Download đề bài tại:
http://www.mediafire.com/?3hh975tcdvici7z
Download cơ sở dữ liệu tại: http://www.mediafire.com/?amchtkg9ilx1wk1
Sau đó import dữ liệu vào oracle.

TRUY VẤN ORACLE

1. Liệt kê tên (last_name) và lương (salary) của những nhân viên có lương lớn hơn 12000$.

```
SELECT LAST_NAME, SALARY
FROM employees;
```

2. Liệt kê tên và lương của những nhân viên có lương thấp hơn 5000$ hoặc lớn hơn 12000$.

```
SELECT FIRST_NAME, LAST_NAME, SALARY
FROM EMPLOYEES
WHERE SALARY<5000 OR SALARY>12000;
```

3. Cho biết thông tin tên nhân viên (last_name), mã công việc (job_id), ngày thuê (hire_date) của những nhân viên được thuê từ ngày 20/02/1998 đến ngày 1/05/1998.
Thông tin được hiển thị tăng dần theo ngày thuê.

```
SELECT last_name,job_id,hire_date
FROM employees
WHERE hire_date BETWEEN '20/FEB/1998' AND '1/MAY/1998'
ORDER BY hire_date ASC;
```

4. Liệt kê danh sách nhân viên làm việc cho phòng 20 và 50. Thông tin hiển thị gồm:
last_name, department_id , trong đó tên nhân viên được sắp xếp theo thứ tự alphabe.

```
SELECT LAST_NAME, DEPARTMENT_ID
FROM EMPLOYEES
WHERE DEPARTMENT_ID IN(20,50);
```

5. Liệt kê danh sách nhân viên được thuê năm 1994.

```
SELECT *
FROM EMPLOYEES
WHERE TO_CHAR(HIRE_DATE,'YY')='05';
```

6. Liệt kê tên nhân viên (last_name), mã công việc (job_id) của những nhân viên
không có người quản lý.

```
SELECT LAST_NAME, FIRST_NAME, JOB_ID
FROM EMPLOYEES
WHERE MANAGER_ID IS NULL;
```

7. Cho biết thông tin tất cả nhân viên được hưởng hoa hồng (commission_pct),
kết quả được sắp xếp giảm dần theo lương và hoa hồng.

```
SELECT FIRST_NAME, LAST_NAME, COMMISSION_PCT
FROM EMPLOYEES
WHERE COMMISSION_PCT IS NOT NULL;
```

8.Liệt kê danh sách nhân viên mà có kí tự thứ 3 trong tên là "a".

```
SELECT *
FROM EMPLOYEES
WHERE FIRST_NAME LIKE ('__a%');
```

9. Liệt kê danh sách nhân viên mà trong tên có chứa một chữ "a" và một chữ "e".

```
SELECT employee_id, first_name, last_name
FROM employees
WHERE first_name LIKE ('%a%e%');
```

10. Cho biết tên (last_name), mã công việc (job_id), lương (salary) của những
nhân viên làm "Sales representative" hoặc "Stock clert" và có mức lương khác
2500$, 3500$, 7000$.

```
--CACH 1
SELECT A.last_name, A.job_id, A.salary
FROM EMPLOYEES A INNER JOIN JOBS b
ON A.job_id =b.job_id
WHERE  job_title IN ('Sales Representative' ,'Stock Clerk')
AND salary NOT IN (2500,3500,7000);
--cach 2
SELECT A.last_name, A.JOB_ID, A.SALARY
FROM EMPLOYEES A, JOBS B
WHERE B.JOB_TITLE IN ('Sales Representative','Stick cleark')
AND A.SALARY NOT IN(2500,3500,7000);
--CACH 3
SELECT A.LAST_NAME, A.JOB_ID, A.SALARY
FROM EMPLOYEES A
```

```
WHERE A.JOB_ID IN
                (SELECT B.JOB_ID
                FROM JOBS B
                WHERE JOB_TITLE IN('Sales Representative','Stick
cleark'))
AND A.SALARY NOT IN(2500, 3500, 7000);
--CACH 3.1
SELECT A.LAST_NAME, A.JOB_ID, A.SALARY
FROM EMPLOYEES A
WHERE A.SALARY NOT IN(2500, 3500, 7000)
AND A.JOB_ID IN
                (SELECT B.JOB_ID
                FROM JOBS B
                WHERE JOB_TITLE IN('Sales Representative','Stick
cleark'));
--CACH 3.2
SELECT A.LAST_NAME, A.JOB_ID, A.SALARY
FROM EMPLOYEES A
WHERE A.JOB_ID IN
                (SELECT B.JOB_ID
                FROM JOBS B
                WHERE JOB_TITLE IN('Sales Representative','Stick
cleark')
                AND A.SALARY NOT IN(2500, 3500, 7000));
```

11. Cho biết mã nhân viên (employee_id), tên nhân viên (last_name), lương sau khi tăng thêm 15% so với lương ban đầu, được làm tròn đến hàng đơn vị và đặt lại tên cột là “New Salary”.

```
SELECT EMPLOYEE_ID, LAST_NAME, ROUND(SALARY+((SALARY*15)/100),1) AS
"NEW SALARY"
FROM EMPLOYEES;
```

12. Cho biết tên nhân viên, chiều dài tương ứng của tên đối với những nhân
viên có kí tự bắt đầu trong tên là “J”, “A”, “L”,”M”. Kết quả hiển thị tăng dần theo tên, kí tự đầu của tên viết hoa, các kí tự còn lại viết thường.(dùng hàm INITCAP, LENGTH, SUBSTR)

```
SELECT INITCAP(FIRST_NAME), LENGTH(FIRST_NAME)
FROM EMPLOYEES
WHERE SUBSTR(FIRST_NAME,1,1)IN('J','A','A','M')
ORDER BY FIRST_NAME;
```

13. Liệt kê danh sách nhân viên, khoảng thời gian (tính theo tháng) mà nhân

viên đã làm việc trong công ty cho đến nay. Kết quả sắp xếp tăng dần theo số
lượng tháng làm việc. (dùng hàm MONTHS_BETWEEN)

```
SELECT EMPLOYEE_ID, FIRST_NAME, LAST_NAME,
MONTHS_BETWEEN(SYSDATE,HIRE_DATE) AS KHOANG_TG
FROM EMPLOYEES;
```

```
13.1 TUONG TU CAU 13 NHUNG LAM TRON HANG CHUC
SELECT EMPLOYEE_ID, FIRST_NAME, LAST_NAME,
ROUND(MONTHS_BETWEEN(SYSDATE,HIRE_DATE),2) AS KHOANG_TG
FROM EMPLOYEES;
```

14. Thực hiện câu truy vấn cho kết quả theo định dạng sau :
<last_name> earns <salary> monthly but wants <3*salary> .
Cột được hiển thị có tên "Dream Salaries"

```
SELECT concat(concat(concat(concat(last_name,' earns '),salary),
'monthly but wants'),(salary*3)) as Dream_Salaries
FROM EMPLOYEES;
```

15. Liệt kê tên nhân viên, mức hoa hồng nhân viên đó nhận được.
Trường hợp nhân viên nào không được hưởng hoa hồng thì hiển thị
"No commission". (dùng hàm NVL)

```
SELECT FIRST_NAME, NVL(TO_CHAR(COMMISSION_PCT,'.9'),'NO COMMISSION')
FROM EMPLOYEES;
```

16. Thực hiện câu truy vấn cho kết quả như sau: (dùng hàm DECODE hoặc
CASE…)
JOB_ID GRADE
AD_PRES A
ST_MAN B
IT_PROG C
SA_REP D
ST_CLERK E
Không thuộc 0

```
select distinct job_id, case job_id when 'AD_PRES' then 'A'
                                   when 'ST_MAN' then 'B'
                                   when 'IT_PROG' then 'C'
                                   when 'SA_REP' then 'D'
                                   when 'ST_CLERK' then 'E'
            else '0' end "GRADE"
```

```
FROM jobs ORDER BY GRADE ;
```

17. Cho biết tên nhân viên, mã phòng, tên phòng của những nhân viên làm việc ở thành phố Toronto.

```
SELECT EMPLOYEE_ID, FIRST_NAME, LAST_NAME, CITY
FROM EMPLOYEES A INNER JOIN
     (SELECT DEPARTMENT_ID, LOCATION_ID FROM DEPARTMENTS) B
      ON A.DEPARTMENT_ID=B.DEPARTMENT_ID INNER JOIN
           (SELECT LOCATION_ID, CITY
            FROM LOCATIONS WHERE CITY='Toronto')C
            ON B.LOCATION_ID=C.LOCATION_ID;
```

18. Liệt kê thông tin nhân viên cùng với người quản lý của nhân viên đó. Kết quả hiển thị: mã nhân viên, tên nhân viên, mã người quản lý, tên người quản lý.

```
 ----Câu 18 cách 1
cách 1
select e1.employee_id,e1.last_name,e1.first_name, e1.manager_id,
e2.name_manager
from employees e1 join
                (select distinct employee_id, last_name, first_name as
name_manager
                  from employees
                  where employee_id in(select manager_id from
employees))e2
on e1.manager_id=e2.employee_id;

--Câu 18 cách 2
select e1.employee_id,e1.last_name,e1.first_name, e1.manager_id,
e2.name_manager
from employees e1 ,(select distinct employee_id,last_name ,first_name
as name_manager
                  from employees
                  where employee_id in(select manager_id from
employees))e2
where e1.manager_id=e2.employee_id;
```

19. Liệt kê danh sách những nhân viên làm việc cùng phòng.

```
select * from employees
order by department_id;
```

20. Liệt kê danh sách nhân viên được thuê sau nhân viên "Davies".

```
select *
from EMPLOYEES where hire_date >
          (select hire_date from EMPLOYEES where last_name='Davies');
```

21. Liệt kê danh sách nhân viên được thuê vào làm trước người quản lý của họ.

```
select a.last_name,a.hire_date,b.name_manager,b.hire_date_manager
from (select last_name,hire_date,manager_id from EMPLOYEES) a
     inner join (select employee_id,last_name as
     name_manager,hire_date as hire_date_manager
     from EMPLOYEES
     where employee_id in(select distinct manager_id from EMPLOYEES))b
     on a.manager_id=b.employee_id
     where a.hire_date<b.hire_date_manager;
```

22. Cho biết lương thấp nhất, lương cao nhất, lương trung bình, tổng lương của từng loại công việc.

```
SELECT A.DEPARTMENT_ID, C.DEPARTMENT_NAME ,MAX(SALARY), MIN(SALARY),
AVG(SALARY), SUM(SALARY)
FROM (SELECT DEPARTMENT_ID, SALARY FROM EMPLOYEES)A,
(SELECT DEPARTMENT_ID, DEPARTMENT_NAME FROM DEPARTMENTS)C
WHERE A.DEPARTMENT_ID=C.DEPARTMENT_ID
AND A.DEPARTMENT_ID IS NOT NULL
GROUP BY A.DEPARTMENT_ID, C.DEPARTMENT_NAME;
```

23. Cho biết mã phòng, tên phòng, số lượng nhân viên của từng phòng ban.

```
SELECT a.DEPARTMENT_ID,b.DEPARTMENT_NAME, COUNT(*)
FROM (SELECT DEPARTMENT_ID FROM EMPLOYEES) a inner join
(select DEPARTMENT_ID,DEPARTMENT_NAME FROM DEPARTMENTS) b
on a.DEPARTMENT_ID=b.DEPARTMENT_ID
group by a.DEPARTMENT_ID, b.DEPARTMENT_NAME
order by a.DEPARTMENT_ID;
```

24. Cho biết tổng số nhân viên, tổng nhân viên được thuê từng năm 1995, 1996, 1997, 1998.

```
SELECT to_char(hire_date,'YYYY') as Nam, count(*)
from EMPLOYEES
where to_char(hire_date,'YYYY') in ('1996','1997','1998','1995')
group by to_char(hire_date,'YYYY');
```

25. Liệt kê tên, ngày thuê của những nhân viên làm việc cùng phòng với nhân viên "Zlotkey".

```
select last_name,hire_date from EMPLOYEES
where department_id = (select department_id
from EMPLOYEES
where last_name=INITCAP('zlotkey'));
```

26. Liệt kê tên nhân viên, mã phòng ban, mã công việc của những nhân viên làm việc cho phòng ban đặt tại vị trí (location_id) 1700.

```
select a.last_name, a.department_id, a.job_id,c.location_id
from EMPLOYEES a inner join (select department_id,location_id from
DEPARTMENTS) b on
a.department_id=b.department_id
inner join (select location_id from LOCATIONS where location_id=1700)c
                on b.location_id=c.location_id;
```

27. Liệt kê danh sách nhân viên có người quản lý tên "King".

```
select * from EMPLOYEES
where manager_id in (select employee_id from EMPLOYEES where
last_name=INITCAP('king'));
```

28. Liệt kê danh sách nhân viên có lương cao hơn mức lương trung bình và làm việc cùng phòng với nhân viên có tên kết thúc bởi "n".

```
select * from EMPLOYEES
where department_id in(select department_id
                       from employees
                       where last_name like ('%n'))
                        and salary>(select avg(salary) from EMPLOYEES);
```

29. Liệt kê danh sách mã phòng ban, tên phòng ban có ít hơn 3 nhân viên.

```
select department_id,department_name
from DEPARTMENTS
where department_id in (select department_id from employees )
group by department_id
having count(*)<3);
```

30. Cho biết phòng ban nào có đông nhân viên nhất, phòng ban nào có ít nhân viên nhất.

```
select a.department_id,b.department_name,count(*)
from EMPLOYEES a inner join (select department_id,department_name
from DEPARTMENTS) b on a. department_id=b.department_id
group by a.department_id,b.department_name
having count(*) >= all (select count(*)
from EMPLOYEES
group by department_id) or count(*) <= all
          (select count(*) from EMPLOYEES group by department_id);
```

31. Liệt kê danh sách nhân viên được thuê vào ngày có số lượng nhân viên được thuê đông nhất. (dùng hàm TO_CHAR(hire_date, "Day")).

```
select first_name,last_name, a.Ngay
from (select first_name,last_name,to_char(hire_date,'day') Ngay
from EMPLOYEES) a inner join (select to_char(hire_date,'day') Ngay,
count(to_char(hire_date,'day'))
from EMPLOYEES
group by to_char(hire_date,'day')
having count(to_char(hire_date,'day')) >= all
          (select count(to_char(hire_date,'day'))
           from EMPLOYEES
           group by to_char(hire_date,'day'))) b
           on a.Ngay=b.Ngay;
```

32. Liệt kê thông tin 3 nhân viên có lương cao nhất.

```
select * from (select * from EMPLOYEES order by salary desc)

where rownum<4;
```

33. Liệt kê danh sách nhân viên đang làm việc ở tiểu bang "California".

```
select *
from EMPLOYEES a inner join (select department_id, location_id from
DEPARTMENTS) b on a.department_id=b.department_id
inner join (select location_id from LOCATIONS
where state_province='California') c
 on b.location_id=c.location_id;
```

34. Cập nhật tên của nhân viên có mã 3 thành "Drexler".

```
UPDATE EMPLOYEES
SET FIRST_NAME='Drexler'
WHERE EMPLOYEE_ID=3;
```

35. Liệt kê danh sách nhân viên có mức lương thấp hơn mức lương trung bình của phòng ban mà nhân viên đó làm việc.

```
select first_name,last_name,salary
from EMPLOYEES a
inner join (select department_id,avg(salary)  LuongTrungBinh
from EMPLOYEES group by department_id) b
on a.department_id=b.department_id
where salary<LuongTrungBinh;
```

36. Tăng thêm 100$ cho những nhân viên có lương nhỏ hơn 900$.

```
UPDATE EMPLOYEES
SET SALARY = SALARY + 100
WHERE SALARY < 900;
```

37. Xóa phòng ban 500.

```
DELETE FROM DEPARTMENTS
WHERE DEPARTMENT_ID=500;
```

38. Xóa phòng ban nào chưa có nhân viên.

```
DELETE
FROM DEPARTMENTS
WHERE DEPARTMENT_ID NOT IN (SELECT DEPARTMENT_ID FROM EMPLOYEES);
```

TẠO VIEW

39. Tạo view chứa thông tin của những quốc gia ở vùng Asia.

```
create or replace view cau39
as
  select country_name
  from countries
  where region_id in
                    (
                      select region_id
                      from regions
                      where region_name='Asia'
                    );
```

40. Tạo view chứa danh sách nhân viên không có người quản lý.

```
create or replace view cau40
as
    select employee_id, first_name, last_name, manager_id
    from employees
    where manager_id is null;
```

41. Tạo view chứa danh sách phòng ban chưa có nhân viên.

```
create or replace view cau41
as
   select department_id
   from DEPARTMENTS
   minus
   select department_id
   from EMPLOYEES ;
```

42. Tạo view chứa mã nhân viên, tên nhân viên, tên phòng, mã công việc, số năm làm việc, lương của những nhân viên có mức lương lớn hơn mức lương trung bình của công ty.

```
create view as cau42
as
select e1.employee_id, e1.first_name,e1.job_id, salary,
a.department_name
from employees e1, departments a
where e1.department_id=a.department_id;
```

43. Liệt kê các mã phòng ban(department_id) không tồn tại trong bảng nhân viên(employees).

```
select department_id
from departments dept
where not exists (select null from employees emp
where emp.department_id = dept.department_id );
```

TẠO STORE PROCEDURE

43. Tạo thủ tục có tên là dept_info cho biết thông tin về phòng ban với tham số truyền vào là mã phòng ban.

```
create or replace procedure dept_info(v_department_id number,
ten out departments.department_name%type)
as
  begin
    select  department_name into ten
    from departments
    where department_id=v_department_id;
    dbms_output.put_line('Ten phong ban: '||ten);
    exception when no_data_found
              then dbms_output.put_line('Khong co phong ban');
  end;

--Thuc thi
set serveroutput on
declare ten departments.department_name%type;
begin
   DEPT_INFO(&v_department_id, ten);
end;
```

44. Tạo thủ tục có tên là add_job thêm một công việc mới với tham số truyền vào là mã công việc, tên công việc.

```
create or replace procedure add_job (v_macv JOBS.job_id%TYPE, v_tencv
JOBS.job_title%TYPE)
as
v_macv_temp JOBS.job_id%TYPE;
v_loi EXCEPTION;
begin
      select job_id into v_macv_temp
      from JOBS
      where job_id=v_macv;
     if v_macv_temp is not null then
           raise v_loi;
     end if;
exception when v_loi then
     dbms_output.put_line('Khong them duoc');
      when no_data_found then
     insert into JOBS (job_id,job_title) values (v_macv,v_tencv);
           dbms_output.put_line('Cong viec ' || v_tencv || ' da duoc
them.');
end;
--thuc thi
set serveroutput on
execute add_job('p_code','Lap Trinh Vien');
```

45. Tạo thủ tục có tên là update_comm cập nhật hoa hồng cho nhân viên tang thêm 5% hoa hồng ban dầu, tham số truyền vào là mã nhân viên.

```
create or replace procedure update_comm2(manhanvien number)
as
  begin
    update employees
    set commission_pct = commission_pct*1.05
    where employee_id=manhanvien;
    dbms_output.put_line('Cap nhat hoa hong cua '|| manhanvien ||'
thanh cong!');
  end;
--thuc thi
set serveroutput on
execute update_comm2(3);
```

46. Tạo thủ tục có tên là add_emp thêm một nhân viên mới với tất cả các giá trị là tham số truyền vào.

```
create or replace procedure add_emp(v_employee_id number
,first_name varchar2, last_name varchar2, email varchar2, phone_number
```

```
varchar2,hire_date employees.hire_date%type, job_id varchar2, salary
number, commission_pct number
, manager_id number, department_id employees.department_id%type)
as
  v_count number;
  v_loi exception;
  begin
    select count(*) into v_count
    from employees
    where employee_id=v_employee_id;
    if v_count > 0 then raise v_loi;
    else
    insert into employees values(v_employee_id, first_name, last_name,
email
    , phone_number, hire_date, job_id, salary, commission_pct,
manager_id,
    department_id);
    dbms_output.put_line('Them thanh cong!');
    end if;
    exception when v_loi then
    dbms_output.put_line('Nhan vien da ton tai');
  end;
```

47. Tạo thủ tục có tên là delete_emp xóa một nhân viên mới với mã nhân viên là tham số truyền vào.

```
create or replace procedure delete_employee(manhanvien number)
as
  begin
    delete from employees where employee_id=manhanvien;
      dbms_output.put_line('Da xoa'||manhanvien||'thanh cong!');
  end;
```

48. Tạo thủ tục có tên find_emp tìm kiếm nhân viên có lương lớn hơn mức lương thấp nhất

```
create or replace procedure find_emp
as
cursor c_nhanvien is
select employee_id, first_name,last_name
from EMPLOYEES a
where salary>(select min_salary from jobs b where a.job_id=b.job_id)
and salary<(select max_salary from jobs c where a.job_id=c.job_id);
v_emp_id EMPLOYEES.employee_id%TYPE;
v_first_name EMPLOYEES.first_name%TYPE;
v_last_name EMPLOYEES.last_name%TYPE;
```

```
begin
    for r_nhanvien in c_nhanvien
    loop
        v_emp_id := r_nhanvien.employee_id;
        v_first_name := r_nhanvien.first_name;
        v_last_name:=r_nhanvien.last_name;
       dbms_output.put_line('Ma NV: ' || v_emp_id || ' TenNV: ' ||
v_first_name || ' ' || v_last_name);
    end loop;
end;
--Thuc thi
set serveroutput on

execute find_emp;
```

49. Tạo thủ tục có tên update_comm cập nhật luong của nhân viên với điều kiện nhân viên nào làm việc trên 2 nam thì tang luong thêm 200$, nhân viên làm việc trên 1 nam và dưới 2 nam thì tang luong thêm 100$, nhân viên nào làm việc dúng 1 nam thì tang 50$, còn lại không tăng.

```
create or replace procedure update_comm
as
v_sonam_lamviec varchar2(2);
cursor c_nhanvien is
select * from EMPLOYEES;
begin
      for r_nhanvien in c_nhanvien
      loop
          v_sonam_lamviec := to_char(sysdate,'yyyy')-
to_char(r_nhanvien.hire_date,'yyyy');
          if  v_sonam_lamviec>=2 then
                 update EMPLOYEES
                 set salary=salary+200
                 where employee_id=r_nhanvien.employee_id;
                dbms_output.put_line('Nhan vien :' ||
r_nhanvien.last_name || ' da duoc tang 200$');
          else
                 update EMPLOYEES
                 set salary=salary+100
                 where employee_id=r_nhanvien.employee_id;
                dbms_output.put_line('Nhan vien :' ||
r_nhanvien.last_name || ' da duoc tang 100$');
         end if;
      end loop;
end;
```

```
--thuc thi
set serveroutput on
execute update_comm;
```

