngôn ngữ dữ liệu SQL (Structure Query Language). Đây là ngôn ngữ con dữ liệu quan hệ được xác nhận là rất mạnh, thông dụng, và dễ sử dụng.

Bài 1. Ngôn ngữ định nghĩa dữ liệu (2 tiết)

1. Giới thiệu về ngôn ngữ SQL

Ngôn ngữ SQL được phát triển từ ngôn ngữ SEQUEL-2, thử nghiệm và cài đặt tại Trung tâm nghiên cứu của hãng IBM ở San Jose, California cho hệ thống QTCSDL lớn điển hình là System -R. Trong System -R, SQL vừa đóng vai trò là một ngôn ngữ có thể thao tác độc lập của người sử dụng đầu cuối, đồng thời lại có khả năng là một ngôn ngữ con được nhúng trong ngôn ngữ chủ PL/1.

Khác với ngôn ngữ đại số quan hệ, là ngôn ngữ dựa trên các phép toán của đại số quan hệ. SQL là một ngôn ngữ phi thủ tục, chuẩn mực và điển hình. DO vậy hiện nay rất nhiều sản phẩm phần mềm thương mại đều được cài đặt SQL như Access, SQL Server, DB2, Oracle,...

Phép toán cơ bản trong SQL là phép ánh xạ được miêu tả như một khối SELECT - FROM - WHERE. Trong phần sau, các mệnh đề của ngôn ngữ SQL sẽ được trình bầy chi tiết bằng các ví dụ.

Các thuật ngữ trong CSDL quan hệ như quan hệ, thuộc tính, bộ,... sẽ được thay bằng các thuật ngữ như bảng (table), cột (colum), bản ghi (record) hoặc hàng (row).

2. Các mệnh đề của ngôn ngữ SQL

2.1. Tạo bảng

Mệnh đề tạo bảng tạo một bảng quan hệ mới thông quan tên bảng, xác định các thuộc tính và các ràng buộc logic của bảng.

Mệnh đề tạo bảng có dạng tổng quát như sau:

```
CREATE TABLE <tên bảng> (<tên cột> <kiểu dữ liệu> [NOT NULL],...)
```

Trong đó:

- tên bảng: là xâu kí tự bất kì không có ký hiệu trống và không trùng với các từ khoá.
- tên cột: là xâu kí tự bất kì không chứa ký hiệu trống,trong một bảng tên cột là duy nhất. Thứ tự của cột trong bảng là không quan trọng.
- kiểu dữ liệu: trong mệnh đề tạo bảng dùng một số kiểu dữ liệu sau:
  - integer: kiểu số nguyên, từ - 2.147.483.648 đến - 2.147.483.647
  - smallinteger: số nguyên nhỏ từ -32.768 đến 32.767
  - decimal (n,p): số thập phân với độ dài tối đa là n kể cả p chữ số phần thập phân (không tính dấu chấm thập phân).
  - char(n): xâu kí tự có độ dài cố định n. Một số HQTCSDL cho phép kích thước tối đa của char là 2000 bytes. Nếu mỗi kí tự tương ứng một byte (mã ASCII như đối với hệ QTCSDL Access) thì số kí tự tối đa là 2000. Trong trường hợp sử dụng bộ mã UCS2 (như đối với hệ QTCSDL SQL Server) thì số kí tự biểu diễn tối đã là 1000 kí tự (sử dụng 2 byte để biểu diễn 1 kí tự).
  - varchar(n): xâu kí tự có độ dài biến đổi, độ dài xâu có thể từ 0 đến n và được xác định tại thời điểm đưa giữ liệu vào lưu trữ. . Một số HQTCSDL cho phép kích thước tối đa của char là 4000 bytes.
  - date: dữ liệu dạng ngày tháng, định dạng ngầm định: DD - MON - YY.
  - Ngoài ra mỗi hệ QTCSDL cũng đưa ra các kiểu dữ liệu riêng.
- NULL là giá trị ngầm định khi không biết chính xác giá trị. Do đó cột đóng vai trò khoá chính không được nhận giá trị NULL, các cột khác có thể tuỳ chọn.

Ví dụ 2.1: Cho CSDL gồm 3 bảng S (Supplier),P (Part) và SP như sau:

S (S#, SNAME, STATUS, CITY) : các hãng cung ứng, với S# là khoá chính

P (P#, PNAME, COLOR, WEIGHT, CITY) : các mặt hàng, P# là khoá chính

SP(S#, P#, QTY,SDATE) : các mặt hàng đã cung cấp.

Tạo bảng S:

Hình 9 - Cấu trúc lệnh tạo bảng nhà cung cấp

Tạo bảng P:

Tạo bảng SP:

Nhận xét:

Trong mệnh đề tạo bảng có thể sử dụng thêm các yếu tố ràng buộc để hạn chế các giá trị cho một hay nhiều cột trong bảng, như: ràng buộc khoá chính, khoá ngoài, ràng buộc toàn vẹn tham chiếu, ràng buộc miền giá trị tương ứng như sau:

- NULL: cột có thể không chứa giá trị.
- NOT NULL: cột phải chứa một giá trị nào đó
- PRIMARY KEY: ràng buộc khoá chính
- REFERENCE: ràng buộc khoá ngoài
- ON DELETE CASCADE: ràng buộc toàn vẹn tham chiếu

2.2. Xoá bảng

Mệnh đề xoá bảng xoá một bảng quan hệ (cả cáu trúc và nội dung của bảng) khỏi CSDL. Bảng này chỉ có thể được tạo lại bằng cách sử dụng mệnh đề CREATE TABLE.

Mệnh đề xoá bảng có dạng tổng quát như sau:

Ví dụ 2.2: Xoá bảng SP trong ví dụ trên: DROP TABLE SP

2.3. Thêm dữ liệu

Đây là mệnh đề thêm một bản ghi vào một bảng và thường được sử dụng để tổ chức vào dữ liệu. Dạng tổng quát như sau:

Có 3 cách biểu diễn mệnh đề thêm dữ liệu:

Cách 1:
```
INSERT INTO S(S#, SNAME, STATUS, CITY)
VALUES (1,'VINH',30,'HA NOI' )
```

Cách 2: Nếu vị trí của các cột trong bảng là cố định mệnh đề có thể viết:
```
INSERT INTO S
VALUES (1,'VINH',30,'HA NOI' )
```

Cách 3: Nếu người vào dữ liệu quên vị trí của các cột, khi đó có thể biểu diễn như sau:
```
INSERT INTO S(S#, STATUS, CITY, SNAME)
VALUES (1, 30,'HA NOI', 'VINH')
```

Bài 2. Ngôn ngữ thao tác dữ liệu (3 tiết)

1. Khối SELECT

Cấu trúc đơn giản nhất trong SQL là khối SELECT đươc miêu tả về cú pháp như một khối **select - from - where.**

Một cách tổng quát khối select bao gồm 3 mệnh đề chính:

- Select: Xác định nội dung của các cột cần đưa ra kết quả
- From: Xác định các bảng cần lấy thông tin ra.
- Where: Xác định các bản ghi thoả mãn yêu cầu chọn lọc để đưa ra kết quả.

Ngoài ra, để mở rộng khả năng của ngôn ngữ, khối SELECT còn được bổ sung thêm các mệnh đề *group by, having, order by,* các hàm mẫu,... Trong các phần sau sẽ trình bày chi tiết từng mệnh đề.

Dạng tổng quát của khối select được biểu diễn như sau:

Trong đó mệnh đề WHERE biểu diễn dưới một số dạng sau:

- WHERE [NOT] <biểu thức> phép_so_sánh <biểu thức>
- WHERE [NOT] <tên cột> [NOT] LIKE <xâu kí tự>
- WHERE [NOT] <biểu thức> [NOT] BETWEEN <biểu thức> AND <biểu thức>
- WHERE [NOT] <biểu thức > [NOT] IN ({danh sách / mệnh đề con})
- WHERE [NOT] <tên cột> phép_kết_nối <tên cột>
- WHERE [NOT] <biểu thức logic>
- WHERE [NOT] <biểu thức> {AND | OR} [NOT] <biểu thức>

2. Các mệnh đề tìm kiếm

2.1. Tìm kiếm theo câu hỏi đơn giản

a) Tìm kiếm không điều kiện

Trước hết, để đơn giản ta làm quen với các câu hỏi chỉ liên quan tới một bảng. Trong mệnh đề select có danh sách cột, danh sách này xác định tên các cột cần có trong bảng kết quả. Nếu sử dụng giá trị '*' có nghĩa là chọn toàn bộ các cột của bảng.

Ví dụ 1: Cho biết mã số các nhà cung cấp ứng với một mặt hàng nào đó

SELECT S# SP (S#, P#, QTY, SDATE)

FROM SP

Kết quả

(trong đại số quan hệ: (SP))

Chú ý: Sau khi thực hiện một lệnh SQL, để bảng kết quả đúng là một quan hệ (có nghĩa là không có bộ trùng nhau), trong mệnh đề select cần thêm từ khoá DISTINCT.

Ví dụ 2: như trong ví dụ trên

```
SELECT DISTINCT S#
FROM SP
```

Tập kết quả chỉ gồm S1 và S2

b) Tìm kiếm với điều kiện đơn giản

Tìm mã số những nhà cung cấp đã cung cấp mặt hàng P2:

```
SELECT DISTINCT S#
FROM SP
WHERE P# = 'P2'
```

Trong SQL các phép so sánh được sử dụng bao gồm >, <, >=, <=, = và <>. Các phép trên dùng cho mọi loại dữ liệu.

c) Tìm kiếm có xử lý xâu kí tự

Xử lý xâu kí tự gần đúng được dùng trong trường hợp người sử dụng không nhớ rõ tên người hoặc địa danh,... ví du là Hoa hay Hoan khi đó có thể viết:

```
SELECT *
FROM S
WHERE SNAME LIKE 'HOA%'
```

Trong SQL sử dụng kí hiệu '%' là thay thế cho một xâu con, dấu phân cách '_' để thay thế cho một kí tự.

Ví dụ 3:

- A%B: xâu ký tự bắt đầu bằng chữ A và kết thúc bằng chữ B
- %A: xâu ký tự bất kì có kết thúc là A
- A_B: xâu gồm ba ký tự có ký tự thứ hai là bất kì.

d) Xử lý ngày tháng

Ngoài các phép tính thông thường, SQL còn có thể xử lý dữ liệu dạng ngày tháng.

Ví dụ 4: Tìm những mặt hàng bán trước ngày 11-10-02 là 10 ngày

```
SELECT P#
FROM SP
WHERE '11-10-02' - sdate = 10
```

e) Tìm kiếm nhờ sử dụng IN và BETWEEN

- Tìm những mặt hàng cung cấp có số lượng từ 100 đến 200

```
SELECT P#
FROM SP
WHERE QTY BETWEEN 100 AND 200
```

- Tìm mã số những nhà cung cấp đã cung cấp ít nhất một trong các mặt hàng P1, P2,P3

```
SELECT S#
FROM SP
WHERE P# IN ('P1', 'P2', 'P3')
```

2.2. Các hàm thư viện

Cũng giống như các ngôn ngữ CSDL khác, trong SQL có các hàm mẫu gồm *count, max, min, sum, avg*. Riêng hàm count khi có đối số là '*' có nghĩa là đếm số bản ghi thoả mãn yêu cầu tìm kiếm mà không cần quan tâm tới bất kì một cột nào.

Ví dụ 2.1: Cho biết số lần mặt hàng P2 đã được cung cấp

```
SELECT COUNT(*)
FROM SP
WHERE P# = 'P2'
```

Tìm số mặt hàng P1 bán một lần nhiều nhất

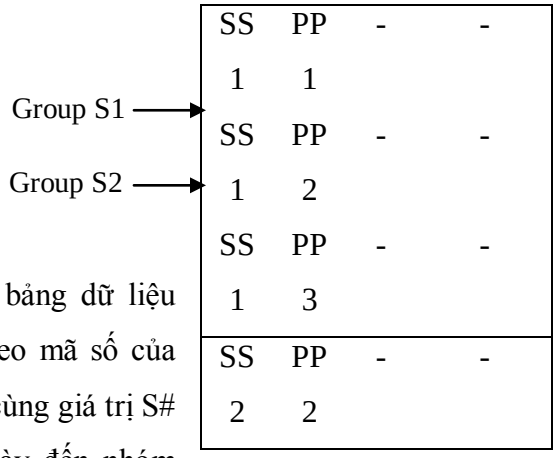
```
SELECT MAX (QTY)
FROM SP
WHERE P# = 'P1'
```

2.3. Tìm kiếm nhờ mệnh đề GROUP BY

Tìm mã số những mặt hàng mà mỗi nhà cung cấp đã cung cấp cho khách hàng

```
SELECT S#,P#
FROM SP                    SP (S#,   P#,  QTY, SDATE)
```

GROUP BY S#



Trong mệnh đề này bảng dữ liệu SP được lấy ra, sau đó phân thành nhóm theo mã số của người cung cấp (S#). Có nghĩa là các bộ có cùng giá trị S# được sắp xếp liênt tiếp nhau, hết nhóm này đến nhóm khác.

a) Tìm kiếm có sử dụng mệnh đề HAVING

Mệnh đề Having thường được sử dụng cùng mệnh đề Group by. Sau Having là biểu thức điều kiện. Biểu thưc điều kiện này không tác động vào toàn bảng được chỉ ra ở mệnh đề From mà chỉ tác động lần lượt từng nhóm các bản ghi đã chỉ ra tại mệnh đề Group by.

Ví dụ 2.3: Tìm mã số những nhà cung cấp đã cung cấp ít nhất hơn hai mặt hàng.

```
SELECT S#
FROM SP
GROUP BY S#
HAVING COUNT (DISTINCT P#) > 2
```

b) Tìm kiếm có sắp xếp

Tìm tên các mặt hàng mầu đỏ và sắp xếp theo thứ tự giảm dần của mã số mặt hàng.

```
SELECT PNAME, P#
FROM P
WHERE COLOUR = 'Đỏ'
ORDER BY P# ASC
```

Sau mệnh đề Order by là tên cột cần sắp xếp rồi đến chiều sắp xếp tăng hoặc giảm (ASC hoặc DESC). Có thể sắp xếp nhiều cột và nếu không chỉ ra chiều sắp xếp thì hệ thống ngầm định là chiều tăng (ASC).

2.4. Tìm kiếm với câu hỏi phức tạp

Trong phần này trình bày việc tìm kiếm với nhiều bảng qua việc sử dụng ánh xạ lồng nhau hoặc qua phép kết nối.

a) Phép kết nối

Trong phép kết nối, các cột tham gia kết nối phải có miền trị là sánh (so sánh) được với nhau (ví dụ: cùng là kiểu integer, char, hay giữa kiểu char và varchar,...). Tên cột của các bảng khác nhau có thể được viết tường minh qua tên bảng.

Trong ví dụ quan hệ người cung cấp - mặt hàng, bảng S chứa cột số hiệu mặt hàng S#, bảng SP cũng chứa số hiệu mặt hàng S#. Để phân biệt, có thể viết tường minh tên cột S# qua tên bảng là S.S# và SP.S#.

Ví dụ 2.4: Với mỗi mặt hàng đã được cung cấp, cho biết mã số của mặt hàng và địa chỉ của hãng đã cung cấp mặt hàng đó.

Ta nhận thấy, trong bảng SP (bảng mặt hàng đã được cung cấp) không chứa địa chỉ của hãng đã cung cấp. Do đó cần kết nối với bảng S để thu được địa chỉ của hãng cung cấp thông qua phép kết nối.

```
SELECT P#, CITY
FROM SP,S
WHERE SP.S# = S.S#
```

PK

S#	SNAME	STATUS	CITY
S1	OMOMO	Goodod	HNHN
S2	DASOSO	Goodod	HCMCM
S3	-	-	-
S4	-	-	-

1 ——— kết nối (quan hệ 1 - n) ——— n

FK

S#	PP#	QTYTY	SDATE
S1	PP1	10	-
S1	PP2	20	-
S2	PP2	50	-
S2	PP3	-	-

Hình 10 – Ví dụ bảng quan hệ S- SP

Chú ý: Trong các phép tìm kiếm có nhiều hơn một bảng, nếu tên các cột là không duy nhất thì bắt buộc phải viết tên cột dạng tường minh.

b) ánh xạ lồng

Tìm tên những hãng đã cung cấp mặt hàng P2.

Phép lồng nhau có thể được lồng nhiều mức hoặc sử dụng ánh xạ lồng với sự dẫn trỏ giữa các khối mỗi khi hướng tới một bảng khác nhau.

Ví *dụ:* Tìm số hiệu và tên các hãng không cung ứng mặt hàng P1.

c) Tìm kiếm có sử dụng lượng từ ANY và ALL

Tìm tên những mặt hàng có mã số mặt hàng là mặt hàng nào đó mà hãng S1 đã bán.

Tìm mã số những hãng cung cấp số lượng một lần một mặt hàng nào đó lớn hơn hoặc bằng số lượng mỗi lần cung cấp của các hãng

Mệnh đề trên hoàn toàn tương đương với:

d) Tìm kiếm có chứa phép tính tập hợp

Tìm mã số những hãng hiện thời chưa cung cấp một mặt hàng nào cả

Tìm tên các hãng cung cấp tất cả các mặt hàng.

3. Các mệnh đề cập nhật dữ liệu

Trong các phần trước đã trình bầy các mệnh đề để tạo bảng, vào dữ liệu. Phần này sẽ trình bầy chi tiết các mệnh đề cập nhật dữ liệu trong SQL

3.1. Thêm một bộ dữ liệu

Mệnh đề thêm một bộ dữ liệu có dạng tổng quát như sau:

```
INSERT INTO <tên bảng[(danh sách tên cột)]>
VALUES (bộ giá trị)
[câu hỏi con]
```

Ví *dụ 3.1:* Có thể bổ sung một tập các bản ghi là kết quả xử lý của một câu hỏi nào đó, chẳng hạn:

Nếu như bảng W và bảng S có cùng lược đồ.

W#	WNAME	STATUS	CTY
w1	Hai Ha	Good	HN
W2	Halida	Normal	HN
W3	Kinh Đô	Good	HCM
W4	-	-	-

SS#	WNAMEME	STATUSUS	CTYTY
SS1	OMOMO	Goodod	HNHN
SS2	DASOSO	Goodod	HCMCM
WW1	HaiaiHaHa	Goodod	HNHN
WW3	KinhnhĐô§«	Goodod	HCMCM

Hình 11 – Ví dụ thêm bộ dữ liệu

3.2. Xoá bản ghi

Mệnh đề xoá bản ghi có thể được thực hiện cho một hoặc nhiều bản ghi thoả mãn một điều kiện nào đó. Dạng tổng quát là:

```
DELETE
[FROM] <tên bảng>
[WHERE <biểu thức điều kiện>]
```

Ví dụ 3.2: Loại bỏ hãng S1 khỏi bảng S

```
DELETE
FROM S
WHERE S# = 'S1'
```

Loại bỏ các mặt hàng đã được cung cấp sau ngày 20 - 5 -2000

```
DELETE FROM SP WHERE SDATE > '20 - 5 -2000'
```

Loại bỏ những hãng chưa cung cấp mặt hàng nào cả

```
DELETE FROM S WHERE S# NOT IN
                    ( SELECT S#
                      FROM SP)
```

3.3. Sửa đổi dữ liệu

Mệnh đề sửa đổi các giá trị của các bản ghi trong bảng của CSDL theo một điều kiện nào đó có dạng tổng quát như sau:

Ví dụ 3.3: Đổi mầu các mặt hàng P2 thành mầu vàng.

```
UPDATE P SET COLOUR = 'Vàng' WHERE P# = 'P2'
```

```
ALTER TABLE <tên bảng>
ADD <tên cột mới> <kiểu dữ liệu>
```

3.4. Thêm cột mới

Ví dụ 3.4: Thêm cột PRICE (giá) cho bảng SP với kiểu số liệu dạng số thập phân.

```
ALTER TABLE SP
ADD PRICE DECIMAL(8.2)
```

4. Kết luận và bài tập

4.1. Kết luận

Chương này trình bầy về ngôn ngữ dữ liệu SQL. Đây là ngôn ngữ rất mạnh, phổ dụng và dễ sử dụng. Hiện nay, các hệ QTCSDL thông dụng như: Access, SQL Server, Oracle, DB2,... đều hỗ trợ tốt ngôn ngữ này. Do đó để có thể thao tác tốt trên các hệ QTCSDL này, cần hiểu và nắm vững ngôn ngữ SQL.

4.2. Bài tập

Bài 2, 3, 4 (trang 175, 176),

Sách Nhập môn Cơ sở dữ liệu quan hệ, Lê Tiến Vương, NXB Thống Kê.

Tổng quan về Cơ sở dữ liệu quan hệ

Lời mở đầu

Thuật ngữ Cơ sở Dữ liệu (Database) không mấy xa lạ với những người làm tin học. Đây là một trong những lĩnh vực được tập trung nghiên cứu và phát triển của Công nghệ thông tin, nhằm giải quyết các bài toán quản lí, tìm kiếm thông tin trên các hệ thống lớn, phức tạp, nhiều người sử dụng.

Từ những năm 70, mô hình dữ liêu quan hệ do Codd đưa ra với cấu trúc hoàn chỉnh đã tạo cơ sở toán học cho các vấn đề nghiên cứu dữ liệu. Với cấu trúc đơn giản và khả năng hình thức hoá phong phú, CSDL quan hệ dễ dàng mô phỏng các hệ thống thông tin đa dạng trong thực tế. Lưu trữ thông tin tiết kiệm, có tính độc lập dữ liệu cao, dễ sửa đổi, bổ sung cũng như khai thác dữ liệu là những ưu điểm nổi bật của CSDL quan hệ .

Sau đây chúng ta sẽ đề cập tới nhưng khái niệm cơ bản của CSDL quan hệ .

I - Khái quát về CSDL & CSDL quan hệ

I.1. Khái niệm CSDL

5. *Định nghĩa*

Một cơ sở dữ liệu là một tập hợp dữ liệu về một xí nghiệp được lưu giữ trên máy tính, được người sử dụng, có cách quản lí bằng một mô hình.

VD: Quản lí thi tuyển sinh thì CSDL bao gồm:

+ *thí sinh* **(tên, ngày sinh, địa chỉ, số báo danh ...).**
+ *phách* **(số báo danh, số phách).**
+ *điểm* **(số phách, điểm).**

6. *Các tiêu chuẩn của một CSDL*

Một CSDL cần :

+ phản ánh tốt xí nghiệp cần quản lí.

+ không dư thừa thông tin: mỗi thông tin chỉ nên có mặt một lần trong hệ thống thông tin để tiết kiệm lưu trữ, đảm bảo truy cập duy nhất.

+ độc lập giữa CSDL và chương trình: sự sửa đổi chương trình không làm kéo theo việc sửa đổi CSDL.

+ tính an toàn: không bị hỏng khi có nhiều người sử dụng hoặc có các sự cố.

+ hiệu suất sử dụng tốt: dù nhiều người sử dụng một lúc, CSDL vẫn đảm bảo hiệu suất như chỉ có một người sử dụng.

7. *Hệ QTCSDL*

Là tập hợp có thứ tự các phần mền cho phép mô tả lưu giữ thao tác các dữ liệu trên một CSDL, đảm bảo tính an toàn, bí mật trong môi trường có nhiều người sử dụng.

8. *Hệ thống thông tin*

Là tập hợp các thông tin được lưu giữ, và một tập hợp các xử lí cho phép xây dựng lại một hình ảnh trung thành về một xí nghiệp.

9. *Kiến trúc một CSDL*

Gồm 3 thành phần cơ bản

+ Thực thể: là đối tượng có trong thực tế mà chúng ta cần mô tả các đặc trưng của nó, đối tượng có thể là cụ thể hoặc trừu tượng .

+ Thuộc tính: là các dữ liệu thể hiện các đặc trưng của thực thể.

+ Ràng buộc: là các mối quan hệ Logic của thực thể.

I.2.Các mô hình CSDL

10. *Mô hình phân cấp*

Mô hình dữ liệu là một cây, trong đó các nút biểu diễn các tập thực thể, giữa các nút con và nút cha được liên hệ theo mối quan hệ xác định chủ-thành viên (1-n).

11. *Mô hình mạng*

Mô hình được biểu diễn là một đồ thị có hướng .

12. *Mô hình quan hệ*

Mô hình dựa trên cơ sở khái niện lý thuyết tập hợp của các quan hệ .

I.3.Các khái niệm cơ bản vê CSDL quan hệ

13. *Thuộc tính*

+ là một lô thông tin nhỏ nhất được sử dụng một cách tự do và có ý nghĩa, độc lập với các lô khác .

+ trong một mô hình , thuộc tính là một định vị cơ sở thông tin, một thuộc tính được định nghĩa bằng tên và miền giá trị của nó .

14. *Quan hệ*

+Quan hệ được định nghĩa là tập con của tích Đề các D1*D2*...*Dn trong đó Di là miền giá trị của thuộc tính i.

Ta có thể xem quan hệ như một bảng gồm nhiều cột chứa tên các thuộc tính, mỗi cột chứa miền gía trị của thuộc tính, mỗi cột chứa miền giá trị của thuộc tính đó.

+ Mỗi hàng trong bảng đó là một bộ giá trị của quan hệ, là một hình ảnh của thực thể.

15. *Phụ thuộc hàm*

Xét tập quan hệ Ri và hai tập thuộc tính G1,G2 luôn có mặt cùng nhau trong các Ri. Ta nói có sự phụ thuộc của hàm giữa G1,G2 nếu với mọi giá trị của G1 chỉ có thể kết hợp với một và chỉ một giá trị của G2 tại một thời điểm cho trước. Tính chất đúng trong mọi Ri mà ở đó có mặt G1,G2 .

Kí hiệu G1 --> G2.

16. *Khoá*

Là một hoặc một tập các thuộc tính là nguồn của một phụ thuộc hàm có đích lần lượt là các thuộc tính khác của quan hệ.

VD: *+ Sinh viên* **(Số thẻ Sv, Họ tên, Ngày sinh, Quê quán)**

Số thẻ Sv --> Họ tên
Số thẻ Sv --> Ngày sinh
Số thẻ Sv --> Quê quán

Như vậy* Số thẻ SV *là khóa của quan hệ *Sinh viên*

+ Giảng dạy **(Số phòng học,Thời gian, Tên giảng viên, Tên môn học)**

Số phòng học, Thời gian --> Tên giảng viên, Tên môn học

Như vậy (*Số phòng học, Thời gian*) là khoá của quan hệ *Giảng dạy*

I.4.Các phép toán trên các quan hệ

17. Phép *chiếu*

Xét các tập các thuộc tính C và R là tập các quan hệ định nghĩa trên C. Phép chiếu của R trên tập G thuộc C là sự thu hẹp của R đến các phần tử của G. Kí hiệu G (R) .

18. *Phép nối*

Nối hai quan hệ có chứa các thuộc tính hoặc tập các thuộc tính như nhau, quan hệ thu được gồm có các hàng ở hai quan hệ ban đầu đặt nối nhau bằng các thuộc tính giống nhau. Phép nối hai quan hệ S và T kí hiệu là S |><| T

19. *Phép tách*

Môt quan hệ R được gọi là phân rã thành hai quan hệ S, T nếu thoả mãn :

+ S, T là các phép chiếu của R
+ R = S |><| T

Chú ý rằng không phải bất kì quan hệ nào cũng tách được

20. *Phép chọn* ***: lấy ra các hàng có các thuộc tính thoả mãn một số tiêu chuẩn cho trước.***

Ngoài ra còn có *phép hợp, phép giao, phép lấy hiệu* ***các quan hệ***

II-Thiết kế hệ thống CSDL quan hệ

II.1.Phân tích tư liệu của XN

21. *Xác định danh sách các thuộc tính*

Từ việc khảo sát thực tế, từ các tin tức trao đổi với người dùng, phân tích các bài toán nghiệp vụ, ta được một danh sách các thuộc tính phục vụ cho xây dựng CSDL. Mỗi một thuộc tính bao gồm :

+ tên thuộc tính
+ kiểu thuộc tính (là tính toán hay không tính toán)
+ miền giá trị

+ quy tắc tính toán (nếu thuộc tính là tính toán)
+ ds các tham số

VD : ***Xét quan hệ :***
Bảng lương **(Họ tên, Đơn vị, Hệ số, Lương, Phụ cấp, Tổng lĩnh)**
Ta thấy ở đây **Lương, Tổng lĩnh** ***là các thuộc tính tính toán với các qui tắc***
Lương = Hệ số **** 14400***
Tổng lĩnh = Lương + Phụ cấp

22.

23. *Tìm phụ thuộc hàm giữa các thuộc tính*

Sau khi có được một danh sách các thuộc tính từ việc thu thập thông tin và khảo sát thực tế, việc cần làm tiếp theo là xác định các phụ thuộc hàm giữa các thuộc tính. Chúng ta xét một ví dụ sau :

Quản lí thư viện **(Số thẻ, Số sách, Tên sách, Loại sách, Ngày mượn, Tên độc giả, Địa chỉ độc giả)**
Ta thấy có các phụ thuộc hàm sau :
Số thẻ mượn, Số sách --> Ngày mượn
Số sách --> Tên sách, Loại sách
Số thẻ mượn -->Tên độc giả, Địa chỉ độc giả

II.2.Chuẩn hoá các quan hệ

Coi danh sách các thuộc tính thu được sau các bước trên cùng các phụ thuộc hàm giữa chúng là một quan hệ, chúng ta thực hiện việc chuẩn hoá quan hệ này.
Mục đích ***của quá trình này là giảm bớt sự dư thừa thông tin, bảo đảm tính duy nhất của mỗi thông tin, do đó tiện lợi cho việc truy nhập và cập nhật cho CSDL.Quá trình chuẩn hoá có thể được tiến hành qua nhiều bước.***
Ta xét ví dụ sau :
Xe máy **(Số xe, Số máy, Loại xe, Ngày đăng kí, Tên chủ xe, Số điên thoại, Địa chỉ)**
+ Nếu một người có nhiều xe thì lặp lại (**Số điện thoại, Địa chỉ** ***) của chủ xe, như vậy*** *dư thừa thông tin****.***
+ Nếu một người chủ xe thay đổi địa chỉ thì phải sửa nhiều lần, như vậy *cập nhật rời rạc*
Như vậy biên pháp để khắc phục là tách ra quan hệ
*Chủ xe***(Tên, Số điện thoại, Địa chỉ)**

24.

25. *Đưa quan hệ về dạng chuẩn 1 NF*

Quan hệ chưa ở dạng chuẩn 1NF là quan hệ còn chứa các nhóm lặp lại. Ta đưa về dạng 1NF bằng cách như sau:
+ bỏ nhóm lặp lại ra khỏi quan hệ, chuyển nhóm đó thành một quan hệ mới
+ cộng thêm vào khoá của nó khoá của quan hệ ban đầu để tạo ra khoá phức hợp

VD: ***xét quan hệ :***

R (*Số hoá đơn, Ngày bán, Số khách hàng, Tên khách hàng, Số sản phẩm, Tên sản phẩm, Lượng yêu cầu*)

Nhóm (*Số sản phẩm, Tên sản phẩm, Lượng yêu cầu *) là nhóm lặp lại, ta có thể tách R thành R1 và R2 như sau :

+ R1 (*Số hoá đơn, Ngày bán, Số khách hàng, Tên khách hàng, Số sản phẩm*)

+ R2 (*Số hoá đơn, Số sản phẩm, Tên sản phẩm, Lương yêu cầu*)

26.

27. *Đưa quan hệ về dạng chuẩn 2NF*

Quan hệ đã ở dạng 1NF nhưng chưa ở dạng 2NF là có tồn tại phụ thuộc hàm có nguồn là tập con của khoá. Ta đưa về dạng 2NF bằng cách như sau:

+ nhóm vào một quan hệ các thuộc tính phụ thuộc hoàn toàn vào khoá và giữ lại khoá của quan hệ đó

+ nhóm vào một quan hệ khác các thuộc tính phụ thuộc vào một phần của khoá, lấy phần đó làm khoá chính cho quan hệ .

VD : trong quan hệ R2 (*Số hoá đơn, Số sản phẩm, Tên sản phẩm, Lượng yêu cầu*)
***có phụ thuộc hàm :* Số sản phẩm --> Tên sản phẩm**
Trong đó* Số sản phẩm *là một phần của khoá, ta tách R2 thành R3 và R4 như sau :

+ R3 (*Số hoá đơn, Số sản phẩm, Lượng yêu cầu*)

+ R4 (*Số sản phẩm, Tên sản phẩm*)

28.

29. *Dạng chuẩn 3NF*

Quan hệ đã ở dạng 2NF nhưng chưa ở dạng 3NF là có tồn tại các phụ thuộc hàm gián tiếp. Đưa về dạng 3NF ta làm như sau :

+ giữ lại trong quan hệ ban đầu các thuộc tính phụ thuộc trực tiếp vào khoá

+ nhóm vào một quan hệ khác các thuộc tính bắc cầu, lấy thuộc tính bắc cầu làm khoá

VD : trong quan hệ

R1 (Số hoá đơn, Ngày bán, Số khách hàng, Tên khách hàng, Số sản phẩm)
Có các phụ thuộc hàm bắc cầu :

Số hoá đơn --> Số khách hàng --> Tên khách hàng
Ta có thể tách R1 thành R5 và R6 như sau :

+ R5 (Số hoá đơn,Ngày bán, Số khách hàng, Số sản phẩm)

+ R6 (Số khách hàng ,Tên khách hàng)

30. *Dạng chuẩn BCNF*

Quan hệ đã ở dạng 3NF nhưng chưa ở dạng BCNF là có tồn tại phụ thuộc hàm có nguồn là thuộc tính không thuộc khoá nhưng có đích là thuộc tính thuộc khoá .

Ta xét ví dụ : R (Học sinh, Môn học, Giáo viên, Điểm)
Phụ thuộc hàm Giáo viên --> Môn học có nguồn không thuộc khoá nhưng đích thuộc khoá, ta tách R thành R1, R2 như sau :

+ R1 (<u>Học sinh, Giáo viên</u>, Điểm)
+ R2 (<u>Giáo viên</u>, Môn học)

Các phụ thuộc hàm đơn trị dừng lại ở dạng chuẩn BCNF (Boyce - Codd). Đến đây chúng ta có thể kết thúc công việc chuẩn hoá .

Lời kết ***:***

Mô hình CSDL quan hệ là một công cụ rất tiện lợi để mô tả cấu trúc logic của các CSDL . Như vậy ở mức logic mô hình này bao gồm các quan hệ được biểu diễn bởi các bảng. Do đó đơn vị của CSDL quan hệ là bảng, trong đó các dòng của bảng là các bản ghi dữ liệu cụ thể, còn các cột là các thuộc tính .

Đối với người sử dụng có thể nói CSDL quan hệ là một tập hợp các bảng biến đổi theo thời gian .

Đôi với công việc thiết kế một CDSL thì các công việc phân tích tư liệu của xí nghiệp và chuẩn hoá là hết sức quan trọng, phục vụ cho việc cài đặt thực tế .

<u>Tài liệu tham khảo</u> ***:***

- ***Cơ sở dữ liệu kiến thức và thực hành***
  PGS Vũ Đức Thi - NXB thống kê 1997

- ***Nhập môn cơ sở dữ liệu quan hệ***
  Lê Tiến Vương - NXB Khoa học và kĩ thuật 1997

- ***Bài giảng môn Cơ sở dữ liệu I - Khoa CNTT - Đại học KHTN***

Chương 1:

GIỚI THIỆU VỀ HỆ QUẢN TRỊ CSDL VISUAL FOXPRO

1.1 Tổng quan về FoxPro và Visual FoxPro

1.1.1 Giới thiệu

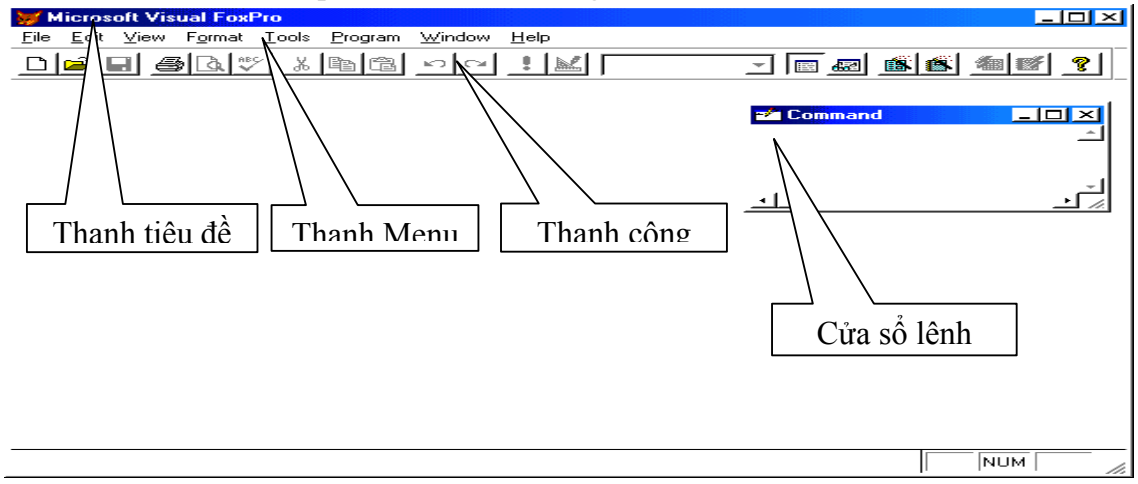
Foxpro là hệ quản trị cơ sở dữ liệu dùng để giải quyết các bài toán trong các lĩnh vực quản lý. FoxPro được thừa kế và phát triển trên phần mềm DBASE III PLUS và DBASE IV, những sản phẩm nổi tiếng của hãng ASTON-TATE. Khi các công cụ lập trình và các ứng dụng trên môi trường Windows ngày nhiều thì Microsoft cho ra đời các phiên bản FoxPro 2.6, chạy được trên hai môi trường DOS và Windows. Visual Foxpro là sản phẩm của hãng Microsoft, nó được kế thừa từ Foxpro for Windows, là một trong những công cụ tiện lợi để giải quyết các bài toán trong lĩnh vực quản lý cho những người chuyên nghiệp và không chuyên nghiệp. Từ khi phát triển đến nay, Hảng Microsoft đã cho ra đời nhiều phiên bản Visual Foxpro 3.0, 4.0, 5.0, 6.0.

1.1.2 Khởi động Visual Foxpro.

Sau khi đã cài đặt Visual FoxPro, ta có thể khởi động nó bằng cách thực hiện file FoxProw.exe hoặc file vfp.exe đối với Visual Foxpro theo các cách sau:

+ Kích chuột vào biểu tượng của FoxPro hoặc Visual Foxpro trên Desktop
+ Chọn menu Start/Program, chọn Microsoft Visual Foxpro và kích chuột vào đó.

Màn hình Visual Foxpro sau khi khởi động:



Visual FoxPro có 2 chế độ làm việc; chế độ tương tác (interactive) và chế độ chương trình (program).

Chế độ tương tác: Là chế độ trả lời từng câu lệnh một của người sử dụng, trong chế độ này có 2 hình thức đưa câu lệnh:

* Đưa câu lệnh qua menu hệ thống (system menu).

* Đưa câu lệnh từ cửa sổ lệnh (command window).

Chế độ chương trình: Các câu lệnh trong cửa sổ lệnh có thể tập trung thành một file và lưu trên đĩa (gọi là file chương trình nguồn). Khi muốn thực hiện các lệnh trong chương trình nầy, tại cửa sổ lệnh đưa vào các câu lệnh: DO < tên chương trình >

Để thoát khỏi Visual FoxPro, tại cửa sổ lệnh sử dụng lệnh QUIT

1.2 Các khái niệm cơ bản

1.2.1 Kiểu dữ liệu

Đối tượng xử lý của V. FOXPRO là dữ liệu, để quản lý và khai thác tốt các dữ liệu này, tuỳ theo tính chất, V.FOXPRO phải chia dữ liệu thành nhiều kiểu dữ liệu khác nhau: kiểu số (numberic), kiểu chuỗi (character), kiểu ngày tháng (date), kiểu lý luận (logical), kiểu bộ nhớ (memo), kiểu hình ảnh (picture).

a. **Kiểu số - Numeric (N)**: dùng để biểu diễn các số liệu mang giá trị số học và có nhu cầu tính toán như trong kế toán, quản lý, ... Mỗi dữ liệu kiểu số chiếm tối đa 20 chữ số gồm cả phần nguyên, phần thập phân và dấu chấm thập phân.

b. **Kiểu số - Float (F):** Dùng để biểu diễn số là các số có dấu chấm động như: 2.03e5 (2.03 x 10^5), thường được sử dụng trong các chương trình thuộc lĩnh vực khoa học kỹ thuật, ...

c. **Kiểu chuỗi - Charater (C):** Chứa các số liệu là tổ hợp một số bất kỳ các ký tự ASCII như tên, họ hoặc là số nhưng không có nhu cầu tính toán như số chứng minh, địa chỉ, số phòng, ... Mỗi dữ liệu kiểu chuỗi có độ dài tối đa 255 ký tự (mỗi ký tự chiếm 1 byte trong bộ nhớ).

d. **Kiểu ngày tháng - Data (D)**: Dùng cho những số liệu dạng ngày tháng như ngày sinh, ngày đến,.... Đó là những số nguyên dạng "yyyymmdd" khi hiển thị ra bên ngoài sẽ được chuyển thành dạng ngày tháng bình thường như mm-dd-yy, dd-mm-yyyy,... tuỳ theo yêu cầu của người

lập trình. Độ dài cố định của dữ liệu kiểu ngày là 8 ký tự.

e. **Kiểu logic - Logical (L)**: Dùng cho những dữ liệu chỉ có một trong hai trường hợp hoặc đúng (T) hoặc sai (F) như giới tính, đối tượng ưu tiên, ... Độ dài cố định của dữ liệu kiểu lý luận là 1 ký tự.

f. **Kiểu ghi nhớ - Memo (M):** Dữ liệu kiểu ghi nhớ là một đoạn văn bản có độ dài lớn hơn 255 ký tự, như khen thưởng, lý kịch, quá trình công tác,... Độ dài khai báo là 10 nhưng nội dung thực sự của kiểu ghi nhớ là tuỳ ý, chúng được lưu trữ trong một tập tin khác có cùng tên nhưng phần mở rộng là .FPT (FoxPro Text).

g. **Kiểu tổng quát - General (G)**: Dùng để chứa dữ liệu như bảng tính, âm thanh,...

h. **Kiểu hình ảnh - Ficture (P)**: Dữ liệu lưu dưới dạng hình ảnh .BMP, thường được dùng trong các chương trình "quản lý như sự", "nhận dạng",...

1.2.2 Các phép toán

a. **Phép toán số học**: Được thực hiện trên các dữ liệu kiểu số, gồm các phép toán:

Phép toán	Ý nghĩa	Ví dụ
-, +	dấu âm và dương	+5, -7
** hay ^	luỹ thừa	5**2,5^2
, /	nhân, chia	25, 5/7
%	phần dư (modulo)	25%5
+, -	cộng, trừ	10-2, 45+4

Độ ưu tiên các phép toán theo thứ tự đã nêu ở trên, có thể thay đổi thứ tự tính toán bằng cách đặt chúng trong 2 dấu ngoặc đơn ( ) như các quy tắc tính toán số học thông thường.

b. **Phép toán chuỗi**: Dùng để xử lý các dữ liệu kiểu chuỗi.

- Phép toán ghép nối (+): dùng để ghép 2 chuỗi cạnh nhau, kết quả của phép toán là một dữ liệu kiểu chuỗi.

Ví dụ: Trung tâm' + 'Tin học' -----> 'Trung tâm Tin học'

• Phép toán ghép nối (-): dùng để ghép 2 chuỗi cạnh nhau và di chuyển các dấu cách ở chuỗi thứ nhất (nếu có) ra cuối chuỗi tạo thành.

Ví dụ: 'Trung tâm ' - ' Tin học' -----> 'Trung tâm Tin học '

• Phép toán $: kiểm tra chuỗi bên trái có nằm trong chuỗi bên phải không. Kết quả của phép toán có kiểu logic.

Ví dụ: 'ab' $ "ABab" cho giá trị .T. nhưng 'ab $ "AaBb" cho giá trị .F.

c. **Phép toán ngày**: Hai dữ liệu kiểu ngày có thể trừ (-) cho nhau để cho khoảng cách đại số giữa 2 ngày.

Ví dụ: {01/08/2003} - {05/09/2003} -------> - 35

{01/08/2003} - {05/07/2003} -------> 25

Một dữ liệu kiểu ngày có thể cộng (+) hay trừ (-) một số nguyên để cho kết quả là một dữ liệu kiểu ngày.

Ví dụ: {01/08/2003}+ 10 -------> {11/08/2003}

{01/08/2003}- 20 -------> {12/07/2003}

Chú ý: • Hai dữ liệu kiểu ngày không thể cộng (+) cho nhau.

• Một số không thể trừ (-) với một dữ liệu kiểu ngày.

Việc diễn tả thứ tự ngày (D), tháng (M), năm (Y) trong một dữ liệu kiểu ngày còn phụ thuộc vào thời điểm hiện tại đang theo hệ thống ngày tháng nào.

(1) Lệnh SET DATE FRENCH |AMERICAN| JAPAN: Cho phép thiết lập dữ liệu dạng ngày theo kiểu Pháp|Mỹ|Nhật.

(2) SET CENTURY ON|OFF: Quy ước năm có một dữ liệu dạng ngày được biểu diễn theo dạng hai số (mặc định) hay dạng bốn số. Nếu SET CENTURY ON thì năm được biểu diễn theo dạng bốn con số, nếu SET CENTURY OFF (dạng mặc định) thì năm được biểu diễn theo dạng hai con số.

(3) Lệnh SET MARK TO <bthức C>: để ấn định ký tự phân cách ngày tháng, năm là <bthức C>. Dùng lệnh SET MARK TO để trở về ký tự phân cách ngày tháng mặc định.

d. **Phép toán quan hệ**: dùng để so sánh hai giá trị của hai biểu thức cùng kiểu

Phép toán	Ý nghĩa	Phép toán	Ý nghĩa
<	nhỏ hơn	<>, !	khác
>	lớn hơn	<=	nhỏ hơn hay bằng
=	bằng	=>	lớn hơn hay bằng
= =	bằng chính xác		

Hai dữ liệu kiểu số được so sánh dựa theo biểu diễn của chúng trên trục số.

Hai dữ liệu kiểu ngày được so sánh dựa theo biểu diễn của chúng theo chiều của thời gian.

Trong kiểu logic, Visual FoxPro quy ước: .T.<.F.

Hai dữ liệu kiểu chuỗi có độ dài bằng nhau được so sánh dựa theo nguyên tắc sau: đầu tiên so sánh 2 mã ASCII của 2 ký tự đầu của hai chuỗi, nếu bằng nhau thì so sánh tiếp.

Ví dụ: 'ABCD' < 'ABCE' ----------> .T. 'a' < 'A' ----------> .F.

Trường hợp hai chuỗi có độ dài khác nhau, thì việc so sánh dựa vào việc thiết lập môi trường SET EXACT ON/OFF, nghĩa là:

Nếu SET EXACT ON thì 'AB' = 'AB ' ----------> .F.

Nếu SET EXACT OFF thì 'ABCD' = 'AB' ----------> .T.

e. **Phép toán logic**: Visual FoxPro có 3 phép toán logic: NOT; AND; OR

NOT hay ! : phủ định của toán hạng theo sau.

AND : cho giá trị .T. nếu cả hai toán hạng đều .T.

OR : cho giá trị .F. nếu cả hai toán hạng đều .F.

1.2.3 Toán hạng

Toán hạng là các dữ liệu tham gia vào các phép toán.

Ví dụ: del=b^2 - 4*a*c thì b,2,4,a,c là các toán hạng.

1.2.4 Hằng

Là đại lượng có giá trị không đổi trong thời gian chương trình thực hiện. Trừ kiểu dữ liệu memo thì mỗi kiểu dữ liệu đều có hằng của nó.

Hằng kiểu số: như -2.5, 100, 4.14

Hằng kiểu chuỗi: hằng loại nầy phải để trong hai dấu "..." hoặc '...' hoặc [...], có độ dài tối đa không quá 253 kí tự.

Ví dụ: "abc"; tổng hợp', '123',.....

Hằng kiểu ngày: phải được đặt trong cặp dấu {...}

Ví dụ: {01/01/96}; {}: ngày rỗng.

Hằng logic: chỉ có 2 giá trị .T. và .F.

1.2.5 Biến

Biến là đại lượng dùng để lưu trữ dữ liệu trong quá trình tính toán. Biến có hai đặc trưng chính: tên biến và giá trị của biến. Tên biến được đặt theo nguyên tắc: dài không quá 10 kí tự, bắt đầu phải là chữ cái hoặc dấu _ phần còn lại là tổ hợp của bất kỳ các chữ cái, chữ số hoặc dấu _. Tên biến không nên đặt trùng tên các từ khoá của Visual FoxPro, tên biến có thể viết bằng chữ in hoa hay chữ thường. Visual FoxPro hiểu kiểu của biến là kiểu của giá trị mà nó đang mang. Số lượng tối đa của biến được phép sử dụng là 2048 biến.

Visual FoxPro chia biến làm 3 loại:

a. ***Biến bộ nhớ***: Gọi chung là biến, do người sử dụng tạo ra trong bộ nhớ, khi không sử dụng nữa có thể giải phóng để tiết kiệm bộ nhớ.

Ví dụ: hsl = 3.12

ngaysinh = {01/01/88}

b. ***Biến hệ thống***: Được tạo ra ngay từ khi khởi động Visual FoxPro. Có tên bắt đầu bằng dấu gạch nối (_) thường được sử dụng trong vấn đề in ấn, người sử dụng không thể giải phóng biến loại nầy.

c. ***Biến trường***: Tên các trường trong tập tin CSDL , nó chỉ có ý nghĩa khi tập tin chứa nó được mở ra để sử dụng.

Nếu có một biến đặt trùng với một biến trường thì biến trường được ưu tiên thực hiện trước.

Nếu tồn hại hai biến trường và biến bộ nhớ trùng tên nhau, để truy nhập đến chúng mà không sợ nhầm lẫn, bạn sử dụng quy cách sau cho biến bộ nhớ:

M.<tên trường> hay M -> <tên trường>

1.2.6 Hàm

Hàm là những đoạn chương trình được viết sẵn nhằm thực hiện một công việc nào đó. Các hàm nầy thường cho ra một giá trị, nhưng cũng có hàm chỉ thi hành một việc nào đó mà không cho ra một trị nào cả. Về hình thức hàm được đặc trưng bởi tên hàm và theo sau là cặp dấu ( ) dùng để bao các đối số, các đối số nầy đặt cách nhau bởi dấu phẩy. Một hàm có thể có nhiều đối số hoặc không có đối số nào cả nhưng phải có ( ) theo sau.

Ví dụ: Date ( ): cho biết ngày tháng năm hệ thống.

Sqrt(x): căn bậc 2 của x.

Có 2 loại hàm: Hàm có sẵn của Visual FoxPro và hàm tự tạo do người sử dụng tạo ra. Chúng ta sẽ nghiên cứu vấn đề nầy kỹ hơn ở chương sau.

1.2.7 Biểu thức

Biểu thức là tập hợp của một hay nhiều thành phần như hằng, hàm, biến, phép toán, dấu ngoặc tròn. Sau khi tính toán biểu thức sẽ cho một trị duy nhất. Trị của biểu thức thuộc về một trong 4 kiểu: N, C, D, L. Một biểu thức có thể rất phức tạp, trị của biểu thức được tính theo nguyên tắc sau:

* Trong ( ) tính trước, ngoài ( ) tính sau,

* Phép toán ưu tiên cao tính trước.

* Bên trái tính trước, bên phải tính sau.

1.2.8 Từ khoá

Từ khoá là những từ được Visual FoxPro sử dụng vào một mục đích riêng, người sử dụng không được đặt tên trùng với các từ khoá nầy. Thông thường từ khoá là những động từ động từ của lệnh thực hiện. Nếu từ khoá có nhiều hơn 4 ký tự thì khi sử dụng chỉ cần ghi 4 ký tự đầu.

Ví dụ: Câu lệnh MODIFY COMMAND LUONG.PRG có 2 từ khoá là MODIFY và COMMAND có thể viết gọn là: MODI COMM LUONG.PRG

1.2.9 Lệnh và chương trình

Lệnh là những yêu cầu để thực hiện một nhiệm vụ nào đó. Lệnh trong Visual FoxPro thường là một động từ, cũng có trường hợp là một kí hiệu như: !. ?, ... Tập hợp các lệnh nhằm đạt được một mục tiêu đề ra gọi là chương trình.

Trong Visual FoxPro có 3 cách để ban hành lệnh:

a. ***Dùng cửa sổ lệnh***:

Lệnh được đưa vào cửa sổ lệnh, sau khi ấn Enter lệnh được thi hành ngay. Thi hành xong một lệnh thì lệnh cũ được lưu lại trên cửa sổ lệnh có thể sử dụng cho lần sau. Cách nầy thường dùng trong những tính toán đơn giản để kiểm tra kết quả của lệnh.

b. *Dùng menu*:

Lệnh được ban hành bằng cách kích hoạt menu tương ứng, sau khi thi hành xong câu lệnh cũng được lưu lại trên cửa sổ lệnh. Cách nầy chỉ hạn chế trong một số lệnh thông thường trên tập tin CSDL.

c. Dùng chương trình: Soạn thảo trước một chương trình gồm nhiều lệnh thích hợp. Chương trình được lưu trên đĩa dưới tên một tập tin có phần mở rộng PRG. Để thực hiện chương trình này, tại cửa sổ lệnh đưa câu lệnh DO <tên file.PRG>. Sau khi ấn Enter chương trình được nạp vào bộ nhớ và từng lệnh được thực hiện theo thứ tự.

Bài thực hành chương 1

1. Giả sử có tập tin HSNV.DBF (có cấu trúc như đã mô tả ở bài 1, thực hành hai) trong đó có ít nhất 15 mẫu tin.

a. Dùng lệnh SORT để sắp xếp lại tập tin HSNV.DBF sang một tập tin mới HSNVSX.DBF theo chỉ tiêu: Các mẫu tin được sắp xếp theo từng đơn vị (giảm dần), trong mỗi đơn vị thứ tự tên, họ được sắp xếp tăng dần.

b. Mở tập tin HSNVSX.DBF

- Sử dụng lệnh LIST liệt kê các trường HOLOT, TEN, NGSINH, M_LUONG, MADV.
- Sử dụng lệnh USE để đóng tập tin lại.

c. Lập 3 tập tin chỉ mục: FMASO.IDX theo trường MASONV, FDONVI.IDX theo trường MADV, FLUONG.IDX theo trường M_LUONG giảm dần.

- Bằng cách thay thế tập tin chỉ mục chủ, hãy liệt kê các mẫu tin theo MASONV tăng dần, theo MADV tăng dần, theo M_LUONG giảm dần.

2. Trong tập tin HSNV.DBF

a. Dùng lệnh LOCATE:

- Tìm người có họ tên là 'LE VAN NAM' (giả sử có họ tên này trong tập tin HSNV.DBF). Dùng lệnh DISPLAY cho hiện nội dung của mẫu tin này, rồi dùng lệnh EDIT để sửa lại.

- Tìm những người ở phòng Hành chính (MADV='HC'), cho hiện đầy đủ thông tin của những người này.

- Tìm những người có mức lương > 310.

b. Dùng lệnh SEEK để tìm kiếm người có MASONV='TCH01' (giả sử mã này có trong tập tin HSNV.DBF). Cho hiện nội dung của mẫu tin này.

c. Cho biết địa chỉ của người có Họ tên là 'HO VAN HAO', sinh ngày 10/11/58 (bằng hai cách: LOCATE và SEEK).

---

1. CHƯƠNG 2: THAO TÁC VỚI BẢNG DỮ LIỆU

1.1. 2.1. KHÁI NIỆM

Bảng dữ liệu chứa dữ liệu theo dạng dòng và cột, mỗi dòng được gọi là một mẫu tin (record), mỗi cột được gọi là một trường (field) của bảng.

Mỗi bảng dữ liệu được lưu trử trên đĩa với tên file có phần mở rộng mặc định là DBF, mỗi bảng dữ liệu có hai phần: cấu trúc và nội dung của bảng.

Ví dụ: bảng nhân viên (nhanvien.dbf) có cấu trúc sau:

Fieldname	Type	Width	Decimal
Hoten	Character	30	
Gioitinh	Logic	1	
Ngaysinh	Date	8	
NamLV	Numberic	4	
Lylich	Memo	10	

Nội dung của NHANVIEN.DBF

Hoten	Gioitinh	Ngaysinh	NamLV	Lylich
Nguyen van A	.T.	10/15/75	1999	Memo
Le thi Nhan	.F.	06/15/70	1995	Memo
.....	.....	.....	.....	.....

1.2. 2.2 FILE VÀ KIỂU FILE TRONG VISUAL FOXPRO

2.2.1 Các kiểu file chính của Foxpro

FoxPro có các kiểu file sau:

*.dbf: File dữ liệu

*.idx: File chỉ mục

*.prg: File chương trình

*.dbc: File cơ sở dữ liệu

*.dll: File thư viện liến kết động

*.pjx: File dự án

*.scx: File Form

*.vcx: File thư viện

2.2.2. Cách tổ chức một file dữ liệu

a. File dữ liệu: Là tập hợp dữ liệu phản ánh về một tập hợp các đối tượng quản lý thông qua các thuộc tính của nó.

b. Bản ghi (Record): Là một bộ giá trị các thuộc tính phản ánh về một đối tượng quản lý.

c. Trường (Field): Là một thuộc tính trong file dữ liệu, mỗi trường được xác định bởi tên trường, kiểu trường và kích thước trường.

+ Tên trường (Field name): Tên trường dài tối đa 10 ký tự bao gồm chữ cái, chữ số, ký tự gạch dưới, ký tự đầu tiên của tên trường phải là chữ cái.

+ Kiểu trường (Field type): Kiểu trường có các dạng sau:

C: Charater	N:Numberic	L:Logic
D:Date	M:Memo	G:General

.....

+ Kích thước trường (Field Width): Là khoảng bộ nhớ cần thiết để lưu trữ các giá trị của trường, kích thước của trường phụ thuộc vào kiểu trường:

Kiểu C: Tối đa 254 Byte

Kiểu N: Tối đa 20 Byte kể cả dấu thập phân

Kiểu L: Chiếm 1 Byte

Kiểu D: Chiếm 8 Byte

Kiểu M: độ dài tuỳ ý, chiếm 10 Byte khi khai báo

Currency: Chiếm 8 byte

+ Cấu trúc file: Mỗi tổ hợp trường sắp xếp theo thứ tự nhất định gọi là cấu trúc của file dữ liệu, mỗi file dữ liệu chỉ có một cấu trúc cụ thể.

2.2.3. Nguyên tắc hoạt động

Chúng ta chỉ có thể truy nhập đến các phần tử của một file DBF nếu file đó đã được mở bằng lệnh USE <tên file DBF>.

Ở mỗi thời điểm bất kỳ, mỗi file DBF đang mở sẽ có một mẫu tin hiện thời, mẫu tin hiện thời

là mẫu tin có thể truy nhập vào thời điểm đó. Mẫu tin hiện thời được trỏ đến bởi con trỏ mẫu tin (record pointer). Mỗi mẫu tin đang mở có 2 vị trí đặc biệt chú ý: đầu file và cuối file. Để biết được con trỏ mẫu tin ở đầu hay ở cuối file ta dùng các hàm logic sau:

. Hàm BOF( ) (begin of file) cho giá trị .T. nếu con trỏ mẫu tin cuối file DBF đang mở, ngược lại hàm cho giá trị .F.

. Hàm BOF( ) (end of file) cho giá trị .T. nếu con trỏ mẫu tin cuối file DBF đang mở, ngược lại hàm cho giá trị .F.

. Số thứ tự của mẫu tin (record number - recno): mô tả số thứ tự vật lý của mẫu tin trong tập tin cơ sở dữ liệu DBF. Số thứ tự nầy do FoxPro qui định một cách tuần tự, được đánh số từ 1 đến mẫu tin cuối cùng. Trong khi làm việc, nếu xoá mọt mẫu tin thì số thứ tự này cũng tự động được cập nhật theo cho phù hợp.

. Hàm RECOUNT( ) dùng để biết số mẫu tin của một tập tin DBF đang mở.

. Hàm RECSIZE( ) dùng để biết được kích thước của một mẫu tin.

. Hàm RECNO( ) cho biết số thứ tự của mẫu tin hiện thời.

. Kích thước của các mẫu tin của một file DBF đều bằng nhau.

1.3. 2.3. CÁC LỆNH CƠ BẢN TRÊN FILE DBF

1.4. 2.3.1 Dạng lệnh tổng quát

Lệnh là một chỉ thị cho máy thực hiện một thao tác cụ thể. Một lệnh trong Foxpro nói chung có cú pháp tổng quát như sau:

Lệnh [phạm vi] [FIELDS <dsách trường>] [FOR <btL1>] [WHILE <btL2>] [FROM <tên file> / ARRAY <tên mảng>] [TO print/tên file/dsách biến]

Trong đó,

Lệnh: một từ khoá, cho biết mục đích của công việc, phải viết đầu tiên và có thể viết 4 kí tự đầu nếu lệnh có nhiều hơn 4 ký tự.

Ví dụ: DISPLAY FIELDS HOTEN, HSLUONG

DISP FIEL HOTEN, HSLUONG

Phạm vi (Scope): chỉ định phạm vi các mẫu tin chịu sự tác động của lệnh, phạm vi có thể là:

• ALL: tất cả các mẫu tin trong file dữ liệu đều bị tác động của lệnh (nếu có sử dụng FOR thì phạm vi được hiểu là ALL).

• NEXT <n>: n mẫu tin tiếp theo tính từ mẫu tin hiện thời bị tác động của lệnh.

• RECORD <n> Lệnh chỉ tác động đến mẫu tin thứ n

• REST Lệnh sẽ tác động từ mẫu tin hiện thời cho đến hết.

FIELDS <dsách trường>: lệnh chỉ có tác dụng trên những trường có tên được nêu trong <dsách trường>.

FOR <btL1>: mẫu tin nào thoả mãn <btL1> mới bị tác động bởi lệnh.

WHILE <btL2>: chừng nào <btL2> còn đúng thì lệnh còn hiệu lực. Nghĩa là, lệnh sẽ tác động lên các bản ghi thoả mãn biểu thức logic đi kèm (có giá trị là .T.) cho đến khi gặp một bản ghi không thoả mãn biểu thức logic (có giá trị .F.) hoặc đến hết file dữ liệu. Nếu điều kiện sai thì lệnh được dừng ngay. Trong lệnh nếu vừa có FOR vừa có WHILE thì mệnh đề WHILE ưu tiên thực hiện trước.

FROM <tên file>: tên của file mà từ đó lệnh lấy số liệu để sử dụng cho file DBF đang mở.

TO PRINT/tên file/dsách biến: chuyển kết quả sau khi thực hiện lệnh đến máy in/file/biến.

Các mệnh đề theo sau lệnh có mặt hay không tuỳ trường hợp và không cần phải viết theo thứ tự như đã nêu.

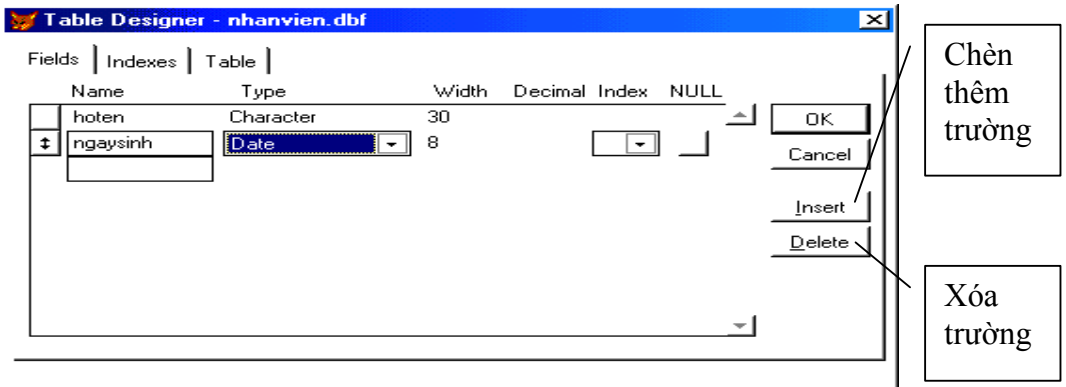
1.5. 2.3.2 Tạo bảng dữ liệu

Cú pháp: Create <tên file DBF> ↵ hoặc

Chọn File/New/<chọn loại file dữ liệu DBF>

Ví dụ: create nhanvien ↵ && tạo bảng nhanvien

Lúc này, màn hình sẽ xuất hiện hộp thoại để ta tạo cấu trúc bảng



Trong đó: Name: Tên trường

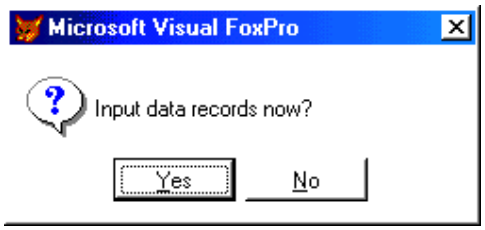
Type: Kiểu trường

Width: Độ rộng của trường

Decimal: Số chữ số lẻ sau phần dấu chấm thập phân, phần này chỉ sử dụng cho dữ liệu kiểu số.

Chú ý: - Các tên trường không được trùng nhau, không được trùng với từ khoá.

- Đối với dữ liệu kiểu số nếu có phần thập phân thì độ rộng của phần thập phân phải nhỏ hơn độ rộng của trường ít nhất là 2 đơn vị.



Để kết thúc việc nhập cấu trúc ta ấn đồng thời phím Ctrl+W, lúc này sẽ nhận được hộp thoại

Lúc này: trả lời <No> thì sẽ quay lại cửa sổ lệnh, trả lời <Yes> để tiến hành nhập các bản ghi. Khi chọn Yes sẽ tiếp tục xuất hiện hộp thoại để nhập dữ liệu.

Khi kết thúc việc nhập dữ liệu, nhấn tổ hợp <Ctrl+W> để lưu file dữ liệu lên đĩa. Khi đó file dữ liệu sẽ có dạng <tên file>.DBF

Chú ý: Để nhập dữ liệu cho trường MEMO, ta đưa con trỏ đến hộp memo rồi nhấn tổ hợp phím Ctrl_PgUp, lúc đó sẽ xuất hiện cửa sổ nhập dữ liệu cho trường này. Sau khi kết thúc việc nhập dữ liệu cho nó, ta ấn tổ hợp Ctrl+W để ghi lại.

1.6. 2.3.3 Định vị con trỏ đến một bản ghi

a. Định vị tuyệt đối

Cỳ phỏp: GO <N>|[TOP]|[BOTTOM]

Tác dụng: Dùng để chuyển con trỏ bản ghi đến bản ghi có số hiệu <n> được chi định trong câu

lệnh.

+ GO TOP: Dùng để chuyển con trỏ bản ghi về đầu file dữ liệu.

+ GO BOTTOM: Dùng để chuyển con trỏ bản ghi về cuối file dữ liệu.

b.Định vị tương đối

Cú pháp: Skip [+|-] [<n>]

Tác dụng: Di chuyển con trỏ bản ghi về trước (-) hay sau (+) so với bản ghi hiện thời.

Chú ý: Khi chỉ gõ lệnh Skip ↵ thì con trỏ bản ghi sẽ được di chuyển về sau bản ghi hiện thời một đơn vị.

1.7. 2.3.4 Lấy dữ liệu từ bảng

a. Lệnh Display

Cú pháp: display [<phạm vi>] [fields<danh sách trường>]

[For<bthức logic>] [While<bthức logic>] [on|off]

Tác dụng: Hiển thị nội dung của các bản ghi trong <phạm vi> được chỉ định và thoả mãn điều kiện của các biểu thức logic đi sau FOR và WHILE nếu có.

Theo mặc định thì tất cả các trường trong bảng dữ liệu sẽ được hiển thị, nếu có [field<danh sách trường>] thì những trường được chỉ ra trong danh sách này mới được hiển thị lên màn hình. <phạm vi> mặc định là bản ghi hiện thời.

Ví dụ: 1. Hiển thị tất cả các bản ghi của bảng dữ liệu nhanvien:

Use nhanvien ↵

Display all ↵

2. Hiển thị tất cả những người có năm làm việc (namlv) trước 1980

Display for namlv <1980 ↵

Chú ý: Trong câu lệnh của Fox, nếu có mệnh đề FOR thì phạm vi mặc định là ALL.

b. Lệnh LIST

Cú pháp: List [<phạm vi>] [fields<danh sách trường>] [For<bthức logic>] [While<bthức logic>] [on|off]

Tác dụng: Hiển thị nội dung của các bản ghi như lệnh Display nhưng mặc định của lệnh này là ALL

Ví dụ:1. Hiển thị tất cả các bản ghi của bảng dữ liệu nhanvien:

Use nhanvien ↵
List ↵

2. Hiển thị tất cả những người có năm làm việc (namlv) trước 1980

List for namlv <1980 ↵

c. Lệnh ?|??

Cú pháp: ?|?? <danh sách biểu thức>

Tác dụng: Lệnh này tính toán và cho hiển thị kết quả của danh sách biểu thức lên màn hình.

Chú ý: lệnh ? trước khi in dữ liệu thì xuống dưới 1 dòng, còn lệnh ?? thì không.

Ví dụ: Cho hiển thị họ tên của người có số hiệu là 5 lên màn hình:

Go 5 ↵
? 'ho ten', hoten ↵

Chú ý: danh sách biểu thức trong Fox được viết cách nhau bởi dấu phẩy ",".

Ví dụ: cho biết họ tên, năm làm việc của người có số hiệu là 2:

Go 2 ↵
?'ho ten:',hoten,'nam lam viec:',namlv↵

1.8. 2.3.5 Chèn, bổ sung bản ghi

a. Chèn bản ghi

Cú pháp: INSERT [BEFORE][BLANK]

Tác dụng: Chèn một bản ghi ngay sau bản ghi hiện thời (nếu có [Before]) với nội dung được nhập vào. Nếu có [BLANK] thì sẽ chèn một bản ghi trắng.

Ví dụ: chèn một bản ghi vào sau bản ghi thứ 3:

Go 3 ↵

Insert

b. Bổ sung bản ghi

Cú pháp: APPEND [BLANK]

Tác dụng: Để chèn 1 bản ghi vào cuối bảng dữ liệu (giá trị được nhập vào), nếu có tham số [BLANK] thì sẽ bổ sung một bản ghi trắng.

2.3.6 Sửa chữa nội dung bản ghi

a. Lệnh BROWSE:

CÚ PHÁP: BROWSE [FIELD<DSÁCH TRƯờNG>] [FREEZE<DSÁCH TRƯờNG>][NODELETE]

[NOEDIT] [FOR<BTHứC LOGIC>]

TÁC DụNG: HIểN THị NộI DUNG CủA BảNG Dữ LIệU, MỗI BảN GHI ĐƯợC THể HIệN TRONG MộT HÀNG (DÒNG), TA CÓ THể XEM VÀ DI CHUYểN HộP SÁNG Từ TRƯờNG NÀY QUA TRƯờNG KHÁC, BảN GHI NÀY SANG BảN GHI KHÁC VÀ CÓ THể SửA ĐổI NộI DUNG CủA TừNG MẫU TIN TRONG BảN GHI.

VÍ Dụ:

USE NHANVIEN ↵

BROWSE ↵

[FIELD<DSÁCH TRƯờNG>]: CHO PHÉP CÁC TRƯờNG TRONG DANH SÁCH NÀY ĐƯợC HIểN THị TRÊN MÀN HÌNH, NếU KHÔNG CÓ THAM Số NÀY THÌ TấT Cả CÁC TRƯờNG TRONG BảNG Dữ LIệU Sẽ ĐƯợC HIểN THị.

[FREEZE<DSÁCH TRƯờNG>]: CHO PHÉP CÁC TRƯờNG TRONG DANH SÁCH NÀY LUÔN ĐƯợC HIểN THị TRÊN MÀN HÌNH.

[NODELETE]: KHÔNG CHO PHÉP XOÁ

[NOEDIT]: KHÔNG CHO PHÉP SửA ĐổI.

VÍ Dụ: HIểN THị NộI DUNG CủA CÁC TRƯờNG HOTEN, NAMLV Để TIếN HÀNH SửA ĐổI.

BROWSE FIELD HOTEN,NAMLV FREEZE NAMLV

[FOR<BTHứC LOGIC>]: CHỉ CHO PHÉP NHữNG BIểU THứC THOả MÃN ĐIềU KIệN CủA BIểU THứC LOGIC MớI ĐƯợC HIểN THị.

b. Lệnh Edit

CÚ PHÁP: EDIT [<PHạM VI>] [FIELD<DSÁCHTRƯờNG>][NOAPPEND][NODELETE] [NOEDIT]

[FOR<BTHứC LOGIC>] [WHILE<BTHứC LOGIC>]

TÁC DụNG: TƯƠNG Tự NHƯ LệNH BROWSE NHƯNG CÁC BảN GHI ĐƯợC XUấT HIệN NHƯ ở LệNH APPEND.

c. Lệnh REPLACE

CÚ PHÁP: REPLACE [<PHạMVI>]<TRƯờNG 1>WITH<BTHứC 1>[ADDITIVE]

[,<TRƯờNG 2> WITH <BTHứC 2> [ADDITIVE]...][FOR<BTHứC LOGIC>]

[WHILE<BTHứC LOGIC>]

TÁC DụNG: DÙNG Để THAY THế NộI DUNG CÁC TRƯờNG ĐƯợC CHỉ RA CủA CÁC BảN GHI NằM TRONG <PHạM VI> VÀ THỏA MÃN ĐIềU KIệN CủA <BIểU THứC LOGIC> ĐI SAU FOR HOặC WHILE BởI CÁC BIểU THứC TƯƠNG ứNG. PHạM VI MặC ĐịNH LÀ BảN GHI HIệN THờI.

CHÚ Ý: KIểU Dữ LIệU CủA <BIểU THứC> VÀ CủA <TRƯờNG> TƯƠNG ứNG PHảI TƯƠNG ĐƯƠNG NHAU, NếU KHÔNG THÌ FOX Sẽ THÔNG BÁO LỗI KIểU Dữ LIệU "DATA TYPE MISMATCH".

VÍ Dụ: 1. THAY THế Họ TÊN CủA NHÂN VIÊN TRONG FILE NHANVIEN BằNG CHữ IN

REPLACE ALL HOTEN WITH UPPER(HOTEN) ↵

2. NÂNG LƯƠNG CủA NHữNG NHÂN VIÊN Nữ LÊNH THÊM 50000 ĐồNG

REPLACE LUONG WITH LUONG+50000 FOR !GIOITINH ↵

1.9. **2.3.7 Xoá bản ghi**

VIệC XOÁ MộT BảN GHI TRONG BảNG Dữ LIệU ĐƯợC THựC HIệN THEO HAI BƯớC:

BƯớC 1: ĐÁNH DấU BảN GHI MUốN XOÁ:

CÚ PHÁP: DELETE [<PHạM VI>] [FOR<BTHứC LOGIC>] [WHILE<BTHứC LOGIC>]

TÁC DụNG: LệNH NÀY ĐÁNH DấU TấT Cả CÁC BảN GHI THOả MÃN ĐIềU KIệN ĐƯợC NÊU, MặC ĐịNH LÀ BảN GHI HIệN THờI.

KHI THựC HIệN LệNH NÀY CÁC BảN GHI ĐƯợC CHỉ ĐịNH ĐÁNH DấU XOÁ Sẽ XUấT HIệN DấU * ở TRƯớC CÁC BảN GHI. LÚC NÀY TA CÓ THể PHụC HồI LạI CÁC BảN GHI ĐÓ ĐƯợC.

VÍ Dụ: ĐÁNH DấU XOÁ NHữNG NHÂN VIÊN CÓ NĂM LÀM VIệC TRƯớC 1951.

DELETE FOR NAMLV <1950↵

BƯớC 2. XOÁ CÁC BảN GHI. CÁC BảN GHI SAU KHI ĐÃ ĐƯợC ĐÁNH DấU XOÁ NếU QUYếT ĐịNH THậT Sự MUốN XOÁ NÓ THÌ THựC HIệN LệNH PACK, NGƯợC LạI NếU KHÔNG MUốN XOÁ NÓ THÌ THựC HIệN LệNH RECALL.

A. LệNH XOÁ CÁC BảN GHI Bị ĐÁNH DấU XÓA (PACK)

CÚ PHÁP: PACK

TÁC DụNG: XOÁ CÁC BảN GHI TRONG BảNG Dữ LIệU ĐÃ ĐƯợC ĐÁNH DấU XOÁ BằNG LệNH DELETE.

B. LệNH PHụC HồI CÁC BảN GHI ĐÃ ĐƯợC ĐÁNH DấU XOÁ (RECALL):

CÚ PHÁP: RECALL [<PHạM VI>] [FOR<BTHứC LOGIC>] [WHILE<BTHứC LOGIC>]

TÁC DụNG: PHụC HồI LạI CÁC BảN GHI MÀ TRƯớC ĐÓ ĐÃ ĐƯợC ĐÁNH DấU XOÁ BởI LệNH DELETE. PHạM VI MặC ĐịNH CủA LệNH NÀY LÀ BảN GHI HIệN THờI.

C. LệNH XÓA Dữ LIệU TRÊN FILE DBF.

CÚ PHÁP: ZAP

TÁC DụNG: XÓA TấT Cả CÁC BảN GHI TRONG MộT FILE DBF ĐANG Mở.

1.10. **2.3.8 Lọc dữ liệu**

Để HạN CHế Số LƯợNG CÁC BảN GHI THAM GIA VÀO QUÁ TRÌNH Xử LÝ, TA CÓ THể LọC CÁC BảN GHI TRONG BÀNG Dữ LIệU THOả MÃN ĐIềU KIệN CHO TRƯớC.

CÚ PHÁP: SET FILTER TO <BTHứC LOGIC>↵

SAU KHI THựC HIệN LệNH LọC THÌ CÁC LệNH TIếP THEO SAU LệNH NÀY CHỉ CÓ TÁC DụNG ĐốI VớI CÁC BảN GHI THOả MÃN ĐIềU KIệN LọC.

MUốN HUỷ Bỏ VIệC LọC Dữ LIệU TA THựC HIệN LệNH: SET FILTER TO ↵

VÍ Dụ:

1. CHỉ HIểN THị NHữNG NHÂN VIÊN Nữ:

```
SET FILTER TO !GIOITINH↵
LIST↵

.....
```

2. CHÍ XÉT NHữNG NHÂN VIÊN CÓ QUÊ QUÁN LÀ HUế

```
SET FILTER TO QUEQUAN=="HUE" ↵
LIST↵

.........
```

1.11. **2.3.9 THAO TÁC VỚI CấU TRÚC BẢNG:**

a. Xem cấu trúc bảng (List|Display structure)

CÚ PHÁP: LIST | DISPLAY STRUCTURE↵

TÁC DụNG: HIểN THị CấU TRÚC CủA BảNG Dữ LIệU ĐANG ĐƯợC Mở, BAO GồM: TÊN TRƯờNG, KIểU VÀ Độ RộNG CủA TRƯờNG.

VÍ Dụ:

USE NHANVIEN↵
LIST STRUCTURE

b. Sửa đổi cấu trúc bảng dữ liệu

CÚ PHÁP: MODIFY STRUCTURE↵

TÁC DụNG: HIểN THị VÀ CHO PHÉP SửA ĐổI CấU TRÚC BảNG Dữ LIệU, KếT THÚC LệNH NÀY NHấN Tổ HợP PHÍM CTRL+W.

VÍ Dụ: USE NHANVIEN↵
MODIFY STRUCTURE↵

c. Sao lưu cấu trúc bảng dữ liệu

CÚ PHÁP: COPY STRUCTURE TO <TEN FILE.DBF> [FIELDS<DANH SÁCH TRƯờNG>]↵

TÁC DụNG: Để SAO CHÉP CấU TRÚC CủA BảNG Dữ LIệU ĐANG ĐƯợC Mở SANG MộT BảNG MớI CÓ TÊN ĐƯợC CHỉ RA TRONG <TEN FILE.DBF> VớI CÁC TRƯờNG ĐƯợC CHỉ RA TRONG MụC [FIELD<DANH SÁCH TRƯờNG>]. MặC ĐịNH CủA LệNH NÀY LÀ TấT Cả CÁC TRƯờNG CÓ TRONG BảNG Dữ LIệU ĐANG ĐƯợC Mở.

VÍ Dụ: SAO LƯU CấU TRÚC CủA NHANVIEN THÀNH FILE CÓ TÊN LÀ LUU.DBF NHƯNG CHỉ GồM CÁC TRƯờNG: HOTE, GIOITINH, NĂMLV.

USE NHANVIEN↵

COPY STRUCTURE TO LUU FIELDS HOTEN, GIOITINH, NAMLV↵

CHÚ Ý: BảNG MớI ĐƯợC TạO RA CHỉ CÓ CấU TRÚC, KHÔNG CÓ NộI DUNG.

1.12. 2.3.10 Sao chép bảng

CÚ PHÁP: COPY TO <TÊN BảNG ĐÍCH> [<PHạM VI>] [FIELDS <DANH SÁCH TRƯờNG>] [FOR<BTHứC LOGIC>] [WHILE<BTHứC LOGIC>]↵

TÁC DụNG: LệNH DÙNG Để TạO BảNG MớI CÓ TÊN ĐƯợC CHỉ RA <TÊN BảNG ĐÍCH> VớI NộI DUNG ĐƯợC LấY Từ BảNG Dữ LIệU ĐANG ĐƯợC Mở. MặC ĐịNH LệNH NÀY LÀ TấT Cả CÁC BảN GHI ĐềU ĐƯợC SAO CHÉP, NếU CÓ PHạM VI VÀ CÁC BIểU THứC LOGIC THÌ NHữNG BảN GHI THOả MÃN ĐIềU KIệN MớI ĐWOCJ SAO CHÉP. DANG SÁCH TRƯờNG Để CHỉ ĐịNH CÁC TRƯờNG ĐƯợC SAO CHÉP.

VÍ Dụ: TạO BảNG Dữ LIệU CÓ TÊN LÀ NU.DBF Từ FILE NHANVIEN.DBF GồM CÁC TRƯờNG HOTEN, NGAYSINH, NAMLV.

```
USE NHANVIEN↵
COPY TO NU FIELDS HOTEN, NGAYSINH, NAMLV FOR !GIOITINH↵
USE NU↵     &&Mở Để XEM KếT QUả
LIST↵
```

1.13. 2.4. MỘT SỐ HÀM THÔNG DỤNG

2.4.1. Các hàm về ngày tháng

A. HÀM DATE(): **CHO NGÀY, THÁNG, NĂM HIệN TạI CủA Hệ THốNG. THứ Tự NGÀY, THÁNG, NĂM CủA LệNH NÀY PHụ THUộC VÀO LệNH SET DATE.**

VÍ Dụ:

NếU TA THựC HIệN LệNH: SET DATE FRENCH↵

RồI THựC HIệN LệNH DATE()↵ THÌ NGÀY HIệN HÀNH CủA Hệ THốNG Sẽ ĐƯợC HIệN RA THEO THứ Tự LÀ NGÀY, THÁNG, NĂM.

B. HÀM YEAR(<BTHứC DATE>): **CHO NĂM (CÓ 4 CHữ Số) CủA <BTHứC DATE>.**

VÍ Dụ: YEAR(DATE()) → CHO NĂM HIệN TạI CủA NGÀY Hệ THốNG.

C. HÀM MONTH(<BTHứC DATE>): **CHO THÁNG HIệN TạI CủA BIểU THứC NGÀY**

VÍ Dụ: MONTH(DATE())→ CHO THÁNG CủA NGÀY Hệ THốNG.

D. HÀM DAY(<BTHứC DATE>): CHO NGÀY CủA BIểU THứC NGÀY.

VÍ Dụ: DAY(DATE())↵ → CHO NGÀY HIệN TạI.

2.4.2. Các hàm về chuỗi

A. HÀM LEN(<BTHứC C>): **CHO CHIềU DÀI CủA BIểU THứC, TÍNH BằNG BYTE CủA <BTHứC C>, CHUỗI RỗNG CÓ CHIềU DÀI LÀ 1.**

B. HÀM LEFT(<BTHứC C>, <N>): **TRÍCH RA MộT CHUỗI Từ <BTHứC C> GồM <N> KÝ Tự TÍNH Từ BÊN TRÁI SANG.**

VÍ Dụ: ?LEFT("NGUYEN VAN AN", 6) → CHO KếT QUả LÀ "NGUYEN".

C. HÀM RIGHT(<BTHứC C>, <N>): **TRÍCH RA MộT CHUỗI Từ <BTHứC C> GồM <N> KÝ Tự TÍNH Từ BÊN PHảI SANG.**

*D. HÀM SUBSTR (<BTHứC C>, <N1>, <N2>):***TRÍCH RA MộT CHUỗI CON CủA <BTHứC C> Từ Vị TRÍ <N1> VÀ GồM <N2>) KÝ Tự.**

VÍ Dụ: ? SUBSTR ("NGUYEN VAN AN", 8, 3" KếT QUả CHO CHUỗI "VAN".