TẠO FUNCTION

```
--51
create or replace function sum_salary(maphongban in number)
return number
as
 v_salary number;
begin
selectsum(salary)
into v_salary
from employees
where department_id=maphongban;
return v_salary;
  exception
when no_data_found
then
return('Du lieu khong tim thay');
when others then
return('loi ham');
end;
--Test
set serveroutput on
  show error;
select column_name, data_type, data_length
from user_tab_columns
where table_name='EMPLOYEES';
--
SELECT DEPARTMENT_ID FROM DEPARTMENTS;
set serveroutput on
set verify off
execute dbms_output.put_line('Tong luong la: '||sum_salary(&maphong));

52. Tao ham co ten name_con cho biet ten quoc gia voi ma quoc gia la
tham so
--truyen vao
```

```
--Xem kieu du lieu

select column_name, data_type, data_length
from user_tab_columns
where table_name='COUNTRIES';
--52
createorreplacefunction name_con
(maqg in countries.country_id%type)
return varchar2
is
tenqg varchar2(50);
begin
      select country_name
      into tenqg
      from countries
      where country_id=maqg;
      return tenqg;
  exception
when no_data_found
then
return('Du lieu khong tim thay');
when others then
return('loi ham');
end;

--Thuc thi
SELECT COUNTRY_ID FROM COUNTRIES;
--Goi thuc thi 1
set serveroutput on
set verify off
declare quocgia varchar2(35);
begin
      quocgia:=name_con('AU');
      dbms_output.put_line(quocgia);
end;
--Goi thuc thi 2
--53
createorreplacefunction annual_comp
(luong employees.salary%type,
hoahong  employees.commission_pct%type)
return number
as
thunhap number;
begin
      thunhap:=luong*12+(hoahong*luong*12);
```

```
      return thunhap;
  exception
when no_data_found
then
return('Du lieu khong tim thay');
when others then
return('loi ham');
end;
--Goi ham
select salary, commission_pct from EMPLOYEES;
--
set serveroutput on
set verify off
declare ThuNhap number;
begin
      ThuNhap:=annual_comp(14200,0.4);
      dbms_output.put_line(ThuNhap);
end;
--54
Create or replace function avg_salary
(mapb employees.department_id%type)
return number
as
luongtb number;
begin
      selectavg(salary)
      into luongtb
      from employees
      where department_id=mapb;
      return luongtb;
  exception
when no_data_found
then
return('Du lieu khong tim thay');
when others then
return('loi ham');
end;
--Goi thuc thi
set serveroutput on;
set verify off
declare LuongTB number;
begin
      LuongTB:=avg_salary(110);
      dbms_output.put_line(LuongTB);
end;
```

```
--55

Create or replace function Time_work(MaNhanVien
EMPLOYEES.EMPLOYEE_ID%TYPE)
return number
as
  tglamviec number;
begin
select round(months_between(to_date(sysdate,'dd/mm/yyyy'),
  to_date(hire_date,'dd/mm/yyyy')),1)
into tglamviec
from EMPLOYEES
where employee_id=MaNhanVien;
return tglamviec;
    exception when no_data_found then
return('Du lieu khong tim thay');
end;
--Goi thuc thi
set serveroutput on
declare tg number(22);
begin
tg:=Time_work(&manv);
  dbms_output.put_line('So thang lam viec cua nhan vien la '||tg);
end;

show error
--TEST
select*from employees;
--xem kieu du lieu cua cot
select column_name,data_type, data_length
from user_tab_columns
where table_name='EMPLOYEES'

select round(months_between
(to_date(sysdate,'dd/mm/yyyy'),to_date(hire_date,'dd/mm/yyyy')),2)
from employees
```

TẠO TRIGGER

60. Cài đặt ràng buộc toàn vẹn ngày thuê nhân viên phải nhỏ hơn hoặc bằng ngày hiện hành khi thêm mới hoặc cập nhật thông tin về nhân viên.

```
create or replace trigger tr_ngaythue
after insert or update
on EMPLOYEES
for each row
declare v_ngaythue EMPLOYEES.HIRE_DATE%TYPE;
  begin
    if(v_ngaythue>sysdate) then
      raise_application_error(-20020,'Ngay thue khong hop le');
    end if;
  end;
```

61. Cài đặt ràng buộc toàn vẹn min_salary luôn nhỏ hơn max_salary khi thêm mới hoặc cập nhật thông tin bảng công việc

```
create or replace trigger tr_luong
before insert or update
on jobs
for each row
  begin
    if(:new.min_salary>:new.max_salary)then
      raise_application_error(-20022,'Luong nhap vao khong hop le');
    end if;
  end;
```

62. Cài đặt ràng buộc toàn vẹn ngày bắt đầu luôn nhỏ hơn hoặc bằng ngày kết thúc khi thêm mới hoặc cập nhật thông tin bảng job_history.

```
create or replace trigger tr_Ngay
before insert or update
on job_history
for each row
  begin
    if(:new.start_date>:new.end_date) then
      raise_application_error(-20021,'Ngay bat dau phai nho hon ngay
ket thuc');
    end if;
  end;
```

63. Cài đặt ràng buộc toàn vẹn lương và hoa hồng của nhân viên phải tăng chứ không giảm khi cập nhật nhân viên.

```
create or replace trigger tr_Luong_HoaHong
before update
on employees
for each row
  begin
    if(:new.salary<:old.salary) then
      raise_application_error(-20022,'Luong cap nhat phai lon hon
luong hien tai');
    end if;
    if(:new.commission_pct<:old.commission_pct) then
      raise_application_error(-20022,'Hoa hong cap nhat phai lon hon
luong hien tai');
    end if;
  end;
```

TẠO PACKAGE

```
create or replace package emp_info
as
procedure salary_table(manv employees.employee_id%TYPE);
function sum_salary (mapb departments.department_id%TYPE)
return number;
end;

create or replace package body emp_info
as
procedure salary_table(manv employees.employee_id%TYPE)
as
luong employees.salary%TYPE;
begin
select salary into luong from employees where employee_id=manv;
dbms_output.put_line('luong cua nhan vien nay la:' || luong);
exception
when no_data_found then
dbms_output.put_line('ko tim thay nhan vien nay');
end salary_table;

function sum_salary(mapb departments.department_id%TYPE)
return number
as
tongluong number;
begin
select sum(salary) into tongluong from employees where
department_id=mapb;
return tongluong;
exception
when no_data_found then
dbms_output.put_line('ko tim thay phong ban');
end sum_salary;
end emp_info;

thuc thi
set serveroutput on;
declare
v_luong number;
v_tongluong number;
begin
emp_info.salary_table(198);
v_tongluong:=emp_info.sum_salary(110);
```

```
dbms_output.put_line(v_luong);
dbms_output.put_line(v_tongluong);
end;
```

58. Tạo package có tên là job_pack chứa 3 thủ tục tên là add_job để thêm công việc, update_job để cập nhật công việc, del_job để xóa công việc và 1 hàm có tên q_job để tìm kiếm tên công việc theo mã.

```
create or replace package job_pack
as
  procedure add_job
  (
    macv jobs.job_id%type,
    tencv jobs.job_title%type,
    luongthapnhat jobs.min_salary%type,
    luongcaonhat jobs.max_salary%type
  );
  procedure update_job
  (
    macv jobs.job_id%type,
    tencv jobs.job_title%type,
    luongthapnhat jobs.min_salary%type,
    luongcaonhat jobs.max_salary%type
  );
  procedure del_job
  (
    macv jobs.job_id%type
  );
  end job_pack;
--phan than
create or replace package body job_pack
as
   procedure add_job
   (
   macv jobs.job_id%type,
   tencv jobs.job_title%type,
   luongthapnhat jobs.min_salary%type,
   luongcaonhat jobs.max_salary%type
```

```
   )
  is
    begin
     insert into JOBS
     values (macv,tencv,luongthapnhat, luongcaonhat);
           dbms_output.put_line('Cong viec '||tencv||' da
duoc them');
      exception when no_data_found then
         dbms_output.put_line('Khong tim thay cong viec');
    end add_job;
  procedure update_job
  (
    macv jobs.job_id%type,
    tencv jobs.job_title%type,
    luongthapnhat jobs.min_salary%type,
    luongcaonhat jobs.max_salary%type
  )
  is
  begin
  update jobs set job_title=tencv,min_salary=luongthapnhat
  , max_salary=luongcaonhat where job_id=macv;
  dbms_output.put_line('CAP NHAT THANH CONG');
  end update_job;
  procedure del_job(macv jobs.job_id%type)
  is
    begin
    delete from jobs where job_id=macv;
      dbms_output.put_line('XOA THANH CONG');
    end del_job;
  end job_pack;

--Thuc thi
--add_job
begin
job_pack.add_job('ADMIN2','ADMINISTRATOR2',20000,60000);
end;
--update_job
begin
      job_pack.update_job('ADMIN2','AA',21000,61000);
```

```
end;
--del_job cach goi 1
EXECUTE job_pack.del_job('ADMIN')
--del_job cach goi 2
begin
    job_pack.del_job('ADMIN3');
end;
```

59. Tạo package có tên emp_pack chứa một thủ tục tên new_emp thêm một nhân viên mới với tất cả các tham số truyền vào và một hàm tên valid_deptid kiểm tra mã phòng ban hợp lệ , khi mã phòng hợp lệ mới được phép thêm nhân viên.

```
create or replace package emp_pack
as
  procedure new_emp
  (
    MaNV employees.employee_id%type,
    TenNV employees.first_name%type,
    HoNV employees.last_name%type,
    Email employees.email%type,
    DienThoai employees.phone_number%type,
    NgayThue employees.hire_date%type,
    MaCV employees.job_id%type,
    Luong employees.salary%type,
    HoaHong employees.commission_pct%type,
    MaQuanLy employees.manager_id%type,
    MaPhong employees.department_id%type
  );
  function valid_deptid(i_department_id in number)
  return boolean;
  end emp_pack;
--phan than
create or replace package body emp_pack
as

  procedure new_emp
  (
    MaNV employees.employee_id%type,
    TenNV employees.first_name%type,
```

```
    HoNV employees.last_name%type,
    Email employees.email%type,
    DienThoai employees.phone_number%type,
    NgayThue employees.hire_date%type,
    MaCV employees.job_id%type,
    Luong employees.salary%type,
    HoaHong employees.commission_pct%type,
    MaQuanLy employees.manager_id%type,
    MaPhong employees.department_id%type
  )
  is
  begin
    insert into employees values(MaNV, TenNV, HoNV, Email,
DienThoai,
    NgayThue, MaCV, Luong, HoaHong, MaQuanLy,MaPhong);
  end new_emp;
  --ket thuc proc new_emp
   function valid_deptid(i_department_id in number)
   return boolean
    is
        v_id_dept number;
        begin
              select count(*) into v_id_dept
              from departments
              where department_id=i_department_id;
            return 1=v_id_dept;
        exception when others then
        return false;
      end valid_deptid;--ket thuc proc valid_deptid
  end emp_pack;

  --goi thuc thi
  set serveroutput on
  begin
        if(emp_pack.valid_deptid(&i_department_id)) then
         emp_pack.new_emp(1, 'First',
'Last','first.last@oracle.com',
           '(123)123-1234','18-JUN-02','IT_PROG',900,00,
100,110);
```

```
            dbms_output.put_line('Them thanh cong');
        else
          dbms_output.put_line('Ma phong ban nay khong ton
tai!');
        end if;
  end;
```

QUẢN TRỊ NGƯỜI DÙNG

Quản trị người dùng.

64. Tạo không gian bảng (tablespace) có kích thước 100M.

65. Tạo không gian bảng tạm (temporary tablespace) có kích thước 50M.

66. Tạo rollback segment rolora để truy xuất đồng thời cho table space vừa tạo.

67. Tạo user có tên là tên sinh viên, mật khẩu do sinh viên tự đặt với tablespace và temporary tablespace vừa tạo.

68. Cấp quyền truy xuất tài nguyên (resource) cho user vừa tạo.

69. Cấp quyền cho phiên làm việc (session) cho user vừa tạo.

70. Cấp quyền tạo bảng (table) cho user vừa tạo.

71. Cấp quyền tạo khung nhìn (view) cho user vừa tạo.

72. Cấp quyền Select, Insert, Update, Delete trên bảng Employees cho user vừa tạo.

73. Cấp quyền Select, Insert, Update, Delete trên bảng Departments cho user vừa tạo.

74. Cấp quyền Select chỉ với các thuộc tính job_id, job_title trên bảng Jobs cho user vừa tạo.

75. Login vào csdl HR với user vừa tạo.

76. Truy vấn các bảng trong csdl HR và cho nhận xét.

77. Cho biết các user hiện có từ view dba_users.

78. Đăng nhập với quyền hệ thống và tao user có tên là mã sinh viên, mật khẩu là tên sinh viên

a. Thay đổi mật khẩu của user.

b. Cấp quyền đăng nhập csdl

c. Truy xuất view v$session để xem phiên làm việc.

d. Tao bảng phòng ban gồm 2 thuộc tính: mã phòng ban, tên phòng ban và cho nhận xét câu c.

e. Cấp phát hạn ngạch (quota) 20M cho user vừa tao.

f. Tao lại bảng Phòng ban và cho nhận xét.

g. Cấp phát không gian giới hạn tablespace cho user vừa tao

h. Cấp phát tài nguyên cho user

i. Cấp phát phiên làm việc cho user

j. Cấp quyền đăng nhập OEM

79. Truy xuất vào view hệ thống dba_profiles.

80. Tao profile giới hạn việc truy xuất tài nguyên và cấp cho user vừa tạo:

a. Thời gian kết nối 120s

b. Số lần thất bại khi cho phép kết nối là 3.

c. Thời gian chờ kết nối là 60s

d. Phiên làm việc cho mỗi user là 2

e. Gán profile cho user vừa tao

f. Thay đổi thông số của profile với phiên làm việc cho mỗi user là 4.

g. Số lần thất bại khi cho phép kết nối là 3

h. Thiết lập thông số để profile có hiệu lực.

i. Xóa profile vừa tạo.

81. Oracle Database Resource Manager(ODRM) để tạo vùng treo,tao nhóm người dùng, tao kế hoạch tài nguyên và định hướng kế hoạch, kiểm tra tính hợp lệ của vùng treo và gởi vùng treo:

a. Tạo vùng treo.

b. Xóa vùng treo.

c. Kiểm tra nhóm người dùng nào đã tồn tại và cho nhận xét liên quan các nhóm người dùng.

d. Kiểm tra kế hoạch của mỗi CSDL từ dba_rsrc_plans

e. Tao 4 nhóm người dùng: cập nhật, tìm kiếm, thống kê, báo cáo

f. Kiểm tra tính hợp lệ của vùng treo.

g. Truy xuất view dba_users và cho nhận xét.

h. Gán người dùng vừa tạo tới nhóm người dùng.

i. Truy xuất view dba_users và cho nhận xét.

j. Khởi tạo user cho nhóm người dùng

k. Truy xuất view dba_users và cho nhận xét.

l. Tạo kế hoạch tài nguyên

m. Tạo định hướng kế hoạch và giới hạn tài nguyên cho từng người dùng (CPU_1 nhóm cập nhật là 60, CPU_1 nhóm thống kê 40, CPU_2 nhóm báo cáo là 30, nhóm khác là 100).

n. Gởi vùng treo.

o. Truy xuất view dba_rsrc_plan_directives và cho nhận xét.

p. Kích hoạt kế hoạch sử dụng tài nguyên

q. Truy xuất view v$rsrc_consumer_group để xem việc sử dụng tài nguyên giữa các nhóm

r. Truy xuất view v$rsrc_plan và cho nhận xét.

```
--64
create tablespace oracle
datafile'oracle.dbf' size 100m;
--65
create temporary tablespace oracle_2
tempfile'oracle2.dbf' size 50m;
--66
create rollback segment seg
tablespace oracle;
```

```
--67
create user sinhvien identified by sinhvien
default tablespace oracle
temporary tablespace oracle_2
password expire;
--68
grant resource to sinhvien;
--69
grant create session to sinhvien;
--70
grant create table to sinhvien;
--71
grant create view to sinhvien;
--72
grant insert, update, select, delete on hr.employees to
sinhvien;
--73
grant insert, select, update, delete on hr.departments to
sinhvien;


--74
grant update(job_id, job_title) on hr.jobs to sinhvien;
--75
```

```
conn hr/hr;
--76
Khong truy van duoc vi chua cap quyen truy van
--77
select username from dba_users;
--78
conn system/system;
grant create user to hr;
conn hr/hr;
create user masinhvien identified by masinhvien
password exprire;
--78bis xoa user masinhvien
conn system/system;
drop user masinhvien;
--78b
create create connect to masinhvien;
--78c
select username, status from v$session;
--78 d Khong tao duoc vi chua cap quyen
--78e
create user sinhvien identified by sinhvien
default tablespace oracle
```

```
temporary tablespace oracle_2
quota 100m on oracle;
--80 a, b, c, d
create profile giang limit
connect_time 120
failed_login_attempts 3
idle_time 60
sessions_per_user 2;
--80e
alter user sinhvien identified by sinhvien
profile giang;
--80h
thiet lap thong so de profile co hieu luc
grant create profile to sinhvien;
--80f,g
alter profile giang limit
sessions_per_user 4
failed_login_attempts 3
--80i
drop profile giang cascade;
--81 a.Tao vung treo
exec dbms_resource_manager.create_pending_area;
```

```
--81 b.xoa vung treo

exec dbms_resource_manager.clear_pending_area;

--81 e

exec dbms_resource_manager.create_pending_area;

exec
dbms_resource_manager.create_consumer_group('capnhat','nhom
cap nhat');

exec
dbms_resource_manager.create_consumer_group('timkiem','nhom
tim kiem');

exec
dbms_resource_manager.create_consumer_group('thongke','nhom
thong ke');

exec
dbms_resource_manager.create_consumer_group('baocao','nhom
bao cao');

--81 f

--kiem tra tinh hop le cua vung treo

exec dbms_resource_manager.validate_pending_area

--81 h

grant capnhat to sinhvien;

grant timkiem to sinhvien;

grant thongke to sinhvien;

grant baocao to sinhvien;

--81 i
```

```
select username, account_status from dba_users

--81 h. Gán người dùng vừa tạo tới nhóm người dùng


conn system/system

exec
dbms_resource_manager_privs.grant_system_privilege('SYSTEM',
'ADMINISTER_RESOURCE_MANAGER',TRUE);

exec dbms_resource_manager.create_pending_area

exec dbms_resource_manager.create_consumer_group('sv','nhom
sv','ROUND-ROBIN'

--khong chay dc

exec
dbms_resource_manager_privs.grant_switch_consumer_group('sin
hvien','sv',true)

--81 l

exec dbms_resource_manager.create_pending_area

exec
dbms_resource_manager.create_plan('kehoachtainguyen','ke
hoach tai nguyen cho nguoi dung')

--81 m

exec
dbms_resource_manager.create_plan_directive('kehoachtainguye
n','capnhat',60,100,100,100,100,100,100,100)

exec
dbms_resource_manager.create_plan_directive('kehoachtainguye
n','thongke',40,100,100,100,100,100,100,100)
```

```
exec
dbms_resource_manager.create_plan_directive('kehoachtainguye
n','baocao',30,100,100,100,100,100,100,100)

--81 n gui ving treo

exec dbms_resource_manager.create_pending_area

exec dbms_resource_manager.submit_pending_area

--81 o

select plan, cpu_p1, cpu_p2, cpu_p3 from
dba_rsrc_plan_directives

--Tao role

create role capnhat;

create role timkiem;

create role thongke;

create role baocao;

--cap quyen

grant capnhat to sinhvien;

--Xoa role

revoke capnhat from sinhvien;
```

THE END

Trường……
Khoa……..

Câu lệnh truy vấn căn bản trong SQL

Welcome to SQL tutorial

Mục lục

SQL căn bản

SQL nâng cao

Giới thiệu SQL

SQL là một ngôn ngữ theo chuẩn ANSI để truy xuất các cơ sở dữ liệu.

SQL là gì?

- SQL là **S**tructured **Q**uery **L**anguage – Ngôn ngữ Truy vấn có Cấu trúc
- SQL cho phép bạn truy xuất một cơ sở dữ liệu
- SQL là một ngữ theo chuẩn ANSI
- SQL có thể thực hiện các truy vấn đến một cơ sở dữ liệu
- SQL có thể truy tìm dữ liệu từ một cơ sở dữ liệu
- SQL có thể chèn các mẩu tin mới vào trong một cơ sở dữ liệu
- SQL có thể xóa các mẩu tin trong một cơ sở dữ liệu

- SQL có thể cập nhật các mẩu tin trong một cơ sở dữ liệu
- SQL rất dễ học

SQL là một chuẩn

SQL là một chuẩn ANSI (American National Standards Institute - Viện Tiêu chuẩn Quốc gia Mỹ) cho các hệ thống truy xuất cơ sở dữ liệu. Các phát biểu SQL dùng để truy tìm và cập nhật dữ liệu trong một cơ sở dữ liệu.
SQL làm việc với các trình quản lý cơ sở dữ liệu như Access, DB2, Informix, Microsoft SQL Server, Oracle, Sybase, và nhiều trình khác (đáng tiếc là đa số trong chúng có các phần mở rộng ngôn ngữ SQL riêng).

Các bảng cơ sở dữ liệu

Cơ sở dữ liệu chứa các đối tượng gọi là các **Bảng** (Tables).
Các **Mẩu tin** (Records) lưu trong các bảng này. Các bảng được gọi theo tên bảng (như "Persons", "Orders", "Suppliers").
Các bảng chứa các **Cột** (Columns) và các **Dòng** (Rows) dữ liệu. Dòng chứa các mẩu tin (như mẩu tin về một người). Cột chứa dữ liệu (như First Name, Last Name, Address, và City).
Một ví dụ là bảng "Persons" sau:

LastName	FirstName	Address	City
Hansen	Ola	Timoteivn 10	Sandnes
Svendson	Tove	Borgvn 23	Sandnes
Pettersen	Kari	Storgt 20	Stavanger

LastName, FirstName, Address, và City là các cột của bảng. Các dòng chứa ba mẩu tin của 3 người.

Các truy vấn SQL

Với SQL, chúng ta có thể **truy vấn** một cơ sở dữ liệu và nhận được một **kết quả** trả về với dạng bảng.
Một truy vấn giống như sau:

```
SELECT LastName FROM Persons
```

Sẽ trả về một kết quả giống như sau:

LastName
Hansen
Svendson
Pettersen

Chú ý: Vài hệ cơ sở dữ liệu cần một dấu ";" ở cuối phát biểu SQL. Chúng ta không dùng dấu ";" trong bài viết này.

Thao tác dữ liệu SQL

SQL là một cú pháp để thực hiện các truy vấn. Nhưng ngôn ngữ SQL cũng chứa các cú pháp cập nhật các mẩu tin (record), chèn các mẩu tin mới và xóa các mẩu tin đang tồn tại.
Các lệnh truy vấn và cập nhật này thuộc dạng Ngôn ngữ Thao tác Dữ liệu (Data Manipulation Language - DML) một phần của SQL:

- SELECT – trích dữ liệu từ một cơ sở dữ liệu
- UPDATE – cập nhật dữ liệu trong một cơ sở dữ liệu
- DELETE – xóa dữ liệu từ một cơ sở dữ liệu
- INSERT – chèn dữ liệu mới vào trong một cơ sở dữ liệu

Định nghĩa dữ liệu SQL

Ngôn ngữ Định nghĩa Dữ liệu (Data Definition Language - DDL) một phần của SQL, cho phép tạo hay xóa các bảng cơ sở dữ liệu. Chúng ta cũng có thể định nghĩa các chỉ mục (các khóa - key), chỉ định liên kết giữa các bảng, và ràng buột giữa các bảng cơ sở dữ liệu.
Các phát biểu DDL quan trọng nhất trong SQL là::

- CREATE TABLE – tạo một bảng cơ sở dữ liệu mới
- ALTER TABLE – thay đổi (alters) một bảng cơ sở dữ liệu

- DROP TABLE – xóa một bảng cơ sở dữ liệu
- CREATE INDEX – tạo một chỉ mục (khóa tìm kiếm)
- DROP INDEX – xoá một chỉ mục

SQL và ASP

SQL là một phần quan trọng của ASP (Active Server Pages), vì ADO (Active Data Object) được dùng trong ASP để truy xuất cơ sở dữ liệu, ADO dựa trên SQL để truy xuất dữ liệu.