E. HÀM ALLTRIM (<BTHứC C>): **CHO KếT QUả LÀ MộT CHUỗI SAU KHI ĐÃ LOạI Bỏ CÁC KÝ Tự TRắNG ở HAI BÊN (NếU CÓ) CủA<BTHứC C>.**

VÍ Dụ: ?ALLTRIM("NGUYEN VAN AN ") → "NGUYEN VAN AN"

F. HÀM UPPER(<BTHứCC>): **CHO KếT QUả LÀ CHUỗI IN HOA CủA <BTHứC C>.**

VÍ Dụ: ?UPPER ("NGUYEN VAN AN") → "NGUYEN VAN AN"

G. HÀM LOWER <BTHứC C>: **NGƯợC LạI CủA HÀM UPPER.**

2.4.3. Các hàm số học

A. ASB(X): **CHO GIÁ TRị TUYệT ĐốI CủA X.**

B. INT(X): **CHO PHầN NGUYÊN CủA X.**

C. ROUND(X,<N>): **LÀM TRÒN X VớI N Số Lẽ.**

E. SIN(X): **CHO GIÁ TRị SIN X**

F. COS (X): **CHO GIÁ TRị COS X.**

Bài thực hành chương 2

1. Tạo tập tin DBF.

Dùng lệnh Create từ cửa sổ lệnh để tạo cấu trúc cho tập tin HSNV.DBF như sau: (Chỉ tạo cấu trúc, không nhập dữ liệu).

2. **Field Name**	**Field Type**	**Width**	**Dec**	**Phần ghi chú**
MASONV	Character	5		Mã số nhân viên
HOLOT	Character	20		Họ lót
TEN	Character	7		Tên
PHAI	Logic	1		Phái (Nam, Nữ)
DIACHI	Character	30		Địa chỉ
NGSINH	Date	8		Ngày sinh
TDVH	Numeric	2		Trình độ văn hoá
M_LUONG	Numeric	3		Mức lương
NGAYLL	Date	8		Ngày lên lương

Ghi chú: Trình độ văn hoá được đánh giá qua các mã sau:

0: Mù chữ, 1-12: Phổ thông, 13: Đại học, 14, Cao học, 15: Tiến sĩ

b. Cho biết công dụng của phím F5 và F6

c. Thêm vào tập tin vừa tạo ra hai trường mới.

3. **Field Name**	**Field Type**	**Width**	**Dec**	**Phần ghi chú**
MADV	Character	2		Mã đơn vị
HOHANG	Memo	10		Họ hàng

d. Ở trường PHAI sửa lại tên là NU có kiểu Logic.

e. Nhập số liệu 10 nhân viên vào tập tin HSNV.BDF này.

Ghi chú: Để nhập dữ liệu vào vùng HOHANG (kiểu Memo) dùng phím Ctrl_Home và kết thúc bằng Ctrl_W.

2. Dùng menu hệ thống tạo cấu trúc tập tin HOCVIEN.DBF sau đây:

4. **Field Name**	**Field Type**	**Width**	**Dec**	**Phần ghi chú**
MASONV	Character	4		Mã số nhân viên
HO	Character	20		Họ lót
TEN	Character	7		Tên
NAM	Logic	1		Nam: .T., Nữ: .F.
NGSINH	Date	8		Ngày sinh
NOISINH	Character	2	5	Nơi sinh
DIACHI	Character	20		Địa chỉ
MALOP	Character	4		Mã lớp
MAGV	Character	3		Mã giáo viên
DIEMLT	Numeric	5	2	Điểm lý thuyết
DIEMTH	Numeric	5	2	Điểm thực hành
UUTIEN	Logic	1		Ưu tiên
GHICHU	Date	10		Ghi chú

Nhập số liệu 10 học viên đầu tiên

b. Dùng lệnh DIR ở cửa sổ lệnh để xem tập tin có trên đĩa hay không, số mẫu tin vừa nhập và dung lượng đĩa còn trống?

c. Gõ lệnh: Use để đóng tập tin HOCVIEN.DBF rồi thoát khỏi FoxPro.

C□p nh□t d□ li□u

1. Mở tập tin HOCVIEN.DBF

- Dùng lệnh LIST hay DISPLAY ALL để xem nội dung các mẫu tin của tập tin HOCVIEN.DBF và chỉ xem các vùng tin MAHV, HO, TEN, NAM, MALOP, MAGV, DIEMLT, DIEMTH, nếu có sai sót hãy điều chỉnh cho đúng.

2. Gõ lệnh SET STATUS ON để xem thanh trạng thái.

- Nếu thanh trạng thái bị che khuất bởi cửa sổ lệnh thì ấn Ctrl_F7 để di chuyển cửa sổ đến vị trí khác.

- Nếu bóng mờ dưới cửa sổ lệnh che lấp thanh trạng thái thì gõ SET SHADOW OFF để tắt đi.

3. Dùng lệnh APPEND để thêm hai mẫu tin mới rồi ấn Ctrl_W để ghi lại.

4. Dùng hàm RECNO() cho biết số hiệu của mẫu tin hiện hành dời con trỏ đến đầu tập tin.

5. Sử dụng lệnh EDIT để sửa nội dung các mẫu tin tuỳ ý thích của bạn, sửa xong ấn Ctrl_W để ghi lại.

6. Gõ lệnh: BROWSE. Quan sát màn hình rồi thử các động tác sau:

a. Đưa vệt sáng đến mẫu tin thứ nhất tại vùng ghi chú, rồi ấn Ctrl_Home để xem phần ghi chú có những nội dung gì? Gõ thêm một ghi chú tuỳ ý rồi ấn Ctrl_W để ghi lại.

b. Ấn Alt+B để gọi MENU của BROWSE, sau đó gọi APPEND, nhập thêm 1 mẫu tin rồi ấn Ctrl_W để ghi lại.

c. Gõ lại lệnh BROWSE lần nữa, ấn Ctrl_N, FoxPro sẽ thêm mẫu tin trắng ở cuối, nhập số liệu cho mẫu tin này.

d. Đưa vệt sáng đến vùng DIEMLT, ấn Alt+B để gọi MENU phụ, sau đó chọn Move rồi chuyển vệt sáng đến vùng DIEMTH, ấn Enter. Kết quả hai cột DIEMLT và DIEMTH sẽ được chuyển cho nhau.

e. Đưa vệt sáng đến vùng NOISINH, ấn Alt+B để gọi menu SIZE, dùng mũi tên trái thu hẹp cột này còn 10 Bytes thôi, sau đó gõ: “Vĩnh lợi-Huế” vào mẫu tin thứ tư.

7. Gõ lệnh DELETE ALL FOR DIEMTH < 7 rồi xem có bao nhiêu mẫu tin bị đánh dấu xoá?

8. Gõ lệnh BROWSE để quan sát, sau đó đưa vệt sáng đến một mẫu tin bị đánh dấu xoá rồi ấn Ctrl_T xem dấu xoá có còn hay không, ấn lại Ctrl_T lần nữa để xem điều gì xảy ra, sau đó ấn Ctrl_W để thoát ra.

9. Gõ lệnh RECALL ALL để phục hồi các mẫu tin bị đánh dấu xoá rồi đóng tập tin HOCVIEN.DBF lại.

10. Mở tập tin HSNV.DBF

a. Dùng lệnh REPLACE để tăng lương gấp đôi cho tất cả nhân viên, sau đó tăng thêm riêng cho các nữ nhân viên 10% nữa.

b. Thêm vào cấu trúc tin HSNV.DBF một trường LOAIBC (loại biên chế: BC/HD) và dùng BROWSE nhập dữ liệu cho vùng tin này, ấn Ctrl_B để thoát khỏi BROWSE.

c. Gõ lệnh DISPLAY STRUCTURE (hay F5) để xem lại cấu trúc.

d. Đánh dấu xoá các nhân viên mù chữ và trình độ phổ thông cho liệt kê trên màn hình những mẫu tin không bị đánh dấu xoá.

e. Nhập thêm hai mẫu tin vào giữa tập tin HSNV.DBF.

- Một mẫu tin sau mẫu tin có STT=5

- Một mẫu tin trước mẫu tin có STT=3

f. Gõ lệnh RECALL ALL để phục hồi các mẫu tin bị đánh dấu xoá.

g. Liệt kê danh sách các nhân viên theo dạng.

MNV	HOLOT	TEN	NU	NGSINH	HSL	TDVH

h. Liệt kê theo ạng câu g những nhân viên nam.

i. Liệt kê theo dạng câu g những nhân viên nam từ 18 đến 30 tuổi.

j. Liệt kê theo dạng câu g những nhân viên nữ có trình độ đại học.

k. Liệt kê theo dạng câu g những nhân viên có tên bắt đầu bằng vần ‘H’

l. Gõ lệnh Use để đóng tập tin HXNV.DBF rồi thoát khỏi FoxPro.

5. CHƯƠNG 3: SẮP XẾP-TÌM KIẾM-THỐNG KÊ

5.1. *3.1. SẮP XẾP*

3.1.1. Khái niệm

Trong m□t b□ng d□ li□u, chỳng ta cú th□ s□p x□p cỏc m□u tin theo m□t tiờu chu□n n□o □ú tu□ theo yờu c□u c□a vi□c khai thỏc thụng tin.

3.1.2. Sắp xếp theo chỉ mục

a. Khái niệm về chỉ mục

Ta đã biết mỗi bảng dữ liệu chứa các bản ghi và mỗi bản ghi đều được đánh số hiệu theo số thứ tự từ 1 đến n.

Ví dụ: bảng NHANVIEN.DBF có dang sau:

Record#	HOTEN	NGAYSINH	GIOITINH	NAMLV
1	**NGUYễN VĂN A**	02/10/75	.T.	1985
2	Lê thị nhàn	05/23/75	.F.	1980
3	Nguyễn An	10/26/80	.T.	1982
4	Trần Hạnh	09/25/70	.T.	1981

Số hiệu các bản ghi

Khi x□ lý thụng tin trong b□ng d□ li□u, ta truy xu□t chỳng theo tr□t t□ c□a s□ hi□u b□n ghi.

Ví dụ: use NHANVIEN ↵
list ↵

Kết quả in ra sẽ như sau:

Record#	HOTEN	NGAYSINH	GIOITINH	NAMLV
1	**NGUYễN VĂN A**	02/10/75	.T.	1985
2	Lê thị nhàn	05/23/75	.F.	1980
3	Nguyễn An	10/26/80	.T.	1982
4	Trần Hạnh	09/25/70	.T.	1981

Sắp xếp bảng dữ liệu theo chỉ mục là tạo ra một file mới (có phần mở rộng mặc định là .IDX) chỉ có hai trường: trường khoá sắp xếp và trường số hiệu bản ghi. Thứ tự của bản ghi ở đây là thứ tự sắp xếp.

Vớ d□: file ch□ m□c c□a b□ng nhanvien theo th□ t□ t□ng d□n c□a n□m lẩm vi□c nh□ sau:

file chỉ mục theo namlv

Namlv	Record#
1980	2
1981	4
1982	3
1985	1

Lúc này, khi truy xuất dữ liệu của bảng, thứ tự của các bản ghi là thứ tự được quy định trong file chỉ mục này.

Vớ d□: Trong b□ng nhanvien, s□ d□ng ch□ m□c theo tr□□ng namlv.idx ta cú th□ t□ truy xu□t:

Record#	Hoten	ngaysinh	gioitinh	namlv
2	Lê thị nhàn	05/23/75	.F.	1980
4	Trần Hạnh	09/25/70	.T.	1981
3	Nguyễn An	10/26/80	.T.	1982
1	Nguyễn văn A	02/10/75	.T.	1985

b. Lập chỉ mục IDX cho bảng dữ liệu

Cú pháp: INDEX ON <bthức khoá> TO <tên file idx>

[FOR<bthức logic>] [UNIQUE]↵

Tác dụng: Lệnh sắp xếp file dữ liệu theo chiều tăng dần của <Bthức khoá> của các bản ghi thoả mãn <Bthức logic> sau FOR, mặc định là tất cả các bản ghi. Nếu có từ khoá [UNIQUE] thì các bản ghi nào có <Bthức khoá> trùng nhau sẽ bị bỏ qua trên file chỉ mục.

Ví dụ 1: Hiển thị theo thứ tự tăng dần của namlv của các nhân viên.

```
use NHANVIEN↵
index on NAMLV to CMNAMLV↵
list↵
```

Ví dụ 2: Hiển thị theo thứ tự tăng dần của hoten

index on HOTEN to CMHOTEN↵

list↵

Chú ý: Lệnh luôn sắp xếp theo thứ tự tăng dần của <bthức khoá>, do vậy khi lựa chọn <bthức khoá> thì phải chọn cho phù hợp.

Ví dụ 1: Hiển thị theo thứ tự giảm dần của namlv của các nhân viên.

use NHANVIEN↵

index on -NAMLV to CMNAMLVG↵

list↵

Ví dụ 2: Hiển thị theo thứ tự giảm dần của ngaysinh.

use NHANVIEN↵

index on date()-NGAYSINH to CMNSINHG↵

list↵

c. Một số lệnh liên quan

+ SET INDEX TO <file chỉ mục>: Dùng để mở file chỉ mục sau khi đã mở một bảng dữ liệu.

+ SET INDEX TO: Dùng để đóng file chỉ mục.

+ REINDEX: Dựng □□ c□p nh□t l□i file ch□ m□c sau khi cú s□ s□a □□i trờn b□ng d□ li□u.

5.2. *3.2. TÌM KIẾM*

3.2.1. Tìm kiếm tuần tự

a. Lệnh Locate:

Cú pháp:

LOCATE [<phạm vi>] FOR<bthức logic> [WHILE<bthức logic>]

Tác dụng: Lệnh sẽ duyệt tuần tự các bàn ghi trong bảng dữ liệu và tìm đến bản ghi đầu tiên trong <phạm vi> thoả mãn điều kiện của <bthức logic>. Nếu tìm được, hàm FOUND() sẽ cho giá trị .T., hàm EOF() có giá trị .F.

Ví dụ: Tìm nhân viên đầu tiên trong bảng dữ liệu sinh năm 1970 trong bảng nhanvien

```
use NHANVIEN↵
Locate for year(NGAYSINH) = 1970↵
Display↵
```

b. Lệnh continue

Cú pháp : CONTINUE

Chức năng : Theo sau lệnh LOCATE, dùng để tìm bảng ghi kế tiếp sau thỏa mãn điều kiện đã nêu.

Ví dụ : Tìm 2 nhân viên đầu tiên sinh năm 1970

```
use NHANVIEN
locate for year ( NGAY SINH) = 1970
display
continue
display
```

3.2.2. Tìm kiếm sau khi đã lập chỉ mục

Cú pháp : SEEK <biểu thức>

Chức năng : sau khi đã lập chỉ mục theo <bthức khóa> đề tìm bản ghi nào thỏa mãn một điều kiện dựa vào <bthức khóa>

Ta sử dụng lệnh SEEK theo sau là <giá trị> của biểu điều kiện cần tìm. nếu tìm thấy thì hàm FOUND() có giá trị .T. và hàm EOF () có giá trị .F.

Ví dụ: 1. Sắp xếp theo thứ tự tăng dần của Họ Tên, tìm nhân viên có tên "Nguyen Van AN".

use NHANVIEN

index on upper(HOTEN) to CMHOTEN

seek “Nguyen Van An”

disp

2. Sắp xếp theo thứ tự giảm dần của NAMLV, tìm nhân viên có năm làm việc 1981.

use NHANVIEN

index on - NAMLV to CMNAMLVG

list

seek -1981

disp

3.3. THỐNG KÊ

3.3.1. Đếm số lượng bản ghi

Cú pháp

COUNT [<phạm vi>][FOR<btlogic>] [WHILE<btlogic>] [TO<biến nhớ>]

Chức năng :lệnh dùng để đếm số mẫu tin trong bảng dữ liệu hiện hành thỏa mãn điều kiện các <bthức logic> nằm trong phạm vi được chỉ ra. Kết quả được đưa ra màn hình hay đưa vào <biến nhớ> nếu có TO.

Ví dụ: Cho biết có bao nhiêu nhân viên có NAMLV là 1980

use NHANVIEN

count for NAMLV = 1980 to songuoi

?’ có songuoi: ‘, songuoi, ‘ làm việc năm 1980’

3.3.2. Tính tổng giá trị các trường kiểu số

Cú pháp: SUM [<phạm vi>] [<dsách bt>] [TO <ds biến>]

[FOR <bt logic>] [WHILE <btlogic>]

Chức năng : Lệnh sẽ lấy tổng theo các biểu thức được xây dựng dựa trên các trường kiểu số, của các bản ghi trong bảng dữ liệu; nằm trong <phạm vi> và thỏa mãn điều kiện của các <bthức logic>. Nừu không có <ds biểu thức> thì các trường kiểu số đều được lấy tổng.

Mặc định, kết quả được đưa ra màn hình; nếu có TO <dsbiến> thì kết quả của các <biểu thức> sẽ được đưa vào các <biến> tương ứng.

Chú ý : Phải tương ứng 1-1 giữa <ds biểu thức> và <ds biến>.

Ví dụ: Dựa vào bảng NHANVIEN, cho biết tổng LUONG phải trả và tổng PHUCAP là bao nhiêu.

```
use NHANVIEN
sum LUONG, PHUCAP to tongluong, tongpc
?’ tong luong la:’ , tong luong
?’ tong phu cap la:’ , tongpc
```

3.3.3. Tính trung bình cộng các trường kiểu số

Cú pháp: AVERAGE [<phạm vi>] [<ds biểu thức>] [TO <ds biến >] [FOR <bt logic>] [WHILE <bt logic>]

Chức năng : giống như lệnh SUM ở trên nhưng sau khi lấy tổng, lệnh sẽ lấy giá trị đó đem chia cho tổng số bản ghi tham gia vào câu lệnh.

Ví dụ: dựa vào bảng NHANVIEN, cho biết trung bình mỗi nhân viên nhận được bao nhiêu LUONG, PHU CAP.

```
use NHANVIEN
average LUONG, PHUCAP  to tbluong, tbphucap
?’ trung binh luong:’ , tbluong
?’ trung bình phu cap:’ , tbphucap
```

3.3.4. Tính tổng các trường số theo nhóm

CÚ PHÁP: TOTAL ON <BT KHÓA> TO <TÊN BảNG MớI.DBF>[<PHạM VI>]

[FIELD <dstrường>][FOR <biểu thức L>][WHILE < biểu thức L>]

Chức năng: Lệnh sẽ cộng dồn các trường kiểu số theo từng nhóm bản ghi có <bt khóa> giống nhau và đưa vào bảng mới có tên được chỉ ra ở <tên bảng .DBF>. Mặc định thì tất cả các trường kiểu số đều được cộng dồn, nếu có FIELDS <danh sách trường> thì chỉ có các trường liệt kê mới được cộng. Lệnh chỉ tác động đến các bàn ghi nằm trong (phạm vi) và thoả mãn điều kiện đi sau các mệnh đề FOR, WHILE.

Chú ý: Trước khi dùng lệnh này, bảng dữ liệu phải định sắp xếp theo khoá.

Ví dụ: Dựa vào bảng VATTV, hãy thống kê xem mỗi mặt hàng đã xuất hay nhập một số lượng là bao nhiêu.

```
use     VATTV
index  on  MAXN + MAVT   to  CMTK
total on   MAXN + MAVT   to THONGKE fields SOLUONG
use   THONGKE
?  'chi tiet la :'
list  MAXN, MAVT, SOLUONG, DONGIA
```

Giả sử bảng VATTU sau khi sắp xếp là:

MAXN	SOCT	MAVTU	SOLUONG	DONGIA
N	9	A01	145	5
N	4	A01	203	500
N	1	F01	123	200
N	2	F01	345	200
N	10	F01	654	180

Kết quả của bảng THONGKE.DBF là:

MAXN	SOCT	MAVTU	SOLUONG	DONGIA
N	9	A01	348	500

N	1	F01	469	200
N	10	F01	654	180

6. CHƯƠNG 4: LẬP TRÌNH TRÊN VISUAL FOXRO

6.1. *4.1. CHƯƠNG TRÌNH*

4.1.1 Khái niệm

Là một dãy lệnh liên tiếp được tổ chức vào 1 file chương trình, file chương trình mặc định có phần mở rộng là *. PRG.

Trong một chương trình, mỗi lệnh được viết trên một hàng và mỗi hàng chỉ chứa một lệnh tại một cột bất kỳ.

4.1.2. Soạn thảo chương trình

Để soạn thảo chương trình, từ cữa sổ lênh đưa vào lệnh;

MODIFY COMAND < tờn file ch□□ng trỡnh >

Lúc này xuất hiện cửa sổ chương trình để ta có thể đưa các lệnh vào cho nó.

Một chương trình foxpro thường có 3 phần.

a) Tạo môi trường làm việc : thường chứa các lệnh sau:

SET DATE FRENCH: đặt ngày tháng năm theo dạng DD-MM-YY

SET CURENCY ON : đặt năm có 4 chữ số

SET TALK OFF/ON : ẩn hiện các kết quả thực hiện lệnh

SET DEFAULT TO <đường dẫn> : đặt đường dẫn hiện thời

CLEAR: xoá màn hình hiển thị kết quả

CLOSE ALL: đóng các bảng dữ liệu, các file cơ sở dữ liệu,...

b) Phần thân chương trình:

Thực hiện các công việc mà chương trình yêu cầu như :

+ Cập nhập dữ liệu

+ xử lý, tính toán

+ Kết xuất thông tin

c) Kết thúc chương trình

+ Đóng các tập tin CSDL, các bảng dữ liệu đang sử dụng

+ Giải phóng biến nhớ

+ Trả lại các chế độ cho hệ thống.

d) Chú thích trong chương trình

Là các giải thích được thêm vào để làm rõ cho chương trình, phải được bắt đầu bởi dấu * hay &&

* : Bắt đầu một dòng

&& : Viết sau một lệnh

6.2. *4.2. BIẾN NHỚ*

4.2.1. Khai báo biến

a) Lệnh gán =

Cú pháp: <biến> = <biển thức>

Ví dụ: a = 5

ngay = Date()

b) Lệnh STORE

Cú pháp: STORE <bthức> to <ds biến>

Công dụng: Gán giá trị <bthức> cho <ds biến> ; nếu <biến> chưa tồn tại nó sẽ khai báo, nếu đã có thì thay thế bởi giá trị mới.

Ví dụ: STORE 0 To a, b, c

4.2.2. Nhập giá trị cho biến từ bàn phím

a) Lệnh ACCEPT

Cú pháp ACCEPT <bthức chuỗi> to <biến chuỗi>

Chức năng : Dùng để nhập một chuỗi từ bàn phím, kết thúc bởi phím ↵, giá trị nhận được sẽ đưa cho <biến>.

Ví dụ:

ACCEPT ‘nhap ho ten’ to bhoten

? ‘Ho ten vua nhap', bhoten

<Bthức chuỗi> là một câu nhắc nhở người sử dụng.

b. Lệnh INPUT

Cú pháp: INPUT <Bthức chuỗi> to <biến>

Tác dụng: Tương tự lệnh trên nhưng có thể nhận dữ liệu theo từng kiểu:

Kiểu Charater: Phải được đặt trong cặp dấu ' ... ' hay "... ".

Kiểu Numberic: Nhập dữ liệu kiểu số.

Kiểu Date: Phải được để trong dấu {}.

Kiểu Logic: Nhập giá trị .T. hay .F.

Ví dụ:

INPUT 'Nhap ngay sinh' TO bngaysinh

INPUT 'Nhap diem" TO bdiem

Chú ý: Trong hai lệnh trên, nếu biến chưa có thì nó sẽ tự khai báo, nếu đã có thì nó sẽ thay giá trị của biến bởi giá trị vừa nhập.

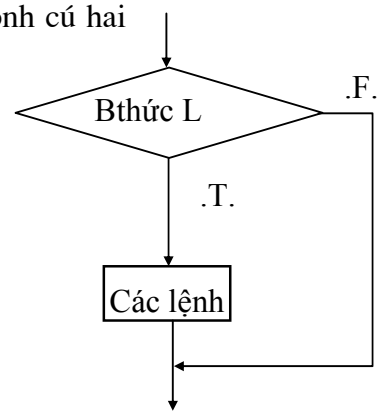
4.3. CÁC CẤU TRUẽC ĐIỀU KHIỂN CHƯƠNG TRÌNH

4.3.1. Cấu trúc tuần tự

Quy ước: Chương trình được thực hiện từ trên xuống dưới.

4.3.2. C□u trỳc r□ nhỏnh

Cũn g□i l□ c□u trỳc ch□n l□a. C□u trỳc r□ nhỏnh cú hai d□ng: d□ng khuy□t v□ d□ng □□y □□:



a. Dạng khuyết:

Cú pháp:

IF <Bthức L>

<các lệnh>

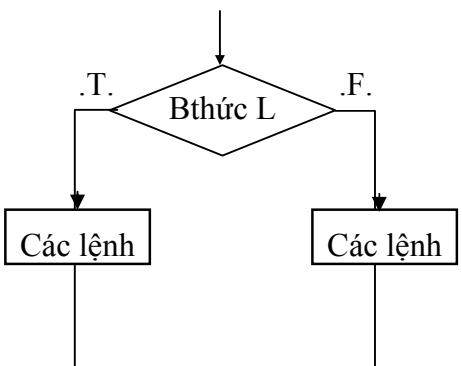
ENDIF

Tác dụng: Khi gặp cấu trúc này, <Bthức L> sẽ được tính, nếu có giá trị .T. thì <các lệnh> sẽ được thực hiện, ngược lại thực hiện các lệnh tiếp theo sau.

Vớ d□: Vi□t ch□□ng trỡnh nh□p v□o hai s□, cho bi□t s□ l□n nh□t

```
set talk off
clear
Input :Nhap so thu nhat' to so1
Input :Nhap so thu hai' to so2
max=so1
If max < so2
    max=so2
Endif
? "So lon nhat la:', max
set talk on
return
```

b. Dạng đầy đủ:



Cú pháp:

```
IF <Bthức L>
        <các lệnh 1>
ELSE
        <các lệnh 2>
ENDIF
```

Tác dụng: Khi gặp cấu trúc này, <Bthức L> sẽ được tính. Nếu có giá trị .T. thì <các lệnh 1> sẽ được thực hiện, ngược lại (có giá trị .F.) thì sẽ thực hiện <các lệnh 2>. Sau đó tiếp tục thực hiện các lênh tiếp theo trong chương trình.

Ví dụ: Dựa vào bảng nhanvien, hãy nhập vào một họ tên nhân viên, tìm xem có đúng là nhân viên của công ty hay không, nếu đúng thì thông báo năm sinh và năm làm việc, ngược lại thì thông báo là không phải nhân viên của công ty.

```
set talk off
clear
close all
use NHANVIEN
accept  'nhap ho ten nhan vien:' to bhoten
locate for HOTEN =bhoten
if found()
    ? ' ngay sinh là:', ngaysinh'
    ?'nam lam viec:', namlv
else
    ?'khong phai la nhan vien cua cong ty'
endif
set talk on
return
```

4.3.2.1. Lựa chọn một trong nhiều trường hợp

Cú pháp

```
    DO CASE
```

```
        CASE <Bthức L1>
                <các lệnh 1>
        CASE <Bthức L2>
                <các lệnh 2>
        .....
        CASE <Bthức Ln>
                <các lệnh n>
        [OTHERWISE
                <Các lệnh n+1>]
ENDCASE
```

Tác dụng: Khi gặp cấu trúc DO CASE, các <Bthức L> điều kiện sẽ được tính. Nếu <Bthức L> điều kiện nào đó có giá trị .T. thì nhóm lệnh tương ứng sẽ được thực hiện và kết thúc cấu trúc này, rồi tiếp tục thực hiện các lênh sau ENDCASE. Trong trường hợp không có <bthức L> nào từ 1 đến n có giá trị .T. thì <nhóm lệnh n+1> (nếu có) sẽ được thực hiện.

Ví dụ: Viết chương trình nhập vào một năm (có 4 chữ số), sau đó nhập thêm một tháng, cho biết tháng này có bao nhiêu ngày.

```
set takl off
clear
input 'nhap vao mot nam' to bnam
input 'nhap vao mot thang' to bthang
do case
        case bthang=4 or bthang = 6 or bthang = 9 or bthang=11
                songay=30
        case bthang=12
                if (mod(bnam, 4)=0 and (mod(bnam, 100)<>0)
                        songay=29
                else
                        songay=28
                endif
        otherwise
                songay=31
```

endcase

? 'thang', bthang, 'nam', bnam, 'co', so ngay, 'ngay'

set talk on

return

4.3.3. Cấu trúc lặp

4.3.3.1. Cấu trúc DO WHILE

Cú pháp:

DO WHILE <Bthức L>

<các lệnh>

[LOOP]

[EXIT]

ENDDO

Tác dụng: Khi gặp cấu trúc này thì <Bthức L> sẽ được tính, nếu có giá trị .F. thì sẽ dừng và thực hiện các lệnh sau ENDDO. Nếu có giá trị .T. thì các lệnh trong thân vòng lặp sẽ được thực hiện và lại quay về kiểm tra điều kiện trong <bthức L> và cứ thế tiếp tục.

[LOOP]: Khi gặp lệnh này, Foxpro sẽ quay về kiểm tra điều kiện logic mà bỏ qua các lệnh phía sau [LOOP].

[EXIT]: Khi gặp lệnh này thĩ sẽ thoát ra khỏi chương trình.

Ví dụ: Cho biết dang sáchhọ tên của các nhân viên trong công ty.

set takl off

clear

close all

use NHANVIEN

? 'danh sach ho ten hoc vien la:'

do while !eof()

?HOTEN

skip

enddo

set talk on

return

Chú ý: Khi sử dụng cấu trúc này, các lệnh trong thân vòng lặp phải thay đổi được giá trị của <Bthức L> để đảm bảo tính kết thúc.

Ví dụ: Nhập vào một năm, hãy thông báo danh sách họ tên, ngày sinh của những nhân viên làm việc trong năm đó, nếu không có thì thông báo là không có.

```
set takl off
set date french
clear
close all
use NHANVIEN
input 'nhap nam lam viec" to bnam
set filter to NAMLV = bnam
count to dem
if dem = 0
        ? ' khong co nhan vien nao'
else
        go top
        ? 'danh sach nhan vien lam viec năm', bnam
        ? "HO TEN                    NGAY SINH'
        do while !eof()
                ?HOTEN, NGAYSINH
                skip
        enddo
endif
set filter to
set talk on
return
```

4.3.3.2. Cấu trúc SCAN

Cú pháp

SCAN [<phạm vi>] [FOR<Bthức L>] [WHILE<bthức L>]

<các lệnh>

[LOOP]

[EXIT]

END SCAN

Tác dụng: Dùng để duyệt lần lượt các bản ghi trong bảng dữ liệu hiện hành nằm trong <phạm vi> được chỉ ra và thoả mãn điều kiện của các <Bthức L> sau FOR hoặc WHILE. Tương ứng với một bản ghi tìm được. <các lệnh> sẽ được thực hiện.

Cấu trúc SCAN sẽ dừng khi nào duyệt đến bản ghi cuối cùng của bảng dữ liệu đang xét.

Vớ d□: Vi□t ch□□ng trỡnh nh□p n□m (b□n ch□ s□), hi□n th□ nh□ng n□ nhõn viờn sinh n□m □ú.

set takl off

set date french

clear

close all

use NHANVIEN

input 'nhap nam lam viec" to bnam

input 'nhap nam:' to bnam

? 'DANH SACH CAC NU NHAN VIEN, SINH NAM', bnam

scan for !GIOITINH and year(NGAYSINH)=bnam

?HOTEN, NGAYSINH

endscan

set talk on

return

Bài thực hành chương 4

7. **Field Name**	**Field Type**	**Width**	**Dec**	**Phần ghi chú**
MASV	Character	5		Mã số nhân viên
HOLOT	Character	20		Họ lót
TEN	Character	7		Tên

NGSINH	Date	8		Ngày sinh
QUEQUAN	Character	20		Quê quán
DOS	Numeric	2		Điểm môn Dos
VRES	Numeric	2		Điểm VRER
FOX	Numeric	2		Điểm FoxPro
DTB	Numeric	4	2	Điểm trung bình
XEPLOAI	Character	4		Xếp loại

Nhập vào 10 mẫu tin cho các vùng: MASV, HOLOT, TEM, NGSINH, QQUAN, DOS, VRES, FOR theo mẫu dưới đây. Các vùng tin còn lại sẽ tính sau:

MASV	**HOLOT**	**TEN**	**NGSINH**	**QQUAN**	**DOS**	**VRES**	**FOR**
CK001	Le Van	Hung	12-04-1972	Da Nang	8	9	10
NT001	Ho Thi	Lan	10-05-1969	Hue	7	6	9
NT002	Tran Van	Long	06-12-1968	Da Lat	5	4	5
CK002	Le	Tung	05-06-1967	TP.HCM	7	7	6
KT001	Ng. Thi	Hoa	10-10-1967	TP.HCM	8	4	7
KT002	Le Van	Chau	05-04-1968	Da Nang	7	6	9
CK003	Vo	Anh	02-10-1969	Hue	9	9	10
CK004	Ho Duc	Tuan	10-02-1968	Da Nang	7	6	9
NT003	Vo Thi	Lan	02-01-1969	Hue	8	9	9
NT004	Le Van	Huy	05-06-1968	Da Nang	8	5	2

2. Dùng lệnh COPY FILE để chép tập tin KETQUA1.DBF thành KQ1.DBF. Sau đó có thể dùng lệnh USE KQ1 để mở tập tin KQ1 không? tại sao?

3. Mở tập tin KETQUA1.DBF

a. Tính (điểm trung bình), biết rằng DOS có hệ số hai, VRES có hệ số 1, FOX có hệ số 3.

b. Xếp loại, biết rằng:

DTB>=9 : Xếp loại ‘GIOI’

7<=DTB<9 : Xếp loại ‘KHA’

5<=DTB<7 : Xếp loại 'TB'

DTB<5 : Xếp loại 'YEU'

c. Sắp xếp giảm dần theo DTB và ghi vào tập tin SX_DTB.DBF. Mở tập tin SX_DTB.DBF rồi dùng lệnh BROWSE để xem.

d. Đổi dữ liệu của trường QQUAN thành chữ hoa.

e. Tính trung bình cộng của các môn học cho toàn bộ các mẫu tin, cho từng nhóm có MASV bắt đầu bằng CK, NT, KT.

f. Cho biết số sinh viên có ít nhất hai môn có điểm >=8.

Bài tập chương 5

1. Tạo tập tin NHAPVT.DBF có cấu trúc như sau:

8. **Field Name**	**Field Type**	**Width**	**Dec**	**Phần ghi chú**
MAVT	Charater	5		Mã số vật tư
NGAYNHAP	Date	8		Ngày nhập
MANX	Charater	1		Nhập: N, xuất: X
SL	Numeric	6		Số lượng
DONGIA	Numeric	8		Đơn giá
THANHTIEN	Numeric	9		Thành tiền

Nhập vào 10 mẫu tin theo mẫu dưới đây:

MAVT	**NGAYNHAP**	**MANX**	**SL**	8.1. **DONGIA**
TV01	01-01-1998	N	12	3850000
TL01	04-01-1998	N	10	4700000
ML01	08-01-1998	X	40	5100000
BU01	04-05-1998	N	30	220000
QB01	05-01-1998	N	28	350000
MG01	05-06-1998	X	12	4000000
ND01	06-06-1998	N	20	650000

HD02	10-10-1998	N	12	13000000
HD02	01-01-1998	N	10	16000000
XD01	01-01-1998	X	30	1200000

2. Tập tin TONKHO98.DBF có cấu trúc như sau:

9. **Field Name**	**Field Type**	**Width**	**Dec**	**Phần ghi chú**
MAVT	Charater	5		Mã số vật tư
TONDAU	Numeric	10		Tồn đầu kỳ
SLN	Numeric	10		Số lượng nhập
SLX	Numeric	10		Số lượng xuất
TONCUOI	Numeric	10		Tồn cuối kỳ

Nhập vào các mẫu tin sau:

MAVT	**TONDAU**
TV01	12
TL01	30
ML01	50
BU01	40
QB01	50
MG01	55
ND01	100
HD02	50
HD02	45
XD01	100

3. Tạo tập tin DMVTU.DBF có cấu trúc sau:

10. **Field Name**	**Field Type**	**Width**	**Dec**
MAVT	Charater	5	

TENVT	Charater	20

- Lấy MAVT trong tập tin TONKH98.DBF thay thế vào trường MAVT
- Nhập vào trường TENVT các dữ liệu sau:

MAVT	NGAYNHAP
TV01	Tivi mau SHAP 14
TL01	Tu lanh TOSHIBA 1401
ML01	May lanh 1.5 HP
BU01	Ban ui Philip
QB01	Quat ban Hitachi
MG01	May giat SANYO 40
ND01	Noi com dien SANYO
HD02	Xe cub 86
XD01	Xe dap NHAT

4. Tính giá trị trường THANHTIEN của tập tin NHAPVT.DBF

5. Tính tổng số tiền nhập của mỗi loại vật tư có chữ cái đầu tiên bên trái giống nhau.

6. Tính SLN, SLX, TONCUOI, sau thời gian nhập xuất trên.

7. Tạo tập tin TONKHO99.DBF có cấu trúc giống như TONKHO98.DBF. Lấy TONCUOI của tập tin TONKHO94.DBF để bỏ vào TONDAU của TONKHO99.DBF

8. Liệt kê danh sách Nhập vật tư gồm các mục sau:

MAVT TENVT NGAYNHAP MANX SL

9. Liệt kê danh sách TONKHO gồm các mục sau:

MAVT TENVT TONDAU TONCUOI

11. CHƯƠNG 5: FORMS

11.1. *5.1. KHÁI NIỆM VỀ LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG:*

Thiết kế và lập trình hướng đối tượng là một sự thay đổi đối với phong cách lập trình cũ, lập trình hướng thủ tục.Ở đây thay vì nghĩ đến các chức năng của chương trình ta chỉ cần nghĩ đến các đối tượng đang tạo: là các thành phần độc lập của một ứng dụng cới chức năng riêng của nó. Mỗi một đối tuợng đều có một bộ thuộc tính mô tả đối tượng; các phương thức là những đoạn trình chứa trong điều khiển, cho điều khiển biết cách thức để thực hiện một đoạn công việc nào đó; và tập hợp những sự kiện đó là những phản ứng của đối tượng.

Trong Visual Foxpro, các form và control là các đối tượng được dùng để xây dựng các ứng dụng

5.1.1. Thuộc tính của đối tượng (Properties)

Để chỉ đến một thuộc tính của đối tượng nào ta dùng cú pháp sau:

<tên đối tượng>.<thuộc tính>

Ví dụ: Myform.caption= “Chương trình ứng dụng”

Các thuộc tính thông dụng:

- Left: Vị trí cạnh trái của đối tượng so với vật chứa nó.
- Top: Vị trí trên của đối tượng so với vật chứa nó.
- Height: Chiều cao của đối tượng.
- Width: Chiều rộng của đối tượng.
- Name: Tên để chỉ đối tượng.
- Enable: Giá trị logic:

    True: có quyền làm việc.

    False: Không có quyền làm việc.

- Visible: Giá trị logic:

    True: Thấy được đối tượng;

False: Không thấy được đối tượng.

5.1.2. Phương thức của đối tượng (Methods)

Để gọi đến phương thức của một đối tượng, ta dùng cú pháp:

<tên đối tượng>.<phương thức>

Ví dụ: Myform.show

Một số phương thức thường dùng:

- Refresh: Làm tươi lại đối tượng.
- Show: Hiện đối tượng.
- Hide: ẩn đối tượng.
- Release: Giải phóng đối tượng.
- SetFocus: Thiết lập "tầm ngắm" cho đối tượng.

5.1.3. Sự kiện của đối tượng

Để chỉ đến sự kiện của đối tượng, ta dùng cú pháp sau:

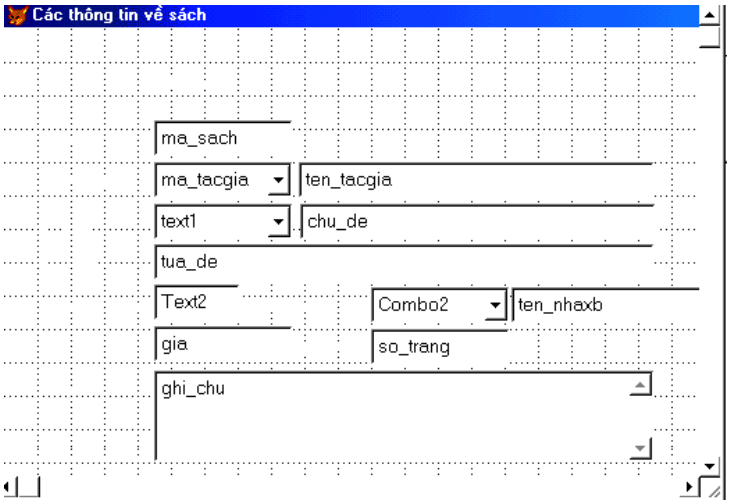
<tên đối tượng>.<sự kiện>

Một số sự kiện thường dùng:

- Click: Được gọi khi kích chuột vào đối tượng.
- DbClick: Được gọi khi kích đúp chuột vào đối tượng.
- MouseMove: Được gọi khi di chuyển chuột trên bề mặt của đối tượng.
- KeyPress: Được gọi khi nhấn một phím kích chuột vào đối tượng.
- Got focus: Được gọi khi đưa đối tượng vào tầm ngắm.
- Lostfocus: Được gọi khi đưa đối tượng ra khỏi tầm ngắm
- Change: Được gọi khi có sự thay đổi nội dung dữ liệu kiểu chuỗi của đối tượng.

11.2. *5.2. FORM*

5.2.1. Giới thiệu



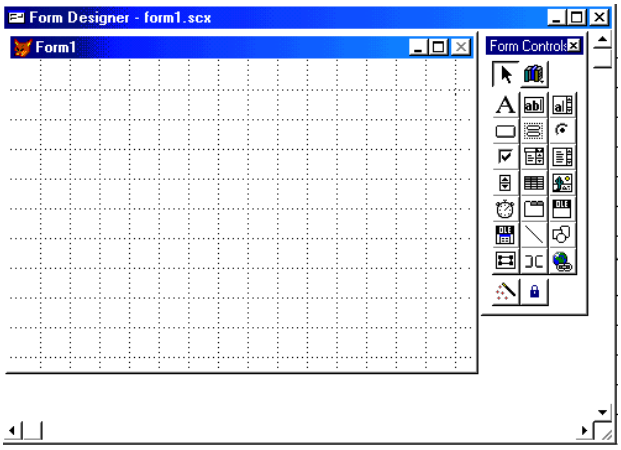
Form được dùng để làm giao diện nhập, hiển thị thông tin, nó cung cấp một tập hợp các đối tượng để đáp lại những thao tác của người dùng làm cho ứng dụng ra dáng chuyên nghiệp.

Ví dụ: Giao diện của một Form nhập dữ liệu

5.2.2. Tạo form thông qua Wizard

Từ menu Tools, chọn wizard, chọn form, xuất hiện giao diện wizard selection, rồi thông qua hướng dẫn.

5.2.3. Tạo form thông qua thiết kế



Để tạo form thông qua thiết kế, từ cửa sổ lệnh ta thực hiện lệnh sau:

CREATE FORM <tờn form>

Khi đó ta được màn hình thiết kế form như sau:

a. Quản lý form

- *Lưu Form*: Từ menu file, chọn save để lưu vào <tên form>, mặc định phần mở rộng là scx.
- *Chạy form*: Từ cửa sổ lệnh, thực hiện lệnh sau:

DO FORM <tên form>

- *Đóng form* (giải phóng khỏi bộ nhớ)

RELEASE <tên form>

b. Tuy cập đến các đối tượng trên form

+ Muốn chỉ đến một đối tượng nào trên form, ta dùng:

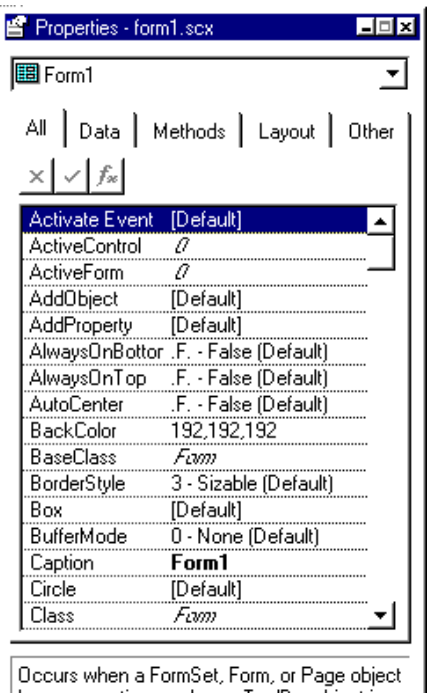
<tên form>. <đối tượng>: nếu <đối tượng> không cùng với form đang thao tác.

<this form>.<đối tượng>: nếu đối tượng nằm trên form đang thao tác.

+ Muốn thay đổi giá trị các thuộc tính trên form, ta dùng:

<tên form>.<thuộc tính>= <giá trị>: nếu muốn thay đổi thuộc tính của form không phải là form hiện hành.

<This form>.<thuộc tính>=<giá trị>: nếu muốn thay đổi các thuộc tính của form hiện hành.



c. Các thuộc tính, phương thức, sự kiện thường dùng trên form.

- *Thuộc tính:*

- BackColor: Màu nền
- BorderStyle: Dạng đường viền
- Caption: Tiêu đề của form
- FillColor: Màu để tô đối tượng
- Fontname: Font chữ cho các đối tượng chứa văn bản
- Fontsize: Kính thước fontname
- Moveable: Cho phép di chuyển hay không

.....

- *Tình huống*

- MoveMouse: Đáp ứng khi di chuyển chuột trên bề mặt form
- Destroy: Đáp ứng khi giải phóng form
- Load: Đáp ứng khi nạp form vào bộ nhớ

5.3. Thanh công cụ Control Toolbar

- Muốn đưa đối tượng trên thanh Control vào form: 4 bước

- Kích chuột vào đối tượng cần đưa

- Vẽ nó trên form để xác định vị trí

- Thiết lập các thuộc tính thích hợp

- Viết mã lệnh cho các tình huống tương ứng

- Quy ước đặt tên cho các đối tượng

Loại đối tượng	Tên
Form	Bắt đầu bởi Frm
Command	Bắt đầu bởi cmd
Edit box	Bắt đầu bởi Edb
Grid	Bắt đầu bởi Grd
Image	Bắt đầu bởi Img
Label	Bắt đầu bởi Lbl
Textbox	Bắt đầu bởi Txt
Timer	Bắt đầu bởi Tmr

5.3.2. Một số đối tượng trên Controls

a. Label : Dùng để thể hiện các chuỗi trên form.

Các thuộc tính thường dùng:

- Caption: Chuỗi thể hiện

- Autosize: Giá trị logic, cho phép kích thước của Label có tự động chỉnh sửa theo độ dài của caption hay không.

b. Command Bottom: Dùng để thể hiện các nút lệnh trên form.

Các thuộc tính thường dùng:

- Caption: tên xuất hiện trên nút lệnh
- Picture: Hình xuất hiện trên nút lệnh
- Enable: giá trị Logic, cho phép chọn nút lệnh hay không

Các Sự kiện thường dùng:

- Click: Khi kích chuột vào nút lệnh thì sự kiện này được gọi.

c. TextBox: Dùng để xem, chỉnh sửa dữ liệu từ các trường trong bảng dữ liệu không phải kiểu memo.

Các thuộc tính thường dùng.

- ControlSource: Tên của trường hay biến mà giá trị của nó được hiện trong textbox
- Value: Giá trị hiện thời của textbox.

Sự kiện thường dùng:

- Change: Khi có sự thay đổi của thuộc tính value
- KeyPress: Khi có phím bất kỳ được ấn.

d. Editbox: Tương tự như textbox, được dùng để chỉnh sửa dữ liệu từ các trường memo.

Các thuộc tính thường dùng.

- Control Source: Tên của trường mà giá trị của nó được thể hiện trong editbox.
- ScrollBars: Có hiện thanh cuộn trong khung editbox hay không.
- ReadOnly: Cho phép có được chỉnh sửa nội dung hay không

Sự kiện thường dùng:

- Change: Khi có sự thay đổi của thuộc tính value.
- Keypress: Khi có phím bất kỳ được ấn.

e. Images: Dùng để đưa các hình ảnh trên form.

Các thuộc tính thường dùng.

- Picture: Xác định file hình ảnh

- Stretch: Xác định cách thức thể hiện hình ảnh (phóng to, thu nhỏ, nguyên mẫu).

f. Timer: Dùng để thiết lập các công việc thực hiện đều đặn sau một khoảng thời gian.

Các thuộc tính thường dùng.

- Enabled: Xác định xem Timer có hiệu lực hay không

- Interval: Quy định khoảng thời gian xác định cho tình huống timer.

Sự kiện thường dùng:

- Timer: Được kích hoạt đều đặn sau một khoảng thời gian xác định ở thuộc tính Interval.

g. Grid: Dùng để thể hiện dữ liệu theo dạng bảng.

Các thuộc tính thường dùng.

- Row Source: Xác định bảng dữ liệu cần thể hiện.

- ColumnCount: Xác định số cột của Grid.

Chú ý:

Nếu Row Source không được chỉ ra thì lấy bảng dữ liệu hiện hành.

Nếu Column count không chỉ ra thì mặc định là tất cả các trường trong bảng dữ liệu (Column count=-1).

Control Source: Được xác định cho từng cột, dùng để khai báo nguồn dữ liệu cho cột đó.

- Allow Addnew: Cho phép thêm các bản ghi mới hay không.

Chú ý:

Muốn thay đổi các thuộc tính trên Grid thì chuyển Grid sang dạng edit bằng cách nhấn phím phải chuột lên Rrid, chọn Properties. Grid đang ở chế độ Edit có một đường viền bao quanh.

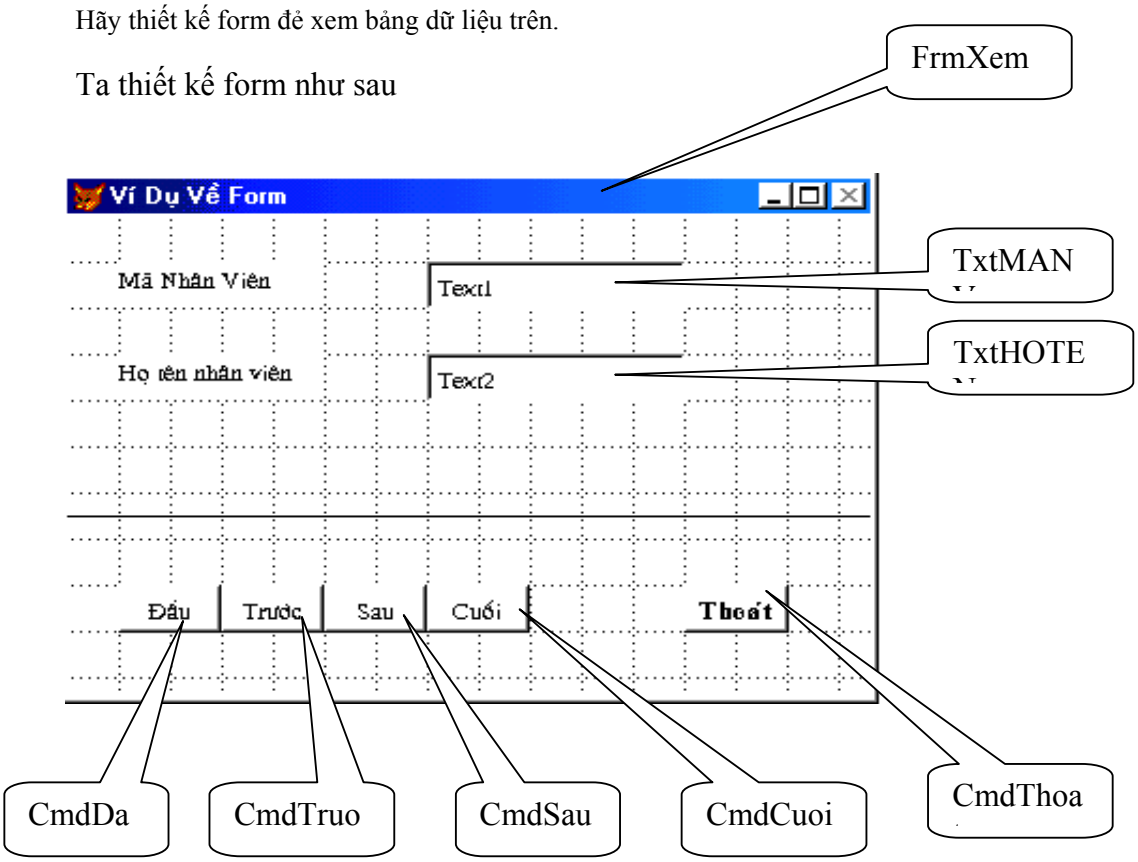
12. **5.4. Ví dụ**

Giả sử có bảng dữ liệu với cấu trúc:

MANV	C	5
HOTEN	C	30

Hãy thiết kế form đẻ xem bảng dữ liệu trên.

Ta thiết kế form như sau



13. **Các** thuộc tính chính :

14. + TxtMANV các thuộc tính ControlSource là MANV

15. + TxtHOTEN các thuộc tính ControlSource là HOTEN

16. Mã lệnh của các đối tượng trên Form là:

+ FrmXem.load

use hoso

+ CmdDau.Click

go top

thisform.refresh

+ CmdCuoi.Click

go bottom

thisform.refresh

+ CmdTruoc.Click

if not bof()

skip -1

endif

thisform.refresh

+ CmdSau.Click

if not eof()

skip

endif

thisform.refresh

+ CmdThoat.Click

use

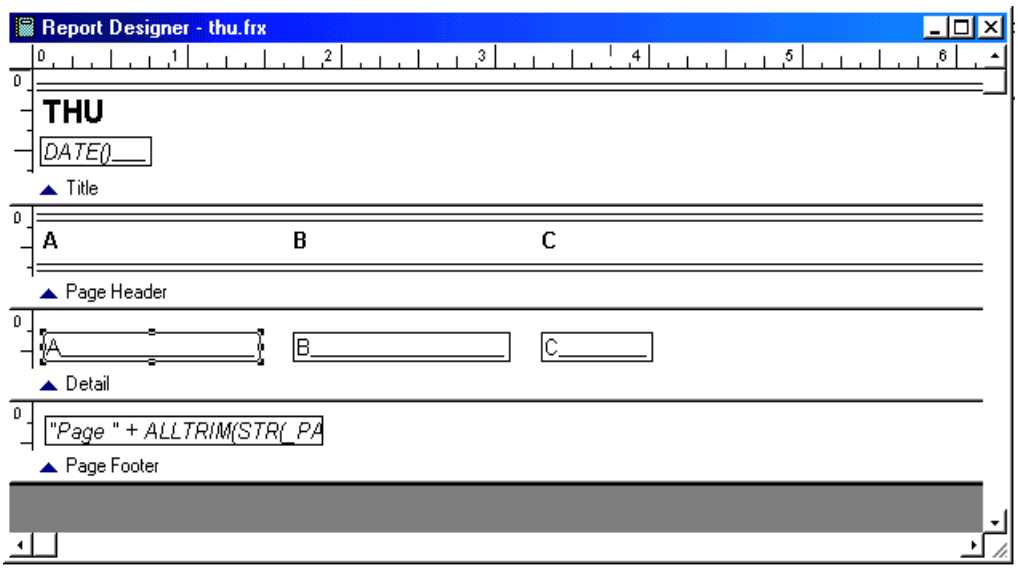
thisform.release

17. CHƯƠNG 6 REPORTS

17.1. *6.1. KHÁI NIỆM*

Reports là công cụ để trình bày và tóm tắt dữ liệu trong một văn bản khi in. Report có hai thành phần cơ bản cấu thành: dữ liệu nguồn, thông thường là các bảng dữ liệu và hình thức trình bày là dạng thức của report sẽ định dạng cách kết xuất dữ liệu.

Màn hình thiết kế Report



17.2. *6.2. CÁC BƯỚC ĐỂ TẠO REPORT*

Ta có thể thiết kế report để thể hiện dữ liệu ở nhiều dạng thức khác nhau trên giấy khi in. Quá trình thiết kế gồm 4 bước chính như sau:

1. Xác định loại Report cần tạo: Tức là quyết định chọn dạng thức mà report hiển thị kết quả.

2. Tạo Report layout: Có thể sử dụng report wizard hay report designer. Report layout được lưu trên đĩa với phần mở rộng của file là FRX: Lưu trử chi tiết của report.

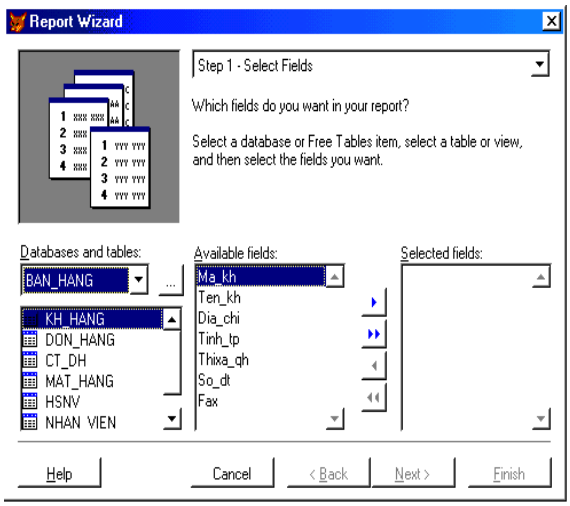
3. Sửa đổi layout của report.

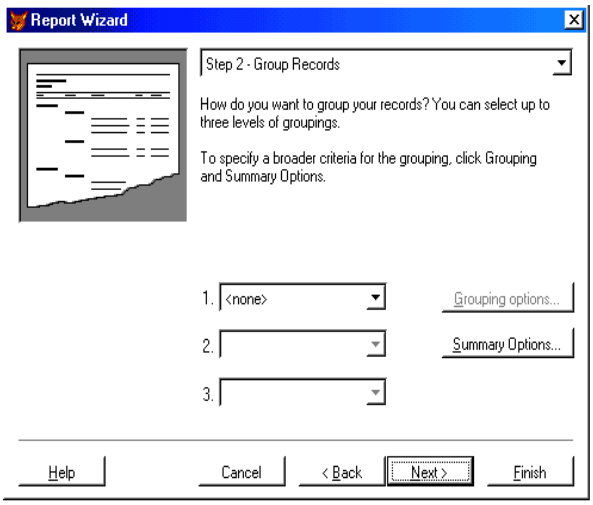
4. Xem và in report.

17.3. *6.3. TẠO REPORT BẰNG WIZARD.*

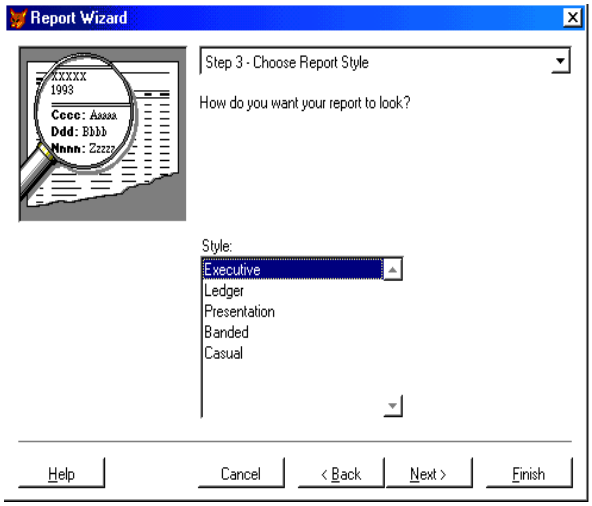
Từ menu Tools, chọn Wizard, chọn Report sau đó làm theo các bước hướng dẫn.

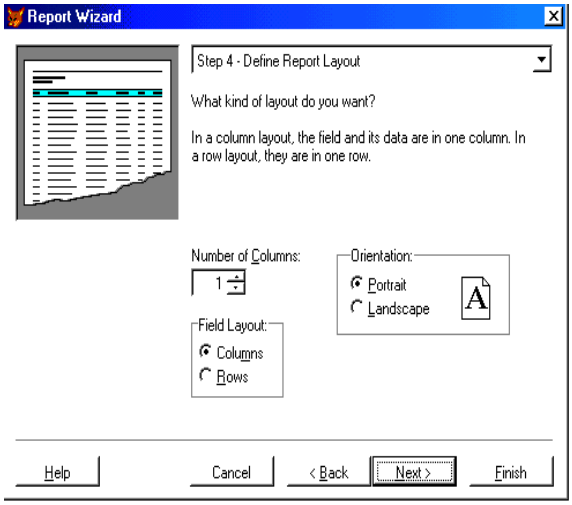
Bước 1: Chọn bảng dữ liệu và các trường cần thể hiện





Bước 2: Tạo nhóm dữ liệu kết xuất

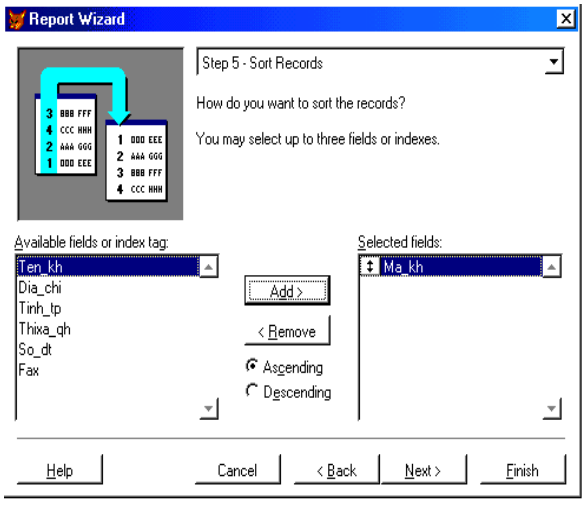


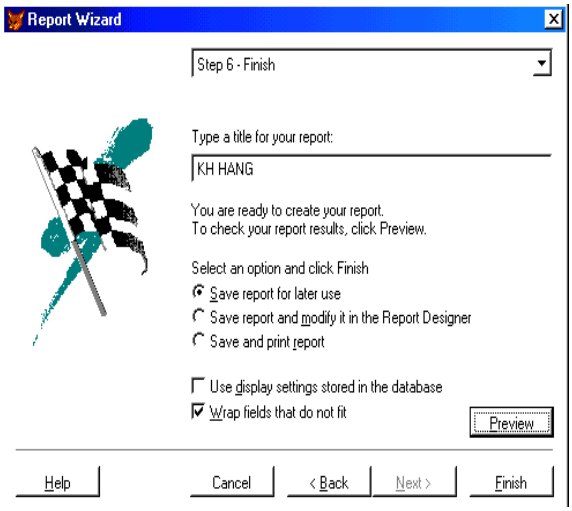


Bước 3: Chọn kiểu Report thể hiện

Bước 4: Chọn cách trình bày trên giấy in

Bước 5: Chọn trường Sắp xếp





Bước 6: Đặt tựa đề, kết thúc

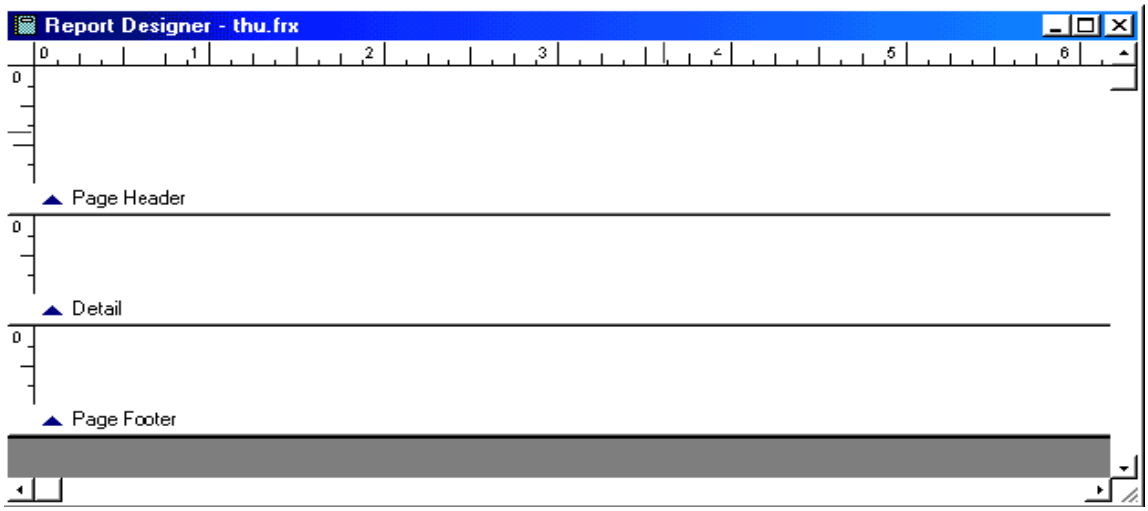
17.4. *6.4. TẠO REPORT BẰNG REPORT DESIGNER*

6.4.1. Quản lý Report

- Tạo mới Report: CREATE REPORT <tên Report>

Ví dụ: create report THU

Lúc này màn hình xuất hiện hộp thoại report

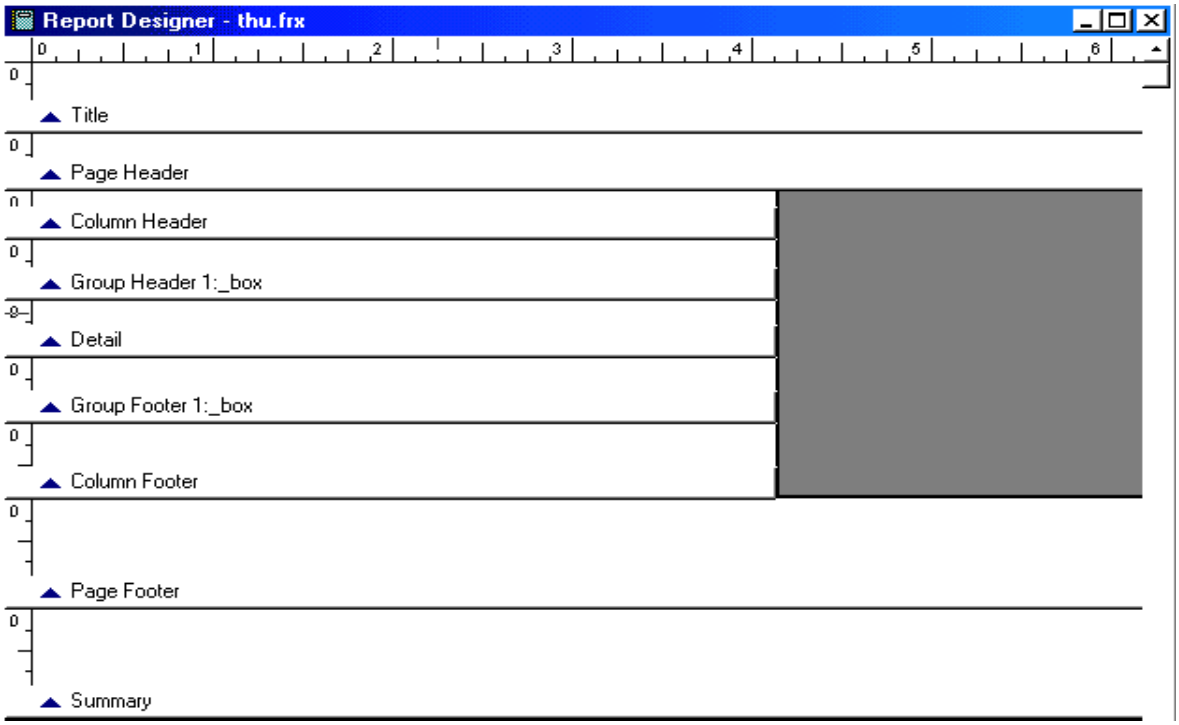


- Mở một report sẵn có: MODIFY <tên report>

- Xem trước khi in: REPORT FORM <tên report> PREVIEW
- Xem trước khi in có điều kiện:

  REPORT FORM <tên report> PREVIEW <điều kiện>
- In report: REPORT FORM <tên report> TO PRINTER

6.4.2. Các thành phần trên Report

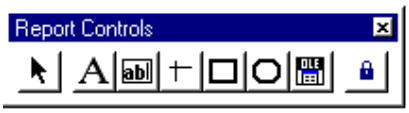


- Title: Dùng để in trên mỗi report: Từ menu report, chọn title summary
- Page Header: Để in trên mỗi header của mỗi trang in.
- Column header: Để in tên header của mỗi cột. Để chọn, từ menu file chọn page setup, chọn giá trị cho column number lớn hơn 1.
- Group header: Xuất hiện mỗi khi bắt đầu nhóm mới. Để chọn, từ menu report chọn data grouping.
- Detail: phần chi tiết trên mỗi record (ứng với từng record trên bảng dữ liệu).
- Group footer: In phần Footer của mỗi nhóm. Để chọn, từ menu report chọn data grouping.

- Column footer: In phần Footer của mỗi cột. Để chọn, từ menu file, chọn page setup, chọn giá trị cho column nimber lớn hơn 1.
- Page Footer: In phần Footer của mỗi trang.
- Sumary: Phần tốm tắt của mỗi report.

6.4.3. Các control trên Report

Thanh công cụ Report Control



Chức năng của các control:

Field trong bàng dữ liệu, biến và các biểuthức toán	Field
Text thuần tuý	Label
Đường kẻ	Line
Hộp và đồng khung	Rectangle
Hình tròn, elip	Rounded Rectangle
Hình ảnh hoặc field General	Picture

6.4.4. Đưa các control vào report

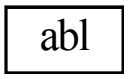
Thực hiện các bước sau:

+ Chọn control thích hợp

+ Kéo rê chuột trên report để xác định vị trí của nó trên report

+ Hiệu chỉnh các control

a. Đưa field vào report:



+ Kích chuột vào

+ Trong hộp report Expression, chọn nút lệnh sau hộp Expression.

+ Trong hộp field, hcọn tên trường hay biến thích hợp.

+ Chọn OK.

b. Đưa label vào report:

+ Chọn 

+ Gỏ nội dung của label

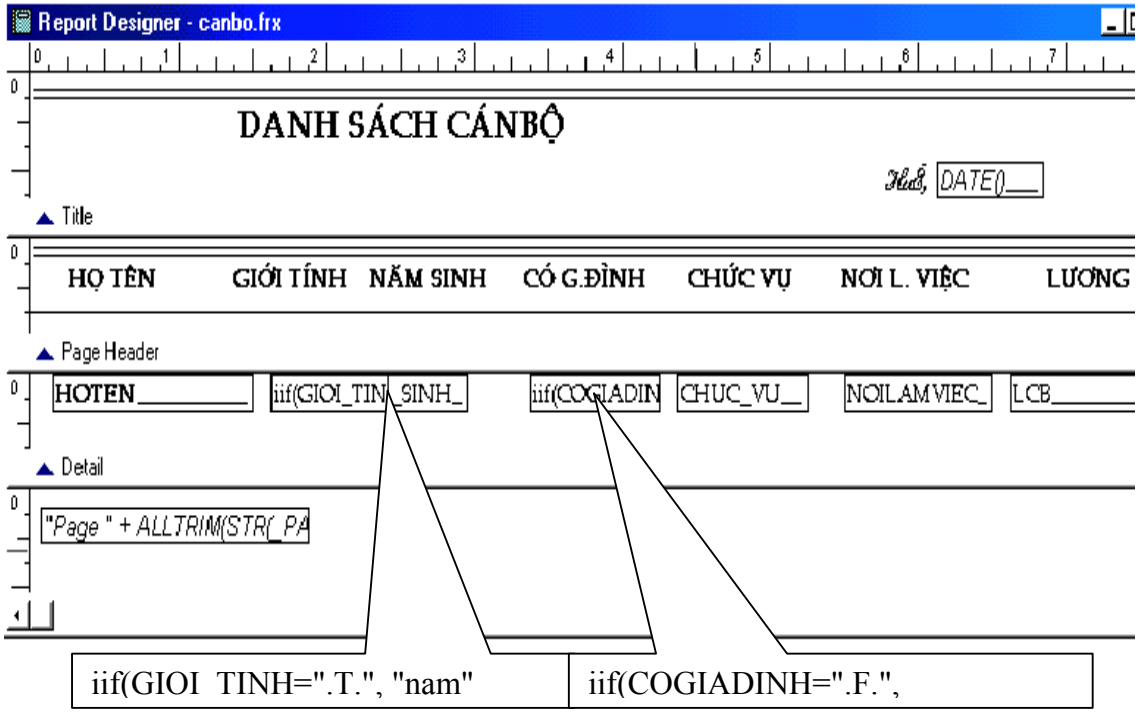
c. Đưa Picture bound control vào report:

+ Chọn picture bound control

+ Xuất hiện hộp hội thoại report picture, chọn file, nếu muốn chèn hình ảnh từ file, chọn field nếu muốn chọn trường General.

+ Chọn Ok

6.5. Ví dụ Thiết kế Report như sau (dựa vào Bảng CANBO.DBF ở bài tập 2):



18. CHƯƠNG 7. TẠO MENU VÀ QUẢN LÝ ĐỀ ÁN

18.1. *7.1. TẠO MENU*

18.2. *7.1.1. GIỚI THIỆU*

Menu cung cấp một phương thức có cấu trúc và giao diện với người dùng để tác động lên những câu lệnh trong ứng dụng.

Việc sắp xếp và thiết kế menu thích hợp sẽ giúp cho người dùng được thuận lợi khi sử dụng hệ thống menu của bạn.

18.3. *7.1.2. CÁC BƯỚC TẠI MỘT MENU HỆ THỐNG*

1. Sắp xếp và thiết kế: Quyết định menu nào bạn cần chúng xuất hiện ở vị trí nào trên màn hình, cần những menu con nào?

2. Sử dụng menu designer, tạo menu và các Submenu.

3. Gắn các câu lệnh tương ứng với công việc.

4. Biên dịch menu

5. Tiến hành chạy thử, kiểm tra.

18.4. *7.1.3. TẠO MENU HỆ THỐNG*

7.1.3.1. Quản lý menu hệ thống

Menu hệ thống được lưu trử tên đĩa với file có phần mở rộng là *.MNX

- Tạo menu bằng công cụ Designer Menu: Thực hiện lệnh:

CREATE MENU <tên menu> ↵

- Mở menu đã có: MODIFY MENU <tên menu>

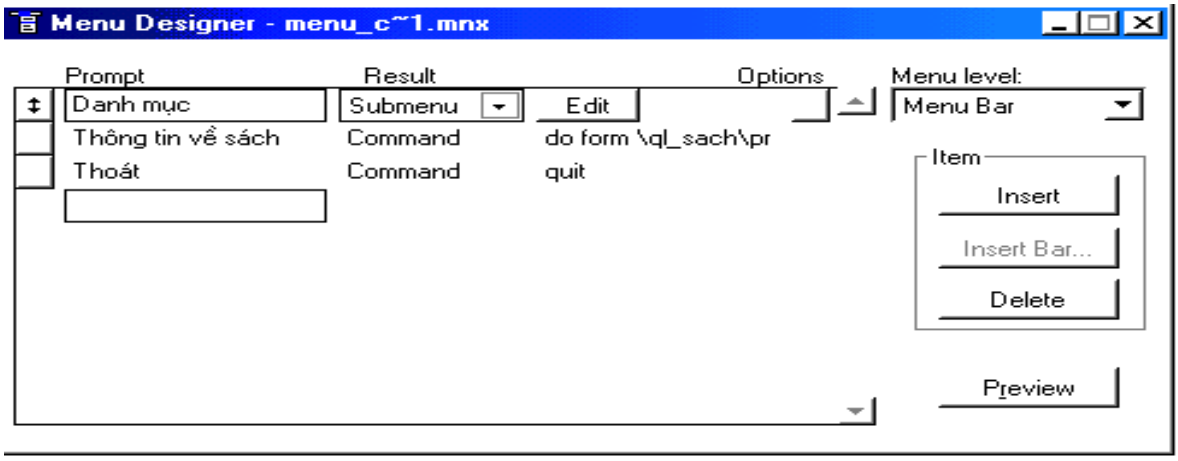
- Dịch file Menu: Để dịch file menu, từ màn hình Menu Designer chọn lệnh Generate.

File menu sau khi dịch sẽ có phần mở rộng là MPR.

7.1.3.2. Tạo menu hệ thống thông qua Menu Designer

Sau khi thực hiện lệnh Create menu, ta được màn hình giao diện Menu: Designer như

sau:



+ Trong hộp Prompt, ta đưa vào tên cần hiển thị trên giao diện.

+ Trong hộp Result, chọn:

- Submenu nếu muốn tạo menu con.

- Procedure nếu muốn thi hành thủ tục

- Command n□u mu□n th□c hi□n m□t l□nh.

+ Kết thúc, ấn Ctrl_W.

18.5. *7.2.QUẢN LÝ ĐỀ ÁN*

18.6. *7.2.1. KHÁI NIỆM ĐỀ ÁN*

Đề án là tên gọi để chỉ đến ứng dụng mà bạn đang xây dựng. Thông thường các thành phần của một đề án bao gồm:

+ Các bảng dữ liệu (table).

+ Các file cơ sở dữ liệu (database)

+ Các form

+ Các report

+ Các query

+ Các file khác như âm thanh, hình ảnh, tài liệu, hình ảnh con trỏ,...

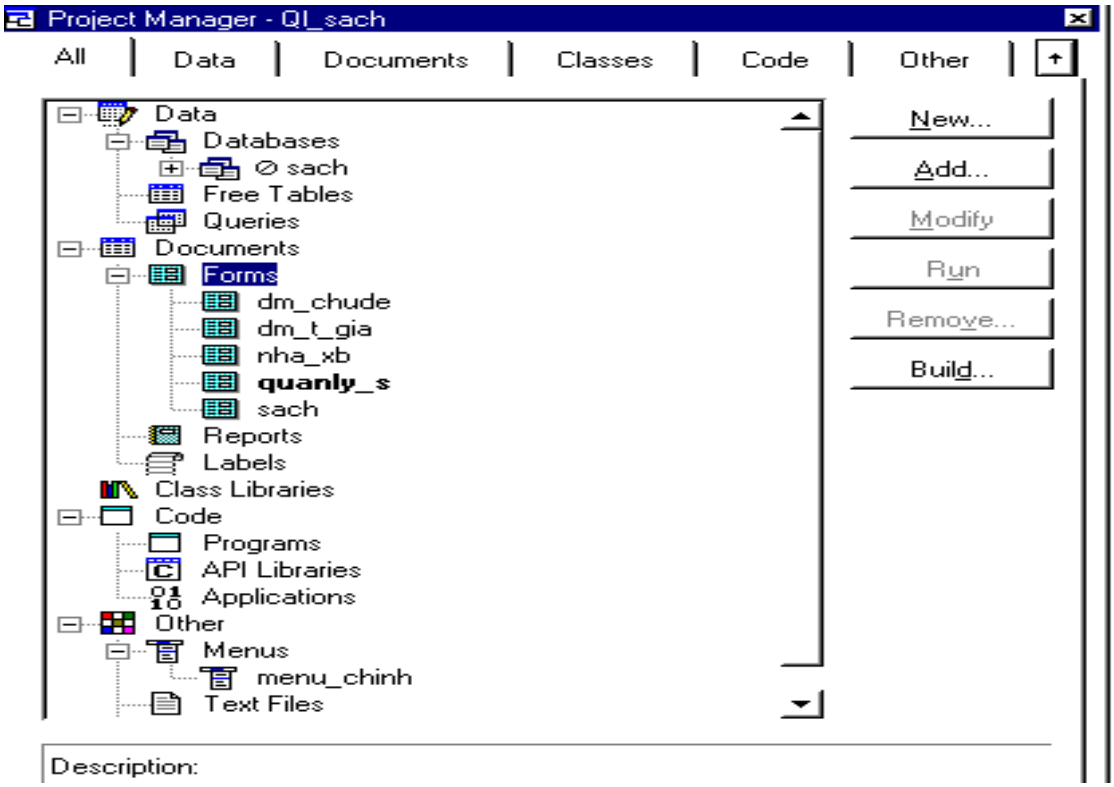
18.7. *7.2.2. QUẢN LÝ ĐỀ ÁN*

Một đề án trong Visual Foxpro được lưu trử trên file có phần mở rộng mặc định là *.PRJ.

7.2.2.1. Tạo mới các đề án

Thực hiện lệnh: CREATE PROJECT <tên đề án>

Lúc này xuất hiện cửa sổ quản lý đề án Project Manager như ở trên.



Database: Bao gồm các:

Table: Các bảng dữ liệu có liên kết với nhau hay các bảng tự do.

Query: Là cấu trúc để lấy thông tin từ các bảng table.

View: Là các Query chuyên dụng mà ta có thể truy xuất dữ liệu cục bộ và từ xa cho phép cập nhật các nguồn dữ liệu bằng cách làm thay đổ Report bởi quyre.

+ Documents: Chứa các tài liệu sử dụng cho đề án; bao gồm các form và report.

+ Class: Liệt kê các thư viện được sử dụng.

+ Code: và những file khác: Liệt kê các file chương trình và các file khác được sử dụng trong chương trình.

Để chỉnh sửa bất kỳ một thành phần nào trong đề án ta chọn nó rồi chọn nút Modify.

Để thêm bất kỳ một file nào cho đề án ta kích nút add (nếu chọn file đã có) hoặc nút new (nếu tạo mới).

Muốn loại bỏ bất kỳ một thành phần nào của đề án ta chọn nó rồi chọn nút remove.

7.2.2.2. Mở một đề án đã có

Thực hiện lệnh: MODIFY PROJECT <tên đề án>

7.2.2.3. Dịch đề án

+ Dịch sang APP: Khi này, để chọn đề án phải có một bản sao của Visual Foxpro.

Dùng lệnh BUILD <tên đề án>

+ Dịch sang file có phần mở rộng là exe: Khi này, người dùng không cần có Visual Foxpro nhưng phải cung cấp hai file: vfp6r.dll và vfp6renu.dll được cài đặt trong đường dẫn hoặc trong cùng thư mục với ứng dụng.

Dùng lệnh: BUILD EXE <tên đề án>

7.2.2.4. Chạy đề án

Sau khi đã dịch, ta có thể chạy đề án thông qua lệnh: DO <tên ứng dụng>

18.8. ***7.2.3. ĐẶT STARTING POINT CHO ĐỀ ÁN***

Khi ứng dụng được thi hành, có một điểm bắt đầu, đó là Starting point.

Để chọn một thành phần của dự án là Starting point:

+ Chọn thành phần được đặt là Starting point.

+ Từ Menu Project, chọn Set main.

Thông thường, Starting point là một chương trình khởi động chứa các thành phần:

Do setup.prg

Do mainmenu.mpr

Read Events

Do cleanup.prg

a. Do Setup.prg: L□ th□c hi□n ch□□ng trỡnh □□ thi□t l□p mụi tr□□ng cho h□ th□ng.

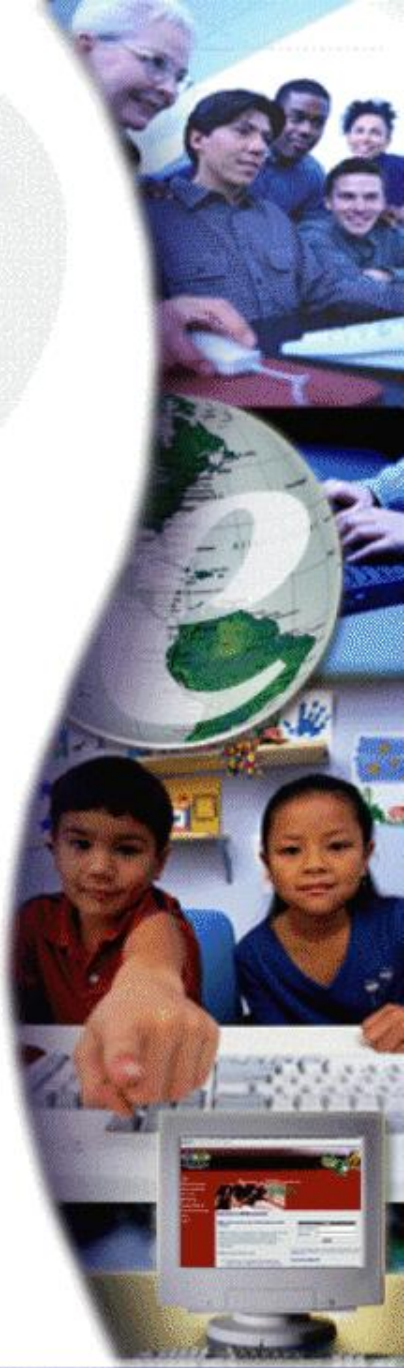
b. Do mainmenu.mpr: Chạy file menu chính để thiết lập giao diện cho hệ thống.

c. Read Events: Bắt đầu thực hiện vòng lặp để thực hiện công việc.

d. Do cleanup.prg: Chạy chương trình dọn dẹp môi trường, trả lại môi trường cho hệ thống và thoát khỏi hệ thống. ổ đây, phải có lệnh Clear Events để thoát khỏi vòng lặp đã được thiết lập bới lệnh Read Events.