Phát biểu SQL Select

Phát biểu SELECT chọn các cột dữ liệu từ một cơ sở dữ liệu.

Kết quả dạng bảng được lưu trong một bảng kết quả (gọi là tập kết quả - result set).

Phát biểu SELECT

Phát biểu SELECT chọn các cột dữ liệu từ một cơ sở dữ liệu.
Dùng phát biểu này để chọn (SELECT) thông tin từ (FROM) một bảng như sau:

```
SELECT column_name(s) FROM table_name
```

Ví dụ: Chọn các cột từ một bảng

Để chọn các cột có tên "LastName" và "FirstName", dùng một phát biểu SELECT như sau:

```
SELECT LastName,FirstName FROM Persons
```

Bảng "Persons":

LastName	FirstName	Address	City
Hansen	Ola	Timoteivn 10	Sandnes
Svendson	Tove	Borgvn 23	Sandnes
Pettersen	Kari	Storgt 20	Stavanger

Kết quả:

LastName	FirstName
Hansen	Ola
Svendson	Tove
Pettersen	Kari

Ví dụ: Chọn tất cả các cột

Để chọn tất cả các cột từ bảng "Person", dùng một ký hiệu * thay thế cho tên các cột như sau:

```
SELECT * FROM Persons
```

Kết quả:

LastName	FirstName	Address	City
Hansen	Ola	Timoteivn 10	Sandnes
Svendson	Tove	Borgvn 23	Sandnes
Pettersen	Kari	Storgt 20	Stavanger

Bảng kết quả

Kết quả từ một truy vấn SQL được lưu trữ trong một tập kết quả. Tập kết quả có thể xem như một bảng kết quả. Đa số các

trình quản lý cơ sở dữ liệu cho phép duyệt tập kết quả với các hàm lập trình như: Move-To-First-Record, Get-Record-Content, Move-To-Next-Record.....

Mệnh đề SQL Where

Mệnh đề WHERE dùng để chỉ định một tiêu chuẩn (criteria) chọn.

Mệnh đề WHERE

Để chọn có điều kiện dữ liệu từ một bảng, một mệnh đề WHERE có thể thêm vào phát biểu SELECT với cú pháp sau:

```
SELECT column FROM table WHERE column condition value
```

Với mệnh đề WHERE, các điều kiện sau có thể được dùng:

Operator	Condition
=	Bằng
<>	Không bằng
>	Lớn hơn
<	Nhỏ hơn
>=	Lớn hơn hoặc bằng
<=	Nhỏ hơn hoặc bằng
LIKE	Sẽ giải thích bên dưới

Chú ý: Vài phiên bản SQL toán tử <> có thể được viết thành !=

Ví dụ: Chọn người từ một công ty

Để chọn những người chỉ sống ở Sandnes, thêm mệnh đề WHERE vào phát biểu SELECT như sau:

```
SELECT * FROM Persons WHERE City='Sandnes'
```

Bảng "Persons":

LastName	FirstName	Address	City	Year
Hansen	Ola	Timoteivn 10	Sandnes	1951
Svendson	Tove	Borgvn 23	Sandnes	1978
Svendson	Ståle	Kaivn 18	Sandnes	1980
Pettersen	Kari	Storgt 20	Stavanger	1960

Kết quả:

LastName	FirstName	Address	City	Year
Hansen	Ola	Timoteivn 10	Sandnes	1951
Svendson	Tove	Borgvn 23	Sandnes	1978
Svendson	Ståle	Kaivn 18	Sandnes	1980

Dùng dấu nháy

Chú ý rằng chúng ta dùng dấu nháy đơn bao quanh các trị điều kiện trong các ví dụ. SQL dùng dấu nháy đơn bao quanh các trị văn bản. Phần lớn các hệ quản lý cơ sở dữ liệu cũng chấp nhận dấu nháy kép. Các trị số không được đóng trong dấu nháy.
Với các trị văn bản:

```
Viết đúng:
SELECT * FROM Persons WHERE FirstName='Tove'
Viết sai:
SELECT * FROM Persons WHERE FirstName=Tove
```

Với các trị số:

```
Viết đúng:
SELECT * FROM Persons WHERE Year>1965
Viết sai:
SELECT * FROM Persons WHERE Year>'1965'
```

Điều kiện LIKE

Điều kiện LIKE dùng chỉ định việc tìm một mẫu trong một cột.
Cú pháp:

```
SELECT column FROM table WHERE column LIKE pattern
```

Một dấu "%" có thể dùng như ký tự đại diện (wildcards) cả trước lẫn sau mẫu.

Ví dụ: Chọn trong bảng Persons với mẫu tên

Phát biểu SQL sẽ trả về những người có firstname bắt đầu với một ký tự 'O'.

```
SELECT * FROM Persons WHERE FirstName LIKE 'O%'
```

Phát biểu SQl sẽ trả về những người có firstname kết thúc với một ký tự 'a'.

```
SELECT * FROM Persons WHERE FirstName LIKE '%a'
```

Phát biểu SQL sẽ trả về những người có firstname chứa mẫu 'la'.

```
SELECT * FROM Persons WHERE FirstName LIKE '%la%'
```

Tất cả các ví dụ trên sẽ trả về kết quả sau:

LastName	FirstName	Address	City	Year
Hansen	Ola	Timoteivn 10	Sandnes	1951

SQL And & Or

AND & OR

AND và OR kết nối hai hay nhiều điều kiện trong một mệnh đề WHERE.
Toán tử AND hiển thị một cột nếu TẤT CẢ các điều kiện liệt kê đều đúng.
Toán tử OR hiển thị một cột nếu MỘT TRONG các điều kiện liệt kê là đúng.

Bảng gốc (dùng trong các ví dụ)

LastName	FirstName	Address	City
Hansen	Ola	Timoteivn 10	Sandnes
Svendson	Tove	Borgvn 23	Sandnes
Svendson	Stephen	Kaivn 18	Sandnes

Ví dụ

Dùng AND để hiển thị người có firstname là "Tove", và lastname là "Svendson":

```
SELECT * FROM Persons
WHERE FirstName='Tove'
AND LastName='Svendson'
```

Kết quả:

LastName	FirstName	Address	City
Svendson	Tove	Borgvn 23	Sandnes

Ví dụ

Dùng OR để hiển thị người có firstname là "Tove", hoặc có lastname là "Svendson":

```
SELECT * FROM Persons
WHERE firstname='Tove'
OR lastname='Svendson'
```

Kết quả:

LastName	FirstName	Address	City
Svendson	Tove	Borgvn 23	Sandnes
Svendson	Stephen	Kaivn 18	Sandnes

Ví dụ

Bạn cũng có thể dùng phối hợp AND và OR (dùng dấu ngoặc đơn để bao các biểu thức phức tạp):

```
SELECT * FROM Persons WHERE
(FirstName='Tove' OR FirstName='Stephen')
AND LastName='Svendson'
```

Kết quả:

LastName	FirstName	Address	City
Svendson	Tove	Borgvn 23	Sandnes
Svendson	Stephen	Kaivn 18	Sandnes

SQL Between...And

BETWEEN ... AND

Toán tử BETWEEN ... AND chọn tất cả các trị trong khoảng giới hạn giữa hai trị. Các trị này có thể là các số, văn bản, hay ngày tháng.

```
SELECT column_name FROM table_name
WHERE column_name
BETWEEN value1 AND value2
```

Bảng gốc (dùng trong các ví dụ)

LastName	FirstName	Address	City
Hansen	Ola	Timoteivn 10	Sandnes
Svendson	Tove	Borgvn 23	Sandnes
Nordmann	Anna	Neset 18	Sandnes
Pettersen	Kari	Storgt 20	Stavanger

Ví dụ 1

Để hiển thị các tên theo thứ tự alphabet giữa hai tên (kể cả hai tên này) "Hansen" và "Pettersen", dùng SQL sau:

```
SELECT * FROM Persons WHERE LastName
BETWEEN 'Hansen' AND 'Pettersen'
```

Kết quả:

LastName	FirstName	Address	City
Hansen	Ola	Timoteivn 10	Sandnes
Nordmann	Anna	Neset 18	Sandnes
Pettersen	Kari	Storgt 20	Stavanger

Ví dụ 2

Để hiển thị các tên ngoài các tên trong ví dụ trên, dùng toán tử NOT:

```
SELECT * FROM Persons WHERE LastName
NOT BETWEEN 'Hansen' AND 'Pettersen'
```

Kết quả:

LastName	FirstName	Address	City
Svendson	Tove	Borgvn 23	Sandnes

SQL Select Distinct

Từ khóa DISTINCT dùng trả về chỉ các trị khác biệt (distinct).

Từ khóa DISTINCT

Phát biểu SQL SELECT trả về thông tin từ các cột của bảng. Nhưng làm thế nào nếu chúng ta chỉ muốn chọn các kết quả không trùng nhau?
Với SQL, chúng ta chỉ cần thêm vào một từ khóa DISTINCT cho phát biểu SELECT với cú pháp sau:

```
SELECT DISTINCT column-name(s) FROM table-name
```

Ví dụ: Chọn tên công ty từ bảng Orders

Ví dụ: Bảng đặt hàng đơn giản:

Company	OrderNumber
Sega	3412
W3Schools	2312
Trio	4678
W3Schools	6798

Phát biểu SQL sau:

```
SELECT Company FROM Orders
```

Sẽ trả về kết quả:

Company
Sega
W3Schools
Trio
W3Schools

Chú ý rằng công ty W3Schools xuất hiện hai lần trong kết quả. Đôi lúc chúng ta không muốn điều này.

Ví dụ: Chọn tên công ty (không trùng tên) từ bảng Orders

Phát biểu SQL sau:

```
SELECT DISTINCT Company FROM Orders
```

Sẽ trả về kết quả:

Company
Sega
W3Schools
Trio

Bây giờ tên công ty W3Schools chỉ xuất hiện một lần trong kết quả.

SQL Order By

Từ khóa ORDER BY dùng sắp xếp kết quả thứ tự kết quả.

Sắp xếp các Dòng

Mệnh đề ORDER BY dùng sắp xếp các dòng.
Một số cách sắp xếp:

Company	OrderNumber
Sega	3412
ABC Shop	5678
W3Schools	2312
W3Schools	6798

Ví dụ

Để hiển thị tên công ty (Company) theo thứ tự alphabet:

```
SELECT Company, OrderNumber FROM Orders
ORDER BY Company
```

Kết quả:

Company	OrderNumber
ABC Shop	5678
Sega	3412
W3Schools	6798
W3Schools	2312

Ví dụ

Để hiển thị tên công ty (Company) theo thứ tự alphabet, nếu tên công ty giống nhau thì sắp xếp theo số thứ tự (OrderNumber):

```
SELECT Company, OrderNumber FROM Orders
ORDER BY Company, OrderNumber
```

Kết quả:

Company	OrderNumber
ABC Shop	5678
Sega	3412
W3Schools	2312
W3Schools	6798

Ví dụ

Để hiển thị tên công ty (Company) theo thứ tự alphabet đảo ngược (từ Z đến A):

```
SELECT Company, OrderNumber FROM Orders
ORDER BY Company DESC
```

Kết quả:

Company	OrderNumber
W3Schools	6798
W3Schools	2312
Sega	3412
ABC Shop	5678

SQL INSERT INTO

Chèn các dòng mới

Phát biểu INSERT INTO chèn các dòng mới vào trong một bảng:

```
INSERT INTO table_name
VALUES (value1, value2,....)
```

Bạn có thể chỉ định các cột bạn muốn chèn chèn dữ liệu vào:

```
INSERT INTO table_name (column1, column2,...)
VALUES (value1, value2,....)
```

Chèn một dòng mới

Bảng "Persons":

LastName	FirstName	Address	City
Pettersen	Kari	Storgt 20	Stavanger

Phát biểu SQL chèn vào bảng trên:

```
INSERT INTO Persons
VALUES ('Hetland', 'Camilla', 'Hagabakka 24', 'Sandnes')
```

Sẽ cho kết quả như sau:

LastName	FirstName	Address	City
Pettersen	Kari	Storgt 20	Stavanger
Hetland	Camilla	Hagabakka 24	Sandnes

Chèn dữ liệu vào trong các cột chỉ định

Bảng "Persons":

LastName	FirstName	Address	City
Pettersen	Kari	Storgt 20	Stavanger
Hetland	Camilla	Hagabakka 24	Sandnes

Phát biểu SQL chèn dữ liệu vào các cột chỉ định:

```
INSERT INTO Persons (LastName, Address)
VALUES ('Rasmussen', 'Storgt 67')
```

Sẽ cho kết quả như sau::

LastName	FirstName	Address	City
Pettersen	Kari	Storgt 20	Stavanger
Hetland	Camilla	Hagabakka 24	Sandnes
Rasmussen		Storgt 67	

SQL Update

Update Rows

Phát biểu UPDATE cập nhật hoặc thay đổi các dòng:

```
UPDATE table_name SET column_name = new_value
WHERE column_name = some_value
```

Bảng Person:

LastName	FirstName	Address	City
Nilsen	Fred	Kirkegt 56	Stavanger
Rasmussen		Storgt 67	

Cập nhật một cột trong một dòng

Chúng ta sẽ thêm một first name “Nina” đến người có lastname="Rasmussen":

```
UPDATE Person SET FirstName = 'Nina'
WHERE LastName = 'Rasmussen'
```

Cập nhật vài cột trong một dòng

Chúng ta sẽ thay đổi địa chỉ (Address) và thêm tên thành phố.

```
UPDATE Person
SET Address = 'Stien 12', City = 'Stavanger'
WHERE LastName = 'Rasmussen'
```

Kết quả

LastName	FirstName	Address	City
Nilsen	Fred	Kirkegt 56	Stavanger
Rasmussen	Nina	Stien 12	Stavanger

SQL Delete

Xóa các cột

Phát biểu DELETE dùng xóa một hay nhiều dòng trong một bảng.

```
DELETE FROM table_name WHERE column_name = some_value
```

Bảng "Person":

LastName	FirstName	Address	City
Nilsen	Fred	Kirkegt 56	Stavanger
Rasmussen	Nina	Stien 12	Stavanger

Xóa một dòng

"Nina Rasmussen" sẽ bị xóa:

```
DELETE FROM Person WHERE LastName = 'Rasmussen'
```

Kết quả

LastName	FirstName	Address	City
Nilsen	Fred	Kirkegt 56	Stavanger

Các hàm SQL Count

SQL có các hàm tạo sẵn để đếm các mẩu tin cơ sở dữ liệu.

Cú pháp hàm Count

Cú pháp của các hàm COUNT tạo sẵn như sau:

```
SELECT COUNT(column) FROM table
```

Hàm COUNT(*)

Hàm COUNT(*) trả về số hàng chọn được trong một phép chọn.
Với bảng "Persons" sau:

Name	Age
Hansen, Ola	34
Svendson, Tove	45
Pettersen, Kari	19

Ví dụ này trả về số hàng trong bảng:

```
SELECT COUNT(*) FROM Persons
```

Kết quả:

```
3
```

Ví dụ này trả về số người lớn hơn 20 tuổi:

```
SELECT COUNT(*) FROM Persons WHERE Age>20
```

Kết quả:

```
2
```

Hàm COUNT(column)

Hàm COUNT(column) trả về số hàng (ngoại trừ hàng có giá trị NULL) trong cột chỉ định.
Với bảng "Persons":

Name	Age
Hansen, Ola	34
Svendson, Tove	45
Pettersen, Kari	

Ví dụ này tìm số người có ghi tuổi tại field “Age” trong bảng "Persons":

```
SELECT COUNT(Age) FROM Persons
```

Kết quả:

```
2
```

Hàm COUNT(column) cũng dùng để tính số hàng không chứa trị. Chú ý kết quả sẽ nhỏ hơn số hàng trong bảng.

COUNT DISTINCT

Từ khóa DISTINCT với COUNT có thể dùng để đếm số kết quả khác nhau (không trùng nhau).
Cú pháp như sau:

```
SELECT DISTINCT COUNT(column(s)) FROM table
```

Với bảng "Orders":

Company	OrderNumber
Sega	3412
W3Schools	2312
Trio	4678
W3Schools	6798

Với phát biểu SQL sau:

```
SELECT COUNT(Company) FROM Orders
```

Sẽ trả về kết quả:

```
4
```

Với phát biểu SQL sau:

```
SELECT DISTINCT COUNT(Company) FROM Orders
```

Sẽ trả về kết quả:

```
3
```

Các hàm SQL

SQL có một số hàm tạo sẵn để đếm và tính toán.

Cú pháp dùng hàm

Cú pháp cho các hàm SQL tạo sẵn như sau::

```
SELECT function(column) FROM table
```

Bảng gốc (dùng trong các ví dụ)

Name	Age
Hansen, Ola	34
Svendson, Tove	45
Pettersen, Kari	19

Hàm AVG(column)

Hàm AVG trị trung bình của dữ liệu trong một cột có đu7ọc nhờ phép chọn. Các trị NULL sẽ không được tính toán.
Ví dụ
Ví dụ này trả về tuổi trung bình của những người trong bảng "Persons":

```
SELECT AVG(Age) FROM Persons
```

Kết quả

```
32.67
```

Ví dụ
Ví dụ này trả về tuổi trung bình của những người có tuổi lớn hơn 20 tuổi:

```
SELECT AVG(Age) FROM Persons where Age>20
```

Kết quả

```
39.5
```

Hàm MAX(column)

Hàm MAX trả về trị lớn nhất trong một cột. Các trị NULL sẽ không được tính toán.
Ví dụ

```
SELECT MAX(Age) FROM Persons
```

Kết quả:

```
45
```

Hàm MIN(column)

Hàm MIN trả về trị lớn nhất trong một cột. Các trị NULL sẽ không được tính toán.
Ví dụ

```
SELECT MIN(Age) FROM Persons
```

Kết quả:

```
19
```

Chú ý: Các hàm MIN và MAX cũng có thể dùng trên các cột văn bản, để tìm trị lớn nhất và nhỏ nhất theo thứ tự alphabet.

Hàm SUM(column)

Hàm SUM tổng của một cột có được nhờ phép chọn. Các trị NULL sẽ không được tính toán.
Ví dụ
Ví dụ này trả về tổng số tuổi của những người trong bảng "Persons":

```
SELECT SUM(Age) FROM Persons
```

Kết quả:

98

Ví dụ
Ví dụ này trả về tổng số tuổi của những người lớn hơn 20 tuổi.

```
SELECT SUM(Age) FROM Persons where Age>20
```

Kết quả:

79

SQL Group By và SQL Having

Các hàm tổng (như SUM) thường kèm theo chức năng GROUP BY.

Từ khóa GROUP BY

Từ khóa GROUP BY được thêm vào SQL vì các hàm tổng (như SUM) trả về tổng của tất cả các trị trong cột mỗi khi chúng ta gọi đến.
Thiếu chức năng GROUP BY, không thể tìm tổng của mỗi nhóm trị riêng trong cột.
Cú pháp của GROUP BY như sau:

```
SELECT column,SUM(column) FROM table GROUP BY column
```

Ví dụ GROUP BY

Bảng "Sales":

Company	Amount
W3Schools	5500
IBM	4500
W3Schools	7100

Với SQL:

```
SELECT Company, SUM(Amount) FROM Sales
```

Trả về kết quả như sau:

Company	SUM(Amount)
W3Schools	17100
IBM	17100
W3Schools	17100

SQL trên không trả về tổng riêng biệt của từng công ty. Dùng mệnh đề GROUP BY như sau:

```
SELECT Company,SUM(Amount) FROM Sales
GROUP BY Company
```

Sẽ trả về kết quả đúng:

Company	SUM(Amount)
W3Schools	12600
IBM	4500

Từ khóa The HAVING

Từ khóa HAVING được thêm vào SQL vì từ khóa WHERE không thể dùng với các hàm tổng (như hàm SUM).
Thiếu từ khóa HAVING sẽ không thể kiểm tra các điều kiện dùng hàm tổng.
Cú pháp của HAVING như sau:

```
SELECT column,SUM(column) FROM table
GROUP BY column
HAVING SUM(column) condition value
```

Bảng "Sales":

Company	Amount
W3Schools	5500
IBM	4500
W3Schools	7100

Với SQL:

```
SELECT Company,SUM(Amount) FROM Sales
GROUP BY Company HAVING SUM(Amount)>10000
```

Trả về kết quả

Company	SUM(Amount)
W3Schools	12600

Các bí danh (Alias) SQL

Với SQL, các bí danh (alias) có thể dùng thay các tên cột và các tên bảng.

Bí danh tên Cột

Cú pháp như sau:

```
SELECT column AS column_alias FROM table
```

Bí danh tên Bảng

Cú pháp như sau:

```
SELECT column FROM table AS table_alias
```

Ví dụ: Dùng bí danh tên Cột

Bảng “Persons”:

LastName	FirstName	Address	City
Hansen	Ola	Timoteivn 10	Sandnes
Svendson	Tove	Borgvn 23	Sandnes
Pettersen	Kari	Storgt 20	Stavanger

Với SQL sau:

```
SELECT LastName AS Family, FirstName AS Name
FROM Persons
```

Sẽ trả về kết quả sau:

Family	Name
Hansen	Ola
Svendson	Tove
Pettersen	Kari

Ví dụ: Dùng bí danh tên Bảng

Bảng “Persons”:

LastName	FirstName	Address	City
Hansen	Ola	Timoteivn 10	Sandnes
Svendson	Tove	Borgvn 23	Sandnes

Pettersen	Kari	Storgt 20	Stavanger

Với SQL sau:

```
SELECT LastName, FirstName
FROM Persons AS Employees
```

Sẽ trả về kết quả sau:
Bảng Employees:

LastName	FirstName
Hansen	Ola
Svendson	Tove
Pettersen	Kari

SQL Join

Joins and các Khóa (Key)

Đôi khi chúng ta chọn dữ liệu từ hai bảng để tạo kết quả, Chúng ta thực hiện một kết nối (join).
Các bảng trong cơ sở dữ liệu có thể liên hệ với các bảng khác thông qua các khóa. Một khóa chính (primary key) là một cột với các trị duy nhất cho mỗi hàng. Mục tiêu là ràng buộc dữ liệu, tham chiếu chéo các bảng, không cần lặp lại tất cả dữ liệu trong từng bảng.
Trong bảng "Employees" phía dưới, cột "ID" là khóa chính, nghĩa là cột này **không** có hai hàng cùng ID. ID dùng phân biệt hai người nếu cả hai có cùng tên.
Khi bạn xem bảng ví dụ phía dưới, chú ý rằng:

- Cột "ID" là khóa chính của bảng "Employees"
- Cột "ID" trong bảng "Orders" dùng để tham chiếu các tên trong bảng "Employees" không cần đưa các tên này vào bảng “Orders”

Employees:

ID	Name
01	Hansen, Ola
02	Svendson, Tove
03	Svendson, Stephen
04	Pettersen, Kari

Orders:

ID	Product
01	Printer
03	Table
03	Chair

Tham chiếu đến hai Bảng

Chúng ta có thể chọn dữ liệu từ hai bảng bằng cách tham chiếu đến hai bảng, như sau:

Ví dụ

Ai đã đăng ký một sản phẩm và đăng ký sản phẩm nào?

```
SELECT Employees.Name, Orders.Product
FROM Employees, Orders
WHERE Employees.ID = Orders.ID
```

Kết quả

Name	Product
Hansen, Ola	Printer
Svendson, Stephen	Table

Svendson, Stephen	Chair

Ví dụ

Ai đã đăng ký một máy in?

```
SELECT Employees.Name
FROM Employees, Orders
WHERE Employees.ID = Orders.ID
AND Orders.Product = 'Printer'
```

Kết quả

Name
Hansen, Ola

Dùng các Kết nối (Join)

HOẶC, chúng ta có thể chọn dữ liệu từ hai bảng với từ khóa JOIN, giống như sau:

Ví dụ INNER JOIN

Cú pháp

```
SELECT field1, field2, field3
FROM first_table
INNER JOIN second_table
ON first_table.keyfield = second_table.foreign_keyfield
```

Ai đã đăng ký một sản phẩm và đăng ký sản phẩm nào?

```
SELECT Employees.Name, Orders.Product
FROM Employees
INNER JOIN Orders
ON Employees.ID = Orders.ID
```

INNER JOIN trả về tất cả các hàng từ hai bảng khi điều kiện được so trùng. Nếu các hàng trong bảng Employees không so trùng trong bảng Orders, hàng đó sẽ **không** được liệt kê ra.