ĐẠI HỌC CÔNG NGHỆ THÔNG TIN

Cơ sở dữ liệu



Giảng viên: ThS. Nguyễn Thị Kim Phụng
Email: phungntk@uit.edu.vn

Nội dung

1. Đại số quan hệ
2. Ngôn ngữ truy vấn SQL
3. Ràng buộc toàn vẹn

1. Đại số quan hệ

1. ĐẠI SỐ QUAN HỆ

- Là một mô hình toán học dựa trên lý thuyết tập hợp
- Đối tượng xử lý là các quan hệ trong cơ sở dữ liệu quan hệ
- Cho phép sử dụng các phép toán rút trích dữ liệu từ các quan hệ
- Tối ưu hóa quá trình rút trích dữ liệu
- Gồm có:
  - Các phép toán đại số quan hệ
  - Biểu thức đại số quan hệ

1. ĐSQH - Các phép toán ĐSQH, biểu thức ĐSQH

- Có năm phép toán cơ bản:
  - **Chọn** (σ) Chọn ra các dòng (bộ) trong quan hệ thỏa điều kiện chọn.
  - **Chiếu** (π) Chọn ra một số cột.
  - **Tích Descartes** ($\times$) Kết hai quan hệ lại với nhau.
  - **Trừ** ($-$) Chứa các bộ của quan hệ 1 nhưng không nằm trong quan hệ 2.
  - **Hội** ($\cup$) Chứa các bộ của quan hệ 1 và các bộ của quan hệ 2.
- Các phép toán khác:
  - **Giao ($\cap$), kết ($\bowtie$), chia (/ hay $\div$), đổi tên ($\leftarrow$)**: là các phép toán không cơ bản (được suy từ 5 phép toán trên, trừ phép đổi tên).
- Biểu thức đại số quan hệ:
  - Là một biểu thức gồm các phép toán ĐSQH.
  - Biểu thức ĐSQH được xem như một quan hệ (không có tên)
  - Kết quả thực hiện các phép toán trên cũng là các quan hệ, do đó có thể kết hợp giữa các phép toán này để tạo nên các quan hệ mới!

1. ĐSQH - Phép chọn σ

Câu hỏi 1: Cho biết các nhân viên nam ?

- Biểu diễn cách 1 : **Cú pháp :** $\sigma_{(\text{Điều kiện 1} \wedge \text{điều kiện 2} \wedge \ldots.)}(\text{Quan hệ})$

Câu hỏi 1: $\sigma_{\text{Phai='Nam'}}(\text{NhanVien})$

- Ngoài ra, có thể biểu diễn cách 2:

Cú pháp : (Quan hệ: điều kiện chọn)

Câu hỏi 1: (NhanVien: Phai='Nam')

NHANVIEN			
MANV	HOTEN	NTNS	PHAI
NV001	Nguyễn Tấn Đạt	10/12/1970	Nam
NV002	Trần Đông Anh	01/08/1981	Nữ
NV003	Lý Phước Mẫn	02/04/1969	Nam

Kết quả phép chọn

NHANVIEN			
MANV	HOTEN	NTNS	PHAI
NV001	Nguyễn Tấn Đạt	10/12/1970	Nam
NV003	Lý Phước Mẫn	02/04/1969	Nam

1. ĐSQH - Phép chọn σ

Câu hỏi 2: Cho biết các nhân viên nam sinh sau năm 1975 ?

- Biểu diễn cách 1 :

Câu hỏi 2: $\sigma_{(Phai='Nam' \wedge Year(NTNS)>1975)}(NhanVien)$

- Biểu diễn cách 2:

Câu hỏi 2: **(NhanVien**: Phai='Nam' $\wedge$ Year(NTNS)>1975)

NHANVIEN			
MANV	HOTEN	NTNS	PHAI
NV001	Nguyễn Tấn Đạt	10/12/1970	Nam
NV002	Trần Đông Anh	01/08/1981	Nữ
NV003	Lý Phước Mẫn	02/04/1969	Nam

Kết quả phép chọn

NHANVIEN			
MANV	HOTEN	NTNS	PHAI

(không có bộ nào thỏa)

1. ĐSQH - Phép chiếu π

Câu hỏi 3: Cho biết họ tên nhân viên và giới tính ?

- Biểu diễn cách 1 : **Cú pháp :** $\pi_{\text{Cột1, cột2, cột 3, ...}}$ **(Quan hệ)**

Câu hỏi 3 : $\pi_{\text{HOTEN, PHAI}}$ **(NhanVien)**

- Ngoài ra, có thể biểu diễn cách 2:

Cú pháp : **Quan hệ [cột1,cột2,cột3,…]**

Câu hỏi 3: **NhanVien [HoTen, Phai]**

NHANVIEN			
MANV	HOTEN	NTNS	PHAI
NV001	Nguyễn Tấn Đạt	10/12/1970	Nam
NV002	Trần Đông Anh	01/08/1981	Nữ
NV003	Lý Phước Mẫn	02/04/1969	Nam

Kết quả phép chiếu

NHANVIEN	
HOTEN	PHAI
Nguyễn Tấn Đạt	Nam
Trần Đông Anh	Nữ
Lý Phước Mẫn	Nam

1. ĐSQH - Phép chiếu π

Câu hỏi 4: Cho biết họ tên và ngày tháng năm sinh của các nhân viên nam?

- Biểu diễn cách 1:

Bước 1: $Q \leftarrow \sigma_{(Phai='Nam')}(NhanVien)$

Kết quả phép chọn (**còn gọi là biểu thức ĐSQH**) được đổi tên thành quan hệ Q

Bước 2: $\pi_{HOTEN, NTNS}(Q)$

- Biểu diễn cách 2:

Câu hỏi 4: **(NhanVien: Phai='Nam') [HoTen, NTNS]**

NHANVIEN			
MANV	HOTEN	NTNS	PHAI
NV001	Nguyễn Tấn Đạt	10/12/1970	Nam
NV002	Trần Đông Anh	01/08/1981	Nữ
NV003	Lý Phước Mẫn	02/04/1969	Nam

Kết quả phép chiếu

NHANVIEN	
HOTEN	NTNS
Nguyễn Tấn Đạt	10/12/1970
Lý Phước Mẫn	02/04/1969

1. ĐSQH - Phép tích Descartes ×

Câu hỏi 5: Tính tích Descartes giữa 2 quan hệ nhân viên và phòng ban

Cú pháp : Quan-hệ-1 × Quan-hệ-2 × …Quan-hệ-k

Câu hỏi 5 được viết lại: NHANVIEN × PHONGBAN

NHANVIEN

MANV	HOTEN	NTNS	PHAI	PHONG
NV001	Nguyễn Tấn Đạt	10/12/1970	Nam	NC
NV002	Trần Đông Anh	01/08/1981	Nữ	DH
NV003	Lý Phước Mẫn	02/04/1969	Nam	NC

PHONGBAN

MAPH	TENPH	TRPH
NC	Nghiên cứu	NV001
DH	Điều hành	NV002

NHANVIEN X PHONGBAN

MANV	HOTEN	NTNS	PHAI	PHONG	MAPH	TENPH	TRPH
NV001	Nguyễn Tấn Đạt	10/12/1970	Nam	NC	NC	Nghiên cứu	NV001
NV001	Nguyễn Tấn Đạt	10/12/1970	Nam	NC	DH	Điều hành	NV002
NV002	Trần Đông Anh	01/08/1981	Nữ	DH	NC	Nghiên cứu	NV001
NV002	Trần Đông Anh	01/08/1981	Nữ	DH	DH	Điều hành	NV002
NV003	Lý Phước Mẫn	02/04/1969	Nam	NC	NC	Nghiên cứu	NV001
NV003	Lý Phước Mẫn	02/04/1969	Nam	NC	DH	Điều hành	NV002

1. ĐSQH - Phép kết ⋈ (Theta-Join)

Câu hỏi 6: Cho biết mã nhân viên, họ tên và tên phòng mà n/v trực thuộc.

-**Đặt vấn đề**: trở lại ví dụ 5, ta thấy nếu thực hiện phép tích Decartes NHANVIEN X PHONGBAN thì mỗi nhân viên đều thuộc 2 phòng (vì có tổng cộng là 2 phòng ban, nếu có 3, 4,…phòng ban thì số dòng cho một nhân viên trong NHANVIEN X PHONGBAN sẽ là 3, 4,..dòng.

- Thực tế mỗi nhân viên chỉ thuộc duy nhất 1 phòng ban do ràng buộc khóa ngoại (PHONG), do đó để lấy được giá trị MAPH đúng của mỗi nhân viên ➔ phải có điều kiện chọn:

NHANVIEN.PHONG = PHONGBAN.MAPH

biểu diễn phép chọn theo cách 2

((NHANVIEN X PHONGBAN) : NHANVIEN.PHONG=PHONGBAN.MAPH)

MANV	HOTEN	NTNS	PHAI	PHONG	MAPH	TENPH	TRPH
NV001	Nguyễn Tấn Đạt	10/12/1970	Nam	NC	NC	Nghiên cứu	NV001
NV002	Trần Đông Anh	01/08/1981	Nữ	DH	DH	Điều hành	NV002
NV003	Lý Phước Mẫn	02/04/1969	Nam	NC	NC	Nghiên cứu	NV001

1. ĐSQH - Phép kết ⋈ (Theta-Join)

- Cách 1: $\sigma_{\text{NHANVIEN.PHONG=PHONGBAN.MAPH}}$ (NHANVIEN X PHONGBAN)

- Cách 2:

(NHANVIEN ⨯ PHONGBAN): (NHANVIEN.PHONG=PHONGBAN.MAPH)

* Phép kết được định nghĩa là phép tích Decartes và có điều kiện chọn liên quan đến các thuộc tính giữa 2 quan hệ, cú pháp :

$$\text{Quan-hệ-1} \underset{\text{Điều kiện kết}}{\bowtie} \text{Quan-hệ-2}$$

(Phép kết với đk tổng quát được gọi là θ-kết, θ có thể là $\neq$, =, >, <, >=, <=. Nếu đk kết là phép so sánh = thì gọi là kết bằng)

➡ **Câu hỏi 6 viết lại cách 1:**

$$\pi_{\text{MANV,HOTEN,TENPH}} (\text{NHANVIEN} \underset{\text{PHONG=MAPH}}{\bowtie} \text{PHONGBAN})$$

➡ **Câu hỏi 6 viết lại cách 2:**

$$(\text{NHANVIEN} \underset{\text{PHONG=MAPH}}{\bowtie} \text{PHONGBAN})\ [\text{MANV,HOTEN,TENPH}]$$

1. ĐSQH - kết bằng, kết tự nhiên

Kết bằng:

Kết tự nhiên:

Nếu PHONG trong NHANVIEN được đổi thành MAPH thì ta bỏ đi 1 cột MAPH thay vì phải để MAPH=MAPH, lúc này gọi là phép kết tự nhiên (natural-join)

Hoặc viết cách khác: NHANVIEN * PHONGBAN

1. ĐSQH - Phép kết ⋈

Câu hỏi 7: Tìm họ tên các trưởng phòng của từng phòng ?

$$\pi_{HOTEN,\ TENPH}\ (PHONGBAN \underset{TRPH=MANV}{\bowtie} NHANVIEN)$$

Câu hỏi 8: Cho lược đồ CSDL như sau:

TAIXE (MaTX, HoTen, NgaySinh, GioiTinh, DiaChi)

CHUYENDI (SoCD, MaXe, MaTX, NgayDi, NgayVe, ChieuDai, SoNguoi)

Cho biết họ tên tài xế, ngày đi, ngày về của những chuyến đi có chiều dài >=300km, chở từ12 người trở lên trong mỗi chuyến?

Cách 1: $Q \leftarrow \underset{(ChieuDai>=300\ \wedge\ SoNguoi>=12)}{\sigma}(CHUYENDI)$

Kết quả: $\pi_{HoTen,\ NgayDi,\ NgayVe}\ (Q \underset{MATX}{\bowtie} TAIXE)$

Cách 2: $((CHUYENDI : ChieuDai>=300 \wedge SoNguoi>=12) \underset{MATX}{\bowtie} TAIXE)\ [HoTen, NgayDi, NgayVe]$

1. ĐSQH - Phép kết ngoài (outer join)

- Mở rộng phép kết để tránh mất thông tin
- Thực hiện phép kết và sau đó thêm vào kết quả của phép kết các bộ của quan hệ mà không phù hợp với các bộ trong quan hệ kia.
- Có 3 loại:
  - Left outer join R ⟕ S
  - Right outer join R ⟖ S
  - Full outer join R ⟗ S
- **Ví dụ**: In ra danh sách tất cả tài xế và số chuyến đi, mã xe mà tài xế đó lái (nếu có)

1. ĐSQH – left outer join (lấy hết tất cả bộ của quan hệ bên trái)

- TAIXE $\overset{matx}{\bowtie}$ CHUYENDI

Matx	Hoten	SoCD	Matx	Maxe
TX01	Huynh Trong Tao	CD01	TX01	8659
TX01	Huynh Trong Tao	CD03	TX01	8659
TX02	Nguyen Sang	CD02	TX02	7715
TX03	Le Phuoc Long	CD04	TX03	4573
TX04	Nguyen Anh Tuan	Null	Null	Null

Bộ của quan hệ TAIXE được thêm
Vào dù không phù hợp với kết quả
của quan hệ CHUYENDI

TAIXE	
MaTX	Hoten
TX01	Huynh Trong Tao
TX02	Nguyen Sang
TX03	Le Phuoc Long
TX04	Nguyen Anh Tuan

CHUYENDI		
SoCD	MaTX	MaXe
CD01	TX01	8659
CD02	TX02	7715
CD03	TX01	8659
CD04	TX03	4573

Tương tự right outer join và full outer join (lấy cả 2)

1. ĐSQH - Phép trừ, phép hội, phép giao tập hợp

- Tất cả các phép toán này đều cần hai quan hệ đầu vào tương thích khả hợp, nghĩa là chúng phải thoả:
  - Cùng số thuộc tính. Ví dụ: R và S đều có 2 thuộc tính.
  - Các thuộc tính `tương ứng ' có cùng kiểu.

R	
HONV	**TENNV**
Vuong	Quyen
Nguyen	Tung

S	
HONV	**TENNV**
Le	Nhan
Vuong	Quyen
Bui	Vu

Phép trừ: R — S

Phép hội: R ∪ S

Phép giao: R ∩ S

NHANVIEN (MaNV, HoTen, Phai, Luong,NTNS, Ma_NQL, MaPH)

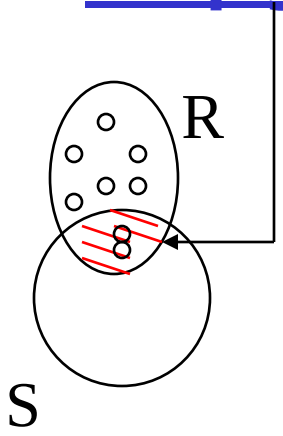
PHANCONG (MaNV, MaDA, ThoiGian)

1. ĐSQH - Phép trừ, phép hội, phép giao tập hợp

Phép trừ: $Q = R - S = \{ t/ t \in R \wedge t \notin S\}$

Phép hội: $Q = R \cup S = \{ t/ t \in R \vee t \in S\}$

Phép giao: $Q = R \cap S = R - (R - S) = \{ t/t \in R \wedge t \in S\}$



R	
HONV	TENNV
Vuong	Quyen
Nguyen	Tung

S	
HONV	TENNV
Le	Nhan
Vuong	Quyen
Bui	Vu

Kết quả phép trừ Q ={Nguyen Tung}

Kết quả phép hội Q ={Vuong Quyen, Nguyen Tung, Le Nhan, Bui Vu}

Kết quả phép giao Q ={Vuong Quyen}

***Lưu ý*: Phép hội và phép giao có tính chất giao hoán**

1. ĐSQH - Phép trừ, phép hội, phép giao tập hợp

Câu hỏi 9: Cho biết nhân viên không làm việc ? **(Phép trừ)**

Cách 1: $\pi_{MANV}(NHANVIEN) - \pi_{MANV}(PHANCONG)$

Cách 2: $(NHANVIEN[MANV]) - (PHANCONG[MANV])$

Câu hỏi 10: Cho biết nhân viên được phân công tham gia đề án có mã số 'TH01' hoặc đề án có mã số 'TH02'? **(Phép hội)**

$((PHANCONG: MADA='TH01')[MANV]) \cup ((PHANCONG : MADA='TH02')[MANV])$

Câu hỏi 11: Cho biết nhân viên được phân công tham gia cả 2 đề án 'TH01' và đề án 'TH02'? **(Phép giao)**

$((PHANCONG : MADA='TH01')[MANV]) \cap ((PHANCONG : MADA='TH02')[MANV])$

1. ĐSQH - Phép chia tập hợp (/ hay ÷)

- Phép chia (R ÷ S) cần hai quan hệ đầu vào R, S thoả:
  - Tập thuộc tính của R là tập cha của tập thuộc tính S.
    Ví dụ: R có m thuộc tính, S có n thuộc tính : n $\subseteq$ m

◆**Định nghĩa**:

R và S là hai quan hệ, R^+ và S^+ lần lượt là tập thuộc tính của R và S. Điều kiện $S^+ \neq \emptyset$ là ***tập con không bằng*** của R^+. Q là kết quả phép chia giữa R và S, $Q^+ = R^+ - S^+$

$$Q = R \div S = \{t / \forall s \in S, \ (t,s) \in R\}$$

$$T_1 \leftarrow \pi_{R^+ - S^+}(R)$$
$$T_2 \leftarrow \pi_{R^+ - S^+}((S \times T_1) - R)$$
$$T \leftarrow T_1 - T_2$$

1. ĐSQH - Phép chia tập hợp (/ hay ÷)

R=PHANCONG

MANV	MADA
001	TH001
001	TH002
002	TH001
002	TH002
002	DT001
003	TH001

S=DEAN

MADA
TH001
TH002
DT001

Kết quả Q

Q= PHANCONG/DEAN

MANV
002

Cho biết nhân viên làm việc cho tất cả các đề án ? (được phân công tham gia tất cả các đề án)

Hoặc viết Q= PHANCONG ÷ DEAN

1. ĐSQH - Phép chia tập hợp (/ hay ÷)

R=KETQUATHI		
Mahv	**Mamh**	**Diem**
HV01	CSDL	7.0
HV02	CSDL	8.5
HV01	CTRR	8.5
HV03	CTRR	9.0
HV01	THDC	7.0
HV02	THDC	5.0
HV03	THDC	7.5
HV03	CSDL	6.0

S=MONHOC	
Mamh	**Tenmh**
CSDL	Co so du lieu
CTRR	Cau truc roi rac
THDC	Tin hoc dai cuong

Mahv
HV01
HV03

Q=KETQUA/MONHOC

$$KETQUA \leftarrow KETQUATHI\,[Mahv, Mamh]$$

$$MONHOC \leftarrow MONHOC[Mamh]$$

*** Viết cách khác**

KETQUATHI[Mahv,Mamh] /MONHOC[Mamh]

1. ĐSQH – Hàm tính toán trên 1 nhóm và tính toán trên nhiều nhóm (gom nhóm – group by)

- Các hàm tính toán gồm 5 hàm: avg(giá-trị), min(giá-trị), max(giá-trị), sum(giá-trị), count(giá-trị).

- Phép toán gom nhóm: (Group by)

$$_{G_1,G_2,\ldots,G_n}\Im_{F_1(A_1),F_2(A_2),\ldots,F_n(A_n)}(E)$$

  - E là biểu thức đại số quan hệ
  - G_i là thuộc tính gom nhóm (nếu không có G_i nào=> không chia nhóm (1 nhóm), ngược lại (nhiều nhóm) => hàm F sẽ tính toán trên từng nhóm nhỏ được chia bởi tập thuộc tính này)
  - F_i là hàm tính toán
  - A_i là tên thuộc tính

1. ĐSQH – Hàm tính toán trên 1 nhóm và tính toán trên nhiều nhóm (gom nhóm – group by)

- Điểm thi cao nhất, thấp nhất, trung bình của môn CSDL ?

$$\Im_{\max(Diem),\min(Diem),agv(Diem)}\sigma_{Mamh='CSDL'}(KETQUATHI)$$

- Điểm thi cao nhất, thấp nhất, trung bình của từng môn ?

$$Mamh\ \Im_{\max(Diem),\min(Diem),avg(Diem)}(KETQUATHI)$$

2. Ngôn ngữ truy vấn SQL

2. NGÔN NGỮ TRUY VẤN SQL

- Là ngôn ngữ chuẩn, có cấu trúc dùng để truy vấn và thao tác trên CSDL quan hệ.
- Câu truy vấn tổng quát:

SELECT [DISTINCT] danh_sách_cột | hàm

FROM danh sách các quan hệ (hay bảng, table)

[WHERE điều_kiện]

[GROUP BY danh_sách_cột_gom_nhóm]

[HAVING điều_kiện_trên_nhóm]

[ORDER BY cột1 ASC | DESC, cột2 ASC | DESC,…]

2. SQL

- Toán tử so sánh:
  - =,>,<,>=,<=,<>
  - BETWEEN
  - IS NULL, IS NOT NULL
  - LIKE (%,_)
  - IN, NOT IN
  - EXISTS, NOT EXISTS
  - SOME, ALL, ANY
- Toán tử logic: AND, OR.
- Các phép toán: +, - ,* , /
- Các hàm xử lý ngày (DAY( )), tháng (MONTH( )), năm (YEAR( ))

2. SQL

- 5 hàm: COUNT( ), SUM( ), MAX( ), MIN( ), AVG( )
- Phân loại câu SELECT: SELECT đơn giản, SELECT có mệnh đề ORDER BY, SELECT lồng (câu SELECT lồng câu SELECT khác), SELECT gom nhóm (GROUP BY), SELECT gom nhóm (GROUP BY)có điều kiện HAVING.

Bài tập: Cho lược đồ CSDL "quản lý đề án công ty" như sau

NHANVIEN (MaNV, HoTen, Phai, Luong,NTNS, Ma_NQL, MaPH)

PHONGBAN (MaPH, TenPH, TRPH)

DEAN (MaDA, TenDA, Phong, NamThucHien)

PHANCONG (MaNV, MaDA, ThoiGian)

NHANVIEN

MANV	HOTEN	NTNS	PHAI	MA_NQL	MaPH	LUONG
001	Vuong Ngoc Quyen	22/10/1957	Nu		QL	3.000.000
002	Nguyen Thanh Tung	09/01/1955	Nam	001	NC	2.500.000
003	Le Thi Nhan	18/12/1960	Nu	001	DH	2.500.000
004	Dinh Ba Tien	09/01/1968	Nam	002	NC	2.200.000
005	Bui Thuy Vu	19/07/1972	Nam	003	DH	2.200.000
006	Nguyen Manh Hung	15/09/1973	Nam	002	NC	2.000.000
007	Tran Thanh Tam	31/07/1975	Nu	002	NC	2.200.000
008	Tran Hong Minh	04/07/1976	Nu	004	NC	1.800.000

DEAN

MADA	TENDA	PHONG	NamThucHien
TH001	Tin hoc hoa 1	NC	2002
TH002	Tin hoc hoa 2	NC	2003
DT001	Dao tao 1	DH	2004
DT002	Dao tao 2	DH	2004

PHONGBAN

MAPH	TENPH	TRPH
QL	Quan Ly	001
DH	Dieu Hanh	003
NC	Nghien Cuu	002

PHANCONG

MANV	MADA	THOIGIAN
001	TH001	30,0
001	TH002	12,5
002	TH001	10,0
002	TH002	10,0
002	DT001	10,0
002	DT002	10,0
003	TH001	37,5
004	DT001	22,5
004	DT002	10,0
006	DT001	30,5
007	TH001	20,0
007	TH002	10,0
008	DT002	12,5

2. SQL – BETWEEN, ORDER BY, IS NULL

Câu hỏi 13: Sử dụng =,>,>=,… Danh sách các nhân viên sinh trong khoảng từ năm 1978 đến 1983?

➡ Select MaNV, HoTen From NhanVien
where Year(NTNS)>=1978 AND Year(NTNS)<=1983

Câu hỏi 14: Sử dụng BETWEEN, ORDER BY. Danh sách các nhân viên sinh trong khoảng từ năm 1978 đến 1983? Sắp xếp theo mức lương giảm dần.

➡ Select * From NhanVien where Year(NTNS) BETWEEN 1978 and 1983 ORDER BY Luong DESC

Câu hỏi 15: Sử dụng IS NULL. Cho biết những nhân viên không có người quản lý trực tiếp? (không chịu sự quản lý trực tiếp của người nào)

➡ Select MaNV, HoTen, NTNS, Ma_NQL from NhanVien where Ma_NQL is Null

2. SQL - SO SÁNH IN & NOT IN

Câu hỏi 16: Sử dụng Is Not Null. Cho biết những nhân viên có người quản lý trực tiếp?Thông tin hiển thị gồm: mã nhân viên, họ tên, mã người quản lý.

➡ Select MaNV, HoTen, Ma_NQL from NhanVien
where Ma_NQL is not Null

Câu hỏi 17: Sử dụng IN (so sánh với một tập hợp giá trị cụ thể). Cho biết họ tên nhân viên thuộc phòng ‘NC’ hoặc phòng ‘DH’?

➡ Select DISTINCT Hoten From NhanVien where MaPH in (‘NC’,’DH’)

Câu hỏi 18: Sử dụng IN (so sánh với một tập hợp giá trị chọn từ câu SELECT khác). Cho biết họ tên nhân viên thuộc phòng ‘NC’ hoặc phòng ‘DH’?

➡ Select Hoten from NhanVien where MaPH in (Select MaPH from PHONGBAN where MaPH=‘NC’ OR MaPH=‘DH’)

2. SQL – SO SÁNH IN & NOT IN

Câu hỏi 19 (tt): Cho biết mã số, họ tên, ngày tháng năm sinh của những nhân viên đã tham gia đề án?

➡ Select MaNV, HoTen, NTNS from NhanVien
where MaNV in (Select MaNv From PhanCong)

Câu hỏi 20: Sử dụng NOT IN. Cho biết mã số, họ tên, ngày tháng năm sinh của những nhân viên không tham gia đề án nào?
Gợi ý cho mệnh đề NOT IN: thực hiện câu truy vấn "tìm nhân viên có tham gia đề án (dựa vào bảng PhanCong)", sau đó lấy phần bù.

➡ Select MaNV, HoTen, NTNS from NhanVien
where MaNV not in (Select MaNv From PhanCong)

Câu hỏi 21 (tt): Cho biết tên phòng ban không chủ trì các đề án triển khai năm 2005? *Gợi ý*: thực hiện câu truy vấn "tìm phòng ban chủ trì các đề án triển khai năm 2005", sau đó lấy phần bù.

➡ Select TenPH from PhongBan where MaPH not in (Select DISTINCT Phong from DEAN where NamThucHien=2005)

2. SQL – SO SÁNH LIKE

Câu hỏi 22: so sánh chuỗi = chuỗi. Liệt kê mã nhân viên, ngày tháng năm sinh, mức lương của nhân viên có tên "Nguyễn Tường Linh"?

```
Select MaNV, NTNS, Luong from NhanVien
where HoTen = 'Nguyễn Tường Linh'
```

Câu hỏi 23: Sử dụng LIKE (%: thay thế 1 chuỗi ký tự). Tìm những nhân viên có họ Nguyễn.

```
Select MaNV, HoTen from NhanVien where HoTen like 'Nguyễn %'
```

Câu hỏi 24 (tt): Tìm những nhân viên có tên Lan.

```
Select MaNV, HoTen from NhanVien where HoTen like '% Lan'
```

Câu hỏi 25 (tt): Tìm những nhân viên có tên lót là "Văn".

```
Select MaNV, HoTen from NhanVien where HoTen like '% Văn %'
```

Câu hỏi 26: Sử dụng LIKE (_: thay thế 1 ký tự bất kỳ). Tìm những nhân viên tên có tên 'Nguyễn La_' (ví dụ Lam, Lan)

```
Select MaNV, HoTen from NhanVien where HoTen like 'Nguyễn La_'
```

2. SQL – HÀM COUNT,SUM,MAX,MIN,AVG

a) Sử dụng các hàm COUNT, SUM, MIN, MAX, AVG trên 1 nhóm lớn (trên toàn bộ quan hệ):

– Câu hỏi 27: Tính số nhân viên của công ty.

```
Select COUNT(MaNV) as SoNV from NhanVien
```

– Câu hỏi 28: Tính số lượng nhân viên quản lý trực tiếp nhân viên khác.

```
Select COUNT (DISTINCT Ma_NQL) from NhanVien
```

– Câu hỏi 29: Tìm mức lương lớn nhất, mức lương trung bình, tổng lương của công ty.

```
Select MAX(Luong), AVG(Luong), SUM(Luong) from NhanVien
```

– Câu hỏi 30: Cho biết nhân viên có mức lương lớn nhất.

```
Select HoTen from NhanVien
Where Luong = (Select MAX(Luong) from NhanVien )
```

2. SQL – MỆNH ĐỀ GROUP BY

Câu hỏi 31: Cho biết nhân viên có mức lương trên mức lương trung bình của công ty.

```
Select HoTen from NhanVien where Luong > (Select
AVG(Luong) from NhanVien )
```

b) Sử dụng các hàm COUNT, SUM, MIN, MAX, AVG trên từng nhóm nhỏ: mệnh đề GROUP BY

- Chia các dòng thành các nhóm nhỏ dựa trên tập thuộc tính chia nhóm.
- Thực hiện các phép toán trên nhóm như: Count (thực hiện phép đếm), Sum (tính tổng), Min(lấy giá trị nhỏ nhất), Max(lấy giá trị lớn nhất), AVG (lấy giá trị trung bình).

2. SQL – MỆNH ĐỀ GROUP BY

Quan hệ NV

Q	S
a	10
a	2
b	9
b	5
c	10
c	8
c	6
c	4
c	10
d	16
d	18
d	50

nhóm

Chia các dòng thành các nhóm dựa trên tập thuộc tính chia nhóm

Q	Count(S)
a	2
b	2
c	5
d	3

Tương tự cho các hàm SUM, MIN, MAX, AVG

Các thuộc tính GROUP BY: Q

Câu SQL:

```
Select Q, count(S)
From NV
Group by Q
```

2. SQL – MỆNH ĐỀ GROUP BY

Câu hỏi 32: Cho biết số lượng nhân viên theo từng phái?

Do cột phái có 2 giá trị “nam” và “nữ”, trường hợp này ta chia bảng NhanVien thành 2 nhóm nhỏ. Thuộc tính chia nhóm là thuộc tính “Phai”.

➡ Select Phai, count(Manv) as SoNV from NhanVien Group by Phai

Câu hỏi 33: Cho biết số lượng nhân viên theo từng phòng?

Do cột MaPH có 3 giá trị “NC” và “DH” và “QL”, trường hợp này ta chia bảng nhân viên thành 3 nhóm nhỏ. Thuộc tính chia nhóm là thuộc tính “MaPH”.

➡ Select MaPH, count(Manv) from NhanVien Group by MaPH

Tương tự: cho biết tổng lương của mỗi phòng, cho biết mức lương thấp nhất của từng phòng, mức lương cao nhất, mức lương trung bình của từng phòng

2. SQL – MỆNH ĐỀ GROUP BY

Câu hỏi 34: Cho biết tên phòng và số lượng nhân viên theo từng phòng?

Giống câu 29 nhưng bổ sung thêm bảng PhongBan để lấy tên phòng. Thuộc tính chia nhóm là (TenPH) thay cho MaPH.

➡ Select TenPH, count(Manv) as SoLuongNV
From NhanVien n, PhongBan p Where n.MaPh=p.MaPH
Group by TenPH

Câu hỏi 35: Với mỗi phòng, cho biết số lượng nhân viên theo từng phái?

Do cột MaPH có 3 giá trị "NC" và "DH" và "QL", mỗi phòng chia nhỏ theo từng phái: 2 nhóm "Nam" và "Nữ", trường hợp này ta chia bảng nhân viên thành 6 nhóm nhỏ. Như vậy, tập thuộc tính chia nhóm cho câu truy vấn là (Phong, Phai).

➡ Select MaPH, Phai, count(Manv) from NhanVien
Group by Phong, Phai

2. SQL – MỆNH ĐỀ GROUP BY

Câu hỏi 36: Đếm số đề án của từng nhân viên tham gia?

- Do cột MaNV có 7 giá trị "NV001",…"NV008" (không có nhân viên "005"), trường hợp này ta chia bảng PhanCong thành 7 nhóm nhỏ. Với mỗi nhóm nhỏ (MaNV), ta đếm số đề án (count(MADA)) tham gia. Thuộc tính chia nhóm là thuộc tính "MaNV".
- Tương tự: tính tổng số giờ làm việc của mỗi nhân viên (SUM), thời gian làm việc thấp nhất của mỗi nhân viên (MIN), thời gian làm việc lớn nhất của mỗi nhân viên (MAX), thời gian làm việc trung bình,…

→ Select MaNV, count(MaDA) as SoDATG From PhanCong
Group by MaNV

Câu hỏi 37: Cho biết mã, tên nhân viên và số đề án mà n/v đã tham gia?

→ Select n.MaNV, HoTen, count(MaDA) as SoDATG
From PhanCong pc, NhanVien n where pc.manv=n.manv
Group by MaNV, HoTen

2. SQL – MỆNH ĐỀ HAVING

- Lọc kết quả theo điều kiện, sau khi đã gom nhóm
- Điều kiện của HAVING là điều kiện về các hàm tính toán trên nhóm (Count, Sum, Min, Max, AVG) và các thuộc tính trong danh sách GROUP BY.

Câu hỏi 38: Cho biết những nhân viên tham gia từ 2 đề án trở lên?

```
Select MaNV, count(MaDA)  as SoDATG From  PhanCong
Group by MaNV
Having count(MaDA) >=2
```

Câu hỏi 39: Cho biết mã phòng ban có trên 4 nhân viên?

```
Select MaPH, count(Manv) from NhanVien Group by MaPH
Having count(Manv)>4
```

3. Ràng buộc toàn vẹn

3. RÀNG BUỘC TOÀN VẸN

- RBTV có bối cảnh trên một quan hệ
  - Ràng buộc miền giá trị
  - Ràng buộc liên bộ
  - Ràng buộc liên thuộc tính
- RBTV có bối cảnh trên nhiều quan hệ
  - Ràng buộc liên thuộc tính liên quan hệ
  - Ràng buộc khóa ngoại (tham chiếu)
  - Ràng buộc liên bộ liên quan hệ
  - Ràng buộc do thuộc tính tổng hợp (Count, Sum)

3. RBTV – CÁC ĐẶC TRƯNG

Các đặc trưng của 1 RBTV:

- Nội dung : phát biểu bằng ngôn ngữ hình thức (phép tính quan hệ, đại số quan hệ, mã giả,…)
- Bối cảnh: là những quan hệ có khả năng làm cho RBTV bị vi phạm.
- Tầm ảnh hưởng: là bảng 2 chiều, xác định các thao tác ảnh hưởng (+) và thao tác không ảnh hưởng (-) lên các quan hệ nằm trong bối cảnh.

3. RBTV – BẢNG TẦM ẢNH HƯỞNG

Bảng tầm ảnh hưởng của RBTV có dạng như sau:

	Thêm	Xóa	Sửa
Quan hệ 1	+	+	- (*)
.....			
Quan hệ n	-	-	+(A)

Ký hiệu + : Có thể gây ra vi phạm RBTV
Ký hiệu - : Không thể gây ra vi phạm RBTV
Ký hiệu +(A) : Có thể gây ra vi phạm RBTV khi thao tác trên thuộc tính A
Ký hiệu –(*) : Không thể gây ra vi phạm RBTV do thao tác không thực hiện được

3. RBTV – TRÊN BỐI CẢNH LÀ 1 QUAN HỆ

3.1. Ràng buộc toàn vẹn miền giá trị

- Xét lược đồ quan hệ
  - **NHANVIEN** (MANV, HONV, TENLOT, TENNV, NGSINH, PHAI, DCHI, MA_NQL, PHONG, MLUONG)

Câu hỏi 40: Phái của nhân viên chỉ có thể là 'Nam' hoặc 'Nữ'

- Nội dung:
  - $\forall n \in$ NHANVIEN: n.PHAI IN {'Nam','Nữ'}
- Bối cảnh: quan hệ NHANVIEN
- Bảng tầm ảnh hưởng (TAH):

	Thêm	Xóa	Sửa
NHANVIEN	+(PHAI)	-	+(PHAI)

3. RBTV – TRÊN BỐI CẢNH LÀ 1 QUAN HỆ

3.2. Ràng buộc toàn vẹn liên thuộc tính: ràng buộc giữa các thuộc tính trong cùng một quan hệ.

Xét lược đồ quan hệ

DEAN (MADA, TENDA, DDIEM_DA, PHONG, NGBD_DK, NGKT_DK)

Câu hỏi 41: Với mọi đề án, ngày bắt đầu dự kiến (NGBD_DK) phải nhỏ hơn ngày kết thúc dự kiến (NGKT_DK)

Nội dung:

$\forall d \in DEAN, d.NGBD\_DK <= d.NGKT\_DK$

3. RBTV – TRÊN BỐI CẢNH LÀ 1 QUAN HỆ

- Bối cảnh: quan hệ DEAN
- Bảng tầm ảnh hưởng:

	Thêm	Xóa	Sửa
DEAN	+ (NGBD_DK, NGKT_DK)	-	+(NGBD_DK, NGKT_DK)

3.3. Ràng buộc toàn vẹn liên bộ: ràng buộc giữa các bộ giá trị trong cùng một quan hệ.

Cho lược đồ quan hệ:

NHANVIEN(MaNV, HoTen, HESO, MucLuong)

Câu hỏi 42: các nhân viên có cùng hệ số lương thì có cùng mức lương.

3. RBTV – TRÊN BỐI CẢNH LÀ 1 QUAN HỆ

– Nội dung:

- $\forall n1,n2 \in$ NHANVIEN: n1.HESO=n2.HESO thì (n1.MUCLUONG = n2.MUCLUONG)

– Bối cảnh: quan hệ NHANVIEN

– Bảng tầm ảnh hưởng:

	Thêm	Xóa	Sửa
NHANVIEN	**+ (HESO, MucLuong)**	**-**	**+(HESO, MucLuong)**

3. RBTV – BỐI CẢNH NHIỀU QUAN HỆ

3.4. Ràng buộc toàn vẹn tham chiếu

- RBTV tham chiếu còn gọi là ràng buộc phụ thuộc tồn tại hay ràng buộc khóa ngoại.

- Xét lược đồ quan hệ

  PHONGBAN (MAPH, TENPH, TRPH, NGNC)

  NHANVIEN (MANV, HOTEN, NTNS, PHAI, MA_NQL, MAPH, LUONG)

  Câu hỏi 43: Mỗi trưởng phòng phải là một nhân viên trong công ty.

  – Nội dung:

  – $\forall p \in$ PHONGBAN, $\exists n \in$ NHANVIEN:

  p.TRPH= n.MANV

  Hay: PHONGBAN[TRPH] $\subseteq$ NHANVIEN[MANV])

3. RBTV – BỐI CẢNH NHIỀU QUAN HỆ

–Bối cảnh: **NHANVIEN, PHONGBAN**

–Bảng tầm ảnh hưởng:

	Thêm	Xóa	Sửa
PHONGBAN	**+(TRPH)**	**-**	**+(TRPH)**
NHANVIEN	**-**	**+**	**- (*)**

3.5. Ràng buộc toàn vẹn liên thuộc tính liên quan hệ

Xét các lược đồ quan hệ:

DATHANG(MADH, MAKH, NGAYDH)

GIAOHANG(MAGH, MADH, NGAYGH)

3. RBTV – BỐI CẢNH NHIỀU QUAN HỆ

Câu hỏi 44: Ngày giao hàng không được trước ngày đặt hàng

- Nội dung:

$\forall g \in GIAO\_HANG,$
$\exists! d \in DAT\_HANG: d.MADH = g.MADH \wedge d.NGAYDH >= g.NGAYGH$

– Bối cảnh: DATHANG, GIAOHANG

– Bảng tầm ảnh hưởng:

	Thêm	Xóa	Sửa
DATHANG	-	-	**+ (ngaydh)**
GIAOHANG	**+(ngaygh)**	-	**+ (ngaygh)**

3. RBTV – BỐI CẢNH NHIỀU QUAN HỆ

3.6. Ràng buộc toàn vẹn liên bộ, liên quan hệ

- RBTV liên bộ, liên quan hệ là điều kiện giữa các bộ trên nhiều quan hệ khác nhau.
- Xét các lược đồ quan hệ
  - **PHONGBAN** (MAPH, TENPH, TRPH, NGNC)
  - **DIADIEM_PHG** (MAPH, DIADIEM)

Câu hỏi 45: Mỗi phòng ban phải có ít nhất một địa điểm phòng

- Nội dung
  - Mỗi phòng ban phải có ít nhất một địa điểm phòng
  - $\forall p \in$ PHONGBAN, $\exists d \in$ DIADIEM_PHG:

    p.MAPH = d.MAPH

3. RBTV – BỐI CẢNH NHIỀU QUAN HỆ

– Bối cảnh: PHONGBAN, DIADIEM_PHG
– Bảng tầm ảnh hưởng:

	Thêm	Xóa	Sửa
PHONGBAN	+	-	-
DIADIEM_PHG	-	+	+ **(MAPH)**

3.7. Ràng buộc toàn vẹn do thuộc tính tổng hợp

PXUAT(SOPHIEU, NGAY, TONGTRIGIA)
CTIET_PX(SOPHIEU, MAHANG, SL, DG)

Câu hỏi 46: Tổng trị giá của 1 phiếu xuất phải bằng tổng trị giá các chi tiết xuất.

3. RBTV – BỐI CẢNH NHIỀU QUAN HỆ

Nội dung

- $\forall px \in PXUAT,$

  $px.TONGTRIGIA = \Sigma_{(ct \in CTIET\_PX \wedge ct.SOPHIEU = px.SOPHIEU)} (ct.SL*ct.DG)$

  – Bối cảnh: PXUAT,CTIET_PX
  – Bảng tầm ảnh hưởng:

	Thêm	Xóa	Sửa
PXUAT	**-(*)**	**-**	**+ (tongtrigia)**
CTIET_PX	**+(sl,dg)**	**+**	**+ (sl,dg)**

-(*) Ở thời điểm thêm một bộ vào PXUAT, giá trị bộ đó tại TONGTRIGIA là trống.

GIẢI BÀI TẬP

HỆ QUẢN TRỊ CƠ SỞ DỮ LIỆU

CHƯƠNG I
GIỚI THIỆU
(Introduction)

MỤC ĐÍCH

Chương này trình bày một cái nhìn bao quát về cơ sở dữ liệu (CSDL/DB), về hệ quản trị cơ sở dữ liệu (HQTCSDL/DBMS) và về hệ cơ sở dữ liệu (HCSDL/DBS). Các đòi hỏi khi xây dựng một HQTCSDL đó cũng chính là những chức năng mà một HCSDL cần phải có. Một khái niệm quan trọng là khái niệm giao dịch (Transaction). Các tính chất một giao dịch phải có để đảm bảo một HQTCSDL, được xây dựng trên HCSDL tương ứng, trong suốt quá trình hoạt động sẽ luôn cho một CSDL tin cậy (dữ liệu luôn nhất quán). Quản trị giao dịch nhằm đảm bảo mọi giao dịch trong hệ thống có các tính chất mà một giao dịch phải có. Một điều cần chú ý là trong các tính chất của một giao dịch, *tính chất nhất quán* trước hết phải được đảm bảo bởi người lập trình-người viết ra giao dịch.

YÊU CẦU

Hiểu các khái niệm.

Hiểu các vấn đề đặt ra khi xây dựng một HQTCSDL: thiết kế CSDL, đảm bảo tính nhất quán của CSDL trong suốt cuộc sống của nó, nền tảng phần cứng trên đó một HQTCSDL được xây dựng.

Hiểu cấu trúc hệ thống tổng thể

Hiểu vai trò của các người sử dụng hệ thống.

MỘT SỐ KHÁI NIỆM

- Một cơ sở dữ liệu (CSDL/ DB: DataBase) là một tập hợp các tập tin có liên quan với nhau, được thiết kế nhằm làm giảm thiểu sự lặp lại dữ liệu.
- Một hệ quản trị cơ sở dữ liệu (HQTCSDL/ DBMS: DataBase Management System) là một hệ thống gồm một CSDL và các thao tác trên CSDL đó, được thiết kế trên một nền tảng phần cứng, phần mềm và với một kiến trúc nhất định.
- Một hệ cơ sở dữ liệu (HCSDL/ DBS: DataBase System) là một phần mềm cho phép xây dựng một HQTCSDL.

HỆ CƠ SỞ DỮ LIỆU

Một số điểm bất lợi chính của việc lưu giữ *thông tin có tổ chức* trong hệ thống xử lý file thông thường:

- **Dư thừa dữ liệu và tính không nhất quán** (Data redundancy and inconsistency): Do các file và các trình ứng dụng được tạo ra bởi các người lập trình khác nhau, nên các file có định dạng khác nhau, các chương trình được viết trong các ngôn ngữ lập trình khác nhau, cùng một thông tin có thể được lưu giữ trong các file khác nhau. Tính không thống nhất và dư thừa này sẽ làm *tăng chi phí truy xuất và lưu trữ*, hơn nũa, nó sẽ dẫn đến tính không nhất quán của dữ liệu: *các bản sao của cùng một dữ liệu có thể không nhất quán*.
- **Khó khăn trong việc truy xuất dữ liệu**: Môi trường của hệ thống xử lý file thông thường không cung cấp các công cụ cho phép truy xuất thông tin một cách hiệu quả và thuận lợi.
- **Sự cô lập dữ liệu** (Data isolation): Các giá trị dữ liệu được lưu trữ trong cơ sở dữ liệu phải thoả mãn một số các *ràng buộc về tính nhất quán của dữ liệu (ràng buộc nhất quán/consistency contraints)*. Trong hệ thống xử lý file thông thường, rất khó khăn trong việc thay đổi các chương trình để thoả mãn các yêu cầu thay đổi ràng buộc. Vấn đề trở nên khó khăn hơn khi các ràng buộc liên quan đến các hạng mục dữ liệu nằm trong các file khác nhau.
- **Các vấn đề về tính nguyên tử** (Atomicity problems): Tính nguyên tử của một hoạt động (giao dịch) là: *hoặc nó được hoàn tất trọn vẹn hoặc không có gì cả*. Điều này có nghĩa là một hoạt động (giao dịch) *chỉ làm thay đổi* các dữ liệu bền vững khi nó đã hoàn tất (kết thúc thành công) nếu không, giao dịch không để lại một dấu vết nào trên CSDL. Trong hệ thống xử lý file thông thường khó đảm bảo được tính chất này.
- **Tính bất thường trong truy xuất cạnh tranh**: Một hệ thống cho phép nhiều người sử dụng cập nhật dữ liệu đồng thời, có thể dẫn đến kết quả là dữ liệu không nhất quán. Điều này đòi hỏi một sự giám sát. Hệ thống xử lý file thông thường không cung cấp chức năng này.
- **Vấn đề an toàn** (Security problems): một người sử dụng hệ cơ sở dữ liệu không cần thiết và cũng không có quyền truy xuất tất cả các dữ liệu. Vấn đề này đòi hỏi hệ thống phải đảm bảo được tính phân quyền, chống truy xuất trái phép ...

Các bất lợi nêu trên đã gợi mở sự phát triển các DBMS. Phần sau của giáo trình sẽ đề cập đến các quan niệm và các thuật toán được sử dụng để phát triển một hệ cơ sở dữ liệu nhằm *giải quyết các vấn đề nêu trên*. Một số khái niệm

GÓC NHÌN DỮ LIỆU

Tính hiệu quả của hệ thống đòi hỏi phải thiết kế các cấu trúc dữ liệu phức tạp để biểu diễn dữ liệu trong cơ sở dữ liệu. Các nhà phát triển che dấu sự phức tạp này thông qua các *mức trừu tượng* nhằm đơn giản hóa sự trao đổi của người sử dụng với hệ thống:

- **Mức vật lý (Physical level)**: Mức thấp nhất của sự trừu tượng, mô tả dữ liệu hiện được lưu trữ thế nào. Ở mức này, cấu trúc dữ liệu mức thấp, phức tạp được mô tả chi tiết.
- **Mức luận lý (Logical level)**: Mức kế cao hơn về sự trừu tượng, mô tả dữ liệu gì được lưu trữ trong cơ sở dữ liệu và các mối quan hệ gì giữa các dữ liệu này. Mức logic của sự trừu tượng được dùng bởi các người quản trị cơ sở dữ liệu.
- **Mức view (view level)**: Mức cao nhất của sự trừu tượng, mô tả chỉ một phần của cơ sở dữ liệu toàn thể. Một người sử dụng cơ sở dữ liệu liên quan đến chỉ một bộ phận của cơ sở dữ liệu. Như vậy sự trao đổi của họ với hệ thống được làm đơn giản bởi việc định nghĩa view. Hệ thống có thể cung cấp nhiều mức view đối với cùng một cơ sở dữ liệu.

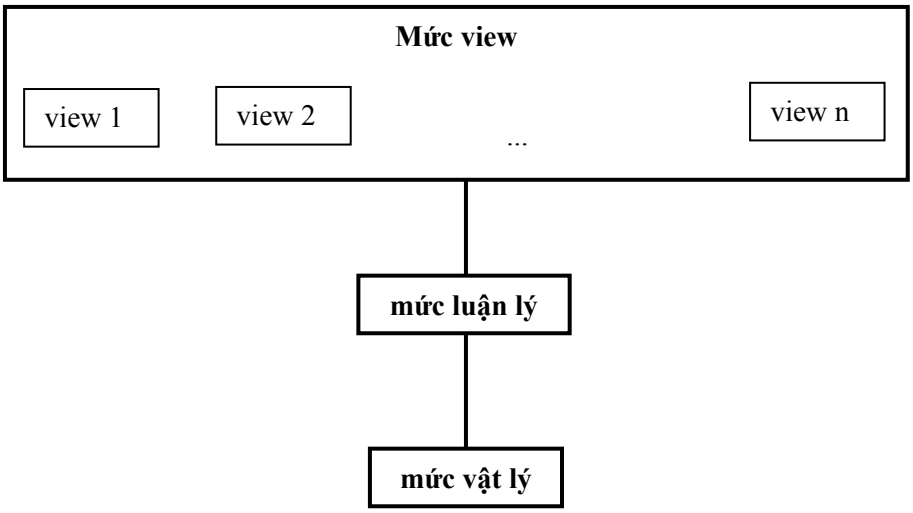


Figure 1

- **Thể hiện và sơ đồ** (instances and schemas): Tập hợp các thông tin được lưu trữ trong cơ sở dữ liệu tại một thời điểm được gọi là một thể hiện (instance) của cơ sở dữ liệu. Thiết kế tổng thể của cơ sở dữ liệu được gọi là sơ đồ (schema).

Một hệ cơ sở dữ liệu có một vài sơ đồ, được phân tương ứng với các mức trừu tượng. ở mức thấp nhất là **sơ đồ vật lý** (physical schema), ở mức trung gian là **sơ đồ luận lý** (logical schema), ở mức cao nhất là **sơ đồ con** (subschema). Nói chung một hệ cơ sở dữ liệu hỗ trợ một sơ đồ vật lý, một sơ đồ luận lý và một vài sơ đồ con.

- Khả năng sửa đổi một định nghĩa ở một mức không ảnh hưởng một định nghĩa sơ đồ ở mức cao hơn được gọi là **sự độc lập dữ liệu** (data independence). Có hai mức độc lập dữ liệu:
  - **Độc lập dữ liệu vật lý** (Physical data independence) là khả năng sửa đổi sơ đồ vật lý không làm cho các chương trình ứng dụng phải viết lại. Các sửa đổi ở mức vật lý là cần thiết để cải thiện hiệu năng.

- **Độc lập dữ liệu luận lý** (Logical data independence) là khả năng sửa đổi sơ đồ luận lý không làm cho các chương trình ứng dụng phải viết lại. Các sửa đổi ở mức luận lý là cần thiết khi cấu trúc luận lý của cơ sở dữ liệu bị thay thế.

MÔ HÌNH DỮ LIỆU

Nằm dưới cấu trúc của một cơ sở dữ liệu là mô hình dữ liệu: một bộ các công cụ quan niệm để mô tả dữ liệu, quan hệ dữ liệu, ngữ nghĩa dữ liệu và các ràng buộc nhất quán. Có ba nhóm mô hình: Các mô hình luận lý dựa trên đối tượng (Object-based logical models), các mô hình luận lý dựa trên mẩu tin (record-based logical models), các mô hình vật lý (physical models).

- **Các mô hình luận lý dựa trên đối tượng** được dùng mô tả dữ liệu ở mức luận lý và mức view. Chúng được đặc trưng bởi việc chúng cung cấp khả năng cấu trúc linh hoạt và cho phép các ràng buộc dữ liệu được xác định một cách tường minh. Dưới đây là một vài mô hình được biết rộng rãi: Mô hình **thực thể - quan hệ** (entity-relationship model), mô hình **hướng đối tượng** (object-oriented model), mô hình **dữ liệu ngữ nghĩa** (semantic data model), mô hình **dữ liệu hàm** (function data model).
- **Các mô hình luận lý dựa trên mẩu tin** được dùng để miêu tả dữ liệu ở mức luận lý hay mức view. Chúng được dùng để xác định cấu trúc luận lý toàn thể của cơ sở dữ liệu và cung cấp sự mô tả mức cao hơn việc thực hiện. Cơ sở dữ liệu được cấu trúc ở dạng mẩu tin định dạng cố định (fixed format record): mỗi mẩu tin xác định một số cố định các trường, mối trường thường có độ dài cố định. Một vài mô hình được biết rộng rãi là: **Mô hình quan hệ, mô hình mạng, mô hình phân cấp**.
- **Mô hình dữ liệu vật lý** được dùng để mô tả dữ liệu ở mức thấp nhất. Hai mô hình dữ liệu vật lý được biết rộng rãi nhất là **mô hình hợp nhất** (unifying model) và **mô hình khung-bộ nhớ** (frame-memory model).

NGÔN NGỮ CƠ SỞ DỮ LIỆU

Một hệ cơ sở dữ liệu cung cấp hai kiểu ngôn ngữ khác nhau: một để xác định sơ đồ cơ sở dữ liệu, một để biểu diễn các vấn tin cơ sở dữ liệu và cập nhật.

- **Ngôn ngữ định nghĩa dữ liệu (Data Definition Language: DDL)** cho phép định nghĩa sơ đồ cơ sở dữ liệu. Kết quả biên dịch các lệnh của DDL là tập hợp các bảng được lưu trữ trong một ***file đặc biệt được gọi là tự điển dữ liệu (data dictionary)*** hay thư mục dữ liệu (***data directory***). Tự điển dữ liệu là một file chứa ***metadata***. File này được tra cứu trước khi dữ liệu hiện hành được đọc hay sửa đổi. Cấu trúc lưu trữ và phương pháp truy cập được sử dụng bởi hệ cơ sở dữ liệu được xác định bởi một tập hợp các định nghĩa trong một kiểu đặc biệt của DDL được gọi là ***ngôn ngữ định nghĩa và lưu trữ dữ liệu (data storage and definition language)***. Kết quả biên dịch của các định nghĩa này là một tập hợp các chỉ thị xác định sự thực hiện chi tiết của các sơ đồ cơ sở dữ liệu (thường được che dấu).
- **Ngôn ngữ thao tác dữ liệu (Data manipulation language: DML)** là ngôn ngữ cho phép người sử dụng truy xuất hoặc thao tác dữ liệu. Có hai kiểu ngôn ngữ thao tác dữ liệu: ***DML thủ tục (procedural DML)*** yêu cầu người sử dụng đặc tả dữ liệu nào cần và làm thế nào để nhận được nó. ***DML không thủ tục (Nonprocedural DML)*** yêu cầu người sử dụng đặc tả dữ liệu nào cần nhưng không cần đặc tả làm thế nào để nhận được nó. Một vấn tin (***query***) là một lệnh yêu cầu tìm lại dữ liệu (***information***

retrieval). Phần ngôn ngữ DML liên quan đến sự tìm lại thông tin được gọi là ngôn ngữ vấn tin (***query language***).

QUẢN TRỊ GIAO DỊCH

Thông thường, một số thao tác trên cơ sở dữ liệu tạo thành một đơn vị logic công việc. Ta hãy xét ví dụ chuyển khoản, trong đó một số tiền x được chuyển từ tài khoản A (A:=A-x) sang một tài khoản B (B:=B+x). Một yếu tố cần thiết là cả hai thao tác này hoặc cùng xảy ra hoặc không hoạt động nào xảy ra cả. Việc chuyển khoản phải xảy ra trong tính toàn thể của nó hoặc không. Đòi hỏi ***toàn thể-hoặc-không*** này được gọi là tính nguyên tử (atomicity). Một yếu tố cần thiết khác là sự thực hiện việc chuyển khoản bảo tồn tính nhất quán của cơ sở dữ liệu: *giá trị của tổng A + B phải được bảo tồn*. Đòi hỏi về tính chính xác này được gọi là tính nhất quán (consistency). Cuối cùng, sau khi thực hiện thành công hoạt động chuyển khoản, các giá trị của các tài khoản A và B phải bền vững cho dù có thể có sự cố hệ thống. Đòi hỏi về tính bền vững này được gọi là tính lâu bền (durability).

Một giao dịch là một tập các hoạt động thực hiện chỉ một chức năng logic trong một ứng dụng cơ sở dữ liệu. Mỗi giao dịch là một đơn vị mang cả tính nguyên tử lẫn tính nhất quán. Như vậy, các giao dịch phải không được vi phạm bất kỳ ràng buộc nhất quán nào: ***Nếu cơ sở dữ liệu là nhất quán khi một giao dịch khởi động thì nó cũng phải là nhất quán khi giao dịch kết thúc thành công***. Tuy nhiên, trong khi đang thực hiện giao dịch, phải cho phép sự không nhất quán tạm thời. Sự không nhất quán tạm thời này tuy là cần thiết nhưng lại có thể dẫn đến các khó khăn nếu xảy ra sự cố.

Trách nhiệm của người lập trình là xác định đúng đắn các giao dịch sao cho mỗi một bảo tồn tính nhất quán của cơ sở dữ liệu.

Đảm bảo tính nguyên tử và tính lâu bền là trách nhiệm của hệ cơ sở dữ liệu nói chung và của ***thành phần quản trị giao dịch (transaction-management component)*** nói riêng. Nếu không có sự cố, tất cả giao dịch hoàn tất thành công và tính nguyên tử được hoàn thành dễ dàng. Tuy nhiên, do sự hiện diện của các sự cố, một giao dich có thể không hoàn tất thành công sự thực hiện của nó. Nếu tính nguyên tử được đảm bảo, một giao dịch thất bại không gây hiệu quả đến trạng thái của cơ sở dữ liệu. Như vậy, cơ sở dữ liệu phải được hoàn lại trạng thái của nó trước khi giao dịch bắt đầu. Hệ cơ sở dữ liệu phải có trách nhiệm phát hiện sự cố hệ thống và trả lại cơ sở dữ liệu về trạng thái trước khi xảy ra sự cố.

Khi một số giao dịch cạnh tranh cập nhật cơ sở dữ liệu, tính nhất quán của dữ liệu có thể không được bảo tồn, ngay cả khi mỗi giao dịch là chính xác. ***Bộ quản trị điều khiển cạnh tranh (concurency-control manager)*** có trách nhiệm điều khiển các trao đổi giữa các giao dịch cạnh tranh để đảm bảo tính thống nhất của CSDL.

QUẢN TRỊ LƯU TRỮ

Các CSDL đòi hỏi một khối lượng lớn không gian lưu trữ, có thể lên đến nhiều terabytes (1 terabyte=10^3 Gigabytes=10^6 Megabytes). Các thông tin phải được lưu trữ trên lưu trữ ngoài (đĩa). Dữ liệu được di chuyển giữa lưu trữ đĩa và bộ nhớ chính khi cần thiết. Do việc di chuyển dữ liệu từ và lên đĩa tương đối chậm so với tốc độ của đơn vị xử lý trung tâm, điều này ép buộc hệ CSDL phải cấu trúc dữ liệu sao cho tối ưu hóa nhu cầu di chuyển dữ liệu giữa đĩa và bộ nhớ chính.

Mục đích của một hệ CSDL là làm đơn giản và dễ dàng việc truy xuất dữ liệu. Người sử dụng hệ thống có thể không cần quan tâm đến chi tiết vật lý của sự thực thi hệ thống. Phần lớn họ chỉ quan đến hiệu năng của hệ thống (thời gian trả lời một câu vấn tin ...).

Bộ quản trị lưu trữ (storage manager) là một module chương trình cung cấp giao diện giữa dữ liệu mức thấp được lưu trữ trong CSDL với các chương trình ứng dụng và các câu vấn tin được đệ trình cho hệ thống. Bộ quản trị lưu trữ có trách nhiệm trao đổi với bộ quản trị file (file manager). Dữ liệu thô được lưu trữ trên đĩa sử dụng hệ thống file (file system), hệ thống này thường được cung cấp bởi hệ điều hành. Bộ quản trị lưu trữ dịch các câu lệnh DML thành các lệnh của hệ thống file mức thấp. Như vậy, *bộ quản trị lưu trữ có nhiệm vụ lưu trữ, tìm lại và cập nhật dữ liệu trong CSDL*.

NHÀ QUẢN TRỊ CƠ SỞ DỮ LIỆU

Một trong các lý do chính đối với việc sử dụng DBMS là có sự điều khiển trung tâm cho cả dữ liệu lẫn các chương trình truy cập các dữ liệu này. Người điều khiển trung tâm trên toàn hệ thống như vậy gọi là nhà quản trị cơ sở dữ liệu (DataBase Administrator - DBA). Các chức năng của DBA như sau:

- ***Định nghĩa sơ đồ***: DBA tạo ra sơ đồ CSDL gốc bằng cách viết một tập các định nghĩa mà nó sẽ được dịch bởi trình biên dịch DDL thành một tập các bảng được lưu trữ thường trực trong tự điển dữ liệu.
- ***Định nghĩa cấu trúc lưu trữ và phương pháp truy xuất***: DBA tạo ra một cấu trúc lưu trữ thích hợp và các phương pháp truy xuất bằng cách viết một tập hợp các định nghĩa mà nó sẽ được dịch bởi trình biên dịch lưu trữ dữ liệu và ngôn ngữ định nghĩa dữ liệu.
- ***Sửa đổi sơ đồ và tổ chức vật lý***
- ***Cấp quyền truy xuất dữ liệu***: Việc cấp các dạng quyền truy cập khác nhau cho phép DBA điều hoà những phần của CSDL mà nhiều người có thể truy xuất. Thông tin về quyền được lưu giữ trong một cấu trúc hệ thống đặc biệt, nó được tham khảo bởi hệ CSDL mỗi khi có sự truy xuất dữ liệu của hệ thống.
- ***Đặc tả ràng buộc toàn vẹn (integrity-contraint)***: Các giá trị dữ liệu được lưu trữ trong CSDL phải thoả mãn một số các ràng buộc nhất quán nhất định. Ví dụ số giờ làm việc của một nhân viên trong một tuần không thể vượt quá một giới hạn 80 giờ chẳng hạn. Một ràng buộc như vậy phải được đặc tả một cách tường minh bởi DBA. Các ràng buộc toàn vẹn được lưu giữ trong một cấu trúc hệ thống đặc biệt được tham khảo bởi hệ CSDL mỗi khi có sự cập nhật dữ liệu.

NGƯỜI SỬ DỤNG CSDL

Mục đích đầu tiên của hệ CSDL là cung cấp một môi trường để tìm lại thông tin và lưu thông tin trong CSDL. Các người sử dụng cơ sở dữ liệu được phân thành bốn nhóm tuỳ theo cách thức họ trao đổi với hệ thống.

- ***Các người lập trình ứng dụng***: Là nhà chuyên môn máy tính người trao đổi với hệ thống thông qua các lời gọi DML được nhúng trong một chương trình được viết trong một ngôn ngữ chủ - *host language* (Pascal, C, Cobol ...). Các chương trình này thường được tham khảo như các chương trình ứng dụng. Vì cú pháp DML thường rất khác với cú pháp của ngôn ngữ chủ, các lời gọi DML thường được bắt đầu bởi một ký tự đặc biệt như vậy mã thích hợp mới có thể được sinh. Một bộ tiền xử lý đặc biệt, được gọi là tiền

biên dịch (precompiler) DML, chuyển các lệnh DML thành các lời gọi thủ tục chuẩn trong ngôn ngữ chủ. Bộ biên dịch ngôn ngữ chủ sẽ sinh mã đối tượng thích hợp. Có những ngôn ngữ lập trình phối hợp cấu trúc điều khiển của các ngôn ngữ giống như Pascal với cấu trúc điều khiển để thao tác đối tượng CSDL. Các ngôn ngữ này (đôi khi được gọi là ngôn ngữ thế hệ thứ tư) thường bao gồm các đặc điểm đặc biệt để làm dễ dàng việc sinh các dạng và hiển thị dữ liệu trên màn hình.

- ***Các người sử dụng thành thạo (Sophisticated users)***: Trao đổi với hệ thống không qua viết trình. Thay vào đó họ đặt ra các yêu cầu của họ trong ngôn ngữ truy vấn CSDL (Database query language). Mỗi câu vấn tin như vậy được đệ trình cho bộ xử lý vấn tin, chức năng của bộ xử lý vấn tin là "dịch" các lệnh DML thành các chỉ thị mà bộ quản trị lưu trữ hiểu. Các nhà phân tích đệ trình các câu vấn tin thăm dò dữ liệu trong cơ sở dữ liệu thuộc vào phạm trù này.
- ***Các người sử dụng chuyên biệt (Specialized users)***: Là các người sử dụng thành thạo, họ viết các ứng dụng CSDL chuyên biệt không nằm trong khung xử lý dữ liệu truyền thống. Trong đó, phải kể đến các hệ thống thiết kế được trợ giúp bởi máy tính (computer-aided design systems), Cơ sở tri thức (knowledge-base) và hệ chuyên gia (expert systems), các hệ thống lưu trữ dữ liệu với kiểu dữ liệu phức tạp (dữ liệu đồ họa, hình ảnh, âm thanh) và các hệ thống mô hình môi trường (environment-modeling systems)
- ***Các người sử dụng ngây thơ (Naive users)***: là các người sử dụng không thành thạo, họ trao đổi với hệ thống bởi cầu dẫn một trong các chương trình ứng dụng thường trực đã được viết sẵn.

CẤU TRÚC HỆ THỐNG TỔNG THỂ

Một hệ CSDL được phân thành các module, mỗi một thực hiện một trách nhiệm trong hệ thống tổng thể. Một số chức năng của hệ CSDL có thể được cung cấp bởi hệ điều hành. Trong hầu hết các trường hợp, hệ điều hành chỉ cung cấp các dịch vụ cơ sở nhất, hệ CSDL phải xây dựng trên cơ sở đó. Như vậy, thiết kế hệ CSDL phải xem xét đến giao diện giữa hệ CSDL và hệ điều hành.

Các thành phần chức năng của hệ CSDL có thể được chia thành các thành phần xử lý vấn tin (query processor components) và các thành phần quản trị lưu trữ (storage manager components).

Các thành phần xử lý vấn tin gồm:

- ***Trình biên dịch DML (DML compiler):*** dịch các lệnh DML trong một ngôn ngữ vấn tin thành các chỉ thị mức thấp mà engine định giá vấn tin (query evaluation engine) có thể hiểu. Hơn nữa, Trình biên dịch DML phải biến đổi một yêu cầu của người sử dụng thành một đích tương đương nhưng ở dạng hiệu quả hơn có nghĩa là tìm một chiến lược tốt để thực hiện câu vấn tin.
- ***Trình tiền biên dịch DML nhúng (Embedded DML Precompiler):*** biến đổi các lệnh DML được nhúng trong một chương trình ứng dụng thành các lời gọi thủ tục chuẩn trong ngôn ngữ chủ. Trình tiền biên dịch phải trao đổi với trình biên dịch DML để sinh mã thích hợp.
- ***Bộ thông dịch DDL (DDL interpreter):*** thông dịch các lệnh DDL và ghi chúng vào một tập hợp các bảng chứa metadata.

- ***Engine định giá vấn tin (Query evaluation engine):*** Thực hiện các chỉ thị mức thấp được sinh ra bởi trình biên dịch DML.

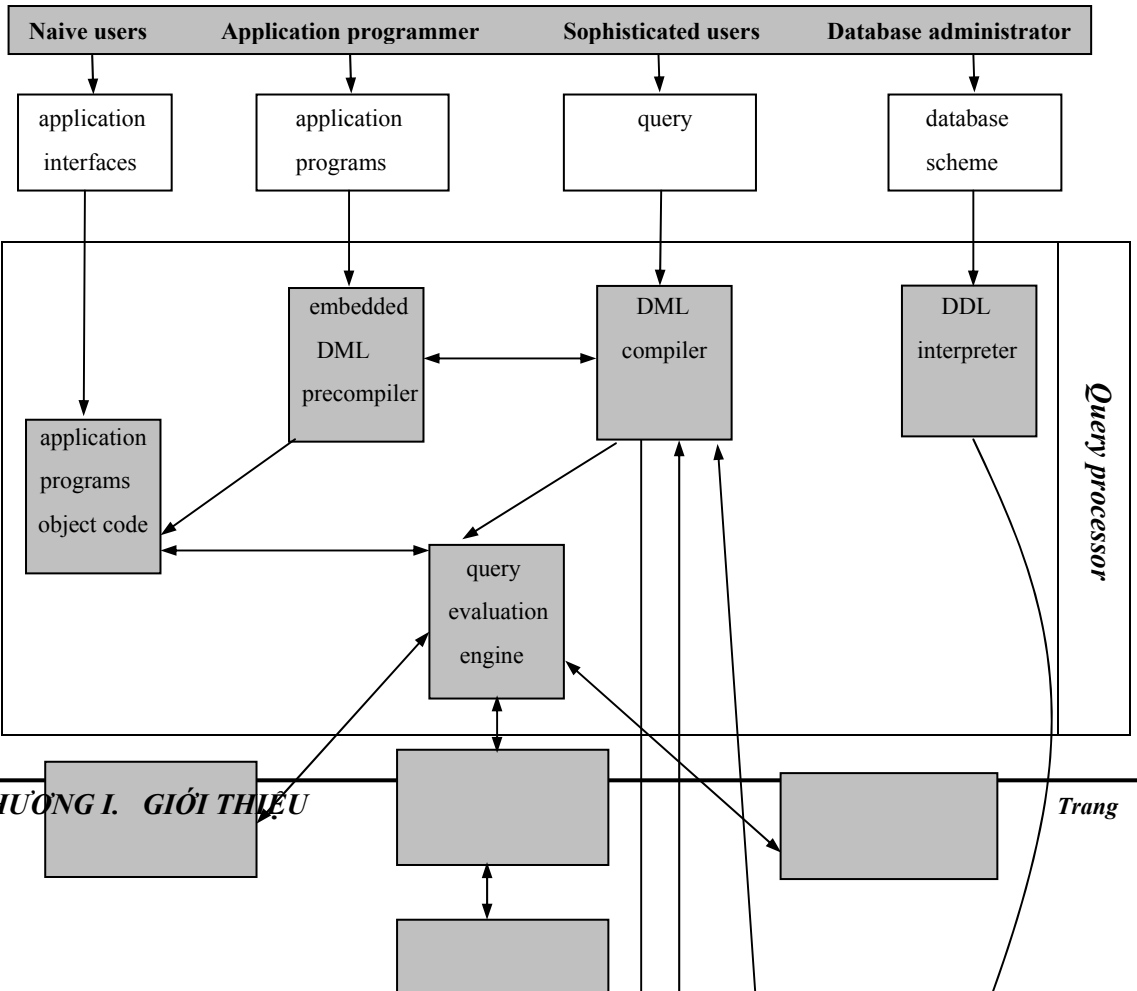
Các thành phần quản trị lưu trữ cung cấp các giao diện giữa dữ liệu mức thấp được lưu trữ trong CSDL và các chương trình ứng dụng, các vấn tin được đệ trình cho hệ thống. Các thành phần quản trị lưu trữ gồm:

- ***Bộ quản trị quyền và tính toàn vẹn (Authorization and integrity manager):*** kiểm tra sự thoả mãn các ràng buộc toàn vẹn và kiểm tra quyền truy xuất dữ liệu của người sử dụng.
- ***Bộ quản trị giao dịch (Transaction manager):*** Đảm bảo rằng CSDL được duy trì trong trạng thái nhất quán cho dù hệ thống có sự cố và đảm bảo rằng các thực hiện giao dịch cạnh tranh tiến triển không xung đột.
- ***Bộ quản trị file (File manager):*** Quản trị cấp phát không gian trên lưu trữ đĩa và các cấu trúc dữ liệu được dùng để biểu diễn thông tin được lưu trữ trên đĩa.
- ***Bộ quản trị bộ đệm (Buffer manager):*** có trách nhiệm đem dữ liệu từ lưu trữ đĩa vào bộ nhớ chính và quyết định dữ liệu nào trữ trong bộ nhớ.

Hơn nữa, một số cấu trúc dữ liệu được cần đến như bộ phận của sự thực thi hệ thống vật lý:

- ***Các file dữ liệu:*** Lưu trữ CSDL
- ***Tự điển dữ liệu (Data Dictionary):*** lưu metadata về cấu trúc CSDL.
- ***Chỉ mục (Indices):*** cung cấp truy xuất nhanh đến các hạng mục dữ liệu chứa các giá trị tìm kiếm.
- ***Dữ liệu thống kê (Statistical data):*** lưu trữ thông tin thống kê về dữ liệu trong cơ sở dữ liệu. Thông tin này được dùng bởi bộ xử lý vấn tin để chọn những phương pháp hiệu quả thực hiện câu vấn tin.

Sơ đồ các thành phần và các nối kết giữa chúng



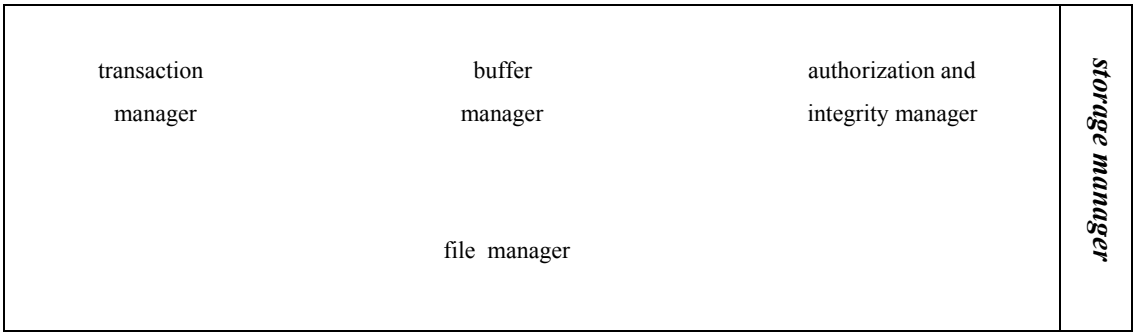


Figure 2

KIẾN TRÚC HỆ CƠ SỞ DỮ LIỆU

Kiến trúc hệ CSDL bị ảnh hưởng nhiều bởi hệ thống máy nền. Các sắc thái của kiến trúc máy như mạng, song song và phân tán được phản ánh trong kiến trúc của hệ CSDL.

- Mạng máy tính cho phép thực hiện một số công việc trên một hệ thống các server, một số công việc trên các hệ thống client. Việc phân chia công việc này dẫn đến sự phát triển hệ CSDL *client-server.*
- Xử lý song song trong một hệ thống máy tính làm tăng tốc độ các hoạt động của hệ CSDL, trả lời các giao dịch nhanh hơn. Các vấn tin được xử lý theo cách khai thác tính song song. Sự cần thiết xử lý vấn tin song song này dẫn tới sự phát triển của hệ CSDL *song song*.
- Dữ liệu phân tán trên các site hoặc trên các bộ phận trong một cơ quan cho phép các dữ liệu thường trú tại nơi chúng được sinh ra nhưng vẫn có thể truy xuất chúng từ các site khác hay các bộ phận khác. Việc lưu nhiều bản sao của CSDL trên các site khác nhau cho phép các tổ chức lớn vẫn có thể tiếp tục hoạt động khi một hay một vài site bị sự cố. Hệ CSDL phân tán được phát triển để quản lý dữ liệu phân tán, trên phương diện địa lý hay quản trị, trải rộng trên nhiều hệ CSDL .

HỆ THỐNG TẬP TRUNG

Các hệ CSDL tập trung chạy trên máy đơn và không trao đổi với các máy khác. Các hệ thống như vậy trải từ các hệ CSDL một người sử dụng chạy trên các máy cá nhân (PC) đến các hệ CSDL hiệu năng cao chạy trên các hệ mainframe. Một hệ máy tính mục đích chung hiện đại gồm một hoặc một vài CPU và một số bộ điều khiển thiết bị được nối với nhau thông qua một bus

chung, cho phép truy xuất đến bộ nhớ chia sẻ. CPU có bộ nhớ cache cục bộ lưu các bản sao của một số phần của bộ nhớ chính nhằm tăng tốc độ truy xuất dữ liệu. Mỗi bộ điều khiển thiết bị phụ trách một kiểu thiết bị xác định. Các CPU và các bộ điều khiển thiết bị có thể thực hiện đồng thời, canh tranh truy cập bộ nhớ. Bộ nhớ cache giúp làm giảm sự tranh chấp truy xuất bộ nhớ. Ta phân biệt hai cách các máy tính được sử dụng: Hệ thống một người dùng và hệ thống nhiều người dùng. Hệ CSDL được thiết kế cho hệ thống một người dùng không hỗ trợ điều khiển cạnh tranh, chức năng phục hồi hoặc là thiếu hoặc chỉ là một sự chép dự phòng đơn giản.

HỆ THỐNG CLIENT-SERVER

Các máy tính cá nhân (PC) ngày càng trở nên mạnh hơn, nhanh hơn, và rẻ hơn. Có sự chuyển dịch trong hệ thống tập trung. Các đầu cuối (terminal) được nối với hệ thống tập trung bây giờ được thế chỗ bởi các máy tính cá nhân. Chức năng giao diện người dùng (user interface) thường được quản lý trực tiếp bởi các hệ thống tập trung nay được quản lý bởi các máy tính cá nhân. Như vậy, các hệ thống tập trung ngày nay hoạt động như các hệ thống server nó làm thoả mãn các đòi hỏi của các client. Chức năng CSDL có thể được chia thành hai phần: phần trước (front-end) và phần sau (back-end). Phần sau quản trị truy xuất cấu trúc, định giá câu vấn tin và tối ưu hoá, điều khiển sự xảy ra đồng thời và phục hồi. Phần trước của hệ CSDL gồm các công cụ như: tạo mẫu (form), các bộ soạn báo cáo (report writer), giao diện đồ hoạ người dùng (graphical user interface). Giao diện giữa phần trước và phần sau thông qua SQL hoặc một chương trình ứng dụng. Các hệ thống server có thể được phân thành các phạm trù : server giao dịch (transaction server), server dữ liệu (data server).

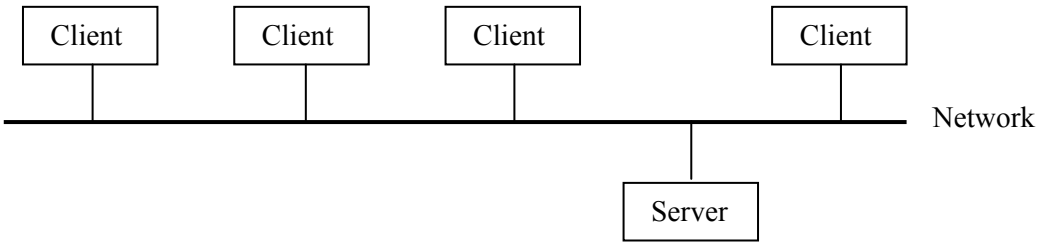


Figure 3

- ***Hệ thống server giao dịch (transaction-server systems)***: còn được gọi là hệ thống server vấn tin (query-server system), cung cấp một giao diện mà các client có thể gửi đến nó các yêu cầu thực hiện một hành động. Để đáp ứng các yêu cầu, hệ thống thực hiện các hành động và gửi lại client các kết quả. Các người sử dụng có thể đặc tả các yêu cầu trong SQL hoặc trong một giao diện trình ứng dụng sử dụng một cơ chế gọi thủ tục xa (remote-procedure-call).
    - ***Các servers giao dịch (Transaction servers):*** Trong các hệ thống tập trung, phần trước (front-end) và phần sau (back-end) được thực hiện trong một hệ thống. Kiến trúc server giao dịch cho phép chia chức năng giữa phần trước và phần sau. Chức năng phần trước được hỗ trợ trên các máy tính cá nhân (PC). Các PC hành động như những khách hàng của các hệ thống server nơi lưu trữ một khối lượng lớn dữ liệu và hỗ trợ các chức năng phần sau. Các clients gửi các giao dịch đến các hệ thống server tại đó các giao dịch được thực hiện và

các kết quả được gửi trả lại cho các clients, người giữ trách nhiệm hiển thị dữ liệu.

ODBC (Open DataBase Connectivity) được phát triển để tạo giao diện giữa các clients và các servers. ODBC là một giao diện trình ứng dụng cho phép các clients sinh ra các lệnh SQL và gửi đến một server tại đó lệnh được thực hiện. Bất kỳ client nào sử dụng giao diện có thể nối với bất kỳ một server nào cung cấp giao diện này.

Các giao diện client-server khác ODBC cũng được sử dụng trong một số hệ thống xử lý giao dịch. Chúng được xác định bởi một giao diện lập trình ứng dụng, sử dụng nó các clients tạo ra các lời gọi thủ tục giao dịch từ xa (transactional remote procedure calls) trên server. Các lời gọi này giống như các lời gọi thủ tục gốc đối với người lập trình nhưng tất cả các lời gọi thủ tục từ xa của một client được bao trong một giao dịch ở server cuối. Như vậy nếu giao dịch bỏ dở, server có thể huỷ bỏ hiệu quả của các lời gọi thủ tục xa riêng lẻ.

- ***Hệ thống server dữ liệu (Data-server systems)***: cho phép các clients trao đổi với các server bằng cách tạo ra các yêu cầu đọc hoặc cập nhật dữ liệu trong các đơn vị như file hoặc trang. Ví dụ, các file-servers cung cấp một giao diện với hệ thống file tại đó các clients có thể tạo, cập nhật, đọc hoặc xoá files. Các servers dữ liệu của cơ sở dữ liệu cung cấp nhiều chức năng hơn; chúng hỗ trợ các đơn vị dữ liệu nhỏ hơn file như trang, bộ (tuple) hoặc đối tượng. Chúng cũng cung cấp phương tiện dễ dàng để lấy chỉ mục (indexing) dữ liệu, phương tiện dễ dàng để tạo giao dịch.
    - ***Các server dữ liệu (Data Servers):*** Các hệ thống server dữ liệu được sử dụng trong các mạng cục bộ, trong đó có một nối kết tốc độ cao giữa các máy clients và máy server, các máy clients có sức mạnh xử lý tương thích với máy server và các công việc phải được thực hiện là tăng cường tính toán. Trong một môi trường như vậy, có thể gửi dữ liệu đến các máy client để thực hiện tất cả các xử lý tại máy clients sau đó gửi dữ liệu trở lại đến máy server. Kiến trúc này đòi hỏi các tính năng back-end đầy đủ tại các clients. Kiến trúc server dữ liệu thường được gặp trong các hệ CSDL hướng đối tượng (Object-Oriented DataBase Systems)

      Gửi trang đối lại với gửi hạng mục (Page shipping versus item shipping): Đơn vị liên lạc dữ liệu có thể là các "hạt thô" (Coarse granularity) như một trang, hay hạt min (fine granularity) như một bộ (tuple)/ đối tượng (object). Ta dùng thuật ngữ hạng mục để chỉ bộ hay đối tượng. Nếu đơn vị liên lạc là một hạng mục sẽ dẫn đến tổng chi phí truyền thông điệp tăng. Đem về hạng mục (fetching item) trước khi nó được yêu cầu, được gọi là đem về trước (Prefetching). Gửi trang có thể được xem như một dạng của đem về trước nếu một trang chứa nhiều hạng mục.

      Chốt (Locking): Các chốt thường được cấp bởi server trên các hạng mục mà nó gửi cho các máy clients. *Khi client giữ một chốt trên một hạng mục dữ liệu, nó có quyền "sử dụng" hạng mục dữ liệu này, hơn nữa trong khoảng thời gian client giữ chốt trên hạng mục dữ liệu không một client nào khác có thể sử dụng hạng mục dữ liệu này*. Bất lợi của gửi trang là các máy client có thể được cấp các chốt "hạt quá thô" -- một chốt trên một trang ẩn chứa các chốt trên tất cả các hạng mục trong trang. Các kỹ thuật nhằm tiết giảm chốt (lock deescalation) được đề nghị, trong đó server có thể yêu cầu các clients truyền trả lại các chốt

trên các hạng mục cấp phát trước. Nếu máy client không cần hạng mục cấp phát trước, nó có thể truyền trả lại các chốt trên hạng mục cho server và các chốt này có thể được cấp phát cho các clients khác.

Trữ dữ liệu (Data caching): Dữ liệu được gửi đến một client với danh nghĩa một giao dịch có thể được trữ ở client, ngay cả khi giao dịch đã hoàn tất, nếu không gian lưu trữ có sẵn. Các giao dịch liên tiếp tại cùng một client có thể dùng dữ liệu được trữ. Tuy nhiên, sự kết dính dữ liệu là một vấn đề cần phải được xem xét: một giao dịch tìm thấy dữ liệu được trữ, nó phải chắc chắn rằng dữ liệu này là "mới nhất" vì các dữ liệu này có thể được cập nhật bởi một client khác sau khi chúng được trữ. Như vậy, vẫn phải trao đổi với server để kiểm tra tính hợp lệ của dữ liệu và để giành được một chốt trên dữ liệu.