Kết quả

Name	Product
Hansen, Ola	Printer
Svendson, Stephen	Table
Svendson, Stephen	Chair

Ví dụ LEFT JOIN

Cú pháp

```
SELECT field1, field2, field3
FROM first_table
LEFT JOIN second_table
ON first_table.keyfield = second_table.foreign_keyfield
```

Liệt kê tất cả nhân viên, và các đăng ký mua của họ nếu có.

```
SELECT Employees.Name, Orders.Product
FROM Employees
LEFT JOIN Orders
ON Employees.ID = Orders.ID
```

LEFT JOIN trả về tất cả các hàng từ bảng thứ nhất (Employees), cho dù nó không được so trùng trong bảng thứ hai (Orders). Nếu các hàng trong bảng Employees không so trùng trong bảng Orders, những hàng này **cũng được** liệt kê.

Kết quả

Name	Product
Hansen, Ola	Printer
Svendson, Tove	
Svendson, Stephen	Table
Svendson, Stephen	Chair
Pettersen, Kari	

Ví dụ RIGHT JOIN

Cú pháp

```
SELECT field1, field2, field3
FROM first_table
RIGHT JOIN second_table
ON first_table.keyfield = second_table.foreign_keyfield
```

Liệt kê tất cả nhân viên, và các đăng ký mua của họ nếu có.

```
SELECT Employees.Name, Orders.Product
FROM Employees
RIGHT JOIN Orders
ON Employees.ID = Orders.ID
```

RIGHT JOIN trả về tất cả các hàng từ bảng thứ hai (Orders), cho dù nó không được so trùng trong bảng thứ nhất (Employees). Nếu có bất kỳ hàng nào trong bảng Orders không được so trùng trong bảng Employees, các hàng này **cũng được** liệt kê.

Kết quả

Name	Product
Hansen, Ola	Printer
Svendson, Stephen	Table
Svendson, Stephen	Chair

Ví dụ

Ai đăng ký một máy in?

```
SELECT Employees.Name
FROM Employees
INNER JOIN Orders
ON Employees.ID = Orders.ID
WHERE Orders.Product = 'Printer'
```

Kết quả

Name
Hansen, Ola

SQL Tạo Cơ sở dữ liệu và Bảng

Tạo một Cơ sở dữ liệu

Để tạo một cơ sở dữ liệu:

```
CREATE DATABASE database_name
```

Tạo một bảng

Để tạo một bảng trong một cơ sở dữ liệu:

```
CREATE TABLE table_name
(
column_name1 data_type,
column_name2 data_type,
.....
)
```

Ví dụ

Ví dụ này minh họa các bạn tạo một bảng tên "Person", với bốn cột tên:"LastName", "FirstName", "Address", và "Age":

```
CREATE TABLE Person
(
LastName varchar,
FirstName varchar,
Address varchar,
Age int
)
```

Ví dụ này minh họa cách bạn chỉ định kích thước tối đa của vài cột:

```
CREATE TABLE Person
(
LastName varchar(30),
FirstName varchar,
Address varchar,
Age int(3)
)
```

Kiểu dữ liệu được chỉ định là kiểu dữ liệu chứa trong cột. Bảng dưới chứa các kiểu dữ liệu thường gặp nhất trong SQL:

Kiểu dữ liệu	**Mô tả**
integer(size) int(size) smallint(size) tinyint(size)	Chỉ chứa số nguyên. Số ký tự số tối đa được chỉ định trong dấu ngoặc đơn
decimal(size, d) numeric(size,d)	Chứa số với phân số. Số ký tự số tối đa được chỉ định trong "size". Số ký tự số tối đa bên phải (phần phân số) được chỉ định trong "d"
char(size)	Chứa chuỗi có kích thước cố định (có thể chứa ký tự chữ, số, và các ký tự đặc biệt). Kích thước cố định được chỉ định trong dấu ngoặc đơn
varchar(size)	Chứa một chuỗi có chiều dài thay đổi (có thể chứa ký tự chữ, số, và các ký tự đặc biệt). Kích thước tối đa được chỉ định trong dấu ngoặc đơn
date(yyyymmdd)	Chứa một ngày

Tạo Chỉ mục (Index)

Chỉ mục được tạo ra trên một bảng có sẵn để định vị thêm nhanh và hiệu quả các hàng. Có thể tạo một chỉ mục trên một hoặc nhiều cột của một bảng, với một chỉ mục cho một tên. Người dùng không nhìn thấy các chỉ mục, chúng chỉ dùng để tăng tốc độ truy vấn.

Chú ý: Cập nhật một bảng chứa chỉ mục cần nhiều thời gian hơn cập nhật một bảng không chứa chỉ mục, vì chỉ mục cũng cần cập nhật. Tuy nhiên, ý tưởng tốt là tạo chỉ mục chỉ trên các cột thường tìm kiếm nhất.

Một Chỉ mục duy nhất

Tạo một chỉ mục duy nhất trên một bảng. một chỉ mục duy nhất nghĩa là không thể có hai hàng có cùng một trị chỉ mục.

```
CREATE UNIQUE INDEX index_name
ON table_name (column_name)
```

"column_name" chỉ định cột bạn muốn chỉ mục.

Một Chỉ mục đơn giản

Tạo một chỉ mục đơn giản trên một bảng. Khi từ khóa UNIQUE không có, các trị trùng sẽ được cho phép.

```
CREATE INDEX index_name
ON table_name (column_name)
```

"column_name" chỉ định cột bạn muốn chỉ mục.

Ví dụ

Ví dụ này tạo một chỉ mục đơn giản, có tên "PersonIndex", trên field LastName của bảng Person:

```
CREATE INDEX PersonIndex
ON Person (LastName)
```

Nếu bạn muốn chỉ mục các trị trong một cột theo thứ tự **giảm** (descending), bạn có thể thêm từ **DESC** sau tên cột:

```
CREATE INDEX PersonIndex
ON Person (LastName DESC)
```

Nếu bạn muốn chỉ mục nhiều hơn một cột bạn có thể liệt kê các tên cột trong dấu ngoặc đơn, tách chúng bằng dấu phẩy:

```
CREATE INDEX PersonIndex
ON Person (LastName, FirstName)
```

Xóa chỉ mục

Bạn có thể xóa một chỉ mục có trong một bảng với phát biểu DROP.

```
DROP INDEX table_name.index_name
```

Xóa một cơ sở dữ liệu hoặc bảng

Để xóa một cơ sở dữ liệu:

```
DROP DATABASE database_name
```

Để xóa một bảng:

```
DROP TABLE table_name
```

Để xóa toàn bộ dữ liệu trong bảng mà không xóa bảng:

```
DELETE TABLE table_name
```

SQL Alter Table

Alter Table

Phát biểu ALTER TABLE dùng để thêm hay loại bỏ các cột trong một bảng cho trước.

```
ALTER TABLE table_name
ADD column_name datatype
ALTER TABLE table_name
DROP column_name
```

Person:

LastName	FirstName	Address
Pettersen	Kari	Storgt 20

Ví dụ

Để thêm một cột tên "City" vào bảng "Person":

```
ALTER TABLE Person ADD City varchar(30)
```

Kết quả:

LastName	FirstName	Address	City
Pettersen	Kari	Storgt 20	

Ví dụ

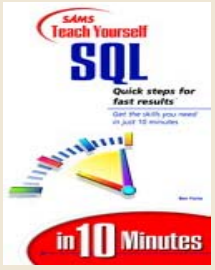
Để loại cột "Address" khỏi bảng "Person":

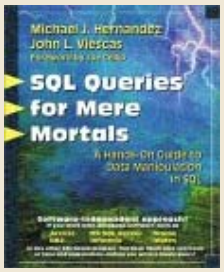
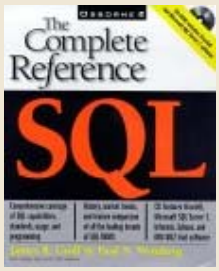
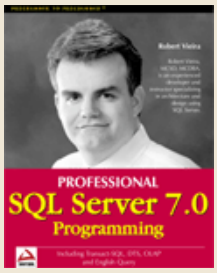
```
ALTER TABLE Person DROP Address
```

Kết quả:

LastName	FirstName	City
Pettersen	Kari	

Sách SQL

Sách	Mô tả
	Teach Yourself SQL in 10 Minutes September 1999 Loại sách tutorial, tổ chức thành chuỗi các bài học-10 phút đơn giản.

	SQL Queries for Mere Mortals August 2000 Giúp người dùng mới học cơ bản về các truy vấn SQL, và cung cấp một hướng dẫn tham chiếu cần thiết với người dùng có trình độ cao hơn.
	SQL: The Complete Reference October 1999 Cung cấp tất cả những gì bạn cần biết về SQL.
	Professional SQL Server 2000 Programming December 2000 Cung cấp một hướng dẫn toàn diện để lập trình với SQL Server 2000.
	Professional SQL Server 7.0 Programming September 1999 Cung cấp tổng quan về tất cả các bộ phận của SQL Server.

SQL Server 2000 : Các câu lệnh truy vấn dữ liệu - Lệnh SELECT FROM – Phần 1

Lập trình trong Transaction-SQL chủ yếu là bạn sử dụng các câu lệnh truy vấn và kết hợp với cấu trúc điều khiển thích hợp cùng các biến đã được khai báo để thực hiện các hành động thích hợp cho việc cập nhật dữ liệu vào bên trong cơ sở dữ liệu.

Các lệnh truy vấn thường dùng như thêm dòng dữ liệu mới vào bảng, xóa các dòng dữ liệu đang có trong bảng, thay đổi giá trị các cột dữ liệu bên trong bảng, chọn lựa các dòng dữ liệu từ các bảng cần thiết. Tuy nhiên đối với cú pháp đầy đủ của lệnh SELECT rất phức tạp và khó nhớ vì thế sẽ hướng dẫn bạn riêng rẽ theo từng thành phần khác nhau nhằm giúp bạn dễ hiểu, dễ nhớ.

1/- Lệnh SELECT FROM

Ý nghĩa hoạt động của câu lệnh SELECT FROM dùng để cho phép bạn có thể chọn lựa các dữ liệu cần thiết từ một hoặc nhiều bảng có quan hệ bên trong một cơ sở dữ liệu.

Câu lệnh này thường được dùng nhiều bên trong Transaction-SQL. Tuy nhiên cũng giống như cú pháp của lệnh CREATE TABLE, bạn vẫn có thể sử dụng cùng lúc đồng thời đầy đủ các mệnh đề của lệnh SELECT FROM.

1.1/- Lệnh SELECT FROM đơn giản :

Với cú pháp SELECT FROM bên dưới cho phép bạn có thể chọn ra dữ liệu của các cột hiện có bên trong một bảng. Với cú pháp này tên các cột phải được chỉ định rõ ràng.

Cú pháp :

```
SELECT Danh_sách_các_cột
    FROM Tên_bảng
```

Trong đó :

• **Dang sách các cột :** là tên các cột hiện đang có bên trong bảng mà bạn cần lấy dữ liệu.

• **Tên bảng :** tên bảng cần hiển thị dữ liệu.

Ví dụ :

Để hiển thị thông tin của các vật tư trong bảng VATTU gồm những cột : mã vật tư, tên vật tư. Bạn thực hiện câu lệnh sau :

```
SELECT MAVTU, TENVTU
       FROM VATTU
```

Kết quả truy vấn trả về :

```
MAVTU TENVTU
----------------------------------------
TG01  Bia lon Tiger
DD01  Đầu DVD Hitachi 1 đĩa
DD02  Đầu DVD Hitachi 3 đĩa
VD01  Đầu VCD Sony 1 đĩa
VD02  Đầu VCD Sony 3 đĩa
TV14  Tivi Sony 14 inches
TV21  Tivi Sony 21 inches
TL15  Tủ lạnh Sanyo 150 lít
TL90  Tủ lạnh Sanyo 90 lít

(10 row(s) affected)
```

1.2/- Mệnh đề sắp xếp dữ liệu :

Với cú pháp SELECT FROM bên dưới kết hợp mệnh đề ORDER BY cho phép bạn có thể lấy dữ liệu của các cột bên trong một bảng, sau đó sắp xếp lại dữ liệu theo thứ tự chỉ định là tăng hoặc giảm.

Cú pháp :

```
SELECT Danh_sách_các_cột
FROM Tên_bảng
ORDER BY Tên_cột_sx [DESC][,...]
```

Trong đó :

• **Tên cột sắp xếp :** là tên cột được sắp xếp dữ liệu. Thứ tự ưu tiên sắp xếp các cột dữ liệu từ trái sang phải và mặc định theo thứ tự tăng dần.

• **Từ khóa DESC :** dùng chỉ thay đổi thứ tự sắp xếp là giảm dần. Mặc định thứ tự sắp xếp là tăng dần.

Ví dụ :

Để hiển thị thông tin của các vật tư trong bảng VATTU gồm những cột : mã vật tư, tên vật tư, phần trăm có sắp xếp dữ liệu theo cột tỷ lệ phần trăm tăng dần. Bạn thực hiện câu lệnh SELECT FROM như sau :

```
SELECT MAVTU, TENVTU, PHANTRAM
FROM VATU
ORDER BY PHANTRAM
```

Kết quả truy vấn trả về :

```
MAVTU TENVTU                    Phần trăm
------------------------------------------------------------
TV21  Tivi Sony 21 inches       15
TV29  Tivi Sony 29 inches       15
TL90  Tủ lạnh Sanyo 90 lít      20
TV14  Tivi Sony 14 inches       20
TL15  Tủ lạnh Sanyo 150 lít     25
VD01  Đầu VCD Sony 1 đĩa        30
VD02  Đầu VCD Sony 3 đĩa        30
DD01  Đầu DVD Hitachi 1 đĩa     40
DD02  Đầu DVD Hitachi 3 đĩa     40
TG01  Bia lon Tiger             60

(10 row(s) affected)
```

1.3/- Mệnh đề chọn các dòng dữ liệu :

Với cú pháp SELECT FROM bên dưới kết hợp mệnh đề WHERE cho phép bạn có thể lọc các dòng dữ liệu bên trong một bảng phải thỏa điều kiện đưa ra trong mệnh đề WHERE.

Cú pháp :

```
SELECT [DISTINCT] [TOP Số [PERCENT]]
Danh_sách_các_cột
FROM Tên_bảng
WHERE Điều_kiện_lọc
[ORDER BY Tên_cột [DESC][,...]]
```

Trong đó :

• **Từ khóa DISTINCT :** dùng để chỉ định truy vấn chỉ chọn ra các dòng dữ liệu duy nhất, không trùng lắp dữ liệu.

• **Từ khóa TOP :** dùng để chỉ định truy vấn chỉ chọn ra chính xác bao nhiêu dòng dữ liệu đầu tiên. Nếu có thêm từ khóa PERCENT đi kèm theo thì truy vấn chỉ chọn ra bao nhiêu phần trăm mẫu tin đầu tiên, lúc bấy giờ con số mà bạn chỉ định phải nằm trong phạm vi từ 0 đến 100. Thông thường khi sử dụng từ khóa TOP thì bạn sẽ kết hợp mệnh đề ORDER BY để sắp xếp lại dữ liệu theo một thứ tự nào đó.

• **Điều kiện lọc :** là điều kiện chỉ định việc lọc ra các mẫu tin bên trong bảng. Thông thường là một biểu thức luận lý.

Ví dụ :

Để hiển thị toàn bộ thông tin của các vật tư trong bảng VATTU sao cho chỉ chọn ra các vật tư có đơn vị tính "Cái". Bạn thực hiện câu lệnh SELECT FROM như sau :

```
SELECT *
       FROM VATTU
       WHERE DVTINH = "Cái"
```

Ký hiệu * trong ví dụ này là đại diện cho tất cả các cột có bên trong bảng. Kết quả truy vấn trả về :

```
Mavtu Tenvtu                    DvTinh  Phần trăm
-------------------------------------------------------------
TL15  Tủ lạnh Sanyo 150 lít     Cái     25
TL90  Tủ lạnh Sanyo 90 lít      Cái     20
TV14  Tivi Sony 14 inches       Cái     20
TV21  Tivi Sony 21 inches       Cái     15
TV29  Tivi Sony 29 inches       Cái     15

(5 row(s) affected)
```

Ví dụ :

Giống như ví dụ trên nhưng bạn chỉ muốn chọn ra vật tư đầu tiên có tỷ lệ phần trăm cao nhất. Bạn thực hiện câu lệnh SELECT FROM có kết hợp mệnh đề TOP như sau :

```
SELECT TOP 1 *
FROM VATTU
WHERE DVTINH = "Cái"
ORDER BY PHANTRAM DESC
```

Kết quả truy vấn trả về :

```
Mavtu Tenvtu                          DvTinh  Phần trăm
-------------------------------------------------------------------
TL15  Tủ lạnh Sanyo 150 lít           Cái     25

(1 row(s) affected)
```

Đối với các người sử dụng ngôn ngữ SQL cũ trước đây, mệnh đề WHERE còn giúp họ có thể liên kết dữ liệu của nhiều bảng có quan hệ trong các truy vấn lấy dữ liệu từ nhiều bảng khác nhau.

Cú pháp :

```
SELECT Danh_sách_các_cột
FROM Tên_bảng1, Tên_bảng2
WHERE Mệnh_đề_liên_kết
```

Trong đó :

• **Mệnh đề liên kết :** thông thường dùng để chỉ định các cột có quan hệ chung của giữa hai bảng tham chiếu trong truy vấn, có dạng như sau :

• **Tên_bảng1.Tên_cột = Tên_bảng2.Tên_cột**

Ví dụ :

Để hiển thị thông tin của các đơn đặt hàng trong bảng DONDH kèm theo cột họ tên của nhà cung cấp tương ứng trong bảng NHACC và sắp xếp dữ liệu hiển thị theo thứ tự mã nhà cung cấp tăng dần. Bạn thực hiện lệnh SELECT FROM như sau :

```
SELECT NCC.MANHACC, TENNHACC, SODH
        FROM DONDH DH, NHACC NCC
        WHERE DH.MANHACC=NCC.MANHACC
        ORDER BY NCC.MANHACC
```

Kết quả truy vấn trả về :

```
MANHACC    TENNHACC          SODH
-----------------------------------------------
C01    Lê Minh Trí           D002
C02    Trần Minh Thạch       D003
C02    Trần Minh Thạch       D005
C02    Trần Minh Thạch       D007
C03    Nguyễn Hồng Phương    D001
C05    Lưu Nguyệt Quế        D004
C05    Lưu Nguyệt Quế        D006

(7 row(s) affected)
```

Trong ví dụ trên hai bảng DONDH và NHACC có chung cột quan hệ là MANHACC sẽ được sử dụng trong mệnh đề WHERE. Do cột MANHACC nằm trong hai bảng DONDH và NHACC vì thế bạn cần phải chỉ định rõ ràng lấy MANHACC trong bảng NHACC bằng cách ghi NCC.MANHACC sau mệnh đề SELECT.

SQL Server 2000 : Các câu lệnh truy vấn dữ liệu – Phần 2

Với cú pháp SELECT FROM bên dưới kết hợp mệnh đề GROUP BY cho phép bạn có thể nhóm dữ liệu của các dòng bên trong một bảng và được phép sử dụng các hàm thống kê đi kèm theo để tính toán các dữ liệu có tính chất thống kê tổng hợp. Thông thường, sau khi nhóm dữ liệu, bạn nên sắp xếp lại dữ liệu để hiển thị theo một thứ tự nào đó. Do vậy bạn sẽ sử dụng mệnh đề ORDER BY sau mệnh đề GROUP BY.

1.4/- Mệnh đề nhóm dữ liệu :

Cú pháp :

```
SELECT Danh_sách_các_cột | Hàm_thống_kê AS Bí_danh
      FROM Tên_bảng
      [WHERE Điều_kiện_lọc]
      GROUP BY Danh_sách_cột_nhómdl
      [ORDER BY Tên_cột [DESC] [, ...]]
```

Trong đó :

• **Hàm thống kê :** là tên của các hàm thống kê và các tham số tương ứng dùng để tính tổng (SUM), tính giá trị thấp nhất (MIN), tính giá trị cao nhất (MAX), đếm các mẫu tin (COUNT), tính giá trị trung bình (AVG) của các dữ liệu bên trong bảng.

• **Bí danh :** là tiêu đề mới của các cột tính toán. Các tiêu đề này chỉ có hiệu lực lúc hiển thị dữ liệu trong câu lệnh truy vấn mà không làm ảnh hưởng đến cấu trúc bên trong của bảng.

• **Danh sách cột nhóm dữ liệu :** là danh sách tên các cột được nhóm dữ liệu để tính toán.

Ví dụ :

Để thống kê tổng số đơn đặt hàng mà công ty đã đặt hàng theo từng nhà cung cấp và sắp xếp dữ liệu hiển thị theo thứ tự tổng số đơn đặt hàng tăng dần. Bạn thực hiện câu lệnh SELECT FROM như sau :

```
SELECT MANHACC, COUNT(*) AS TONG_SODH
      FROM DONDH
      GROUP BY MANHACC
      ORDER BY TONG_SODH
```

Kết quả truy vấn trả về :

```
MANHACC    TONG_SODH
-------------------------------
C01        1
C03        1
C05        2
C02        3

(4 row(s) affected)
```

1.5/- Mệnh đề lọc dữ liệu sau khi đã nhóm :

Với cú pháp SELECT FROM bên dưới kết hợp mệnh đề HAVING cho phép bạn có thể lọc lại dữ liệu sau khi đã nhóm dữ liệu của các dòng bên trong một bảng. Khác với mệnh đề WHERE dùng để lọc các dòng dữ liệu hiện đang có bên trong bảng, mệnh đề HAVING chỉ được phép sử dụng đi kèm theo mệnh đề GROUP BY dùng để lọc lại dữ liệu sau khi đã nhóm. Điều này có nghĩa là mệnh đề HAVING chỉ được dùng kèm với mệnh đề GROUP BY.