Trữ chốt (Lock caching): Các chốt cũng có thể được trữ lại tại máy client. Nếu một hạng mục dữ liệu được tìm thấy trong cache và chốt yêu cầu cho một truy xuất đến hạng mục dữ liệu này cũng tìm thấy trong cache, thì việc truy xuất có thể tiến hành không cần một liên lạc nào với server. Tuy nhiên, server cũng phải lưu lại vết của các chốt được trữ. Nếu một client đòi hỏi một chốt từ server, server phải gọi lại tất cả các chốt xung đột trên cùng hạng mục dữ liệu từ tất cả các máy clients đã trữ các chốt.

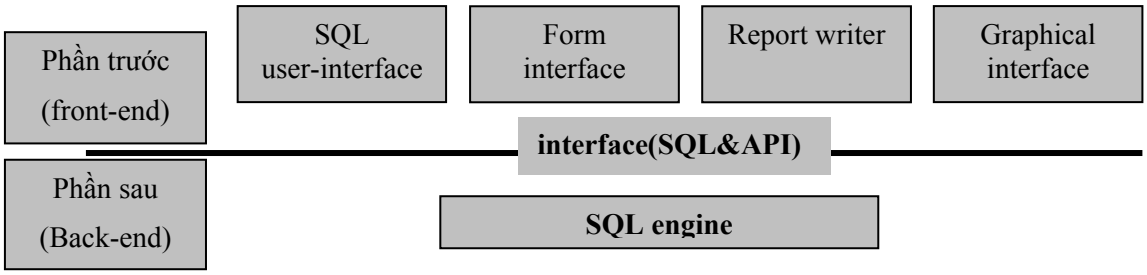


Figure 4

CÁC HỆ SONG SONG (Parallel Systems):

Các hệ song song cải tiến tốc độ xử lý và tốc độ I/O bằng cách sử dụng nhiều CPU và nhiều đĩa song song. Trong xử lý song song, nhiều hoạt động được thực hiện đồng thời. Một máy song song "hạt thô" (coarse-grain) gồm một số nhỏ các bộ xử lý mạnh. Một máy song song đồ sộ (massively parallel) hay "hạt mịn" (fine-grain) sử dụng hàng ngàn bộ xử lý nhỏ hơn. Có hai biện pháp chính để đánh giá hiệu năng của một hệ CSDL. Thứ nhất là năng lực truyền qua (throughput): *số công việc có thể được hoàn tất trong một khoảng thời gian đã cho*. Thứ hai là thời gian đáp ứng (response time): *lượng thời gian cần thiết để hoàn thành một công việc từ lúc nó được đệ trình*. Một hệ thống xử lý một lượng lớn các giao dịch nhỏ có thể cải tiến năng lực truyền qua bởi xử lý song song nhiều giao dịch. Một hệ thống xử lý các giao dịch lớn có thể cải tiến thời gian đáp ứng cũng như năng lực truyền qua bởi thực hiện song song các công việc con (subtask) của mỗi giao dịch.

- **Tăng tốc độ và tăng quy mô (Speedup & Scaleup)**: Tăng tốc độ ám chỉ việc chạy một công việc đã cho trong thời gian ngắn hơn bằng cách tăng bậc song song. Tăng quy mô ám chỉ việc quản lý các công việc lớn bằng cách tăng bậc song song. Chúng ta hãy xét một ứng dụng CSDL chạy trên một hệ thống song song với một số processor và một số đĩa. Giả sử, chúng ta tăng kích cỡ của hệ thống bằng cách tăng số processor, đĩa, và các thành phần khác của hệ thống. Mục đích là xử lý công việc trong thời gian tỷ lệ nghịch với số processor và đĩa được cấp phát.

Giả sử, thời gian thực hiện một công việc trên một máy tính lớn là $\mathbf{T_L}$, thời gian thực hiện cùng công việc này trên máy tính nhỏ là $\mathbf{T_S}$. Tăng tốc độ nhờ song song được định nghĩa là tỷ số $\mathbf{T_S/T_L}$, hệ thống song song được gọi là tăng tốc độ tuyến tính nếu tốc độ tăng là N khi hệ thống lớn có N lần tài nguyên (CPU, đĩa ...) lớn hơn hệ thống nhỏ. Nếu tốc độ tăng nhỏ hơn N, hệ thống được gọi là tăng tốc độ hạ tuyến tính (sublinear).

Tăng quy mô liên quan đến khả năng xử lý các công việc lớn trong trong cùng một lượng thời gian bằng cách cung cấp thêm tài nguyên. Giả sử, Q là một công việc, Q_N là một công việc N lần lớn hơn Q. Giả sử thời gian thực hiện công việc Q trên một máy M_S là T_S và thời gian thực hiện công việc Q_N trên một máy song song M_L, N lần lớn hơn M_S, là T_L. Tăng quy mô được định nghĩa là $\mathbf{T_S/T_L}$. Hệ song song M_L được gọi là tăng quy mô tuyến tính trên công việc Q nếu $\mathbf{T_S = T_L}$. Nếu $T_L > T_S$, hệ thống được gọi là tăng quy mô hạ tuyến tính. Tăng quy mô là một độ đo (metric) quan trọng hơn trong đo lường hiệu quả của các hệ CSDL song song. Đích của song song trong các hệ CSDL là đảm bảo hệ CSDL có thể tiếp tục thực hiện ở một tốc độ chấp nhận được, ngay cả khi kích cỡ của CSDL và số giao dịch tăng lên. Tăng khả năng của hệ thống bằng cách tăng sự song song cung cấp một con đường thuận tiện hơn cho sự phát triển hơn là thay thế một hệ tập trung bởi một máy nhanh hơn. Một số nhân tố ảnh hưởng xấu đến tính hiệu quả của hoạt động song song và có thể làm giảm cả tăng tốc độ và tăng quy mô là:

- *Chi phí khởi động (Startup Costs):* Có một chi phí khởi động kết hợp với sự khởi động một xử lý. Trong một hoạt động song song gồm hàng ngàn xử lý, thời gian khởi động (Startup time) có thể làm lu mờ thời gian xử lý hiện tại, ảnh hưởng bất lợi tới tăng tốc độ.
- *Sự giao thoa (Interference):* Các xử lý thực hiện trong một hệ song song thường truy nhập đến các tài nguyên chia sẻ, một sự giao thoa của mỗi xử lý mới khi nó cạnh tranh với các xử lý đang tồn tại trên các tài nguyên bị chiếm như bus hệ thống, đĩa chia sẻ, thậm chí cả đồng hồ. Hiện tượng này ảnh hưởng đến cả tăng tốc độ lẫn tăng quy mô.
- *Sự lệch (Skew):* Bằng cách chia một công việc thành các bước song song, ta làm giảm kích cỡ của bước trung bình. Tuy nhiên, thời gian phục vụ cho bước chậm nhất sẽ xác định thời gian phục vụ cho toàn bộ công việc. Thường khó có thể chia một công việc thành các phần cùng kích cỡ, như vậy cách mà kích cỡ được phân phối là bị lệch. Ví dụ: một công việc có kích cỡ 100 được chia thành 10 phần và sự phân chia này bị lệch, có thể có một số phần có kích cỡ nhỏ hơn 10 và một số nhiệm vụ có kích cỡ lớn hơn 10, giả sử trong đó có phần kích cỡ 20, độ tăng tốc nhận được bởi chạy các công việc song song chỉ là 5, thay vì 10.

- **Các mạng hợp nhất (Interconnection Network):** Các hệ thống song song gồm một tập hợp các thành phần (Processors, memory và các đĩa) có thể liên lạc với nhau thông qua một mạng hợp nhất. Các ví dụ về các mạng hợp nhất là:
  - *Bus:* Toàn bộ các thành phần hệ thống có thể gửi và nhận dữ liệu qua một bus liên lạc. Các kiến trúc bus làm việc tốt với một số nhỏ các processor. Tuy nhiên, chúng không có cùng quy mô khi tăng sự song song vì bus chỉ điều khiển liên lạc từ chỉ một thành phần tại một thời điểm.

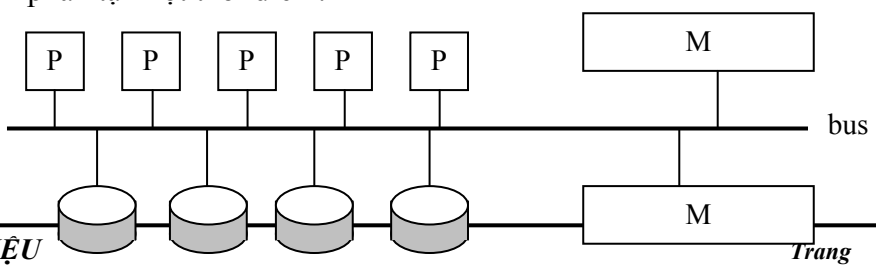
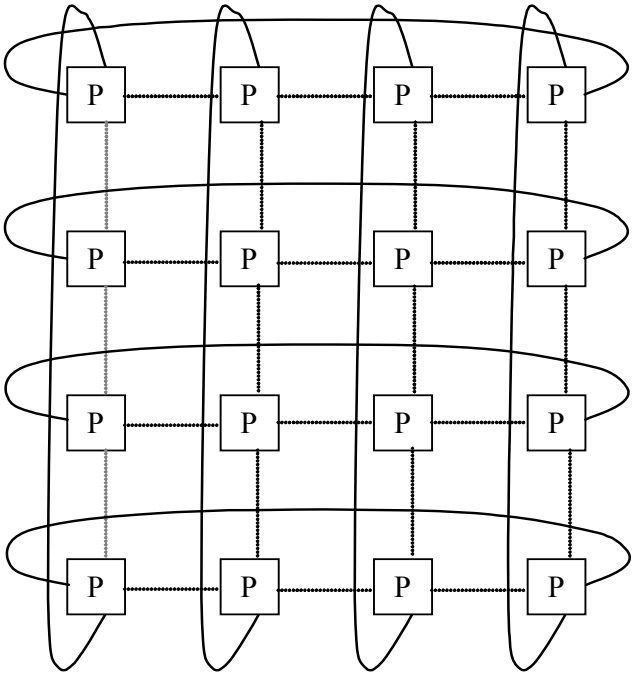


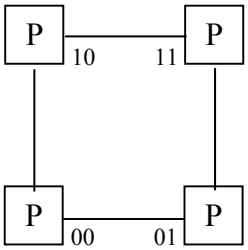
Figure 5

- *Lưới (Mesh):* Các thành phần được sắp xếp như các nút của một lưới, mỗi thành phần được nối với tất cả các thành phần kề với nó trong lưới. Trong một lưới hai chiều mỗi nút được nối với 4 nút kề.

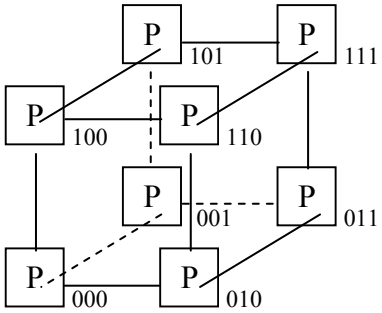


mạng hợp nhất hình lưới

Siêu lập phương một chiều



Siêu lập phương hai chiều



Siêu lập phương ba chiều

Figure 6

- *Siêu lập phương (Hypercube)*: Các thành phần được đánh số theo nhị phân, một thành phần được nối với thành phần khác nếu biểu diễn nhị nhân số của chúng sai khác nhau đúng một bit. Như vậy, mỗi một thành phần trong n thành phần được nối với log(n) thành phần khác. Có thể kiểm nghiệm được rằng trong một sự hợp nhất siêu lập phương một thông điệp từ một thành phần đến một thành phần khác đi quá nhiều nhất log(n) nối kết, còn trong lưới là $\sqrt{n}-1$

- **Các kiến trúc cơ sở dữ liệu song song:** Có một vài mô hình kiến trúc cho các máy song song:

  (Ký hiệu: P = processor, M = Memory, = đĩa)

  - *Bộ nhớ chia sẻ (Shared memory)*: Tất cả các các processor chia sẻ một bộ nhớ chung. Trong mô hình này các processor và các đĩa truy xuất một bộ nhớ chung, thường thông qua một bus hoặc một mạng hợp nhất. Thuận lợi của chia sẻ bộ nhớ là liên lạc giữa các processor là cực kỳ hiệu quả: dữ liệu trong bộ nhớ chia sẻ có thể được truy xuất bởi bất kỳ processor nào mà không phải di chuyển bởi phần mềm. Một processor có thể gửi thông điệp cho một processor khác bằng cách viết vào bộ nhớ chia sẻ, cách liên lạc này nhanh hơn nhiều so với các liên lạc khác. Tuy nhiên, các máy bộ nhớ chia sẻ không thể hỗ trợ nhiều hơn 64 processor vì nếu nhiều hơn, bus hoặc mạng hợp nhất sẽ trở nên dễ bị nghẽn (bottle-neck). Kiến trúc bộ nhớ chia sẻ thường có những cache lớn cho mỗi processor, như vậy việc tham khảo bộ nhớ chia sẻ có thể tránh được mỗi khi có thể.

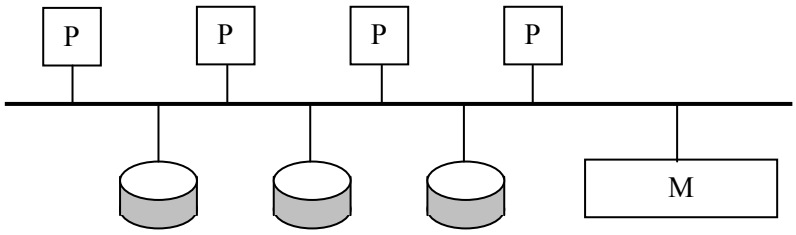


Figure 7

- *Đĩa chia sẻ (Shared disk):* Tất cả các processor chia sẻ đĩa chung. Mô hình này còn được gọi là cụm (cluster). Trong mô hình này tất cả các processor có thể truy xuất trực tiếp đến tất cả các đĩa thông qua một mạng hợp nhất, nhưng mỗi processor có bộ nhớ riêng.

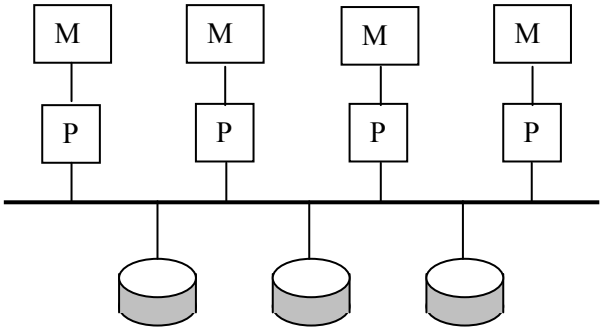


Figure 8

Kiến trúc này có các điểm thuận lợi là: thứ nhất, bus bộ nhớ không bị bottle-neck, thứ hai, cho một phương pháp rẻ để cung cấp một mức độ lượng thứ lỗi--một processor bị hỏng hóc, các processor khác có thể tiếp tục công việc của nó. Ta có thể tạo ra hệ thống con các đĩa tự lượng thứ lỗi bằng cách sử dụng kiến trúc RAID (được trình bày sau này). Vấn đề chính của chia sẻ đĩa là sự hợp nhất các hệ thống con các đĩa trở nên bottle-neck, đặc biệt trong tình huống CSDL truy xuất đĩa nhiều. So sánh với bộ nhớ chia sẻ, chia sẻ đĩa có thể hỗ trợ một số lượng processor lớn hơn, nhưng việc liên lạc giữa các processor chậm hơn.

- *Không chia sẻ (Shared nothing):* Các processor không chia sẻ bộ nhớ chung, cũng không chia sẻ đĩa chung. Trong hệ thống này mỗi nút của máy có một processor, bộ nhớ và một vài đĩa.

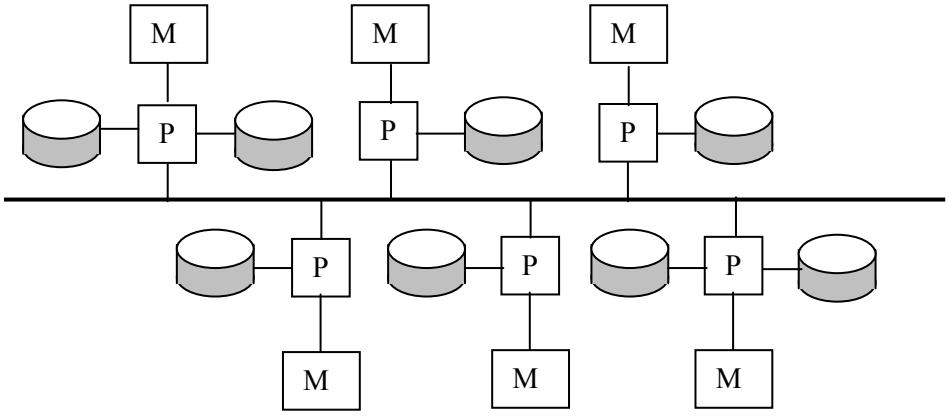


Figure 9

Các processor ở mỗi nút có thể liên lạc với các processor khác qua mạng hợp nhất tốc độ cao. Chức năng của một nút, như server, dữ liệu được chứa trên các đĩa của nó. Mô hình không chia sẻ gì chỉ có vấn đề về việc truy xuất các đĩa không cục bộ và việc truyền các quan hệ kết quả qua mạng. Hơn nữa, đối với các hệ thống không chia sẻ gì, các mạng hợp nhất thường được thiết kế để có thể tăng quy mô, sao cho khả năng truyền của chúng tăng khi các nút mới được thêm vào.

- *Phân cấp (hierarchical):* Mô hình này là một sự lai kiểu của các kiến trúc trước.

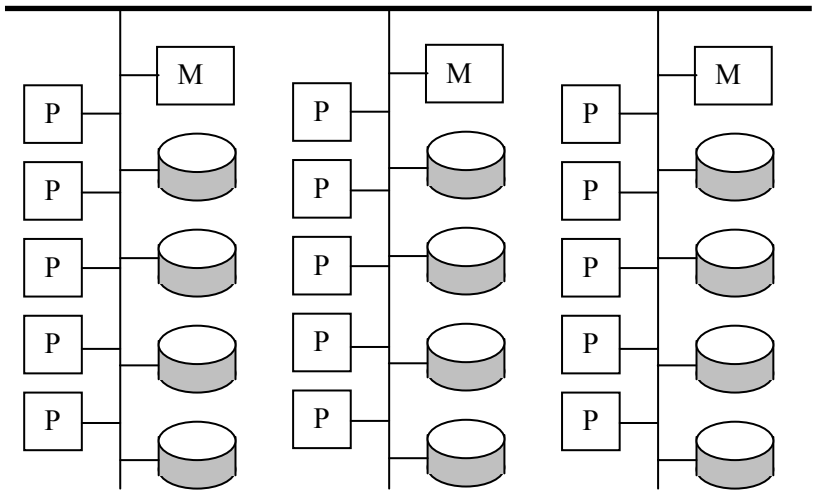


Figure 10

Kiến trúc này tổ hợp các đặc trưng của các kiến trúc chia sẻ bộ nhớ, chia sẻ đĩa và không chia sẻ gì. Ở mức cao nhất, hệ thống bao gồm những nút được nối bởi mạng hợp nhất và không chia sẻ đĩa cũng như bộ nhớ với nút khác. Như vậy, mức cao nhất là kiến trúc không chia sẻ gì. Mỗi nút của hệ thống có thể là hệ thống chia sẻ bộ nhớ với một vài processor. Kế tiếp, mỗi nút có thể là một hệ thống chia sẻ đĩa. Mỗi một hệ thống chia sẻ đĩa lại có thể là một hệ thống chia sẻ bộ nhớ... Như vậy, hệ thống có thể được xây dựng như một sự phân cấp.

CÁC HỆ THỐNG PHÂN TÁN (Distributed Systems):

Trong một hệ thống CSDL phân tán, CSDL được lưu trữ trên một vài máy tính. Các máy tính trong một hệ thống phân tán liên lạc với một máy khác qua nhiều dạng phương tiện liên lạc khác nhau: mạng tốc độ cao, đường điện thoại... Chúng không chia sẻ bộ nhớ cũng như đĩa. Các máy tính trong hệ thống phân tán có thể rất đa dạng về kích cỡ cũng như chức năng: từ các workstation đến các mainframe. Các máy tính trong hệ thống phân tán được tham chiếu bởi một số các tên khác nhau: site , node -- phụ thuộc vào ngữ cảnh mà máy được đề cập. Ta sẽ sử dụng thuật ngữ **site** để nhấn mạnh sự phân tán vật lý của các hệ thống này.

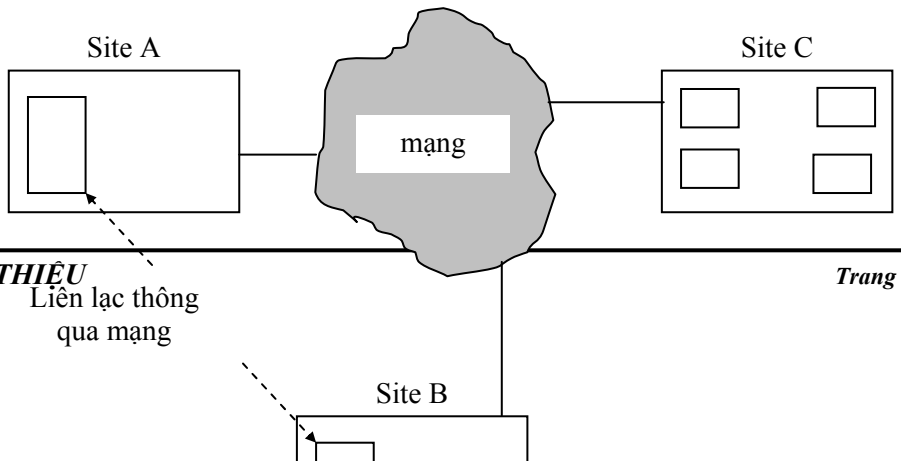


Figure 11

Sự sai khác chính giữa CSDL song song không chia sẻ gì và hệ thống phân tán là CSDL phân tán được tách biệt về mặt địa lý, được quản trị tách biệt và có một sự hợp nhất chậm. Hơn nữa, trong hệ thống phân tán người ta phân biệt giữa các giao dịch cục bộ (local) và toàn thể (global). Giao dịch cục bộ là một giao dịch truy xuất dữ liệu trong một **site** tại đó giao dịch đã được khởi xướng. Giao dịch toàn thể là một giao dịch mà nó hoặc truy xuất dữ liệu trong một site từ một site khác tại đó nó được khởi xướng hoặc truy xuất dữ liệu trong một vài site khác nhau.

BÀI TẬP CHƯƠNG I

I.1 Bốn điểm khác nhau chính giữa một hệ thống xử lý file và một hệ quản trị CSDL là gì ?

I.2 Giải thích sự khác nhau giữa độc lập dữ liệu vật lý và độc lập dữ liệu logic

I.3 Liệt kê năm nhiệm vụ của nhà quản trị CSDL. Đối với mỗi nhiệm vụ, giải tích rõ những vấn đề nảy sinh nếu nhiệm vụ đó không được hoàn thành.

I.4 Năm chức năng chính của nhà quản trị CSDL là gì ?

I.5 Sử dụng một mảng hai chiều kích cỡ n x m như một ví dụ để minh hoạ sự khác nhau:

1. giữa ba mức trừu tượng dữ liệu
2. giữa sơ đồ và thể hiện

I.6 Trong các hệ thống client-server tiêu biểu, máy server thường mạnh hơn máy client rất nhiều: Có bộ xử lý nhanh hơn thậm chí có thể có nhiều bộ xử lý, có bộ nhớ lớn hơn, có dung lượng đĩa lớn hơn. Ta xét một kịch bản trong đó các máy client và các máy server là mạnh như

nhau. Xây dựng một hệ thống client-server theo kịch bản như vậy có ưu nhược điểm gì ? Kịch bản nào phù hợp hơn với kiến trúc server dữ liệu ?

I.7 Giả sử một giao dịch được viết trong C với SQL nhúng, và khoảng 80% thời gian được dùng cho code SQL, 20% còn lại cho code C. Nếu song song được dùng chỉ cho code SQL, Tăng tốc độ có thể đạt tới bao nhiêu ? Giải thích.

I.8 Những nhân tố nào chống lại việc tăng quy mô tuyến tính trong một hệ thống xử lý giao dịch ? Nhân tố nào là quan trọng nhất trong mỗi một kiến trúc sau: bộ nhớ chia sẻ, đĩa chia sẻ, không chia sẻ gì ?

I.9 Xét một mạng dựa trên đường điện thoại quay số tự động, trong đó các site liên lạc theo định kỳ, ví dụ hàng đêm. Các mạng như vậy thường được cấu hình với một site server và nhiều site client. Các site client chỉ nói với server và trao đổi dữ liệu với client khác bởi lưu dữ liệu tại server và lấy dữ liệu được lưu trên server bởi client khác. Ưu, nhược điểm của kiến trúc như vậy là gì ?

CHƯƠNG II

SQL

MỤC ĐÍCH

Giới thiệu một hệ CSDL chuẩn, SQL, các thành phần cơ bản của của nó.

YÊU CẦU

Hiểu các thành phần cơ bản của SQL-92

Hiểu và vận dụng phương pháp "dịch" từ câu vấn tin trong ngôn ngữ tự nhiên sang ngôn ngữ SQL và ngược lại

Hiểu và vận dụng cách thêm (xen), xóa dữ liệu

SQL là ngôn ngữ CSDL quan hệ chuẩn, gốc của nó được gọi là Sequel. SQL là viết tắt của **S**tructured **Q**uery **L**anguage. Có nhiều phiên bản của SQL. Phiên bản được trình bày trong giáo trình này là phiên bản chuẩn SQL-92.

SQL có các phần sau:

- ***Ngôn ngữ định nghĩa dữ liệu (DDL)***. **DDL** của SQL cung cấp các lệnh để định nghĩa các sơ đồ quan hệ, xoá các quan hệ, tạo các chỉ mục, sủa đổi các sơ đồ quan hệ
- ***Ngôn ngữ thao tác dữ liệu tương tác (Interactive DML)*****. IDML** bao gồm một ngôn ngữ dựa trên cả đại số quan hệ lẫn phép tính quan hệ bộ. Nó bao hàm các lệnh xen các bộ, xoá các bộ, sửa đổi các bộ trong CSDL
- ***Ngôn ngữ thao tác dữ liệu nhúng (Embedded DML)*****.** Dạng SQL nhúng được thiết kế cho việc sử dụng bên trong các ngôn ngữ lập trình mục đích chung (genaral-purpose programming languages) như PL/I, Cobol, Pascal, Fortran, C.
- ***Đinh nghĩa view***. **DDL SQL** cũng bao hàm các lệnh để định nghĩa các view.
- ***Cấp quyền (Authorization)*****. DDL SQL** bao hàm cả các lệnh để xác định các quyền truy xuất dến các quan hệ và các view
- ***Tính toàn vẹn (Integrity)*****. DDL SQL** chứa các lệnh để xác định các ràng buộc toàn vẹn mà dữ liệu được lưu trữ trong CSDL phải thoả.
- ***Điều khiển giao dịch*****.** SQL chứa các lệnh để xác định bắt đầu và kết thúc giao dịch, cũng cho phép chốt tường minh dữ liệu để điều khiển cạnh tranh

Các ví dụ minh hoạ cho các câu lệnh SQL được thực hiện trên các sơ đồ quan hệ sau:

- **Branch_schema = (Branch_name, Branch_city, Assets)**: Sơ đồ quan hệ chi nhánh nhà băng gồm các thuộc tính Tên chi nhánh (Branch_name), Thành phố (Branch_city), tài sản (Assets)
- **Customer_schema = (Customer_name, Customer_street, Customer_city)**: Sơ đồ quan hệ Khách hàng gồm các thuộc tính Tên khách hàng (Customer_name), phố (Customer_street), thành phố (Customer_city)
- **Loan_schema = (Branch_name, loan_number, amount)**: Sơ đồ quan hệ cho vay gồm các thuộc tính Tên chi nhánh, số cho vay (Loan_number), số lượng (Amount)
- **Borrower_schema = (Customer_name, loan_number)**: Sơ đồ quan hệ người mượn gồm các thuộc tính Tên khách hàng, số cho vay
- **Account_schema = (Branch_name, account_number, balance)**: Sơ đồ quan hệ tài khoản gồm các thuộc tính Tên chi nhánh, số tài khoản (Account_number), số cân đối (Balance: dư nợ/có)
- **Depositor_schema = (Customer_name, account_number)**: Sơ đồ người gửi gồm các thuộc tính Tên khách hàng, số tài khoản

Cấu trúc cơ sở của một biểu thức SQL gồm ba mệnh đề: **SELECT, FROM và WHERE**

- ♦ Mệnh đề **SELECT** tương ứng với phép chiếu trong đại số quan hệ, nó được sử dụng để liệt kê các thuộc tính mong muốn trong kết quả của một câu vấn tin
- ♦ Mệnh đề **FROM** tương ứng với phép tích Đề các , nó nó liệt kê các quan hệ được quét qua trong sự định trị biểu thức
- ♦ Mệnh đề **WHERE** tương ứng với vị từ chọn lọc, nó gồm một vị từ chứa các thuộc tính của các quan hệ xuất hiện sau FROM

Một câu vấn tin kiểu mẫu có dạng:

SELECT A_1, A_2, ..., A_k
FROM R_1, R_2, ..., R_m
WHERE P

trong đó A_i là các thuộc tính (Attribute), R_j là các quan hệ (Relation) và P là một vị từ (Predicate). Nếu thiếu WHERE vị từ P là TRUE.

Kết quả của một câu vấn tin SQL là một quan hệ.

MỆNH ĐỀ SELECT

Ta tìm hiểu mệnh đề SELECT bằng cách xét một vài ví dụ:

"Tìm kiếm tất cả các tên các chi nhánh trong quan hệ cho vay (loan)":

SELECT Branch_name
FROM Loan;

Kết quả là một quan hệ gồm một thuộc tính Tên chi nhánh (Branch_name)

Nếu muốn quan hệ kết quả không chứa các tên chi nhánh trùng nhau:

SELECT **DISTINCT** Branch_name
FROME Loan;

Từ khoá ALL được sử dụng để xác định tường minh rằng các giá trị trùng không bị xoá và nó là mặc nhiên của mệnh đề SELECT.

Ký tự * được dùng để chỉ tất cả các thuộc tính:

SELECT *
FROM Loan;

Sau mệnh đề SELECT cho phép các biểu thức số học gồm các phép toán +, -, *, / trên các hằng hoặc các thuộc tính:

```
SELECT Branch_name, Loan_number, amount * 100
FROM Loan;
```

MỆNH ĐỀ WHERE

"Tìm tất cả các số cho vay ở chi nhánh tên Perryridge với số lượng vay lớn hơn1200$"

```
SELECT Loan_number
FROM Loan
WHERE Branch_name = 'Perryridge' AND Amount > 1200;
```

SQL sử dụng các phép nối logic: **NOT, AND, OR**. Các toán hạng của các phép nối logic có thể là các biểu thức chứa các toán tử so sánh =, >=, <>, <, <=.

Toán tử so sánh **BETWEEN** được dùng để chỉ các giá trị nằm trong một khoảng:

```
SELECT Loan_number
FROM Loan
WHERE Amount  BETWEEN 50000 AND 100000;
```

≈

```
SELECT  Loan_number
FROM Loan
WHERE Amount >= 50000 AND Amount <= 100000;
```

Ta cũng có thể sử dụng toán tử **NOT BETWEEN**.

MỆNH ĐỀ FROM

"Trong tất cả các khách hàng có vay ngân hàng tìm tên và số cho vay của họ"

```
SELECT DISTINCT Customer_name, Borrower.Loan_number
FROM Borrower, Loan
WHERE Borrower.Loan_number = Loan.Loan_number;
```

SQL sử dụng cách viết <tên quan hệ >.< tên thuộc tính > để che dấu tính lập lờ trong trường hợp tên thuộc tính trong các sơ đồ quan hệ trùng nhau.

"Tìm các tên và số cho vay của tất cả các khách hàng có vay ở chi nhánh Perryridge"

```
SELECT Customer_name, Borrower.Loan_number
FROM Borrower, Loan
WHERE Borrower.Loan_number = Loan.Loan_number AND
        Branch_name ='Perryridge';
```

CÁC PHÉP ĐỔI TÊN

SQL cung cấp một cơ chế đổi tên cả tên quan hệ lẫn tên thuộc tính bằng mệnh đề dạng:

< tên cũ > **AS** < tên mới >

mà nó có thể xuất hiện trong cả mệnh đề SELECT lẫn FROM

```
SELECT DISTINCT Customer_name, Borrower.Loan_number
FROM Borrower, Loan
WHERE Borrower.Loan_number = Loan.Loan_number AND
        Branch_name ='Perryridge';
```

Kết quả của câu vấn tin này là một quan hệ hai thuộc tính: Customer_name, Loan_number

Đổi tên thuộc tính của quan hệ kết quả:

```
SELECT Customer_name, Borrower.Loan_number AS Loan_Id
FROM Borrower, Loan
```

```
        WHERE Borrower.Loan_number = Loan.Loan_number AND
              Branch_name =’Perryridge’;
```

CÁC BIẾN BỘ (Tuple Variables)

Các biến bộ được định nghĩa trong mệnh đề FROM thông qua sử dụng mệnh đề AS:

```
SELECT DISTINCT Customer_name, T.Loan_number
FROM Borrower  AS  T, Loan  AS  S
WHERE T.Loan_number = S.Loan_number AND
      Branch_name =’Perryridge’;
```

“Tìm các tên của tất cả các chi nhánh có tài sản lớn hơn ít nhất một chi nhánh ở Brooklyn“

```
SELECT DISTINCT T.branch_name
FROM  Branch AS T, Banch AS S
WHERE T.assets > S.assets AND S.Branch_City = ‘Brooklyn’
```

SQL92 cho phép sử dụng các viết (v_1, v_2, ..., vn) để ký hiệu một n-bộ với các giá trị v_1, v_2, ..., vn. Các toán tử so sánh có thể được sử dụng trên các n-bộ và theo thứ tự tự điển. Ví dụ (a1, b1) <= (a2, b2) là đúng nếu ($a_1 < b_1$) OR (($a_1 = b_1$) AND ($a_2 < b_2$)).

CÁC PHÉP TOÁN TRÊN CHUỖI

Các phép toán thường được dùng nhất trên các chuỗi là phép đối chiếu mẫu sử dụng toán tử **LIKE**. Ta mô tả các mẫu dùng hai ký tự đặc biệt:

- ký tự phần trăm (%): ký tự % tương xứng với ***chuỗi con*** bất kỳ
- ký tự gạch nối (_): ký tự gạch nối tương xứng với ***ký tự*** bất kỳ.
  - ‘Perry%’ tương xứng với bất kỳ chuỗi nào bắt đầu bởi ‘Perry’
  - ‘%idge%’ tương xứng với bất kỳ chuỗi nào chứa ‘idge’ như chuỗi con
  - ‘___’ tương xứng với chuỗi bất kỳ có đúng ba ký tự
  - ‘___%’ tương xứng với chuỗi bất kỳ có ít nhất ba ký tự

"Tìm tên của tất cả các khách hàng tên phố của họ chứa chuỗi con ‘Main’

```
SELECT Customer_name
FROM Customer
WHERE Customer_street LIKE ‘%Main%’
```

Nếu trong chuỗi mẫu có chứa các ký tự % _ \ , để tránh nhầm lẫn ký tự với"dấu hiệu thay thế", SQL sử dụng cách viết: ký tự escape (\) đứng ngay trước ký tự"đặc biệt". Ví dụ nếu chuỗi mẫu là ab%cd được viết là ‘ab\%cd’, chuỗi mẫu là ab_cde được viết là ‘ab_cde’, chuỗi mẫu là ab\cd được viết là ‘ab\\cd’

SQL cho phép đối chiếu không tương xứng bằng cách sử dụng **NOT LIKE**

SQL cũng cho phép các hàm trên chuỗi: nối hai chuỗi (|), trích ra một chuỗi con, tìm độ dài chuỗi, biến đổi một chuỗi chữ thường sang chuỗi chữ hoa và ngược lại ...

THỨ TỰ TRÌNH BÀY CÁC BỘ (dòng)

Mệnh đề ORDER BY tạo ra sự trình bày các dòng kết quả của một câu vấn tin theo một trình tự. Để liết kê theo thứ tự alphabet tất cả các khách hàng có vay ở chi nhánh Perryridge:

```
SELECT  DISTINCT  Customer_name
FROM Borrower, Loan
WHERE Borrower.Loan_number = Loan.Loan_number AND
            Branch_name = ‘Perryridge’
ORDER BY Customer_name;
```

Mặc nhiên, mệnh đề ORDER BY liệt kê theo thứ tự tăng, tuy nhiên ta có thể làm liệt kê theo thứ tự **giảm**/**tăng** bằng cách chỉ rõ bởi từ khoá **DESC**/ **ASC**

```
SELECT *
FROM Loan
ORDER BY Amount DESC, Loan_number ASC;
```

CÁC PHÉP TOÁN TẬP HỢP

SQL92 có các phép toán **UNION, INTERSECT, EXCEPT** chúng hoạt động giống như các phép toán hợp, giao, hiệu trong đại số quan hệ. Các quan hệ tham gia vào các phép toán này phải tương thích (có cùng tập các thuộc tính).

- Phép toán **UNION**

"tìm kiếm tất cả các khách hàng có vay, có tài khoản hoặc cả hai ở ngân hàng"

```
(SELECT  Customer_name
 FROM Depositor)
UNION
(SELECT Customer_name
 FROM Borrower);
```

Phép toán hợp UNION tự động loại bỏ các bộ trùng, nếu ta muốn giữ lại các bộ trùng ta phải sử dụng **UNION ALL**

```
(SELECT  Customer_name
 FROM Depositor)
UNION ALL
(SELECT Customer_name
 FROM Borrower);
```

- Phép toán **INTERSECT**

"tìm kiếm tất cả các khách hàng có vay và cả một tài khoản tại ngân hàng"

```
(SELECT DISTINCT Customer_name
 FROM Depositor)
INTERSECT
(SELECT DISTINCT Customer_name
 FROM Borrower);
```

Phép toán INTERESCT tự động loại bỏ các bộ trùng, Để giữ lại các bộ trùng ta sử dụng **INTERSECT ALL**

```
(SELECT  Customer_name
 FROM Depositor)
INTERSECT ALL
(SELECT  Customer_name     FROM Borrower);
```

- Phép toán **EXCEPT**

"Tìm kiếm tất cả các khách hàng có tài khoản nhưng không có vay tại ngân hàng"

```
(SELECT Customer_name
 FROM Depositor)
EXCEPT
(SELECT Customer_name
 FROM Borrower);
```

EXCEPT tự động loại bỏ các bộ trùng, nếu muốn giữ lại các bộ trùng phải dùng **EXCEPT ALL**

```
(SELECT Customer_name
 FROM Depositor)
EXCEPT ALL
```

```
(SELECT Customer_name
 FROM Borrower);
```

CÁC HÀM TÍNH GỘP

SQL có các hàm tính gộp (aggregate functions):

- Tính trung bình (Average): **AVG**()
- Tính min : **MIN**()
- Tính max: **MAX**()
- Tính tổng: **SUM**()
- Đếm: **COUNT**()

Đối số của các hàm AVG và SUM phải là **kiểu dữ liệu số**

"Tìm số cân đối tài khoản trung bình tại chi nhánh Perryridge"

```
SELECT AGV(balace)
FROM Account
WHERE Branch_name = 'Perryridge';
```

SQL sử dụng mệnh để **GROUP BY** vào mục đích nhóm các bộ có cùng giá trị trên các thuộc tính nào đó

"Tìm số cân đối tài khoản trung bình tại mỗi chi nhánh ngân hàng"

```
SELECT Branch_name, AVG(balance)
FROM Account
GROUP BY Branch_name;
```

"Tìm số các người gửi tiền đối với mỗi chi nhánh ngân hàng"

```
SELECT Branch_name, COUNT(DISTINCT Customer_name)
FROM Depositor, Account
WHERE Depositor.Account_number = Account.Acount_number
GROUP BY Branch_name
```

Giả sử ta muốn liệt kê các chi nhánh ngân hàng có số cân đối trung bình lớn hơn 1200$. Điều kiện này không áp dụng trên từng bộ, nó áp dụng trên từng nhóm. Để thực hiện được điều này ta sử dụng mệnh đề **HAVING** của SQL

```
SELECT Branch_name, AVG(balance)
FROM Account
GROUP BY Branch_name
HAVING AGV(Balance) > 1200$;
```

Vị từ trong mệnh đề HAVING được áp dụng sau khi tạo nhóm, như vậy hàm AVG có thể được sử dụng

"Tìm số cân đối đối với tất cả các tài khoản"

```
SELECT AVG(Balance)   FROM Account;
```

"Đếm số bộ trong quan hệ Customer"

```
SELECT Count(*)     FROM Customer;
```

SQL không cho phép sử dụng **DISTINCT** với **COUNT(*)**, nhưng cho phép sử dụng DISTINCT với MIN và MAX.

Nếu WHERE và HAVING có trong cùng một câu vấn tin, vị từ sau WHERE được áp dụng trước. Các bộ thoả mãn vị từ WHERE được xếp vào trong nhóm bởi GROUP BY, mệnh đề HAVING (nếu có) khi đó được áp dụng trên mỗi nhóm. Các nhóm không thoả mãn mệnh đề HAVING sẽ bị xoá bỏ.

"Tìm số cân đối trung bình đối với mỗi khách hàng sống ở Harrison và có ít nhất ba tài khoản"

```
SELECT Depositor.Customer_name, AVG(Balance)
```

```
FROM Depositor, Account, Customer
WHERE Depositor.Account_number = Account.Account_number AND
      Depositor.Customer_name = Customer.Customer_name AND
      Customer.city ='Harrison'
GROUP BY Depositor.Customer_name
HAVING COUNT(DISTINT Depositor.Account_number) >= 3;
```

CÁC GIÁ TRỊ NULL

SQL cho phép sử dụng các giá trị null để chỉ sự vắng mặt thông tin tạm thời về giá trị của một thuộc tính. Ta có thể sử dụng từ khoá đặc biệt **null** trong vị từ để thử một giá trị null.

"Tìm tìm tất cả các số vay trong quan hệ Loan với giá trị Amount là null"

```
SELECT Loan_number
FROM Loan
WHERE Amount is null
```

Vị từ **not null** thử các giá trị không rỗng

Sử dụng giá trị null trong các biểu thức số học và các biểu thức so sánh gây ra một số phiền phức. Kết quả của một biểu thức số học là null nếu một giá trị input bất kỳ là null. Kết quả của một biểu thức so sánh chứa một giá trị null có thể được xem là false. SQL92 xử lý kết quả của một phép so sánh như vậy như là một giá trị **unknown**, là một giá trị không là **true** mà cũng không là **false**. SQL92 cũng cho phép thử kết quả của một phép so sánh là unknown hay không. Tuy nhiên, trong hầu khắp các trường hợp, unknown được xử lý hoàn toàn giống như false.

Sự tồn tại của các giá trị null cũng làm phức tạp việc sử lý các toán tử tính gộp. Giả sử một vài bộ trong quan hệ Loan có các giá trị null trên trường Amount. Ta xét câu vấn tin sau:

```
SELECT SUM(Amount)
FROM LOAN
```

Các giá trị được lấy tổng trong câu vấn tin bao hàm cả các trị null. Thay vì tổng là null, SQL chuẩn thực hiện phép tính tổng bằng cách bỏ qua các giá trị input là null.

Nói chung, các hàm tính gộp tuân theo các quy tắc sau khi xử lý các giá trị null: Tất cả các hàm tính gộp ngoại trừ COUNT(*) bỏ qua các giá trị input null. Khi các giá trị nul bị bỏ qua, tập các giá trị input có thể là rõng. COUNT() của một tập rỗng được định nghĩa là 0. Tất cả các hàm tính gộp khác trả lại giá trị null khi áp dụng trên tập hợp input rỗng.

CÁC CÂU VẤN TIN CON LỒNG NHAU (Nested Subqueries)

SQL cung cấp một cơ chế lòng nhau của các câu vấn tin con. Một câu vấn tin con là một biểu thức SELECT-FROM-WHERE được lồng trong một caau vấn tin khác. Các câu vấn tin con thường được sử dụng để thử quan hệ thành viên tập hợp, so sánh tập hợp và bản số tập hợp.

QUAN HỆ THÀNH VIÊN TẬP HỢP (Set relationship)

SQL đưa vào các phép tính quan hệ các phép toán cho phép thử các bộ có thuộc một quan hệ nào đó hay không. Liên từ **IN** thử quan hệ thành viên này. Liên từ NOT IN thử quan hệ không là thành viên.

"Tìm tất cả các khách hàng có cả vay lẫn một tài khoản tại ngân hàng"

Ta đã sử dụng INTERSECTION để viết câu vấn tin này. Ta có thể viết câu vấn tin này bằng các sử dụng IN như sau:

```
SELECT DISTINCT Customer_name
```

```
FROM Borrower
WHERE Customer_name IN (        SELECT Customer_name
                                FROM Depositor)
```

Ví dụ này thử quan hệ thành viên trong một quan hệ một thuộc tính. SQL92 cho phép thử quan hệ thành viên trên một quan hệ bất kỳ.

"Tìm tất cả các khách hàng có cả vay lãn một tài khoản ở chi nhánh Perryridge"

Ta có thể viết câu truy vấn như sau:

```
SELECT DISTINCT Customer_name
FROM Borrower, Loan
WHERE       Borrower. Loan_number = Loan.Loan_number AND
            Branch_name = 'Perryridge'   AND
            (Branch_name. Customer_name IN
                  (SELECT Branch_name, Customer_name
                   FROM Depositor, Account
                   WHERE       Depositor.Account_number   =
                               Account.Account_number )
```

"Tìm tất cả các khách hàng có vay ngân hàng nhưng không có tài khoản tại ngân hàng"

```
SELECT DISTINCT Customer_name
FROM borrower
WHERE Customer_name NOT IN (  SELECT Customer_name
                              FROM Depositor)
```

Các phép toán IN và NOT IN cũng có thể được sử dụng trên các tập hợp liệt kê:

```
SELECT DISTINCT Customer_name
FROM borrower
WHERE Customer_name NOT IN ('Smith', 'Jone')
```

SO SÁNH TẬP HỢP (Set Comparision)

"Tìm tên của tất cả các chi nhánh có tài sản lớn hơn ít nhất một chi nhánh đóng tại Brooklyn"

```
SELECT DISTINCT Branch_name
FROM Branch AS T, Branch AS S
WHERE T.assets > S.assets AND S.branch_city = 'Brooklyn'
```

Ta có thể viết lại câu vấn tin này bằng cách sử dụng mệnh đề"lớn hơn ít nhất một"trong SQL

- **SOME** :

```
SELECT Branch_name
FROM Branch
WHERE Assets > SOME (    SELECT Assets
                         FROM Branch
                         WHERE Branch_city ='Brooklyn')
```

Câu vấn tin con

```
( SELECT Assets
  FROM Branch
  WHERE Branch_city ='Brooklyn')
```

sinh ra tập tất cả các Assets của tất cả các chi nhánh đóng tại Brooklyn. So sánh > SOME trong mệnh đề WHERE nhận giá trị đúng nếu giá trị Assets của bộ được xét lớn hơn ít nhất một trong các giá trị của tập hợp này.

SQL cũng có cho phép các so sánh **< SOME, >= SOME, <= SOME, = SOME, <> SOME**

- **ALL**

"Tìm tất cả các tên của các chi nhánh có tài sản lớn hơn tài sản của bất kỳ chi nhánh nào đóng tại Brooklyn"

```
SELECT Branch_name
FROM Branch
WHERE Assets > ALL (    SELECT Assets
                        FROM Branch
                        WHERE Branch_citty = 'Brooklyn')
```

SQL cũng cho phép các phép so sánh: **< ALL, <= ALL, > ALL, >= ALL, = ALL, <> ALL**.

"Tìm chi nhánh có số cân đối trung bình lớn nhất"

SQL không cho phép hợp thành các hàm tính gộp, như vậy MAX(AVG (...)) là không được phép. Do vậy, ta phải sử dụng câu vấn tin con như sau:

```
SELECT Branch_name
FROM Account
GROUP BY Branch_name
HAVING AVG (Balance) >= ALL ( SELECT AVG (balance)
                              FROM Account
                              GROUP BY Branch_name)
```

THỬ CÁC QUAN HỆ RỖNG

"tìm tất cả các khách hàng có cả vay lẫn tài khoản ở ngân hàng"

```
SELECT Customer_name
FROM Borrower
WHERE  EXISTS ( SELECT *
                FROM Depositor
                WHERE Depositor.Customer_name = Borrower.Customer_name)
```

Cấu trúc **EXISTS** trả lại giá trị true nếu quan hệ kết quả của câu vấn tin con không rỗng. SQL cũng cho phép sử dụng cấu trúc **NOT EXISTS** để kiểm tra tính không rỗng của một quan hệ.

"Tìm tất cả các khách hàng có tài khoản tại mỗi chi nhánh đóng tại Brooklyn"

```
SELECT DISTINCT S.Customer_name
FROM Depositor  AS S
WHERE NOT EXISTS      ( ( SELECT Branch_name
                          FROM Branch
                          WHERE Branch_city = 'Brooklyn')
                        EXCEPT
                          ( SELECT R.branch_name
                            FROM Depositor AS T, Account AS R
                            WHERE    T.Acoount_number = R.Account_number
                            AND  S.Customer_name = T.Customer_name) )
```

THỬ KHÔNG CÓ CÁC BỘ TRÙNG

SQL đưa vào cấu trúc **UNIQUE** để kiểm tra việc có bộ trùng trong quan hệ kết quả của một câu vấn tin con.

"Tìm tất cả khách hàng chỉ có một tài khoản ở chi nhánh Perryridge"

```
SELECT T.Customer_name
FROM Depositor AS T
WHERE UNIQUE ( SELECT   R.Customer_name
               FROM Account, Depositor AS  R
               WHERE T.Customer_name = R.Customer_name AND
                     R.Account_number    =    Account.Acount_number
                     AND   Account.Branch_name = 'Perryridge')
```

Ta có thể thử sự tồn tại của các bộ trùng trong một vấn tin con bằng cách sử dụng cấu trúc **NOT UNIQUE**

"Tìm tất cả các khách hàng có ít nhất hai tài khoản ở chi nhánh Perryridge"

```
SELECT DISTINCT T.Customer_name
FROM  Account, Depositor AS T
WHERE NOT UNIQUE ( SELECT R.Customer_name
                   FROM Account, Depositor AS  R
                   WHERE T.Customer_name=R.Customer_name
                AND   R.Account_number = Account.Account_number
                AND   Account.Branch_name = 'Perryridge')
```

UNIQUE trả lại giá rị false khi và chỉ khi quan hệ có hai bộ trùng nhau. Nếu hai bộ t_1, t_2 có ít nhất một trường null, phép so sánh $t_1 = t_2$ cho kết quả false. Do vậy UNIQUE có thể trả về giá trị true trong khi quan hệ có nhiều bộ trùng nhau nhưng chứa trường giá trị null !

QUAN HỆ DẪN XUẤT

SQL92 cho phép một biểu thức vấn tin con được dùng trong mệnh đề FROM. Nếu biểu thức như vậy được sử dụng, quan hệ kết quả phải được cho một cái tên và các thuộc tính có thể được đặt tên lại (bằng mệnh đề AS)

Ví dụ câu vấn tin con:

```
(SELECT  Branch_name, AVG(Balance)
  FROM Account
  GROUP BY Branch_name)
AS result (Branch_name, Avg_balace)
```

Sinh ra quan hệ gồm tên của tất cả các chi nhánh, và số cân đối trung bình tương ứng. Quan hệ này được đặt tên là result với hai thuộc tính Branch_name và Avg_balance.

"Tìm số cân đối tài sản trung bình của các chi nhánh tại đó số cân đối tài khoản trung bình lớn hơn 1200$"

```
SELECT Branch_name, avg_balance
FROM  ( SELECT  Branch_name, AVG(Balance)
          FROM Account
        GROUP BY Branch_name)
        AS result (Branch_name, Avg_balace)
WHERE avg_balance > 1200
```

VIEWS

Trong SQL, để định nghĩa view ta sử dụng lệnh **CREATE VIEW**. Một view phải có một tên.

CREATE VIEW < tên view > AS < Biểu thức vấn tin >

"Tạo một view gồm các tên chi nhánh, tên của các khách hàng có hoặc một tài khoản hoặc vay ở chi nhánh này"

Giả sử ta muốn đặt tên cho view này là All_customer.

```
CREATE VIEW All_customer        AS
(       SELECT Branch_name, Customer_name
        FROM Depositor, Account
        WHERE Depositor.Account_number = Account.Account_number )
UNION
(       SELECT Branch_name, Customer_name
        FROM Borrower, Loan
        WHERE Borrower.Loan_number = Loan.Loan_number)
```

Tên thuộc tính của một view có thể xác định một cách tường minh như sau:

```
CREATE VIEW Branch_total_loan (Branch_name, Total_loan)      AS
        ( SELECT Branch_name, sum(Amount)
          FROM Loan
          GROUP BY Branch_name)
```

Một view là một quan hệ, nó có thể tham gia vào các câu vấn tin với vai trò của một quan hệ.

```
SELECT Customer_name
FROM All_customer
WHERE Branch_name = 'Perryridge'
```

Một câu vấn tin phức tạp sẽ dễ hiểu hơn, dễ viết hơn nếu ta cấu trúc nó bằng cách phân tích nó thành các view nhỏ hơn và sau đó tổ hợp lại.

Định nghĩa view được giữ trong CSDL đến tận khi một lệnh DROP VIEW < tên view > được gọi.

Trong chuẩn SQL 3 hiện đang được phát triển bao hàm một đề nghị hỗ trợ những view tạm không được lưu trong CSDL.

SỬA ĐỔI CƠ SỞ DỮ LIỆU

DELETE
INSERT
UPDATE

XÓA (Delete)

Ta chỉ có thể xoá nguyên vẹn một bộ trong một quan hệ, không thể xoá các giá trị của các thuộc tính. Biểu thức xoá trong SQL là:

```
DELETE FROM  r
[WHERE  P]
```

Trong đó p là một vị từ và r là một quan hệ.

Lệnh DELETE duyệt qua tất cả các bộ t trong quan hệ r, nếu P(t) là true, DELETE xoá t khỏi r. Nếu không có mệnh đề WHERE, tất cả các bộ trong r bị xoá.

Lệnh DELETE chỉ hoạt động trên một quan hệ.

- DELETE FROM Loan = Xoá tất cả các bộ của quan hệ Loan
- DELETE FROM Depositor WHERE Customer_name = 'Smith'
- DELETE FROM Loan
  WHERE Amount BETWEEN 1300 AND 1500
- DELETE FROM Account
  WHERE Branch_name IN (SELECT Branch_name
  FROM Branch
  WHERE Branch_city = 'Brooklyn')
- DELETE FROM Account
  WHERE Balance < (SELECT AVG(Balance)
  FROM Account)

XEN (Insert)

Để xen dữ liệu vào một quan hệ, ta xác định một bộ cần xen hoặc viết một câu vấn tin kết quả của nó là một tập các bộ cần xen. Các giá trị thuộc tính của bộ cần xen phải thuộc vào miền giá trị của thuộc tính và số thành phần của bộ phải bằng với ngôi của quan hệ.

"Xen vào quan hệ Account một bộ có số tài khoản là A-9732, số cân đối là 1200$ và tài khoản này được mở ở chi nhánh Perryridge"

```
INSERT INTO Account
VALUES ('Perryridge', 'A-9732', 1200);
```

Trong ví dụ này thự tự các giá trị thuộc tính cần xen trùng khớp với thứ tự các thuộc tính trong sơ đồ quan hệ. SQL cho phép chỉ rõ các thuộc tính và các giá trị tương ứng cần xen:

```
INSERT INTO Account (Branch_name, Account_number, Balance)
VALUES ('Perryridge', 'A-9732', 1200);

INSERT INTO Account (Account_number, Balance, Branch_name)
VALUES ('A-9732', 1200, 'Perryridge');
```

"Cấp cho tất cả các khách hàng vay ở chi nhánh Perryridge một tài khoản với số cân đối là 200$ như một quà tặng sử dụng số vay như số tài khoản"

```
INSERT INTO Account
        SELECT Branch_name, Loan_number, 200
        FROM Loan
        WHERE Branch_name = 'Perryridge'
INSERT INTO Depositor
        SELECT Customer_name, Loan_number
        FROM Borrower, Loan
        WHERE Borrower.Loan_number = Loan.Loan_number AND
                Branch_name = 'Perryridge'
```

CẬP NHẬT (Update)

Câu lệnh UPDATE cho phép thay đổi giá trị thuộc tính của các bộ

"Thêm lãi hàng năm vào số cân đối với tỷ lệ lãi suất 5%"

```
UPDATE Account
SET Balance = Balance*1.05
```

Giả sử các tài khoản có số cân đối > 10000$ được hưởng lãi suất 6%, các tài khoản có số cân đối nhỏ hơn hoặc bằng 10000 được hưởng lãi suất 5%

```
UPDATE Account
SET Balance = Balance*1.06
WHERE Balance > 10000
UPDATE Account
SET Balance = Balance*1.05
WHERE Balance <= 10000
```

SQL92 đưa vào cấu trúc CASE như sau:

```
CASE
        WHEN P1 THEN Result1
        WHEN P2 THEN Result2
        ...
        WHEN Pn THEN Resultn
        ELSE  Result0
END
```

trong đó P_i là các vị từ, $Result_i$ là các kết quả trả về của hoạt động CASE tương ứng với vị từ P_i đầu tiên thỏa mãn. Nếu không vị từ P_i nào thỏa mãn CASE trả về $Result_0$.

Với cấu trúc CASE như vậy ta có thể viết lại yêu cầu trên như sau:

```
UPDATE Account
SET Balance = CASE
        WHEN Balance > 10000 THEN Balance*1.06
        ELSE Balance*1.05
END
```

"Trả 5% lãi cho các tài khoản có số cân đối lớn hơn số cân đối trung bình"

```
UPDATE Account
```

```
SET Balance = Balance*1.05
WHERE Balance >  SELECT AVG(Balance)
                 FROM Account
```

CÁC QUAN HỆ NỐI

SQL92 cung cấp nhiều cơ chế cho nối các quan hệ bao hàm nối có điều kiện và nối tự nhiên cũng như các dạng của nối ngoài.

```
Loan INNER JOIN Borrower
ON Loan.Loan_number = Borrower.Loan_number
```

Nối quan hệ Loan và quan hệ Borrower với điều kiên:

```
Loan.Loan_number = Borrower.Loan_number
```

Quan hệ kết quả có các thuộc tính của quan hệ Loan và các thuộc tính của quan hệ Borrower (như vậy thuộc tính Loan_number xuất hiện 2 lần trong quan hệ kết quả).

Để đổi tên quan hệ (kết quả) và các thuộc tính, ta sử dụng mệnh đề **AS**

```
Loan INNER JOIN Borrower
ON Loan.Loan_number = Borrower.Loan_number
AS  LB(Branch, Loan_number, Amount, Cust, Cust_Loan_number)
Loan LEFT OUTER JOIN Borrower
ON Loan.Loan_number = Borrower.Loan_number
```

Phép nối ngoài trái được tính như sau: Đầu tiên tính kết quả của nối trong INNER JOIN. Sau đó đối với mỗi bộ t của quan hệ trái (Loan) không tương xứng với bộ nào trong quan hệ bên phải (borrower) khi đó thêm vào kết quả bộ r gồm các giá trị thuộc tính trái là các giá trị thuộc tính của t, các thuộc tính còn lại (phải) được đặt là null.

```
Loan NATURAL INNER JOIN Borrower
```

Là nối tự nhiên của quan hệ Loan và quan hệ Borrower (thuộc tính trùng tên là Loan_number).

NGÔN NGỮ ĐỊNH NGHĨA DỮ LIỆU (DDL)

DDL SQL cho phép đặc tả:

- Sơ đồ cho mỗi quan hệ
- Miền giá trị kết hợp với mỗi thuộc tính
- các ràng buộc toàn vẹn
- tập các chỉ mục được duy trì cho mỗi quan hệ
- thông tin về an toàn và quyền cho mỗi quan hệ
- cấu trúc lưu trữ vật lý của mỗi quan hệ trên đĩa

CÁC KIỂU MIỀN TRONG SQL

SQL-92 hỗ trợ nhiều kiểu miền trong đó bao hàm các kiểu sau:

- **char(n) / charater:** chuỗi ký tự dộ dài cố định, với độ dài n được xác định bởi người dùng
- **vachar(n) / character varying (n):** chuỗi ký tự độ dài thay đổi, với độ dài tối đa được xác dịnh bởi người dung là n
- **int / integer:** tập hữu hạn các số nguyên
- **smallint:** tập con của tập các số nguyên **int**
- **numeric(p, d):** số thực dấu chấm tĩnh gồm p chữ số (kể cả dấu) và d trong p chữ số là các chữ số phần thập phân
- **real, double precision:** số thực dấu chấm động và số thực dấu chấm động chính xác kép
- **float(n):** số thực dấu chấm động với độ chính xác được xác định bởi người dùng ít nhất là n chữ số thập phân

- **date:** kiểu năm tháng ngày (YYYY, MM, DD)
- **time:** kiểu thời gian (HH, MM, SS)

SQL-92 cho phép định nghĩa miền với cú pháp:

CREATE DOMAIN < tên miền > < Type >

Ví dụ: CREATE DOMAIN hoten char(30);

Sau khi đã định nghĩa miền với tên hoten ta có thể sử dụng nó để định nghĩa kiểu của các thuộc tính

ĐỊNH NGHĨA SƠ ĐỒ TRONG SQL.

Lệnh **CREATE TABLE** với cú pháp

CREATE TABLE < tên bảng > (
< Thuộc tính 1 > < miền giá trị thuộc tính 1 > ,
...
< Thuộc tính n > < miền giá trị thuộc tính n> ,
< ràng buộc toàn vẹn 1 > ,
...
< ràng buộc toàn vẹn k >)

Các ràng buộc toàn vẹn cho phép bao gồm:

primary key ($A_{i_1}, A_{i_2}, ..., A_{i_m}$ **)**

và

check(P)

Đặc tả **primary key** chỉ ra rằng các thuộc tính $A_{i_1}, A_{i_2}, ..., A_{i_m}$ tạo nên khoá chính của quan hệ. Mệnh đề **check** xác định một vị từ P mà mỗi bộ trong quan hệ phải thoả mãn.

Ví dụ:

```
CREATE TABLE customer (
        customer_name       CHAR(20) not null,
        customer_street     CHAR(30),
        customer_city       CHAR(30),
        PRIMARY KEY(customer_name));
CREATE TABLE branch (
        branch_name         CHAR(15) not null,
        branch_city         CHAR(30),
        assets              INTEGER,
        PRIMARY KEY (branch_name),
        CHECK (assets >= 0));
CREATE TABLE account (
        account_number      CHAR(10) not null,
        branch_name         CHAR(15),
        balance             INTEGER,
        PRIMARY KEY (account_number),
        CHECK       (balance >= 0));
CREATE TABLE depositor (
        customer_name       CHAR(20) not null,
        account_number      CHAR(10) not null,
        PRIMARY KEY (customer_name, account_namber));
```

Giá trị **null** là giá trị hợp lệ cho mọi kiểu trong SQL. Các thuộc tính được khai báo là primary key đòi hỏi phải là ***not null*** và ***duy nhất***. do vậy các khai báo not null trong ví dụ trên là dư (trong SQL-92).

```
CREATE TABLE student (
```

```
name            CHAR(15) not null,
student_ID      CHAR(10) not null,
degree_level    CHAR(15) not null,
PRIMARY KEY (student_ID),
CHECK (degree_level  IN  ('Bachelors', 'Masters', 'Doctorats'));
```

- Xoá một quan hệ khỏi CSDL sử dụng lệnh **Drop table** với cú pháp:
  DROP TABLE < tên bảng >
- Thêm thuộc tính vào bảng đang tồn tại sử dụng lệnh **Alter table** với cú pháp:
  ALTER TABLE < tên bảng > ADD < thuộc tính > < miền giá trị >
- Xoá bỏ một thuộc tính khỏi bảng đang tồn tại sử dụng lệnh **Alter table** với cú pháp:
  ALTER TABLE < Tên bảng > DROP < tên thuộc tính >

SQL NHÚNG (Embedded SQL)

Một ngôn ngữ trong đó các vấn tin SQL được nhúng gọi là ngôn ngữ chủ *(host language)*, cấu trúc SQL cho phép trong ngôn ngữ chủ tạo nên SQL nhúng. Chương trình được viết trong ngôn ngữ chủ có thể sử dụng cú pháp SQL nhúng để truy xuất và cập nhật dữ liệu được lưu trữ trong CSDL.

BÀI TẬP CHƯƠNG II

II.1. Xét CSDL bảo hiểm sau:

person(<u>ss#</u>, name, address): Số bảo hiểm ss# sở hữu bởi người tên name ở địa chỉ address

car(<u>license</u>, <u>year</u>, model): Xe hơi số dăng ký license, sản xuất năm year, nhãn hiệu Model

accident(<u>date</u>, <u>driver</u>, damage_amount): tai nạn xảy ra ngày date, do người lái driver, mức hư hại damage_amount

owns(<u>ss#</u>, <u>license</u>): người mang số bảo hiểm ss# sở hữu chiếc xe mang số đăng ký license

log(<u>license</u>, <u>date</u>, <u>driver</u>): ghi sổ chiếc xe mang số đăng ký license, bị tai nạn ngày do người lái driver

các thuộc tính được gạch dưới là các primary key. Viết trong SQL các câu vấn tin sau:

1. Tìm tổng số người xe của họ gặp tai nạn năm 2001
2. Tìm số các tai nạn trong đó xe của"John"liên quan tới
3. Thêm khách hàng mới: ss# =”A-12345”, name ="David”, address ="35 Chevre Road”, license ="109283”, year =”2002”, model ="FORD LASER"vào CSDL
4. xoá các thông tin lien quan dến xe model "MAZDA"của"John Smith”
5. Thêm thông tin tai nạn cho chiếc xe"TOYOTA"của khách hàng mang số bảo hiểm số"A-84626”

II.2. Xét CSDL nhân viên:

employee (E_name, street, city): Nhân viên có tên E_name, cư trú tại phố street, trong thành phố city

works (E_name, C_name, salary): Nhân viên tên E_name làm việc cho công ty C_name với mức lương salary

copany (C_name, city): Công ty tên C_name đóng tại thành phố city

manages(E_name, M_name): Nhân viên E_name dưới sự quản lý của nhân viên M_name

Viết trong SQL các câu vấn tin sau:

1. Tìm tên của tất cả các nhân viên làm việc cho First Bank
2. Tìm tên và thành phố cư trú của các nhân viên làm việc cho First Bank
3. Tìm tên, phố, thành phố cư trú làm việc cho First Bank hưởng mức lương > 10000$
4. Tìm tất cả các nhân viên trong CSDL sống trong cùng thành phố với công ty mang họ làm việc cho
5. Tìm tất cả các nhân viên sông trong cùng thành phố, cùng phố với người quản lý của họ
6. Tìm trong CSDL các nhân viên không làm việc cho First Bank
7. Tìm trong CSDL, các nhân viên hưởng mức lương cao hơn mọi nhân viên của Small Bank
8. Giả sử một công ty có thể đóng trong một vaì thành phố. Tìm tất cả các công ty đóng trong mỗi thành phố trong đó Small Bank đóng.
9. Tìm tất cả các nhân viên hưởng múc lương cao hơn mức lương trung bình của công ty họ làm việc
10. Tìm công ty có nhiều nhân viên nhất
11. Tìm công ty có tổng số tiền trả lương nhỏ nhất
12. Tìm tất cả các công ty có mức lương trung bình cao hơn mức luong trung bình của công ty First Bank
13. Thay đổi thành phố cư trú của nhân viên"Jones"thành NewTown
14. Nâng lương cho tất cả các nhân viên của First Bank lên 10%
15. nâng lương cho các nhà quản lý của công ty First Bank lên 10%
16. Xoá tất cả các thông tin liên quan tới cong ty Bad Bank

CHƯƠNG III

LƯU TRỮ VÀ CẤU TRÚC TẬP TIN
(Storage and File Structure)

MỤC ĐÍCH

Chương này trình bày các vấn đề liên quan đến vấn đề lưu trữ dữ liệu (trên lưu trữ ngoài, chủ yếu trên đĩa cứng). Việc lưu trữ dữ liệu phải được tổ chức sao cho có thể cất giữ một lượng lớn, có thể rất lớn dữ liệu nhưng quan trọng hơn cả là sự lưu trữ phải cho phép lấy lại dữ liệu cần thiết mau chóng. Các cấu trúc trợ giúp cho truy xuất nhanh dữ liệu được trình bày là: chỉ mục (indice), B^+ cây (B^+-tree), băm (hashing) ... Các thiết bị lưu trữ (đĩa) có thể bị hỏng hóc không lường trước, các kỹ thuật RAID cho ra một giải pháp hiệu quả cho vấn đề này.

YÊU CẦU

Hiểu rõ các đặc điểm của các thiết bị lưu trữ, cách tổ chức lưu trữ, truy xuất đĩa.

Hiểu rõ nguyên lý và kỹ thuật của tổ chức hệ thống đĩa RAID

Hiểu rõ các kỹ thuật tổ chức các mẩu tin trong file

Hiểu rõ các kỹ thuật tổ chức file

Hiểu và vận dụng các kỹ thuật hỗ trợ tìm lại nhanh thông tin: chỉ mục (được sắp, B^+-cây, băm)

KHÁI QUÁT VỀ PHƯƠNG TIỆN LƯU TRỮ VẬT LÝ

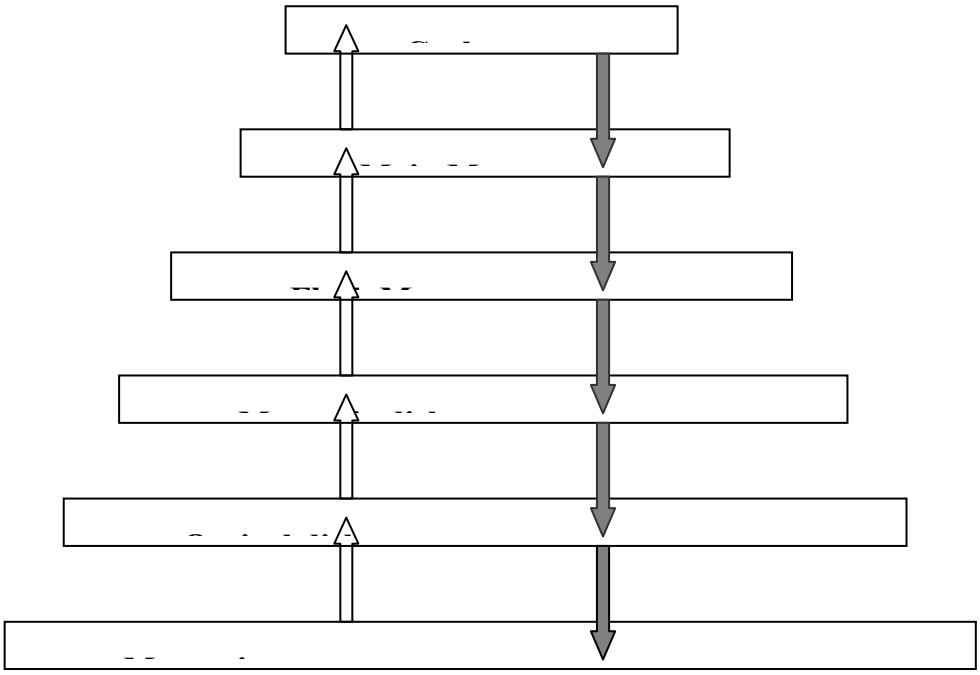
Có một số kiểu lưu trữ dữ liệu trong các hệ thống máy tính. Các phương tiện lưu trữ được phân lớp theo *tốc độ truy xuất, theo giá cả và theo độ tin cậy của phương tiện*. Các phương tiện hiện có là:

- **Cache**: là dạng lưu trữ nhanh nhất và cũng đắt nhất trong các phương tiện lưu trữ. Bộ nhớ cache nhỏ; sự sử dụng nó được quản trị bởi hệ điều hành
- **Bộ nhớ chính (main memory)**: Phương tiện lưu trữ dùng để lưu trữ dữ liệu sẵn sàng được thực hiện. Các chỉ thị máy mục đích chung (general-purpose) hoạt động trên bộ nhớ chính. Mặc dầu bộ nhớ chính có thể chứa nhiều megabytes dữ liệu, nó vẫn là quá nhỏ (và quá đắt giá) để lưu trữ toàn bộ một cơ sở dữ liệu. Nội dung trong bộ nhớ chính thường bị mất khi mất cấp nguồn
- **Bộ nhớ Flash:** Được biết như bộ nhớ chỉ đọc có thể lập trình, có thể xoá (EEPROM: Electrically Erasable Programmable Read-Only Memory), Bộ nhớ Flash khác bộ nhớ chính ở chỗ dữ liệu còn tồn tại trong bộ nhớ flash khi mất cấp nguồn. Đọc dữ liệu từ bộ nhớ flash mất ít hơn 100 ns , nhanh như đọc dữ liệu từ bộ nhớ chính. Tuy nhiên, viết dữ liệu vào bộ nhớ flash phức tạp hơn nhiều. Dữ liệu được viết (một lần mất khoảng 4 đến 10 μs) nhưng không thể viết đè trực tiếp. Để viết đè bộ nhớ đã được viết, ta phải xoá trắng toàn bộ bộ nhớ sau đó mới có thể viết lên nó.
- **Lưu trữ đĩa từ (magnetic-disk): (**ở đây, được hiểu là đĩa cứng) Phương tiện căn bản để lưu trữ dữ liệu trực tuyến, lâu dài. Thường toàn bộ cơ sở dữ liệu được lưu trữ trên đĩa từ. Dữ liệu phải được chuyển từ đĩa vào bộ nhớ chính trước khi được truy nhập. Khi dữ liệu trong bộ nhớ chính này bị sửa đổi, nó phải được viết lên đĩa. Lưu trữ đĩa được xem là truy xuất trực tiếp vì có thể đọc dữ liệu trên đĩa theo một thứ tự bất kỳ. Lưu trữ đĩa vẫn tồn tại khi mất cấp nguồn. Lưu trữ đĩa có thể bị hỏng hóc, tuy không thường xuyên.
- **Lưu trữ quang (Optical storage):** Dạng quen thuộc nhất của đĩa quang học là loại đĩa **CD-ROM** : Compact-Disk Read-Only Memory. Dữ liệu được lưu trữ trên các đĩa quang học được đọc bởi laser. Các đĩa quang học CD-ROM chỉ có thể đọc. Các phiên bản khác của chúng là loại đĩa quang học: viết một lần, đọc nhiều lần (write-once, read-many: **WORM**) cho phép viết dữ liệu lên đĩa một lần, không cho phép xoá và viết lại, và các đĩa có thể viết lại (rewritable) v..v
- **Lưu trữ băng từ (tape storage):** Lưu trữ băng từ thường dùng để backup dữ liệu. Băng từ rẻ hơn đĩa, truy xuất dữ liệu chậm hơn (vì phải truy xuất tuần tự). Băng từ thường có dung lượng rất lớn.

Các phương tiện lưu trữ có thể được tổ chức phân cấp theo tốc độ truy xuất và giá cả. Mức cao nhất là nhanh nhất nhưng cũng là đắt nhất, giảm dần xuống các mức thấp hơn.

Các phương tiện lưu trữ nhanh (*cache, bộ nhớ chính*) được xem như là lưu trữ sơ cấp (primary storage), các thiết bị lưu trữ ở mức thấp hơn như đĩa từ được xem như lưu trữ thứ cấp hay lưu trữ trực tuyến (on-line storage), còn các thiết bị lưu trữ ở mức thấp nhất và gần thấp nhất như đĩa quang học, băng từ kể cả các đĩa mềm được xếp vào lưu trữ tam cấp hay lưu trữ không trực tuyến (off-line).

Bên cạnh vấn đề tốc độ và giá cả, ta còn phải xét đến tính lâu bền của các phương tiện lưu trữ.



Phân cấp thiết bị lưu trữ

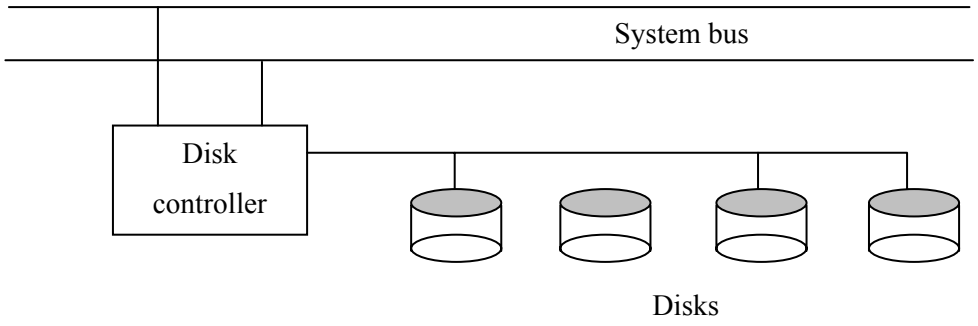
ĐĨA TỪ

ĐẶC TRƯNG VẬT LÝ CỦA ĐĨA

Mỗi tấm đĩa có dạng hình tròn, hai mặt của nó được phủ bởi vật liệu từ tính, thông tin được ghi trên bề mặt đĩa. Đĩa gồm nhiều tấm đĩa. Ta sẽ sử dụng thuật ngữ đĩa để chỉ các đĩa cứng.

Khi đĩa được sử dụng, một động cơ ổ đĩa làm quay nó ở một tốc độ không đổi. Một đầu đọc-viết được định vị trên bề mặt của tấm đĩa. Bề mặt tấm đĩa được chia logic thành các rãnh, mỗi rãnh lại được chia thành các sector, một sector là một đơn vị thông tin nhỏ có thể được đọc, viết lên đĩa. Tuỳ thuộc vào kiểu đĩa, sector thay đổi từ 32 bytes đến 4095 bytes, thông thường là 512 bytes. Có từ 4 đến 32 sectors trên một rãnh, từ 20 đến 1500 rãnh trên một bề mặt. Mỗi bề mặt của một tấm đĩa có một đầu đọc viết, nó có thể chạy dọc theo bán kính đĩa để truy cập đến các rãnh khác nhau. Một đĩa gồm nhiều tấm đĩa, các đầu đọc-viết của tất cả các rãnh được gắn vào một bộ được gọi là cánh tay đĩa, di chuyển cùng nhau. Các tấm đĩa được gắn vào một trục quay. Vì các đầu đọc-viết trên các tấm đĩa di chuyển cùng nhau, nên khi đầu đọc-viết trên một tấm đĩa đang ở rãnh thứ i thì các đầu đọc-viết của các tấm đĩa khác cũng ở rãnh thứ i , do vậy các rãnh thứ i của tất cả các tấm đĩa được gọi là trụ (cylinder) thứ i . Một bộ điều khiển đĩa -- giao diện giữa hệ thống máy tính và phần cứng hiện thời của ổ đĩa. Nó chấp nhận các lệnh mức cao để đọc và viết một sector, và khởi động các hành động như di chuyển cánh tay đĩa đến các rãnh đúng và đọc viết dữ liệu. bộ điều khiển đĩa cũng tham gia vào checksum mỗi sector được viết. Checksum được tính từ dữ liệu được viết lên sector. Khi sector được đọc lại, checksum được tính lại từ dữ liệu được lấy ra và so sánh với checksum đã lưu trữ. Nếu dữ liệu bị sai lạc, checksum được tính sẽ không khớp với checksum đã lưu trữ. Nếu lỗi như vậy xảy ra, bộ điều khiển sẽ lặp lại việc đọc vài lần, nếu lỗi vẫn xảy ra, bộ điều khiển sẽ thông báo việc đọc thất bại. Bộ điều khiển đĩa còn có

chức năng tái ánh xạ các sector xấu: ánh xạ các sector xấu đến một vị trí vật lý khác. Hình dưới bày tỏ các đĩa được nối với một hệ thống máy tính:



Các đĩa được nối với một hệ thống máy tính hoặc một bộ điều khiển đĩa qua một sự hợp nhất tốc độ cao. Hợp nhất hệ thống máy tính nhỏ (Small Computer-System Interconnect: SCSI) thường được sử dụng để nối kết các đĩa với các máy tính cá nhân và workstation. Mainframe và các hệ thống server thường có các bus nhanh hơn và đắt hơn để nối với các đĩa.

Các đầu đọc-viết được giữ sát với bề mặt đĩa như có thể để tăng độ dày đặc (density).

Đĩa đầu cố định (Fixed-head) có một đầu riêng biệt cho mỗi rãnh, sự sắp xếp này cho phép máy tính chuyển từ rãnh này sang rãnh khác mau chóng, không phải di chuyển đầu đọc-viết. Tuy nhiên, cần một số rất lớn đầu đọc-viết, điều này làm nâng giá của thiết bị.

ĐO LƯỜNG HIỆU NĂNG CỦA ĐĨA

Các tiêu chuẩn đo lường chất lượng chính của đĩa là ***dung lượng, thời gian truy xuất, tốc độ truyền dữ liệu và độ tin cậy***.

- ***Thời gian truy xuất (access time)***: là khoảng thời gian từ khi yêu cầu đọc/viết được phát đi đến khi bắt đầu truyền dữ liệu. Để truy xuất dữ liệu trên một sector đã cho của một đĩa, đầu tiên cánh tay đĩa phải di chuyển đến rãnh đúng, sau đó phải chờ sector xuất hiện dưới nó, thời gian để định vị cánh tay được gọi là thời gian tìm kiếm (seek time), nó tỷ lệ với khoảng cách mà cánh tay phải di chuyển, thời gian tìm kiếm nằm trong khoảng 2..30 ms tuỳ thuộc vào rãnh xa hay gần vị trí cánh tay hiện tại.
- ***Thời gian tìm kiếm trung bình (average seek time)***: Thời gian tìm kiếm trung bình là trung bình của thời gian tìm kiếm, được đo luờng trên một dãy các yêu cầu ngẫu nhiên (phân phối đều), và bằng khoảng 1/3 thời gian tìm kiếm trong trường hợp xấu nhất.
- ***Thời gian tiềm ẩn luân chuyển (rotational latency time)***: Thời gian chờ sector được truy xuất xuất hiện dưới đầu đọc/viết. Tốc độ quay của đĩa nằm trong khoảng 60..120 vòng quay trên giây, trung bình cần nửa vòng quay để sector cần thiết nằm dưới đầu đọc/viết. Như vậy, thời gian tiềm ẩn trung bình (average latency time) bằng nửa thời gian quay một vòng đĩa.

Thời gian truy xuất bằng tổng của thời gian tìm kiếm và thời gian tiềm ẩn và nằm trong khoảng 10..40 ms.

- ***Tốc độ truyền dữ liệu:*** là tốc độ dữ liệu có thể được lấy ra từ đĩa hoặc được lưu trữ vào đĩa. Hiện nay tốc này vào khoảng1..5 Mbps

- ***Thời gian trung bình không sự cố (mean time to failure):*** lượng thời gian trung bình hệ thống chạy liên tục không có bất kỳ sự cố nào. Các đĩa hiện nay có thời gian không sự cố trung bình khoảng 30000 .. 800000 giờ nghĩa là khoảng từ 3,4 đến 91 năm.

TỐI ƯU HÓA TRUY XUẤT KHỐI ĐĨA (disk-block)

Yêu cầu I/O đĩa được sinh ra cả bởi hệ thống file lẫn bộ quản trị bộ nhớ ảo trong hầu hết các hệ điều hành. Mỗi yêu cầu xác định địa chỉ trên đĩa được tham khảo, địa chỉ này ở dạng số khối. Một khối là một dãy các sector kề nhau trên một rãnh. Kích cỡ khối trong khoảng 512 bytes đến một vài Kbytes. Dữ liệu được truyền giữa đĩa và bộ nhớ chính theo đơn vị khối. Mức thấp hơn của bộ quản trị hệ thống file sẽ chuyển đổi địa chỉ khối sang số của trụ, của mặt và của sector ở mức phần cứng.

Truy xuất dữ liệu trên đĩa chậm hơn nhiều so với truy xuất dữ liệu trong bộ nhớ chính, do vậy cần thiết một chiến lược nhằm nâng cao tốc độ truy xuất khối đĩa. Dưới đây ta sẽ thảo luận một vài kỹ thuật nhằm vào mục đích đó.

- ***Scheduling:*** Nếu một vài khối của một trụ cần được truyền từ đĩa vào bộ nhớ chính, ta có thể tiết kiệm thời gian truy xuất bởi yêu cầu các khối theo thứ tự mà nó chạy qua dưới đầu đọc/viết. Nếu các khối mong muốn ở trên các trụ khác nhau, ta yêu cầu các khối theo thứ tự sao cho làm tối thiểu sự di chuyển cánh tay đĩa. Các thuật toán scheduling cánh tay đĩa (Disk-arm-scheduling) nhằm lập thứ tự truy xuất các rãnh theo cách làm tăng số truy xuất có thể được xử lý. Một thuật toán thường dùng là thuật toán thang máy (elevator algorithm): Giả sử ban đầu cánh tay di chuyển từ rãnh trong nhất hướng ra phía ngoài đĩa, đối với mỗi rãnh có yêu cầu truy xuất, nó dừng lại, phục vụ yêu cầu đối với rãnh này, sau đó tiếp tục di chuyển ra phía ngoài đến tận khi không có yêu cầu nào chờ các rãnh xa hơn phía ngoài. Tại điểm này, cánh tay đổi hướng, di chuyển vào phía trong, lại dừng lại trên các rãnh được yêu cầu, và cứ như vậy đến tận khi không còn rãnh nào ở trong hơn được yêu cầu, rồi lại đổi hướng .. v .. v .. Bộ điều khiển đĩa thường làm nhiệm vụ sắp xếp lại các yêu cầu đọc để cải tiến hiệu năng.
- ***Tổ chức file:*** Để suy giảm thời gian truy xuất khối, ta có thể tổ chức các khối trên đĩa theo cách tương ứng gần nhất với cách mà dữ liệu được truy xuất. Ví dụ, Nếu ta muốn một file được truy xuất tuần tự, khi đó ta bố trí các khối của file một cách tuần tự trên các trụ kề nhau. Tuy nhiên việc phân bố các khối lưu trữ kề nhau này sẽ bị phá vỡ trong quá trình phát triển của file $\Rightarrow$ file không thể được phân bố trên các khối kề nhau được nữa, hiện tượng này dược gọi là sự phân mảnh (fragmentation). Nhiều hệ điều hành cung cấp tiện ích giúp suy giảm sự phân mảnh này (Defragmentation) nhằm làm tăng hiệu năng truy xuất file.
- ***Các buffers viết không hay thay đổi:*** Vì nội dung của bộ nhớ chính bị mất khi mất nguồn, các thông tin về cơ sở dữ liệu cập nhật phải được ghi lên đĩa nhằm đề phòng sự cố. Hiệu năng của các ứng dụng cập nhật cường độ cao phụ thuộc mạnh vào tốc độ viết đĩa. Ta có thể sử dụng bộ nhớ truy xuất ngẫu nhiên không hay thay đổi (nonvolatile RAM) để nâng tốc độ viết đĩa. Nội dung của nonvolatile RAM không bị mất khi mất nguồn. Một phương pháp chung để thực hiện nonvolatile RAM là sử dụng RAM pin dự phòng (battery-back-up RAM). Khi cơ sở dữ liệu yêu cầu viết một khối lên đĩa, bộ điều khiển dĩa viết khối này lên buffer nonvolatile RAM, và thông báo ngay cho hệ điều hành là việc viết đã thành công. Bộ điều khiển sẽ viết dữ liệu đến đích của nó trên đĩa, mỗi khi đĩa rảnh hoặc buffer nonvolatile RAM đầy. Khi hệ cơ sở dữ liệu yêu cầu một viết khối, nó chỉ chịu một khoảng lặng chờ đợi khi buffer nonvolatile RAM đầy.

- ***Đĩa log (log disk):*** Một cách tiếp cận khác để làm suy giảm tiềm năng viết là sử dụng log-disk: Một đĩa được tận hiến cho việc viết một log tuần tự. Tất cả các truy xuất đến log-disk là tuần tự, nhằm loại bỏ thời gian tìm kiếm, và một vài khối kề có thể được viết một lần, tạo cho viết vào log-disk nhanh hơn viết ngẫu nhiên vài lần. Cũng như trong trường hợp sử dụng nonvolatile RAM, dữ liệu phải được viết vào vị trí hiện thời của chúng trên đĩa, nhưng việc viết này có thể được tiến hành mà hệ cơ sở dữ liệu không cần thiết phải chờ nó hoàn tất. Log-disk có thể được sử dụng để khôi phục dữ liệu. Hệ thống file dựa trên log là một phiên bản của cách tiếp cận log-disk: Dữ liệu không được viết lại lên đích gốc của nó trên đĩa; thay vào đó, hệ thống file lưu vết nơi các khối được viết mới đây nhất trên log-disk, và hoàn lại chúng từ vị trí này. Log-disk được "cô đặc" lại (compacting) theo một định kỳ. Cách tiếp cận này cải tiến hiệu năng viết, song sinh ra sự phân mảnh đối với các file được cập nhật thường xuyên.

RAID

Trong một hệ thống có nhiều đĩa, ta có thể cải tiến tốc độ đọc viết dữ liệu nếu cho chúng hoạt động song song. Mặt khác, hệ thống nhiều đĩa còn giúp tăng độ tin cậy lưu trữ bằng cách lưu trữ dư thừa thông tin trên các đĩa khác nhau, nếu một đĩa có sự cố dữ liệu cũng không bị mất. Một sự đa dạng các kỹ thuật tổ chức đĩa, được gọi là **RAID (Redundant Arrays of Inexpensive Disks)**, được đề nghị nhằm vào vấn đề tăng cường hiệu năng và độ tin cậy.

CẢI TIẾN ĐỘ TIN CẬY THÔNG QUA SỰ DƯ THỪA

Giải pháp cho vấn đề độ tin cậy là đưa vào sự dư thừa: lưu trữ thông tin phụ, bình thường không cần thiết, nhưng nó có thể được sử dụng để tái tạo thông tin bị mất khi gặp sự cố hỏng hóc đĩa, như vậy thời gian trung bình không sự cố tăng lên (xét tổng thể trên hệ thống đĩa).

Đơn giản nhất, là làm bản sao cho mỗi đĩa. Kỹ thuật này được gọi là mirroring hay shadowing. Một đĩa logic khi đó bao gồm hai đĩa vật lý, và mỗi việc viết được thực hiện trên cả hai đĩa. Nếu một đĩa bị hư, dữ liệu có thể được đọc từ đĩa kia. Thời gian trung bình không sự cố của đĩa mirror phụ thuộc vào thời gian trung bình không sự cố của mỗi đĩa và phụ thuộc vào thời gian trung bình được sửa chữa (mean time to repair): thời gian trung bình để một đĩa bị hư được thay thế và phục hồi dữ liệu trên nó.

CẢI TIẾN HIỆU NĂNG THÔNG QUA SONG SONG

Với đĩa mirror, tốc độ đọc có thể tăng lên gấp đôi vì yêu cầu đọc có thể được gửi đến cả hai đĩa. Với nhiều đĩa, ta có thể cải tiến tốc độ truyền bởi phân nhỏ (striping data) dữ liệu qua nhiều đĩa. Dạng đơn giản nhất là tách các bít của một byte qua nhiều đĩa, sự phân nhỏ này được gọi là sự phân nhỏ mức bit (bit-level striping). Ví dụ, ta có một dàn 8 đĩa, ta viết bít thứ i của một byte lên đĩa thứ i . dàn 8 đĩa này có thể được xử lý như một đĩa với các sector 8 lần lớn hơn kích cỡ thông thường, quan trọng hơn là tốc dộ truy xuất tăng lên tám lần. Trong một tổ chức như vậy, mỗi đĩa tham gia vào mỗi truy xuất (đọc/viết), như vậy, số các truy xuất có thể được xử lý trong một giây là tương tự như trên một đĩa, nhưng mỗi truy xuất có thể đọc/viết nhiều dữ liệu hơn tám lần.

Phân nhỏ mức bit có thể được tổng quát cho số đĩa là bội hoặc ước của 8, Ví dụ, ta có một dàn 4 đĩa, ta sẽ phân phối bít thứ i và bít thứ 4+i vào đĩa thứ i. Hơn nữa, sự phân nhỏ không nhất thiết phải ở mức bit của một byte. Ví dụ, trong sự phân nhỏ mức khối, các khối của một file được phân nhỏ qua nhiều đĩa, với n đĩa, khối thứ i có thể được phân phối qua đĩa (i mod n) + 1. Ta cũng

có thể phân nhỏ ở mức byte, sector hoặc các sector của một khối. Hai đích song song trong một hệ thống đĩa là:

1. Nạp nhiều truy xuất nhỏ cân bằng (truy xuất trang) sao cho lượng dữ liệu được nạp trong một đơn vị thời gian của truy xuất như vậy tăng lên.
2. Song song hoá các truy xuất lớn sao cho thời gian trả lời các truy xuất lớn giảm.

CÁC MỨC RAID

Mirroring cung cấp độ tin cậy cao, nhưng đắt giá. Phân nhỏ cung cấp tốc độ truyền dữ liệu cao, nhưng không cải tiến được độ tin cậy. Nhiều sơ đồ cung cấp sự dư thừa với giá thấp bằng cách phối hợp ý tưởng của phân nhỏ với "parity" bit. Các sơ đồ này có sự thoả hiệp *giá-hiệu năng* khác nhau và được phân lớp thành các mức được gọi là các mức RAID.

• **Mức RAID 0 :** Liên quan đến các dàn đĩa với sự phân nhỏ mức khối, nhưng không có một sự dư thừa nào.

• **Mức RAID 1 :** Liên quan đến mirror đĩa

• **Mức RAID 2 :** Cũng được biết dưới cái tên mã sửa lỗi kiểu bộ nhớ (memory-style error-correcting-code : ECC). Hệ thống bộ nhớ thực hiện phát hiện lỗi bằng bit parity. Mỗi byte trong hệ thống bộ nhớ có thể có một bit parity kết hợp với nó. Sơ đồ sửa lỗi lưu hai hoặc nhiều hơn các bit phụ, và có thể dựng lại dữ liệu nếu một bit bị lỗi. ý tưởng của mã sửa lỗi có thể được sử dụng trực tiếp trong dàn đĩa thông qua phân nhỏ byte qua các đĩa. Ví dụ, bít đầu tiên của mỗi byte có thể được lưu trên đĩa 1, bit thứ hai trên đĩa 2, và cứ như vậy, bit thứ 8 trên đĩa 8, các bit sửa lỗi được lưu trên các đĩa thêm vào. Nếu một trong các đĩa bị hư, các bít còn lại của byte và các bit sửa lỗi kết hợp được đọc từ các đĩa khác có thể giúp tái tạo bít bị mất trên đĩa hư, như vậy ta có thể dựng lại dữ liệu. Với một dàn 4 đĩa dữ liệu, RAID mức 2 chỉ cần thêm 3 đĩa để lưu các bit sửa lỗi (các đĩa thêm vào này được gọi là các đĩa overhead), so sánh với RAID mức 1, cần 4 đĩa overhead.

• **Mức RAID 3 :** Còn được gọi là tổ chức parity chen bit (bit-interleaved parity). Bộ điều khiển đĩa có thể phát hiện một sector được đọc đúng hay sai, như vậy có thể sử dụng chỉ một bit parity để sửa lỗi: Nếu một trong các sector bị hư, ta biết chính xác đó là sector nào, Với mỗi bit trong sector này ta có thể hình dung nó là bít 1 hay bit 0 bằng cách tính parity của các bit tương ứng từ các sector trên các đĩa khác. Nếu parity của các bit còn lại bằng với parity được lưu, bit mất sẽ là 0, ngoài ra bit mất là 1. RAID mức 3 tốt như mức 2 nhưng it tốn kém hơn (chỉ cần một đĩa overhead).

• **Mức RAID 4 :** Còn được gọi là tổ chức parity chen khối (Block-interleaved parity), lưu trữ các khối đúng như trong các đĩa chính quy, không phân nhỏ chúng qua các đĩa nhưng lấy một khối parity trên một đĩa riêng biệt đối với các khối tương ứng từ N đĩa khác. Nếu một trong các đĩa bị hư, khối parity có thể được dùng với các khối tương ứng từ các đĩa khác để khôi phục khối của đĩa bị hư.

Một đọc khối chỉ truy xuất một đĩa, cho phép các yêu cầu khác được xử lý bởi các đĩa khác. Như vậy, tốc độ truyền dữ liệu đối với mỗi truy xuất chậm, nhưng nhiều truy xuất đọc có thể được xử lý song song, dẫn đến một tốc độ I/O tổng thể cao hơn. Tốc độ truyền đối với các đọc dữ liệu lớn (nhiều khối) cao do tất cả các đĩa có thể được đọc song song; các viết dữ liệu lớn (nhiều khối) cũng có tốc độ truyền cao vì dữ liệu và parity có thể được viết song song. Tuy nhiên, viết một khối đơn phải truy xuất đĩa trên đó khối được lưu trữ, và đĩa parity (do khối parity cũng phải được cập nhật). Như vậy, viết một khối đơn yêu cầu 4 truy xuất: hai để đọc hai khối cũ, và hai để viết lại hai khối.

- **Mức RAID 5 :** Còn gọi là parity phân bố chen khối (Block-interleaved Distributed Parity), cải tiến của mức 4 bởi phân hoạch dữ liệu và parity giữa toàn bộ N+1 đĩa, thay vì lưu trữ dữ liệu trên N đĩa và parity trên một đĩa riêng biệt như trong RAID 4. Trong RAID 5, tất cả các đĩa có thể tham gia làm thoả mãn các yêu cầu đọc, như vậy sẽ làm tăng tổng số yêu cầu có thể được đặt ra trong một đơn vị thời gian. Đối với mỗi khối, một đĩa lưu trữ parity, các đĩa khác lưu trữ dữ liệu. Ví dụ, với một dàn năm đĩa, parity đối với khối thứ n được lưu trên đĩa (n mod 5)+1. Các khối thứ n của 4 đĩa khác lưu trữ dữ liệu hiện hành của khối đó.

- **Mức RAID 6 :** Còn được gọi là sơ đồ dư thừa P+Q (P+Q redundancy scheme), nó rất giống RAID 5 nhưng lưu trữ thông tin dư thừa phụ để canh chừng nhiều đĩa bị hư. Thay vì sử dụng parity, người ta sử dụng các mã sửa lỗi.

CHỌN MỨC RAID ĐÚNG

Nếu đĩa bị hư, Thời gian tái tạo dữ liệu của nó là đáng kể và thay đổi theo mức RAID được dùng. Sự tái tạo dễ dàng nhất đối với mức RAID 1. Đối với các mức khác, ta phải truy xuất tất cả các đĩa khác trong dàn đĩa để tái tạo dữ liệu trên đĩa bị hư. Hiệu năng tái tạo của một một hệ thống RAID có thể là một nhân tố quan trọng nếu việc cung cấp dữ liệu liên tục được yêu cầu (thường xảy ra trong các hệ CSDL hiệu năng cao hoặc trao đổi). Hơn nữa, hiệu năng tái tạo ảnh hưởng đến thời gian trung bình không sự cố.

Vì RAID mức 2 và 4 được gộp lại bởi RAID mức 3 và 5, Việc lựa chọn mức RAID thu hẹp lại trên các mức RAID còn lại. Mức RAID 0 được dùng trong các ứng dụng hiệu năng cao ở đó việc mất dữ liệu không có gì là trầm trọng cả. RAID mức 1 là thông dụng cho các ứng dụng lưu trữ các log-file trong hệ CSDL. Do mức 1 có overhead cao, mức 3 và 5 thường được ưa thích hơn đối với việc lưu trữ khối lượng dữ liệu lớn. Sự khác nhau giữa mức 3 và mức 5 là tốc độ truyền dữ liệu đối lại với tốc độ I/O tổng thể. Mức 3 được ưa thích hơn nếu truyền dữ liệu cao được yêu cầu, mức 5 được ưa thích hơn nếu việc đọc ngẫu nhiên là quan trọng. Mức 6, tuy hiện nay ít được áp dụng, nhưng nó có độ tin cậy cao hơn mức 5.

MỞ RỘNG

Các quan niệm của RAID được khái quát hoá cho các thiết bị lưu trữ khác, bao hàm các dàn băng, thậm chí đối với quảng bá dữ liệu trên các hệ thống không dây. Khi áp dụng RAID cho dàn băng, cấu trúc RAID cho khả năng khôi phục dữ liệu cả khi một trong các băng bị hư hại. Khi áp dụng đối với quảng bá dữ liệu, một khối dữ liệu được phân thành các đơn vị nhỏ và được quảng bá cùng với một đơn vị parity; nếu một trong các đơn vị này không nhận được, nó có thể được dựng lại từ các đơn vị còn lại.

LƯU TRỮ TAM CẤP (tertiary storage)

ĐĨA QUANG HỌC

CR-ROM có ưu điểm là có khả năng lưu trữ lớn, dễ di chuyển (có thể đưa vào và lấy ra khỏi ổ đĩa như đĩa mềm), hơn nữa giá lại rẻ. Tuy nhiên, so với ổ đĩa cứng, thời gian tìm kiếm của ổ CD-ROM chậm hơn nhiều (khoảng 250ms), tốc độ quay chậm hơn (khoảng 400rpm), từ đó dẫn đến độ trễ cao hơn; tốc độ truyền dữ liệu cũng chậm hơn (khoảng 150Kbytes/s). Gần đây, một định dạng mới của đĩa quang học - Digital video disk (DVD) - được chuẩn hoá, các đĩa này có dung lượng trong khoảng 4,7GBytes đến 17 GBytes. Các đĩa WORM, REWRITABLE cũng trở thành phổ biến. Các WORM jukeboxes là các thiết bị có thể lưu trữ một số lớn các đĩa WORM và có thể nạp tự động các đĩa theo yêu cầu đến một hoặc một vài ổ WORM.

BĂNG TỪ

Băng từ có thể lưu một lượng lớn dữ liệu, tuy nhiên, chậm hơn so với đĩa từ và đĩa quang học. Truy xuất băng buộc phải là truy xuất tuần tự, như vậy nó không thích hợp cho hầu hết các đòi hỏi của lưu trữ thứ cấp. Băng từ được sử dụng chính cho việc backup, cho lưu trữ các thông tin không được sử dụng thường xuyên và như một phương tiện ngoại vi (off-line medium) để truyền thông tin từ một hệ thống đến một hệ thống khác. Thời gian để định vị đoạn băng lưu dữ liệu cần thiết có thể kéo dài đến hàng phút. Jukeboxes băng chứa một lượng lớn băng, với một vài ổ băng và có thể lưu trữ được nhiều TeraBytes (10^{12} Bytes)

TRUY XUẤT LƯU TRỮ

Một cơ sở dữ liệu được ánh xạ vào một số các file khác nhau được duy trì bởi hệ điều hành nền. Các file này lưu trú thường trực trên các đĩa với backup trên băng. Mỗi file được phân hoạch thành các đơn vị lưu trữ độ dài cố định được gọi là khối - đơn vị cho cả cấp phát lưu trữ và truyền dữ liệu.

Một khối có thể chứa một vài hạng mục dữ liệu (data item). Ta giả thiết không một hạng mục dữ liệu nào trải ra trên hai khối. Mục tiêu nổi trội của hệ CSDL là tối thiểu hoá số khối truyền giữa đĩa và bộ nhớ. Một cách để giảm số truy xuất đĩa là giữ nhiều khối như có thể trong bộ nhớ chính. Mục đích là để khi một khối được truy xuất, nó đã nằm sẵn trong bộ nhớ chính và như vậy không cần một truy xuất đĩa nào cả.

Do không thể lưu tất cả các khối trong bộ nhớ chính, ta cần quản trị cấp phát không gian sẵn có trong bộ nhớ chính để lưu trữ các khối. Bộ đệm (Buffer) là một phần của bộ nhớ chính sãn có để lưu trữ bản sao khối đĩa. Luôn có một bản sao trên đĩa cho mỗi khối, song các bản sao trên đĩa của các khối là các phiên bản cũ hơn so với phiên bản trong buffer. Hệ thống con đảm trách cấp phát không gian buffer được gọi là bộ quản trị buffer.

BỘ QUẢN TRỊ BUFFER

Các chương trình trong một hệ CSDL đưa ra các yêu cầu cho bộ quản trị buffer khi chúng cần một khối đĩa. Nếu khối này đã sẵn sàng trong buffer, địa chỉ khối trong bộ nhớ chính được chuyển cho người yêu cầu. Nếu khối chưa có trong buffer, bộ quản trị buffer đầu tiên cấp phát không gian trong buffer cho khối, rút ra một số khối khác, nếu cần thiết, để lấy không gian cho khối mới. Khối được rút ra chỉ được viết lại trên đĩa khi nó có bị sửa đổi kể từ lần được viết lên đĩa gần nhất. Sau đó bộ quản trị buffer đọc khối từ đĩa vào buffer, và chuyển địa chỉ của khối trong bộ nhớ chính cho người yêu cầu. Bộ quản trị buffer không khác gì nhiều so với bộ quản trị bộ nhớ ảo, một điểm khác biệt là kích cỡ của một CSDL có thể rất lớn không đủ chứa toàn bộ trong bộ nhớ chính do vậy bộ quản trị buffer phải sử dụng các kỹ thuật tinh vi hơn các sơ đồ quản trị bộ nhớ ảo kiểu mẫu.

- **Chiến lược thay thế.** Khi không có chỗ trong buffer, một khối phải được xoá khỏi buffer trước khi một khối mới được đọc vào. Thông thường, hệ điều hành sử dụng sơ đồ LRU (Least Recently Used) để viết lên đĩa khối ít được dùng gần đây nhất, xoá bỏ nó khỏi buffer. Cách tiếp cận này có thể được cải tiến đối với ứng dụng CSDL.

- **Khối chốt (pinned blocks).** Để hệ CSDL có thể khôi phục sau sự cố, cần thiết phải hạn chế thời gian khi viết lại lên đĩa một khối. Một khối không cho phép viết lại lên đĩa được gọi là khối chốt.

- **Xuất ra bắt buộc các khối (Forced output of blocks).** Có những tình huống trong đó cần phải viết lại một khối lên đĩa, cho dù không gian buffer mà nó chiếm là không cần đến. Việc

viết này được gọi là *sự xuất ra bắt buộc của một khối*. Lý do ngắn gọn của yêu cầu xuất ra bắt buộc khối là nội dung của bộ nhớ chính bị mất khi có sự cố, ngược lại dữ liệu trên dĩa còn tồn tại sau sự cố.

CÁC ĐỐI SÁCH THAY THẾ BUFFER (Buffer-Replacement Policies).

Mục đích của chiến lược thay thế khối trong buffer là tối thiểu hoá các truy xuất đĩa. Các hệ điều hành thường sử dụng chiến lược LRU để thay thế khối. Tuy nhiên, một hệ CSDL có thể dự đoán mẫu tham khảo tương lai. Yêu cầu của một người sử dụng đối với hệ CSDL bao gồm một số bước. Hệ CSDL có thể xác định trước những khối nào sẽ là cần thiết bằng cách xem xét mỗi một trong các bước được yêu cầu để thực hiện hoạt động được yêu cầu bởi người sử dụng. Như vậy, khác với hệ điều hành, hệ CSDL có thể có thông tin liên quan đến tương lai, chí ít là tương lai gần. Trong nhiều trường hợp, chiến lược thay thế khối tối ưu cho hệ CSDL lại là MRU (Most Recently Used): Khối bị thay thế sẽ là khối mới được dùng gần đây nhất!

Bộ quản trị buffer có thể sử dụng thông tin thống kê liên quan đến xác suất mà một yêu cầu sẽ tham khảo một quan hệ riêng biệt nào đó. Tự điển dữ liệu là một trong những phần được truy xuất thường xuyên nhất của CSDL. Như vậy, bộ quản trị buffer sẽ không nên xoá các khối tự điển dữ liệu khỏi bộ nhớ chính trừ phi các nhân tố khác bức chế làm điều đó. Một chỉ mục (Index) đối với một file được truy xuất thường xuyên hơn chính bản thân file, vậy thì bộ quản trị buffer cũng không nên xoá khối chỉ mục khỏi bộ nhớ chính nếu có sự lựa chọn.

Chiến luợc thay thế khối CSDL lý tưởng cần hiểu biết về các hoạt động CSDL đang được thực hiện. Không một chiến lược đơn lẻ nào được biết nắm bắt được toàn bộ các viễn cảnh có thể. Tuy vậy, một điều đáng ngạc nhiên là phần lớn các hệ CSDL sử dụng LRU bất chấp các khuyết điểm của chiến lược đó.

Chiến lược được sử dụng bởi bộ quản trị buffer để thay thế khối bị ảnh hưởng bởi các nhân tố khác hơn là nhân tố thời gian tại đó khối được tham khảo trở lại. Nếu hệ thống đang xử lý các yêu cầu của một vài người sử dụng cạnh tranh, hệ thống (con) điều khiển cạnh tranh (concurrency-control subsystem) có thể phải làm trễ một số yêu cầu để đảm bảo tính nhất quán của CSDL. Nếu bộ quản trị buffer được cho các thông tin từ hệ thống điều khiển cạnh tranh mà nó nêu rõ những yêu cầu nào đang bị làm trễ, nó có thể sử dụng các thông tin này để thay đổi chiến lược thay thế khối của nó. Đặc biệt, các khối cần thiết bởi các yêu cầu tích cực (active requests) có thể được giữ lại trong buffer, toàn bộ các bất lợi đổ dồn lên các khối cần thiết bởi các yêu cầu bị làm trễ.

Hệ thống (con) khôi phục (crash-recovery subsystem) áp đặt các ràng buộc nghiêm nhặt lên việc thay thế khối. Nếu một khối bị sửa đổi, bộ quản trị buffer không được phép viết lại phiên bản mới của khối trong buffer lên đĩa, vì điều này phá huỷ phiên bản cũ. Thay vào đó, bộ quản trị khối phải tìm kiếm quyền từ hệ thống khôi phục trước khi viết khối. Hệ thống khôi phục có thể đòi hỏi một số khối nhất định khác là xuất bắt buộc (forced output) trước khi cấp quyền cho bộ quản trị buffer để xuất ra khối được yêu cầu.

TỔ CHỨC FILE

Một file được tổ chức logic như một dãy các mẩu tin (record). Các mẩu tin này được ánh xạ lên các khối đĩa. File được cung cấp như một xây dựng cơ sở trong hệ điều hành, như vậy ta sẽ giả thiết sự tồn tại của hệ thống file nền. Ta cần phải xét những phương pháp biểu diễn các mô hình dữ liệu logic trong thuật ngữ file.

Các khối có kích cỡ cố định được xác định bởi tính chất vật lý của đĩa và bởi hệ điều hành, song kích cỡ của mẩu tin lại thay đổi. Trong CSDL quan hệ, các bộ của các quan hệ khác nhau nói chung có kích cỡ khác nhau.

Một tiếp cận để ánh xạ một CSDL đến các file là sử dụng một số file, và lưu trữ các mẩu tin thuộc chỉ một độ dài cố định vào một file đã cho nào đó. Một cách khác là cấu trúc các file sao cho ta có thể điều tiết nhiều độ dài cho các mẩu tin. Các file của các mẩu tin độ dài cố định dễ dàng thực thi hơn file của các mẩu tin độ dài thay đổi.

MẪU TIN ĐỘ DÀI CỐ ĐỊNH (Fixed-Length Records)

Xét một file các mẩu tin account đối với CSDL ngân hàng, mỗi mẩu tin của file này được xác định như sau:

```
type
    depositor = record
        branch_name: char(20);
        account_number: char(10);
        balance:real;
    end
```

Giả sử mỗi một ký tự chiếm 1 byte và mỗi số thực chiếm 8 byte, như vậy mẩu tin account có độ dài 40 bytes. Một cách tiếp cận đơn giản là sử dụng 40 byte đầu tiên cho mẩu tin thứ nhất, 40 byte kế tiếp cho mẩu tin thứ hai, ... Cách tiếp cận đơn giản này nảy sinh những vấn đề sau;

0	Perryridge	A-102	400
1	Round Hill	A-305	350
2	Mianus	A-215	700
3	Downtown	A-101	500
4	Redwood	A-222	700
5	Perryridge	A-201	900
6	Brighton	A-217	750
7	Downtown	A-110	600
8	Perryridge	A-218	700

1. File F chứa các mẩu tin account

0	Perryridge	A-102	400
1	Round Hill	A-305	350
3	Downtown	A-101	500
4	Redwood	A-222	700
5	Perryridge	A-201	900
6	Brighton	A-217	750
7	Downtown	A-110	600
8	Perryridge	A-218	700

2. File F sau khi xóa mẩu tin 2 và di chuyển các mẩu tin sau nó

0	Perryridge	A-102	400
1	Round Hill	A-305	350
8	Perryridge	A-218	700
3	Downtown	A-101	500
4	Redwood	A-222	700
5	Perryridge	A-201	900
6	Brighton	A-217	750
7	Downtown	A-110	600

3. File F sau khi xóa mẩu tin 2 và di chuyển mẩu tin cuối vào vị chí của

header				
0	Perryridge	A-102	400	
1				
2	Mianus	A-215	700	
3	Downtown	A-101	500	
4				
5	Perryridge	A-201	900	
6				
7	Downtown	A-110	600	
8	Perryridge	A-218	700	

1. Khó khăn khi xoá một mẩu tin từ cấu trúc này. Không gian bị chiếm bởi mẩu tin bị xoá phải được lấp đầy với mẩu tin khác của file hoặc ta phải đánh dấu mẩu tin bị xoá.
2. Trừ khi kích cỡ khối là bội của 40, nếu không một số mẩu tin sẽ bắt chéo qua biên khối, có nghĩa là một phần mẩu tin được lưu trong một khối, một phần khác được lưu trong một khối khác. như vậy đòi hỏi phải truy xuất hai khối để đọc/viết một mẩu tin "bắc cầu" đó.

Khi một mẩu tin bị xoá, ta có thể di chuyển mẩu tin kề sau nó vào không gian bị chiếm một cách hình thức bởi mẩu tin bị xoá, rồi mẩu tin kế tiếp vào không gian bị chiếm của mẩu tin vừa được di chuyển, cứ như vậy cho đến khi mỗi mẩu tin đi sau mẩu tin bị xoá được dịch chuyển hướng về đầu. Cách tiếp cận này đòi hỏi phải di chuyển một số lớn các mẩu tin. Một cách tiếp cận khác đơn giản hơn là di chuyển mẩu tin cuối cùng vào không gian bị chiếm bởi mẩu tin bị xoá. Song cách tiếp cận này đòi hỏi phải truy xuất khối bổ xung. Vì hoạt động xen xảy ra thường xuyên hơn hoạt động xoá, ta có thể chấp nhận việc để "ngỏ" không gian bị chiếm bởi mẩu tin bị xoá, và chờ một hoạt động xen đến sau để tái sử dụng không gian đó. Một dấu trên mẩu tin bị xoá là không đủ vì sẽ gây khó khăn cho việc tìm kiếm không gian "tự do" đó khi xen. Như vậy ta cần đưa vào cấu trúc bổ xung. ở đầu của file, ta cấp phát một số byte nhất định làm header của file. Header này sẽ chứa đựng thông tin về file. Header chứa địa chỉ của mẩu tin bị xoá thứ nhất, trong nội dung của mẩu tin này có chứa địa chỉ của mẩu tin bị xoá thứ hai và cứ như vậy. Như vậy, các mẩu tin bị xoá sẽ tạo ra một danh sách liên kết dược gọi là danh sách tự do (free list). Khi xen mẩu tin mới, ta sử dụng con trỏ đầu danh sách được chứa trong header để xác định danh sách, nếu danh sách không rỗng ta xen mẩu tin mới vào vùng được trỏ bởi con trỏ đầu danh sách nếu không ta xen mẩu tin mới vào cuối file.

Xen và xoá đối với file mẩu tin độ dài cố định thực hiện đơn giản vì không gian được giải phóng bởi mẩu tin bị xoá đúng bằng không gian cần thiết để xen một mẩu tin. Đối với file của các mẩu tin độ dài thay đổi vấn đề trở nên phức tạp hơn nhiều.

MẨU TIN ĐỘ DÀI THAY ĐỔI (Variable-Length Records)

Mẩu tin độ dài thay đổi trong CSDL do bởi:

- Việc lưu trữ nhiều kiểu mẩu tin trong một file
- Kiểu mẩu tin cho phép độ dài trường thay đổi
- Kiểu mẩu tin cho phép lặp lại các trường

Có nhiều kỹ thuật để thực hiện mẩu tin độ dài thay đổi. Để minh hoạ ta sẽ xét các biểu diễn khác nhau trên các mẩu tin độ dài thay đổi có định dạng sau:

```
Type   account_list = record
                branch_name: char(20)          ;
                account_info: array[ 1.. ∞ ] of record
                        account_number: char(10);
                        balance: real;
                end;
       end
```

Biểu diễn chuỗi byte (Byte-String Representation)

Một cách đơn giản để thực hiện các mẩu tin độ dài thay đổi là ***gắn một ký hiệu đặc biệt End-of-record (⊥) vào cuối mỗi record***. Khi đó, ta có thể lưu mỗi mẩu tin như một chuỗi byte liên tiếp. Thay vì sử dụng một ký hiệu đặc biệt ở cuối của mỗi mẩu tin, một phiên bản của biểu diễn chuỗi byte ***lưu trữ độ dài mẩu tin ở bắt đầu của mỗi mẩu tin***.

0	Perryridge	A-102	400	A-201	900	A210	700	⊥
1	Round Hill	A-301	350	⊥				
2	Mianus	A-101	800	⊥				
3	Downtown	A-211	500	A-222	600	⊥		
4	Redwood	A-300	650	A-200	1200	A-255	950	⊥
5	Brighton	A-111	750	⊥				

Biểu diễn chuỗi byte của các mẩu tin độ dài thay đổi

Biểu diễn chuỗi byte có các bất lợi sau:

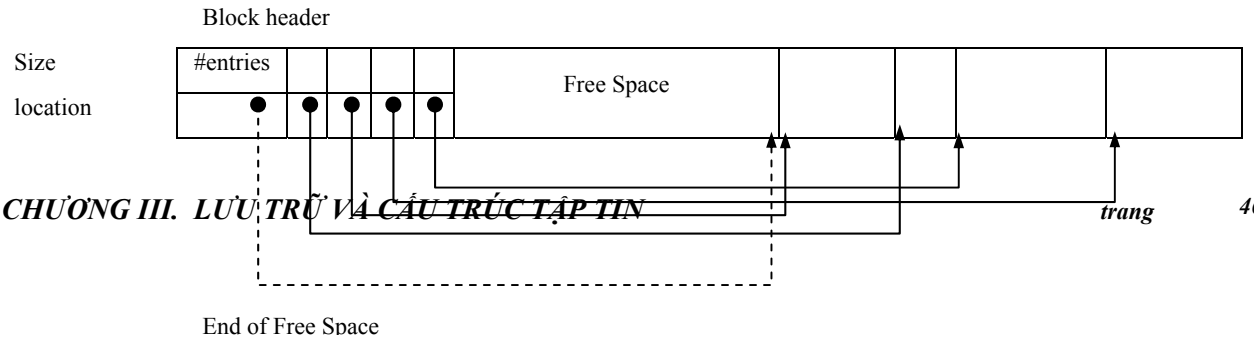
- Khó sử dụng không gian bị chiếm hình thức bởi một mẩu tin bị xoá, điều này dẫn đến một số lớn các mảnh nhỏ của lưu trữ đĩa bị lãng phí.
- Không có không gian cho sự phát triển các mẩu tin. Nếu một mẩu tin độ dài thay đổi dài ra, nó phải được di chuyển và sự di chuyển này là đắt giá nếu mẩu tin bị chốt.

Biểu diễn chuỗi byte không thường được sử dụng để thực hiện mẩu tin độ dài thay đổi, song một dạng sửa đổi của nó được gọi là ***cấu trúc khe-trang*** (slotted-page structure) thường được dùng để tổ chức mẩu tin trong một khối đơn.

Trong cấu trúc slotted-page, có một header ở bắt đầu của mỗi khối, chứa các thông tin sau:

- Số các đầu vào mẩu tin (record entries) trong header
- Điểm cuối không gian tự do (End of Free Space) trong khối
- Một mảng các đầu vào chứa vị trí và kích cỡ của mỗi mẩu tin

Các mẩu tin hiện hành được cấp phát kề nhau trong khối, bắt đàu từ cuối khối, Không gian tự do trong khối là một vùng kề nhau, nằm giữa đầu vào cuối cùng trong mảng header và mẩu tin đàu tiên. Khi một mẩu tin được xen vào, không gian cấp phát cho nó ở cuối của không gian tự do, và đầu vào tương ứng với nó được thêm vào header.



Nếu một mẩu tin bị xoá, không gian bị chiếm bởi nó được giải phóng, đầu vào ứng với nó được đặt là bị xoá (kích cỡ của nó được đặt chẳng hạn là -1). Sau đó, các mẩu tin trong khối trước mẩu tin bị xoá được di chuyển sao cho không gian tự do của khối lại là phần nằm giữa đầu vào cuối cùng của mảng header và mẩu tin đầu tiên. Con trỏ điểm cuối không gian tự do và các con trỏ ứng với mẩu tin bị di chuyển được cập nhật. Sự lớn lên hay nhỏ đi của mẩu tin cũng sử dụng kỹ thuật tương tự (trong trường hợp khối còn không gian cho sự lớn lên của mẩu tin). Cái giá phải trả cho sự di chuyển không quá cao vì các khối có kích cỡ không lớn (thường 4Kbytes).

Biểu diễn độ dài cố định

Một cách khác để thực hiện mẩu tin độ dài thay đổi một cách hiệu quả trong một hệ thống file là sử dụng một hoặc một vài mẩu tin độ dài cố định để biểu diễn một mẩu tin độ dài thay đổi. Hai kỹ thuật thực hiện file của các mẩu tin độ dài thay đổi sử dụng mẩu tin độ dài cố định là:

1. **Không gian dự trữ (reserved space)**. Giả thiết rằng các mẩu tin có độ dài không vượt quá một ngưỡng (độ dài tối đa). Ta có thể sử dụng mảu tin độ dài cố định (có độ dài tối đa), Phần không gian chưa dùng đến được lấp đầy bởi một ký tự đặc biệt: null hoặc End-of-record.
2. **Contrỏ (Pointers)**. Mẩu tin độ dài thay đổi được biểu diễn bởi một danh sách các mẩu tin độ dài cố định, được "móc xích" với nhau bởi các con trỏ.

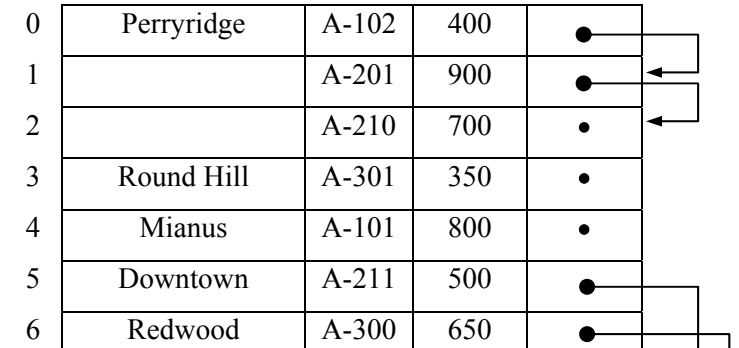
Sự bất lợi của cấu trúc con trỏ là lãng phí không gian trong tất cả các mẩu tin ngoại trừ mẩu tin đầu tiên trong danh sách (mẩu tin đầu tiên cần trường branch_name, các mẩu tin sau trong danh sách không cần thiết có trường này!). Để giải quyết vấn đề này người ta đề nghị phân các khối trong file thành hai loại:

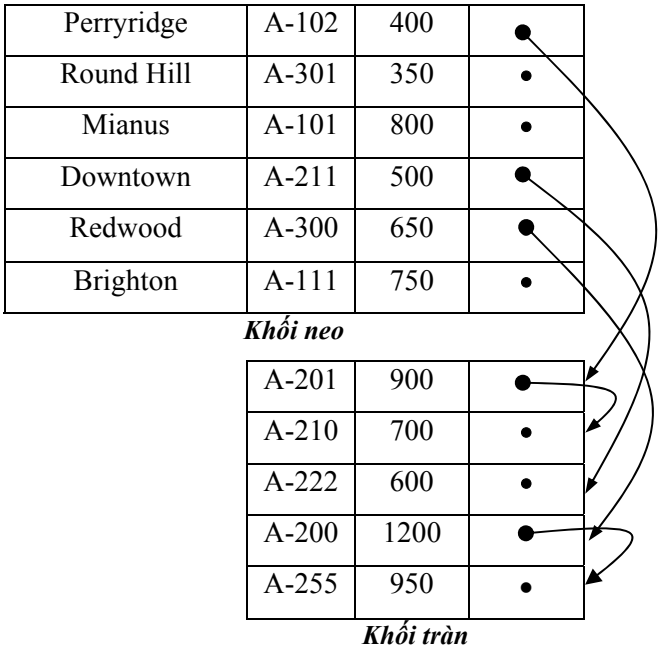
- **Khối neo (Anchor block).** chứa chỉ các mẩu tin đầu tiên trong danh sách
- **Khối tràn (Overflow block).** chứa các mẩu tin còn lại của danh sách

Như vậy, tất cả các mẩu tin trong một khối có cùng độ dài, cho dù file có thể chứa các mẩu tin không cùng độ dài.

0	Perryridge	A-102	400	A-201	900	A210	700	⊥
1	Round Hill	A-301	350	⊥	⊥	⊥	⊥	⊥
2	Mianus	A-101	800	⊥	⊥	⊥	⊥	⊥
3	Downtown	A-211	500	A-222	600	⊥	⊥	⊥
4	Redwood	A-300	650	A-200	1200	A-255	950	⊥
5	Brighton	A-111	750	⊥	⊥	⊥	⊥	⊥

Sử dụng phương pháp không gian dự trữ





Cấu trúc khối neo và khối tràn

TỔ CHỨC CÁC MẪU TIN TRONG FILE

Ta đã xét làm thế nào để biểu diễn các mẩu tin trong một cấu trúc file. Một thể hiện của một quan hệ là một tập hợp các mẩu tin. Đã cho một tập hợp các mẩu tin, vấn đề đặt ra là làm thế nào để tổ chức chúng trong một file. Có một số cách tổ chức sau:

- **Tổ chức file đống (Heap File Organization).** Trong tổ chức này, một mẩu tin bất kỳ có thể được lưu trữ ở bất kỳ nơi nào trong file, ở đó có không gian cho nó. Không có thứ tự nào giữa các mẩu tin. Một file cho một quan hệ.

- **Tổ chức file tuần tự (Sequential File Organization).** Trong tổ chức này, các mẩu tin được lưu trữ *thứ tự tuần tự*, dựa trên ***giá trị của khoá tìm kiếm*** của mỗi mẩu tin.

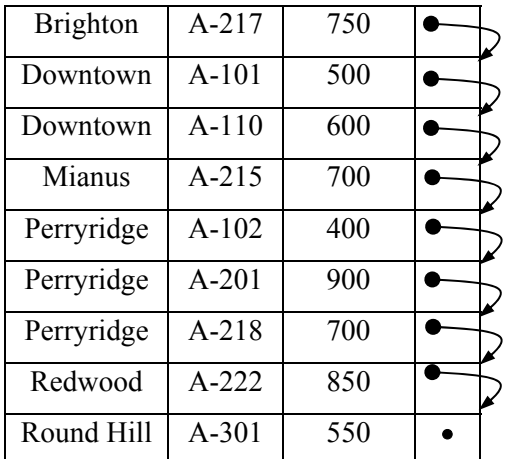
- **Tổ chức file băm (Hashed File Organization).** Trong tổ chức này, có một hàm băm được tính toán trên thuộc tính nào đó của mẩu tin. Kết quả của hàm băm xác định mẩu tin được bố trí trong khối nào trong file. Tổ chức này liên hệ chặt chẽ với cấu trúc chỉ mục.

• **Tổ chức file cụm (Clustering File Organization).** Trong tổ chức này, các mẩu tin của một vài quan hệ khác nhau có thể được lưu trữ trong cùng một file. Các mẩu tin có liên hệ của các quan hệ khác nhau được lưu trữ trên cùng một khối sao cho một hoạt động I/O đem lại các mẩu tin có liên hệ từ tất cả các quan hệ.

TỔ CHỨC FILE TUẦN TỰ

Tổ chức file tuần tự được thiết kế để xử lý hiệu quả các mẩu tin trong thứ tự được sắp dựa trên một khoá tìm kiếm (*search key*) nào đó. Để cho phép tìm lại nhanh chóng các mẩu tin theo thứ tự khoá tìm kiếm, ta "xích" các mẩu tin lại bởi các con trỏ. Con trỏ trong mỗi mẩu tin trỏ tới mẩu tin kế theo thứ tự khoá tìm kiếm. Hơn nữa, để tối ưu hoá số khối truy xuất trong xử lý file tuần tự, ta lưu trữ vật lý các mẩu tin theo thứ tự khoá tìm kiếm hoặc gần với khoá tìm kiếm như có thể.

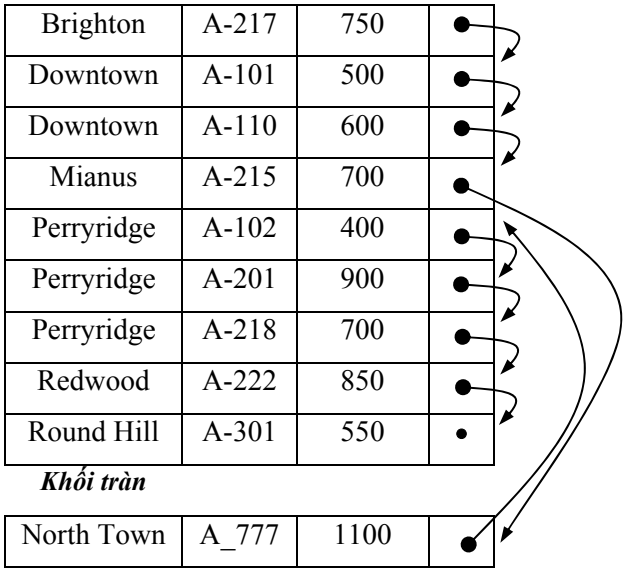
Tổ chức file tuần tự cho phép đọc các mẩu tin theo thứ tự được sắp mà nó có thể hưu dụng cho mục đích trình bày cũng như cho các thuật toán xử lý vấn tin (*query-processing algorithms*).



Brighton	A-217	750	
Downtown	A-101	500	
Downtown	A-110	600	
Mianus	A-215	700	
Perryridge	A-102	400	
Perryridge	A-201	900	
Perryridge	A-218	700	
Redwood	A-222	850	
Round Hill	A-301	550	

Khó khăn gặp phải của tổ chức này là việc duy trì thứ tự tuần tự vật lý của các mẩu tin khi xẩy ra các hoạt động xen, xoá, do cái giá phải trả cho việc di chuyển các mẩu tin khi xen, xoá. Ta có thể quản trị vấn đề xoá bởi dùng dây chuyền các con trỏ như đã trình bày trước đây. Đối với xen, ta có thể áp dụng các quy tắc sau:

1. Định vị mẩu tin trong file mà nó đi trước mẩu tin được xen theo thứ tự khoá tìm kiếm.
2. Nếu có mẩu tin tự do (không gian của mẩu tin bị xoá) trong cùng khối, xen mẩu tin vào khối này. Nếu không, xen mẩu tin mới vào một khối tràn. Trong cả hai trường hợp, điều chỉnh các con trỏ sao cho nó móc xích các mẩu tin theo thứ tự của khoá tìm kiếm.



Brighton	A-217	750	
Downtown	A-101	500	
Downtown	A-110	600	
Mianus	A-215	700	
Perryridge	A-102	400	
Perryridge	A-201	900	
Perryridge	A-218	700	
Redwood	A-222	850	
Round Hill	A-301	550	

Khối tràn

North Town	A_777	1100	

TỔ CHỨC FILE CỤM

Nhiều hệ CSDL quan hệ, mỗi quan hệ được lưu trữ trong một file sao cho có thể lợi dụng được toàn bộ những cái mà hệ thống file của điều hành cung cấp. Thông thường, các bộ của một quan hệ được biểu diễn như các mẩu tin độ dài cố định. Như vậy các quan hệ có thể ánh xạ vào một cấu trúc file. Sự thực hiện đơn giản đó của một hệ CSDL quan hệ rất phù hợp với các hệ CSDL được thiết kế cho các máy tính cá nhân. Trong các hệ thống đó, kích cỡ của CSDL nhỏ. Hơn nữa, trong một số máy tính cá nhân, chủ yếu kích cỡ tổng thể mã đối tượng đối với hệ CSDL là nhỏ. Một cấu trúc file đơn giản làm suy giảm lượng mã cần thiết để thực thi hệ thống.

Cách tiếp cận đơn giản này, để thực hiện CSDL quan hệ, không còn phù hợp khi kích cỡ của CSDL tăng lên. Ta sẽ thấy những điểm lợi về mặt hiệu năng từ việc gán một cách thận trọng các mẩu tin với các khối, và từ việc tổ chức kỹ lưỡng chính bản thân các khối. Như vậy, có vẻ như là một cấu trúc file phức tạp hơn lại có lợi hơn, ngay cả trong trường hợp ta giữ nguyên chiến lược lưu trữ mỗi quan hệ trong một file riêng biệt.

Tuy nhiên, nhiều hệ CSDL quy mô lớn không nhờ cậy trực tiếp vào hệ điều hành nền để quản trị file. Thay vào đó, một file hệ điều hành được cấp phát cho hệ CSDL. Tất cả các quan hệ được lưu trữ trong một file này, và sự quản trị file này thuộc về hệ CSDL. Để thấy những điểm lợi của việc lưu trữ nhiều quan hệ trong cùng một file, ta xét vấn tin SQL sau:

SELECT *account_number, customer_number, customer_treet, customer_city*

FROM *depositor, customer*

WHERE *depositor.customer_name = customer.customername*;

Câu vấn tin này tính một *phép nối* của các quan hệ *depositor* và *customer*. Như vậy, đối với mỗi bộ của *depositor*, hệ thống phải tìm bộ của *customer* có cùng giá trị *customer_name*. Một cách lý tưởng là việc tìm kiếm các mẩu tin này nhờ sự trợ giúp của chỉ mục. Bỏ qua việc tìm kiếm các mẩu tin như thế nào, ta chú ý vào việc truyền từ đĩa vào bộ nhớ. Trong trường hợp xấu nhất, mỗi mẩu tin ở trong một khối khác nhau, điều này buộc ta phải đọc một khối cho một mẩu tin được yêu cầu bởi câu vấn tin. Ta sẽ trình bày một cấu trúc file được thiết kế để thực hiện hiệu quả các câu vấn tin liên quan đến *depositor* ⋈ *customer*. Các bộ *depositor* đối với mỗi *customer_name* được lưu trữ gần bộ *customer* có cùng *customer_name*. Cấu trúc này trộn các bộ của hai quan hệ với nhau, nhưng cho phép xử lý hiệu quả phép nối. Khi một bộ của của quan hệ *customer* được đọc, toàn bộ khối chứa bộ này được đọc từ đĩa vào trong bộ nhớ chính. Do các bộ tương ứng của *depositor* được lưu trữ trên đĩa gần bộ *customer*, khối chứa bộ *customer* chứa các bộ của quan hệ *depositor* cần cho xử lý câu vấn tin. Nếu một *customer* có nhiều *account* đến nỗi các mẩu tin *depositor* không lấp đầy trong một khối, các mẩu tin còn lại xuất hiện trong khối kế cận. Cấu trúc file này, được gọi là ***gom cụm (clustering)***, cho phép ta đọc nhiều mẩu tin được yêu cầu chỉ sử dụng một đọc khối, như vậy ta có thể xử lý câu vấn tin đặc biệt này hiệu quả hơn.

customer_name	customer_street	customer_city
Hays	Main	Brooklyn
Turner	Putnam	Stamford

Quan hệ customer

Hays	Main	Brooklyn
Hays	A-102	
Hays	A-220	
Hays	A-503	
Turner	Putnam	Stamford
Turner	A-305	

Cấu trúc file cụm

Tuy nhiên, cấu trúc gom cụm trên lại tỏ ra không có lợi bằng tổ chức lưu mỗi quan hệ trong một file riêng, đối với một số câu vấn tin, chẳng hạn:

SELECT *

FROM *customer*

Việc xác định khi nào thì gom cụm thường phụ thuộc vào kiểu câu vấn tin mà người thiết kế CSDL nghĩ rằng nó xẩy ra thường xuyên nhất. Sử dụng thận trọng gom cụm có thể cải thiện hiệu năng đáng kể trong việc xử lý câu vấn tin.

LƯU TRỮ TỰ ĐIỂN DỮ LIỆU

Một hệ CSDL cần thiết duy trì *dữ liệu về các quan hệ,* như sơ đồ của các quan hệ. Thông tin này được gọi là ***tự điển dữ liệu (data dictionary)*** hay ***mục lục hệ thống (system catalog).*** Trong các kiểu thông tin mà hệ thống phải lưu trữ là:

- Các tên của các quan hệ
- Các tên của các thuộc tính của mỗi quan hệ
- Các miền (giá trị) và các độ dài của các thuộc tính
- Các tên của các View được định nghĩa trên CSDL và định nghĩa của các view này
- Các ràng buộc toàn vẹn

Nhiều hệ thống còn lưu trữ các thông tin liên quan đến người sử dụng hệ thống:

- Tên của người sử dụng được phép
- Giải trình thông tin về người sử dụng

Các dữ liệu thống kê và mô tả về các quan hệ có thể cũng được lưu trữ:

- Số bộ trong mỗi quan hệ
- Phương pháp lưu trữ được sử dụng cho mỗi quan hệ (cụm hay không)

Các thông tin về mỗi chỉ mục trên mỗi quan hệ cũng cần được lưu trữ :

- Tên của chỉ mục
- Tên của quan hệ được chỉ mục
- Các thuộc tính trên nó chỉ mục được định nghĩa

- Kiểu của chỉ mục được tạo

Toàn bộ các thông tin này trong thực tế bao hàm một CSDL nhỏ. Một số hệ CSDL sử dụng những cấu trúc dữ liệu và mã *mục đích đặc biệt* để lưu trữ các thông tin này. Nói chung, lưu trữ dữ liệu về CSDL trong chính CSDL vẫn được ưa chuộng hơn. Bằng cách sử dụng CSDL để lưu trữ dữ liệu hệ thống, ta đơn giản hoá cấu trúc tổng thể của hệ thống và cho phép sử dụng đầy đủ sức mạnh của CSDL trong việc truy xuất nhanh đến dữ liệu hệ thống.

Sự chọn lựa chính xác biểu diễn dữ liệu hệ thống sử dụng các quan hệ như thế nào là do người thiết kế hệ thống quyết định. Như một ví dụ, ta đề nghị sự biểu diễn sau:

System_catalog_schema = (relation_name, number_of_attributes)

Attribute_schema = (attribute_name, relation_name, domain_type, position, length)

User_schema = (user_name, encrypted_password, group)

Index_schema = (index_name, relation_name, index_type, index_attributes)

View_schema = (view_name, definition)

CHỈ MỤC

Ta xét hoạt động tìm sách trong một thư viện. Ví dụ ta muốn tìm một cuốn sách của một tác giả nào đó. Đầu tiên ta tra trong mục lục tác giả, một tấm thẻ trong mục lục này sẽ chỉ cho ta biết có thể tìm thấy cuốn sách đó ở đâu. Các thẻ trong một mục lục được thư viện sắp xếp thứ tự theo vần chữ cái , như vậy giúp ta có thể tìm đến thẻ cần tìm nhanh chóng không cần phải duyệt qua tất cả các thẻ. Chỉ mục của một file trong các công việc hệ thống rất giống với một mục lục trong một thư viện. Tuy nhiên, chỉ mục được làm như mục lục được mô tả như trên, trong thực tế, sẽ quá lớn để được quản lý một cách hiệu quả. Thay vào đó, người ta sử dụng các kỹ thuật chỉ mục tinh tế hơn. Có hai kiểu chỉ mục:

- **Chỉ mục được sắp (Ordered indices)**. được dựa trên một thứ tự sắp xếp theo các giá trị
- **Chỉ mục băm (Hash indices).** được dựa trên các giá trị được phân phối đều qua các bucket. Bucket mà một giá trị được gán với nó được xác định bởi một hàm, được gọi là hàm băm (hash function)

Đối với cả hai kiểu này, ta sẽ nêu ra một vài kỹ thuật, đáng lưu ý là không kỹ thuật nào là tốt nhất. Mỗi kỹ thuật phù hợp với các ứng dụng CSDL riêng biệt. Mỗi kỹ thuật phải được đánh giá trên cơ sở của các nhân tố sau:

- **Kiểu truy xuất**: Các kiểu truy xuất được hỗ trợ hiệu quả. Các kiểu này bao hàm cả tìm kiếm mẩu tin với một giá trị thuộc tính cụ thể hoặc tìm các mẩu tin với giá trị thuộc tính nằm trong một khoảng xác định.
- **Thời gian truy xuất**: Thời gian để tìm kiếm một hạng mục dữ liệu hay một tập các hạng mục.
- **Thời gian xen**: Thời gian để xen một hạng mục dữ liệu mới. giá trị này bao hàm thời gian để tìm vị trí xen thích hợp và thời gian cập nhật cấu trúc chỉ mục.
- **Thời gian xoá**: Thời gian để xoá một hạng mục dữ liệu. giá trị này bao hàm thời gian tìm kiếm hạng mục cần xoá, thời gian cập nhật cấu trúc chỉ mục.
- **Tổng phí tổn không gian**: Không gian phụ bị chiếm bởi một cấu trúc chỉ mục.

Một file thường đi kèm với một vài chỉ mục. Thuộc tính hoặc tập hợp các thuộc tính được dùng để tìm kiếm mẩu tin trong một file được gọi là ***khoá tìm kiếm***. Chú ý rằng định nghĩa này

khác với định nghĩa khoá sơ cấp (primary key), khoá dự tuyển (candidate key), và siêu khoá (superkey). Như vậy, nếu có một vài chỉ mục trên một file, có một vài khoá tìm kiếm tương ứng.

CHỈ MỤC ĐƯỢC SẮP.

Một chỉ mục lưu trữ các giá trị khoá tìm kiếm trong thứ tự được sắp, và kết hợp với mỗi khoá tìm kiếm, các mẩu tin chứa khoá tìm kiếm này. Các mẩu tin trong file được chỉ mục có thể chính nó cũng được sắp. Một file có thể có một vài chỉ mục trên những khoá tìm kiếm khác nhau. Nếu file chứa các mẩu tin được sắp tuần tự, chỉ mục trên khoá tìm kiếm xác định thứ tự này của file được gọi chỉ mục sơ cấp (primary index). Các chỉ mục sơ cấp cũng được gọi là chỉ mục cụm (clustering index). Khoá tìm kiếm của chỉ mục sơ cấp thường là khoá sơ cấp (khoá chính). Các chỉ mục, khoá tìm kiếm của nó xác định một thứ tự khác với thứ tự của file, được gọi là các chỉ mục thứ cấp (secondary indices) hay các chỉ mục không cụm (nonclustering indices).

Brighton	A-217	750	●
Downtown	A-101	500	●
Downtown	A-110	600	●
Mianus	A-215	700	●
Perryridge	A-102	400	●
Perryridge	A-201	900	●
Perryridge	A-218	700	●
Redwood	A-222	850	●
Round Hill	A-301	550	•

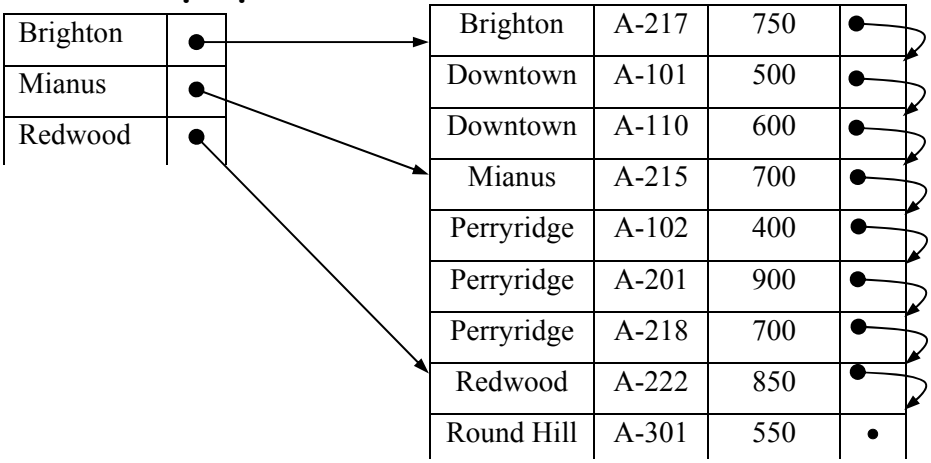
file tuần tự các mẩu tin account

Chỉ mục sơ cấp.

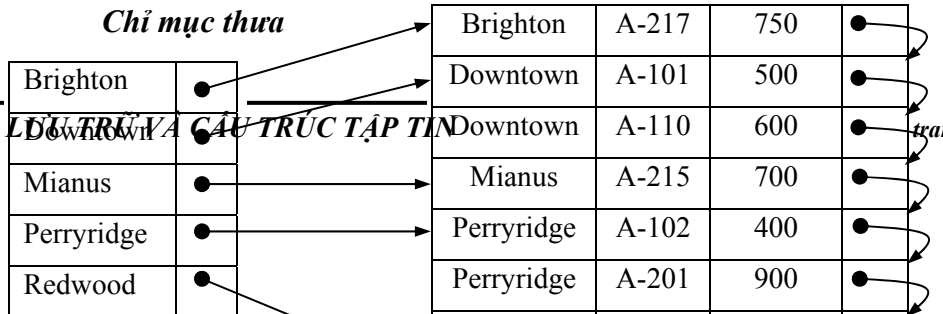
Trong phần này, ta giả thiết rằng tất cả các file được sắp thứ tự tuần tự trên một khoá tìm kiếm nào đó. Các file như vậy, với một *chỉ mục sơ cấp* trên khoá tìm kiếm này, được gọi là file tuần tự chỉ mục (index-sequential files). Chúng biểu diễn một trong các sơ đồ xưa nhất được dùng trong hệ CSDL. Chúng được thiết kế cho các ứng dụng đòi hỏi cả xử lý tuần tự toàn bộ file lẫn truy xuất ngẫu nhiên đến một mẩu tin.

Chỉ mục đặc và chỉ mục thưa (Dense and Sparse Indices)

Chỉ mục đặc



Chỉ mục thưa



Có hai loại chỉ mục được sắp:

- **Chỉ mục đặc**. Mỗi mẩu tin chỉ mục (đầu vào chỉ mục/ index entry) xuất hiện đối với mỗi giá trị khoá tìm kiếm trong file. mẩu tin chỉ mục chứa giá trị khoá tìm kiếm và một con trỏ tới mẩu tin dữ liệu đầu tiên với giá trị khoá tìm kiếm đó.
- **Chỉ mục thưa**. Một mẩu tin chỉ mục được tạo ra chỉ với một số giá trị. Cũng như với chỉ mục đặc, mỗi mẩu tin chỉ mục chứa một giá trị khoá tìm kiếm và một con trỏ tới mẩu tin dữ liệu đầu tiên với giá trị khoá tìm kiếm này. Để định vị một mẩu tin, ta tìm *đầu vào chỉ mục* với giá trị khoá tìm kiếm lớn nhất trong các giá trị khoá tìm kiếm nhỏ hơn hoặc bằng giá trị khoá tìm kiếm đang tìm. Ta bắt đầu từ mẩu tin được trỏ tới bởi đầu vào chỉ mục, và lần theo các con trỏ trong file (dữ liệu) đến tận khi tìm thấy mẩu tin mong muốn.

Ví dụ: Giả sử ta tìm các kiếm mẩu tin đối với chi nhánh Perryridge, sử dụng chỉ mục đặc. Đầu tiên, tìm Perryridge trong chỉ mục (tìm nhị phân!), đi theo con trỏ tương ứng đến mẩu tin dữ liệu (với Branch_name = Perryridge) đầu tiên, xử lý mẩu tin này, sau đó đi theo con trỏ trong mẩu tin này để định vị mẩu tin kế trong thứ tự khoá tìm kiếm, xử lý mẩu tin này, tiếp tục như vậy đến tận khi đạt tới mẩu tin có Branch_name khác với Perryridge.

Đối với chỉ mục thưa, đầu tiên tìm trong chỉ mục, đầu vào có Branch_name lớn nhất trong các đầu vào có Branch_name nhỏ hơn hoặc bằng Perryridge, ta tìm được đầu vào với Mianus, lần theo con trỏ tương ứng đến mẩu tin dữ liệu, đi theo con trỏ trong mẩu tin Mianus để định vị mẩu tin kế trong thứ tự khoá tìm kiếm và cứ như vậy đến tận khi đạt tới mẩu tin dữ liệu Perryridge đầu tiên, sau đó xử lý bắt đầu từ điểm này.

Chỉ mục đặc cho phép tìm kiếm mẩu tin nhanh hơn chỉ mục thưa, song chỉ mục thưa lại đòi hỏi ít không gian hơn chỉ mục đặc. Hơn nữa, chỉ mục thưa yêu cầu một tổn phí duy trì nhỏ hơn đối với các hoạt động xen, xoá.

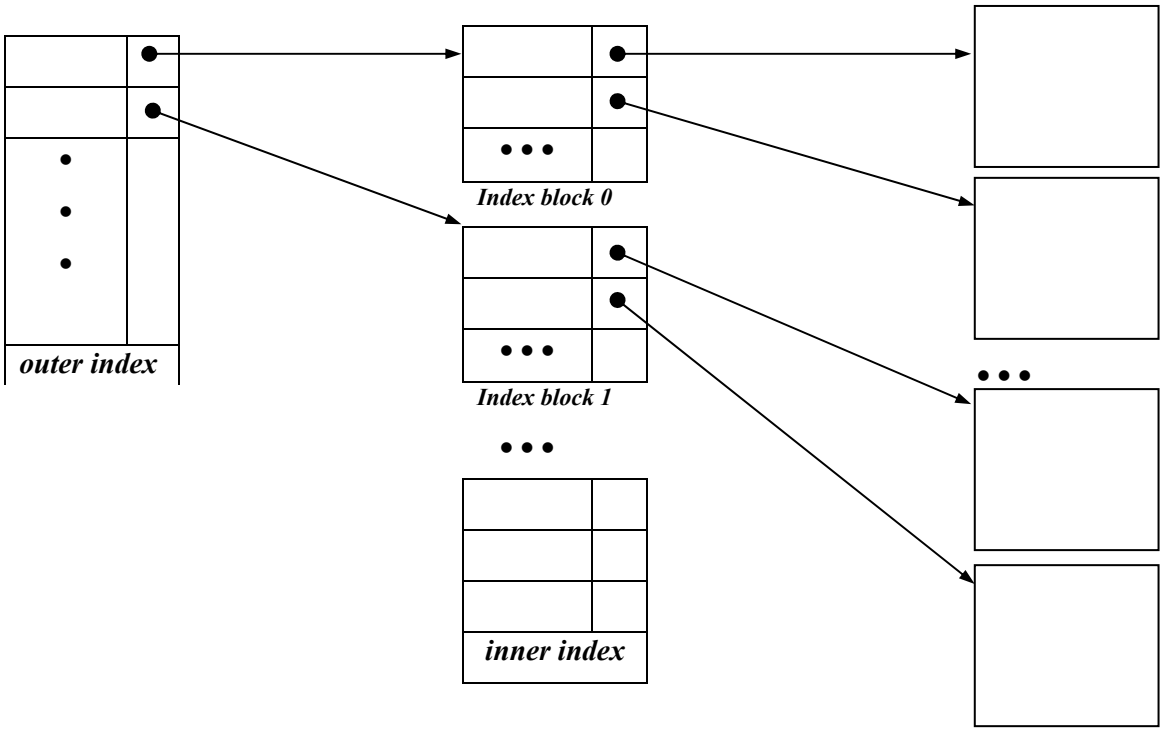
Người thiết kế hệ thống phải cân nhắc sự cân đối giữa thời truy xuất và tổn phí không gian. Một thoả hiệp tốt là có một chỉ mục thưa với một đầu vào chỉ mục cho mỗi khối, vì như vậy cái giá nổi trội trong xử lý một yêu cầu CSDL là thời gian mang một khối từ đĩa vào bộ nhớ chính. Mỗi khi một khối được mang vào, thời gian quét toàn bộ khối là không đáng kể. Sử dụng chỉ mục thưa, ta tìm khối chứa mẩu tin cần tìm. Như vậy, trừ phi mẩu tin nằm trên khối tràn, ta tối thiểu hoá được truy xuất khối, trong khi giữ được kích cỡ của chỉ mục nhỏ như có thể.

Chỉ mục nhiều mức

Chỉ mục có thể rất lớn, ngay cả khi sử dụng chỉ mục thưa, và không thể chứa đủ trong bộ nhớ một lần. Tìm kiếm đầu vào chỉ mục đối với các chỉ mục như vậy đòi hỏi phải đọc vài khối đĩa. Tìm kiếm nhị phân có thể được sử dụng để tìm một đầu vào trên file chỉ mục, song vẫn phải truy xuất khoảng $\lceil \log B \rceil$ khối, với B là số khối đĩa chứa chỉ mục. Nếu B lớn, thời gian truy xuất

này là đáng kể! Hơn nữa nếu sử dụng các khối tràn, tìm kiếm nhị phân không sử dụng được và như vậy việc tìm kiếm phải làm tuần tự. Nó đòi hỏi truy xuất lên đến B khối!!

Để giải quyết vấn đề này, Ta xem file chỉ mục như một file tuần tự và xây dựng chỉ mục thưa cho nó. Để tìm đầu vào chỉ mục, ta tìm kiếm nhị phân trên chỉ mục "ngoài" để được mẩu tin có khoá tìm kiếm lớn nhất trong các mẩu tin có khoá tìm kiếm nhỏ hơn hoặc bằng khoá muốn tìm. Con trỏ tương ứng trỏ tới khối của chỉ mục "trong". Trong khối này, tìm kiếm mẩu tin có khoá tìm kiếm lớn nhất trong các mẩu tin có khoá tìm kiếm nhỏ hơn hoặc bằng khoá muốn tìm, trường con trỏ của mẩu tin này trỏ đến khối chứa mẩu tin cần tìm. Vì chỉ mục ngoài nhỏ, có thể nằm sẵn trong bộ nhớ chính, nên một lần tìm kiếm chỉ cần một truy xuất khối chỉ mục. Ta có thể lặp lại quá trình xây dựng trên nhiều lần khi cần thiết. Chỉ mục với không ít hơn hai mức được gọi là chỉ mục nhiều mức. Với chỉ mục nhiều mức, việc tìm kiếm mẩu tin đòi hỏi truy xuất khối ít hơn đáng kể so với tìm kiếm nhị phân.



Cập nhật chỉ mục

Mỗi khi xen hoặc xoá một mẩu tin, bắt buộc phải cập nhật các chỉ mục kèm với file chứa mẩu tin này. Dưới đây, ta mô tả các thuật toán cập nhật cho các chỉ mục một mức

- **Xoá.** Để xoá một mẩu tin, đầu tiên phải tìm mẩu tin muốn xoá. Nếu mẩu tin bị xoá là mẩu tin đầu tiên trong dây chuyền các mẩu tin được xác định bởi con trỏ của đầu vào chỉ mục trong quá trình tìm kiếm, có hai trường hợp phải xét: nếu mẩu tin bị xoá là mẩu tin duy nhất trong dây chuyền, ta xoá đầu vào trong chỉ mục tương ứng, nếu không, ta thay thế khoá tìm kiếm trong đầu vào chỉ mục bởi khoá tìm kiếm của mẩu tin kề sau mẩu tin bị xoá trong dây chuyền, con trỏ bởi địa chỉ mẩu tin kề sau đó. Trong trường hợp khác, việc xoá mẩu tin không dẫn đến việc điều chỉnh chỉ mục.
- **Xen.** Trước tiên, tìm kiếm dựa trên khoá tìm kiếm của mẩu tin được xen. Nếu là chỉ mục đặc và giá trị khoá tìm kiếm không xuất hiện trong chỉ mục, xen giá trị khoá này và con trỏ tới mẩu tin vào chỉ mục. Nếu là chỉ mục thưa và lưu đầu vào cho mỗi khối, không cần

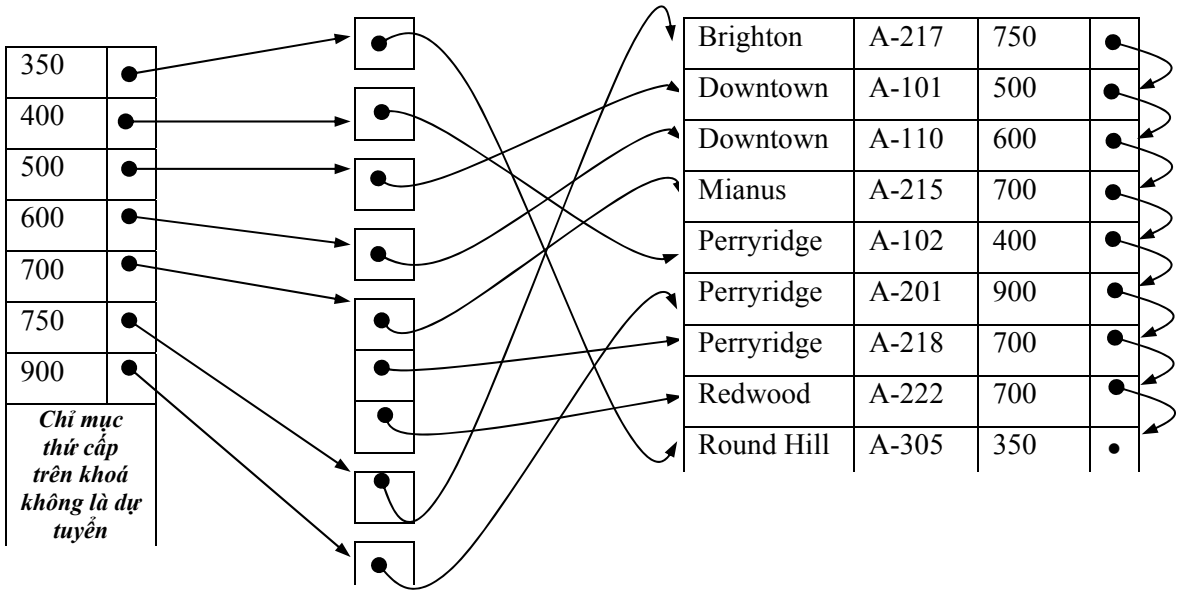
thiết phải thay đổi trừ phi khối mới được tạo ra. Trong trường hợp đó, giá trị khoá tìm kiếm đầu tiên trong khối mới được xen vào chỉ mục.

Giả thuật xen và xoá đối với chỉ mục nhiều mức là một mở rộng đơn giản của các giả thuật vừa được mô tả.

Chỉ mục thứ cấp.

Chỉ mục thứ cấp trên một khoá dự tuyển giống như chỉ mục sơ cấp đặc ngoại trừ các mẩu tin được trỏ đến bởi các giá trị liên tiếp trong chỉ mục không được lưu trữ tuần tự. Nói chung, chỉ mục thứ cấp có thể được cấu trúc khác với chỉ mục sơ cấp. Nếu khoá tìm kiếm của chỉ mục sơ cấp không là khoá dự tuyển, chỉ mục chỉ cần trỏ đến mẩu tin đầu tiên với một giá trị khoá tìm kiếm riêng là đủ (các mẩu tin khác cùng giá trị khoá này có thể tìm lại được nhờ quét tuần tự file).

Nếu khoá tìm kiếm của một chỉ mục thứ cấp không là khoá dự tuyển, việc trỏ tới mẩu tin đầu tiên với giá trị khoá tìm kiếm riêng không đủ, do các mẩu tin trong file không còn được sắp tuần tự theo khoá tìm kiếm của chỉ mục thứ cấp, chúng có thể nằm ở bất kỳ vị trí nào trong file. Bởi vậy, chỉ mục thứ cấp phải chứa tất cả các co trỏ tới mỗi mẩu tin. Ta có thể sử dụng mức phụ gián tiếp để thực hiện chỉ mục thứ cấp trên các khoá tìm kiếm không là khoá dự tuyển. Các con trỏ trong chỉ mục thứ cấp như vậy không trực tiếp trỏ tới mẩu tin mà trỏ tới một bucket chứa các con trỏ tới file.



Chỉ mục thứ cấp phải là đặc, với một đầu vào chỉ mục cho mỗi mẩu tin. Chỉ mục thứ cấp cải thiện hiệu năng các vấn tin sử dụng khoá tìm kiếm không là khóa của chỉ mục sơ cấp, tuy nhiên nó lại đem lại một tổn phí sửa đổi CSDL đáng kể.Việc quyết định các chỉ mục thứ cấp nào là cần thiết dựa trên đánh giá của nhà thiết kế CSDL về tần xuất vấn tin và sửa đổi.

FILE CHỈ MỤC B^+-CÂY (B^+-Tree Index file)

Tổ chức file chỉ mục tuần tự có một nhược điểm chính là làm giảm hiệu năng khi file lớn lên. Để khắc phục nhược điểm đó đòi hỏi phải tổ chức lại file. Cấu trúc chỉ mục B^+-cây là cấu trúc được sử dụng rộng rãi nhất trong các cấu trúc đảm bảo được tính hiệu quả của chúng bất chấp các hoạt động xen, xoá. Chỉ mục B^+-cây là một dạng cây cân bằng (mọi đường dẫn từ gốc đến lá có cùng độ dài). Mỗi nút không là lá có số con nằm trong khoảng giữa $\lceil m/2 \rceil$ và m, trong đó m là một số cố định được gọi là bậc của B^+-cây. Ta thấy rằng cấu trúc B^+-cây cũng đòi hỏi một tổn phí

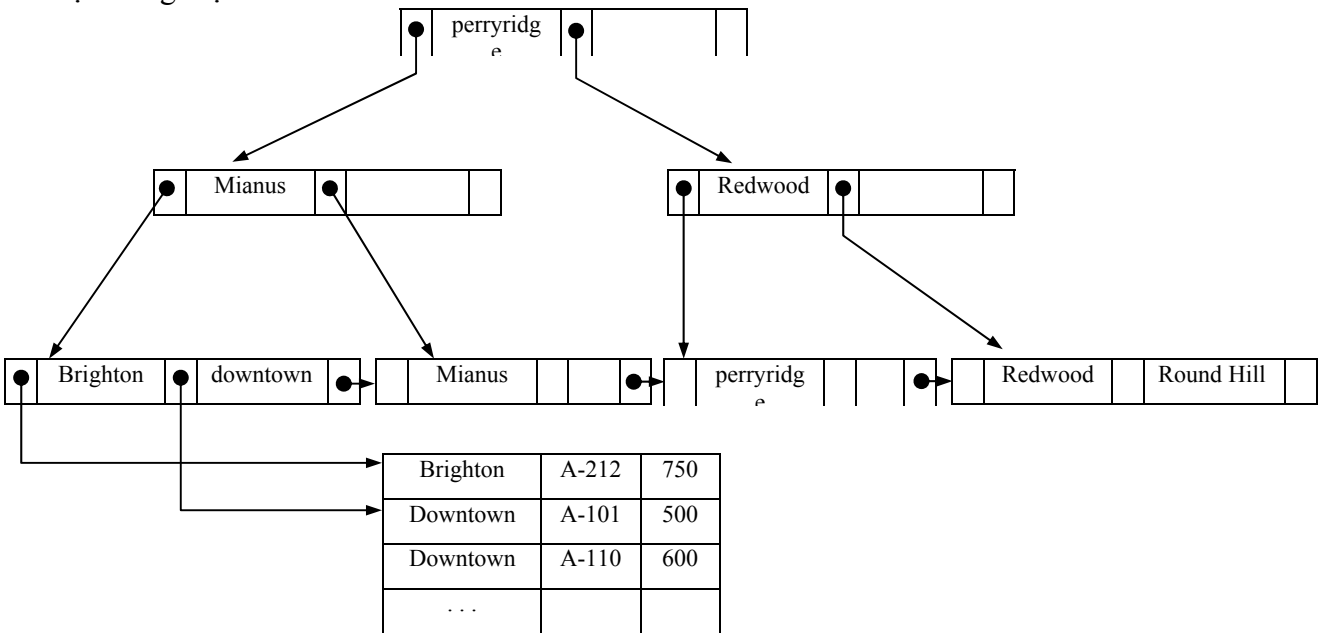
hiệu năng trên xen và xoá cũng như trên không gian. Tuy nhiên, tổn phí này là chấp nhận được ngay cả đối với các file có tần suất sửa đổi cao.

Cấu trúc của B$^+$-cây

Một chỉ mục B$^+$-cây là một chỉ mục nhiều mức, nhưng có cấu trúc khác với file tuần tự chỉ mục nhiều mức (multilevel index-sequential). Một nút tiêu biểu của B$^+$-cây chứa đến n-1 giá trị khoá tìm kiếm. K_1, K_2, ..., K_{n-1}, và n con trỏ P_1, P_2, ..., P_n, các giá trị khoá trong nút được sắp thứ tự: $i < j \Rightarrow K_i < K_j$.

P_1	K_1	P_2	K_2	. . .	P_{n-1}	K_{n-1}	P_n

Trước tiên, ta xét cấu trúc của nút lá. Đối với i = 1, 2, ..., n-1, con trỏ P_i trỏ tới hoặc mẩu tin với giá trị khoá K_i hoặc tới một bucket các con trỏ mà mỗi một trong chúng trỏ tới một mẩu tin với
giá trị khoá K_i. Cấu trúc bucket chỉ được sử dụng trong các trường hợp: hoặc khoá tìm kiếm không là khoá sơ cấp hoặc file không được sắp theo khoá tìm kiếm. Con trỏ P_n được dùng vào mục đích đặc biệt: P_n được dùng để móc xích các nút lá lại theo thứ tự khoá tìm kiếm, điều này cho phép xử lý tuần tự file hiệu quả. Bây giờ ta xem các giá trị khoá tìm kiếm được gắn với một nút lá như thế nào. Mỗi nút nút lá có thể chứa đến n-1 giá trị. Khoảng giá trị mà mỗi nút lá chứa là không chồng chéo. Như vậy, nếu L_i và L_j là hai nút lá với i < j thì mỗi giá trị khoá trong nút L_i nhỏ hơn mọi giá trị khoá trong L_j . Nếu chỉ mục B$^+$-cây là đặc, mỗi giá trị khoá tìm kiếm phải xuất hiện trong một nút lá nào đó.



Các nút không là lá của một B$^+$-cây tạo ra một chỉ mục nhiều mức trên các nút lá. Cấu trúc của các nút không là lá tương tự như cấu trúc nút lá ngoại trừ tất cả các con trỏ đều trỏ đến các nút của cây. Các nút không là lá có thể chứa đến m con trỏ và phải chứa không ít hơn $\lceil m/2 \rceil$ con trỏ ngoại trừ nút gốc. Nút gốc được phép chứa ít nhất 2 con trỏ. Số con trỏ trong một nút được gọi là số nan (fanout) của nút.

Con trỏ P_i của một nút không là lá (chứa p con trỏ, 1 < i < p) trỏ đến một cây con chứa các giá trị khoá tìm kiếm nhỏ hơn K_i và lớn hơn hoặc bằng K_{i-1}. Con trỏ P_1 trỏ đến cây con chứa các giá trị khoá tìm kiếm nhỏ hơn K_1. Con trỏ P_p trỏ tới cây con chứa các khoá tìm kiếm lớn hơn K_{p-1}.

Các vấn tin trên B^+-cây

Ta xét xử lý vấn tin sử dụng B^+-cây như thế nào ? Giả sử ta muốn tìn tất cả các mẩu tin với giá trị khoá tìm kiếm k. Đầu tiên, ta kiểm tra nút gốc, tìm giá trị khoá tìm kiếm nhỏ nhất lớn hơn k, giả sử giá trị khoá đó là K_i. Đi theo con trỏ Pi để di tới một nút khác. Nếu nút có p con trỏ và $k > K_{p-1}$, đi theo con trỏ P_p. Đến một nút tới, lặp lại quá trình tìm kiếm giá trị khoá tìm kiếm nhỏ nhất lớn hơn k và theo con trỏ tương ứng để đi tới một nút khác và tiếp tục như vậy đến khi đạt tới một nút lá. Con trỏ tương ứng trong nút lá hướng ta tới mẩu tin/bucket mong muốn. Số khối truy xuất không vượt quá $\left\lceil \log_{\lceil m/2 \rceil} K \right\rceil$, trong đó K là số giá trị khoá tìm kiếm trong B^+-cây, m là bậc của cây.

Cập nhật trên B^+-cây

- **Xen**. Sử dụng cùng kỹ thuật như tìm kiếm, ta tìm nút lá trong đó giá trị khoá tìm kiếm cần xen sẽ xuất hiện. Nếu khoá tìm kiếm đã xuất hiện rồi trong nút lá, xen mẩu tin vào trong file, thêm con trỏ tới mẩu tin vào trong bucket tương ứng. Nếu khoá tìm kiếm chưa hiện diện trong nút lá, ta xen mẩu tin vào trong file rồi xen giá trị khoá tìm kiếm vào trong nút lá ở vị trí đúng (bảo tồn tính thứ tự), tạo một bucket mới với con trỏ tương ứng. Nếu nút lá không còn chỗ cho giá trị khoá mới, Một khối mới được yêu cầu từ hệ điều hành, các giá trị khoá trong nút lá được tách một nửa cho nút mới, giá trị khoá mới được xen vào vị trí đúng của nó vào một trong hai khối này. Điều này kéo theo việc xen giá trị khoá đầu khối mới và con trỏ tới khối mới vào nút cha. Việc xen cặp giá trị khoá và con trỏ vào nút cha này lại có thể dẫn đến việc tách nút ra làm hai. Quá trình này có thể dẫn đến tận nút gốc. Trong trường hợp nút gốc bị tách làm hai, một nút gốc mới được tạo ra và hai con của nó là hai nút được tách ra từ nút gốc cũ, chiều cao cây tăng lên một.