Cú pháp :

```
SELECT Danh_sách_các_cột | Hàm_thống_kê AS Bí_danh
        FROM Tên_bảng
        [WHERE Điều_kiện_lọc]
        GROUP BY Danh_sách_cột_nhómdl
        HAVING Điều_kiện_lọc_nhóm
        [ORDER BY Tên_cột [DESC] [, ...]]
```

Trong đó :

• **Điều kiện lọc nhóm :** là điều kiện dùng để lọc lại dữ liệu sau khi đã nhóm dữ liệu. Thông thường là các biểu thức luận lý.

Ví dụ :

Theo ví dụ trên nhưng bạn chỉ cần lọc ra những nhà cung cấp có mã bắt đầu bằng chữ "C" và tổng số các đơn đặt hàng lớn hơn 1 sau khi đã tính toán dữ liệu theo nhóm. Bạn thực hiện câu lệnh SELECT FROM như sau :

```
SELECT MANHACC, COUNT(*) AS TONG_SODH
        FROM DONDH
        WHERE MANHACC LIKE "C%"
        GROUP BY MANHACC
        HAVING COUNT(*)>1
        ORDER BY TONG_SODH
```

Kết quả truy vấn trả về :

```
MANHACC    TONG_SODH
-------------------------------
C05          2
C02          3

(2 row(s) affected)
```

Trong ví dụ này bạn thấy rằng việc lọc dữ liệu được chia ra ở hai mệnh đề khác nhau. Thứ nhất mệnh đề WHERE MANHACC LIKE "C%" dùng để lọc ra các mẫu tin trong bảng DONDH sao cho mã nhà cung cấp phải bắt đầu bằng chữ "C", thứ hai mệnh đề HAVING COUNT(*)>1 dùng để lọc lại các nhà cung cấp nào có tổng số các đơn đặt hàng lớn hơn 1 sau khi đã nhóm để tính ra tổng số các đơn đặt hàng theo từng nhà cung cấp.

1.6/- Mệnh đề liên kết dữ liệu trong hai bảng :

Với cú pháp SELECT FROM bên dưới kết hợp mệnh đề JOIN cho phép bạn liên kết hai bảng có quan hệ với nhau để lấy ra các dữ liệu chung. Điểm quan trọng giữa những bảng

này phải có các cột quan hệ chung nhau va thứ tự quan hệ khi bạn chỉ định giữa các bảng cũng sẽ làm ảnh hưởng đến kết quả của truy vấn.

Cú pháp :

```
SELECT Danh_sách_các_cột
        FROM Tên_bảng
        INNER {LEFT | RIGHT | FULL [OUTER]}
        JOIN Tên_bảng_quan_hệ ON Điều_kiện_quan_hệ
```

Trong đó :

• **Từ khóa INNER JOIN :** dùng để chỉ định việc so sánh giá trị trong các cột của các bảng là tương đương (dữ liệu đều có ở cả hai bảng). Hệ thống sẽ trả về các mẫu tin thỏa điều kiện quan hệ ở cả hai bảng.

• **Từ khóa LEFT RIGHT FULL :** dùng để chỉ định việc so sánh giá trị các cột của bảng được ưu tiên cho mối quan hệ bên nhánh trái, phải hoặc cả hai bên. Việc thay đổi thứ tự ưu tiên này sẽ làm ảnh hưởng đến kết quả truy vấn.

• **Từ khóa OUTER :** được dùng kết hợp cho các quan hệ ưu tiên dữ liệu. Tuy nhiên bạn được phép bỏ đi khi sử dụng loại quan hệ ưu tiên LEFT RIGHT FULL.

• **Điều kiện quan hệ :** là một biểu thức so sánh bằng, chỉ ra tên các cột quan hệ giữa hai bảng gần giống như biểu thức sau mệnh đề WHERE dùng để liên kết hai bảng.

Ví dụ :

Để hiển thị thông tin của các đơn đặt hàng trong bảng DONDH kèm theo cột họ tên nhà cung cấp tương ứng trong bảng NHACC và sắp xếp dữ liệu theo cột mã nhà cung cấp tăng dần. Bạn thực hiện câu lệnh SELECT FROM như sau :

```
SELECT NCC.MANHACC, TENNHACC, SODH
FROM DONDH DH INNER JOIN NHACC NCC
                        ON DH.MANHACC = NCC.MANHACC
ORDER BY NCC.MANHACC
```

Kết quả truy vấn trả về :

```
MANHACC    TENNHACC                SODH
-------------------------------------------------------------
C01        Lê Minh Trí             D002
C02        Trần Minh Thạch         D003
C02        Trần Minh Thạch         D005
C02        Trần Minh Thạch         D007
C03        Nguyễn Hồng Phương      D001
C05        Lưu Nguyệt Quế          D004
C05        Lưu Nguyệt Quế          D006

(7 row(s) affected)
```

Lưu ý :

Trong các truy vấn lấy dữ liệu từ nhiều bảng có quan hệ, bạn phải bắt buộc sử dụng định dạng : tên_bảng.tên_cột cho các cột trùng tên giữa các bảng. Theo ví dụ trên thì cột MANHACC xuất hiện ở cả hai bảng DONDH và NHACC do vậy bạn phải ghi : NCC.MANHACC và DH.MANHACC.

Ngoài ra bạn cũng có thể sử dụng khái niệm bí danh cho các bảng nhằm để làm ngắn gọn câu lệnh mỗi khi tham chiếu đến tên các bảng. Theo ví dụ trên bảng DONDH có bí danh là DH, bảng NHACC có bí danh là NCC. Các bí danh này bạn có thể đặt tên tùy thích, tuy nhiên bạn nên đặt ngắn gọn, gợi nhớ và không được phép trùng nhau bên trong một câu lệnh truy vấn.

Ví dụ :

Giống như ví dụ trên nhưng yêu cầu hiển thị ra tất cả các nhà cung cấp hiện có trong bảng NHACC. Để làm được điều này, bạn thấy rằng thứ tự quan hệ phải ưu tiên dữ liệu bên bảng NHACC. Bạn thực hiện câu lệnh SELECT FROM như sau :

```
SELECT NCC.MANHACC, TENNHACC, SODH
      FROM DONDH DH RIGHT OUTER JOIN NHACC NCC
                                ON DH.MANHACC=NCC.MANHACC
      ORDER BY NCC.MANHACC
```

Kết quả truy vấn trả về :

MANHACC	TENNHACC	SODH
C01	Lê Minh Trí	D002
C02	Trần Minh Thạch	D003
C02	Trần Minh Thạch	D005
C02	Trần Minh Thạch	D007
C03	Nguyễn Hồng Phương	D001
C04	**Trương Nhật Thắng**	**NULL**
C05	Lưu Nguyệt Quế	D004
C05	Lưu Nguyệt Quế	D006
C07	**Cao Minh Trung**	**NULL**

(9 row(s) affected)

Bạn thấy rằng có thêm hai nhà cung cấp mới trong kết quả truy vấn sau khi thay đổi thứ tự ưu tiên quan hệ dữ liệu cho bảng NHACC (RIGHT JOIN bởi vì bảng NHACC nằm bên phải trong quan hệ của bảng DONDH và NHACC). Tuy nhiên giá trị dữ liệu tại cột số đơn đặt hàng của hai nhà cung cấp này là NULL bởi vì công ty chưa bao giờ đặt hàng các nhà cung cấp này.

Ví dụ :

Hoàn toàn giống như ví dụ trên nhưng lần này bạn sẽ sử dụng thứ tự ưu tiên quan hệ dữ liệu bên trái. Bạn thực hiện câu lệnh SELECT FROM như sau :

```
SELECT NCC.MANHACC, TENNHACC, SODH
      FROM NHACC NCC LEFT JOIN DONDH DH
                              ON DH.MANHACC=NCC.MANHACC
      ORDER BY NCC.MANHACC
```

Bạn xem xét kết quả truy vấn trả về có gì khác so với ví dụ ở trên khi thay đổi thứ tự ưu tiên quan hệ dữ liệu bên phải (RIGHT JOIN) hay không ? Kết quả của hai câu lệnh truy vấn hoàn toàn như nhau. Tuy nhiên khi sử dụng thứ tự ưu tiên quan hệ dữ liệu bên trái trong ví dụ này đã thay đổi tên bảng NHACC ngay phía sau mệnh đề FROM để muốn chỉ định thứ tự ưu tiên lấy dữ liệu bên bảng NHACC.

Trong thực tế việc chọn lựa để sử dụng mệnh đề LEFT JOIN hoặc RIGHT JOIN là không quan trọng mà thay vào đó bạn phải hiểu rằng dữ liệu mà bạn cần chọn ra phải ưu tiên nằm bên trong bảng nào. Thông thường có một quy định là bảng nào ưu tiên dữ liệu sẽ được ghi ngay sau mệnh đề FROM, kế tiếp sử dụng mệnh đề LEFT JOIN chỉ định tên của bảng quan hệ cần lấy thông tin.

SQL Server 2000 : Các câu lệnh truy vấn dữ liệu – Lệnh SELECT FROM - Phần cuối

Với cú pháp SELECT FROM kết hợp mệnh đề JOIN của phần trình bày ở trên, bạn có liên kết tối đa 256 bảng dữ liệu trong một câu truy vấn. Một bảng có thể liên kết với nhiều bảng khác nhau trong một câu truy vấn. Các liên kết có thể được định nghĩa dựa trên các cột giống nhau của các bảng.

1.7/- Mệnh đề liên kết dữ liệu nhiều bảng :

Với cú pháp SELECT FROM kết hợp mệnh đề JOIN của phần trình bày ở trên, bạn có liên kết tối đa 256 bảng dữ liệu trong một câu truy vấn. Một bảng có thể liên kết với nhiều bảng khác nhau trong một câu truy vấn. Các liên kết có thể được định nghĩa dựa trên các cột giống nhau của các bảng.

Ví dụ :

Để biết được danh sách tên các vật tư đã đặt hàng trong tháng 01/2002, bạn phải lấy thông tin từ các bảng : VATTU (lấy cột tên vật tư), CTDONDH (lấy cột mã vật tư), DONDH (lấy cột ngày đặt hàng so sánh trong tháng 01/2002 và tạo quan hệ trung gian cho hai bảng VATTU và DONDH). Nhận xét thấy rằng trong truy vấn này dữ liệu cần lấy ra từ 3 bảng khác nhau nhưng có quan hệ.

Bạn thực hiện câu lệnh SELECT FROM như sau :

```
SELECT DISTINCT VT.MAVTU, TENVTU
FROM DONDH DH
        INNER JOIN CTDONDH CTDH
                ON DH.SODH = CTDH.SODH
        INNER JOIN VATTU VT
                ON VT.MAVTU = CTDH.MAVTU
WHERE MONTH(NGAYDH) = 01 AND YEAR(NGAYDH) = 2002
```

Kết quả truy vấn trả về :

```
MAVTU  TENVTU
------------------------------
DD01   Đầu DVD Hitachi 1 đĩa
DD02   Đầu DVD Hitachi 3 đĩa
VD02   Đầu VCD Sony 3 đĩa

(3 row(s) affected)
```

1.8/- Mệnh đề nối dữ liệu từ hai truy vấn :

Việc kết hợp dữ liệu của hai truy vấn SELECT FROM bằng mệnhd 9ề UNION cho phép bạn có thể tạo ra một tập hợp các mẫu tin từ các mẫu tin có trong câu lệnh SELECT FROM thứ nhất và các mẫu tin có trong câu lệnh SELECT FROM thứ hai. Khác với việc liên kết dữ liệu bằng mệnh đề JOIN, mệnh đề UNION thực ra chỉ thực hiện việc thêm vào các dòng dữ liệu trong câu lệnh SELECT FROM thứ nhất vào cuối các dòng dữ liệu trong câu lệnh SELECT FROM thứ hai.

Thông thường bạn sử dụng mệnh đề UNION dùng để nối dữ liệu từ các bảng khác nhau trong cơ sở dữ liệu thành một bộ các mẫu tin liên tục nhau. Các cột chỉ định trong hai câu lệnh SELECT FROM phải có cùng kiểu dữ liệu tương thích thứ tự như nhau, tổng số các cột phải bằng nhau. Việc định dạng tiêu đề của các cột tính toán chỉ cần thực hiện trong câu lệnh truy vấn đầu tiên.

Cú pháp :

```
SELECT Danh_sách_các_cột1
FROM Tên_bảng1
UNION
SELECT Danh_sách_các_cột2
FROM Tên_bảng2
[ORDER BY Danh_sách_cột sx]
```

Trong đó :

• **Mệnh đề UNION :** dùng để nối dữ liệu của hai truy vấn.

Ví dụ :

Để tính ra đồng thời tổng số lượng nhập, tổng số lượng xuất của các vật tư trong tháng 01/2002. Bạn thực hiện câu lệnh SELECT FROM như sau :

```
SELECT MAVTU, "X" AS LOAI, SUM(SLXUAT) AS TONGSL
FROM CTPXUAT CTPX INNER JOIN PXUAT PX
                                ON PX.SOPX = CTPX.SOPX
WHERE CONVERT(CHAR(7), NGAYXUAT, 21) = "2002-01"
GROUP BY MAVTU
UNION
SELECT MAVTU, "N", SUM(SLNHAP)
FROM CTPNHAP CTPN INNER JOIN PNHAP PN
                                ON PN.SOPN = CTPN.SOPN
WHERE CONVERT(CHAR(7), NGAYNHAP, 21) = "2002-01"
GROUP BY MAVTU
```

Kết quả truy vấn trả về :

```
MAVTU  LOAI    TONGSL
-------------------------------
DD01   N       10
DD02   N       15
VD02   N       30
DD01   X       6
VD02   X       10
DD02   X       7

(6 row(s) affected)
```

Nhận xét thấy rằng tổng số cột mà các truy vấn trả về sẽ là 3 cột, thứ tự kiểu dữ liệu của các cột phải tương thích nhau (chuỗi, số), việc định dạng tiêu đề cột chỉ thực hiện tại truy vấn SELECT FROM thứ nhất.

Bạn thấy rằng các dòng dữ liệu trong truy vấn thứ nhất sẽ được thêm vào cuối các dòng dữ liệu của truy vấn thứ hai. Tuy nhiên các mẫu tin hiển thị chưa được sắp xếp theo một thứ tự nào cả. Do đó nếu muốn các mẫu tin được sắp xếp lại theo một thứ tự nào đó thì bạn sẽ kết hợp mệnh đề ORDER BY vào cuối. Thực hiện lại truy vấn trên nhưng có kết hợp thêm mệnh đề ORDER By để thấy rõ số lượng nhập, số lượng xuất của từng vật tư trong tháng 01/2002.

```
SELECT MAVTU, "X" AS LOAI, SUM(SLXUAT) AS TONGSL
FROM CTPXUAT CTPX INNER JOIN PXUAT PX
                                ON PX.SOPX = CTPX.SOPX
WHERE CONVERT(CHAR(7), NGAYXUAT, 21) = "2002-01"
GROUP BY MAVTU
UNION
SELECT MAVTU, "N", SUM(SLNHAP)
FROM CTPNHAP CTPN INNER JOIN PNHAP PN
                                ON PN.SOPN = CTPN.SOPN
WHERE CONVERT(CHAR(7), NGAYNHAP, 21) = "2002-01"
GROUP BY MAVTU
ORDER BY MAVTU, LOAI
```

Kết quả truy vấn trả về :

```
MAVTU  LOAI    TONGSL
-------------------------------
DD01   N       10
DD01   X       6
DD02   N       15
DD02   X       7
VD02   N       30
VD02   X       10

(6 row(s) affected)
```

1.9/- Mệnh đề chép dữ liệu ra bảng mới :

Với cú pháp SELECT FROM bên dưới kết hợp mệnh đề INTO cho phép bạn sao chép dữ liệu và cấu trúc từ kết quả của một truy vấn cho ra một bảng dữ liệu mới bên trong cơ sở dữ liệu hiện hành hoặc các bảng dữ liệu tạm thời dùng để tính toán các xử lý phức tạp. Trong trường hợp nếu bạn muốn tạo ra bảng dữ liệu mới thì bắt buộc tên của bảng phải duy nhất trong cơ sở dữ liệu.

Bạn có thể chỉ định các ký tự dấu thăng (#) hoặc hai ký tự dấu thăng (##) phía trước tên bảng được tạo trong câu lệnh SELECT INTO dùng để tạo ra các bảng tạm cục bộ (local) hoặc các bảng tạm toàn cục (global). Bảng tạm cục bộ chỉ được sử dụng bởi người tao ra nó và hệ thống sẽ tự động hủy bỏ bảng tạm cục bộ khi người tạo ra bảng ngưng nối kết vào Microsoft SQL Server. Ngược lại bảng tạm toàn cục được sử dụng cho nhiều người khác nhau và hệ thống tự động hủy bảng tạm toàn cục khi không còn người sử dụng nào nối kết vào Microsoft SQL Server.

Cú pháp :

```
SELECT Danh_sách_các_cột INTO Tên_bảng_mới
FROM Tên_bảng_dl
```

Trong đó :

• **Tên bảng mới :** là tên của bảng mới sẽ được tạo lập có cấu trúc và dữ liệu từ truy vấn.

• **Tên bảng dữ liệu :** là tên của bảng chứa dữ liệu nguồn cho việc sao chép.

Ví dụ :

Để tạo ra bảng tạm cục bộ chứa thông tin thuế giá trị gia tăng (VAT) là 10% thành tiền của các phiếu nhập hàng trong tháng 01/2002. Bạn thực hiện câu lệnh SELECT INTO như sau :

```
SELECT PN.SOPN, THANH_TIEN = SUM(SLNHAP*DGNHAP),
            SUM(SLNHAP*DGNHAP)*0.1 AS THUE_VAT
                        INTO #THUE_PNHAP_200201
FROM CTPNHAP CTPN INNER JOIN PNHAP PN
                        ON PN.SOPN = CTPN.SOPN
WHERE CONVERT(CHAR(7), NGAYNHAP, 21) = "2002-01"
GROUP BY PN.SOPN
```

Sau đó thực hiện lệnh kế tiếp để xem dữ liệu hiện đang được lưu trữ trong bảng tạm #THUE_PNHAP_200201.

```
SELECT *
FROM #THUE_PNHAP_200201
```

Kết quả truy vấn trả về :

```
SOPN THANH_TIEN      THUE_VAT
-------------------------------------------------
N001 55000000.0000   5500000.00000
N002 22500000.0000   2250000.00000
N003 75000000.0000   7500000.00000

(3 row(s) affected)
```

Bạn tạm thời ngắt nối kết vào Microsoft SQL Server và sau đó thực hiện nối kết lại vào Micorsoft SQL Server, thực hiện lại truy vấn xem dữ liệu của bảng tạm cục bộ #THUE_PNHAP_200201 đã tạo trước đó.

(Bảng tạm #THUE_PNHAP_200201 đã không còn vì hệ thống đã tự động hủy bỏ bảng tạm cục bộ khi bạn ngưng nối kết vào Microsoft SQL Server).

1.10/- Mệnh đề thống kê dữ liệu :

Với cú pháp SELECT FROM bên dưới kết hợp mệnh đề COMPUTE cho phép bạn có thể tạo ra dòng thống kê dữ liệu ở bên cuối kết quả truy vấn. Tuy nhiên nếu bạn sử dụng thêm mệnh đề COMPUTE BY tiếp theo thì hệ thống sẽ thống kê dữ liệu theo từng nhóm dữ liệu.

Cú pháp :

```
SELECT Danh_sách_các_cột
FROM Tên_bảng
COMPUTE COUNT|MIN|MAX|SUM|AVG (Tên_cột)
        [ BY Tên_cột_nhóm ]
```

Trong đó :

• **Count, Min, Max, Sum, Avg :** là các hàm thống kê tính toán dữ liệu mà kết quả sẽ xuất hiện ở cuối kết quả truy vấn hoặc từng nhóm dữ liệu.

• **Tên cột :** tên các cột hoặc biểu thức được tính toán kèm với các hàm thống kê chỉ định trước đó.

Ví dụ :

Để hiển thị thông tin chi tiết các vật tư đã đặt hàng cho các nhà cung cấp có mã là "C02" hoặc "C03". Có thống kê tổng số lượng đặt, số lượng nhiều nhất, số lượng đặt ít nhất trên kết quả truy vấn. Bạn thực hiện các câu lệnh SELECT FROM như sau :

```
SELECT DH.SODH, VT.VATTU, TENVTU, SLDAT, MANHACC
FROM CTDONDH CTDH
      INNER JOIN VATTU VT ON CTDH.MAVTU = VT.MAVTU
      JOIN DONDH DH ON DH.SODH = CTDH.SODH
WHERE MANHACC IN ("C02", "C03")
COMPUTE SUM(SLDAT), MAX(SLDAT), MIN(SLDAT)
```

Kết quả truy vấn trả về :

```
SODH MAVTU   TENVTU                SLDAT  MANHACC
-------------------------------------------------------------------------
D001 DD01    Đầu DVD Hitachi 1 đĩa 10     C03
D001 DD02    Đầu DVD Hitachi 3 đĩa 15     C03
D003 TV14    Tivi Sony 14 inches   10     C02
D003 TV29    Tivi Sony 29 inches   20     C02
D005 TV14    Tivi Sony 14 inches   10     C02
D005 TV29    Tivi Sony 29 inches   20     C02

                    Sum
                    =============
                    85

                    Max
                    =============
                    20

                    Min
                    =============
                    10

(7 row(s) affected)
```

Ví dụ :

Theo ví dụ trên nhưng bạn muốn thống kê theo từng nhà cung cấp. Lúc này bạn bắt buộc sử dụng mệnh đề COMPUTE BY tuy nhiên cần kết hợp với mệnh đề ORDER BY. Bạn thực hiện các câu lệnh SELECT FROM như sau :

```
SELECT DH.SODH, VT.MAVTU, TENVTU, SLDAT, MANHACC
FROM CTDONDH CTDH
      INNER JOIN VATTU VT ON CTDH.MAVTU = VT.MAVTU
      JOIN DONDH DH ON DH.SODH = CTDH.SODH
WHERER MANHACC In ("C02", "C03")
ORDER BY MANHACC
COMPUTE SUM(SLDAT) BY MANHACC
```

Kết quả truy vấn trả về :

```
SODH MAVTU   TENVTU                 SLDAT  MANHACC
-------------------------------------------------------
D003 TV14    Tivi Sony 14 inches    10     C02
D003 TV29    Tivi Sony 29 inches    20     C02
D005 TV14    Tivi Sony 14 inches    10     C02
D005 TV29    Tivi Sony 29 inches    20     C02

                    Sum
                    =============
                    60
D001 DD01    Đầu DVD Hitachi 1 đĩa  10     C03
D001 DD02    Đầu DVD Hitachi 3 đĩa  15     C03

                    Sum
                    =============
                    25

(8 row(s) affected)
```

SQL Server 2000 : Các câu lệnh truy vấn dữ liệu – Biểu thức CASE

Biểu thức CASE trong Transaction-SQL vô cùng hữu ích. Hoạt động của biểu thức CASE rất đơn giản chỉ là thực hiện việc so sánh một biểu thức bất kỳ với hàng loạt các giá trị chỉ định trước đó, nếu bạn là người lập trình trong môi trường Visual Basic thì biểu thức CASE của Transaction-SQL gần giống như cấu trúc điều khiển Select Case.