Procedure Insert(*value* V, *pointer* P)

Tìm nút lá L sẽ chứa giá trị V

Insert_entry(L, V, P)

end procedure

Procedure Insert_entry(*node* L, *value* V, *pointer* P)

If (L có không gian cho (V, P) **then**

Xen (V, P) vào L

else begin /* tách L */

Tạo nút L'

If (L là nút lá) **then begin**

V' là giá trị sao cho $\lceil m/2 \rceil$ giá trị trong các giá trị $L.K_1$, $L.K_2$, ..., $L.K_{m-1}$, V nhỏ hơn V'

n là chỉ số nhỏ nhất sao cho $L.K_n \geq V'$

Di chuyển $L.P_n$, $L.K_n$, ..., $L.P_{m-1}$, $L.K_{n-1}$ sang L'

If (V < V') then xen (V, P) vào trong L else xen (P, V) vào trong L'

end else begin

V' là giá trị sao cho $\lceil m/2 \rceil$ giá trị trong các giá trị $L.K_1$, $L.K_2$, ..., $L.K_{m-1}$, V lớn hơn hoặc bằng V'

n là chỉ số nhỏ nhất sao cho $L.K_n \geq V'$

Thêm Nil, $L.K_n$, $L.P_{n+1}$, $L.K_{n+1}$, ..., $L.P_{m-1}$, $L.K_{m-1}$, $L.P_m$ vào L'

Xoá $L.K_n$, $L.P_{n+1}$, $L.K_{n+1}$, ..., $L.P_{m-1}$, $L.K_{m-1}$, $L.P_m$ khỏi L

```
                    If (V < V') then xen (P, V) vào trong L  else  xen (P, V) vào trong L'
                    xoá (Nil, V') khỏi L'
                end
                If (L không là nút gốc) then Insert_entry(parent(L), V', L')
                else begin
                    Tạo ra nút mới R với các nút con là L và L' với giá trị duy nhất trong nó là V'
                    Tạo R là gốc của cây
                end
                If (L) là một nút lá then begin
                    đặt L'.Pm = L.Pm
                    đặt L.Pm = L'
                end
        end
end procedure
```

- **Xoá**. Sử dụng kỹ thuật tìm kiếm tìm mẩu tin cần xoá, xoá nó khỏi file, xoá giá trị khoá tìm kiếm khỏi nút lá trong B^+-cây nếu không có bucket kết hợp với giá trị khoá tìm kiếm hoặc bucket trở nên rỗng sau khi xoá con trỏ tương ứng trong nó. Việc xoá một giá trị khoá khỏi một nút của B^+-cây có thể dẫn đến nút lá trở nên rỗng, phải trả lại, từ đó nút cha của nó có thể có số con nhỏ hơn ngưỡng cho phép, trong trường hợp đó hoặc phải chuyển một con từ nút anh em của nút cha đó sang nút cha nếu điều đó có thể (nút anh em của nút cha này còn số con ≥ $\lceil m/2 \rceil$ sau khi chuyển đi một con). Nếu không, phải gom nút cha này với một nút anh em của nó, điều này dẫn tới xoá một nút trong khỏi cây, rồi xoá khỏi nút cha của nó một hạng, ... quá trình này có thể dẫn đến tận gốc. Trong trường hợp nút gốc chỉ còn một con sau xoá, cây phải thay nút gốc cũ bởi nút con của nó, nút gốc cũ phải trả lại cho hệ thống, chiều cao cây giảm đi một.

```
Procedure delete(value V, pointer P)
        Tìm nút lá chứa (V, P)
        delete_entry(L, V, P)
end procedure
Procedure delete_entry(node L, value V, pointer P)
        xoá (V, P) khỏi L
        If (L là nút gốc and L chỉ còn lại một con) then
                Lấy con của L làm nút gốc mới của cây, xoá L
        else If (L có quá ít giá trị/ con trỏ) then begin
                L' là anh em kề trái hoặc phải của L
                V' là giá trị ở giữa hai con trỏ L, L' (trong nút parent(L))
                If (các đầu vào của L và L' có thể lấp đầy trong một khối) then begin
                    If (L là nút trước của L') then wsap_variables(L, L')
                    If (L không là lá) then nối V' và tất cả con trỏ, giá trị trong L với L'
                    else  begin nối tất cả các cặp (K, P) trong L với L'; L'.Pp = L.Pp  end
```

```
            delete_entry(parent(L), V', L); xoá nút L
        end
        else begin
            If (L' là nút trước của L) then begin
                If (L không là nút lá) then begin
                    p là chỉ số sao cho L'.P_p là con trỏ cuối trong L'
                    xoá (L'.K_p-1, L'.P_p) khỏi L'
                    xen (L'.P_p, V') như phần tử đầu tiên trong L (right_shift tất cả các phần tử của L)
                    thay thế V' trong parent(L) bởi L'.K_p-1
                end else begin
                    p là chỉ số sao cho L'.P_p là con trỏ cuối trong L'
                    xoá (L'.P_p, L'.K_p) khỏi L'
                    xen (L'.P_p, L'.K_p) như phần tử đầu tiên trong L (right_shift tất cả các phần tử của L)
                    thay thế V' trong parent(L) bởi L'.K_p
                end
        end < đối xứng với trường hợp then >
    end
end procedure
```

Tổ chức file B^+-cây

Trong tổ chức file B^+-cây, các nút lá của cây lưu trữ các mẩu tin, thay cho các con trỏ tới file. Vì mẩu tin thường lớn hơn con trỏ, số tối đa các mẩu tin được lưu trữ trong một khối lá ít hơn số con trỏ trong một nút không lá. Các nút lá vẫn được yêu cầu được lấp đầy ít nhất là một nửa.

Xen và xoá trong tổ chức file B^+-cây tương tự như trong chỉ mục B^+-cây.

Khi B^+-cây được sử dụng để tổ chức file, việc sử dụng không gian là đặc biệt quan trọng, vì không gian bị chiếm bởi mẩu tin là lớn hơn nhiều so với không gian bị chiếm bởi (khoá,con trỏ). Ta có thể cải tiến sự sử sụng không gian trong B^+-cây bằng cách bao hàm nhiều nút anh em hơn khi tái phân phối trong khi tách và trộn. Khi xen, nếu một nút là đầy, ta thử phân phối lại một số đầu vào đến một trong các nút kề để tạo không gian cho đầu vào mới. Nếu việc thử này thất bại, ta mới thực hiện tách nút và phân chia các đầu vào giữa một trong các nút kề và hai nút nhận được do tách nút. Khi xoá, nếu nút chứa ít hơn $\lfloor 2m/3 \rfloor$ đầu vào, ta thử mượn một đầu vào từ một trong hai nút anh em kề. Nếu cả hai đều có đúng $\lfloor 2m/3 \rfloor$ mẩu tin, ta phân phối lại các đầu vào của nút cho hai nút anh em kề và xoá nút thứ 3. Nếu k nút được sử dụng trong tái phân phối (k-1 nút anh em), mỗi nút đảm bảo chứa ít nhất $\lfloor (k-1)m/k \rfloor$ đầu vào. Tuy nhiên, cái giá phải trả cho cập nhật của cách tiếp cận này sẽ cao hơn.

FILE CHỈ MỤC B-CÂY (B-Tree Index Files)

Chỉ mục B-cây tương tự như chỉ mục B^+-cây. Sự khác biệt là ở chỗ B-cây loại bỏ lưu trữ dư thừa các giá trị khoá tìm kiếm. Trong B-cây, các giá trị khoá chỉ xuất hiện một lần. Do các khoá tìm kiếm xuất hiện trong các nút không là lá không xuất hiện ở bất kỳ nơi nào khác nữa trong B-cây, ta phải thêm một trường con trỏ cho mỗi khoá tìm kiếm trong các nút không là lá. Con trỏ thêm vào này trỏ tới hoặc mẩu tin trong file hoặc bucket tương ứng.

Một nút lá B-cây tổng quát có dạng:

P_1	K_1	P_2	K_2	...	P_{m-1}	K_{m-1}	P_m

Một nút không là lá có dạng:

P_1	B_1	K_1	P_2	B_2	K_2		$P_{...}$	$B_{m-...}$	$K_{...}$	P

Các con trỏ P_i là các con trỏ cây và được dùng như trong B^+-cây. Các con trỏ B_i trong các nút không là lá là các con trỏ mẩu tin hoặc con trỏ bucket. Rõ ràng là số giá trị khoá trong nút không lá nhỏ hơn số giá trị trong nút lá. Số nút được truy xuất trong quá trình tìm kiếm trong một B-cây phụ thuộc nơi khoá tìm kiếm được định vị.

Xoá trong một B-cây phức tạp hơn trong một B^+-cây. Xoá một đầu vào xuất hiện ở một nút không là lá kéo theo việc tuyển chọn một giá trị thích hợp trong cây con của nút chứa đầu vào bị xoá. Nếu khoá K_i bị xoá, khoá nhỏ nhất trong cây con được trỏ bởi P_{i+1} phải được di chuyển vào vị trí của K_i. Nếu nút lá còn lại quá ít đầu vào, cần thiết các hoạt động bổ xung.

III.9.4 Định nghĩa chỉ mục trong SQL

Một chỉ mục được tạo ra bởi lệnh **CREATE INDEX** với cú pháp

CREATE INDEX < index-name > **ON** < relation_name > (< attribute-list >)

attribute-list là danh sách các thuộc tính của quan hệ được dùng làm khoá tìm kiếm cho chỉ mục. Nếu muốn khai báo là khoá tìm kiếm là khoá dự tuyển, thêm vào từ khoá UNIQUE:

CREATE UNIQUE INDEX < index-name > **ON** < relation_name > (< attribute-list >)

attribute-list phải tạo thành một khoá dự tuyển, nếu không sẽ có một thông báo lỗi.

Bỏ đi một chỉ mục sử dụng lệnh DROP:

DROP INDEX < index-name >

BĂM (HASHING)

BĂM TĨNH (Static Hashing)

Bất lợi của tổ chức file tuần tự là ta phải truy xuất một cấu trúc chỉ mục để định vị dữ liệu, hoặc phải sử dụng tìm kiếm nhị phân, và kết quả là có nhiều hoạt động I/O. Tổ chức file dựa trên kỹ thuật băm cho phép ta tránh được truy xuất một cấu trúc chỉ mục. Băm cung cung cấp một phương pháp để xây dựng các chỉ mục.

Tổ chức file băm

Trong tổ chức file băm, ta nhận được địa chỉ của khối đĩa chứa một mẩu tin mong muốn bởi tính toán một hàm trên giá trị khoá tìm kiếm của mẩu tin. thuật ngữ bucket được dùng để chỉ một đơn vị lưu trữ. Một bucket kiểu mẫu là một khối đĩa, nhưng có thể được chọn nhỏ hơn hoặc lớn hơn một khối đĩa.

K ký hiệu tập tất cả các giá trị khoá tìm kiếm, B ký hiệu tập tất cả các địa chỉ bucket. Một hàm băm h là một hàm từ K vào B : h: $K \rightarrow \mathrm{B}$

Xen một mẩu tin với giá trị khoá K vào trong file: ta tính h(K). Giá trị của h(K) là địa chỉ của bucket sẽ chứa mẩu tin. Nếu có không gian trong bucket cho mẩu tin, mẩu tin được lưu trữ trong bucket.

Tìm kiếm một mẩu tin theo giá trị khoá K: đầu tiên tính h(K), ta tìm được bucket tương ứng. sau đó tìm trong bucket này mẩu tin với giá trị khoá K mong muốn.

Xoá mẩu tin với giá trị khoá K: tính h(K), tìm trong bucket tương ứng mẩu tin mong muốn, xoá nó khỏi bucket.

Hàm băm

Hàm băm xấu nhất là hàm ánh xạ tất cả các giá trị khoá vào cùng một bucket. Hàm băm lý tưởng là hàm phân phối ***đều*** các giá trị khoá vào các bucket, như vậy mỗi bucket chứa một số lượng mẩu tin như nhau. Ta muốn chọn một hàm băm thoả mãn các tiêu chuẩn sau:

- ***Phân phối đều***: Mỗi bucket được gán cùng một số giá trị khoá tìm kiếm trong tập hợp tất cả các giá trị khoá có thể
- ***Phân phối ngẫu nhiên***: Trong trường hợp trung bình, các bucket được gán một số lượng giá trị khoá tìm kiếm *gần bằng nhau*.

Các hàm băm phải được thiết kế thận trọng. Một hàm băm xấu có thể dẫn đến việc tìm kiếm chiếm một thời gian tỷ lệ với số khoá tìm kiếm trong file.

Điều khiển tràn bucket

Khi xen một mẩu tin, nếu bucket tương ứng còn chỗ, mẩu tin được xen vào bucket, nếu không sẽ xảy ra tràn bucket. Tràn bucket do các nguyên do sau:

- **Các bucket không đủ.** Số các bucket n_B phải thoả mãn $n_B > n_r / f_r$ trong đó n_r là tổng số mẩu tin sẽ lưu trữ, f_r là số mẩu tin có thể lấp đầy trong một bucket.
- **Sự lệch.** Một vài bucket được gán cho một số lượng mẩu tin nhiều hơn các bucket khác, như vậy một bucket có thể tràn trong khi các bucket khác vẫn còn không gian. Tình huống này được gọi là sự lệch bucket. Sự lệch xảy ra do hai nguyên nhân:
  1. Nhiều mẩu tin có cùng khoá tìm kiếm
  2. Hàm băm được chọn phân phối các giá trị khoá không đều

Ta quản lý tràn bucket bằng cách dùng các bucket tràn. Nếu một mẩu tin phải được xen vào bucket B nhưng bucket B đã đầy, khi đó một bucket tràn sẽ được cấp cho B và mẩu tin được xen vào bucket tràn này. Nếu bucket tràn cũng đầy một bucket tràn mới lại được cấp và cứ như vậy. Tất cả các bucket tràn của một bucket được "móc xích" với nhau thành một danh sách liên kết. Việc điều khiển tràn dùng danh sách liên kết như vậy được gọi là dây chuyền tràn. Đối với dây chuyền tràn, thuật toán tìm kiếm thay đổi chú ít: trước tiên ta cũng tính giá trị hàm băm trên khoá tìm kiếm, ta được bucket B, kiểm tra các mẩu tin, trong bucket B và tất cả các bucket tràn tương ứng, có giá trị khoá khớp với giá trị tìm không.

Một cách điều khiển tràn bucket khác là: Khi cần xen một mẩu tin vào một bucket nhưng nó đã đầy, thay vì cấp thêm một bucket tràn, ta sử dụng một hàm băm kế trong một dãy các hàm băm được chọn để tìm bucket khác cho mẩu tin, nếu bucket sau cũng đầy, ta lại sử dụng một hàm băm kế và cứ như vậy... Dãy các hàm băm thường được sử dụng là { $h_i(K) = (h_{i-1}(K) + 1) \bmod n_B$ với $1 \le i \le n_B - 1$ và h_0 là hàm băm cơ sở }.

Dạng cấu trúc băm sử dụng dây chuyền bucket được gọi là băm mở. Dạng sử dụng dãy các hàm băm được gọi là băm đóng. Trong các hệ CSDL, cấu trúc băm đóng thường được ưa dùng hơn.

Chỉ mục băm

Một chỉ mục băm tổ chức các khoá tìm kiếm cùng con trỏ kết hợp vào một cấu trúc file băm như sau: áp dụng một hàm băm trên khoá tìm kiếm để định danh bucket sau đó lưu giá trị khoá và con trỏ kết hợp vào bucket này (hoặc vào các bucket tràn). Chỉ mục băm thường là chỉ mục thứ cấp.

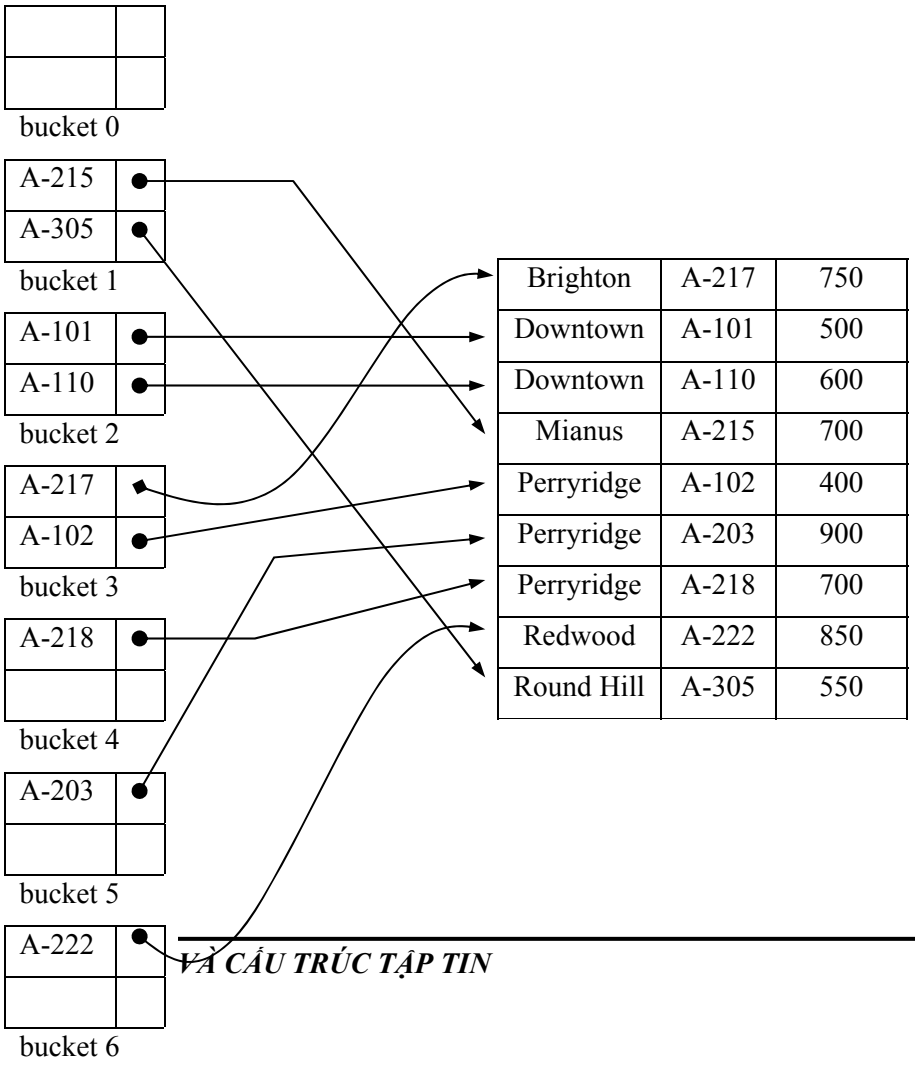
Hàm băm trên số tài khoản được tính theo công thức:

h(Account_number) = (tổng các chữ số trong số tài khoản) mod 7

BĂM ĐỘNG (Dynamic Hashing)

Trong kỹ thuật băm tĩnh (static hashing), tập B các địa chỉ bucket phải là cố định. Các CSDL phát triển lớn lên theo thời gian. Nếu ta sử dụng băm tĩnh cho CSDL, ta có ba lớp lựa chọn:

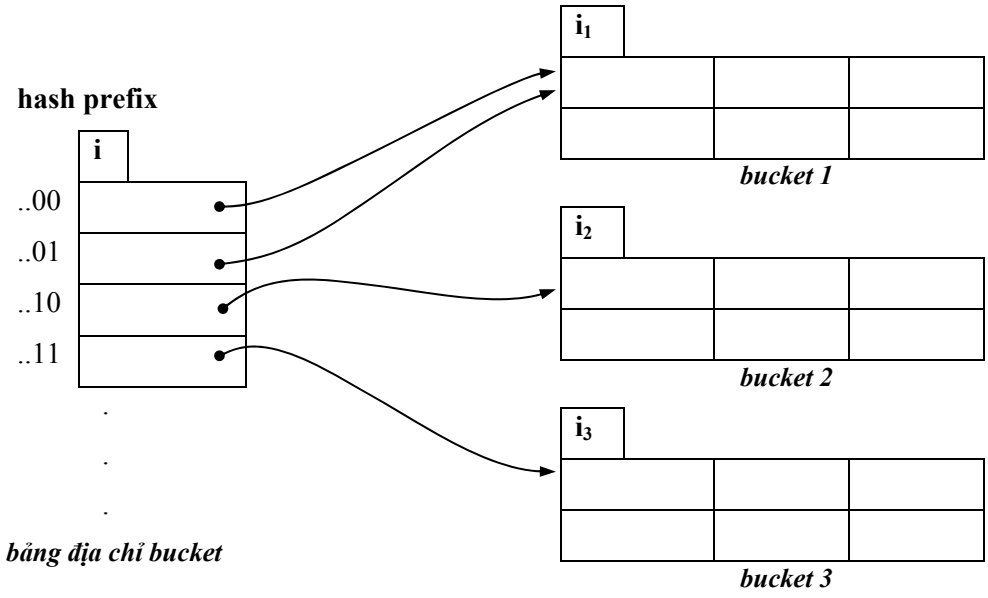
1. Chọn một hàm băm dựa trên kích cỡ file hiện hành. Sự lựa chọn này sẽ dẫn đến sự suy giảm hiệu năng khi CSDL lớn lên.
2. Chọn một hàm băm dựa trên kích cỡ file dự đoán trước cho một thời điểm nào đó trong tương lai. Mặc dù sự suy giảm hiệu năng được cải thiện, một lượng đáng kể không gian có thể bị lãng phí lúc khởi đầu.
3. Tổ chức lại theo chu kỳ cấu trúc băm đáp ứng sự phát triển kích cỡ file. Một sự tổ chức lại như vậy kéo theo việc lựa chọn một hàm băm mới, tính lại hàm băm trên mỗi mẩu tin trong file và sinh ra các gán bucket mới. Tổ chức lại là một hoạt động tốn thời gian. Hơn nữa, nó đòi hỏi cấm truy xuất file trong khi đang tổ chức lại file.



Chỉ mục băm trên khoá tìm kiếm account-number của file account

Kỹ thuật băm động cho phép sửa đổi hàm băm để phù hợp với sự tăng hoặc giảm của CSDL. Một dạng băm động được gọi là băm có thể mở rộng (extendable hashing) được thực hiện như sau: Chọn một hàm băm h với các tính chất đều, ngẫu nhiên và có miền giá trị tương đối rộng, chẳng hạn, là một số nguyên b bit (b thường là 32). Khi khởi đầu ta không sử dụng toàn bộ b bit giá trị băm. Tại một thời điểm, ta chỉ sử dụng i bit $0 \le i \le b$. i bit này được dùng như một độ dời (offset) trong một bảng địa chỉ bucket phụ. giá trị i tăng lên hay giảm xuống tuỳ theo kích cỡ CSDL.

Số i xuất hiện bên trên bảng địa chỉ bucket chỉ ra rằng i bit của giá trị băm h(K) được đòi hỏi để xác định bucket đúng cho K, số này sẽ thay đổi khi kích cỡ file thay đổi. Mặc dù i bit dược đòi hỏi để tìm đầu vào đúng trong bảng địa chỉ bucket, một số đầu vào bảng kề nhau có thể trỏ đến cùng một bucket. Tất cả các như vậy có chung hash prefix chung, nhưng chiều dài của prefix này có thể nhỏ hơn i. Ta kết hợp một số nguyên chỉ độ dài của hash prefix chung này, ta sẽ ký hiệu số nguyên kết hợp với bucket j là i_j. Số các đầu vào bảng địa chỉ bucket trỏ đến bucket j là $2^{(i-i_j)}$.



Cấu trúc băm có thể mở rộng tổng quát

Để định vị bucket chứa giá trị khoá tìm kiếm K , ta lấy i bit cao đầu tiên của h(K), tìm trong đầu vào bảng tương ứng với chuỗi bit này và lần theo con trỏ trong đầu vào bảng này. Để xen một mẩu tin với giá trị khoá tìm kiếm K, tiến hành thủ tục dịnh vị trên, ta được bucket, giả sử là bucket j. Nếu còn cho cho mẩu tin, xen mẩu tin vào trong bucket đó. Nếu không, ta phải tách bucket ra và phân phối lại các mẩu tin hiện có cùng mẩu tin mới. Để tách bucket, đầu tiên ta xác định từ giá trị băm có cần tăng số bit lên hay không.

- Nếu $i = i_j$, chỉ có một đầu vào trong bảng địa chỉ bucket trỏ đến bucket j. ta cần tăng kích cỡ của bảng địa chỉ bucket sao cho ta có thể bao hàm các con trỏ đến hai bucket kết quả

của việc tách bucket j bằng cách xét thêm một bit của giá trị băm. tăng giá trị i lên một, như vậy kích cỡ của bảng địa chỉ bucket tăng lên gấp đôi. Mỗi một đầu vào được thay bởi hai đầu vào, cả hai cùng chứa con trỏ của đầu vào gốc. Bây giờ hai đầu vào trong bảng địa chỉ bucket trỏ tới bucket j. Ta định vị một bucket mới (bucket z), và đặt đầu vào thứ hai trỏ tới bucket mới, đặt i_j và i_z về i, tiếp theo đó mỗi một mẩu tin trong bucket j được băm lại, tuỳ thuộc vào i bit đầu tiên, sẽ hoặc ở lại bucket j hoặc được cấp phát cho bucket mới được tạo.

- Nếu $i > i_j$ khi đó nhiều hơn một đầu vào trong bảng địa chỉ bucket trỏ tới bucket j. như vậy ta có thể tách bucket j mà không cần tăng kích cỡ bảng địa chỉ bucket. Ta cấp phát một bucket mới (bucket z) và đặt i_j và i_z đến giá trị là kết quả của việc thêm 1 vào giá trị i_j gốc. Kế đến, ta điều chỉnh các đầu vào trong bảng địa chỉ bucket trước đây trỏ tới bucket j. Ta để lại nửa đầu các đầu vào, và đặt tất cả các đầu vào còn lại trỏ tới bucket mới tạo (z). Tiếp theo, mỗi mẩu tin trong trong bucket j được băm lại và được cấp phát cho hoặc vào bucket j hoặc bucket z.

Để xoá một mẩu tin với giá trị khoá K, trước tiên ta thực hiện thủ tục định vị, ta tìm được bucket tương ứng, gọi là j, ta xoá cả khoá tìm kiếm trong bucket lẫn mẩu tin mẩu tin trong file. bucket cũng bị xoá, nếu nó trở nên rỗng. Chú ý rằng, tại điểm này, một số bucket có thể được kết hợp lại và kích cỡ của bảng địa chỉ bucket sẽ giảm đi một nửa.

Ưu điểm chính của băm có thể mở rộng là hiệu năng không bị suy giảm khi file tăng kích cỡ, hơn nữa, tổng phí không gian là tối tiểu mặc dù phải thêm vào không gian cho bảng địa chỉ bucket. Một khuyết điểm của băm có thể mở rộng là việc tìm kiếm phải bao hàm một mức gián tiếp: ta phải truy xuất bảng địa chỉ bucket trước khi truy xuất đến bucket. Vì vậy, băm có thể mở rộng là một kỹ thuật rất hấp dẫn.

CHỌN CHỈ MỤC HAY BĂM ?

Ta đã xét qua các sơ đồ: chỉ mục thứ tự, băm. Ta có thể tổ chức file các mẩu tin bởi hoặc sử dụng tổ chức file tuần tự chỉ mục, hoặc sử dụng B^+-cây, hoặc sử dụng băm ... Mỗi sơ đồ có những các ưu điểm trong các tình huống nhất định. Một nhà thực thi hệ CSDL có thể cung cấp nhiều nhiều sơ đồ, để lại việc quyết định sử dụng sơ đồ nào cho nhà thiết kế CSDL. Để có một sự lựa chọn khôn ngoan, nhà thực thi hoặc nhà thiết kế CSDL phải xét các yếu tố sau:

- Cái giá phải trả cho việc tổ chức lại theo định kỳ của chỉ mục hoặc băm có chấp nhận được hay không?
- Tần số tương đối của các hoạt động xen và xoá là bao nhiêu ?
- Có nên tối ưu hoá thời gian truy xuất trung bình trong khi thời gian truy xuất trường hợp xấu nhất tăng lên hay không ?
- Các kiểu vấn tin mà các người sử dụng thích đặt ra là gì ?

CẤU TRÚC LƯU TRỮ CHO CSDL HƯỚNG ĐỐI TƯỢNG

SẮP XẾP CÁC ĐỐI TƯỢNG VÀO FILE

Phần dữ liệu của đối tượng có thể được lưu trữ bởi sử dụng các cấu trúc file được mô tả trước đây với một số thay đổi do đối tượng có kích cỡ không đều, hơn nữa đối tượng có thể rất lớn. Ta có thể thực thi các trường tập hợp ít phần tử bằng cách sử dụng danh sách liên kết, các trường tập hợp nhiều phần tử bởi B-cây hoặc bởi các quan hệ riêng biệt trong cơ sở dữ liệu. Các trường tập hợp cũng có thể bị loại trừ ở mức lưu trữ bởi chuẩn hoá. Các đối tượng cực lớn khó có

thể phân tích thành các thành phần nhỏ hơn có thể được lưu trữ trong một file riêng cho mỗi đối tượng.

THỰC THI ĐỊNH DANH ĐỐI TƯỢNG

Vì đối tượng được nhận biết bởi định danh của đối tượng (OID = objject Identifier), Một hệ lưu trữ đối tượng cần phải có một cơ chế để tìm kiếm một đối tượng được cho bởi một OID. Nếu các OID là logic, có nghĩa là chúng không xác định vị trí của dối tượng, hệ thống lưu trữ phải duy trì một chỉ mục mà nó ánh xạ OID tới vị trí hiện hành của đối tượng. Nếu các OID là vật lý, có nghĩa là chúng mã hoá vị trí của đối tượng, đối tượng có thể dược tìm trực tiếp. Các OID điển hình có ba trường sau:

1. Một volume hoặc định danh file
2. Một định danh trang bên trong volume hoặc file
3. Một offset bên trong trang

Hơn nữa, OID vật lý có thẻ chứa một định danh duy nhất, nó là một số nguyên tách biệt OID với các định danh của các đối tượng khác đã được lưu trữ ở cùng vị trí trước đây và đã bị xoá hoặc dời đi. Định danh duy nhất này cũng được lưu với đối tượng, các định danh trong một OID và đối tượng tương ứng phù hợp. Nếu định danh duy nhất trong một OID vật lý không khớp với với định danh duy nhất trong đối tượng mà OID này trỏ tới, hệ thống phát hiện ra rằng con trỏ là bám và báo một lỗi. Lỗi con trỏ như vậy xảy ra khi OID vật lý tương ứng với đối tượng cũ đã bị xoá do tai nạn. Nếu không gian bị chiếm bởi đối tượng được cấp phát lại, có thể có một đối tượng mới ở vào vị trí này và có thể được định địa chỉ không đúng bởi định danh của đối tượng cũ. Nếu không phát hiện được, sử dụng con trỏ bám có thể gây nên sự sai lạc của một đối tượng mới được lưu ở cùng vị trí. Định danh duy nhất trợ giúp phát hiện lỗi như vậy. Giả sử một đối tượng phải di chuyển sang trang mới do sự lớn lên của đối tượng và trang cũ không có không gian phụ. Khi đó OID vật lý trỏ tới trang cũ bây giờ không còn chứa đối tượng. Thay vì thay đổi OID của đối tượng (điều này kéo theo sự thay đổi mỗi đối tượng trỏ tới đối tượng này) ta để địa chỉ forward ở vị trí cũ. Khi CSDL tìm đối tượng, nó tìm địa chỉ forward thay cho tìm đối tượng và sử dụng địa chỉ forward để tìm đối tượng.

QUẢN TRỊ CÁC CON TRỎ BỀN (persistent pointers)

Ta thực thi các con trỏ bền trong ngôn ngữ lập trình bền (persistent programming language) bằng cách sử dụng các OID. Các con trỏ bền có thể là các OID vật lý hoặc logic. Sự khác nhau quan trọng giữa con trỏ bền và con trỏ trong bộ nhớ là kích thước concủa con trỏ. Con trỏ trong bộ nhớ chỉ cần đủ lớn để định địa chỉ toàn bộ bộ nhớ ảo, hiện tại kích cỡ con trỏ trong bộ nhớ là 4 byte. Con trỏ bền để định địa chỉ toàn bộ dữ liệu trong một CSDL, nên kích cỡ của nó ít nhất là 8 byte.

Pointer Swizzling

Hành động tìm một đối tượng được cho bởi định danh được gọi là *dereferencing*. Đã cho một con trỏ trong bộ nhớ, tìm đối tượng đơn thuần là một sự tham khảo bộ nhớ. Đã cho một con trỏ bền, dereferencing một đối tượng có một bước phụ: phải tìm vị trí hiện hành của đối tượng trong bộ nhớ bởi tìm con trỏ bền trong một bảng. Nếu đối tượng chưa nằm trong bộ nhớ, nó phải được nạp từ đĩa. Ta có thể thực thi bảng tìm kiếm này hoàn toàn hiệu quả bởi sử dụng băm, song tìm kiếm vẫn chậm.

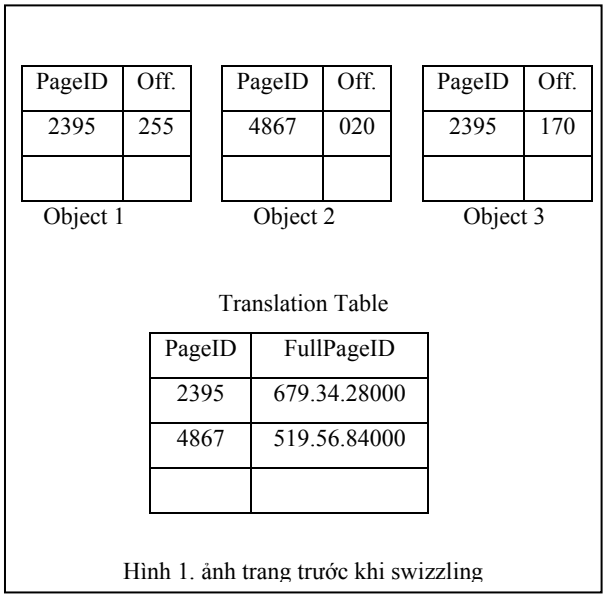
pointer swizling là một phương pháp để giảm cái giá tìm kiếm các đối tượng bền đã hiện diện trong bộ nhớ. ý tưởng là khi một con trỏ bền được dereference, đối tượng được định vị và mang vào trong bộ (nhớ nếu nó chưa có ở đó). Bây giờ một bước phụ được thực hiện: một con trỏ trong bộ nhớ tới đối tượng được lưu vào vị trí của con trỏ bền. Lần kế con trỏ bền tương tự được dereference, vị trí trong bộ nhớ có thể được đọc ra trực tiếp. Trong trường hợp các đối tượng bền phải di chuyển lên đĩa để lấy không gian cho đối tượng bền khác, cần một bước phụ để đảm bảo đối tượng vẫn trong bộ nhớ cũng phải được thực hiện. Khi một đối tượng được viết ra. bất kỳ con trỏ bền nào mà nó chứa và bị swizzling phải được unswizzling như vậy được chuyển đổi về biểu diễn bền của chúng. pointer swizzling

trên poiter dereferenc được mô tả này được gọi là software swizzling. Quan trị buffer sẽ phức tạp hơn nếu pointer swizzling được sử dụng.

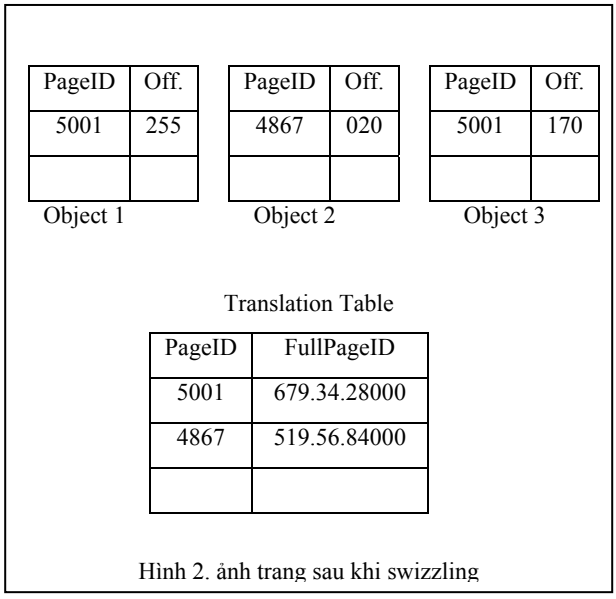
Hardware swizzling

Việc có hai kiểu con trỏ, con trỏ bền (persistent pointer) và con trỏ tạm (transient pointer / con trỏ trong bộ nhớ), là điều khá bất lợi. Người lập trình phải nhớ kiểu con trỏ và có thể phải viết mã chương trình hai lần- một cho các con trỏ bền và một cho con trỏ tạm. Sẽ thuận tiên hơn nếu cả hai kiểu con trỏ này cùng kiểu. Một cách đơn giản để trộn lãn hai con rỏ này là mở rộng chiều dài con trỏ bộ nhớ cho bằng kích cỡ con trỏ bền và sử dụng một bit của phần định danh để phân biệt chúng. Cách làm này sẽ làm tăng chi phí lưu trữ đối với các con trỏ tạm. Ta sẽ mô tả một kỹ thuật được gọi là hardware swizzling nó sử dụng phần cứng quản trị bộ nhớ để giải quyết vấn đề này. Hardware swizzling có hai điểm lợi hơn so với software swizzling: Thứ nhất, nó cho phép lưu trữ các con trỏ bền trong đối tượng trong lượng không gian bằng với lượng không gian con trỏ bộ nhớ đòi hỏi. Thứ hai, nó chuyển đổi trong suốt giữa các con trỏ bền và các con trỏ tạm một cách thông minh và hiệu quả. Phần mềm được viết để giải quyết các con trỏ trong bộ nhớ có thể giải quyết các con trỏ bền mà không cần thay đổi.

hardware swizzling sử dụng sự biểu diễn các con trỏ bền được chứa trong đối tượng trên đĩa như sau: Một con trỏ bền được tách ra thành hai phần, một là định danh trang và một là offset bên trong trang. Định danh trang thường là một con trỏ trực tiếp nhỏ: mỗi trưng có một bảng dịch (translation table) cung cấp một ánh xạ từ các định danh trang ngắn đến các định danh CSDL đầy đủ. Hệ thống phải tìm định danh trang nhỏ trong một con trỏ bền trong bảng dịch để tìm định danh trang đầy đủ. Bảng dịch, trong trường hợp xấu nhất, chỉ lớn bằng số tối đa các con trỏ có thể được chứa trong các đối tượng trong một trang. Với một trang kích thước 4096 byte, con trỏ kích thước 4 byte, số tối đa các con trỏ là 1024. Trong thực tế số tối đa nhỏ hơn con số này rất nhiều. Định danh trang nhỏ chỉ cần đủ số bit để định danh một dòng trong bảng, nếu số dòng tối đa là 1024, chỉ cần 10 bit để định danh trang nhỏ. Bảng dịch cho phép toàn bộ một con trỏ bền lấp đầy một không gian bằng không gian cho một con trỏ trong bộ nhớ.



Hình 1. ảnh trang trước khi swizzling



Hình 2. ảnh trang sau khi swizzling

Trong hình 1, trình bày sơ đồ biểu diễn con trỏ bền, có ba đối tượng trong trang, mỗi một chứa một con trỏ bền. Bảng dịch cho ra ánh xạ giữa định danh trang ngắn và định danh trang CSDL đầy đủ đối với mỗi định danh trang ngắn trong các con trỏ bền này. Định danh trang CSDL được trình bày dưới dạng ***volume.file.offset***. Thông tin phụ được duy trì với mỗi trang sao cho tất cảc các con trỏ bền trong trang có thể tìm thấy. Thông tin được cập nhất khi một đối tượng được tạo ra hay bị xoá khỏi trang. Khi một con trỏ trong bộ nhớ được dereferencing, nếu hệ điều hành phát hiện trang trong không gian địa chỉ ảo được trỏ tới không được cấp phát lưu trữ hoặc có truy xuất được bảo vệ, khi đó một sự vi phạm đoạn được ước đoán là xảy ra. Nhiều hệ điều hành cung cấp một cơ chế xác định một hàm sure được gọi khi vi phạm đoạn xảy ra, một cơ chế cấp phát lưu trữ cho các trang trong không gian địa chỉ ảo, và một tập các quyền truy xuất trang. Đầu tiên, ta xét một con trỏ trong bộ nhớ trỏ tới trang v được khử tham chiếu, khi lưu trữ chưa được cấp phát cho trang này. Một vi phạm đoạn sẽ xảy ra và kết quả là một lời gọi hàm trên hệ CSDL. Hệ CSDL dầu tiên xác định trang CSDL nào đã được cấp phát cho trang bộ nhớ ảo v, gọi định danh trang đầy đủ của trang CSDL là P, nếu không có trang CSDL cấp phát cho v, một lỗi được thông báo., nếu không, hệ CSDL cấp phát không gian lưu trữ cho trang v và nạp trang CSDL P vào trong v. Pointer swizzling bây giờ được làm đối với trang P như sau: Hệ thông tìm tất cả các con trỏ bèn được chứa trong các đối tương trong trang, bằng cách sử dụng thông tin phụ được lưu trữ trong trang. Ta xét một con trỏ như vậy và gọi nó là (p_i, o_i), trong đó p_i là định danh trang ngắn và

o_i là offset trong trang. Giả sử P_i là định danh trang đầy đủ của p_i được tìm thấy trong bảng dịch trong trang P. Nếu trang P_i chưa có một trang bộ nhớ ảo được cấp cho nó, một trang tự do trong không gian địa chỉ ảo sẽ được cấp cho nó. Trang P_i sẽ nằm ở vị trí địa chỉ ảo nàynếu và khi nó được mang vào. Tại điểm này, trang trong không gian địa chỉ ảo không có bất kỳ một lưu trữ nào được cấp cho nó, cả trong bộ nhớ lẫn trên đĩa, đơn tuần chỉ là một khoảng địa chỉ dự trữ cho trang CSDL. Bây giờ giả sử trang bộ nhớ ảo đã được cấp phát cho P_i là v_i . Ta cập nhật con trỏ (p_i, o_i) bởi thay thế p_i bởi v_i , cuối cùng sau khi swizzling tất cả các con trỏ bền trong P, sự khử tham chiếu gây ra vi phạm đoạn được cho phép tiếp tục và sẽ tìm thấy đối tượng đang được tìm kiếm trong bộ nhớ.

Trong hình 2, trình bày trạng thái trang trong hình 1 sau khi trang này được mang vào trong bộ nhớ và các con trỏ trong nó đã được swizzling. ở đây ta giả thiết trang định danh trang CSDL của nó là 679.34.28000 được ánh xạ đến trang 5001 trong bộ nhớ, trong khi trang định danh của nó là 519.56.84000 được ánh xạ đến trang 4867. Tất cả các con trỏ trong đối tượng đã được cập nhật để phản ánh tương ứng mới và bây giờ có thể được dùng như con trỏ trong bộ nhớ. ở cuối của giai đoạn dịch đối với một trang, các đối tượng trong trang thoả mãn một tính chất quan trọng: Tất cả các con trỏ bền được chứa trong đối tượng trong trang được chuyển đổi thành các con trỏ trong bộ nhớ.

CÂU HỎI VÀ BÀI TẬP CHƯƠNG III

III.1 Xét sự sắp xếp các khối dữ liệu và các khối parity trên bốn đĩa sau:

Đĩa 1	Đĩa 2	Đĩa 3	Đĩa 4
B_1	B_2	B_3	B_4
P_1	B_5	B_6	B_7
B_8	P_2	B_9	B_{10}
⋮	⋮	⋮	⋮

Trong đó các B_i biểu diễn các khối dữ liệu, các khối P_i biểu diễn các khối parity. Khối P_i là khối parity đối với các khối dữ liệu B_{4i-3} , B_{4i-2} , B_{4i-1} , B_{4i} . Hãy nêu các vấn đề gặp phải của cách sắp xếp này.

III.2 Một sự mất điện xảy ra trong khi một khối đang được viết sẽ dẫn tới kết quả là khối đó có thể chỉ được viết một phần. Giả sử rằng khối được viết một phần có thể phát hiện được. Một viết khối nguyên tử là hoặc toàn bộ khối được viết hoặc không có gì được viết (không có khối được viết một phần). Hãy đề nghị những sơ đồ để có được các viết khối nguyên tử hiệu quả trên các sơ đồ RAID:

1. Mức 1 (mirroring)
2. Mức 5 (block interleaved, distributed parity)

III.3 Các hệ thống RAID tiêu biểu cho phép thay thế các đĩa hư không cần ngưng truy xuất hệ thống. Như vậy dữ liệu trong đĩa bị hư sẽ phải được tái tạo và viết lên đĩa thay thế trong khi hệ thống vẫn tiếp tục hoạt động. Với mức RAID nào thời lượng giao thoa giữa việc tái tạo và các truy xuất đĩa còn đang chạy là ít nhất ? Giải thích.

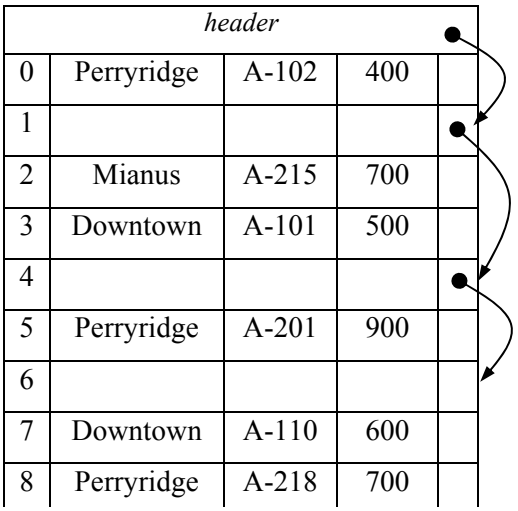
III.4 Xét việc xoá mẩu tin 5 trong file:

0	**Perryridge**	A-102	400
1	Round Hill	A-305	350
8	Perryridge	A-218	700
3	Downtown	A-101	500
4	Redwood	A-222	700
5	Perryridge	A-201	900
6	Brighton	A-217	750
7	Downtown	A-110	600

So sánh các điều hay/dở tương đối của các kỹ thuật xoá sau:

1. Di chuyển mẩu tin 6 đến không gian chị chiếm bởi mẩu tin 5, rồi di chuyển mẩu tin 7 đến chỗ bị chiếm bởi mẩu tin 6.
2. Di chuyển mẩu tin 7 đến chỗ bị chiếm bởi mẩu tin 5
3. Đánh dấu xoá mẩu tin 5.

III.5 Vẽ cấu trúc của file:



Sau mỗi bước sau:

1. Insert(Brighton, A-323, 1600)
2. Xoá mẩu tin 2
3. Insert(Brighton, A-636, 2500)

III.6 Vẽ lại cấu trúc file:

0	Perryridge	A-102	400	A-201	900	A210	700	⊥
1	Round Hill	A-301	350	⊥				
2	Mianus	A-101	800	⊥				
3	Downtown	A-211	500	A-222	600	⊥		
4	Redwood	A-300	650	A-200	1200	A-255	950	⊥
5	Brighton	A-111	750	⊥				

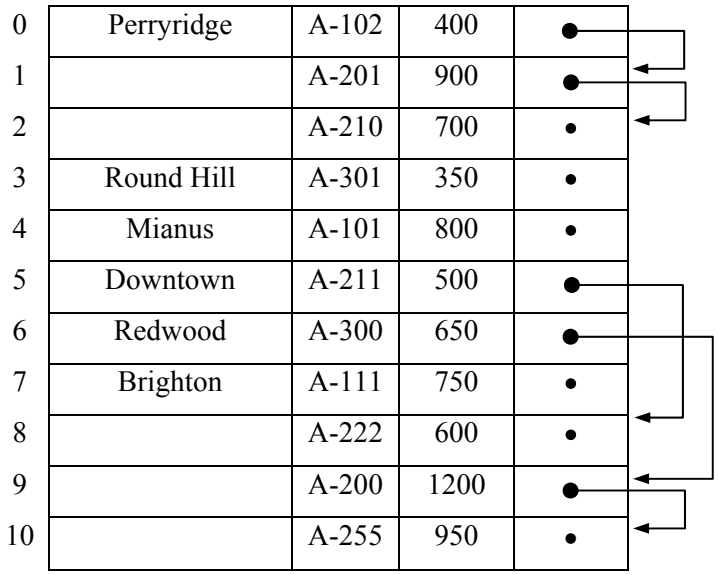
Sau mỗi bước sau:

1. Insert(Mianus, A-101, 2800)
2. Insert(Brighton, A-323, 1600)
3. Delete (Perryridge, A-102, 400)

III.7 Điều gì sẽ xảy ra nếu xen mẩu tin (Perryridge, A-999, 5000) vào file trong **III.6.**

III.8 Vẽ lại cấu trúc file dưới đây sau mỗi bước sau:

1. Insert(Mianus, A-101, 2800
2. Insert(Brighton, A-323, 1600)
3. Delete (Perryridge, A-102, 400)



(• = con trỏ nil)

III.9 Nêu lên một ví dụ, trong đó phương pháp không gian dự trữ để biểu diễn các mẩu tin độ dài thay đổi phù hợp hơn phương pháp con trỏ.

III.10 Nêu lên một ví dụ, trong đó phương pháp con trỏ để biểu diễn các mẩu tin độ dài thay đổi phù hợp hơn phương pháp không gian dự trữ.

III.11 Nếu một khối trở nên rỗng sau khi xoá. Khối này được tái sử dụng vào mục đích gì ?

III.12 Trong tổ chức file tuần tự, tại sao khối tràn được sử dụng thậm chí, tại thời điểm đang xét, chỉ có một mẩu tin tràn ?

III.13 Liệt kê các ưu điểm và nhược điểm của mỗi một trong các chiến lược lưu trữ CSDL quan hệ sau:

1. Lưu trữ mỗi quan hệ trong một file
2. Lưu trữ nhiều quan hệ trong một file

III.14 Nêu một ví dụ biểu thức đại số quan hệ và một chiến lược xử lý vấn tin trong đó:

1. MRU phù hợp hơn LRU
2. LRU phù hợp hơn MRU

III.15 Khi nào sử dụng chỉ mục đặc phù hợp hơn chỉ mục thưa ? Giải thích.

III.16 Nêu các điểm khác nhau giữa chỉ mục sơ cấp và chỉ mục thứ cấp .

III.17 Có thể có hai chỉ mục sơ cấp đối với hai khoá khác nhau trên cùng một quan hệ ? Giải thích.

III.18 Xây dựng một B^+-cây đối với tập các giá trị khoá: (2, 3, 5, 7, 11, 15, 19, 25, 29, 33, 37, 41, 47). Giả thiết ban đầu cây là rỗng và các giá trị được xen theo thứ tự tăng. Xét trong các trường hợp sau:

1. Mỗi nút chứa tối đa 4 con trỏ
2. Mỗi nút chứa tối đa 6 con trỏ
3. Mỗi nút chứa tối đa 8 con trỏ

III.19 Đối với mỗi B^+-cây trong bài tập **III.18** Bày tỏ các bước thực hiện trong các vấn tin sau:

1. Tìm mẩu tin với giá trị khoá tìm kiếm 11
2. Tìm các mẩu tin với giá trị khoá nằm trong khoảng [7..19]

III.20 Đối với mỗi B^+-cây trong bài tập **III.18.** Vẽ cây sau mỗi một trong dãy hoạt động sau:

1. Insert 9
2. Insert 11
3. Insert 11
4. Delete 25
5. Delete 19

III.21 Cùng câu hỏi như trong **III.18** nhưng đối với B-cây

III.22 Nêu và giải thích sự khác nhau giữa băm đóng và băm mở. Nêu các ưu, nhược điểm của mỗi kỹ thuật này.

III.23 Điều gì gây ra sự tràn bucket trong một tổ chức file băm ? Làm gì để giảm sự tràn này ?

III.24 Giả sử ta đang sử dụng băm có thể mở rộng trên một trên một file chứa các mẩu tin với các giá trị khoá tìm kiếm sau:

2, 3, 5, 7, 11, 17, 19, 23, 37, 31, 35, 41, 49, 55

Vẽ cấu trúc băm có thể mở rộng đối với file này nếu hàm băm là **h(x) = x mod 8** và mỗi bucket có thể chứa nhiều nhất được ba mẩu tin.

III.25 Vẽ lại cấu trúc băm có thể mở rộng trong bài tập **III.24** sau mỗi bước sau:

1. Xoá 11
2. Xoá 55
3. Xen 1

4. Xen 15

CHƯƠNG IV

GIAO DỊCH
(Transaction)

MỤC ĐÍCH

Giới thiệu khái niệm giao dịch, các tính chất một giao dịch cần phải có để sự hoạt động của nó trong môi trường có "biến động" vẫn đảm bảo cho CSDL luôn ở trạng thái nhất quán.

Giới thiệu các khái niệm khả tuần tự, khả tuần tự xung đột, khả tuần tự view, khả phục hồi và cascadeless, các thuật toán kiểm thử tính khả tuần tự xung đột và khả tuần tự view

YÊU CẦU

Hiểu khái niệm giao dịch, các tính chất cần phải có của giao dịch và "ai" là người đảm bảo các tính chất đó

Hiểu các khái niệm về khả tuần tự, khả phục hồi và mối tương quan giữa chúng

Hiểu và vận dụng các thuật toán kiểm thử

KHÁI NIỆM

Một giao dịch là một đơn vị thực hiện chương trình truy xuất và có thể cập nhật nhiều hạng mục dữ liệu. Một giao dịch thường là kết quả của sự thực hiện một chương trình người dùng được viết trong một ngôn ngữ thao tác dữ liệu mức cao hoặc một ngôn ngữ lập trình (SQL, COBOL, PASCAL ...), và được phân cách bởi các câu lệnh (hoặc các lời gọi hàm) có dạng **begin transaction** và **end transaction**. Giao dịch bao gồm tất cả các hoạt động được thực hiện giữa begin và end transaction.

Để đảm bảo tính toàn vẹn của dữ liệu, ta yêu cầu hệ CSDL duy trì các tính chất sau của giao dịch:

- **Tính nguyên tử (Atomicity)**. Hoặc toàn bộ các hoạt động của giao dịch được phản ánh đúng đắn trong CSDL hoặc không có gì cả.
- **Tính nhất quán (consistency)**. Sự thực hiện của một giao dịch là cô lập (Không có giao dịch khác thực hiện đồng thời) để bảo tồn tính nhất quán của CSDL.
- **Tính cô lập (Isolation)**. Cho dù nhiều giao dịch có thể thực hiện đồng thời, hệ thống phải đảm bảo rằng đối với mỗi cặp giao dịch T_i, T_j , hoặc T_j kết thúc thực hiện trước khi T_i khởi động hoặc T_j bắt đầu sự thực hiện sau khi T_i kết thúc. Như vậy mỗi giao dịch không cần biết đến các giao dịch khác đang thực hiện đồng thời trong hệ thống.
- **Tính bền vững (Durability)**. Sau một giao dịch hoàn thành thành công, các thay đổi đã được tạo ra đối với CSDL vẫn còn ngay cả khi xảy ra sự cố hệ thống.

Các tính chất này thường được gọi là các tính chất ACID (Các chữ cái đầu của bốn tính chất). Ta xét một ví dụ: Một hệ thống nhà băng gồm một số tài khoản và một tập các giao dịch truy xuất và cập nhật các tài khoản. Tại thời điểm hiện tại, ta giả thiết rằng CSDL nằm trên đĩa, nhưng một vài phần của nó đang nằm tạm thời trong bộ nhớ. Các truy xuất CSDL được thực hiện bởi hai hoạt động sau:

- **READ(X)**. chuyển hạng mục dữ liệu X từ CSDL đến buffer của giao dịch thực hiện hoạt động **READ** này.
- **WRITE(X)**. chuyển hạng mục dữ liệu X từ buffer của giao dịch thực hiện **WRITE** đến CSDL.

Trong hệ CSDL thực, hoạt động **WRITE** không nhất thiết dẫn đến sự cập nhật trực tiếp dữ liệu trên đĩa; hoạt động **WRITE** có thể được lưu tạm thời trong bộ nhớ và được thực hiện trên đĩa muộn hơn. Trong ví dụ, ta giả thiết hoạt động **WRITE** cập nhật trực tiếp CSDL.

T_i là một giao dịch chuyển 50 từ tài khoản A sang tài khoản B. Giao dịch này có thể được xác định như sau:

T_i : READ(A);
A:=A - 50;
WRITE(A)
READ(B);
B:=B + 50;
WRITE(B);

figure IV- 1

Ta xem xét mỗi một trong các yêu cầu ACID

- **Tính nhất quán**: Đòi hỏi nhất quán ở đây là tổng của A và B là không thay đổi bởi sự thực hiện giao dịch. Nếu không có yêu cầu nhất quán, tiền có thể được tạo ra hay bị phá huỷ bởi giao dịch. Dễ dàng kiểm nghiệm rằng nếu CSDL nhất quán trước một thực hiện giao dịch, nó vẫn nhất quán sau khi thực hiện giao dịch. Đảm bảo tính nhất quán cho một giao dịch là trách nhiệm của người lập trình ứng dụng người đã viết ra giao dịch. Nhiệm vụ này có thể được làm cho dễ dàng bởi kiểm thử tự động các ràng buộc toàn vẹn.

- **Tính nguyên tử**: Giả sử rằng ngay trước khi thực hiện giao dịch T_i, giá trị của các tài khoản A và B tương ứng là 1000 và 2000. Giả sử rằng trong khi thực hiện giao dịch T_i, một sự cố xảy ra cản trở T_i hoàn tất thành công sự thực hiện của nó. Ta cũng giả sử rằng sự cố xảy ra sau khi hoạt động **WRITE**(A) đã được thực hiện, nhưng trước khi hoạt động **WRITE**(B) được thực hiện. Trong trường hợp này giá trị của tài khoản A và B là 950 và 2000. Ta đã phá huỷ 50$. Tổng A+B không còn được bảo tồn.

  Như vậy, kết quả của sự cố là trạng thái của hệ thống không còn phản ánh trạng thái của thế giới mà CSDL được giả thiết nắm giữ. Ta sẽ gọi trạng thái như vậy là trạng thái không nhất quán. Ta phải đảm bảo rằng tính bất nhất này không xuất hiện trong một hệ CSDL. Chú ý rằng, cho dù thế nào tại một vài thời điểm, hệ thống cũng phải ở trong trạng thái không nhất quán. Ngay cả khi giao dịch T_i , trong quá trình thực hiện cũng tồn tại thời điểm tại đó giá trị của tài khoản A là 950 và tài khoản B là 2000 – một trạng thái không nhất quán. Trạng thái này được thay thế bởi trạng thái nhất quán khi giao dịch đã hoàn tất. Như vậy, nếu giao dịch không bao giờ khởi động hoặc được đảm bảo sẽ hoàn tất, trạng thái không nhất quán sẽ không bao giờ xảy ra. Đó chính là lý do có yêu cầu về tính nguyên tử: Nếu tính chất nguyên tử được cung cấp, ***tất cả các hành động của giao dịch được phản ánh trong CSDL hoặc không có gì cả***. ý tưởng cơ sở để đảm bảo tính nguyên tử là như sau: hệ CSDL lưu vết (trên đĩa) các giá trị cũ của bất kỳ dữ liệu nào trên đó giao dịch đang thực hiện viết, nếu giao dịch không hoàn tất, giá trị cũ được khôi phục để đặt trạng thái của hệ thống trở lại trạng thái trước khi giao dịch diễn ra. Đảm bảo tính nguyên tử là trách nhiệm của hệ CSDL, và được quản lý bởi một thành phần được gọi là thành phần quản trị giao dịch (transaction-management component).

- **Tính bền vững**: Tính chất bền vững đảm bảo rằng mỗi khi một giao dịch hoàn tất, tất cả các cập nhật đã thực hiện trên cơ sở dữ liệu vẫn còn đó, ngay cả khi xảy ra sự cố hệ thống sau khi giao dịch đã hoàn tất. Ta giả sử một sự cố hệ thống có thể gây ra việc mất dữ liệu trong bộ nhớ chính, nhưng dữ liệu trên đĩa thì không mất. Có thể đảm bảo tính bền vững bởi việc đảm bảo hoặc ***các cập nhật được thực hiện bởi giao dịch đã được viết lên đĩa trước khi giao dịch kết thúc*** hoặc ***thông tin về sự cập nhật được thực hiện bởi giao dịch và được viết lên đĩa đủ cho phép CSDL xây dựng lại các cập nhật khi hệ CSDL được khởi động lại sau sự cố***. Đảm bảo tính bền vững là trách nhiệm của một thành phần của hệ CSDL được gọi là thành phần quản trị phục hồi (recovery-management component). Hai thành phần quản trị giao dịch và quản trị phục hồi quan hệ mật thiết với nhau.

- **Tính cô lập:** Ngay cả khi tính nhất quán và tính nguyên tử được đảm bảo cho mỗi giao dịch, trạng thái không nhất quán vẫn có thể xảy ra nếu trong hệ thống có một số giao dịch được thực hiện đồng thời và các hoạt động của chúng đan xen theo một cách không mong muốn. Ví dụ, CSDL là không nhất quán tạm thời trong khi giao dịch chuyển khoản từ A sang B đang thực hiện, nếu một giao dịch khác thực hiện đồng thời đọc A và B tại

thời điểm trung gian nàyvà tính A+B, nó đã tham khảo một giá trị không nhất quán, sau đó nó thực hiện cập nhật A và B dựa trên các giá trị không nhất quán này, như vậy CSDL có thể ở trạng thái không nhất quán ngay cả khi cả hai giao dịch hoàn tất thành công. Một giải pháp cho vấn đề các giao dịch thực hiện đồng thời là ***thực hiện tuần tự các giao dịch***, tuy nhiên giải pháp này làm giả hiệu năng của hệ thống. Các giải pháp khác cho phép nhiều giao dịch thực hiện cạnh tranh đã được phát triển ta sẽ thảo luận về chúng sau này. Tính cô lập của một giao dịch đảm bảo rằng sự thực hiện đồng thời các giao dịch dẫn đến một trạng thái hệ thống tương đương với một trạng thái có thể nhận được bởi thực hiện các giao dịch này một tại một thời điểm theo một thứ nào đó. Đảm bảo tính cô lập là trách nhiệm của một thành phần của hệ CSDL được gọi là thành phần quản trị cạnh tranh (concurrency-control component).

TRẠNG THÁI GIAO DỊCH

Nếu không có sự cố, tất cả các giao dịch đều hoàn tất thành công. Tuy nhiên, một giao dịch trong thực tế có thể không thể hoàn tất sự thực hiện của nó. Giao dịch như vậy được gọi là bị bỏ dở. Nếu ta đảm bảo được tính nguyên tử, một giao dịch bị bỏ dở không được phép làm ảnh hưởng tới trạng thái của CSDL. Như vậy, bất kỳ thay đổi nào mà giao dịch bị bỏ dở này phải bị huỷ bỏ. Mỗi khi các thay đổi do giao dịch bị bỏ dở bị huỷ bỏ, ta nói rằng giao dịch bị cuộn lại (rolled back). Việc này là trách nhiệm của sơ đồ khôi phục nhằm quản trị các giao dịch bị bỏ dở. Một giao dịch hoàn tất thành công sự thực hiện của nó được gọi là được bàn giao (committed). Một giao dịch được bàn giao (committed), thực hiện các cập nhật sẽ biến đổi CSDL sang một trạng thái nhất quán mới và nó là bền vững ngay cả khi có sự cố. Mỗi khi một giao dịch là được bàn giao (committed), ta không thể huỷ bỏ các hiệu quả của nó bằng các bỏ dở nó. Cách duy nhất để huỷ bỏ các hiệu quả của một giao dịch được bàn giao (committed) là thực hiện một giao dịch bù (compensating transaction); nhưng không phải luôn luôn có thể tạo ra một giao dịch bù. Do vậy trách nhiệm viết và thực hiện một giao dịch bù thuộc về người sử dụng và không được quản lý bởi hệ CSDL.

Một giao dịch phải ở trong một trong các trạng thái sau:

- **Hoạt động (Active)**. Trạng thái khởi đầu; giao dịch giữ trong trạng thái này trong khi nó đang thực hiện.
- **được bàn giao bộ phận (Partially Committed).** Sau khi lệnh cuối cùng được thực hiện.
- **Thất bại (Failed).** Sau khi phát hiện rằng sự thực hiện không thể tiếp tục được nữa.
- **Bỏ dở (Aborted).** Sau khi giao dịch đã bị cuộn lại và CSDL đã phục hồi lại trạng thái của nó trước khi khởi động giao dịch.
- **được bàn giao (Committed).** Sau khi hoàn thành thành công giao dịch.

Ta nói một giao dịch đã được bàn giao (committed) chỉ nếu nó đã di vào trạng thái Committed, tương tự, một giao dịch bị bỏ dở nếu nó đã đi vào trạng thái Aborted. Một giao dịch được gọi là kết thúc nếu nó hoặc là committed hoặc là Aborted. Một giao dịch khởi đầu bởi trạng thái Active. Khi nó kết thúc lệnh sau cùng của nó, nó chuyển sang trạng thái partially committed. Tại thời điểm này, giao dịch đã hoàn thành sự thực hiện của nó, nhưng nó vẫn có thể bị bỏ dở do đầu ra hiện tại vẫn có thể trú tạm thời trong bộ nhớ chính và như thế một sự cố phần cứng vẫn có thể ngăn cản sự hoàn tất của giao dịch. Hệ CSDL khi đó đã kịp viết lên đĩa đầy đủ thông tin giúp việc tái tạo các cập nhật đã được thực hiện trong quá trình thực hiện giao dịch, khi hệ thống tái

khởi động sau sự cố. Sau hhi các thông tin sau cùng này được viết lên đĩa, giao dịch chuyển sang trạng thái committed.

Biểu đồ trạng thái tương ứng với một giao dịch như sau:

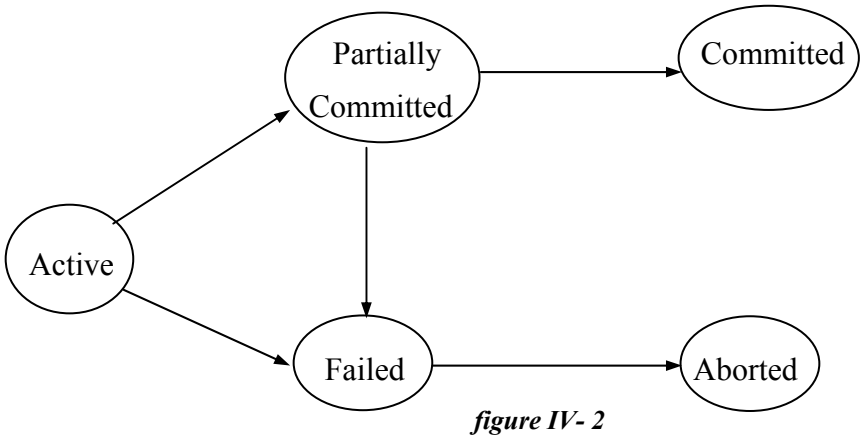


figure IV- 2

Với giả thiết sự cố hệ thống không gây ra sự mất dữ liệu trên đĩa, Một giao dịch đi vào trạng thái Failed sau khi hệ thống xác định rằng giao dịch không thể tiến triển bình thường được nữa (do lỗi phần cứng hoặc phần mềm). Như vậy, giao dịch phải được cuộn lại rồi chuyển sang trạng thái bỏ dở. Tại điểm này, hệ thống có hai lựa chọn:

- ***Khởi động lại giao dịch***, nhưng chỉ nếu giao dịch bị bở dở là do lỗi phần cứng hoặc phần mềm nào đó không liên quan đến logic bên trong của giao dịch. Giao dịch được khởi động lại được xem là một giao dịch mới.
- ***Giết giao dịch*** thường được tiến hành hoặc do lỗi logic bên trong giao dịch, lỗi này cần được chỉnh sửa bởi viết lại chương trình ứng dụng hoặc do đầu vào xấu hoặc do dữ liệu mong muốn không tìm thấy trong CSDL.

Ta phải thận trọng khi thực hiện viết ngoài khả quan sát (observable external **Write** - như viết ra terminal hay máy in). Mỗi khi một viết như vậy xẩy ra, nó không thể bị xoá do nó có thể phải giao tiếp với bên ngoài hệ CSDL. Hầu hết các hệ thống cho phép các viết như thế xẩy ra chỉ khi giao dịch đã di vào trạng thái committed. Một cách để thực thi một sơ đồ như vậy là cho hệ CSDL lưu trữ tạm thời bất kỳ giá trị nào kết hợp với các viết ngoài như vậy trong lưu trữ không hay thay đổi và thực hiện các viết hiện tại chỉ sau khi giao dịch đã đi vào trạng thái committed. Nếu hệ thống thất bại sau khi giao dịch đi vào trạng thái committed nhưng trước khi hoàn tất các viết ngoài, hệ CSDL sẽ làm các viết ngoài này (sử dụng dữ liệu trong lưu trữ không hay thay đổi) khi hệ thống khởi động lại.

Trong một số ứng dụng, có thể muốn cho phép giao dịch hoạt động trình bày dữ liệu cho người sử dụng, đặc biệt là các giao dịch kéo dài trong vài phút hay vài giờ. Ta không thể cho phép xuất ra dữ liệu khả quan sát như vậy trừ phi ta buộc phải làm tổn hại tính nguyên tử giao dịch. Hầu hết các hệ thống giao dịch hiện hành đảm bảo tính nguyên tử và do vậy cấm dạng trao đổi với người dùng này.

THỰC THI TÍNH NGUYÊN TỬ VÀ TÍNH BỀN VỮNG

Thành phần quản trị phục hồi của một hệ CSDL hỗ trợ tính nguyên tử và tính bền vững. Trước tiên ta xét một sơ đồ đơn giản (song cực kỳ thiếu hiệu quả). Sơ đồ này giả thiết rằng chỉ một giao dịch là hoạt động tại một thời điểm và được dựa trên tạo bản sao của CSDL được gọi là

các bản sao bóng (shadow copies). Sơ đồ giả thiết rằng CSDL chỉ là một file trên đĩa. Một con trỏ được gọi là db_pointer được duy trì trên đĩa; nó trỏ tới bản sao hiện hành của CSDL.

Trong sơ đồ CSDL bóng (shadow-database), một giao dịch muốn cập nhật CSDL, đầu tiên tạo ra một bản sao đầy đủ của CSDL. Tất cả các cập nhật được làm trên bản sao này, không đụng chạm tới bản gốc (bản sao bóng). Nếu tại một thời điểm bất kỳ giao dịch bị bỏ dở, bản sao mới bị xoá. Bản sao cũ của CSDL không bị ảnh hưởng. Nếu giao dịch hoàn tất, nó được được bàn giao (committed) như sau. Đầu tiên, Hỏi hệ điều hành để đảm bảo rằng tất cả các trang của bản sao mới đã được viết lên đĩa (flush). Sau khi flush con trỏ db_pointer được cập nhật để trỏ đến bản sao mới; bản sao mới trở thành bản sao hiện hành của CSDL. Bản sao cũ bị xoá đi. Giao dịch được gọi là đã được được bàn giao (committed) tại thời điểm sự cập nhật con trỏ db_pointer được ghi lên đĩa. Ta xét kỹ thuật này quản lý sự cố giao dịch và sự cố hệ thống ra sao? Trước tiên, ta xét sự cố giao dịch. Nếu giao dịch thất bại tại thời điểm bất kỳ trước khi con trỏ db_pointer được cập nhật, nội dung cũ của CSDL không bị ảnh hưởng. Ta có thể bỏ dở giao dịch bởi xoá bản sao mới. Mỗi khi giao dịch được được bàn giao (committed), tất cả các cập nhật mà nó đã thực hiện là ở trong CSDL được trỏ bởi db_pointer. Như vậy, hoặc tất cả các cập nhật của giao dịch đã được phản ánh hoặc không hiệu quả nào được phản ánh, bất chấp tới sự cố giao dịch. Bây giờ ta xét sự cố hệ thống. Giả sử sự cố hệ thống xảy ra tại thời điểm bất kỳ trước khi db_pointer đã được cập nhật được viết lên đĩa. Khi đó, khi hệ thống khởi động lại, nó sẽ đọc db_pointer và như vậy sẽ thấy nội dung gốc của CSDL – không hiệu quả nào của giao dịch được nhìn thấy trên CSDL. Bây giờ lại giả sử rằng sự cố hệ thống xảy ra sau khi db_pointer đã được cập nhật lên đĩa. Trước khi con trỏ được cập nhật, tất cả các trang được cập nhật của bản sao mới đã được viết lên đĩa. Từ giả thiết file trên đĩa không bị hư hại do sự cố hệ thống. Do vậy, khi hệ thống khởi động lại, nó sẽ đọc db_pointer và sẽ thấy nội dung của CSDL sau tất cả các cập nhật đã thực hiện bởi giao dịch. Sự thực thi này phụ thuộc vào việc viết lên db_pointer, việc viết này phải là nguyên tử, có nghĩa là hoặc tất cả các byte của nó được viết hoặc không byte nào được viết. Nếu chỉ một số byte của con trỏ được cập nhật bởi việc viết nhưng các byte khác thì không thì con trỏ trở thành vô nghĩa và cả bản cũ lẫn bản mới của CSDL có thể tìm thấy khi hệ thống khởi động lại. May mắn thay, hệ thống đĩa cung cấp các cập nhật nguyên tử toàn bộ khối đĩa hoặc ít nhất là một sector đĩa. Như vậy hệ thống đĩa đảm bảo việc cập nhật con trỏ db_pointer là nguyên tử. Tính nguyên tử và tính bền vững của giao dịch được đảm bảo bởi việc thực thi bản sao bóng của thành phần quản trị phục hồi. Sự thực thi này cực kỳ thiếu hiệu quả trong ngữ cảnh CSDL lớn, do sự thực hiện một giao dịch đòi hỏi phải sao toàn bộ CSDL. Hơn nữa sự thực thi này không cho phép các giao dịch thực hiện đồng thời với các giao dịch khác. Phương pháp thực thi tính nguyên tử và tính lâu bền mạnh hơn và đỡ tốn kém hơn được trình bày trong chương *hệ thống phục hồi*.

CÁC THỰC HIỆN CẠNH TRANH

Hệ thống xử lý giao dịch thường cho phép nhiều giao dịch thực hiện đồng thời. Việc cho phép nhiều giao dịch cập nhật dữ liệu đồng thời gây ra những khó khăn trong việc bảo đảm sự nhất quán dữ liệu. Bảo đảm sự nhất quán dữ liệu mà không đếm xỉa tới sự thực hiện cạnh tranh các giao dịch sẽ cần thêm các công việc phụ. Một phương pháp dễ tiến hành là cho các giao dịch thực hiện tuần tự: đảm bảo rằng một giao dịch khởi động chỉ sau khi giao dịch trước đã hoàn tất. Tuy nhiên có hai lý do hợp lý để thực hiện cạnh tranh là:

- Một giao dịch gồm nhiều bước. Một vài bước liên quan tới hoạt động I/O; các bước khác liên quan đến hoạt động CPU. CPU và các đĩa trong một hệ thống có thể hoạt động song song. Do vậy hoạt động I/O có thể được tiến hành song song với xử lý tại CPU. Sự song song của hệ thống CPU và I/O có thể được khai thác để chạy nhiều giao dịch song song. Trong khi một giao dịch tiến hành một hoạt động đọc/viết trên một đĩa, một giao dịch khác có thể đang chạy trong CPU, một giao dịch thứ ba có thể thực hiện đọc/viết

trên một đĩa khác ... như vậy sẽ tăng lượng đầu vào hệ thống có nghĩa là tăng số lượng giao dịch có thể được thực hiện trong một lượng thời gian đã cho, cũng có nghĩa là hiệu suất sử dụng bộ xử lý và đĩa tăng lên.

- Có thể có sự trộn lẫn các giao dịch đang chạy trong hệ thống, cái thì dài cái thì ngắn. Nếu thực hiện tuần tự, một quá trình ngắn có thể phải chờ một quá trình dài đến trước hoàn tất, mà điều đó dẫn đến một sự trì hoãn không lường trước được trong việc chạy một giao dịch. Nếu các giao dịch đang hoạt động trên các phần khác nhau của CSDL, sẽ tốt hơn nếu ta cho chúng chạy đồng thời, chia sẻ các chu kỳ CPU và truy xuất đĩa giữa chúng. Thực hiện cạnh tranh làm giảm sự trì hoãn không lường trước trong việc chạy các giao dịch, đồng thời làm giảm thời gian đáp ứng trung bình: *Thời gian để một giao dịch được hoàn tất sau khi đã được đệ trình*.

Động cơ để sử dụng thực hiện cạnh tranh trong CSDL cũng giống như động cơ để thực hiện đa chương trong hệ điều hành. Khi một vài giao dịch chạy đồng thời, tính nhất quán CSDL có thể bị phá huỷ cho dù mỗi giao dịch là đúng. Một giải pháp để giải quyết vấn đề này là sử dụng định thời. Hệ CSDL phải điều khiển sự trao đổi giữa các giao dịch cạnh tranh để ngăn ngừa chúng phá huỷ sự nhất quán của CSDL. Các cơ chế cho điều đó được gọi là sơ đồ điều khiển cạnh tranh (concurrency-control scheme).

Xét hệ thống nhà băng đơn giản, nó có một số tài khoản và có một tập hợp các giao dịch, chúng truy xuất, cập nhật các tài khoản này. Giả sử T_1 và T_2 là hai giao dịch chuyển khoản từ một tài khoản sang một tài khoản khác. Giao dịch T_1 chuyển 50$ từ tài khoản A sang tài khoản B và được xác định như sau:

T_1 : **Read**(A);
A:=A-50;
Write(A);
Read(B);
B:=B+50;
Write(B);

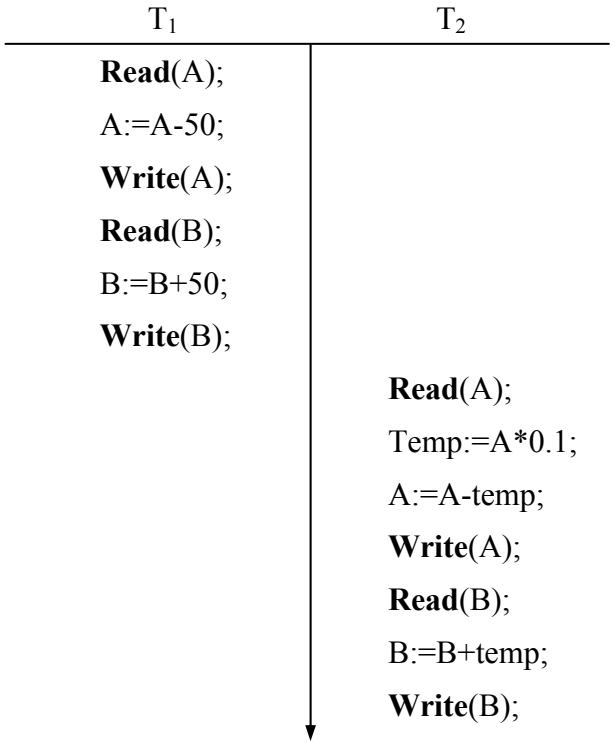
figure IV- 3

Giao dịch T_2 chuyển 10% số dư từ tài khoản A sang tài khoản B, và được xác định như sau:

T_2 : **Read**(A);
Temp:=A*0.1;
A:=A-temp;
Write(A);
Read(B);
B:=B+temp;
Write(B);

figure IV- 4

Giả sử giá trị hiện tại của A và B tương ứng là 1000$ và 2000$. Giả sử rằng hai giao dịch này được thực hiện mỗi một tại một thời điểm theo thứ tự T_1 rồi tới T_2. Như vậy, dãy thực hiện này là như hình bên dưới, trong đó dãy các bước chỉ thị ở trong thứ tự thời gian từ đỉnh xuống đáy, các chỉ thị của T_1 nằm ở cột trái còn các chỉ thị của T_2 nằm ở cột phải:

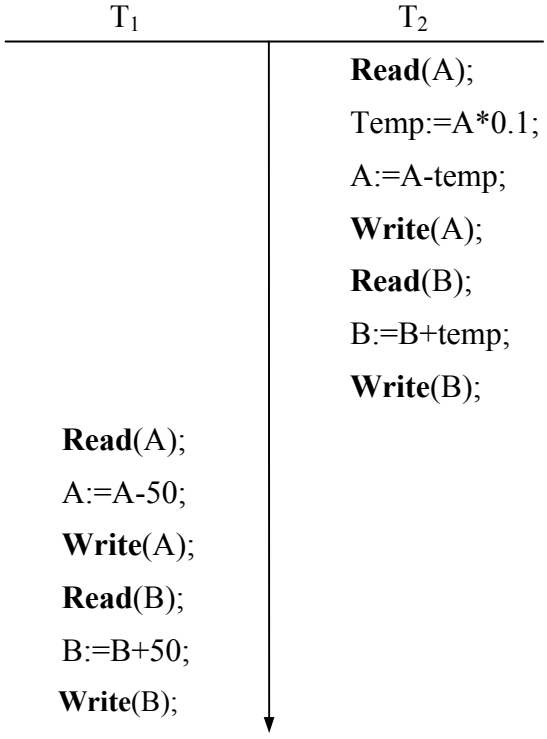


Schedule-1

figure IV- 5

Giá trị sau cùng của các tài khoản A và B, sau khi thực hiện dãy các chỉ thị theo trình tự này là 855$ và 2145$ tương ứng. Như vậy, tổng giá trị của hai tài khoản này (A + B) được bảo tồn sau khi thực hiện cả hai giao dịch.

Tương tự, nếu hai giao dịch được thực hiện mỗi một tại một thời điểm song theo trình tự T_2 rồi đến T_1 , khi đó dãy thực hiện sẽ là:



Schedule-2

figure IV- 6

Và kết quả là các giá trị cuối cùng của tài khoản A và B tương ứng sẽ là 850$ và 2150$.

Các dãy thực hiện vừa được mô tả trên được gọi là các lịch trình (schedules). Chúng biểu diễn trình tự thời gian các chỉ thị được thực hiện trong hệ thống. Một lịch trình đối với một tập các giao dịch phải bao gồm tất cả các chỉ thị của các giao dich này và phải bảo tồn thứ tự các chỉ thị xuất hiện trong mỗi một giao dịch. Ví dụ, đối với giao dịch T_1 , chỉ thị **Write**(A) phải xuất hiện trước chỉ thị **Read**(B), trong bất kỳ lịch trình hợp lệ nào. Các lịch trình schedule-1 và schedule-2 là tuần tự. Mỗi lịch trình tuần tự gồm một dãy các chỉ thị từ các giao dịch, trong đó các chỉ thị thuộc về một giao dịch xuất hiện cùng nhau trong lịch trình. Như vậy, đối với một tập n giao dịch, có n! lịch trình tuần tự hợp lệ khác nhau. Khi một số giao dịch được thực hiện đồng thời, lịc trình tương ứng không nhất thiết là tuần tự. Nếu hai giao dịch đang chạy đồng thời, hệ điều hành có thể thực hiện một giao dịch trong một khoảng ngắn thời gian, sau đó chuyển đổi ngữ cảnh, thực hiện giao dịch thứ hai một khoảng thời gian sau đó lại chuyển sang thực hiện giao dịch thứ nhất một khoảng và cứ như vậy (hệ thống chia sẻ thời gian).

Có thể có một vài dãy thực hiện, vì nhiều chỉ thị của các giao dịch có thể đan xen nhau. Nói chung, không thể dự đoán chính xác những chỉ thị nào của một giao dịch sẽ được thực hiện trước khi CPU chuyển cho giao dịch khác. Do vậy, số các lịch trình có thể đối với một tập n giao dịch lớn hơn n! nhiều.

T_1	T_2
Read(A);	
A:=A-50;	
Write(A);	
	Read(A);
	Temp:=A*0.1;
	A:=A-temp;
	Write(A);
Read(B);	
B:=B+50;	
Write(B);	
	Read(B);
	B:=B+temp;
	Write(B);

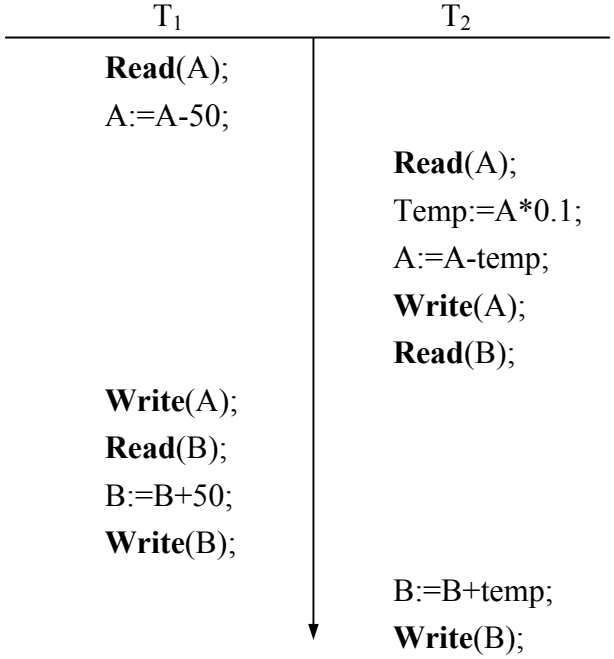
Schedule-3 --- một lịch trình cạnh tranh tương đương schedule-1

figure IV- 7

Không phải tất cả các thực hiện cạnh tranh cho ra một trạng thái đúng. Ví dụ schedule-4 sau cho ta một minh hoạ về nhận định này:

Sau khi thực hiện giao dịch này, ta đạt tới trạng thái trong đó giá trị cuối của A và B tương ứng là 950$ và 2100$. Trạng thái này là một trạng thái không nhất quán (A+B trước khi thực hiện giao dịch là 3000$ nhưng sau khi giao dịch là 3050$). Như vậy, nếu giao phó việc điều khiển thực hiện cạnh tranh cho hệ điều hành, sẽ có thể dẫn tới các trạng thái không nhất quán. Nhiệm vụ của

hệ CSDL là đảm bảo rằng một lịch trình được phép thực hiện sẽ đưa CSDL sang một trạng thái nhất quán. Thành phần của hệ CSDL thực hiện nhiệm vụ này được gọi là thành phần điều khiển cạnh tranh (concurrency-control component). Ta có thể đảm bảo sự nhất quán của CSDL với thực hiện cạnh tranh bằng cách nắm chắc rằng một lịch trình được thực hiện có cùng hiệu quả như một lịch trình tuần tự.