Tuy nhiên biểut hức CASE hoàn toàn không phải là một cấu trúc điều khiển, điều này có nghĩa là nó chỉ được sử dụng lồng vào các câu lệnh khác mà không thể thực hiện đơn lẻ như các cấu trúc điều khiển khác. Biểu thức CASE có thể sử dụng ở hai dạng khác nhau.

Cú pháp CASE dạng đơn giản :

```
CASE Biểu_thức
        WHEN Giá_trị_1 THEN Biểu_thức_kết_quả_1
        [WHEN Giá_trị_2 THEN Biểu_thức_kết_quả_2
          ...]
        [ELSE Biểu_thức_kết_quả_N]
END
```

Trong đó :

• **Biểu thức :** biểu thức tính toán hoặc tên cột dữ liệu của bảng được dùng để so sánh.

• **Giá trị 1, giá trị 2 :** là các giá trị cụ thể để so sánh bằng (=) với biểu thức.

• **Biểu thức kết quả 1, biểu thức kết quả 2 :** là các biểu thức sẽ được trả về khi việc so sánh của biểu thức bằng với các giá trị so sánh tương ứng.

• **Biểu thức kết quả N :** là biểu thức sẽ được trả về khi tất cả các trường hợp so sánh đều không bằng với các giá trị đưa ra.

Ví dụ :

Để hiển thị danh sách các vật tư có trong bảng VATTU theo từng loại hàng, có đếm tổng số các vật tư theo từng loại hàng. Bạn sử dụng lệnh SELECT FROM có kết hợp biểu thức CASE đơn giản như sau :

```
SELECT LOAI =
        CASE LEFT(MAVTU, 2)
          WHEN "DD" THEN "Đầu DVD"
          WHEN "VD" THEN "Đầu VCD"
          WHEN "TV" THEN "Tivi"
          WHEN "TL" THEN "Tủ lạnh"
          WHEN "BI" THEN "Bia lon"
          WHEN "LO" THEN "Loa thùng"
          ELSE "Chưa phân loại"
        END, MAVTU, TENVTU, DVTINH
FROM VATTU
ORDER BY LEFT(MAVTU, 2)
COMPUTER COUNT(MAVTU) BY LEFT(MAVTU, 2)
```

Kết quả truy vấn trả về :

```
LOAI        MAVTU     TENVTU                  DVTINH
-------------------------------------------------------------------------
Bia lon     BI01      Bia lon Tiger           Thùng
        cnt
        ============
        1
Đầu DVD     DD01      Đầu DVD Hitachi 1 đĩa   Bộ
Đầu DVD     DD02      Đầu DVD Hitachi 3 đĩa   Bộ
        cnt
        ============
        2
Loa thùng   LO01      Loa Panasonic 1000W     Bộ
        cnt
        ============
        1
Tủ lạnh     TL15      Tủ lạnh Sanyo 150 lít   Cái
Tủ lạnh     TL90      Tủ lạnh Sanyo 90 lít    Cái
        cnt
        ============
        2
Tivi        TV14      Tivi Sony 14 inches     Cái
Tivi        TV21      Tivi Sony 21 inches     Cái
Tivi        TV29      Tivi Sony 29 inches     Cái
        cnt
        ============
        3
Đầu VCD     VD01      Đầu VCD Sony 1 đĩa      Bộ
Đầu VCD     VD02      Đầu VCD Sony 3 đĩa      Bộ
        cnt
        ============
        2
(17 row(s) affected)
```

Cú pháp CASE dạng tìm kiếm :

```
CASE
        WHEN Bt_logic_1 THEN Biểu_thức_kết_quả_1
        [WHEN Bt_logic_2 THEN Biểu_thức_kết_quả_2
          ...]
        [ELSE Biểu_thức_kết_quả_N]
END
```

Trong đó :

• **Biểu thức logic1, biểu thức logic2 :** là các biểu thức luận lý dùng để thực hiện các phép so sánh trong biểu thức CASE.

• **Biểu thức kết quả 1, biểu thức kết quả 2 :** là các biểu thức sẽ được trả về khi một trong các biểu thức luận lý so sánh có kết quả là đúng.

• **Biểu thức kết quả N :** là biểu thức sẽ được trả về khi tất cả các biểu thức logic so sánh đưa ra đều sai.

Ví dụ :

Để hiển thị danh sách các vật tư có trong bảng VATTU, thông tin bổ sung thêm chuỗi ghi chú, tùy thuộc vào giá trị của cột tỷ lệ phần trăm giá bán. Bạn sử dụng lệnh SELECT FROM có kết hợp biểu thức CASE tìm kiếm như sau :

```
SELECT GHICHU=
      CASE
          WHEN PHANTRAM < 20 THEN "Lời ít"
          WHEN PHANTRAM BETWEEN 20 AND 40 THEN "Lời nhiều"
          ELSE "Rất lời"
      END, TENVTU, DVTINH, PHANTRAM
FROM VATTU
ORDER BY PHANTRAM
```

Kết quả truy vấn trả về :

```
GHICHU      TENVTU                DVTINH    PHANTRAM
-------------------------------------------------------------------------------
Lời ít      Loa Panasonic 100W    Bộ            10
Lời ít      Tivi Sony 21 inches   Cái           15
Lời ít      Tivi Sony 29 inches   Cái           15
Lời nhiều   Tủ lạnh Sanyo 90 lít  Cái           20
Lời nhiều   Tivi Sony 14 inches   Cái           20
Lời nhiều   Tủ lạnh Sanyo 150 lít Cái           25
Lời nhiều   Đầu VCD Sony 1 đĩa    Bộ            30
Lời nhiều   Đầu VCD Sony 3 đĩa    Bộ            30
Lời nhiều   Đầu DVD Hitachi 1 đĩa Bộ            40
Lời nhiều   Đầu DVD Hitachi 3 đĩa Bộ            40
Rất lời     Bia lon Tiger         Thùng         60

(11 row(s) affected)
```

Thực tế thì nhưng người lập trình trong môi tường Transaction-SQL thường sử dụng biểu thức CASE tìm kiếm bởi vì khi đó các biểu thức luận lý mà bạn dùng để so sánh được

phép chứa nhiều toán tử so sánh khác nhau, trong khi đó biểu thức CASE đơn giản ở phần trên chỉ cho phép bạn thực hiện phép so sánh bằng (=) trên một biểu thức đơn giản.

Ví dụ :

Để giảm giá bán hàng trong tháng 02/2002 theo quy tắc :

• Nếu số lượng hàng <= 2 thì không giảm giá.

• Nếu số lượng hàng từ 3 đến 10 thì giảm 10%.

• Nếu số lượng hàng > 10 thì giảm 20%.

Bạn sử dụng lệnh UPDATE SET có kết hợp biểu thức CASE tìm kiếm như sau :

```
UPDATE CTPXUAT
SET DGXUAT=
      CASE
        WHEN SLXUAT <= 2 THEN DGXUAT
        WHEN SLXUAT BETWEEN 3 AND 10 THEN DGXUAT*0.9
        ELSE DGXUAT*0.8
      END
FROM CTPXUAT CTPX
      INNER JOIN PXUAT PX ON PX.SOPX=CTPX.SOPX
WHERE CONVERT(CHAR(7), NGAYXUAT, 21)="2002-02"
```

Tóm lại biểu thức CASE có thể được phép kết hợp sử dụng trong các câu lệnh SELECT, UPDATE SET, DELETE dùng để biện luận các trường hợp khác nhau của các giá trị dữ liệu bên trong một câu lệnh truy vấn.

SQL Server 2000 : Các câu lệnh truy vấn dữ liệu – Truy vấn con

Trong khi lập trình bên trong Transaction-SQL, có đôi lúc bạn sẽ sử dụng đến truy vấn con đề tính toán dữ liệu. Truy vấn con chỉ là một câu lệnh truy vấn chọn lựa (SELECT) được lồng vào các câu lệnh truy vấn khác nhằm thực hiện các truy vấn tính toán phức tạp.

Khi sử dụng đến truy vấn con, bạn cần lưu tâm đến một vài yếu

tố sau :

• Cần mở và đóng ngoặc đơn cho câu lệnh truy vấn con.

• Bạn chỉ được phép tham chiếu đến tên một cột hoặc một biểu thức sẽ trả về giá trị trong truy vấn con.

• Kết quả của truy vấn con có thể trả về là một giá trị đơn lẻ hoặc một danh sách các giá trị.

• Cấp độ lồng nhau của các truy vấn con bên trog Microsoft SQL Server là không giới hạn.

1/- Truy vấn con trả về một giá trị đơn :

Là các truy vấn mà kết quả trả về của nó luôn luôn đảm bảo chỉ là một giá trị đơn. Thông thường các truy vấn dạnh này sẽ sử dụng các hàm tính toán thống kê dữ liệu.

Ví dụ :

Để biết được danh sách các đơn đặt hàng gần đây nhất. Trước tiên bạn phải tính ra được ngày đặt hàng gần đây là bao nhiêu bằng câu lệnh truy vấn như sau :

```
SELECT MAX(NGAYDH)
FROM DONDH
```

Kết quả truy vấn trả về :

```
------------------------------
2002-03-15 00:00:00.000

(6 row(s) affected)
```

Sau đó bạn sẽ lọc ra danh sách các đơn đặt hàng có ngày đặt hàng là ngày "2002-03-15". Thực hiện truy vấn như sau :

```
SELECT *
FROm DONDH
WHERE NGAYDH = "2002-03-15"
```

Tuy nhiên không thể nào chắc rằng ngày đặt hàng gần nhất trong bảng DONDH luôn là "2002-03-15".

Do đó câu lệnh truy vấn ở trên chỉ đúng tại thời điểm này mà thôi. Để đảm bảo rằng bạn luôn có được danh sách các đơn đặt hàng gần đây nhất, bạn sẽ kết hợp cả hai câu truy vấn đã thực hiện ở trên như sau :

```
SELECT *
FROM DONDH
WHERE NGAYDH = (SELECT MAX(NGAYDH) FROM DONDH)
```

Nhận xét thấy rằng câu lệnh truy vấn con chỉ sử dụng một hàm tính toán thống kê là MAX nên kết quả luôn luôn trả về một giá trị đơn.

Ví dụ :

Muốn biết tổng số lượng đã đặt hàng của từng vật tư. Bạn thực hiện câu truy vấn như sau :

```
SELECT TENVTU, SUM(SLDAT) AS TONGSLDAT
FROM VATTU VT
        INNER JOIN CTDONDH CTDH ON CTDH.MAVTU=VT.MAVTU
GROUP BY TENVTU
```

Kết quả truy vấn trả về :

```
TENVTU                  TONGSLDAT
--------------------------------------------
Đầu DVD Hitachi 1 đĩa   10
Đầu DVD Hiatchi 3 đĩa   15
Đầu VCD Sony 1 đĩa      20
Đầu VCD Sony 3 đĩa      30
Tivi Sony 14 inches     30
Tivi Sony 29 inches     60
Tủ lạnh Sanyo 90 lít    10

(7 row(s) affected)
```

Với câu lệnh truy vấn bên dưới sẽ trả về tổng cộng số lượng đặt hàng của tất cả các vật tư có trong bảng CTDONDH.

```
SELECT SUM(SLDAT) AS TONGCONG
FROM CTDONDH
```

Bạn kết hợp hai câu truy vấn trên để biết được tỉ lệ phần trăm số lượng đặt hàng của từng vật tư trên tổng cộng các số lượng đặt hàng của toàn bộ các vật tư.

```
SELECT TENVTU, SUM(SLDAT) AS TONGSLDAT,
       (SELECT SUM(SLDAT) FROM CTDONDH) AS TONGCONG,
       ((CONVERT(MONEY, SUM(SLDAT)))/
       (SELECT SUM(SLDAT) FROM CTDONDH))*100) AS PHTRAM
FROM VATTU VT
       INNER JOIN CTDONDH CTDH ON CTDH.MAVTU=VT.MAVTU
GROUP BY TENVTU
```

Kết quả truy vấn trả về :

```
TENVTU                TONGSLDAT  TONGCONG PHTRAM
--------------------------------------------------------------------------
Đầu DVD Hitachi 1 đĩa 10         175      5.7100
Đầu DVD Hiatchi 3 đĩa 15         175      8.5700
Đầu VCD Sony 1 đĩa    20         175      11.4200
Đầu VCD Sony 3 đĩa    30         175      17.1400
Tivi Sony 14 inches   30         175      17.1400
Tivi Sony 29 inches   60         175      34.2800
Tủ lạnh Sanyo 90 lít  10         175      5.7100

(7 row(s) affected)
```

2/- Truy vấn con trả về danh sách các giá trị :

Là các truy vấn mà kết quả trả về của nó là một danh sách các giá trị hay còn gọi là một tập hợp các phần tử. Thông thường các truy vấn con dạng này sẽ lấy dữ liệu của một hoặc nhiều bảng khác thực hiện việc so sánh trong mệnh đề WHERE của truy vấn cha. Toán tử IN sẽ được sử dụng để so sánh trong truy vấn con dạng này bởi vì nó dùng chỉ định việc so sánh một phần tử có thuộc trong một tập hợp các phần tử hay không.

Ví dụ :

Để biết các nhà cung cấp nào mà công ty đã đặt hàng trong tháng 01/2002. Bạn thực hiện câu truy vấn như sau :

```
SELECT MANHACC
FROM DONDH
WHERE CONVERT(CHAR(7), NGAYDH, 21)="2002-01"
```

Kết quả truy vấn trả về :

```
MANHACC
-----------
C03
C01

(2 row(s) affected)
```

Tuy nhiên thông tin mà bạn muốn hiển thị đầy đủ sẽ là họ tên các nhà cung cấp chứ không phải là mã nhà cung cấp. Do thế bạn sẽ sử dụng truy vấn như sau :

```
SELECT TENNHACC, DIENTHOAI
        FROM NHACC
        WHERE MANHACC IN ("C01", "C03")
```

Đâu đảm bảo rằng trong tháng 01/2002 công ty chỉ đặt hàng cho hai nhà cung cấp C01 và C03. Do thế để luôn luôn có được danh sách họ tên các nhà cung cấp mà công ty đã đặt hàng trong tháng 01/2002, bạn thực hiện truy vấn con như sau :

```
SELECT TENNHACC, DIENTHOAI
        FROM NHACC
        WHERE MANHACC IN
                (SELECT MANHACC
                FROM DONDH
                WHERE CONVERT(CHAR(7), NGAYDH, 21)="2002-01")
```

Kết quả truy vấn trả về :

```
TENNHACC                DIENTHOAI
----------------------------------------
Lê Minh Trí             8781024
Nguyễn Hồng Phương      9600125

(2 row(s) affected)
```

Bạn có thể sử dụng từ khóa EXISTS hoặc mệnh đề JOIN đã thực hiện việc so sánh các dòng dữ liệu trong các truy vấn con trả về danh sách các giá trị. Từ khóa EXISTS dùng để kiểm tra tính tồn tại của dữ liệu, ngay sau EXISTS là một câu lệnh SELECT mà kết quả trả về của nó là một tập hợp trống hoặc có chứa nhiều phần tử. Hai câu lệnh truy vấn bên dưới đều có kết quả trả về như câu lệnh truy vấn ở ví dụ bên trên.

```
SELECT TENNHACC, DIENTHOAI
FROM NHACC NCC
WHERE EXISTS
        (SELECT *
        FROM DONDH DH
        WHERE CONVERT(CHAR(7), NGAYDH, 21)="2002-01"
                        AND DH.MANHACC=NCC.MANHACC)
```

Hoặc

```
SELECT TENNHACC, DIENTHOAI
        FROM NHACC NCC
        INNER JOIN DONDH DH
        ON DH.MANHACC=NCC.MANHACC
        WHERE CONVERT(CHAR(7), NGAYDH, 21)="2002-01"
```

Nếu muốn sử dụng các toán tử so sánh bình thường (=, >, <, <>, …) trong truy vấn con trả về danh sách các giá trị thì bắt buộc bạn phải kết hợp các từ khóa ANY, ALL phía trước câu lệnh truy vấn con. Bạn nên nhớ một quy tắc như sau :

"IN sẽ tương đương = ANY và NOT IN sẽ tương đương <> ALL"

Ví dụ :

Để biết danh sách các nhà cung cấp nào mà công ty chưa bao giờ đặt hàng. Bạn có thể thực hiện câu truy vấn như sau :

```
SELECT TENNHACC, DIENTHOAI
FROM NHACC
WHERE MANHACC NOT IN
                        (SELECT DISTINCT MANHACC
                        FROM DONDH)
```

Hoặc

```
SELECT TENNHACC, DIENTHOAI
        FROM NHACC
        WHERE MANHACC <> ALL
                        (SELECT DISTINCT MANHACC
                        FROM DONDH)
```

Kết quả truy vấn trả về :

```
TENNHACC              DIENTHOAI
--------------------------------------------
Trương Nhật Thắng     8757757
Cao Minh Trung        Chưa có

(2 row(s) affected)
```

Tuy nhiên nếu bạn hiểu sai các quy tắc trên "NOT IN sẽ tương đương <> ANY". Khi đó với câu truy vấn bên dưới sẽ trả về kết quả sai hoàn toàn.

```
SELECT TENNHACC, DIENTHOAI
FROM NHACC
WHERE MANHACC <> ANY
                      (SELECT DISTINCT MANHACC
                      FROM DONDH)
```

Kết quả truy vấn trả về :

```
TENNHACC              DIENTHOAI
--------------------------------------------
Lê Minh Trí           8781024
Trần Minh Thạch       7698154
Nguyễn Hồng Phương    9600125
Trương Nhật Thắng     8757757
Lưu Nguyệt Quế        7964251
Cao Minh Trung        Chưa có

(6 row(s) affected)
```

Các từ khóa ALL, ANY trong các truy vấn con nhằm giúp bạn hiểu thêm trong thực hiện việc so sánh dữ liệu của các truy vấn con với truy vấn cha, tuy nhiên bạn chỉ cần nhớ các toán tử IN hoặc NOT IN và sử dụng cho đúng trong các trường hợp so sánh cần thiết đối với dạng truy vấn con trả về danh sách các giá trị.

SQL Server 2000 : Các câu lệnh truy vấn dữ liệu – Lệnh UPDATE SET

Các dữ liệu sau khi được lưu trữ vào bên trong bảng chưa đảm bảo rằng các giá trị đó sẽ đúng mãi mãi. Đôi khi các giá trị này cũng cần phải được thay đổi lại cho đúng theo một yêu cầu nào đó. Bằng tiện ích Enterprise Manager, bạn có thể thay đổi trực tiếp các giá trị tại các dòng dữ liệu hiện có bên trong bảng trên màn hình hiển thị các dòng dữ liệu.

Bên cạnh đó trong các trường hợp cần sửa đổi đồng thời nhiều dòng dữ liệu bên trong một bảng, bạn sẽ sử dụng lệnh UPDATE SET. Tuy nhiên khi sử dụng lệnh này bạn không thể khôi phục lại các giá trị sau khi thay đổi. Cú pháp đầy đủ của lệnh UPDATE SET được mô tả như sau :

Cúp pháp :

```
UPDATE Tên_bảng
SET Tên_cột = Biểu_thức [, ...]
[FROM Tên_bảng1
      INNER | LEFT | RIGHT JOIN Tên_bảng2 ON
      Biểu_thức_liên_kết]
[WHERE Điều_kiện_sửa_đổi]
```

Trong đó :

• **Tên bảng :** tên bảng có chứa các dòng dữ liệu muốn sửa đổi.

• **Tên cột :** tên cột muốn sửa đổi giá trị dữ liệu. Bạn có thể thay đổi giá trị của nhiều cột bên trong một bảng trong cùng một câu lệnh UPDATE SET.

• **Biểu thức :** là một giá trị cụ thể hoặc một hàm tính toán mà giá trị trả về của nó sẽ được cập nhật vào tên cột trong bảng chỉ định trước đó.

• **Tên bảng1, tên bảng2 :** tên các bảng có quan hệ dữ liệu, được dùng để kết nối quan hệ trong khi sửa đổi dữ liệu.

• **Điều kiện sửa đổi :** là biểu thức luận lý chỉ định các dòng dữ liệu phải thỏa điều kiện đưa ra thì mới bị sửa đổi.

Lưu ý :

Trong lệnh UPDATE nếu không có sử dụng mệnh đề WHERE thì tất cả các mẫu tin trong bảng sẽ bị sửa đổi.

Ví dụ :

Để có được tổng trị giá của các phiếu nhập hàng. Bạn có thể thêm một cột mới có tên TGNHAP (trị giá nhập) trong bảng PNHAP. Sử dụng lệnh ALTER TABLE để thêm vào một cột như sau :

```
ALTER TABLE PNHAP
      ADD TGNHAP MONEY
```

Sau đó sử dụng lệnh UPDATE SET có kết hợp truy vấn con để tính ra tổng giá trị nhập dựa vào giá trị dữ liệu của các cột SLNHAP và TGNHAP trong bảng CTPNHAP theo từng số phiếu nhập hàng.

```
UPDATE PNHAP
      SET TGNHAP = 0
GO
UPDATE PNHAP
SET TGPNHAP = (SELECT SUM(SLNHAP*DGNHAP)
                FROM CTPNHAP CTPN
                WHERE PN.SOPN = CTPN.SOPN)
FROM PNHAP PN
GO
```

Cuối cùng kiểm tra lại kết quả sau khi đã thực hiện cập nhập tính giá trị cho cột trị giá nhập (TGNHAP).

```
SELECT *
FROM PNHAP
```

Kết quả truy vấn trả về :

```
Sopn  Sodh  Ngaynhap                TGNHAP
-------------------------------------------------------------------
N001  D001  2002-01-17 00:00:00.000   55000000.0000
N002  D001  2002-01-20 00:00:00.000   22500000.0000
N003  D002  2002-01-31 00:00:00.000   75000000.0000

(3 row(s) affected)
```

Ví dụ :

Đề giảm giá 10% cho tất cả các phiếu bán hàng trong ngày cuối cùng của tháng 01/2002. Bạn sử dụng lệnh UPDATE SET như sau :

```
UPDATE CTPXUAT
SET DGXUAT = DGXUAT*0.9
FROM PXUAT PX
        INNER JOIN CTPXUAT CTPX ON PX.SOPX=CTPX.SOPX
WHERE NGAYXUAT="2002-01-31"
```

Tóm lại việc cập nhật các dữ liệu trong bảng bao gồm các hành động thêm, hủy và sửa đổi dữ liệu. Các hành động này chỉ tác động đến dữ liệu bên trong của một và chỉ một bảng mà thôi. Tuy nhiên bên trong các lệnh INSERT … SELECT, DELETE, UPDATE SET, bạn có thể tham chiếu đến một hoặc nhiều bảng khác để tạo ra các kết nối quan hệ nhằm lấy ra thông tin của các bảng khác dùng trong các điều kiện so sánh bên trong mệnh đề WHERE.

Bạn nên thận trọng khi sử dụng lệnh DELETE và UPDATE SET vì khi đó bạn sẽ không thể phục hồi lại dữ liệu cũ trước đó. Trước khi thực hiện các lệnh này, bạn nên tạo ra bảng dữ liệu dự phòng (backup) bằng lệnh SELECT INTO hoặc chèn thêm các cột tạm vào bên trong bảng để chứa giá trị trước khi thay đổi. Ngoài ra nếu hiểu rõ chế độ giao tác (transaction) là gì thì bạn nên thực hiện các hành động cập nhật dữ liệu bên trong các giao tác bởi vì trong chế độ này bạn có thể phục hồi lại các giá trị dữ liệu đã bị cập nhật bên trong bảng.

SQL Server 2000 : Các câu lệnh truy vấn dữ liệu – Lệnh DELETE FROM

Trái ngược với hành động thêm mới dữ liệu vào bảng, bạn có thể hủy bỏ các dòng dữ liệu hiện đang có trong bảng khi không còn sử dụng nữa. Với hành động này, bạn không thể khôi phục lại các dữ liệu sau khi đã ra lệnh hủy bỏ nó. Lệnh DELETE FROM bên dưới cho phép bạn hủy bỏ các dòng dữ liệu hiện đang có bên trong một bảng.

Cú pháp :

```
DELETE [FROM] Tên_bảng
        [FROM Tên_bảng1
        INNER | LEFT | RIGHT JOIN Tên_bảng2
        ON Biểu_thức_liên_kết]
        [WHERE Điều_kiện_xóa dl]
```

Trong đó :

• **Tên bảng :** tên bảng có các dòng dữ liệu muốn hủy bỏ.

• **Tên bảng1, tên bảng2 :** tên các bảng có quan hệ dữ liệu, được dùng để kết nối các quan hệ nhằm tra cứu các thông tin trong khi xóa dữ liệu.

• **Điều kiện xóa dữ liệu :** là biểu thức luận lý chỉ định các dòng dữ liệu phải thỏa điều kiện đưa ra thì mới bị hủy bỏ.

Lưu ý :

Trong lệnh DELETE nếu quên không sử dụng mệnh đề WHERE thì tất cả các dòng dữ liệu hiện đang có bên trong bảng dữ liệu sẽ bị hủy tất cả.

Ví dụ :

Để hủy bỏ các nhà cung cấp mà công ty chưa bao giờ đặt hàng. Bạn sử dụng lệnh DELETE như sau :

```
DELETE NHACC
        FROM NHACC NCC
        LEFT JOIN DONDH DH
        ON DH.MANHACC=NCC.MANHACC
        WHERE DH.SODH IS NULL
```

Nhận xét thấy rằng trong ví dụ này sử dụng mệnh đề LEFT JOIN để thay đổi ưu tiên quan hệ dữ liệu bên bảng NHACC, kế tiếp trong mệnh đề WHERE DH.SODH IS NULL dùng để chỉ định việc xóa đi những nhà cung cấp nào chưa có số đặt hàng (số đặt hàng đang là trống).

Bạn cũng có thể sử dụng truy vấn con để thực hiện hành động hủy bỏ các nhà cung cấp trong bảng NHACC mà công ty chưa bao giờ đặt hàng bằng câu lệnh như sau :

```
DELETE NHACC
        WHERE MANHACC NOT IN
        (SELECT DISTINCT MANHACC
        FROM DONDH)
```

Nhận xét thấy rằng kết quả của truy vấn con sẽ trả về tập hợp các nhà cung cấp mà công ty đã đặt hàng, sau đó trong mệnh đề WHERE MANHACC NOT IN dùng để xác định các nhà cung cấp không nằm trong tập hợp các nhà cung cấp đã được đặt hàng.

Ví dụ :

Để hủy bỏ các đơn đặt hàng trong tháng 01/2002. Bạn sử dụng lệnh DELETE như sau :

```
DELETE DONDH
        WHERE CONVERT(CHAR(7), NGAYDH, 21) = "2002-01"
```

Hệ thống Microsoft SQL Server sẽ xuất hiện thông báo lỗi vì việc xóa dữ liệu trong bảng DONDH sẽ vi phạm ràng buộc khóa ngoại bên bảng CTDONDH.

```
Server : Msg 547, Level 16, State 1, Line 1
DELETE statement conflicted with COLUMN REFERENCE constraint
'FRK_CTDONDH_SODH'. The conflict occurred in database
'QLBanHang', table 'CTDONDH', column 'Sodh'.
The statement has been terminated.
```

Thông thường khi hủy bỏ các dòng dữ liệu hiện có bên trong một bảng nào đó nếu bạn vô tình vi phạm các ràng buộc toàn vẹn dữ liệu về khóa ngoại đã được định nghĩa trước đó thì tuyệt đối dữ liệu sẽ không bị hủy ra khỏi bảng. Muốn tránh những sai sót này, trước tiên bạn cần phải hủy bỏ dữ liệu bên nhánh quan hệ nhiều (nhánh quan hệ con) trước. Do đó, theo ví dụ trên, bạn cần phải xóa đi chi tiết các đơn đặt hàng có liên quan trong tháng 01/2002 bên bảng CTDONDH. Thực hiện truy vấn như sau :

```
DELETE CTDONDH
        FROM CTDONDH CTDH
        INNER JOIN DONDH DH
        ON DH.SODH = CTDH.SODH
        WHERE CONVERT(CHAR(7), NGAYDH, 21) = "2002-01"
```

Trong câu lệnh truy vấn này, bạn muốn xóa các dòng dữ liệu trong bảng CTDONDH, tuy nhiên cần phải liên kết thêm bảng DONDH vào để có thời gian chỉ lọc ra các đơn đặt hàng trong tháng 01/2002.

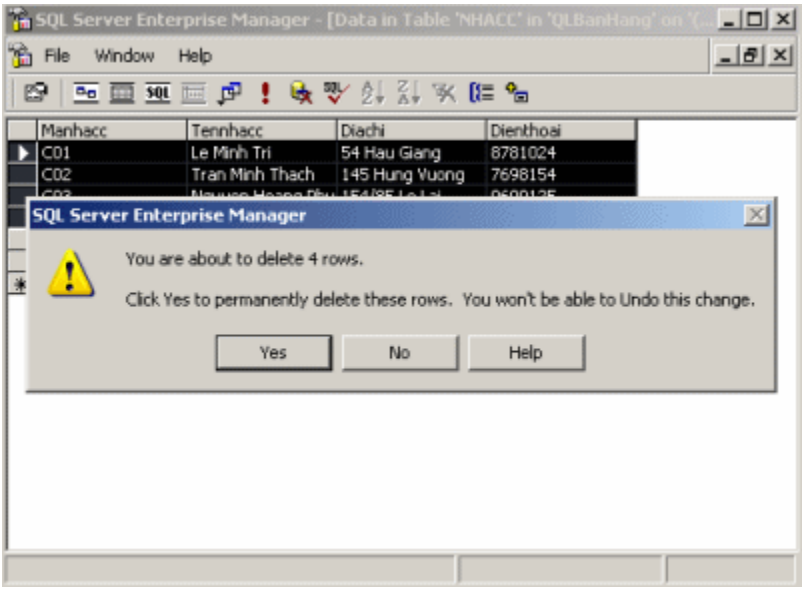
Sau đó tiếp tục cho thực hiện lại truy vấn hủy bỏ các đơn đặt hàng trong tháng 01/2002 ở ví dụ trước đó.

Kết quả truy vấn chỉ trả về thành công khi nòa các đơn đặt hàng trong tháng 01/2002 chưa được nhập hàng về (dữ liệu liên quan của các đơn đặt hàng chưa có trong bảng PNHAP), ngược lại khi các đơn đặt hàng này đã có nhập hàng về rồi thì hệ thống sẽ xuất hiện thông báo. Bởi vì bảng PNHAP đã định nghĩa có quan hệ khóa ngoại với bảng DONDH theo cột số đặt hàng trước đó.

```
Server : Msg 547, Level 16, State 1, Line 1
DELETE statement conflicted with COLUMN REFERENCE constraint
'FRK_PNHAP_SODH'. The conflict occurred in database
'QLBanHang', table 'PNHAP', column 'Sodh'.
The statement has been terminated.
```

Tóm lại việc hủy bỏ dữ liệu bên trong một bảng bằng lệnh DELETE có thể cùng lúc sẽ hủy bỏ được nhiều dòng thỏa điều kiện đưa ra. Kết quả của lệnh truy vấn này đôi khi có thể gây ra lỗi nếu chúng có vi phạm các ràng buộc toàn vẹn khóa ngoại bên nhánh quan hệ dữ liệu nhiều.

Ngoài ra bạn cũng có thể sử dụng tiện ích Enterprise Manager để hủy bỏ các dòng dữ liệu hiện có bên trong bảng. Chọn chức năng Open Table (Return all rows trong thực đơn tắt sau khi nhấn chuột phải trên tên bảng muốn hủy bỏ các dòng dữ liệu. Trong màn hình hiển thị các dòng dữ liệu trong bảng, đánh dấu chọn ra các dòng dữ liệu liên tiếp nhau muốn hủy bỏ, nhấn phím Del và chọn Yes đề đồng ý việc hủy bỏ các dòng đã chọn.



Cách thức hủy bỏ dữ liệu trong tiện ích Enterprise Manager

SQL Server 2000 : Các câu lệnh truy vấn dữ liệu – Lệnh INSERT INTO

Lệnh truy vấn INSERT TO cho phép bạn thêm mới một hoặc nhiều dòng dữ liệu vào bên trong một bảng. Trước tiên bạn sẽ làm quen với lệnh INSERT TO là lệnh chỉ cho phép bạn thêm mới một dòng dữ liệu vào bên trong bảng.

Cú pháp :

```
INSERT INTO Tên_bảng [ (Danh_sách_cột) ]
        VALUES (Danh_sách_giá_trị)
```

Trong đó :

• **Tên bảng :** tên bảng được thêm mới dòng dữ liệu.

• **Danh sách cột :** danh sách tên các cột có trong bảng. Bạn có thể không cần chỉ định ra tên của các cột, tuy nhiên khi đó danh sách các giá trị mà bạn đưa vào phải theo đúng thứ tự vật lý của các cột bên trong bảng khi tạo cấu trúc bảng trước đó.

Ví dụ :

Để thêm một vật tư mới vào bảng VATTU, bạn sử dụng lệnh INSERT To như sau :

```
INSERT INTO VATTU (MAVTU, TENVTU, DVTINH, PHANTRAM)
VALUES ("LOO1", "Loa Panasonic 1000W", "Bộ", 10)
```

Hoặc bạn cũng có thể thực hiện nhanh lệnh như sau :

```
INSERT INTO VATTU
VALUES ("LOO1", "Loa Panasonic 1000W", "Bộ", 10)
```

Tuy nhiên lúc bấy giờ câu lệnh thứ hai chỉ đúng khi thứ tự của các cột trong bảng VATTU phải là : mã vật tư, tên vật tư, đơn vị tính và tỷ lệ phần trăm.

Trong trường hợp khi bạn có một danh sách các dữ liệu hiện đang có bên trong một hoặc nhiều bảng khác nhau, theo yêu cầu muốn các dữ liệu này sẽ được thêm mới vào bên trong một bảng. Nếu bảng này chưa có trong cơ sở dữ liệu thì bạn có thể sử dụng lệnh SELECT TO như đã trình bày trước đây để sao chép dữ liệu và tạo cấu trúc cho bảng mới. Tuy nhiên nếu theo yêu cầu muốn bạn phải chèn thêm (append) các dòng dữ liệu

này vào bảng dữ liệu hiện có trong cơ sở dữ liệu thì bắt buộc bạn phải sử dụng lệnh INSERT … SELECT.

Cú pháp :

```
INSERT [INTO] Tên_bảng [ (Danh_sách_cột) ]
        SELECT Danh_sách_cột
        FROM Tên_bảng_dl_nguồn
        WHERE Điều_kiện_lọc
```

Trong đó :

• **Tên bảng :** tên bảng được thêm các dòng dữ liệu mới.

• Các mệnh đề của lệnh SELECT bên trên hoàn toàn giống như lệnh SELECT FROM.

Ví dụ :

Để tính ra tổng số lượng nhập, tổng số lượng xuất của các vật tư trong tháng 01/2002, sau đó lấy các dữ liệu này thêm mới vào bảng TONKHO để cập nhật lại tình hình nhập xuất hoàng hóa trong tháng 01/2002, bạn sử dụng các lệnh như sau :

Trước tiên tạo ra bảng ảo dùng để tính tổng số lượng nhập của các vật tư trong tháng 01/2002.

```
CREATE VIEWW vw_TONGN_200201
AS
        SELECT MAVTU, SUM(SLNHAP) AS TONGNHAP
        FROM CTPNHAP CTPN
          INNER JOIN PNHAP PN ON PN.SOPN=CTPN.SOPN
        WHERE CONVERT(CHAR(7), NGAYNHAP, 21)="2002-01"
        GROUP BY MAVTU
GO
```

Kế tiếp tạo ra bảng dùng để tính tổng số lượng xuất của các vật tư trong tháng 01/2002.

```
CREATE VIEW vw_TONGX_200201
AS
	SELECT MAVTU, SUM(SLXUAT) AS TONGXUAT
	FROM CTPXUAT CTPX
	INNER JOIN PXUAT PX
	ON PX.SOPX=CTPX.SOPX
	WHERE CONVERT(CHAR(7), NGAYXUAT, 21)="2002-01"
	GROUP BY MAVTU
GO
```

Sử dụng lệnh INSERT … SELECT để thêm dữ liệu vào bảng TONKHO.

```
INSERT TONKHO
SELECT "2002-01", TN.MAVTU, 0, TONGNHAP, TONGXUAT,
                                TONGNHAP-TONGXUAT
FROM vw_TONGN_200201 TN
	LEFT JOIN vw_TONGX_200201 TX ON TN.MAVTU=TX.MAVTU
```

Trên đây chỉ là ví dụ về lệnh INSERT … SELECT, trong thực tế việc cập nhật dữ liệu trong bảng TONKHO sẽ hoàn toàn được tự động thực hiện thông qua các hành động nhập xuất vật tư. Để làm được việc này, bạn cần biết đến khái niệm trigger bên trong bảng.

Thông thường khi thêm mới dữ liệu vào bảng nếu bạn vô tình có vi phạm các ràng buộc toàn vẹn dữ liệu đã định nghĩa trước đó thì tuyệt đối dữ liệu sẽ không được lưu vào bên trong bảng. Một vài ràng buộc toàn vẹn dữ liệu thường vi phạm là : khóa ngoại, miền giá trị không được phép bỏ trống dữ liệu khi thêm mới.

Tóm lại việc thêm mới dữ liệu vào bên trong bảng băng lệnh INSERT có thể được thêm từng dòng một (INSERT INTO) hoặc cùng lúc nhiều dòng (INSERT … SELECT).

Ngoài ra bạn cũng có thể sử dụng tiện ích Enterprise Manager để nhập mới dữ liệu trực tiếp vào bên trong bảng. Chọn chức năng Open Table (Return all rows) trong thực đơn tắt sau khi nhấn chuột phải trên tên bảng muốn nhập mới dữ liệu. Trong màn hình hiển thị các dòng dữ liệu trong bảng, nhập thông tin dữ liệu mới tại dòng dữ liệu trắng bên dưới cùng.

Màn hình hiển thị các dòng dữ liệu trong bảng

SQL Server 2000 : Cấu trúc điều khiển – Cấu trúc lặp WHILE

Với cấu trúc lặp, người lập trình có thể chỉ định một hoặc nhiều câu lệnh sẽ được thực hiện lặp lại nhiều lần trong khi giá trị của biểu thức luận lý so sánh vẫn còn đúng. Giống như cấu trúc rẽ nhánh, cấu trúc lặp được phép sử dụng bên trong một lô (batch) các lệnh hoặc bên trong một thủ tục nội tại. Giữa cấu trúc rẽ nhánh và cấu trúc lặp không có thứ tự ưu tiên khi chúng lồng vào nhau và cấp độ lồng nhau là không có giới hạn.

Thực tế việc sử dụng cấu trúc lặp WHILE bị giới hạn trong nhiều trường hợp. Bởi vì bản thân các lệnh truy vấn cập nhật dữ liệu như là : SELECT, UPDATE SET, DELETE trong Transaction-SQL đã tự động thực hiện việc lặp từ dòng dữ liệu đầu tiên đến dòng dữ liệu cuối cùng bên trong bảng.

Cấu trúc lặp WHILE thông thường được dùng với các biến có kiểu dữ liệu cursor, cách thức sử dụng biến kiểu dữ liệu cursor sẽ được hướng dẫn trong các phần sau.

Cú pháp :

```
WHILE Biểu_thức_luận_lý
BEGIN
      Các_lệnh_lặp
END
```

Trong đó :

• Biểu thức luận lý : thông thường là một biểu thức so sánh để chỉ các lệnh sẽ được lặp lại trong khi mà giá trị của biểu thức vẫn còn đúng.

• Các lệnh lặp : các câu lệnh được thực hiện bên trong vòng lặp.

Ví dụ :

Để in ra 10 số nguyên dương bắt đầu từ 100. Bạn sử dụng cấu trúc lặp WHILE như sau :

```
DECLARE @Songuyen INT
SET @Songuyen = 100
WHILE (@Songuyen<110)
BEGIN
	PRINT "Số nguyên : " + CONVERT(CHAR(3), @Songuyen)
	SET @Songuyen = @Songuyen + 1
END
```

Kết quả trả về :

```
Số nguyên : 100
Số nguyên : 101
Số nguyên : 102
Số nguyên : 103
Số nguyên : 104
Số nguyên : 105
Số nguyên : 106
Số nguyên : 107
Số nguyên : 108
Số nguyên : 109
```

Bạn có thể sử dụng từ khóa BREAK lồng vào cấu trúc lặp WHILE để có thể kết thúc việc lặp của các lệnh bên trong vòng lặp mà không cần xét đến giá trị trả về của biểu thức luận lý dùng để so sánh phía sau từ khóa WHILE phải là sai. Tuy nhiên từ khóa BREAK thường được sử dụng kèo theo với một biểu thức luận lý khác.

Ví dụ :

Thực hiện việc lặp giống ví dụ trên, tuy nhiên muốn rằng vòng lặp sẽ bị kết thúc khi mới in tới số nguyên 105. Bạn sử dụng cấu trúc lặp WHILE như sau :

```
DECLARE @Songuyen INT
SET @Songuyen = 100
WHILE (@Songuyen<110)
BEGIN
	PRINT "Số nguyên : " + CONVERT(CHAR(3), @Songuyen)
	IF @Songuyen=105
		BREAK
	SET @Songuyen = @Songuyen + 1
END
PRINT "Kết thúc vòng lặp"
```

Kết quả trả về :

```
Số nguyên : 100
Số nguyên : 101
Số nguyên : 102
Số nguyên : 103
Số nguyên : 104
Số nguyên : 105
Kết thúc vòng lặp
```

Bạn cụng có thể sử dụng từ khóa CONTINUE lồng vào cấu trúc lặp WHILE để chỉ định các lệnh bên trong vòng lặp ở phía dưới từ khóa CONTINUE tạm thời không thực hiện tiếp, khi đó con trỏ vòng lặp sẽ nhảy về đầu dòng lặp để kiểm tra giá trị của biểu thức luận lý so sánh là vẫn còn đúng hay không. Tuy nhiên từ khóa CONTINUE thông thường được dùng kèm theo với một biểu thức luận lý khác.

Ví dụ :

Thực hiện việc lặp giống các ví dụ trên, tuy nhiên muốn rằng vòng lặp sẽ in xót số nguyên 105. Bạn sử dụng cấu trúc lặp WHILE như sau :

```
DECLARE @Songuyen INT
SET @Songuyen = 99
WHILE (@Songuyen<110)
BEGIN
	SET @Songuyen = @Songuyen + 1
	IF @Songuyen=105
		CONTINUE
	PRINT "Số nguyên : " + CONVERT(CHAR(3), @Songuyen)
END
PRINT "Kết thúc vòng lặp"
```

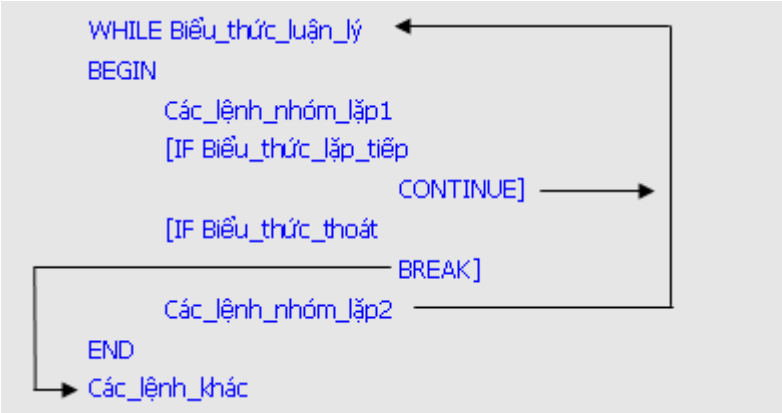
Kết quả trả về :

```
Số nguyên : 100
Số nguyên : 101
Số nguyên : 102
Số nguyên : 103
Số nguyên : 104
Số nguyên : 106
Số nguyên : 107
Số nguyên : 108
Số nguyên : 109
Số nguyên : 110
Kết thúc vòng lặp
```

Kết chứng kết quả trả về bạn suy luận khi vòng lặp thực hiện đến giá trị của biến @Songuyen = 105 thì khi đó lệnh PRINT "Số nguyên : " ... không được thực hiện và con trỏ chương trình được quay lên đầu vòng lặp để kiểm tra tiếp biểu thức luận lý so sánh.

Sơ đồ tóm tắt ý nghĩa của cấu trúc lặp WHILE kèm với các từ khóa CONTINUE hoặc BREAK :

Cú pháp :



Ví dụ :

Để tăng tự động tỷ lệ phần trăm cho các vật tư trong bảng VATTU theo quy tắc sau :

• Mỗi lần chỉ tăng lên 5% cho các vật tư có giá trị tại cột tỷ lệ nhỏ hơn 30%.

• Lặp lại hành động tăng trong khi mà giá trị trung bình tỷ lệ phần trăm của các vật tư vẫn còn thấp hơn 40%.