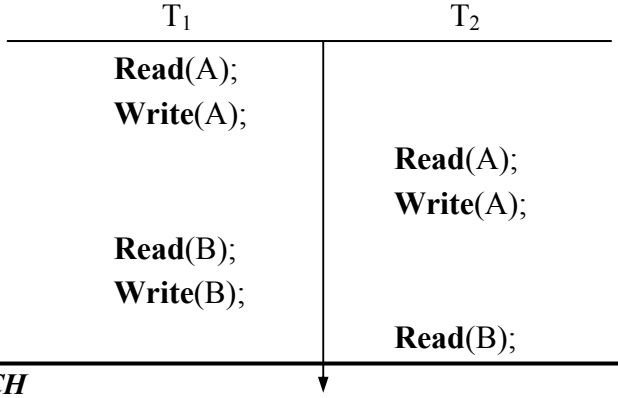


Schedule-4 --- một lịch trình cạnh tranh

figure IV- 8

TÍNH KHẢ TUẦN TỰ (Serializability)

Hệ CSDL phải điều khiển sự thực hiện cạnh tranh các giao dịch để đảm bảo rằng trạng thái CSDL giữ nguyên ở trạng thái nhất quán. Trước khi ta kiểm tra hệ CSDL có thể thực hiện nhiệm vụ này như thế nào, đầu tiên ta phải hiểu các lịch trình nào sẽ đảm bảo tính nhất quán và các lịch trình nào không. Vì các giao dịch là các chương trình, nên thật khó xác định các hoạt động chính xác được thực hiện bởi một giao dịch là hoạt động gì và những hoạt động nào của các giao dịch tác động lẫn nhau. Vì lý do này, ta sẽ không giải thích kiểu hoạt động mà một giao dịch có thể thực hiện trên một hạng mục dữ liệu. Thay vào đó, ta chỉ xét hai hoạt động: **Read** và **Write**. Ta cũng giả thiết rằng giữa một chỉ thị **Read(Q)** và một chỉ thị **Write(Q)** trên một hạng mục dữ liệu **Q,** một giao dịch có thể thực hiện một dãy tuỳ ý các hoạt động trên bản sao của Q được lưu trú trong buffer cục bộ của giao dịch. Vì vậy ta sẽ chỉ nêu các chỉ thị **Read** và **Write** trong lịch trình, như trong biểu diễn với quy ước như vậy của schedule-3 dưới đây:



Write(B);

Schedule-3 (viết dưới dạng thoả thuận)

figure IV- 9

TUẦN TỰ XUNG ĐỘT (Conflict Serializability)

Xét lịch trình S trong đó có hai chỉ thị liên tiếp I_i và I_j của các giao dịch T_i , T_j tương ứng ($i \neq j$). Nếu I_i và I_j tham khảo đến các hạng mục dữ liệu khác nhau, ta có thể đổi chỗ I_i và I_j mà không làm ảnh hưởng đến kết quả của bất kỳ chỉ thị nào trong lịch trình. Tuy nhiên, nếu I_i và I_j tham khảo cùng một hạng mục dữ liệu Q, khi đó thứ tự của hai bước này có thể rất quan trọng. Do ta đang thực hiện chỉ các chỉ thị **Read** và **Write**, nên ta có bốn trường hợp cần phải xét sau:

1. I_i = **Read**(Q); I_j = **Read**(Q): Thứ tự của I_i và I_j không gây ra vấn đề nào, do T_i và T_j đọc cùng một giá trị Q bất kể đến thứ tự giữa I_i và I_j.
2. I_i = **Read**(Q); I_j = **Write**(Q): Nếu I_i thực hiện trước I_j, Khi đó T_i không đọc giá trị được viết bởi T_j bởi chỉ thị I_j . Nếu I_j thực hiện trước I_i , T_i sẽ đọc giá trị của Q được viết bởi I_j , như vậy thứ tự của I_i và I_j là quan trọng.
3. I_i = **Write**(Q); I_j = **Read**(Q): Thứ tự của I_i và I_j là quan trọng do cùng lý do trong trường hợp trước.
4. I_i = **Write**(Q); I_j = **Write**(Q): Cả hai chỉ thị là hoạt động **Write**, thứ tự của hai chỉ thị này không ảnh hưởng đến cả hai giao dịch T_i và T_j . Tuy nhiên, giá trị nhận được bởi chỉ thị **Read** kế trong S sẽ bị ảnh hưởng do kết quả phụ thuộc vào chỉ thị **Write** được thực hiện sau cùng trong hai chỉ thị **Write** này. Nếu không còn chỉ thị **Write** nào sau I_i và I_j trong S, thứ tự của I_i vf I_j sẽ ảnh hưởng trực tiếp đến giá trị cuối của Q trong trạng thái CSDL kết quả (của lịch trình S).

Như vậy chỉ trong trường hợp cả I_i và I_j là các chỉ thị **Read**, thứ tự thực hiện của hai chỉ thị này (trong S) là không gây ra vấn đề.

Ta nói I_i và I_j xung đột nếu các hoạt động này nằm trong các giao dịch khác nhau, tiến hành trên cùng một hạng mục dữ liệu và có ít nhất một hoạt động là **Write**. Ta xét lịch trình schedule-3 như ví dụ minh hoạ cho các chỉ thị xung đột.

T_1	T_2
Read(A);	
Write(A);	
	Read(A);
	Write(A);
Read(B);	
Write(B);	
	Read(B);
	Write(B);

figure IV- 10

Chỉ thị **Write(A)** trong T_1 xung đột với **Read(A)** trong T_2. Tuy nhiên, chỉ thị **Write(A)** trong T_2 không xung đột với chỉ thị **Read(B)** trong T_1 do các chỉ thị này truy xuất các hạng mục dữ liệu khác nhau.

I_i và I_j là hai chỉ thị liên tiếp trong lịch trình S. Nếu I_i và I_j là các chỉ thị của các giao dịch khác nhau và không xung đột, khi đó ta có thể đổi thứ tự của chúng mà không làm ảnh hưởng gì đến kết quả xử lý và như vậy ta nhận được một lịch trình mới S' tương đương với S. Do chỉ thị **Write(A)** của T_2 không xung đột với chỉ thị **Read(B)** của T_1, ta có thể đổi chỗ các chỉ thị này để được một lịch trình tương đương – schedule-5 dưới đây

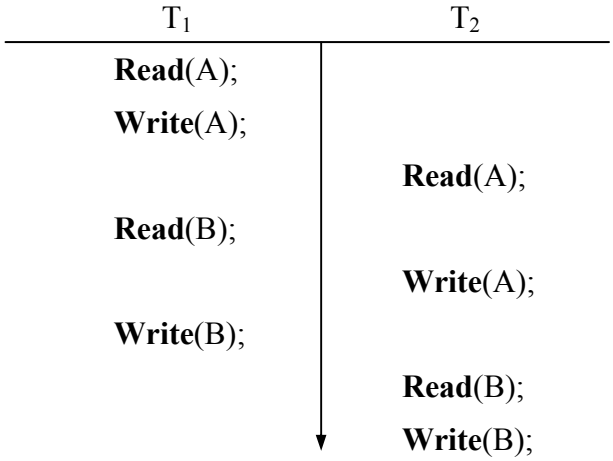
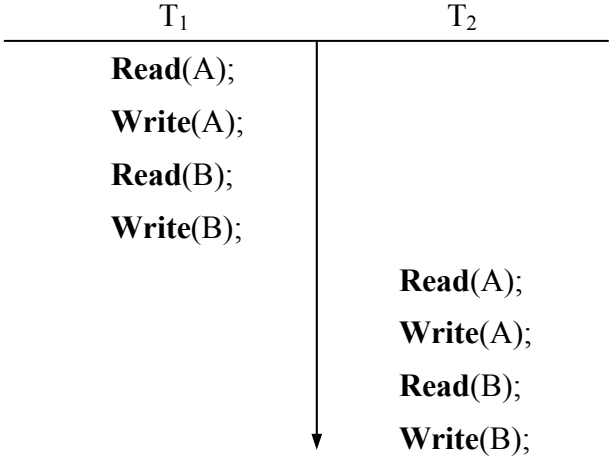


figure IV- 11

Ta tiếp tục đổi chỗ các chỉ thị không xung đột như sau:

- Đổi chỗ chỉ thị **Read(B)** của T_1 với chỉ thị **Read(A)** của T_2
- Đổi chỗ chỉ thị **Write(B)** của T_1 với chỉ thị **Write(A)** của T_2
- Đổi chỗ chỉ thị **Write(B)** của T_1 với chỉ thị **Read(A)** của T_2

Kết quả cuối cùng của các bước đổi chỗ này là một lịch trình mới (schedule-6 –lịch trình tuần tự) tương đương với lịch trình ban đầu (schedule-3):



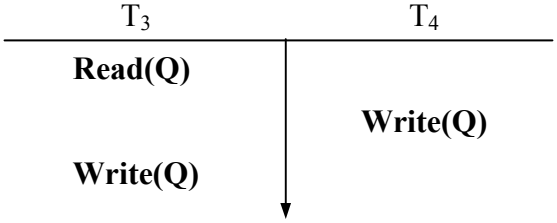
Schedule-6

figure IV- 12

Sự tương đương này cho ta thấy: bất chấp trạng thái hệ thống ban đầu, schedule-3 sẽ sinh ra cùng trạng thái cuối như một lịch trình tuần tự nào đó.

Nếu một lịch trình S có thể biến đổi thành một lịch trình S' bởi một dãy các đổi chỗ các chỉ thị không xung đột, ta nói S và S' là tương đương xung đột (*conflict equivalent*). Trong các schedule đã được nêu ở trên, ta thấy schedule-1 tương đương xung đột với schedule-3.

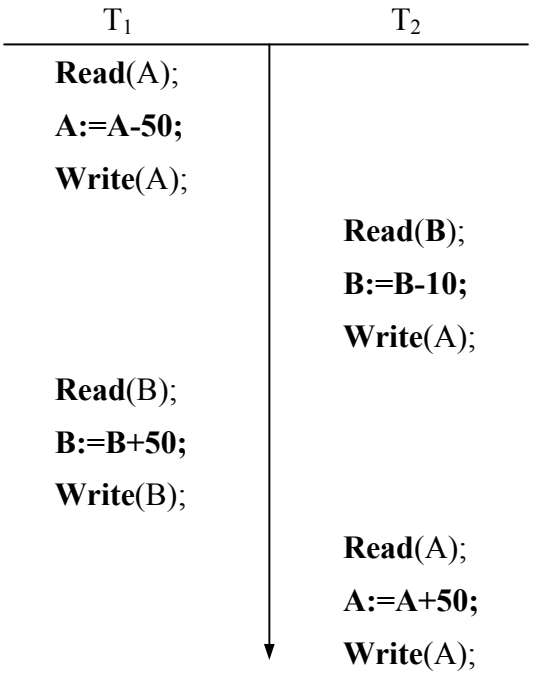
Khái niệm tương đương xung đột dẫn đến khái niệm tuần tự xung đột. Ta nói một lịch trình S là khả tuần tự xung đột (*conflict serializable*) nếu nó tương đương xung đột với một lịch trình tuần tự. Như vậy, schedule-3 là khả tuần tự xung đột. Như một ví dụ, lịch trình schedule-7 dưới đây không tương đương xung đột với một lịch trình tuần tự nào do vậy nó không là khả tuần tự xung đột:



Schedule-7

figure IV- 13

Có thể có hai lịch trình sinh ra cùng kết quả, nhưng không tương đương xung đột. Ví dụ, giao dịch T_5 chuyển 10$ từ tài khoản B sang tài khoản A. Ta xét lịch trình schedule-8 như dưới đây, lịch trình này không tương đương xung đột với lịch trình tuần tự $< T_1, T_5 >$ do trong lịch trình schedule-8 chỉ thị **Write(B)** của T_5 xung đột với chỉ thị **Read(B)** của T_1 như vậy ta không thể di chuyển tất cả các chỉ thị của T_1 về trước các chỉ thị của T_5 bởi việc hoán đổi liên tiếp các chỉ thị không xung đột. Tuy nhiên, các giá trị sau cùng của tài khoản A và B sau khi thực hiện lịch schedule-8 hoặc sau khi thực hiện lịch trình tuần tự $<T_1, T_5 >$ là như nhau--- là 960 và 2040 tương ứng. Qua ví dụ này ta thấy cần thiết phải phân tích cả sự tính toán được thực hiện bởi các giao dịch mà không chỉ các hoạt động **Read** và **Write.** Tuy nhiên sự phân tích như vậy sẽ nặng nề và phải trả một giá tính toán cao hơn.



Schedule-8

figure IV- 14

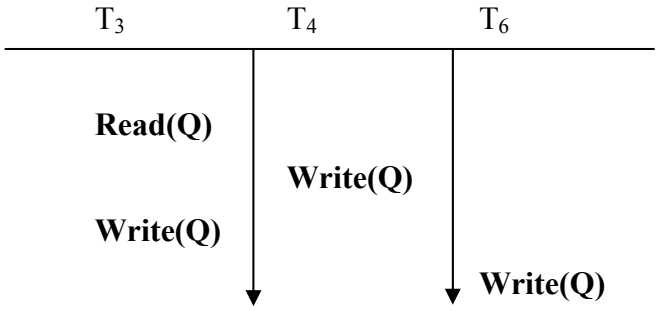
TUẦN TỰ VIEW (View Serializability)

Xét hai lịch trình S và S', trong đó cùng một tập hợp các giao dịch tham gia vào cả hai lịch trình. Các lịch trình S và S' được gọi là tương đương view nếu ba điều kiện sau được thoả mãn:

1. Đối với mỗi hạng mục dữ liệu Q, nếu giao dịch T_i đọc giá trị khởi đầu của Q trong lịch trình S, thì giao dịch T_i phải cũng đọc giá trị khởi đầu của Q trong lịch trình S'.
2. Đối với mỗi hạng mục dữ liệu Q, nếu giao dịch T_i thực hiện **Read(Q)** trong lịch trình S và giá trị đó được sản sinh ra bởi giao dịch T_j thì T_i cũng phải đọc giá trị của Q được sinh ra bởi giao dịch T_j trong S'.
3. Đối với mỗi hạng mục dữ liệu Q, giao dịch thực hiện hoạt động **Write(Q)** sau cùng trong lịch trình S, phải thực hiện hoạt động **Write(Q)** sau cùng trong lịch trình S'.

Điều kiện 1 và 2 đảm bảo mỗi giao dịch đọc cùng các giá trị trong cả hai lịch trình và do vậy thực hiện cùng tính toán. Điều kiện 3 đi cặp với các điều kiện 1 và 2 đảm bảo cả hai lịch trình cho ra kết quả là trạng thái cuối cùng của hệ thống như nhau. Trong các ví dụ trước, schedule-1 là không tương tương view với lịch trình 2 do, trong schedule-1, giá trị của tài khoản A được đọc bởi giao dịch T_2 được sinh ra bởi T_1 , trong khi điều này không xảy ra trong schedule-2. Schedule-1 tương đương view với schedule-3 vì các giá trị của các tài khoản A và B được đọc bởi T_2 được sinh ra bởi T_1 trong cả hai lịch trình.

Quan niệm tương đương view đưa đến quan niệm tuần tự view. Ta nói lịch trình S là khả tuần tự view (view serializable) nếu nó tương dương view với một lịch trình tuần tự. Ta xét lịch trình sau:



Schedule-9

figure IV- 15

Nó tương đương view với lịch trình tuần tự < T_3, T_4, T_6 > do chỉ thị **Read(Q)** đọc giá trị khởi đầu của Q trong cả hai lịch trình và T_6 thực hiện **Write** sau cùng trong cả hai lịch trình như vậy schedule-9 khả tuần tự view.

Mỗi lịch trình khả tuần tự xung đột là khả tuần tự view, nhưng có những lịch trình khả tuần tự view không khả tuần tự xung đột (ví dụ schedule-9).

Trong schedule-9 các giao dịch T_4 và T_6 thực hiện các hoạt động **Write(Q)** mà không thực hiện hoạt động **Read(Q)**, Các Write dạng này được gọi là các Write mù (blind write). Các Write mù xuất hiện trong bất kỳ lịch trình khả tuần tự view không khả tuần tự xung đột.

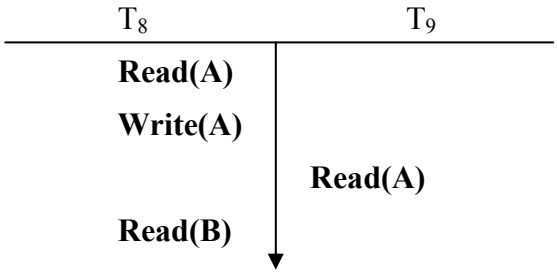
TÍNH KHẢ PHỤC HỒI (Recoverability)

Ta đã nghiên cứu các lịch trình có thể chấp nhận dưới quan điểm sự nhát quán của CSDL với giả thiết không có giao dịch nào thất bại. Ta sẽ xét hiệu quả của thất bại giao dịch trong thực hiện cạnh tranh.

Nếu giao dịch T_i thất bại vì lý do nào đó, ta cần huỷ bỏ hiệu quả của giao dịch này để đảm bảo tính nguyên tử của giao dịch. Trong hệ thống cho phép thực hiện cạnh tranh, cũng cần thiết đảm bảo rằng bất kỳ giao dịch nào phụ thuộc vào T_i cũng phải bị bỏ. Để thực hiện sự chắc chắn này, ta cần bố trí các hạn chế trên kiểu lịch trình được phép trong hệ thống.

LỊCH TRÌNH KHẢ PHỤC HỒI (Recoverable Schedule)

Xét lịch trình schedule-10 trong đó T_9 là một giao dịch chỉ thực hiện một chỉ thị **Read(A)**. Giả sử hệ thống cho phép T_9 bàn giao (commit) ngay sau khi thực hiện chỉ thị **Read(A).** Như vậy T_9 bàn giao trước T_8 . Giả sử T_8 thất bại trước khi bàn giao, vì T_9 vì T_9 đã đọc giá trị của hạng mục giữ liệu A được viết bởi T_8 , ta phải bỏ dở T_9 để đảm bảo tính nguyên tử giao dịch. Song T_9 đã được bàn giao và không thể bỏ dở được. Ta có tình huống trong đó không thể khôi phục đúng sau thất bại của T_8 .



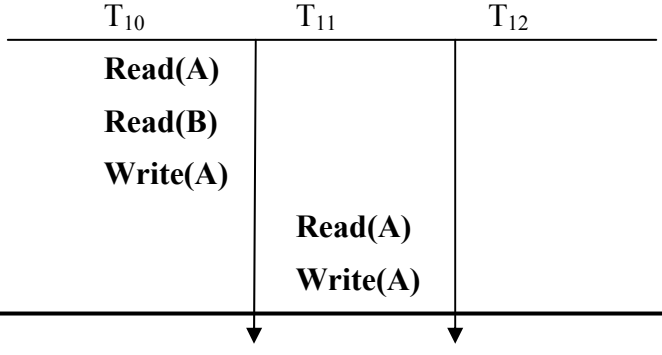
Schedule-10

figure IV- 16

Lịch trình schedule-10 là một ví dụ về lịch trình không phục hồi được và không dược phép. Hầu hết các hệ CSDL đòi hỏi tất cả các lịch trình phải phục hồi được. *Một lịch trình khả phục hồi là lịch trình trong đó, đối với mỗi cặp giao dịch T_i , T_j , nếu T_j đọc hạng mục dữ liệu được viết bởi T_i thì hoạt động bàn giao của T_j phải xảy ra sau hoạt động bàn giao của T_i .*

LỊCH TRÌNH CASCADELESS (Cascadeless Schedule)

Ngay cả khi lịch trình là khả phục hồi, để phục hồi đúng sau thất bại của một giao dịch T_i ta phải cuộn lại một vài giao dịch. Tình huống như thế xảy ra khi các giao dịch đọc dữ liệu được viết bởi T_i . Ta xét lich trình schedule-11 sau



Read(A)

Schedule-11

figure IV- 17

Giao dịch T_{10} viết một giá trị được đọc bởi T_{11} . Giao dịch T_{12} đọc một giá trị được viết bởi T_{11} . Giả sử rằng tại điểm này T_{10} thất bại. T_{10} phải cuộn lại, do T_{11} phụ thuộc vào T_{10} nên T_{11} cũng phải cuộn lại và cũng như vậy với T_{12}. Hiện tượng trong đó một giao dịch thất bại kéo theo một dãy các giao dịch phải cuộn lại được gọi là sự cuộn lại hàng loạt (cascading rollback).

Cuộn lại hàng loạt dẫn đến việc huỷ bỏ một khối lượng công việc đáng kể. Phải hạn chế các lịch trình để việc cuộn lại hàng loạt không thể xảy ra. Các lịch trình như vậy được gọi là các lịch trình cascadeless. *Một lịch trình cascadeless là một lịch trình trong đó mỗi cặp giao dịch T_i , T_j nếu T_j đọc một hạng mục dữ liệu được viết trước đó bởi T_i, hoạt động bàn giao của T_i phải xuất hiện trước hoạt động đọc của T_J* . Một lịch trình cascadeless là khả phục hồi.

THỰC THI CÔ LẬP (Implementation of Isolation)

Có nhiều sơ đồ điều khiển cạnh tranh có thể được sử dụng để đảm bảo các tính chất một lịch trình phải có (nhằm giữ CSDL ở trạng thái nhất quán, cho phép quản lý các giao dịch ...), ngay cả khi nhiều giao dịch thực hiện cạnh tranh, chỉ các lịch trình có thể chấp nhận được sinh ra, bất kể hệ điều hành chia sẻ thời gian tài nguyên như thế nào giữa các giao dịch.

Như một ví dụ, ta xét một sơ đồ điều khiển cạnh tranh sau: Một giao dịch tậu một chốt (lock) trên toàn bộ CSDL trước khi nó khởi động và tháo chốt khi nó đã bàn giao. Trong khi giao dịch giữ chốt không giao dịch nào khác được phép tậu chốt và như vậy phải chờ đến tận khi chốt được tháo. Trong đối sách chốt, chỉ một giao dịch được thực hiện tại một thời điểm và như vậy chỉ lịch trình tuần tự được sinh ra. Sơ đồ điều khiển cạnh tranh này cho ra một hiệu năng cạnh tranh nghèo nàn. Ta nói nó cung cấp một bậc cạnh tranh nghèo (poor degree of concurrency).

Mục đích của các sơ đồ điều khiển cạnh tranh là cung cấp một bậc canh tranh cao trong khi vẫn đảm bảo các lịch trình được sinh ra là khả tuần tự xung đột hoặc khả tuần tự view và cascadeless.

ĐỊNH NGHĨA GIAO DỊCH TRONG SQL

Chuẩn SQL đặc tả sự bắt đầu một giao dịch một cánh không tường minh. Các giao dịch được kết thúc bởi một trong hai lệnh SQL sau:

- **Commit work** bàn giao giao dịch hiện hành và bắt đầu một giao dịch mới
- **Rollback work** gây ra sự huỷ bỏ giao dịch hiện hành

Từ khoá **work** là chọn lựa trong cả hai lệnh. Nếu một chương trình kết thúc thiếu cả hai lệnh này, các cập nhật hoặc được bàn giao hoặc bị cuộn lại là các sự thực hiện phụ thuộc.

Chuẩn cũng đặc tả hệ thống phải đảm bảo cả tính khả tuần tự và tính tự do từ việc cuộn lại hàng loạt. Định nghĩa tính khả tuần tự được ding bởi chuẩn là một lịch trình phải có cùng hiệu quả như một lịch trình tuần tự như vậy tính khả tuần tự xung đột và view đều được chấp nhận.

Chuẩn SQL-92 cũng cho phép một giao dịch đặc tả nó có thể được thực hiện theo một cách mà có thể làm cho nó trở nên không khả tuần tự với sự tôn trọng các giao dịch khác. Ví dụ, một giao dịch có thể hoạt động ở mức **Read uncommitted**, cho phép giao dịch đọc các mẩu tin them chí nếu chúng không được bàn giao. Đặc điểm này được cung cấp cho các giao dich dài các

kết quả của chúng không nhất thiết phải chính xác. Ví dụ, thông tin xấp xỉ thường là đủ cho các thống kê được dùng cho tối ưu hoá vấn tin.

Các mức nhất quán được đặc tả trong SQL-92 là:

- **Serializable :** mặc nhiên
- **Repeatable read :** chỉ cho phép đọc các record đã được bàn giao, hơn nữa yêu cầu giữa hai **Read** trên một record bởi một giao dịch không một giao dịch nào khác được phép cập nhật record này. Tuy nhiên, giao dịch có thể không khả tuần tự với sự tôn trọng các giao dịch khác. Ví dụ, khi tìm kiếm các record thoả mãn các điều kiện nào đó, một giao dịch có thể tìm thấy một vài record được xen bởi một giao dịch đã bàn giao,
- **Read committed:** Chỉ cho phép đọc các record đã được bàn giao, nhưng không có yêu cầu thêm trên các **Read** khả lặp. Ví dụ, giữa hai **Read** của một record bởi một giao dịch, các mẩu tin có thể được cập nhật bởi các giao dịch đã bàn giao khác.
- **Read uncommitted:** Cho phép đọc cả các record chưa được bàn giao. Đây là mức nhất quán thấp nhất được phép trong SQL-92.

KIỂM THỬ TÍNH KHẢ TUẦN TỰ

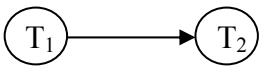
Khi thiết kế các sơ đồ điều khiển cạnh tranh, ta phải chứng tỏ rằng các lịch trình được sinh ra bởi sơ đồ là khả tuần tự. Để làm điều đó, trước tiên ta phải biết làm thế nào để xác định, với một lịch trình cụ thể đã cho, có là khả tuần tự hay không.

KIỂM THỬ TÍNH KHẢ TUẦN TỰ XUNG ĐỘT

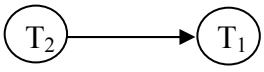
Giả sử S là một lịch trình. Ta xây dựng một đồ thị định hướng, được gọi là đồ thị trình tự (precedence graph), từ S. Đồ thị gồm một cặp (V, E) trong đó V là tập các đỉnh và E là tập các cung. Tập các đỉnh bao gồm tất cả các giao dịch tham gia vào lịch trình. Tập các cung bao gồm tất cả các cung dạng $T_i \rightarrow T_j$ sao cho một trong các điều kiện sau được thoả mãn:

1. T_i thực hiện **Write(Q)** trước T_j thực hiện **Read(Q).**
2. T_i thực hiện **Read(Q)** trước khi T_j thực hiện **Write(Q).**
3. T_i thực hiện **Write(Q)** trước khi T_j thực hiện **Write(Q).**

Nếu một cung $T_i \rightarrow T_j$ tồn tại trong đồ thị trình tự, thì trong bất kỳ lịch trình tuần tự S' nào tương đương với S, T_i phải xuất hiện trước T_j .

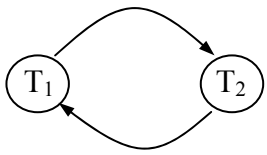
Đồ thị trình tự đối với schedule-1 là: 

vì tất cả các chỉ thị của T_1 được thực hiện trước chỉ thị đầu tiên của T_2

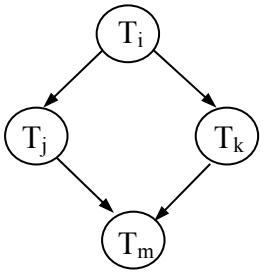
Đồ thị trình tự đối với schedule-2 là: 

vì tất cả các chỉ thị của T_2 được thực hiện trước chỉ thị đầu tiên của T_1

Đồ thị trình tự đối với schedule-4 chứa các cung $T_1 \rightarrow T_2$ vì T_1 thực hiện **Read(A)** trước T_2 thực hiện **Write(A).** Nó cũng chứa cung $T_2 \rightarrow T_1$ vì T_2 thực hiện **Read(B)** trước khi T_1 thực hiện **Write(B):**



Nếu đồ thị trình tự đối với S có chu trình, khi đó lịch trình S không là khả tuần tự xung đột. Nếu đồ thị không chứa chu trình, khi đó lịch trình S là khả tuần tự xung đột. Thứ tự khả tuần tự có thể nhận được thông qua sắp xếp topo (topological sorting), nó xác định một thứ tự tuyến tính nhất quán với thứ tự bộ phận của đồ thị trình tự. Nói chung, có một vài thứ tự tuyến tính có thể nhận được qua sắp xếp topo. Ví dụ, đồ thị sau:



Có hai thứ tự tuyến tính chấp nhận được là:

figure IV- 18

Như vậy, để kiểm thử tính khả tuần tự xung đột, ta cần xây dựng đồ thị trình tự và gọi thuật toán phát hiện chu trình. Ta nhận được một sơ đồ thực nghiệm để xác định tính khả tuần tự xung đột. Như ví dụ, schedule-1 và schedule-2, đồ thị trình tự của chúng không có chu trình, do vậy chúng là các chu trình khả tuần tự xung đột, trong khi đồ thị trình tự của schedule-4 chứa chu trình do vậy nó không là khả tuần tự xung đột.

KIỂM THỬ TÍNH KHẢ TUẦN TỰ VIEW

Ta có thể sửa đổi phép kiểm thử đồ thị trình tự đối với tính khả tuần tự xung đột dể kiểm thử tính khả tuần tự view. Tuy nhiên, phép kiểm thử này phải trả giá cao về thời gian chạy.

Xét lịch trình schedule-9, nếu ta tuân theo quy tắc trong phép kiểm thử tính khả tuần tự xung đột để tạo đồ thị trình tự, ta nhận được đồ thị sau:

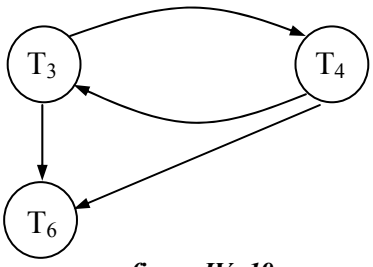


figure IV- 19

Đồ thị này có chu trình, do vậy schedule-9 không là khả tuần tự xung đột. Tuy nhiên, đã đã thấy nó là khả tuần tự view (do nó tương đương với lịch trình tuần tự < T_3, T_4 , T_6 >).

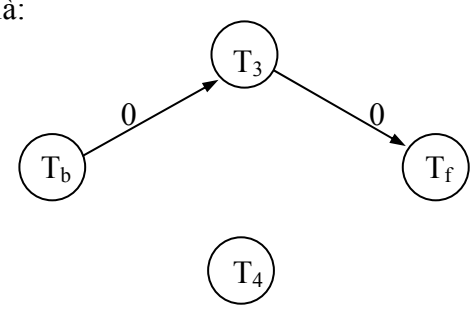
Cung $T_3 \rightarrow T_4$ không được xen vào đồ thị vì các giá trị của hạng mục Q được sản sinh bởi T_3 và T_4 không được dùng bởi bất kỳ giao dịch nào khác và T_6 sản sinh ra giá trị cuối mới của Q. Các chỉ thị **Write(Q)** của T_3 và T_4 được gọi là các **Write vô dụng (Useless Write)**. Điều trên chỉ ra rằng không thể sử dụng đơn thuần sơ đồ đồ thị trình tự để kiểm thử tính khả tuần tự view. Cần thiết phát triển một sơ đồ cho việc quyết định cung nào là cần phải xen vào đồ thị trình tự.

Xét một lịch trình S. Giả sử giao dịch T_j đọc hạng mục dữ liệu Q được viết bởi T_i . Rõ ràng là nếu S là khả tuần tự view, khi đó, trong bất kỳ lịch trình tuần tự S' tương đương với S, T_i phải đi trước T_j . Bây giờ giả sử rằng, trong lịch trình S, giao dịch T_k thực hiện một **Write(Q)**, khi đó, trong lịch trình S', T_k phải hoặc đi trước T_i hoặc đi sau T_j . Nó không thể xuất hiện giữa T_i và T_j vì như vậy T_j không đọc giá trị của Q được viết bởi T_i và như vậy S không tương đương view với S'. Các ràng buộc này không thể biểu diễn được trong thuật ngữ của mô hình đồ thị trình tự đơn giản được nêu lên trước đây. Như trong ví dụ trước, khó khăn nảy sinh ở chỗ ta biết một trong hai cung $T_k \rightarrow T_i$ và $T_j \rightarrow T_k$ phải được xen vào đồ thị nhưng ta chưa tạo được quy tắc để xác định sự lựa chọn thích hợp. Để tạo ra quy tắc này, ta cần mở rộng đồ thị định hướng để bao hàm các cung gán nhãn, ta gọi đồ thị như vậy là đồ thị trình tự gán nhãn (Label precedence graph). Cũng như trước đây, các nút của đồ thị là tất cả các giao dịch tham gia vào lịch trình. Các quy tắc xen cung gán nhãn được diễn giải như sau:

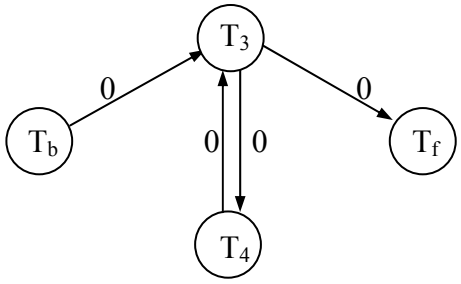
Giả sử S là lịch trình gồm các giao dịch { T_1 , T_2 , ... , T_n }. T_b và T_f là hai giao dịch giả: T_b phát ra **Write(Q)** đối với mỗi Q được truy xuất trong S, T_f phát ra **Read(Q)** đối với mỗi Q được truy xuất trong S. Ta xây dựng lich trình mới S' từ S bằng cách xen T_b ở bắt đầu của S và T_f ở cuối của S. Đồ thị trình tự gán nhãn đố với S' được xây dung dựa trên các quy tắc:

1. Thêm cung $T_i \rightarrow^0 T_j$, nếu T_j đọc giá trị của hạng mục dữ liệu Q được viết bởi T_i
2. Xoá tất cả các cung liên quan tới các giao dịch vô dụng. Một giao dịch T_i được gọi là vô dụng nếu không có con đường nào trong đồ thị trình tự dẫn từ T_i đến T_f .
3. Đối với mỗi hạng mục dữ liệu Q sao cho T_j đọc giá trị của Q được viết bởi T_i và T_k thực hiện **Write(Q),** $T_k \neq T_b$ tiến hành các bước sau
   a. Nếu $T_i = T_b$ và $T_j \neq T_f$, khi đó xen cung $T_j \rightarrow^0 T_k$ vào đồ thị trình tự gán nhãn
   b. Nếu $T_i \neq T_b$ và $T_j = T_f$ khi đó xen cung $T_k \rightarrow^0 T_i$ vào đồ thị trình tự gán nhãn
   c. Nếu $T_i \neq T_b$ và $T_j \neq T_f$ khi đó xen cả hai cung $T_k \rightarrow^p T_i$ và $T_j \rightarrow^p T_k$ vào đồ thị trình tự gán nhãn, trong đó p là một số nguyên duy nhất lớn hơn 0 mà chưa được sử dụng trước đó để gán nhãn cung.

Quy tắc 3c phản ánh rằng nếu T_i viết hạng mục dữ liệu được đọc bởi T_j thì một giao dịch T_k viết cùng hạng mục dữ liệu này phải hoặc đi trước T_i hoặc đi sau T_j . Quy tắc 3a và 3b là trường hợp đặc biệt là kết quả của sự kiện T_b và T_f cần thiết là các giao dịch đầu tiên và cuối cùng tương ứng. Như một ví dụ, ta xét schedule-7. Đồ thị trình tự gán nhãn của nó được xây dựng qua các bước 1 và 2 là:



Đồ thị sau cùng của nó là (cung $T_3 \rightarrow T_4$ là kết quả của 3a, cung $T_4 \rightarrow T_3$ là kết quả của 3b) :



Đồ thị trình tự gán nhãn của schedule-9 là:

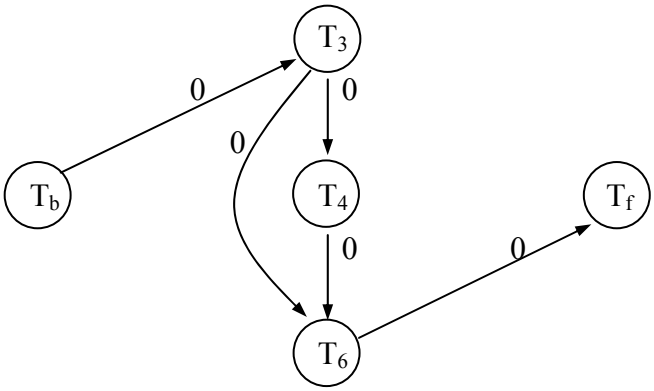
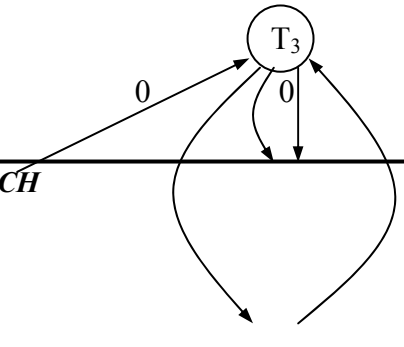


figure IV- 20

Cuối cùng, ta xét lịch trình schedule-10:

T_3	T_3	T_7
Read(Q)		
	Write(Q)	
		Read(Q)
Write(Q)		
		Write(Q)

Đồ thị trình tự gán nhãn của schedule-10 là:



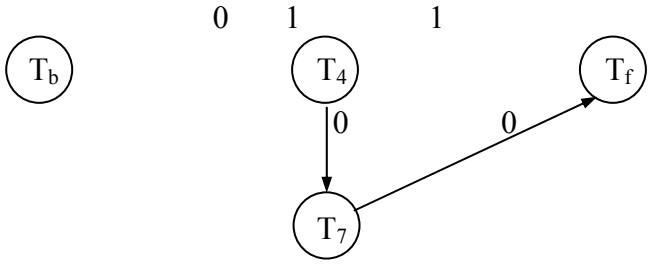


figure IV- 21

Đồ thị trình tự gán nhãn của schedule-7 chứa chu trình tối tiểu : $T_3 \rightarrow T_4 \rightarrow T_3$

Đồ thị trình tự gán nhãn của schedule-10 chứa chu trình tối tiểu: $T_3 \rightarrow T_1 \rightarrow T_3$

Đồ thị trình tự gán nhãn của schedule-9 không chứa chứa chu trình nào.

Nếu đồ thị trình tự gán nhận không chứa chu trình, lịch trình tương ứng là khả tuàn tự view, như vậy schedule-9 là khả tuần tự view. Tuy nhiên, nếu đồ thị chứa chu trình, điều kiện này không kéo theo lịch trình tương ứng không là khả tuần tự view. Đồ thị trình tự gán nhãn của schedule-7 chứa chu trình và lịch trình này không là khả tuần tự view. Bên cạnh đó, lịch trình schedule-10 là khả tuần tự view, nhưng đồ thị trình tự gán nhãn của nó có chứa chu trình. Bây giờ ta giả sử rằng có n cặp cung tách biệt, *đó là do ta đã áp dụng n lần quy tắc 3c trong sự xây dựng đồ thị trình tự.* Khi đó có 2^n đồ thị khác nhau, mỗi một chứa đúng một cung trong mỗi cặp. Nếu một đồ thị nào đó trong các đồ thị này là phi chu trình, khi đó lịch trình tương ứng là khả tuần tự view. Thuật toán này đòi hỏi một phép kiểm thử vét cạn các đồ thị riêng biệt, và như vậy thuộc về lớp vấn đề NP-complet !!!

Ta xét đồ thị schedule-10. nó có đúng một cặp tách biệt. Hai đồ thị triên biệt là:

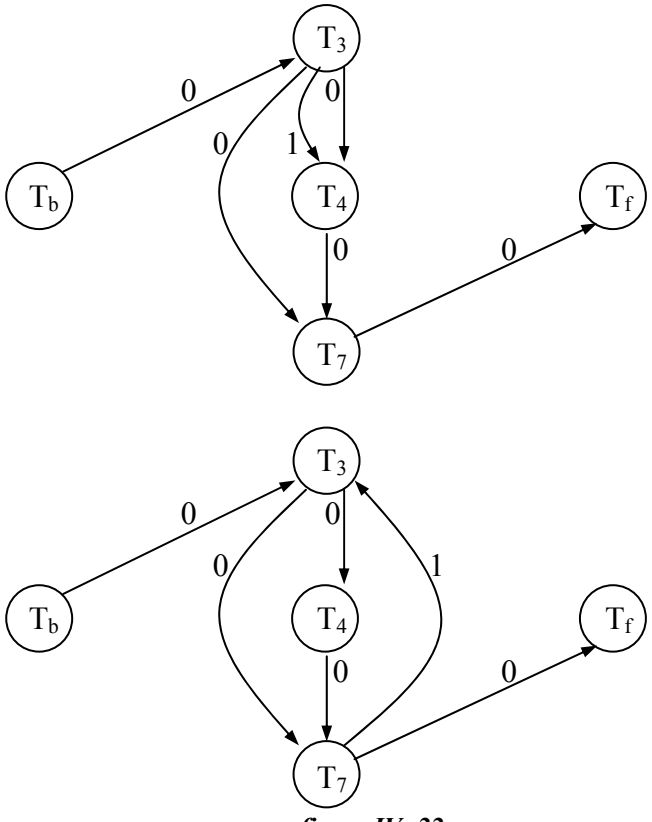


figure IV- 22

Đồ thị thứ nhất không chứa chu trình, do vậy lịch trình là khả tuần tự view.

BÀI TẬP CHƯƠNG IV

IV.1 Liệt kê các tính chất ACID. Giải thích sự hữu ích của mỗi một trong chúng.

IV.2 Trong khi thực hiện, một giao dịch trải qua một vài trạng thái đến tận khi nó bàn giao hoặc bỏ dở. Liệt kê tất cả các dãy trạng thái có thể giao dịch có thể trải qua. Giải thích tại sao mỗi bắc cầu trạng thái có thể xảy ra.

IV.3 Giải thích sự khác biệt giữa lịch trình tuần tự (Serial schedule) và lịch trình khả tuần tự (Serializable schedule).

IV.4 Xét hai giao dịch sau:

T_1 : **Read(A);**
Read(B);
If A=0 then B:=B+1;
Write(B).

T_2 : **Read(B);**
Read(A);
If B=0 then A:=A+1;
Write(A).

Giả thiết yêu cầu nhất quán là A=0 V B=0 với A=B=0 là các giá trị khởi đầu

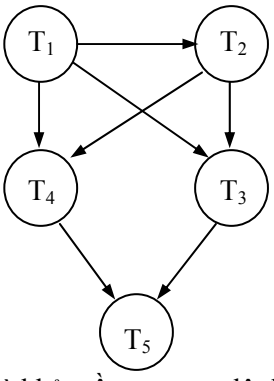
a. Chứng tỏ rằng mỗi sự thự hiện tuần tự bao gồm hai giao dịch này bảo tồn tính nhất quán của CSDL.

b. Nêu một sự thực hiện cạnh tranh của T_1 và T_2 sinh ra một lịch trình không khả tuần tự.

c. Có một sự thực hiện cạnh tranh của T_1 và T_2 sinh ra một lịch trình khả tuần tự không ?

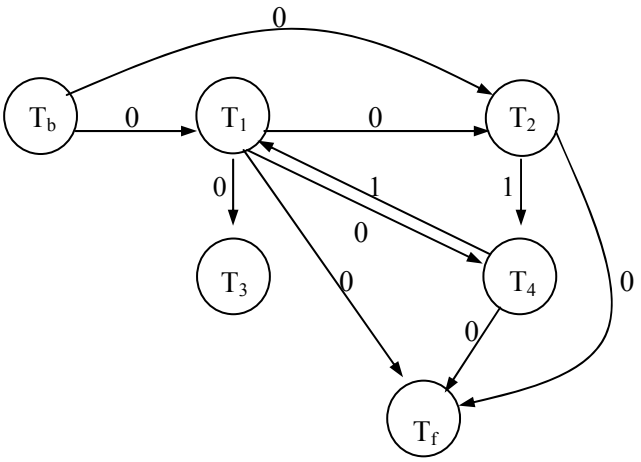
IV.5 Do một lịch trình khả tuần tự xung đột là một lịch trình khả tuần tự view. Tại sao ta lại nhấn mạnh tính khả tuần tự xung đột hơn tính khả tuần tự view?

IV.6 Xét đồ thị trình tự sau:



Lịch trình tương ứng là khả tuần tự xung đột không? Giải thích

IV.7 Xét đồ thị trình tự gán nhãn sau:



Lịch trình tương ứng là khả tuần tự view không? Giải thích.

IV.8 Lịch trình khả phục hồi là gì? Tại sao cần thiết tính khả phục hồi của một lịch trình?

IV.9 Lịch trình cascadeless là gì? Tại sao cần thiết tính cascadeless của lịch trình?

CHƯƠNG V
ĐIỀU KHIỂN CẠNH TRANH
(Concurrency Control)

MỤC ĐÍCH

Một trong các tính chất cơ bản của một giao dịch là *tính cô lập*. Khi một vài giao dịch thực hiện một cách cạnh tranh trong CSDL, tính cô lập có thể không được bảo tồn. Đối với hệ thống, cần phải điều khiển sự trao đổi giữa các giao dịch cạnh tranh; sự điều khiển này được thực hiện thông qua một trong tập hợp đa dạng các cơ chế được gọi là *sơ đồ điều khiển cạnh tranh*.

Các sơ đồ điều khiển cạnh tranh được xét trong chương này được dựa trên tính khả tuần tự. Trong chương này ta cũng xét sự quản trị các giao dịch thực hiện cạnh canh nhưng không xét đến sự cố hỏng hóc.

YÊU CẦU

Hiểu các khái niệm

Hiểu các kỹ thuật điều khiển cạnh tranh:

- Các kỹ thuật dựa trên chốt (lock)
- Các kỹ thuật dựa trên tem thởi gian
- Các kỹ thuật hỗp hợp

Hiểu nguyên lý của các kỹ thuật này

Hiểu các kỹ thuật điều khiển deadlock

V.1. GIAO THỨC DỰA TRÊN CHỐT

Một phương pháp để đảm bảo tính khả tuần tự là yêu cầu việc truy xuất đến hạng mục dữ liệu được tiến hành theo kiểu loại trừ tương hỗ; có nghĩa là trong khi một giao dịch đang truy xuất một hạng mục dữ liệu, không một giao dịch nào khác có thể sửa đổi hạng mục này. Phương pháp chung nhất được dùng để thực thi yêu cầu này là cho phép một giao dịch truy xuất một hạng mục dữ liệu chỉ nếu nó đang giữ chốt trên hạng mục dữ liệu này.

V.1.1. CHỐT (Lock)

Có nhiều phương thức chốt hạng mục dữ liệu. Ta hạn chế việc nghiên cứu trên hai phương thức:

1. **Shared.** Nếu một giao dịch T_i nhận được một chốt ở phương thức shared (ký hiệu là S) trên hạng mục Q, khi đó T_i có thể đọc, nhưng không được viết Q.
2. **Exclusive.** Nếu một giao dịch T_i nhận được một chốt ở phương thức Exclusive (ký hiệu là X), khi đó T_i có thể cả đọc lẫn viết Q.

Ta yêu cầu là mỗi giao dịch đòi hỏi một chốt ở một phương thức thích hợp trên hạng mục dữ liệu Q, phụ thuộc vào kiểu hoạt động mà nó sẽ thực hiện trên Q. Giả sử một giao dịch T_i đồi hỏi một chốt phương thức A trên hạng mục Q mà trên nó giao dịch T_j ($T_j \neq T_i$) hiện đang giữ một chốt phương thức B. Nếu giao dịch T_i có thể được cấp một chốt trên Q ngay, bất chấp sự hiện diện của chốt phương thức B, khi đó ta nói phương thức A tương thích với phương thức B. Một hàm như vậy có thể được biểu diễn bởi một ma trận. Quan hệ tương thích giữa hai phương thức chốt được cho bởi ma trận **comp** sau:

	S	X
S	True	False
X	False	False

Comp(A, B)= true có nghĩa là các phương thức A và B tương thích.

figure V- 1

Các chốt phương thức **shared** có thể được giữ đồng thời trên một hạng mục dữ liệu. Một chốt **exclusive** đến sau phải chờ đến tận khi tất cả các chốt phương thức **shared** đến trước được tháo ra.

Một giao dịch yêu cầu một chốt **shared** trên hạng mục dữ liệu Q bằng cách thực hiện chỉ thị **lock-S(Q)**, yêu cầu một chốt **exclusive** thông qua chỉ thị **lock-X(Q)**. Một hạng mục dữ liệu Q có thể được tháo chốt thông qua chỉ thị **unlock(Q)**.

Để truy xuất một hạng mục dữ liệu, giao dịch T_i đầu tiên phải chốt hạng mục này. Nếu hạng mục này đã bị chốt bởi một giao dịch khác ở phương thức không tương thích, bộ điều khiển cạnh tranh sẽ không cấp chốt cho đến tận khi tất cả các chốt không tương thích bị giữ bởi các giao dịch khác được tháo. Như vậy T_i phải chờ đến tận khi tất cả các chốt không tương thích bị giữ bởi các giao dịch khác được giải phóng.

Giao dịch T_i có thể tháo chốt một hạng mục dữ liệu mà nó đã chốt trước đây. Một giao dịch cần thiết phải giữ một chốt trên một hạng mục dữ liệu chừng nào mà nó còn truy xuất hạng mục này. Hơn nữa, đối với một giao dịch việc tháo chốt ngay sau truy xuất cuối cùng đến hạng mục dữ liệu không luôn luôn là điều mong muốn vì như vậy tính khả tuần tự có thể không được đảm bảo. Để minh hoạ cho tình huống này, ta xét ví dụ sau: A và B là hai tài khoản có thể được

truy xuất bởi các giao dịch T_1 và T_2 . Giao dịch T_1 chuyển 50$ từ tài khoản B sang tài khoản A và đươch xác định như sau:

T_1 : **Lock-X(B);**
Read(B);
B:=B-50;
Write(B);
Unlock(B);
Lock-X(A);
Read(A);
A:=A+50;
Write(A);
Unlock(A);

figure V- 2

Giao dịch T_2 hiển thị tổng số lượng tiền trong các tài khoản A và B (A + B) và được xác định như sau;

T_2 : **Lock-S(A);**
Read(A);
Unlock(A);
Lock-S(B);
Read(B);
Unlock(B);
Display(A+B);

figure V- 3

Giả sử giá trị của tài khoản A và B tương ứng là 100$ và 200$. Nếu hai giao dịch này thực hiện tuần tự, hoặc theo thứ tự T_1, T_2 hoặc theo thứ tự T_2 , T_1 , và khi dó T_2 sẽ hiển thị giá trị 300$. Tuy nhiên nếu các giao dịch này thực hiện cạnh tranh, giả sử theo lịch trình schedule-1, trong trường hợp như vậy giao dịch T_2 sẽ hiển thị giá trị 250$ --- một kết quả không đúng. Lý do của sai lầm này là do giao dịch T_1 đã tháo chốt hạng mục B quá sớm và T_2 đã tham khảo một trạng thái không nhất quán !!!

Lịch trình schedule 1 bày tỏ các hành động được thực hiện bởi các giao dịch cũng như các thời điểm khi các chốt được cấp bởi bộ quản trị điều khiển cạnh tranh. Giao dịch đưa ra một yêu cầu chốt không thể thực hiện hành động kế của mình đến tận khi chốt được cấp bởi bộ quản trị điều khiển cạnh tranh; do đó, chốt phải được cấp trong khoảng thời gian giữa hoạt động yêu cầu chốt và hành động sau của giao dịch. Sau này ta sẽ luôn giả thiết chốt được cấp cho giao dịch ngay trước hành động kế và như vậy ta có thể bỏ qua cột bộ quản trị điều khiển cạnh tranh trong bảng

T_1	T_2	Bộ quản trị điều khiển cạnh tranh
Lock-X(B)		
		Grant-X(B,T_1)
Read(B)		
B:=B-50		
Write(B)		
Unlock(B)		
	Lock-S(A)	
		Grant-S(A,T_2)
	Read(A)	
	Unlock(A)	
	Lock-S(B)	
		Grant-S(B,T_2)
	Read(B)	
	Unlock(B)	
	Display(A+B)	
Lock-X(A)		
		Grant-X(A,T_1)
Read(A)		
A:=A+50		
Write(A)		
Unlock(A)		

Schedule-1

figure V- 4

Bây giờ giả sử rằng tháo chốt bị làm trễ đến cuối giao dịch. Giao dịch T_3 tương ứng với T_1 với tháo chốt bị làm trễ được định nghĩa như sau:

T_3 : **Lock-X(B);**
Read(B);
B:=B-50;
Write(B);
Lock-X(A);
Read(A);
A:=A+50;
Write(A);
Unlock(B);
Unlock(A);

figure V- 5

Giao dịch T_4 tương ứng với T_2 với tháo chốt bị làm trễ được xác định như sau:

T_4 : **Lock-S(A);**
Read(A);
Lock-S(B);
Read(B);
Display(A+B);
Unlock(A);
Unlock(B);

figure V- 6

Các lịch trình có thể trên T_3 và T_4 không để cho T_4 hiển thị trạng thái không nhất quán.

Tuy nhiên, sử dụng chốt có thể dẫn đến một tình huống không mong đợi. Ta hãy xét lịch trình bộ phận schedule-2 trên T_3 và T_4 sau:

T_3	T_4
Lock-X(B)	

Schedule-2

figure V- 7

Do T_3 giữ một chốt phương thức Exclusive trên B, nên yêu cầu một chốt phương thức shared của T_4 trên B phải chờ đến khi T_3 tháo chốt. Cũng vậy, T_3 yêu cầu một chốt Exclusive trên A trong khi T_4 đang giữ một chốt shared trên nó và như vậy phải chờ. Ta gặp phải tình huống trong đó T_3 chờ đợi T_4 đồng thời T_4 chờ đợi T_3 -- một sự chờ đợi vòng tròn -- và như vậy không giao dịch nào có thể tiến triển. Tình huống này được gọi là deadlock (khoá chết). Khi tình huống khoá chết xảy ra hệ thống buộc phải cuộn lại một trong các giao dịch. Mỗi khi một giao dịch bị cuộn lại, các hạng mục dữ liệu bị chốt bởi giao dịch phải được tháo chốt và nó trở nên sẵn có cho giao dịch khác, như vậy các giao dịch này có thể tiếp tục được sự thực hiện của nó.

Nếu ta không sử dụng chốt hoặc tháo chốt hạng mục dữ liệu ngay khi có thể sau đọc hoặc viết hạng mục, ta có thể rơi vào trạng thái không nhất quán. Mặt khác, nếu ta không tháo chốt một hạng mục dữ liệu trước khi yêu cầu một chốt trên một hạng mục khác, dealock có thể xảy ra. Có các phương pháp tránh dealock trong một số tình huống, tuy nhiên nói chung dealock là khó tránh khi sử dụng chốt nếu ta muốn tránh trạng thái không nhất quán. Dealock được ưa thích hơn trạng thái không nhất quán vì chúng có thể điều khiển được bằng cách cuộn lại các giao dịch trong khi

đó trạng thái không nhất quán có thể dẫn đến các vấn đề thực tế mà hệ CSDL không thể điều khiển.

Ta sẽ yêu cầu mỗi giao dịch trong hệ thống tuân theo một tập các quy tắc , được gọi là giao thức chốt (locking protocol), chỉ định khi một giao dịch có thể chốt và tháo chốt mỗi một trong các hạng mục dự liệu. Giao thức chốt hạn chế số các lịch trình có thể. Tập các lịch trình như vậy là một tập con thực sự của tập tất cả các lịch trình khả tuần tự có thể.

Xét { T_0 , T_1 , ..., T_n } một tập các giao dịch tham gia vào lịch trình S. Ta nói T_i đi trước T_j trong S, và được viết là $T_i \rightarrow T_j$, nếu tồn tại một hạng mục dữ liệu Q sao cho T_i giữ chốt phương thức A trên Q , T_j giữ chốt phương thức B trên Q muộn hơn và **comp(A,B)** = false. Nếu $T_i \rightarrow T_j$, thì T_i sẽ xuất hiện trước T_j trong bất kỳ lịch trình tuần tự nào.

Ta nói một lịch trình S là hợp lệ dưới một giao thức chốt nếu S là một lịch trình tuân thủ các quy tắc của giao thức chốt đó. Ta nói rằng một giao thức chốt đảm bảo tính khả tuần tự xung đột nếu và chỉ nếu đối với tất cả các lịch trình hợp lệ, quan hệ $\rightarrow$ kết hợp là phi chu trình.

V.1.2. CẤP CHỐT

Khi một giao dịch yêu cầu một chốt trên một hạng mục dữ liệu ở một phương thức và không có một giao dịch nào khác giữ một chốt trên cùng hạng mục này ở một phương thức xung đột, chốt có thể được cấp. Tuy nhiên, phải thận trọng để tránh kịch bản sau: giả sử T_2 giữ một chốt phương thức shared trên một hạng mục dữ liệu, một giao dịch khác T_1 yêu cầu một chốt phương thức exclusive cũng trên hạng mục này, rõ ràng T_1 phải chờ T_2 tháo chốt. Trong khi đó một giao dịch khác T_3 yêu cầu một chốt phương thức shared, do yêu cầu chốt này tương thích với phương thức chốt được giữ bởi T_1 nên nó được cấp cho T_3. Tại thời điểm T_2 tháo chốt, T_1 vẫn phải chờ sự tháo chốt của T_3, nhưng bây giờ lại có một giao dịch T_4 yêu cầu một chốt phương thức shared và nó lại được cấp do tính tương thích và cứ như vậy, có thể T_1 sẽ không bao giờ được cấp chốt mà nó yêu cầu trên hạng mục dữ liệu. Ta gọi hiện tượng này là bị chết đói (starved).

Để tránh sự chết đói của các giao dịch, việc cấp chốt được tiến hành như sau: Khi một giao dịch T_i yêu cầu một chốt trên một hạng mục dữ liệu Q ở phương thức M, chốt sẽ được cấp nếu các điều kiện sau được thoả mãn:

1. Không có giao dịch khác đang giữ một chốt trên Q ở phương thức xung đột với M
2. Không có một giao dịch nào đang chờ được cấp một chốt trên M và đã đưa ra yêu cầu về chốt trước T_i

V.1.3. GIAO THỨC CHỐT HAI KỲ (Two-phase locking protocol)

Giao thức chốt hai kỳ là một giao thức đảm bảo tính khả tuần tự. Giao thức này yêu cầu mỗi một giao dịch phát ra yêu cầu chốt và tháo chốt thành hai kỳ:

1. **Kỳ xin chốt (Growing phase).** Một giao dịch có thể nhận được các chốt, nhưng có không thể tháo bất kỳ chốt nào
2. **Kỳ tháo chốt (Shrinking phase).** Một giao dịch có thể tháo các chốt nhưng không thể nhận được một chốt mới nào.

Khởi đầu, một giao dịch ở kỳ xin chốt. Giao dịch tậu được nhiều chốt như cần thiết. Mỗi khi giao dịch tháo một chốt, nó đi vào kỳ tháo chốt và nó không thể phát ra bất kỳ một yêu cầu chốt nào nữa. Các giao dich T_3 và T_4 là hai kỳ. Các giao dịch T_1 và T_2 không là hai kỳ. Người ta có thể chứng minh được giao thức chốt hai kỳ đảm bảo tính khả tuần tự xung đột, nhưng không đảm bảo tránh được dealock và việc cuộn lại hàng loạt. Cuộn lại hàng loạt có thể tránh được bởi một sự sửa đổi chốt hai kỳ được gọi là giao thức chốt hai kỳ nghiêm ngặt. Chốt hai kỳ nghiêm

ngặt đòi hỏi thêm tất cả các chốt phương thức exclusive phải được giữ đến tận khi giao dịch bàn giao. Yêu cầu này đảm bảo rằng bất kỳ dữ liệu nào được viết bởi một giao dịch chưa bàn giao bị chốt trong phương thức exclusive đến tận khi giao dịch bàn giao, điều đó ngăn ngừa bất kỳ giao dịch khác đọc dữ liệu này.

Một biến thể khác của chốt hai kỳ là giao thức chốt hai kỳ nghiêm khắc. Nó đòi hỏi tất cả các chốt được giữ đến tận khi giao dịch bàn giao. Hầu hết các hệ CSDL thực hiện chốt hai kỳ nghiêm ngặt hoặc nghiêm khắc.

Một sự tinh chế giao thức chốt hai kỳ cơ sở dựa trên việc cho phép chuyển đổi chốt: nâng cấp một chốt shared sang exclusive và hạ cấp một chốt exclusive thành chốt shared. Chuyển đổi chốt không thể cho phép một cách tuỳ tiện, nâng cấp chỉ được phép diễn ra trong kỳ xin chốt, còn hạ cấp chỉ được diễn ra trong kỳ tháo chốt. Một giao dịch thử nâng cấp một chốt trên một hạng mục dữ liệu Q có thể phải chờ. Giao thức chốt hai kỳ với chuyển đổi chốt cho phép chỉ sinh ra các lịch trình khả tuần tự xung đột. Nếu các chốt exclusive được giữ đến tận khi bàn giao, các lịch trình sẽ là cascadeless.

Ta xét một ví dụ: Các giao dịch T_8 và T_9 được nêu trong ví dụ chỉ được trình bày bởi các hoạt động ý nghĩa là **Read** và **Write.**

T_8 : **Read(A_1);**
Read(A_2);
...
Read(A_n);
Write(A_1).

T_9 : **Read(A_1);**
Read(A_2);
Display($A_1 + A_2$).

figure V- 8

Nếu ta sử dụng giao thức chốt hai kỳ, khi đó T_8 phải chốt A_1 ở phương thức exclusive. Bởi vậy, sự thực hiện cạnh tranh của hai giao dịch rút cuộc trở thành thực hiện tuần tự. Ta thấy rằng T_8 cần một chốt exclusive trên A_1 chỉ ở cuối sự thực hiện của nó, khi nó write(A_1). Như vậy, T_8 có thể khởi động chốt A_1 ở phương thức shared, và đổi chốt này sang phương thức exclusive sau này. Như vậy ta có thể nhận được tính cạnh tranh cao hơn, vì như vậy T_8 và T_9 có thể truy xuất đến A_1 và A_2 đồng thời.

Ta biểu thị sự chuyển đổi từ phương thức shared sang phương thức exclusive bởi upgrade và từ phương thức exclusive sang phương thức shared bởi downgrade. Upgrade chỉ được phép xảy ra trong kỳ xin chốt và downgrade chỉ được phép xảy ra trong kỳ tháo chốt. Lịch trình chưa hoàn tất dưới đây cho ta một minh hoạ về giao thức chốt hai kỳ với chuyển đổi chốt.

Chú ý rằng một giao dịch thử cập nhật một chốt trên một hạng mục dữ liệu Q có thể buộc phải chờ. Việc chờ bắt buộc này xảy ra khi Q đang bị chốt bởi giao dịch khác ở phương thức shared.

Giao thức chốt hai kỳ với chuyển đổi chốt chỉ sinh ra các lịch trình khả tuần tự xung đột, các giao dịch có thể được tuần tự hoá bởi các điểm chốt của chúng. Hơn nữa, nếu các chốt exclusive được giữ đến tận khi kết thúc giao dịch, lịch trình sẽ là cascadeless.

T_8	T_9
Lock-S(A_1)	
	Lock-S(A_2)
Lock-S(A_2)	
	Lock-S(A_2)
Lock-S(A_3)	
…	
	Unlock(A_1)
	Unlock(A_2)
UpGrade(A_1)	

figure V- 9

Ta mô tả một sơ đồ đơn giản nhưng dược sử dụng rộng rãi để sinh tự động các chỉ thị chốt và tháo chốt thích hợp cho một giao dịch: Mỗi khi giao dich T xuất ra một chỉ thị **Read(Q),** hệ thống sẽ xuất ra một chỉ thị **Lock-S(Q)** ngay trước chỉ thị **Read(Q)**. Mỗi khi giao dịch T xuất ra một hoạt động **Write(Q),** hệ thống sẽ kiểm tra xem T đã giữ một chốt shared nào trên Q hay chưa, nếu đã, nó xuất ra một chỉ thị **Upgrade(Q)** ngay trước chỉ thị **Write(Q),** nếu chưa, nó xuất ra chỉ thị **Lock-X(Q)** ngay trước **Write(Q).** Tất cả các chốt giao dịch nhận được sẽ được tháo chốt sau khi giao dịch bàn giao hay bỏ dở.

V.1.4. GIAO THỨC DỰA TRÊN ĐỒ THỊ (Graph-Based Protocol)

Ta đã biết, trong trường hợp thiếu vắng các thông tin liên quan đến cách thức các hạng mục dữ liệu được truy xuất, giao thức chốt hai kỳ là cần và đủ để đảm bảo tính khả tuần tự. Nếu ta muốn phát triển các giao thức không là hai kỳ, ta cần các thông tin bổ xung trên cách thức mỗi giao dịch truy xuất CSDL. Có nhiều mô hình khác nhau về lượng thông tin được cung cấp. Mô hình đơn giản nhất đòi hỏi ta phải biết trước thứ tự trong đó các hạng mục dữ liệu sẽ được truy xuất. Với các thông tin như vậy, có thể xây dựng các giao thức chốt không là hai kỳ nhưng vẫn đảm bảo tính khả tuần tự xung đột.

Để có được hiểu biết trước như vậy, ta áp đặt một thứ tự bộ phận, ký hiệu $\rightarrow$, trên tập tất cả các hạng mục dữ liệu $D = \{ d_1 , d_2 , ..., d_n \}$. Nếu $d_i \rightarrow d_j$, bất kỳ giao dịch nào truy xuất cả d_i và d_j phải truy xuất d_i trước khi truy xuất d_j . Thứ tự bộ phận này cho phép xem D như một đồ thị định hướng phi chu trình, được gọi là đồ thị CSDL (DataBase Graph). Trong phần này, để đơn giản, ta hạn chế chỉ xét các đồ thị là các cây và ta sẽ đưa ra một giao thức đơn giản, được gọi là giao thức cây (tree protocol), giao thức này hạn chế chỉ dùng các chốt exclusive.

Trong giao thức cây, chỉ cho phép chỉ thị chốt **Lock-X**, mỗi giao dịch T có thể chốt một hạng mục dữ liệu nhiều nhất một lần và phải tuân theo các quy tắc sau:

1. Chốt đầu tiên bởi T có thể trên bất kỳ hạng mục dữ liệu nào
2. Sau đó, một hạng mục dữ liệu Q có thể bị chốt bởi T chỉ nếu cha của Q hiện đang bị chốt bởi T
3. Các hạng mục dữ liệu có thể được tháo chốt bất kỳ lúc nào

4. Một hạng mục dữ liệu đã bị chốt và được tháo chốt bởi T, không thể bị T chốt lại lần nữa.

Các lịch trình hợp lệ trong giao thức cây là khả tuần tự xung đột.

Ví dụ: Cây CSDL là:

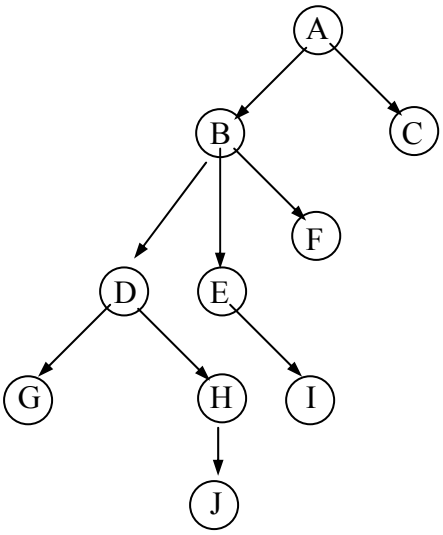


figure V- 10

Chỉ các chỉ thị chốt và tháo chốt của các giao dịch được trình bày:

T_{10} : **Lock-X(B); Lock-X(E); Lock-X(D); Unlock(B); Unlock(E); Lock-X(G); Unlock(D); Unlock(G).**

T_{11} : **Lock-X(D); Lock-X(H); Unlock(D); Unlock(H).**

T_{12} : **Lock-X(B); Lock-X(E); Unlock(B); Unlock(E).**

T_{13} : **Lock-X(D); Lock-X(H); Unlock(D); Unlock(H).**

figure V- 11

Một lịch trình tuân theo giao thức cây chứa tất cả bốn giao dịch trên được cho trong hình bên dưới. Ta nhận thấy, các lịch trình tuân thủ giao thức cây không chỉ là khả tuần tự xung đột mà còn đảm bảo không có dealock. Giao thức cây có mặt thuận lợi so với giao thức hai kỳ là tháo chốt có thể xảy ra sớm hơn. Việc tháo chốt sớm có thể dẫn đến rút ngắn thời gian chờ đợi và tăng tính cạnh tranh. Hơn nữa, do giao thức là không dealock, nên không có cuộn lại. Tuy nhiên giao thức cây có điểm bất lợi là, trong một vài trường hợp, một giao dịch có thể phải chốt những hạng mục dữ liệu mà nó không truy xuất. Chẳng hạn, một giao dịch cần truy xuất các hạng mục dữ liệu A và J trong đồ thị CSDL trên, phải chốt không chỉ A và J mà phải chốt cả các hạng mục B, D, H. Việc chốt bổ xung này có thể gây ra việc tăng tổng phí chốt, tăng thời gian chờ đợi và giảm tính cạnh tranh. Hơn nữa, nếu không biết trước các hạng mục dữ liệu nào sẽ cần thiết phải chốt, các giao dịch sẽ phải chốt gốc của cây mà điều này làm giảm mạnh tính cạnh tranh.

Đối với một tập các giao dịch, có thể có các lịch trình khả tuần tự xung đột không thể nhận được từ việc tuân theo giao thức cây. Có các lịch trình được sinh ra bởi tuân theo giao thức chốt hai kỳ nhưng không thể được sinh ra bởi tuân theo giao thức cây và ngược lại.

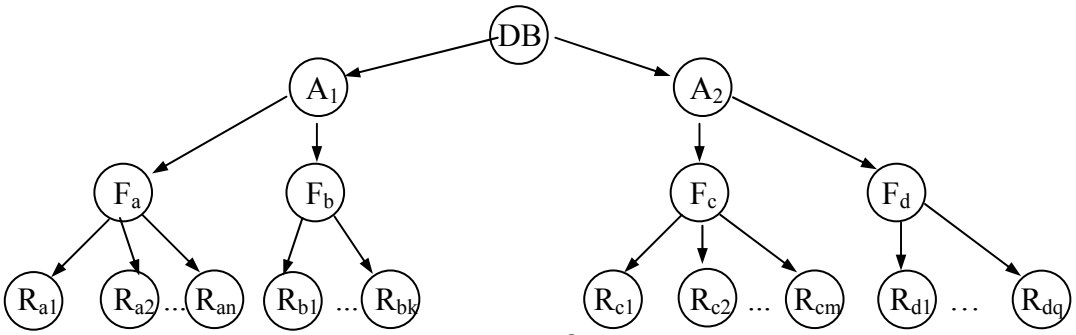
T_{10}	T_{11}	T_{12}	T_{13}
Lock-X(B)			
	Lock-X(D)		
	Lock-X(H)		
	Unlock(D)		
Lock-X(E)			
Lock-X(D)			
Unlock(B)			
Unlock(E)			
		Lock-X(B)	
		Lock-X(E)	
	Unlock(H)		
Lock-X(G)			
Unlock(D)			
			Lock-X(D)
			Lock-X(H)
			Unlock(D)
			Unlock(H)
		Unlock(E)	
		Unlock(B)	
Unlock(G)			

Lịch trình khả tuần tự dưới giao thức cây

figure V- 12

V.1.5. ĐA HẠT (Multiple Granularity)

Trong các sơ đồ điều khiển cạnh tranh được mô tả trước đây, ta đã sử dụng hạng mục dữ liệu như đơn vị trên nó sự đồng bộ hoá được thực hiện. Tuy nhiên, có các hoàn cảnh trong đó việc nhóm một vài hạng mục dữ liệu và xử lý chúng như một đơn vị đồng bộ hoá mang lại nhiều lợi ích. Nừu một giao dich T_i phải truy xuất toàn bộ CSDL và giao thức chốt được sử dụng, khi đó T_i phải chốt mỗi hạng mục dữ liệu trong CSDL. Như vậy việc thực hiện các chốt này sẽ tiêu tốn một thời gian đáng kể. Sẽ hiệu quả hơn nếu T_i chỉ cần một yêu cầu chốt để chốt toàn bộ CSDL. Mặt khác, nếu T_i cần truy xuất chỉ một vài hạng mục dữ liệu, nó không cần thiết phải chốt toàn bộ CSDL vì như vậy sẽ giảm tính cạnh tranh. Như vậy, cái mà ta cần là một cơ chế cho phép hệ thống xác định nhiều mức hạt. Một cơ chế như vậy là cho phép các hạng mục dữ liệu có kích cỡ khác nhau và xác định một sự phân cấp các hạt dữ liệu, trong đó các hạt nhỏ được ẩn náu bên trong các hạt lớn. Sự phân cấp như vậy có thể được biểu diễn đồ thị như một cây. Một nút không là lá của cây đa hạt biểu diễn dữ liệu được kết hợp với con cháu của nó. Như một ví dụ minh hoạ, ta xét cây sau:



Phân cấp hạt

figure V- 13

Nó gồm bốn mức nút. Mức cao nhất là toàn bộ CSDL. Thấp hơn là các nút kiểu vùng: CSDL bao gồm các vùng này. Mỗi vùng lại có các nút kiểu file như các con của nó, mỗi vùng chứa đúng các file này và không file nào nằm trong nhiều hơn một vùng. Cuối cùng, mỗi file có các nút con kiểu mẩu tin, không mẩu tin nào hiện diện trong hơn một file.

Mỗi nút trong cây có thể được chốt một các cá nhân. Như đã làm trong giao thức chốt hai kỳ, ta sẽ sử dụng các phương thức chốt **shared** và **exclusive** . Khi một giao dịch chốt một nút, trong phương thức **shared** hoặc **exclusive**, giao dịch cũng chốt tất cả các nút con cháu của nút này ở cùng phương thức. Ví dụ T_i chốt tường minh file F_b ở phương thức exclusive, nó đã chốt ẩn tất cả các mẩu tin của F_b cũng trong phương thức exclusive.

Giả sử giao dịch T_j muốn chốt mẩu tin R_{b6} của file F_b. vì giao dịch T_i đã chốt tường minh file F_b, mẩu tin R_{b6} cũng bị chốt ẩn. Song làm thế nào để hệ thống biết được T_j có thể chốt R_{b6} hay không: T_j phải duyệt cây từ gốc đến mẩu tin R_{b6} , nếu có một nút bất kỳ tren đường dẫn bị chốt ở phương thức không tương thích, T_j phải chờ. Bây giờ, nếu T_k muốn chốt toàn bộ CSDL, nó phải chốt nút gốc. Tuy nhiên, do T_i hiện đang giữ một chốt trên F_b, một bộ phận của cây, nên T_k sẽ không thành công. Vậy làm thế nào để hệ có thể chốt được nút gốc: Một khả năng là tìm kiếm trên toàn bộ cây, giải pháp này phá huỷ hoàn toàn sơ đồ mục đích của sơ đồ chốt đa hạt. Một giải pháp hiệu quả hơn là đưa vào một lớp mới các phương thức chốt, được gọi là phương thức chốt tăng cường (intension lock mode). Nếu một nút bị chốt ở phương thức tăng cường, chốt tường minh được tiến hành ở mức thấp hơn của cây (hạt min hơn). Chốt tăng cường được được đặt trên tất cả các tổ tiên của một nút trước khi nút đó được chốt tường minh. Như vậy một giao dịch không cần thiết phải tìm kiếm toàn bộ cây để xác định nó có thể chốt một nút thành công hay không. Một giao dịch muốn chốt một nút, chẳng hạn Q, phải duyệt một đường dẫn từ gốc đến Q, trong khi duyệt cây, giao dịch chốt các nút trên đường đi ở phương thức tăng cường.

Có một phương thức tăng cường kết hợp với phương thức **shared** và một với phương thức **exclusive**. Nếu một nút bị chốt ở phương thức tăng cường shared (IS), chốt tường minh được tiến hành ở mức thấp hơn trong cây, nhưng chỉ là một các chốt phương thức shared. Tương tự, nếu một nút bị chốt ở phương thức tăng cường exclusive (IX), chốt tường minh được tiến hành ở mức thấp hơn với các chốt exclusive hoặc shared. Nếu một nút bị chốt ở phương thức shared và phương thức tăng cường exclusive (SIX), cây con có gốc là nút này bị chốt tường minh ở phương thức shared và chốt tường minh được tiến hành ở mức thấp hơn với các chốt exclusive. Hàm tính tương thích đối với các phương thức chốt này được cho bởi ma trận:

	IS	IX	S	SIX	X
IS	True	True	True	True	False
IX	True	True	False	False	False
S	True	False	True	False	False
SIX	True	False	False	False	False
X	False	False	False	False	False

figure V- 14

Giao thức chốt đa hạt dưới đây đảm bảo tính khả tuần tự. Mỗi giao dịch T có thể chốt một nút Q theo các quy tắc sau:

1. Hàm tương thích chốt phải được kiểm chứng
2. Gốc của cây phải được chốt đầu tiên, và có thể được chốt ở bất kỳ phương thức nào
3. Một nút Q có thể được chốt bởi T ở phương thức S hoặc IS chỉ nếu cha của Q hiện đang bị chốt bởi T ở hoặc phương thức IX hoặc phương thức IS.
4. Một nút Q có thể được chốt bởi T ở phương thức X, SIX hoặc IX chỉ nếu cha của Q hiện đang bị chốt ở hoặc phương thức IX hoặc phương thức SIX
5. T có thể chốt một nút chỉ nếu trước đó nó chưa tháo chốt một nút nào.
6. T có thể tháo chốt một nút Q chỉ nếu không con nào của Q hiện đang bị chốt bởi T

Ta thấy rằng giao thức đa hạt yêu cầu các chốt được tậu theo thứ tự Top-Down, được tháo theo thứ tự Bottom-Up.

Ví dụ: Xét cây phân cấp hạt như trên và các giao dịch sau:

- Giả sử giao dịch T_{18} đọc mẩu tin R_{a2} của file F_a. Khi đó T_{18} cần phải chốt DB, vùng A_1 và F_a ở phương thức IS và R_{a2} ở phương thức S: **Lock-IS(DB); Lock-IS(A_1); Lock-IS(F_a); lock-S(R_{a2})**
- Giả sử giao dịch T_{19} sửa đổi mẩu tin R_{a9} trong file F_a , khi đó T_{19} cần phải chốt CSDL, vùng A_1 và file F_a ở phương thức IX và R_{a9} ở phương thức X: **Lock-IX(DB); Lock-IX(A_1); lock-IX(F_a); lock-X(R_{a9})**
- Giả sử giao dịch T_{20} đọc tất cả các mẩu tin của file F_a , khi đó T_{20} cần phải chốt CSDL, và vùng A_1 ở phương thức IS và chốt F_a ở phương thức S: **Lock-IS(DB); Lock-IS(A_1); Lock-S(F_a)**
- Giả sử giao dịch T_{21} đọc toàn bộ CSDL, nó có thể làm điều đó sau khi chốt CSDL ở phương thức S: **Lock-S(DB)**

Chú ý rằng T_{18}, T_{20} và T_{21} có thể truy xuất đồng thời CSDL, giao dịch T_{19} có thể thực hiện cạnh tranh với T_{18} nhưng không với T_{20} hoặc T_{21}

V.2. GIAO THỨC DỰA TRÊN TEM THỜI GIAN (Timestamp-based protocol)

V.2.1. TEM THỜI GIAN (Timestamp)

Ta kết hợp với mỗi giao dịch T_i trong hệ thống một tem thời gian cố định duy nhất, được biểu thị bởi $TS(T_i)$. Tem thời gian này được gán bởi hệ CSDL trước khi giao dịch T_i bắt đầu thực

hiện. Nếu một giao dịch T_i đã được gán tem thời gian $TS(T_i)$ và một giao dịch mới T_j đi vào hệ thống, khi đó $TS(T_i) < TS(T_j)$. Có hai phương pháp đơn giản để thực hiện sơ đồ này:

1. Sử dụng giá trị của đồng hồ hệ thống như tem thời gian: Một tem thời gian của một giao dịch bằng giá trị của đồng hồ khi giao dịch đi vào hệ thống.
2. Sử dụng bộ đếm logic: bộ đếm được tăng lên mỗi khi một tem thời gian đã được gán, tem thời gian của một giao dịch bằng với giá trị của bộ đếm khi giao dich đi vào hệ thống.

Tem thời gian của các giao dịch xác định thứ tự khả tuần tự. Như vậy, *nếu $TS(T_i) < TS(T_j)$, hệ thống phải đảm bảo rằng lịch trình được sinh ra là tương đương với một lịch trình tuần tự trong đó T_i xuất hiện trước T_j* .

Để thực hiện sơ đồ này, ta kết hợp với mỗi hạng mục dữ liệu Q hai giá trị tem thời gian:

- **W-timestamp(Q)** biểu thị tem thời gian lớn nhất của giao dịch bất kỳ đã thực hiện **Write(Q)** thành công
- **R-timestamp(Q)** biểu thị tem thời gian lớn nhất của giao dịch bất kỳ đã thực hiện **Read(Q)** thành công

Các tem thời gian này được cập nhật mỗi khi một **Write** hoặc một **Read** mới được thực hiện.

V.2.2. GIAO THỨC THỨ TỰ TEM THỜI GIAN (Timestamp-Ordering Protocol)

Giao thức thứ tự tem thời gian đảm bảo rằng các **Write** và **Read** xung đột bất kỳ được thực hiện theo thứ tự tem thời gian. Giao thức này hoạt động như sau:

1. Giả sử giao dịch T_i phát ra **Read(Q)**.
   a. Nếu $TS(T_i) <$ W-Timestamp(Q), T_i cần đọc một giá trị của Q đã được viết rồi. Do đó, hoạt động **Read** bị vứt bỏ và T_i bị cuộn lại.
   b. Nếu $TS(T_i) \geq$ W-Timestamp(Q), hoạt động **Read** được thực hiện và R-Timestamp được đặt bằng giá trị lớn nhất trong hai giá trị R-Timestamp và $TS(T_i)$.
2. Giả sử giao dịch T_i phát ra **Write(Q)**.
   a. Nếu $TS(T_i) <$ R-Timestamp(Q), Giá trị của Q mà T_i đang sinh ra được giả thiết là để được dùng cho các giao dịch đi sau nó (theo trình tự thời gian), nhưng nay không cần đến nữa. Do vậy, hoạt động **Write** này bị vứt bỏ và T_i bị cuộn lại
   b. Nếu $TS(T_i) <$ W-Timestamp(Q), T_i đang thử viết một giá trị đã quá hạn của Q, Từ đó, hoạt động **Write** bị vứt bỏ và T_i bị cuộn lại
   c. Ngoài ra, hoạt động **Write** được thực hiện và W-Timestamp(Q) được đặt là $TS(T_i)$

Một giao dịch T_i bị cuộn lại bởi sơ đồ điều khiển cạnh tranh như kết quả của hoạt động **Read** hoặc **Write** đang được phát ra, được gán với một tem thời gian mới và được tái khởi động lại (được xem như một giao dịch mới tham gia vào hệ thống)

Ta xét các giao dịch T_{14} và T_{15} được xác định như dưới đây:

T_{14} : **Read(B);**
Read(A);
Display(A+B);.

T_{15} : **Read(B);**
B:=B-50;
Write(B);
Read(A);
A:=A+50;
Write(A);
Display(A+B).

figure V- 15

Ta giả thiết rằng một giao dịch được gán cho một tem thời gian ngay trước chỉ thị đầu tiên của nó. Như vậy, lịch trình schedule-3 dưới đây có $TS(T_{14}) < TS(T_{15})$, và là một lịch trình hợp lệ dưới giao thức tem thời gian:

T_{14}	T_{15}
Read(B)	
	Read(B)
	B:=B-50
	Write(B)
Read(A)	
	Read(A)
Display(A+B)	
	A:=A+50
	Write(A)

Schedule-3

figure V- 16

Giao thức thứ tự tem thời gian đảm bảo tính khả tuần tự xung đột và không dealock.

V.2.3. QUY TẮC VIẾT THOMAS (Thomas' Write rule)

Một biến thể của giao thức tem thời gian cho phép tính cạnh tranh cao hơn giao thức thứ tự tem thời gian. Trước hết ta xét lịch trình schedule-4 sau:

T_{16}	T_{17}
Read(Q)	
	Write(Q)
Write(Q)	

Schedule-4

figure V- 17

Nếu áp dụng giao thức thứ tự tem thời gian, ta có $TS(T_{16}) < TS(T_{17})$. Hoạt dộng **Read(Q)** của T_{16} và **Write(Q)** của T_{17} thành công, khi T_{16} toan thực hiện hoạt động **Write(Q)** của nó, vì $TS(T16) < TS(T_{17}) =$ W-timestamp(Q), nên **Write(Q)** của T_{16} bị vứt bỏ và giao dịch T_{16} bị cuộn lại. Sự cuộn lại này là không cần thiết. Nhận xét này cho ta một sửa đổi phiên bản giao thức thứ tự tem thời gian:

Các quy tắc giao thức đối với **Read** không thay đổi, các quy tắc đối với **Write** được thay đổi chút ít như sau:

Giả sử giao dịch T_i phát ra **Write(Q).**

1. Nếu $TS(T_i) <$ R-timestamp(Q), Giá trị của Q mà T_i đang sinh ra được giả thiết là để được dùng cho các giao dịch đi sau nó (theo trình tự thời gian), nhưng nay không cần đến nữa. Do vậy, hoạt động **Write** này bị vứt bỏ và T_i bị cuộn lại.
2. Nếu $TS(T_i) <$ W-timestamp(Q), T_i đang thử viết một giá trị lỗi thời của Q. Do vậy, hoạt động **Write** này có thể bị bỏ lơ (không được thực hiện, nhưng T_i không bị cuộn lại).
3. Ngoài ra, hoạt động **Write** được thực hiện và W-timestamp(Q) được đặt là $TS(T_i)$.

Sự sửa đổi đối với giao thức thứ tự tem thời gian này được gọi là quy tắc viết Thomas. Quy tắc viết Thomas cho khả năng sinh các lịch trình khả tuần tự mà các giao thức thức trước đây không thể.

V.2.4. GIAO THỨC DỰA TRÊN TÍNH HỢP LỆ

Trong trường hợp đa số các giao dịch trong hệ thống là các giao dịch chỉ đọc (read-only), tỷ suất xung đột giữa các giao dịch là thấp. Như vậy nhiều giao dịch trong chúng thực hiện thiếu sự giám sát của sơ đồ điều khiển cạnh tranh cũng vẫn giữ cho hệ thống ở trạng thái nhất quán. Hơn nữa, một sơ đồ điều khiển cạnh tranh đưa vào một tổng phí đáng kể (cho thực hiện mã lệnh, thời gian chờ của giao dịch ...). Việc tìm một sơ đồ với tổng phí nhỏ là một mục tiêu. Nhưng khó khăn là ta phải biết trước những giao dịch sẽ bị dính líu vào một xung đột. Để có được các hiểu bét đó, ta cần một sơ đồ để giám sát hệ thống.

Ta giả thiết rằng mỗi giao dịch T_i , trong thời gian sống của nó, thực hiện trong hai hoặc ba kỳ khác nhau, phụ thuộc vào nó là một giao dịch chỉ đọc hay là một giao dịch cập nhật. Các kỳ này, theo thứ tự, là như sau:

1. **Kỳ đọc.** Trong kỳ này, các giá trị của các hạng mục dữ liệu khác nhau được đọc vào các biến cục bộ của T_i . Tất cả các hoạt động **Write** được thực hiện trên các biến cục bộ tạm, không cập nhật CSDL hiện hành.
2. **Kỳ hợp lệ.** Giao dịch T_i thực hiện một phép kiểm thử sự hợp lệ để xác định xem nó có thể sao chép đến CSDL các biến cục bộ tạm chứa các kết quả của các hoạt **Write** mà không vi phạm tính khả tuần tự xung đột hay không.

3. **Kỳ viết.** Nếu T_i thành công trong kỳ hợp lệ, các cập nhật hiện hành được áp dụng vào CSDL, nếu không T_i bị cuộn lại.

Mỗi giao dịch phải trải qua ba kỳ theo thứ tự trên, tuy nhiên, ba kỳ của các giao dịch đang thực hiện cạnh tranh có thể đan xen nhau.

Các kỳ đọc và kỳ viết tự nó đã rõ ràng. Chỉ có kỳ hợp lệ là cần thảo luận thêm. Để thực hiện kiểm thử sự hợp lệ, ta cần biết khi nào các kỳ khác nhau của giao dịch T_i xảy ra. Do vậy, ta sẽ kết hợp ba tem thời gian với giao dịch T_i :

1. **Start(T_i).** Thời gian khi T_i bắt đầu sự thực hiện.
2. **Validation(T_i).** Thời gian khi T_i kết thúc kỳ đọc và khởi động kỳ hợp lệ.
3. **Finish(T_i).** Thời gian khi T_i kết thúc kỳ viết.

Ta xác định thứ tự khả tuần tự bởi kỹ thuật thứ tự tem thời gian sử dụng giá trị tem thời gian **Validation(T_i).** Như vậy, giá trị $TS(T_i) = Validation(T_i)$ và nếu $TS(T_j) < TS(T_k)$ thì bất kỳ lịch trình nào được sinh ra phải tương đương với một lịch trình tuần tự trong đó giao dịch T_i xuất hiện trước giao dịch T_k . Lý dota chọn $Validation(T_i)$ như tem thời gian của T_i , mà không chọn $Start(T_i)$, là vì ta hy vọng thời gian trả lời sẽ nhanh hơn.

Phép kiểm thử hợp lệ đối với T_j đòi hỏi rằng, đối với mỗi giao dịch T_i với $TS(T_i) < TS(T_j)$, một trong các điều kiện sau phải được thoả mãn:

1. **Finish(T_i) < Start(T_j).** Do T_i hoàn tất sự thực hiện của nó trước khi T_j bắt đầu, thứ tự khả tuần tự được duy trì.
2. **Tập các hạng mục dữ liệu được viết bởi T_i không giao với tập các hạng mục dữ liệu được đọc bởi T_j và T_i hoàn tất kỳ viết của nó trước khi T_j bắt đầu kỳ hợp lệ (Start(T_j) < Finish(T_i) < Validation(T_j)).** Điều kiện này đảm bảo rằng các viết của T_i và T_j là không chồng chéo. Do các viết của T_i không ảnh hưởng tới đọc của T_j và do T_j không thể ảnh hưởng tới đọc của T_i , thứ tự khả tuần tự được duy trì.

Lịch trình schedule-5 cho ta một minh hoạ về giao thức dựa trên tính hợp lệ:

T_{14}	T_{15}
Read(B)	
	Read(B)
	B:=B-50
	Read(A)
	A:=A+50
Read(A)	
Xác nhận tính hợp lệ	
Display(A+B)	
	Xác nhận tính hợp lệ
	Write(B)
	Write(A)

figure V- 18

Sơ đồ hợp lệ tự động canh chừng việc cuộn lại hàng loạt, do các **Write** hiện tại xảy ra chỉ sau khi giao dịch phát ra **Write** đã bàn giao.

V.3. CÁC SƠ ĐỒ ĐA PHIÊN BẢN (Multiversion Schemes)

Các sơ đồ điều khiển cạnh tranh được thảo luận trước đây đảm bảo tính khả tuần tự hoặc bởi làm trễ một hoạt động hoặc bỏ dở giao dịc đã phát ra hoạt động đó. Chẳng hạn, một hoạt động **Read** có thể bị làm trễ vì giá trị thích hợp còn chưa được viết hoặc nó có thể bị vứt bỏ vì giá trị mà nó muốn đọc đã bị viết đè rồi. Các khó khăn này có thể được che đi nếu bản sao cũ của mỗi hạng mục dữ liệu được giữ trong một hệ thống.

Trong các hệ CSDL đa phiên bản, mỗi hoạt động **Write(Q)** tạo ra một bản mới của Q. Khi một hoạt động **Read(Q)** được phát ra, hệ thống chọn lựa một trong các phiên bản của Q để đọc. Sơ đồ điều khiển cạnh tranh phải đảm bảo rằng việc chọn lựa này được tiến hành sao cho tính khả tuần tự được đảm bảo. Do lý do hiệu năng, một giao dịch phải có khả năng xác định dễ dàng và mau chóng phiên bản dhạng mục dữ liệu sẽ đọc.

V.3.1. THỨ TỰ TEM THỜI GIAN ĐA PHIÊN BẢN

Kỹ thuật chung được dùng trong các sơ đồ đa phiên bản là tem thời gian. Ta kết hợp với một giao dịch một tem thời gian tĩnh duy nhất, ký hiệu $TS(T_i)$. Tem thời gian này được gán trước khi khi giao dịch bắt đầu sự thực hiện. Mỗi hạng mục dữ liệu Q kết hợp với một dãy $<Q_1, Q_2, ... , Q_m>$ mỗi phiên bản Q_k chứa ba trường dữ liệu:

- **Content** là giá trị của phiên bản Q_i
- **W-timestamp(Q_k)** là tem thời gian của giao dịch đã tạo ra phiên bản Q_k
- **R-timestamp(Q_k)** là tem thời gian lớn nhất của giao dịch đã đọc thành công phiên bản Q_k

Một giao dịch, gọi là T_i , tạo ra phiên bản Q_k của hạng mục dữ liệu Q bằng cách phát ra một hoạt động **Write(Q)**. Trường **Content** của phiên bản chứa giá trị được viết bởi T_i . **W-timestamp** và **R-timestamp** được khởi động là $TS(T_i)$. Giá trị **R-timestamp** được cập nhật mỗi khi một giao dịch T_j đọc nội dung của Q_k và **R-timestamp(Q_k)** < $TS(T_j)$.

Sơ đồ tem thời gian đa phiên bản dưới đây sẽ đảm bảo tính khả tuần tự. Sơ đồ hoạt động như sau: giả sử T_j phát ra một hoạt động **Read(Q)** hoặc **Write(Q).** Q_k ký hiệu phiên bản của Q, tem thời gian viết của nó là tem thời gian viết lớn nhất nhỏ hơn hoặc bằng $TS(T_j)$.

1. Nếu giao dịch T_j phát ra một **Read(Q)**, khi đó giá trị trả lại là nội dung của phiên bản Q_k
2. Nếu T_j phát ra một **Write(Q)** và nếu $TS(T_j)$ < **R-timestamp(Q_k)** khi đó giao dịch T_j bị cuộn lại. Nếu không, nếu $TS(T_j)$ = **W-timestamp(Q_k)** nội dung của Q_k bị viết đè, khác đi một phiên bản mới của Q được tạo.

Các phiên bản không còn được dùng đến nữa bị xoá đi dựa trên quy tắc sau: Giả sử có hai phiên bản Q_i và Q_j của một hạng mục dữ liệu và cả hai phiên bản này cùng có W-timestamp nhỏ hơn tem thời gian của giao dịch già nhất trong hệ thống, khi đó phiên bản già hơn trong hai phiên bản Q_i và Q_j sẽ không còn được dùng nữa và bị xoá đi.

Sơ đồ thứ tự tem thời gian đa phiên bản có tính chất hay đó là một yêu cầu **Read** không bao giờ thất bại và không phải chờ đợi. Trong một hệ thống mà hoạt động **Read** xảy ra nhiều hơn **Write** cái lợi này là đáng kể. Tuy nhiên có vài điều bất lợi của sơ đồ này là: thứ nhất đọc một

hạng mục dữ liệu cũng đòi hỏi cập nhật trường **R-timestamp**, thứ hai là xung đột giữa các giao dịch được giải quyết bằng cuộn lại.

V.3.2. CHỐT HAI KỲ ĐA PHIÊN BẢN

Giao thức chốt hai kỳ đa phiên bản cố gắng tổ hợp những ưu điểm của điều khiển cạnh tranh với các ưu điểm của chốt hai kỳ. Giao thức này phân biệt các giao dịch ***chỉ đọc*** và các giao dịch ***cập nhật***.

Các giao dịch ***cập nhật*** thực hiện chốt hai kỳ nghiêm khắc (các chốt được giữ đến tận khi kết thúc giao dịch). Mỗi hạng mục dữ liệu có một tem thời gian. Tem thời gian trong trường hợp này không là tem thời gian dựa trên đồng hồ thực mà là một bộ đếm, sẽ được gọi là TS-counter.

Các giao dịch ***chỉ đọc*** được gán tem thời gian là giá trị hiện hành của TS-counter trước khi chúng bắt đầu sự thực hiện: chúng tuân theo giao thức thứ tự tem thời gian đa phiên bản để thực hiện đọc. Như vậy, khi một giao dịch chỉ đọc T_i phát ra một **Read(Q),** giá trị trả lại là nội dung của phiên bản mà tem thời gian của nó là tem thời gian lớn nhất nhỏ hơn $TS(T_i)$.

Khi một giao dịch ***cập nhật*** đọc một hạng mục, nó tậu một chốt **shared** trên hạng mục, rồi đọc phiên bản mới nhất của hạng mục (đối với nó). Khi một giao dịch cập nhật muốn viết một hạng mục, đầu tiên nó tậu một chốt exclusive trên hạng mục này, rồi tạo ra một phiên bản mới cho hạng mục. **Write** được thực hiện trên phiên bản mới này và tem thời gian của phiên bản mới được khởi động là $+\infty$.

Khi một giao dịch cập nhật T_i hoàn tất các hoạt động của nó, nó thực hiện xử lý bàn giao như sau: Đầu tiên, T_i đặt tem thời gian trên mỗi phiên bản nó đã tạo là TS-counter+1; sau đó T_i tăng TS-counter lên 1. Chỉ một giao dịch cập nhật được phép thực hiện xử lý bàn giao tại một thời điểm.

Các phiên bản bị xoá cùng kiểu cách với thứ tự tem thời gian đa phiên bản.

V.4. QUẢN TRỊ DEADLOCK

Một hệ thống ở trạnh thái deadlock nếu tồn tại một tập hợp các giao dịch sao cho mỗi giao dịch trong tập hợp đang chờ một giao dịch khác trong tạp hợp. Chính xác hơn, tồn tại một tập các giao dịch $\{ T_0 , T_2 , ..., T_n \}$ sao cho T_0 đang chờ một hạng mục dữ liệu được giữ bởi T_1 , T_1 đang chờ một hạng mục dữ liệu đang bị chiếm bởi T_2 , ..., T_{n-1} đang chờ một hạng mục dữ liệu được giữ bởi T_n và T_n đang chờ một hạng mục T_0 đang chiếm. Không một giao dịch nào có thể tiến triển được trong tình huống như vậy. Một cách chữa trị là viện dẫn một hành động tẩy rửa, chẳng hạn cuộn lại một vài giao dịch tham gia vào deadlock.

Có hai phương pháp chính giải quyết vấn đề deadlock: Ngăn ngừa deadlock, phát hiện deadlock và khôi phục. Giao thức ngăn ngừa deadlock đảm bảo rằng hệ thống sẽ không bao giờ đi vào trạng thái deadlock. Sơ đồ phát hiện deadlock và khôi phục (deadlock-detection and deadlock-recovery scheme) cho phép hệ thống đi vào trạng thái deadlock và sau đó cố gắng khôi phục. Cả hai phương pháp đều có thể dẫn đến việc cuộn lại giao dịch. Phòng ngừa deadlock thường được sử dụng nếu xác suất hệ thống đi vào deadlock cao, phát hiện và khôi phục hiệu quả hơn trong các trường hợp còn lại.

V.4.1. PHÒNG NGỪA DEADLOCK (Deadlock prevention)

Có hai cách tiếp cận phòng ngừa deadlock: Một đảm bảo không có chờ đợi vòng tròn xảy ra bằng cách sắp thứ tự các yêu cầu chốt hoặc đòi hỏi tất cả các chốt được tậu cùng nhau. Một

cách tiếp cận khác gần hơn với khắc phục deadlock và thực hiện cuộn lại thay vì chờ đợi một chốt. Chờ đợi là tiềm ẩn của deadlock.

Sơ đồ đơn giản nhất dưới cách tiếp cận thứ nhất đòi hỏi mỗi giao dịch chốt tất cả các hạng mục dữ liệu trước khi nó bắt đầu thực hiện. Hơn nữa, hoặc tất cả được chốt trong một bước hoặc không hạng mục nào được chốt. Giao thức này có hai bất lợi chính: một là khó dự đoán, trước khi giao dịch bắt đầu, các hạng mục dữ liệu nào cần được chốt, hai là hiệu suất sử dụng hạng mục dữ liệu rất thấp do nhiều hạng mục có thể bị chốt nhưng không được sử dụng trong một thời gian dài.

Sơ đồ phòng ngừa deadlock khác là áp đặt một thứ tự bộ phận trên tất cả các hạng mục dữ liệu và yêu cầu một giao dịch chốt một hạng mục dữ liệu theo thứ tự được xác định bởi thứ tự bộ phận này.

Cách tiếp cận thứ hai để phòng ngừa deadlock là sử dụng ưu tiên và cuộn lại quá trình. Với ưu tiên, một giao dịch T_2 yêu cầu một chốt bị giữ bởi giao dịch T_1 , chốt đã cấp cho T_1 có thể bị lấy lại và cấp chgo T_2 , T_1 bị cuộn lại. Để điều khiển ưu tiên, ta gán một tem thời gian duy nhất cho mỗi giao dịch. Hệ thống sử dụng các tem thời gian này để quyết định một giao dịch phải chờ hay cuộn lại. Việc chốt vẫn được sử dụng để điều khiển cạnh tranh. Nếu một giao dịch bị cuộn lại, nó vẫn giữ tem thời gian cũ của nó khi tái khởi động. Hai sơ đồ phòng ngừa deadlock sử dụng tem thời gian khác nhau được đề nghị:

1. Sơ đồ **Wait-Die** dựa trên kỹ thuật không ưu tiên. Khi giao dịch T_i yêu cầu một hạng mục dữ liệu bị chiếm bởi T_j , T_i được phép chờ chỉ nếu nó có tem thời gian nhỏ hơn của T_j nếu không T_i bị cuộn lại (die).
2. Sơ đồ **Wound-Wait** dựa trên kỹ thuật ưu tiên. Khi giao dịch T_i yêu cầu một hạng mục dữ liệu hiện đang bị giữ bởi T_j , T_i được phép chờ chỉ nếu nó có tem thời gian lớn hơn của T_j , nếu không T_j bị cuộn lại (Wounded).

Một điều quan trọng là phải đảm bảo rằng, mỗi khi giao dịch bị cuộn lại, nó không bị chết đói (starvation) có nghĩa là nó sẽ không bị cuộn lại lần nữa và được phép tiến triển.

Cả hai sơ đồ **Wound-Wait** và **Wait-Die** đều tránh được sự chết đói: tại một thời điểm, có một giao dịch với tem thời gian nhỏ nhất. Giao dịch này không thể bị yêu cầu cuộn lại trong cả hai sơ đồ. Do tem thời gian luôn tăng và do các giao dịch không được gán tem thời gian mới khi chúng bị cuộn lại, một giao dịch bị cuộn lại sẽ có tem thời gian nhỏ nhất (vào thời gian sau) và sẽ không bị cuộn lại lần nữa.

Tuy nhiên, có những khác nhau lớn trong cách thức hoạt động của hai sơ đồ:

- Trong sơ đồ **Wait-Die**, một giao dịch già hơn phải chờ một giao dịch trẻ hơn giải phóng hạng mục dữ liệu. Như vậy, giao dịch già hơn có xu hướng bị chờ nhiều hơn. Ngược lại, trong sơ đồ **Wound-Wait,** một giao dịch già hơn không bao giờ phải chờ một giao dịch trẻ hơn.
- Trong sơ đồ **Wait-Die,** nếu một giao dịch T_i chết và bị cuộn lại vì nó đòi hỏi một hạng mục dữ liệu bị giữ bởi giao dịch T_j , khi đó T_i có thể phải tái phát ra cùng dãy các yêu cầu khi nó khởi động lại. Nếu hạng mục dữ liệu vẫn bị chiếm bởi T_j , T_i bị chết lần nữa. Như vậy, T_i có thể bị chết vài lần trước khi tậu được hạng mục dữ liệu cần thiết. Trong sơ đồ **Wound-Wait,** Giao dịch T_i bị thương và bị cuộn lại do T_j yêu cầu hạng mục dữ liệu nó chiếm giữ. Khi T_i khởi động lại, và yêu cầu hạng mục dữ liệu, bây giờ, đang bị T_j giữ, T_i chờ. Như vậy, có ít cuộn lại hơn trong sơ đồ **Wound-Wait.**

Một vấn đề nổi trội đối với cả hai sơ đồ là có những cuộn lại không cần thiết vẫn xảy ra.

V.4.2. SƠ ĐỒ DỰA TRÊN TIMEOUT

Một cách tiếp cận khác để quản lý deadlock được dựa trên lock timeout. Trong cách tiếp cận này, một giao dịch đã yêu cầu một chốt phải chờ nhiều nhất một khoảng thời gian xác định. Nếu chốt không được cấp trong khoảng thời gian này, giao dịch được gọi là mãn kỳ (time out), giao dịch tự cuộn lại và khởi động lại. Nếu có một deadlock, một hoặc một vài giao dịch dính líu đến deadlock sẽ time out và cuộn lại, để các giao dịch khác tiến triển. Sơ đồ này nằm trung gian giữa phòng ngừa deadlock và phát hiện và khôi phục deadlock.

Sơ đồ timeout dễ thực thi và hoạt động tốt nếu giao dịch ngắn và nếu sự chờ đợi lâu là do deadlock. Tuy nhiên, khó quyết định được khoảng thời gian timeout. Sơ đồ này cũng có thể đưa đến sự chết đói.

V.4.3. PHÁT HIỆN DEADLOCK VÀ KHÔI PHỤC

Nếu một hệ thống không dùng giao thức phòng ngừa deadlock, khi đó sơ đồ phát hiện và khôi phục phải được sử dụng. Một giải thuật kiểm tra trạng thái của hệ thống được gọi theo một chu kỳ để xác định xem deadlock có xẩy ra hay không. Nếu có, hệ thống phải khôi phục lại từ deadlock, muốn vậy hệ thống phải:

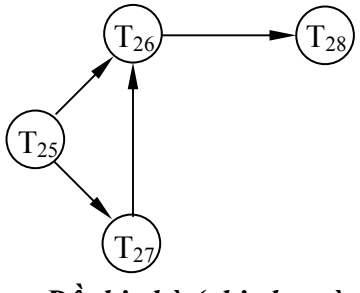
- Duy trì thông tin về sự cấp phát hiện hành các hạng mục dữ liệu cho các giao dịch cũng như các yêu cầu hạng mục dữ liệu chưa được chưa được giải quyết.
- Cung cấp một thuật toán sử dụng các thông tin này để xác định hệ thống đã đi vào trạng thái deadlock chưa.
- Phục hồi từ deadlock khi phát hiện được deadlock đã xảy ra.

V.4.3.1 PHÁT HIỆN DEADLOCK

Deadlock có thể mô tả chính xác bằng đồ thị định hướng được gọi là đồ thị chờ (wait for graph). Đồ thị này gồm một cặp G = < V, E >, trong đó V là tập các đỉnh và E là tập các cung. Tập các đỉnh gồm tất cả các giao dịch trong hệ thống. Mỗi phần tử của E là một cặp $T_i \rightarrow T_j$, nó chỉ ra rằng T_i chờ T_j giải phóng một hạng mục dữ liệu nó cần.

Khi giao dịch T_i yêu cầu một hạng mục dữ liệu đang bị giữ bởi giao dịch T_j khi đó cung $T_i \rightarrow T_j$ được xen vào đồ thị. Cạnh này bị xoá đi chỉ khi giao dịch T_j không còn giữ hạng mục dữ liệu nào mà T_i cần.

Deadlock tồn tại trong hệ thống nếu và chỉ nếu đồ thị chờ chứa một chu trình. Mỗi giao dịch tham gia vào chu trình này được gọi là bị deadlock. Để phát hiện deadlock, hệ thống phải duy trì đồ thị chờ và gọi theo một chu kỳ thủ tục tìm kiếm chu trình. Ta xét ví dụ sau:



Đồ thị chờ (phi chu trình)

figure V- 19

Do đồ thị không có chu trình nên hệ thông không trong trạng thái deadlock. Bây giờ, giả sử T_{28} yêu cầu một hạng mục dữ liệu được giữ bởi T_{27} , cung $T_{28} \rightarrow T_{27}$ được xen vào đồ thị, điều này dẫn đến tồn tại một chu trình $T_{26} \rightarrow T_{27} \rightarrow T_{28} \rightarrow T_{26}$ có nghĩa là hệ thống rơi vào tình trạng deadlock và T_{26} , T_{27} , T_{28} bị deadlock.

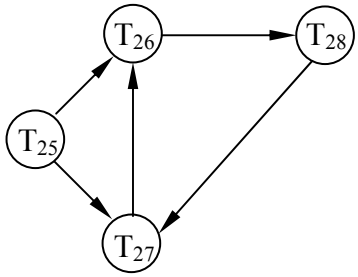


figure V- 20

Vấn đề đặt ra là khi nào thì chạy thủ tục phát hiện ? câu trả lời phụ thuộc hai yêu tố sau:

1. Deadlock thường xảy ra hay không ?
2. Bao nhiêu giao dịch sẽ bị ảnh hưởng bởi deadlock

Nếu deadlock thường xảy ra, việc chạy thủ tục phát hiện diễn ra thường xuyên hơn. Các hạng mục dữ liệu được cấp cho các giao dịch bị deadlock sẽ không sẵn dùng cho các giao dịch khác đến khi deadlock bị phá vỡ. Hơn nữa, số chu trình trong đồ thị có thể tăng lên. Trong trường hợp xấu nhất, ta phải gọi thủ tục phát hiện mỗi khi có một yêu cầu cấp phát không được cấp ngay.

V.4.3.2 KHÔI PHỤC TỪ DEADLOCK

Khi thuật toán phát hiện xác định được sự tồn tại của deadlock, hệ thống phải khôi phục từ deadlock. Giải pháp chung nhất là cuộn lại một vài giao dịch để phá vỡ deadlock. Ba việc cần phải làm là:

1. **Chọn nạn nhân**. Đã cho một tập các giao dịch bị deadlock, ta phải xác định giao dịch nào phải cuộn lại để phá vỡ deadlock. Ta sẽ cuộn lại các giao dịch sao cho giá phải trả là tối thiểu. Nhiều nhân tố xác định giá của cuộn lại:
    a. Giao dịch đã tính toán được bao lâu và bao lâu nữa.
    b. Giao dịch đã sử dụng bao nhiêu hạng mục dữ liệu
    c. Giao dịch cần bao nhiêu hạng mục dữ liệu nữa để hoàn tất.
    d. Bao nhiêu giao dịch bị cuộn lại.
2. **Cuộn lại (Rollback).** Mỗi khi ta đã quyết định được giao dịch nào phải bị cuộn lại, ta phải xác định giao dịch này bị cuộn lại bao xa. Giải pháp đơn giản nhất là cuộn lại toàn bộ: bỏ dở giao dịch và bắt đầu lại nó. Tuy nhiên, sẽ là hiệu quả hơn nếu chỉ cuộn lại giao dịch đủ xa như cần thiết để phá vỡ deadlock. Nhưng phương pháp này đòi hỏi hệ thống phải duy trì các thông tin bổ xung về trạng thái của tất cả các giao dịch đang chạy.
3. **Sự chết đói (Starvation).** Trong một hệ thống trong đó việc chọn nạn nhân dựa trên các nhân tố giá, có thể xảy ra là một giao dịch luôn là nạn nhân của việc chọn này và kết quả là giao dịch này không bao giờ có thể hoàn thành. Tình huống này được gọi là sự chết đói. Phải đảm bảo việc chọn nạn nhân không đưa đến chết đói. Một giải pháp xem số lần bị cuộn lại của một giao dịch như một nhân tố về giá.

V.5. BÀI TẬP CHƯƠNG V

V.1 Chứng tỏ rằng giao thức chốt hai kỳ đảm bảo tính khả tuần tự xung đột và các giao dịch có thể được làm tuần tự tương ứng với các điểm chốt của chúng.

V.2 Xét hai giao dịch sau:

T_{31} : **Read(A);**
Read(B);
If A=0 then B:=B+1;
Write(B);

T_{32} : **Read(B);**
Read(A);
If B=0 then A:=A+1;
Write(A);

Thêm các chỉ thị chốt và tháo chốt vào hai giao dịch T_{31} và T_{32} sao cho chúng tuân theo giao thức chốt hai kỳ. Sự thực hiện các giao dịch này có thể dẫn đến deadlock ?

V.3 Nêu các ưu điểm và nhược điểm của giao thức chốt hai kỳ nghiêm ngặt.

V.4 Nêu các ưu điểm và nhược điểm của giao thức chốt hai kỳ nghiêm khắc.

V.5 Bộ quản trị điều khiển cạnh tranh của một hệ CSDL phải quyết định cấp cho một đòi hỏi chốt hoặc bắt giao dịch yêu cầu phải chờ. Hãy thiết kế một cấu trúc dữ liệu (tiết kiệm không gian) cho phép bộ quản trị điều khiển cạnh tranh cho ra quyết định mau chóng.

V.6 Xét sự mở rộng thành giao thức cây-chốt sau. Nó cho phép cả chốt shared và exclusive:

1. Một giao dịch chỉ đọc chỉ có thể yêu cầu các chốt shared. Một giao dịch cập nhật chỉ có thể yêu cầu các chốt exclusive
2. Mỗi giao dịch phải tuân theo các quy tắc của giao thức cây-chốt. Các giao dịch chỉ đọc đầu tiên có thể chốt bất kỳ hạng mục dữ liệu nào, các giao dịch cập nhật đầu tiên phải chốt gốc.

   Chứng tỏ giao thức đảm bảo tính khả tuần tự và không có deadlock

V.7 Cho ra ví dụ về các lịch trình có thể dưới giao thức cây nhưng không thể dưới giao thức chốt hai kỳ và ngược lại.

V.8 Xét một biến thể của giao thức cây, được gọi là giao thức rừng. CSDL được tổ chức như một rừng. Mỗi giao dịch T phải tuân theo các quy tắc sau:

1. Chốt đầu tiên trong mỗi cây có thể trên bất kỳ hạng mục dữ liệu nào
2. Chốt từ thứ hai trở về sau có thể được yêu cầu chỉ nếu cha của nút được yêu cầu hiện đang bị chốt
3. Các hạng mục dữ liệu có thể được tháo chốt bất kỳ lúc nào
4. Một hạng mục dữ liệu không được chốt lại bởi T sau khi T đã tháo chốt cho nó

   Chứng tỏ giao thức rừng không đảm bảo tính khả tuần tự

V.9 Khi một giao dịch bị cuộn lại dưới thứ tự tem thời gian, nó được gán một tem thời gian mới. Tại sao không đơn giản để nó giữ lại tem thời gian cũ ?

V.10 Trong chốt đa hạt, sự khác nhau giữa chốt tường minh và chốt ẩn là gì ?

V.11 phương thức chốt SIX là hữu dụng trong chốt đa hạt. Phương thức exclusive và shared tăng cường không được sử dụng. Tại sao nó lại vô dụng ?

V.12 Đối với mỗi một trong các giao thức sau, mô tả sắc thái ứng dụng thực tế gợi ý sử dụng giao thức và sắc thái gợi ý không nên sử dụng giao thức:

1. Chốt hai kỳ
2. Chốt hai kỳ với chốt đa hạt
3. Giao thức cây
4. Thứ tự tem thời gian
5. Hợp lệ
6. Thứ tự tem thời gian đa phiên bản
7. Chốt hai kỳ đa phiên bản

V.13 Chứng tỏ rằng có những lịch trình là có thể dưới giao thức chốt hai kỳ nhưng không thể dưới giao thức tem thời gian và ngược lại.

V.14 Với điều kiện nào tránh deadlock ít đắt giá hơn cho phép deadlock rồi phát hiện?

CHƯƠNG VI
HỆ THỐNG PHỤC HỒI
(Recovery system)

MỤC ĐÍCH

Một hệ thống máy tính, cũng giống như các thiết bị cơ - điện khác, luôn có nguy cơ bị hỏng hóc do nhiều nguyên nhân hư đĩa, mất nguồn, lỗi phần mềm v..v... Điều này dẫn đến hậu quả là sự mất thông tin. Vì vậy, hệ quản trị cơ sở dữ liệu phải có các cơ chế đáp ứng lại nguy cơ hệ thống bị hỏng hóc, nhằm đảm bảo ***tính nguyên tử và tính lâu bền*** của các giao dịch. Chương này trình bày các nguyên lý của một hệ thống phục hồi nhằm khôi phục CSDL đến một trạng thái nhất quán trước khi xảy ra sự cố.

YÊU CẦU

Hiểu rõ các sự cố có thể xảy ra trong đời sống của một cơ sở dữ liệu, các nguyên nhân của sự không nhất quán dữ liệu.

Hiểu các kỹ thuật phục hồi, các ưu nhược điểm của mỗi kỹ thuật.

PHÂN LỚP HỎNG HÓC:

Có nhiều kiểu hỏng hóc có thể xảy đến với hệ thống, mỗi một trong chúng cần được ứng xử một cách riêng biệt. Trong chương này ta chỉ xét các kiểu hỏng hóc sau:

- **Hỏng hóc trong giao dịch**: Có hai loại lỗi làm cho giao dịch bị hỏng hóc: **1. Lỗi luận lý**: Giao dịch không thể tiếp tục thực hiện bình thường được nữa do một số điều kiện bên trong không được thoả. ví dụ như: dữ liệu đầu vào không đúng, không tìm thấy dữ liệu, trào dữ liệu hoặc do việc sử dụng tài nguyên vượt hạn định. **2. Lỗi hệ thống:** Hệ thống rơi vào trạng thái không mong muốn ví dụ như trạng thái deadlock.
- **Hệ thống bị hư hỏng:** Có một phần cứng sai chức năng hoặc có một sai sót trong phần mềm cơ sở dữ liệu hay hệ điều hành.
- **Đĩa bị hư hỏng:** Một khối đĩa bị mất nội dung.

Để hệ thống có thể đề ra được chiến lược phục hồi lỗi phù hợp, trước tiên cần phải xác định các loại hỏng hóc trên các thiết bị lưu trữ dữ liệu. Sau đó, cần xác định những hỏng hóc này ảnh hưởng như thế nào đến nội dung cơ sở dữ liệu. Nhiệm vụ quan trọng sau cùng là đề ra các giải pháp nhằm đảm bảo tính nhất quán của cơ sở dữ liệu và tính nguyên tử của giao dịch mỗi khi hỏng hóc đã phát sinh. Các giải pháp này thường được gọi là các giải thuật phục hồi (recovery algorithms).

Các giải thuật phục hồi gồm có hai phần:

1. Các hành động được thực hiện trong suốt quá trình hoạt động bình thường của giao dịch nhằm đảm bảo có đầy đủ thông tin cho việc phục hồi sau này.
2. Các hành động được thực hiện sau khi lỗi phát sinh. Nhằm khôi phục nội dung của cơ sở dữ liệu trở về một trạng thái trước đó, và trạng thái này thoã mãn được các yêu cầu về tính nhất quán của cơ sở dữ liệu, tính bền và tính nguyên tử của giao dịch .

CẤU TRÚC LƯU TRỮ:

Như đã xét trong chương II, các hạng mục dữ liệu khác nhau của cơ sở dữ liệu có thể được lưu trên nhiều phương tiện lưu trữ khác nhau. Để nắm được cách thức đảm bảo tính nguyên tử và tính lâu bền của một giao dịch, cần phải có cái nhìn sâu hơn về các loại thiết bị lưu trữ dữ liệu và cách thức truy xuất chúng.

CÁC LOẠI LƯU TRỮ:

- **Lưu trữ không ổn định (volatile storage):** Thông tin lưu trong thiết bị lưu trữ không ổn định sẽ bị mất khi hệ thống bị hỏng hóc. Ví dụ của thiết bị lưu trữ không ổn định là: bộ nhớ chính, bộ nhớ cache. Sự truy cập đến các thiết bị lưu trữ không ổn định là cực nhanh. Lý do: một là: do tính chất của bộ nhớ cho phép như vậy; hai là: có thể truy xuất trực tiếp các hạng mục dữ liệu chứa trong nó.
- **Lưu trữ ổn định (nonvolatile storage):** Thông tin lưu trữ trong thiết bị lưu trữ ổn định thường không bị mất khi hệ thống bị sự cố. Tuy nhiên, nguy cơ bản thân thiết bị lưu trữ ổn định bị hỏng vẫn có thể xảy ra. Ví dụ của thiết bị lưu trữ ổn định là: đĩa từ và băng từ. Trong hầu hết các hệ cơ sở dữ liệu, thiết bị lưu trữ ổn định thường được dùng là đĩa từ. Các loại thiết bị lưu trữ ổn định khác được dùng để lưu trữ phòng hờ (back up) dữ liệu.

- **Lưu trữ bền (stable storage):** Theo lý thuyết thì thông tin chứa trong thiết bị lưu trữ bền ***không bao giờ*** bị mất khi hệ thống bị hư hỏng. Tuy nhiên, trong thực tế, ta khó lòng tạo ra được một thiết bị đạt được tính chất lý tưởng như vậy. Chỉ có giải pháp tăng cường độ bền mà thôi.

THỰC THI LƯU TRỮ BỀN:

Tiêu chí để thực hiện việc lưu trữ bền là nhân bản thông tin cần thiết trong một vài phương tiện lưu trữ ổn định khác nhau với các phương thức hỏng hóc độc lập và cập nhật các phiên bản thông tin này một cách có tổ chức, sao cho dù có lỗi xuất hiện trong quá trình chuyển dữ liệu, thông tin vẫn không bị hư hại.

- Các hệ thống RAID đảm bảo rằng việc hỏng hóc của một đĩa không gây sự mất dữ liệu. Dạng thức đơn giản và nhanh nhất của RAID là dùng đĩa gương (mirrored disk). Các dạng thức khác giúp tiết kiệm chi phí, nhưng cái giá phải trả là thời gian đọc ghi chậm hơn.
- Tuy nhiên các hệ thống RAID vẫn không đảm bảo được tính an toàn dữ liệu khi gặp phải tai họa như: cháy nổ, lụt lội. Người ta đề nghị một hệ thống lưu trữ mới an toàn hơn hoạt động theo nguyên tắc sau: Sao lưu dữ liệu sang một vài vị trí địa lý khác nhau thông qua mạng máy tính.

Sau đây là cách thức đảm bảo thông tin lưu trữ không bị lỗi trong quá trình đọc ghi dữ liệu:

Việc chuyển một khối dữ liệu giữa bộ nhớ và đĩa có thể dẫn đến kết quả:

- ***Thành công hoàn toàn***: Thông tin được chuyển đến đích an toàn.
- ***Bị lỗi một phần***: Có lỗi xuất hiện trong quá trình chuyển dữ liệu và khối đích chứa thông tin không đúng.
- ***Bị lỗi hoàn toàn***: Lỗi xuất hiện ngay ở giai đoạn đầu của quá trình truyền dữ liệu. Khối đích giữ nguyên như ban đầu.

Nếu có lỗi xuất hiện trong quá trình truyền dữ liệu, hệ thống phải phát hiện được và thực thi thủ tục phục hồi lỗi. Để làm được như vậy, hệ thống phải duy trì hai khối dữ liệu vật lý cho mỗ khối dữ liệu luận lý. (Trong tình huống dùng hệ thống đĩa gương thì hai khối vật lý này ở cùng một địa điểm, trong tình huống dùng hệ thống sao lưu từ xa, hai khối này ở hai địa điểm khác nhau).

Một thao tác ghi dữ liệu được thực thi như sau:

1. Viết thông tin lên khối vật lý thứ nhất.
2. Khi hành động ghi thứ nhất thành công, tiếp tục ghi phần thông tin trên lên khối vật lý thứ hai.
3. Thao tác ghi được coi là thành công khi thao tác ghi thứ hai thành công.

Trong quá trình phục hồi, từng cặp khối vật lý được kiểm tra:

1. Nếu nội dung của cả hai như nhau và không có lỗi có thể phát hiện, khi đó không cần làm gì thêm.
2. Nếu một trong hai khối có lỗi phát hiện được, khi đó thay thế khối bị lỗi bởi nội dung của khối còn lại.
3. Nếu cả hai khối không có lỗi phát hiện được, nhưng nội dung của chúng khác nhau, thay thế khối thứ nhất bởi khối thứ hai.

Yêu cầu phải so sánh từng cặp khối vật lý một đòi hỏi phải mất nhiều thời gian. Người ta có thể cải thiện tình huống này bằng cách lưu vết những thao tác viết khối trong tiến trình thực thi. Khi phục hồi, chỉ những khối nào thao tác ghi ở trong tiến trình thực thi mới cần được đem so sánh. Giao thức để viết ra một khối đến một site xa tương tự như viết khối trong hệ thống đĩa gương..

TRUY CẬP DỮ LIỆU

Như đã xét trong chương II, hệ cơ sở dữ liệu nằm thường trực trên các thiết bị lưu trữ ổn định (thường là đĩa từ) và thường được phân thành các đơn vị lưu trữ kích thước cố định được gọi là khối (blocks). Khối là đơn vị truyền nhận dữ liệu từ/ra đĩa. Một khối có thể chứa vài hạng mục dữ liệu. Ta giả thiết không có hạng mục dữ liệu nào trải ra trên nhiều hơn một khối.

Các giao dịch nhập (input) thông tin từ đĩa vào bộ nhớ chính và xuất (output) thông tin theo chiều ngược lại. Các thao tác nhập/xuất này được thực hiện theo đơn vị khối. Khối nằm trên đĩa được gọi là khối vật lý (physical block), khối được trữ tạm trong bộ nhớ chính được gọi là khối đệm (buffer block). Vùng bộ nhớ tạm chứa các khối dữ liệu được gọi là vùng đệm đĩa (disk buffer).

Việc di chuyển khối giữa đĩa và bộ nhớ được thực hiện thông qua hai thao tác:

1. **Input(B)** chuyển khối vật lý B vào bộ nhớ chính.
2. **Output(B)** chuyển khối đệm B ra đĩa và thay thế cho khối vật lý tương ứng ở đó.

Hình dưới đây sẽ mô phỏng cho hai thao tác này

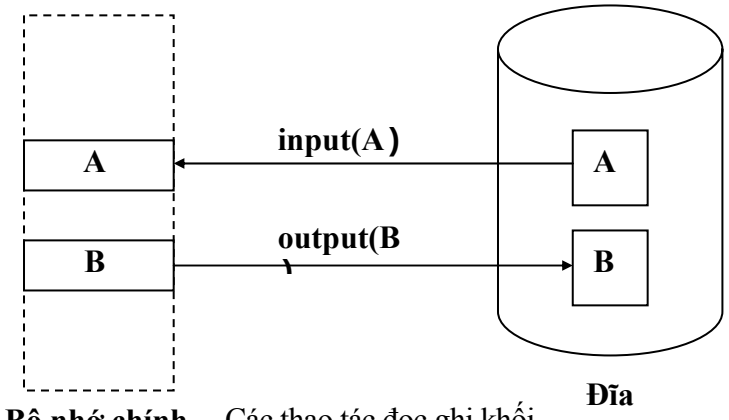


figure VI- 1

Mỗi giao dịch T_i có một vùng làm việc riêng ở đó các bản sao của tất cả các hạng mục dữ liệu được truy xuất và cập nhật được lưu giữ. Vùng làm việc này được tạo ra khi giao dịch khởi động. Nó bị xoá đi khi giao dịch bàn giao (commit) hoặc huỷ bỏ (abort). Mỗi hạng mục dữ liệu ***x*** được trữ trong vùng làm việc của giao dịch T_i sẽ được ký hiệu là x_i. Giao dịch T_i trao đổi với hệ cơ sở dữ liệu bằng cách chuyển dữ liệu đến/ra vùng làm việc của nó sang vùng đệm của hệ thống.

Hai thao tác dùng để chuyển dữ liệu:

1. **read(X)** gán giá trị của hạng mục dữ liệu X cho biến cục bộ x_i. Thao tác này được thực hiện như sau:
   a. Nếu khối B_X chứa X không có trong bộ nhớ chính thì thực hiện thao tác **input(B_X).**
   b. Gán cho x_i giá trị của X trong khối đệm.
2. **write(X)** gán giá trị của biến cục bộ x_i cho hạng mục dữ liệu X trong khối đệm. Thao tác này được thực hiện như sau:
   a. Nếu khối B_X chứa X không có trong bộ nhớ thì thực hiện thao tác **input(B_X).**
   b. Gán giá trị của x_i cho X trong vùng đệm B_X.

Chú ý rằng cả hai thao tác đều có thể đòi hỏi chuyển một khối từ đĩa vào bộ nhớ chính nhưng không yêu cầu chuyển một khối từ bộ nhớ chính ra đĩa.

Đôi khi một khối đệm bị ghi bắt buộc ra đĩa do bộ quản lý vùng đệm cần không gian bộ nhớ cho các mục đích khác hoặc do hệ cơ sở dữ liệu muốn phản ánh những thay đổi trong khối dữ

liệu B trên đĩa. Khi hệ cơ sở dữ liệu thực hiện thao tác **Output(B)** ta nói nó đã xuất bắt buộc khối đệm B ra đĩa.

Khi một giao dịch cần truy xuất hạng mục dữ liệu X lần đầu, nó phải thực hiện **Read(X)**. Khi đó tất cả các cập nhật đối với X được thực hiện trên x_i. Sau khi giao dịch truy xuất X lần cuối, nó thực hiện **Write(X)** để ghi lại sự thay đổi của X trong CSDL.

Không nhất thiết phải thực hiện thao tác **Output(B_X)** ngay sau khi thao tác **write(X)** hoàn thành. Lý do là: khối đệm B_X có thể còn chứa các hạng mục dữ liệu khác đang được truy xuất. Nếu hệ thống bị hư hỏng ngay sau khi thao tác **write(X)** hoàn thành, nhưng trước khi thực hiện thao tác **Output(B_X)**, giá trị mới của X sẽ không bao giờ được ghi ra đĩa, do đó, nó bị mất!

PHỤC HỒI VÀ TÍNH NGUYÊN TỬ:

Trở lại với ví dụ đơn giản về hệ thống ngân hàng: Giao dịch T_i thực hiện việc chuyển $50 từ tài khoản A sang tài khoản B. Giả sử giá trị ban đầu của các tài khoản A và B là $1000 và $2000. Giả sử hệ thống bị hư hỏng trong khi T_i đang thực thi: sau khi thao tác **output(B_A)** được thực hiện và trước khi thực hiện thao tác **output(B_B)** (B_A và B_B là hai khối đệm chứa hai hạng mục A và B). Người ta có thể thực hiện một trong hai giải pháp phục hồi sau:

1. Thực hiện lại T_i. Thủ tục này sẽ dẫn đến kết quả: giá trị của A là $900 thay vì phải là $950. Do đó, hệ thống ở trong trạng thái không nhất quán.
2. Không thực hiện lại T_i . Kết quả: giá trị của A và B tương ứng sẽ là $950 và $2000. Hệ thống cũng trong trạng thái không nhất quán.

Vấn đề phát sinh ở chỗ: T_i thực hiện nhiều thao tác sửa đổi nội dung cơ sở dữ liệu, do đó cần nhiều thao tác xuất dữ liệu ra đĩa, nhưng lỗi phát sinh không cho phép tất cả các thao tác xuất dữ liệu hoàn thành.

Giải pháp nhằm đạt được tính nguyên tử là: trước khi thực hiện các thao tác sửa đổi cơ sở dữ liệu, cần ghi ra các thiết bị lưu trữ bền những thông tin mô tả các sửa đổi này. Cụ thể của giải pháp trên sẽ được trình bày trong các phần V.4, V.5 và V.6.

PHỤC HỒI DỰA TRÊN SỔ GHI LỘ TRÌNH (Log-based recovery)

Một cấu trúc thường được dùng để ghi lại những thay đổi trên cơ sở dữ liệu là sổ ghi lộ trình (log). Log là một dãy các mẩu tin lộ trình (log records). Một thao tác cập nhật trên cơ sở dữ liệu sẽ được ghi nhận bằng một log record. Một log record kiểu mẫu chứa các trường sau:

- **Định danh giao dịch (transaction identifier)**: định danh duy nhất của giao dịch thực hiện hoạt động **write**
- **Định danh hạng mục dữ liệu (Data-item identifier)**: định danh duy nhất của hạng mục dữ liệu được viết (thường là vị trí của hạng mục dữ liệu trên đĩa)
- **Giá trị cũ (Old value)**: giá trị của hạng mục dữ liệu trước khi viết
- **Giá trị mới (New value)**: giá trị hạng mục dữ liệu sẽ có sau khi viết

Có một vài log record đặc biệt mang các ý nghĩa riêng. Bảng sau đây chỉ ra một số loại log record và ý nghĩa của chúng:

LOẠI LOG RECORD	Ý NGHĨA
< T_i **start** >	Giao dịch T_i đã khởi động.
< T_i, X_j, V_1, V_2 >	Giao dịch T_i đã thực hiện thao tác ghi trên hạng mục dữ liệu X_j, X_j có giá trị V_1 trước khi ghi và nhận giá trị V_2 sau khi ghi.
< T_i **commit** >	Giao dịch T_i đã bàn giao.
< T_i **abort** >	Giao dịch Ti đã huỷ bỏ.

Mỗi khi một giao dịch thực hiện một thao tác ghi, trước tiên phải tạo ra một log record cho thao tác ghi đó (trong log file), trước khi giao dịch thay đổi cơ sở dữ liệu. Như vậy, hệ thống có cơ sở để huỷ bỏ (undo) một thay đổi đã được làm trên cơ sở dữ liệu bằng cách sử dụng trường **Old-value** trong log record.

log phải được lưu trong những thiết bị lưu trữ bền. Mỗi một log record mới ngầm định sẽ được thêm vào cuối tập tin log.

CẬP NHẬT TRÌ HOÃN CƠ SỞ DỮ LIỆU (Deferred Database Modification):

Kỹ thuật cập nhật trì hoãn đảm bảo tính nguyên tử của giao dịch bằng cách ghi lại tất cả những sửa đổi cơ sở dữ liệu vào sổ ghi lộ trình (log), nhưng trì hoãn sự thực hiện tất cả các thao tác viết dữ liệu ra đĩa của giao dịch cho đến khi giao dịch bàn giao một phần (partially commits). Nhắc lại rằng: một giao dịch được gọi là bàn giao một phần khi hành động cuối cùng của nó được thực hiện xong. Kỹ thuật cập nhật trì hoãn được trình bày trong phần này giả thiết rằng các giao dịch được thực hiện một cách tuần tự.

Khi giao dịch bàn giao một phần, thông tin trên log kết hợp với giao dịch được sử dụng trong việc viết trì hoãn. Nếu hệ thống có sự cố trước khi giao dịch hoàn thành việc thực hiện của nó hoặc giao dịch bị bỏ dở khi đó thông tin trên log bị bỏ lơ.

Sự thực thi của một giao dịch được tiến triển như sau:

- Trước khi giao dịch T_i bắt đầu thực hiện, một mẫu tin < T_i **start** > được ghi ra sổ lộ trình.
- Trước khi T_i thực hiện thao tác **write(X),** một mẫu tin < T_i, X, V_2 > được ghi ra sổ lộ trình.
- Cuối cùng, khi giao dịch T_i bàn giao một phần, mẫu tin < T_i **commit** > được ghi ra sổ lộ trình.

Khi giao dịch bàn giao một phần, các mẫu tin trong sổ lộ trình kết hợp với giao dịch sẽ được sử dụng để thực hiện việc ghi trì hoãn các hạng mục dữ liệu ra đĩa. Nhà thiết kế phải đảm bảo rằng, trước khi hoạt động ghi hạng mục dữ liệu diễn ra, các mẫu tin log đã được ghi thành công ra các thiết bị lưu trữ bền. Ngoài ra cũng cần để ý: kỹ thuật cập nhật trì hoãn chỉ cần ghi lại giá trị mới của hạng mục dữ liệu (V_2) mà thôi.

Để minh hoạ, ta sử dụng ví dụ hệ thống ngân hàng đơn giản. Gọi T_0 là giao dịch có nhiệm vụ chuyển $50 từ tài khoản A sang tài khoản B, T_1 là giao dịch có nhiệm vụ rút $100 từ tài khoản C. Giả sử giá trị ban đầu của các tài khoản A, B, C là $1000, $2000 và $700. Hành động của T_0 và T_1 được mô tả như sau:

T_0	T_1
read(A)	Read(C)
A:=A-50	C:=C-100
write(A)	write(C)
read(B)	
B:=B+50	
write(B)	

Giả thiết các giao dịch được thực hiện tuần tự: T_0 rồi tới T_1. Một phần của sổ lộ trình ghi lại những thông tin liên quan đến hoạt động của hai giao dịch trên được cho trong bảng dưới đây:

<T_0 start>
<T_0 ,A, 950>
<T_0 ,B, 2050>

<T_0 commit>
<T_1 start>
<T_1 ,C, 600>
<T_1 commit>

figure VI- 2

Sau khi có sự cố xảy ra, hệ thống phục hồi sẽ tham khảo sổ lộ trình để chọn ra những giao dịch nào cần được làm lại (redo). Giao dịch T_i cần được làm lại khi và chỉ khi sổ nhật ký có chứa cả hai mẫu tin <T_i start> và <T_i commit>.

Thủ tục làm lại giao dịch T_i như sau:

- **redo(T_i)** đặt giá trị mới cho tất cả các hạng mục dữ liệu được cập nhật bởi giao dịch T_i. Các giá trị mới sẽ được tìm thấy trong sổ lộ trình (log).

Hoạt động redo phải đồng hiệu lực (idempotent) có nghĩa là việc thực hiện nó nhiều lần tương đương với việc thực hiện nó một lần.

Trở lại ví dụ vừa nêu, ta có bảng mô tả trạng thái của sổ ghi lộ trình và cơ sở dữ liệu như sau:

LOG	CƠ SỞ DỮ LIỆU
<T_0 start>	
<T_0 ,A, 950>	
< T_0 ,B, 2050>	
<T_0 commit>	
	A=950
	B=2050
<T_1 start>	
<T_1 ,C, 600>	
<T_1 commit>	
	C=600

figure VI- 3

Sau đây là một số tình huống mô phỏng:

1. Giả sử lỗi hệ thống xảy ra sau khi mẫu tin log cho hành động **write(B)** của giao dịch T_0 vừa được ghi ra thiết bị lưu trữ bền. Khi hệ thống khởi động trở lại, sẽ không có hành động "thực hiện lại giao dịch" nào cần phải làm, do không có mẫu tin ghi **commit** nào xuất hiện trong sổ lộ trình. Nghĩa là giá trị của A,B và C vẫn giữ nguyên là $1000, $2000 và $700.
2. Giả sử lỗi hệ thống xảy ra sau khi mẫu tin log cho hành động **write(C)** của giao dịch T_1 vừa được ghi ra thiết bị lưu trữ bền. Khi hệ thống hoạt động trở lại, thủ tục **redo(T_0)** sẽ được thực hiện do có sự xuất hiện của mẫu tin **<T_0 commit>** trong sổ lộ trình. Sau khi thủ tục này được thực thi, giá trị của A và B sẽ là $950 và $2050.

CẬP NHẬT TỨC THỜI CƠ SỞ DỮ LIỆU (Immediate Database Modification):

Kỹ thuật cập nhật tức thời cho phép các thao tác sửa đổi cơ sở dữ liệu có quyền xuất dữ liệu tức thời ra đĩa trong khi giao dịch vẫn còn ở trong trạng thái hoạt động (active state). Hành động thay đổi nội dung dữ liệu tức thời của các giao dịch đang hoạt động được gọi là "những thay đổi chưa được bàn giao" (uncommitted modifications).

Sự thực thi của một giao dịch được tiến hành như sau:

- Trước khi giao dịch T_i bắt đầu sự thực hiện, một mẫu tin < T_i **start** > được ghi ra sổ lộ trình.

- Trước khi T_i thực hiện thao tác **write(X),** một mẫu tin < T_i, X, V_1, V_2 > được ghi ra sổ lộ trình.
- Cuối cùng, khi giao dịch T_i bàn giao một phần, mẫu tin < T_i **commit** > được ghi ra sổ lộ trình.

Cần phải đảm bảo rằng, trước khi hoạt động ghi hạng mục dữ liệu diễn ra, các mẫu tin log đã được ghi thành công ra các thiết bị lưu trữ bền. Ngoài ra, cũng cần chú ý là mẫu tin log cho hành động **write(X)** của giao dịch T_i, tức là mẫu tin <T_i, X, V_1, V_2> có chứa cả hai giá trị mới (V_2) và cũ (V_1) của hạng mục dữ liệu X.

Trở lại với ví dụ trong phần V.4.1, ta có một phần của sổ lộ trình liên quan đến các hoạt động của T_0 và T_1 như sau:

<T_0 start>
<T_0 , A, 1000, 950>
<T_0 , B, 2000, 2050>
<T_0 commit>
<T_1 start>
<T_1 , C, 700, 600>
<T_1 commit>

figure VI- 4

Bảng mô tả trạng thái của sổ ghi lộ trình và cơ sở dữ liệu như sau:

LOG	CƠ SỞ DỮ LIỆU
<T_0 start>	
<T_0 , A, 1000, 950>	
< T_0 , B, 2000, 2050>	
	A=950
	B=2050
<T_0 commit>	
<T_1 start>	
<T_1 , C, 700, 600>	
	C=600
<T_1 commit>	

figure VI- 5

Kỹ thuật cập nhật tức thời sử dụng hai thủ tục khôi phục sau lỗi:

- **undo(T_i)** đặt lại giá trị cũ cho tất cả các hạng mục dữ liệu được cập nhật bởi giao dịch T_i. Các giá trị cũ sẽ được tìm thấy trong sổ lộ trình (log).
- **redo(T_i)** đặt giá trị mới cho tất cả các hạng mục dữ liệu được cập nhật bởi giao dịch T_i. Các giá trị mới sẽ được tìm thấy trong sổ lộ trình (log).

Sau khi lỗi xuất hiện, hệ thống phục hồi tham khảo sổ ghi để quyết định những giao dịch nào cần được làm lại (redo) và những giao dịch nào cần được huỷ bỏ (undo).

- Giao dịch T_i cần được huỷ bỏ khi sổ ghi chứa mẫu tin <T_i start> nhưng không có mẫu tin <T_i commit>.
- Giao dịch T_i cần được làm lại khi sổ ghi có chứa cả mẫu tin <T_i start> lẫn mẫu tin <T_i commit>.

Sau đây là một số tình huống mô phỏng:

1. Giả sử lỗi hệ thống xảy ra sau khi mẫu tin log cho hành động **write(B)** của giao dịch T_0 vừa được ghi ra thiết bị lưu trữ bền. Khi hệ thống khởi động trở lại, nó sẽ tìm thấy mẫu tin <T_0 start> trong sổ ghi, nhưng không có mẫu tin <T_0 commit> tương ứng. Do đó giao dịch T_0 cần phải được huỷ bỏ. Nghĩa là thủ tục **undo(T_0)** sẽ được gọi và giá trị của A,B và C vẫn giữ nguyên là \$1000, \$2000 và \$700.
2. Giả sử lỗi hệ thống xảy ra sau khi mẫu tin log cho hành động **write(C)** của giao dịch T_1 vừa được ghi ra thiết bị lưu trữ bền. Khi hệ thống hoạt động trở lại, hai thủ tục **redo(T_0)** và **undo(T_1)** sẽ được thực hiện. Do có sự xuất hiện của các mẫu tin **<T_0 start>**, **<T_0 commit>**, **<T_1 start>** trong sổ lộ trình. Sau khi hai thủ tục này được thực thi, giá trị của A, B, C sẽ là \$950,ì \$2050 và \$700.

ĐIỂM KIỂM SOÁT (Checkpoint):

Khi lỗi hệ thống xuất hiện, hệ thống phục hồi phải tham khảo sổ ghi lộ trình để quyết định những giao dịch nào cần được làm lại và những giao dịch nào cần được huỷ bỏ. Theo nguyên lý thì cần phải tìm kiếm toàn bộ nội dung của sổ ghi để có được quyết định trên.

Hướng tiếp cận trên sẽ gặp phải hai khó khăn lớn:

1. Quá trình tìm kiếm mất nhiều thời gian.
2. Theo các giải thuật vừa nêu, hầu hết các giao dịch cần được làm lại đã ghi những dữ liệu được cập nhật ra cơ sở dữ liệu rồi. Việc làm lại chúng tuy không có hại gì, nhưng lại làm cho tiến trình khôi phục trở nên lâu hơn.

Công cụ "điểm kiểm soát" (checkpoint) được sử dụng để cải thiện hiệu năng của quá trình khôi phục. Trong quá trình hoạt động của mình, hệ thống sẽ duy trì một sổ ghi lộ trình bằng cách sử dụng một trong hai kỹ thuật được giới thiệu trong phần V.4.1 và V.4.2. Ngoài ra, hệ thống còn phải thực hiện một cách chu kỳ các hành động đặt điểm kiểm soát. Hành động này đòi hỏi một dãy các thao tác sau:

1. Xuất ra lưu trữ bền tất cả các mẫu tin ghi nhận lộ trình (log record) đang nằm trong bộ nhớ chính.
2. Xuất ra đĩa tất cả những khối đệm đã được cập nhật.
3. Xuất ra thiết bị lưu trữ bền một log-record <**checkpoint**>

Các giao dịch sẽ không được phép thực hiện bất kỳ thao tác cập nhật dữ liệu nào (ví dụ như ghi các khối đệm, ghi các mẫu tin log) khi hành động đặt điểm kiểm soát đang được thực hiện.

Sự hiện diện của điểm kiểm soát trong sổ ghi cho phép hệ thống tổ chức quá trình phục hồi tốt hơn. Xét một giao dịch T_i đã bàn giao (commit) trước một điểm kiểm soát. Ta có mẫu tin **<T_i commit>** xuất hiện trước mẫu tin **<checkpoint>.** Có nghĩa là tất cả các thay đổi mà T_i đã làm đối với cơ sở dữ liệu phải được thực hiện trước khi người ta đặt điểm kiểm soát trên. Vì vậy, trong giai đoạn phục hồi sau lỗi, người ta không cần phải làm lại (**redo)** giao dịch T_i.

Dựa trên điểm cải tiến này, ta cải tiến lại các kỹ thuật đã được trình bày trong phần V.4.1 và V.4.2 như sau:

1. Sau khi lỗi hệ thống xuất hiện, hệ thống phục hồi sẽ kiểm tra lại sổ lộ trình (log) để tìm ra giao dịch T_i thoả điều kiện: đó là giao dịch gần đây nhất được khởi động trước điểm kiểm soát gần đây nhất. Qui trình tìm T_i như sau: dò ngược trong sổ ghi lộ trình cho đến khi tìm thấy mẫu tin <**checkpoint**> đầu tiên. Từ điểm kiểm soát này, lại tiếp tục dò ngược trong sổ ghi cho đến khi tìm thấy mẫu tin <**T_i start**> đầu tiên. Mẫu tin này chỉ ra giao dịch T_i .
2. Khi đã xác định được giao dịch T_i rồi, các thủ tục **undo** và **redo** chỉ được áp dụng cho giao dịch T_i và các giao dịch diễn ra sau T_i. Chúng ta ký hiệu tập những giao dịch vừa nói là **T**.
3. Với kỹ thuật "Cập nhật tức thời cơ sở dữ liệu", tiến trình phục hồi như sau:

- Với mọi giao dịch $T_k \in \mathbf{T}$ mà không có mẫu tin <**T_k commit**> trong sổ ghi lộ trình, thực thi **undo(T_k)**.
- Với mọi giao dịch $T_k \in T$ mà có mẫu tin <**T_k commit**> trong sổ ghi lộ trình, thực thi **redo(T_k)**.

4. Không cần thực thi thao tác **undo** khi sử dụng kỹ thuật "Cập nhật có trì hoãn cơ sở dữ liệu".

PHÂN TRANG BÓNG (Shadow Paging):

Kỹ thuật "Phân trang bóng" cũng là kỹ thuật cho phép phục hồi sau lỗi, nhưng ý tưởng thực hiện khác với các kỹ thuật dựa trên sổ ghi lộ trình vừa trình bày ở phần trên.
Sau đây là một số khái niệm cần được giải trình:

- ***Trang (page) là gì***? Như đã trình bày ở các phần trước, cơ sở dữ liệu được lưu vào thiết bị lưu trữ không phai thành nhiều khối có kích thước cố định. Người ta gọi những khối này là trang (page).
- ***Bảng trang và ý nghĩa của nó***: Khái niệm trang đã nói được mượn từ lý thuyết về Hệ điều hành. Cách quản lý trang cũng được thừa kế từ đó. Giả sử rằng cơ sở dữ liệu được phân thành n trang và sự phân bố trên đĩa của chúng có thể không theo một thứ tự cụ thể nào cả. Tuy nhiên, phải có cách để tìm ra nhanh và đúng trang thứ i của cơ sở dữ liệu ($1 \le i \le n$). Người ta dùng bảng trang (được mô phỏng như trong hình 5.2) cho mục đích này. Bảng trang có n đầu vào (entry). Mỗi đầu vào ứng với một trang. Một đầu vào chứa một con trỏ, trỏ đến một trang trên đĩa. Đầu vào đầu tiên chỉ đến trang đầu tiên của cơ sở dữ liệu, đầu vào thứ hai chỉ đến trang thứ hai ...

Ý tưởng then chốt của kỹ thuật "Phân trang bóng" là người ta sẽ duy trì hai bảng trang trong suốt chu kỳ sống của giao dịch, một bảng trang gọi là "*bảng trang hiện hành"* (current page table), bảng trang còn lại gọi là "*bảng trang bóng"* (shadow page table). Khi giao dịch khởi động, hai bảng trang này giống nhau. Bảng trang bóng sẽ không thay đổi suốt quá trình hoạt động của giao dịch. Bảng trang hiện hành sẽ bị thay đổi mỗi khi giao dịch thực hiện tác vụ **write.** Tất cả các tác vụ **input** và **output** đều sử dụng bảng trang hiện hành để định vị các trang trong đĩa. Điểm quan trọng khác là nên lưu bảng trang bóng vào thiết bị lưu trữ bền.

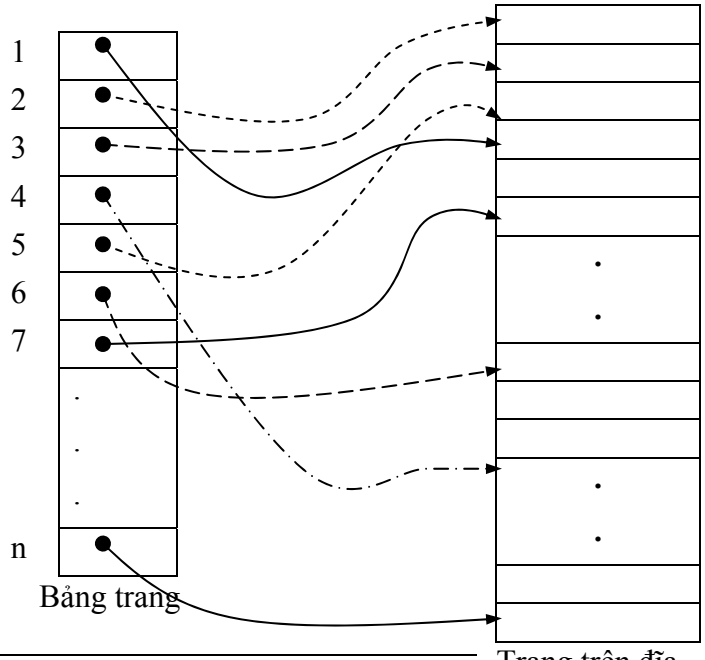


figure VI- 6

Giả sử giao dịch thực hiện tác vụ **write(X)** và hạng mục dữ liệu X được chứa trong trang thứ ***i.*** Tác vụ **write** được thực thi như sau:

1. Nếu trang thứ ***i*** chưa có trong bộ nhớ chính, thực hiện **input(X).**
2. Nếu đây là lệnh ghi được thực hiện ***lần đầu tiên*** trên trang thứ ***i*** bởi giao dịch, sửa đổi bảng trang hiện hành như sau:
   a. Tìm một trang chưa được dùng trên đĩa.
   b. Xoá trang vừa được tìm xong ở bước 2.a khỏi danh sách các khung trang tự do.
   c. Sửa lại bảng trang hiện hành sao cho đầu vào thứ ***i*** trỏ đến trang mới vừa tìm được trong bước 2.a.
3. Gán giá trị x_i cho X trong trang đệm (buffer page).

Để bàn giao một giao dịch, cần làm các bước sau:

1. Đảm bảo rằng tất cả các trang đệm trong bộ nhớ chính đã được giao dịch sửa đổi phải được xuất ra đĩa.
2. Xuất bảng trang hiện hành ra đĩa. chú ý là không được viết đè lên trang bóng
3. Xuất địa chỉ đĩa của bảng trang hiện hành ra vị trí cố định trong thiết bị lưu trữ bền. Vị trí này chính là nơi chứa địa chỉ của bảng trang bóng. Hành động này sẽ ghi đè lên địa chỉ của bảng trang bóng cũ. Như vậy, bảng trang hiện hành sẽ trở thành bảng trang bóng và giao dịch được bàn giao.

Nếu sự cố xảy ra trước khi hoàn thành bước thứ 3, hệ thống sẽ trở về trạng thái trước khi giao dịch được thực hiện. Nếu sự cố xảy ra sau khi bước thứ 3 hoàn thành, hiệu quả của giao dịch được bảo tồn; không cần thực hiện thao tác **redo** nào cả. Ví dụ trong hình 5.3 dưới đây mô phỏng lại trạng thái của các bảng trang hiện hành và bảng trang bóng khi giao dịch thực hiện thao tác ghi lên trang thứ tư của cơ sở dữ liệu có 10 trang.

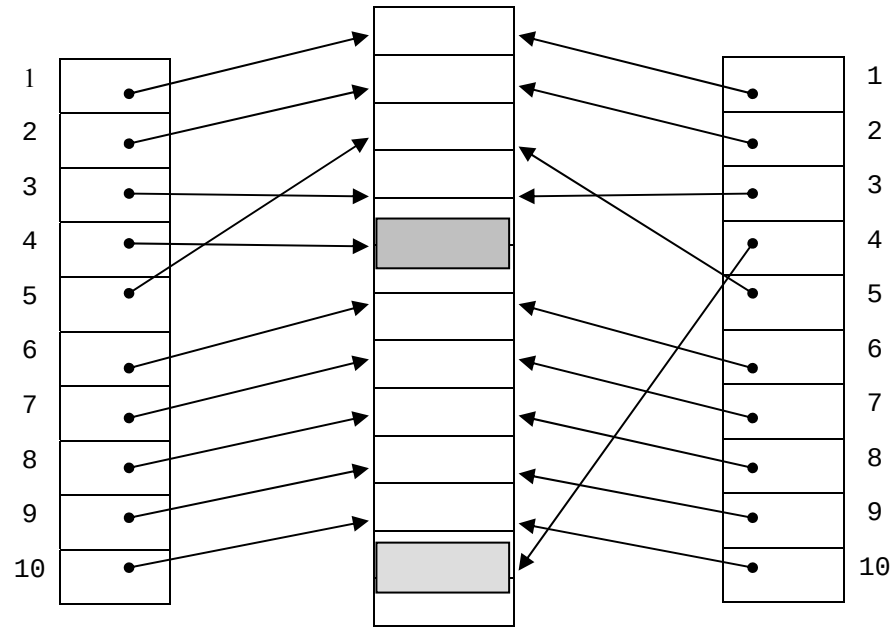


figure VI- 7 Ví dụ về bảng trang bóng và bảng trang hiện hành

Kỹ thuật phân trang bóng có một số điểm lợi hơn so với các kỹ thuật dựa trên sổ ghi:

1. Không mất thời gian ghi ra các log record.
2. Khôi phục sau sự cố nhanh hơn, do không cần các thao tác **undo** hoặc **redo**.

Tuy nhiên kỹ thuật phân trang bóng lại có nhiều nhược điểm:

- **Tổng phí bàn giao.** Xuất nhiều khối ra đĩa: các khối dữ liệu hiện tại, bảng trang hiện hành, địa chỉ của bảng trang hiện hành. Trong kỹ thuật dựa vào sổ ghi, chỉ cần xuất ra các log record, mà thông thường, các log record này vừa đủ chứa trong một khối.
- **Sự phân mảnh dữ liệu**. Trong chương II có trình bày chiến lược gom cụm vật lý các trang dữ liệu có liên quan với nhau. Sự gom cụm này cho phép việc vận chuyển dữ liệu nhanh hơn. Kỹ thuật phân trang bóng lại đổi vị trí của trang khi trang này bị sửa đổi. Điều này dẫn đến tính gom cụm dữ liệu không còn, hoặc phải dùng các giải pháp gom cụm lại rất mất thời gian.
- **Phải thu nhặt rác**. Mỗi khi giao dịch bàn giao, các trang chứa giá trị dữ liệu cũ đã bị sửa đổi bởi giao dịch sẽ trở thành không truy xuất được. Vì chúng không thuộc danh sách các trang tự do nhưng cũng không chứa dữ liệu hữu dụng. Ta gọi chúng là "rác". Cần thiết phải định kỳ tìm kiếm và thêm các trang rác vào trong danh sách các trang tự do. Hành động này được gọi là "thu nhặt rác".
- Ngoài ra, kỹ thuật phân trang bóng sẽ gặp nhiều khó khăn hơn kỹ thuật dựa vào sổ ghi khi cần được tinh chỉnh để đáp ứng cho yêu cầu phục vụ song song cho nhiều giao dịch. Vì những lý do trên, kỹ thuật phân trang bóng không được sử dụng rộng rãi lắm.

PHỤC HỒI VỚI CÁC GIAO DỊCH CẠNH TRANH

Cho dến bây giờ, ta chỉ xét các kỹ thuật phục hồi áp dụng cho các giao dịch được thực thi tuần tự. Bây giờ chúng ta sẽ tìm cách cải tiến kỹ thuật dựa vào sổ ghi nhằm đáp ứng yêu cầu phục vụ đồng thời cho nhiều giao dịch cạnh tranh. Ý tưởng thực hiện là: Không quan tâm đến số lượng các giao dịch cạnh tranh, hệ thống vẫn sử dụng một vùng đệm đĩa và một sổ ghi lộ trình. Các khối đệm được chia sẻ bởi tất cả các giao dịch. Chúng ta sẽ cho phép việc cập nhật tức thời cơ sở dữ liệu và cho phép một khối đệm có nhiều hạng mục dữ liệu được cập nhật bởi một hoặc nhiều giao dịch.

TRAO ĐỔI VỚI ĐIỀU KHIỂN CẠNH TRANH

Cơ chế phục hồi phụ thuộc rất nhiều vào cơ chế điều khiển cạnh tranh được sử dụng. Để cuộn lại một giao dịch thất bại (failed transaction), người ta phải huỷ bỏ (undo) các cập nhật được thực hiện bởi giao dịch. Giả sử giao dịch T_0 phải bị cuộn lại và một hạng mục dữ liệu Q đã bị T_0 thay đổi giá trị và cần phải được đặt lại giá trị cũ. Bằng cách sử dụng kỹ thuật dựa vào sổ ghi lộ trình, ta trả lại giá trị cũ cho Q bằng cách sử dụng một log record. Giả thiết lại có giao dịch thứ hai T_1 cũng vừa cập nhật Q xong, trước khi T_0 bị cuộn lại. Như vậy, sự cập nhật được thực hiện bởi T_1 sẽ bị mất đi nếu T_0 bị cuộn lại.

Biện pháp khắc phục là: nếu một giao dịch T đã cập nhật một hạng mục dữ liệu Q, thì không một giao dịch nào khác có quyền cập nhật lên hạng mục dữ liệu đó trong khi T chưa bàn giao hoặc chưa bị cuộn lại. Chúng ta có thể đảm bảo yêu cầu trên được thoả bằng cách sử dụng kỹ thuật "chốt hai kỳ nghiêm ngặt" (strict two-phase locking).

CUỘN LẠI GIAO DỊCH:

Phương pháp để cuộn lại (rollback) một giao dịch T_i, sử dụng sổ ghi, trong môi trường cạnh tranh như sau:

1. Dò ngược sổ ghi lộ trình để tìm ra các log record có dạng <T_i, Xj, V_1, V_2>.

2. Hạng mục dữ liệu Xj sẽ được trả lại giá trị cũ V_1.
3. Việc dò tìm kết thúc khi tìm thấy mẫu tin <T_i start>.

Việc dò ngược sổ ghi lộ ¹eó một ý nghĩa rất quan trọng, do một giao dịch có thể đã cập nhật một hạng mục dữ liệu nhiều hơn một lần. Một ví dụ: Xét một cặp log records như sau:

<T_i, A, 10, 20>
<T_i, A, 20, 30>

Cặp mẫu tin này thể hiện hai hành động cập nhật hạng mục dữ liệu A của giao dịch T_i. Nếu dò ngược sổ ghi lộ trình, A sẽ được trả về giá trị đúng là 10. Ngược lại, A sẽ nhận giá trị sai là 20.

Nếu kỹ thuật strict two-phase locking được sử dụng để điều khiển cạnh tranh, thì việc trả về giá trị cũ cho một hạng mục dữ liệu sẽ không xoá đi những tác động của các giao dịch khác lên hạng mục dữ liệu này.

CÁC ĐIỂM KIỂM SOÁT

Ở phần V.4.3, người ta đã sử dụng điểm kiểm soát (checkpoint) để làm giảm số lượng các log record mà hệ thống phục hồi phải dò tìm trong sổ ghi trong giai đoạn phục hồi sau lỗi. Nhưng, do đã giả thiết là không có cạnh tranh nên giải pháp V.4.3 chỉ xét đến những giao dịch sau trong quá trình khôi phục lỗi:

- Những giao dịch được khởi động sau điểm kiểm soát gần đây nhất.
- Một giao dịch (nếu có) đang trong trạng thái hoạt động (active) tại thời điểm người ta đặt điểm kiểm soát gần đây nhất.

Tình huống càng phức tạp khi các giao dịch được thực thi cạnh tranh. Có nghĩa là tại thời điểm đặt điểm kiểm soát, có thể có nhiều giao dịch đang ở trong trạng thái hoạt động.

Trong một hệ thống xử lý các giao dịch cạnh tranh, ta yêu cầu rằng: một mẫu tin ghi dấu kiểm soát (checkpoint log record) phải có dạng như sau:

<**checkpoint L**>

Trong đó L là danh sách các giao dịch đang hoạt động tại thời điểm đặt điểm kiểm soát.

Một lần nữa, ta qui ước rằng: khi hành động đặt điểm kiểm soát đang diễn ra, các giao dịch không được phép thực hiện bất kỳ thao tác cập nhật dữ liệu nào cả trên các khối đệm lẫn trên sổ ghi lộ trình.

Tuy nhiên, qui ước trên lại gây phiền toái, bởi vì các giao dịch phải ngừng hoạt động khi đặt điểm kiểm soát. Một kỹ thuật nâng cao giải quyết điểm phiền toái này là "Điểm kiểm soát mờ" (fuzzy checkpoint).

PHỤC HỒI KHỞI ĐỘNG LẠI (Restart Recovery)

Khi hệ thống phục hồi sau lỗi, nó tạo ra hai danh sách: *undo-list* bao gồm các giao dịch cần phải huỷ bỏ và *redo-list* bao gồm danh sách các giao dịch cần được làm lại.

Qui trình tạo lập hai danh sách *redo-list, undo-list* được thực hiện như sau:

1. Đầu tiên, chúng sẽ rỗng.
2. Dò ngược sổ ghi lộ trình, kiểm tra các mẫu tin cho đến khi tìm được mẫu tin <**checkpoint**> đầu tiên:
   - **a.** Với mỗi mẫu tin được tìm thấy theo dạng <**T_i commit**>, ta thêm T_i vào trong *redo-list.*
   - **b.** Với mỗi mẫu tin được tìm thấy theo dạng <**T_i start**>, nếu T_i không thuộc *redo-list* thì thêm T_i vào trong *undo-list.*
   - **c.** Khi tất cả các log record đã được xem xét, ta kiểm tra danh sách **L** trong mẫu tin <**checkpoint L**>. Với mọi giao dịch T_i trong **L**, nếu T_i không thuộc *redo-list* thì thêm T_i vào *undo-list*.

Khi hai danh sách *redo-list, undo-list* được thiết lập xong, tiến trình phục hồi được tiến hành như sau:

1. Dò ngược lại sổ ghi và thực hiện thủ tục **undo** đối với mỗi log record thuộc về giao dịch T_i trong *undo-list.* Các log record của các giao dịch nằm trong danh sách *redo-list* sẽ bị bỏ qua trong giai đoạn này. Việc dò ngược sẽ ngưng khi mẫu tin <**T_i start**> được tìm thấy cho mọi giao dịch T_i thuộc danh sách *undo-list.*
2. Định vị mẫu tin <**checkpoint L**> gần đây nhất trong log.
3. Dò sổ ghi theo chiều xuôi bắt đầu từ mẫu tin <**checkpoint L**> gần đây nhất và thực hiện thủ tục **redo** đối với mỗi log record thuộc về giao dịch T_i nằm trong danh sách *redo-list*. Trong giai đoạn này, bỏ qua các log record của các giao dịch thuộc danh sách *undo-list*.

Việc xử lý ngược ở bước 1 là rất quan trọng, nhằm đảm bảo kết quả trả về của cơ sở dữ liệu là đúng.

Sau khi tất cả các giao dịch trong danh sách *undo-list* bị huỷ bỏ, tất cả các giao dịch trong danh sách *redo-list* sẽ được làm lại. Sau khi tiến trình phục hồi thành công, xử lý giao dịch được tiếp tục.

Việc thực hiện huỷ bỏ các giao dịch trong *undo-list* trước khi làm lại các giao dịch trong *redo-list* có ý nghĩa rất quan trọng. Nếu làm ngược lại, vấn đề sau sẽ phát sinh: Giả sử hạng mục dữ liệu A có giá trị khởi đầu là 10. Giao dịch T_i đổi A thành 20 sau đó T_i bị huỷ bỏ. Sau đó, một giao dịch khác T_j cập nhật A thành 30. Đến đây thì hệ thống bị lỗi ngừng hoạt động. Hiện trạng của sổ ghi tại thời điểm hệ thống bị lỗi như sau:

<T_i, A, 10, 20>
<T_j, A, 10, 30>
<T_j commit>

Nếu thực hiện redo trước, A sẽ được đặt giá trị 30. Sau đó thực hiện undo, A sẽ được đặt giá trị 10, mà giá trị này sai. Giá trị cuối cùng của A phải là 30.

ĐIỂM KIỂM SOÁT MỜ (fuzzy checkpoint):

Kỹ thuật fuzzy checkpoint cho phép các giao dịch được cập nhật dữ liệu trên các khối đệm khi checkpoint-record đã được viết xong nhưng trước thời điểm các khối đệm đã sửa đổi được ghi ra đĩa.

Ý tưởng thực hiện fuzzy checkpoint như sau: Thay vì phải dò ngược sổ ghi để tìm mẫu tin checkpoint, ta sẽ lưu vị trí của mẫu tin checkpoint cuối cùng trong sổ ghi vào một chỗ cố định trong đĩa gọi là ***last_checkpoint***. Tuy nhiên, thông tin này sẽ không được cập nhật khi một mẫu tin checkpoint được ghi ra đĩa. Thay vào đó, trước khi một mẫu tin checkpoint được ghi ra sổ ghi, ta tạo ra một danh sách các khối đệm bị sửa đổi. Thông tin ***last_checkpoint*** được cập nhật chỉ sau khi tất cả các khối đệm bị sửa đổi được ghi ra đĩa.

last_checkpoint chỉ được dùng cho mục đích ***undo***.

BÀI TẬP CHƯƠNG VI

VI.1. Trình bày các điểm khác nhau giữa 3 kiểu lưu trữ: lưu trữ không ổn định, lưu trữ ổn định và lưu trữ bền theo tiêu chuẩn đánh giá là *chi phí cài đặt*.

VI.2. Thực tế, lưu trữ bền không thể thực hiện được. Giải thích tại sao? Hệ cơ sở dữ liệu giải quyết vấn đề này như thế nào?

VI.3. So sánh các kỹ thuật cập nhật tức thời và cập nhật có trì hoãn trong sơ đồ phục hồi dựa vào sổ ghi lộ trình theo các tiêu chuẩn: *tính dễ cài đặt và tổng chi phí thực hiện*.

VI.4. Giả sử rằng kỹ thuật cập nhật tức thời được sử dụng trong hệ thống. Bằng ví dụ, hãy chỉ ra rằng: tình trạng không nhất quán dữ liệu sẽ xảy ra nếu các log record không được ghi ra thiết bị lưu trữ bền trước khi giao dịch bàn giao (commit).

VI.5. Giải thích mục đích của cơ chế điểm kiểm soát (checkpoint). Hành động đặt điểm kiểm soát nên được thực hiện theo chu kỳ bao lâu là hợp lý?

VI.6. Khi hệ thống phục hồi sau lỗi, nó xây dựng 2 danh sách: *undo-list* và *redo-list*. Giải thích tại sao các log record của các giao dịch trong danh sách *undo-list* phải được xử lý theo thứ tự ngược, trong khi những log record trong danh sách *redo-list* lại được xử lý theo chiều xuôi?

VI.7. So sánh sơ đồ phục hồi phân trang bóng và sơ đồ phục hồi bằng sử dụng sổ ghi lộ trình theo tiêu chuẩn: *tính dễ cài đặt và tổng chi phí thực hiện*.

VI.8. Giả sử một cơ sở dữ liệu có 10 khối đĩa liên tiếp (khối 1, 2, 3, ..., 10). Hãy thể hiện trạng thái của buffer và thứ tự vật lý có thể có của các khối sau các thao tác cập nhật sau, giả sử: kỹ thuật phân trang bóng được sử dụng, buffer trong bộ nhớ chỉ đủ chứa 3 khối, chiến lược quản lý buffer là LRU (Least Recently Used)

Đọc khối 3
Đọc khối 7
Đọc khối 5

Đọc khối 3

Đọc khối 1

Sửa đổi khối 1

Đọc khối 10

Sửa khối 5