Bạn sử dụng các lệnh như sau :

```
-- In giá trị trung bình tỷ lệ trước khi tăng
PRINT "Trung bình tỷ lệ phần trăm trước khi tăng"
SELECT AVG(PHANTRAM) FROM VATTU

-- Bắt đầu thực hiện việc lặp tăng tự động
DECLARE @Lantang INT
SET @Lantang = 0
WHILE (SELECT AVG(PHANTRAM) FROM VATTU) < 40
BEGIN
        UPDATE VATTU
        SET PHANTRAM = PHANTRAM + 5
        WHERE PHANTRAM < 30

        SET @Lantang = @Lantang + 1
        IF NOT EXISTS (SELECT * FROM VATTU
                                WHERE PHANTRAM < 30)
        BREAK
END
-- In giá trị trung bình tỷ lệ sau khi tăng
PRINT "Đã tăng : " + CONVERT(VARCHAR(7), @Lantang) + "lần"
PRINT "Trung bình tỷ lệ phần trăm sau khi tăng"
SELECT AVG(PHANTRAM) FROM VATTU
```

Nhận xét : ví dụ trên khá phức tạp, do thế bạn phải chèn thêm vào các ghi chú (comment) để giúp bạn thất được từng xử lý rời rạc nhằm dễ xem, dễ hiểu. Trong Transaction-SQL, bạn có thể chèn các ghi chú trong các câu lệnh bằng hai dấu trừ liên tiếp nhau hoặc muốn che lại một khối các lệnh liên tiếp nhau thì bạn sẽ sử dụng cặp ký tự như bên dưới :

```
/*
Các lệnh được che đậy
*/
```

Ngoài ra trong vòng lặp bạn có thể sử dụng lệnh IF EXISTS dùng để kiểm tra trường hợp sau khi đã tăng hết tất cả tỷ lệ phần trăm các vật tư với đều lớn hơn 30 mà trung bình tỷ lệ phần trăm của các vật tư vẫn chưa lớn hơn 40 thì bắt buộc vòng lặp phải được thoát ra ngoài, bởi vì nếu không thì vòng lặp sẽ bị lặp vô tận không bao giờ thoát ra được.

Tạo các truy vấn dữ liệu bằng QBE

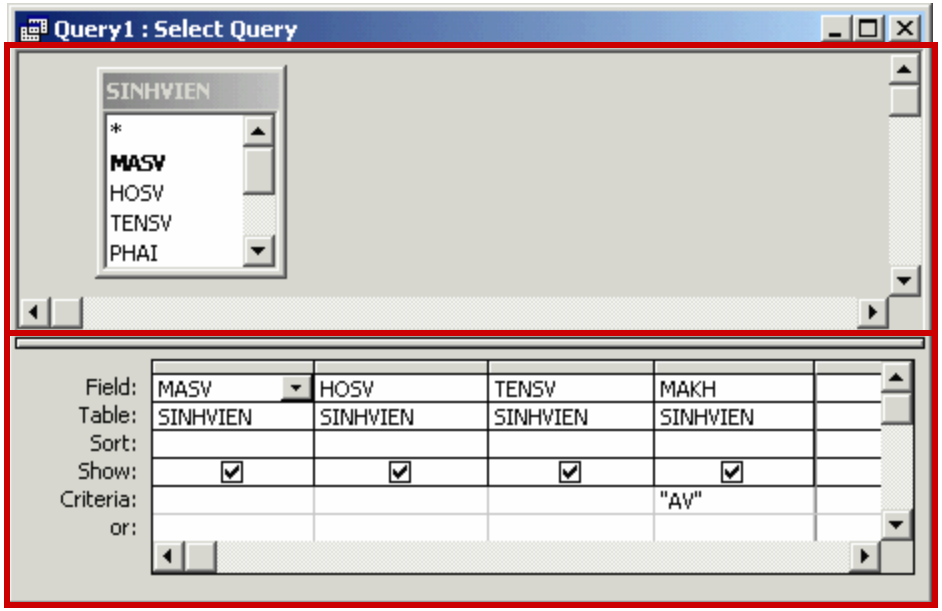
- Màn hình làm việc QBE
- Tạo mới truy vấn
- Thực hiện truy vấn
- Các chế độ hiển thị
- Sử dụng QBE để tạo các loại truy vấn
- Thay đổi tính chất quan hệ giữa các bảng trong truy vấn

Màn hình làm việc QBE

- Hỗ trợ tạo nhanh các loại truy vấn thông qua một số bước thực hiện đơn giản
- Cho phép xem kết quả và hiệu chỉnh nội dung của truy vấn một cách dễ dàng.

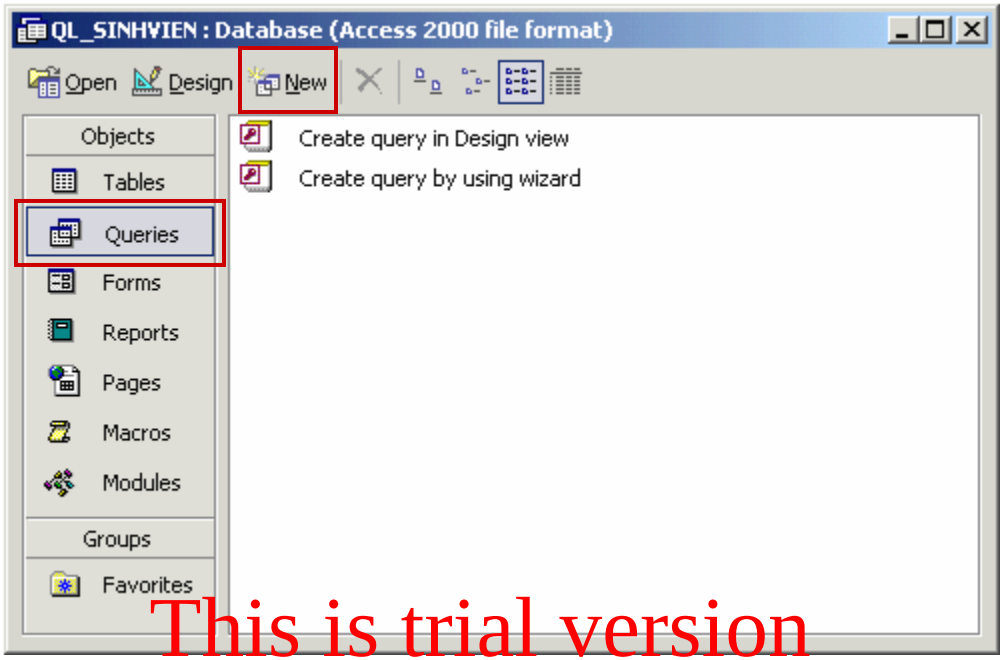
Màn hình làm việc QBE

- **Các thành phần trong màn hình QBE**
  - Vùng chứa bảng
  - Vùng lưới QBE
    - **Field**
    - **Table**
    - **Sort**
    - **Show**
    - **Criteria**



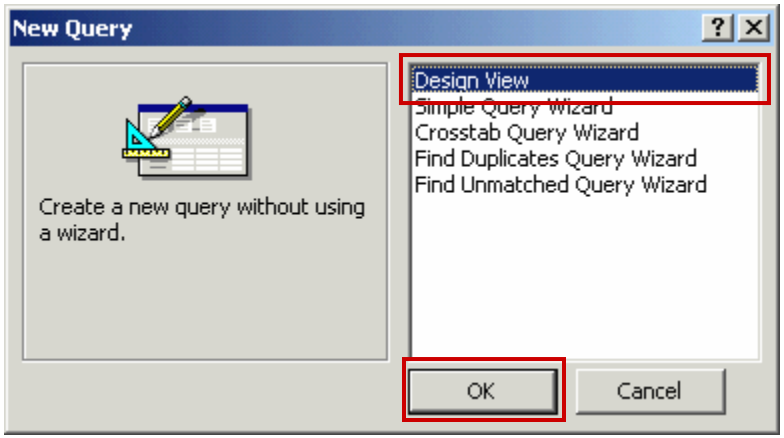
Tạo mới truy vấn bằng QBE

- **Bước 1: Tạo mới đối tượng Query**
  - Trong cửa sổ Database, chọn thẻ Queries và nhấn New



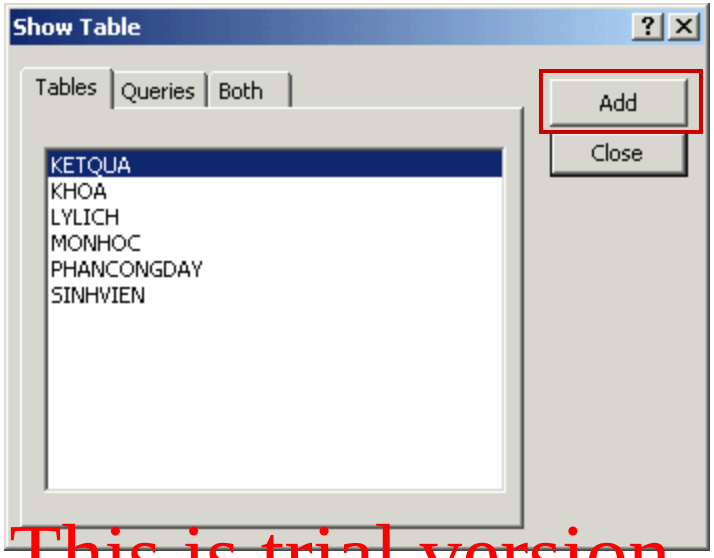
Tạo mới truy vấn bằng QBE

- **Bước 1: Tạo mới đối tượng Query**
  - Trong cửa sổ New Query, chọn mục Design View và nhấn nút OK



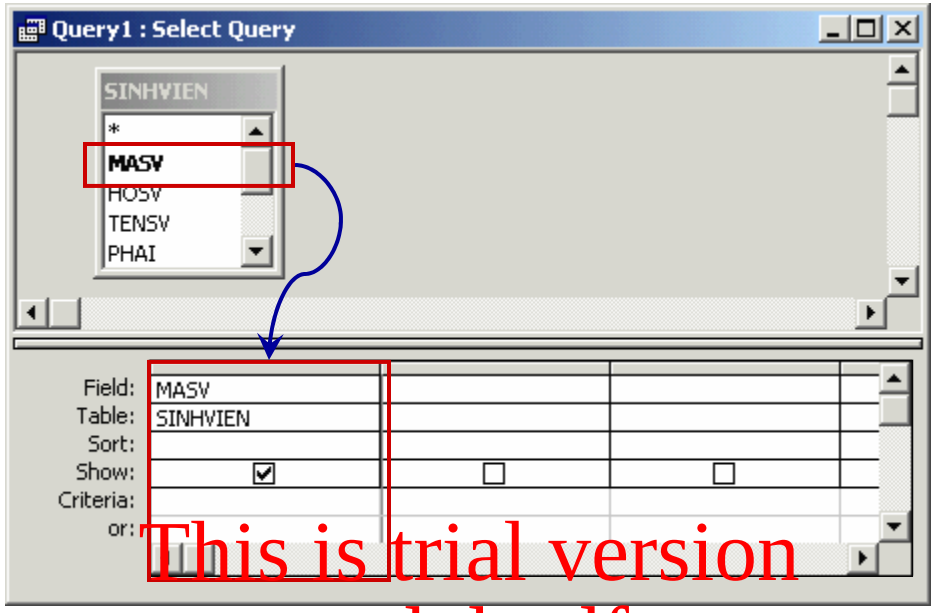
Tạo mới truy vấn bằng QBE

- **Bước 1: Tạo mới đối tượng Query**
  - Trong cửa sổ Show Table, chọn các bảng cần lấy dữ liệu và nhấn nút Add để chọn, sau đó nhấn Close để đóng.



Tạo mới truy vấn bằng QBE

- **Bước 2: Thiết kế cấu trúc câu truy vấn**
  - Trong cửa sổ thiết kế Query, chọn các cột cần lấy dữ liệu trong vùng chứa bảng và kéo thả vào dòng Field



Tạo mới truy vấn bằng QBE

- **Bước 3: lưu lại cấu trúc câu truy vấn**
  - Chọn thực đơn File → Save để lưu lại truy vấn
  - Đặt tên cho truy vấn vừa tạo

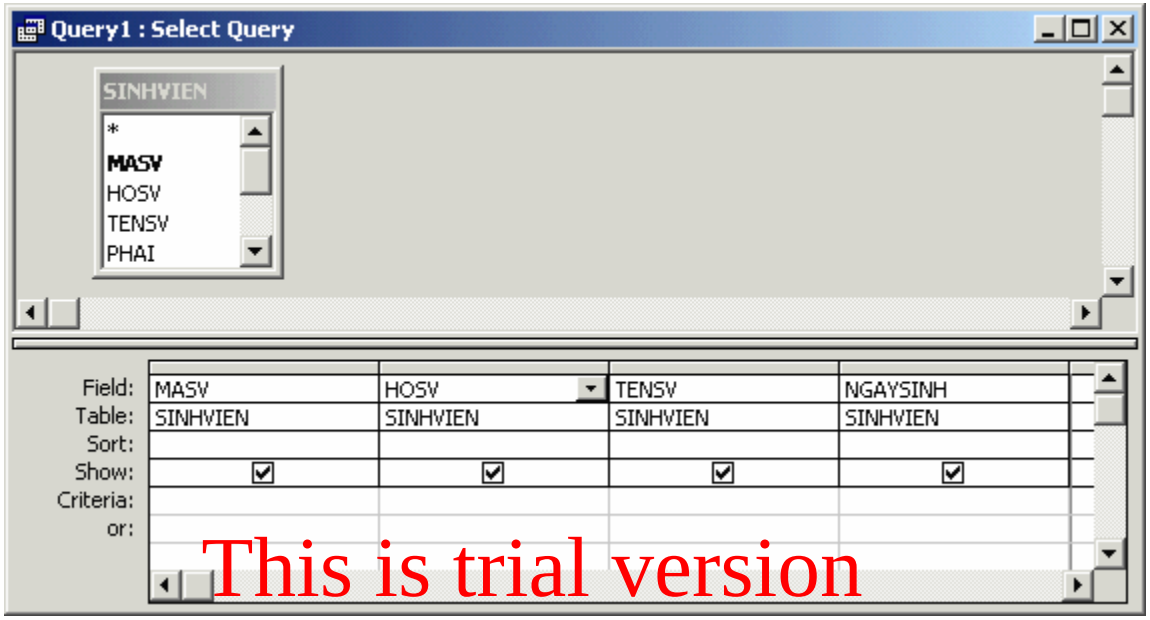
Thực hiện truy vấn

- Nhấn nút Run trên thanh Toolbar

- Chọn thực đơn Query → Run

Các chế độ hiển thị

- **Các chế độ hiển thị khi thiết kế truy vấn**
  - Design View: hiển thị cấu trúc của câu truy vấn trong cửa sổ QBE



Điều chỉnh SQL với Optim Query Tuner, Phần 2: Điều chỉnh các truy vấn riêng lẻ

Giới thiệu

Trong bài đầu tiên của loạt bài này, Điều chỉnh SQL với Optim Query Tuner, Phần 1: Tìm hiểu về các đường dẫn truy cập, đã giới thiệu khái niệm về một *đường dẫn truy cập*. Với một câu lệnh SQL cụ thể, thường có nhiều sự lựa chọn đường dẫn truy cập, và các đường dẫn truy cập khác nhau thường có đặc điểm hiệu năng khác nhau. Trước khi thực hiện SQL, trình tối ưu hóa DB2 ước tính giá của các đường dẫn truy cập ứng cử viên và chọn đường dẫn có giá ước tính thấp nhất. Quá trình này được bao gồm trong bước PREPARE (Chuẩn bị) cho một câu lệnh SQL động, hoặc trong bước BIND (Kết buộc) cho một câu lệnh SQL tĩnh.

Mặc dù trình tối ưu hóa DB2 có ích cho việc chọn đường dẫn truy cập tốt nhất, nhưng kết quả lại phụ thuộc vào dữ liệu đầu vào, mà thường trình tối ưu hóa không truy cập hoặc kiểm soát được. Nếu bạn là một người phát triển hoặc người quản trị cơ sở dữ liệu (DBA), với bạn để hiểu cách điều chỉnh các truy vấn đến mức bạn có thể cung cấp đầu vào tốt nhất cho trình tối ưu hóa DB2 rất có ích.

Trong bài này, tác giả giới thiệu cho bạn một phương pháp luận để điều chỉnh các truy vấn riêng lẻ, bao gồm lý do cơ bản để hiểu tại sao cách điều chỉnh các truy vấn lại quan trọng ngay cả khi có trình tối ưu hóa tầm cỡ thế giới tồn tại trong DB2. Sau đó bạn sử dụng một truy vấn mẫu để giải thích phương pháp luận để điều chỉnh một truy vấn, khi sử dụng các tính năng liên quan của Optim Query Tuner, có thể rất có ích trong việc giúp bạn hiểu, phân tích, và điều chỉnh các truy vấn riêng lẻ.

Lưu ý rằng bài này được thiết kế chủ yếu để điều chỉnh SQL trên DB2 cho z/OS, nhưng hầu hết các khái niệm tối ưu hóa truy vấn và phương pháp luận điều chỉnh SQL trong bài này cũng áp dụng được với DB2 cho Linux®, UNIX® và Windows®.

Nếu bạn muốn tự mình dùng thử truy vấn mẫu trong bài này, bạn có thể tải về các tệp dự án mẫu trong phần tải về của bài này, và sau đó nhập khẩu tệp dự án vào Data Studio (gói độc lập hoặc gói IDE với Fix Pack 1 hoặc mới hơn) hoặc bất kỳ các sản phẩm Optim Query Tuner nào.

Để nhập khẩu dự án mẫu hãy làm như sau:

1. Mở **IBM Query Tuning** Perspective (Phối cảnh điều chỉnh truy vấn IBM) của sản phẩm Data Studio hoặc Optim Query Tuner của bạn.
2. Chọn **File** > **Import.**.
3. Trong Import wizard (Trình hướng dẫn nhập khẩu), chuyển hướng đến **Query Tuner** > **Projects**, rồi nhấn **Next**.
4. Nhấn **Browse** (Duyệt) và chọn thư mục có chứa tệp zip đã tải về để xem một danh sách các dự án trong cửa sổ Projects (Các dự án).
5. Chọn **samplequerytuningproject** và nhấn **Finish**.

6. Bây giờ dự án mẫu sẽ xuất hiện trong Project Explorer (Trình thám hiểm dự án) của bạn. Nếu bạn không thấy một Project Explorer Window (Cửa sổ Project Explorer), hãy chắc chắn bạn đang ở trong **IBM Query Tuning** Perspective và chọn **Window** > **Reset Perspective**. Ngoài ra, bạn có thể chọn **Window** > **Show View** > **Project Explorer**.

Về các giải pháp điều chỉnh truy vấn Optim

Các giải pháp điều chỉnh truy vấn Optim cung cấp một môi trường để nhận biết và điều chỉnh việc thực hiện các câu lệnh SQL không chạy với các lời khuyên và các công cụ có thể trợ giúp hướng dẫn bạn đến một giải pháp. Các khả năng điều chỉnh truy vấn được cung cấp trong các sản phẩm sau:

- Các khả năng định dạng truy vấn và điều chỉnh truy vấn đơn lẻ, cơ bản có sẵn trong bản Data Studio 2.2.1 (cả bản độc lập lẫn bản IDE). Sản phẩm này có sẵn miễn phí cho cả hai DB2 cho z/OS và DB2 cho Linux, UNIX và Windows. Cần hiểu rõ rằng trong khi thông tin trong loạt bài này giải thích cách bạn có thể sử dụng Data Studio để giải thích các biểu đồ đường dẫn truy cập, thì không phải tất cả các khả năng được mô tả có sẵn trong Data Studio.
- Định dạng truy vấn và điều chỉnh truy vấn đơn lẻ, cũng như tập các trình tư vấn lớn hơn, có sẵn trong Optim Query Tuner. Sản phẩm này hiện có sẵn cho cả hai DB2 cho z/OS và DB2 cho Linux, UNIX và Windows.
- Điều chỉnh tải truy vấn, điều chỉnh truy vấn đơn lẻ và toàn bộ tập các trình tư vấn có sẵn trong Optim Query Workload Tuner (Trình điều chỉnh tải truy vấn Optim). Sản phẩm này chỉ có sẵn cho DB2 cho z/OS (tại thời điểm viết bài này).

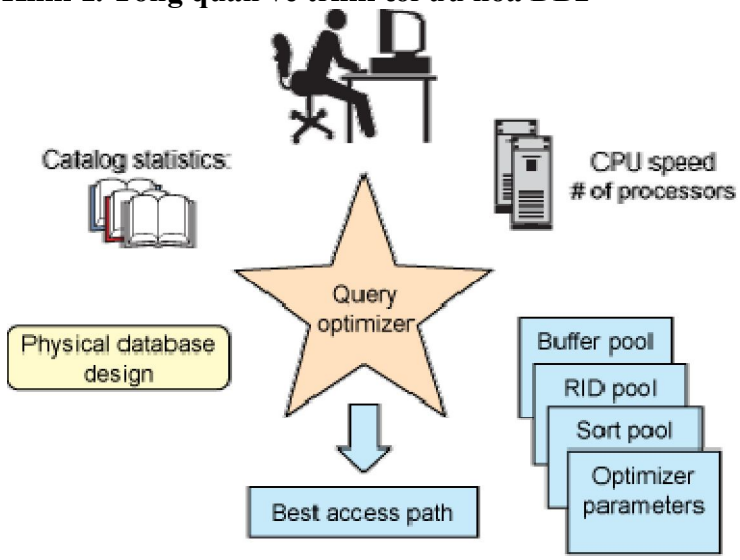
Tóm lại, loạt bài này sử dụng tên *Optim Query Tuner* (OQT-Trình điều chỉnh truy vấn Optim) để nói đến tập các trình tư vấn và các công cụ mà các giải pháp điều chỉnh truy vấn Optim cung cấp. Ở đây các tên sản phẩm cụ thể, thích hợp được cung cấp khi mô tả các khả năng có thể không có sẵn trong tất cả các sản phẩm được liệt kê ở trên.

Lưu ý rằng bài này chủ yếu tập trung vào phương pháp luận điều chỉnh truy vấn và sử dụng các ảnh chụp màn hình từ Optim Query Tuner để minh họa cho các ý kiến này. Bài này không nhằm mục đích cung cấp thông tin "hướng dẫn" bằng cách sử dụng Query Tuner. Để có thêm thông tin về chuyển hướng đến các tính năng sản phẩm khác nhau, và để xem một giới thiệu chi tiết về cách khởi chạy các hàm khác nhau trong OQT, hãy tham khảo phần Tài nguyên.

Tổng quan về tối ưu hóa truy vấn

Như cho thấy trong Hình 1, trình tối ưu hóa DB2 chọn đường dẫn truy cập tốt nhất.

Hình 1. Tổng quan về trình tối ưu hóa DB2



Trình tối ưu hóa so sánh giá của mỗi đường dẫn truy cập ứng cử viên dựa trên thông tin từ nhiều đầu vào, ví dụ, hãy xem dưới đây:

- **Số liệu thống kê danh mục**
  Trình tối ưu hóa DB2 là một tối ưu hóa dựa trên giá. Nền tảng của sự tối ưu hóa dựa trên giá là một tập số liệu thống kê cho phép trình tối ưu hóa đánh giá chính xác giá của tất cả các đường dẫn truy cập ứng cử viên và phân biệt các đường dẫn truy cập hiệu quả với các đường dẫn truy cập không hiệu quả. Số liệu thống kê trong các bảng danh mục DB2 được sử dụng để ước tính giá của đường dẫn truy cập. Ví dụ, thông tin trong bảng danh mục SYSTABLES và SYSTABLESPACE cho bạn biết có bao nhiêu hàng và trang chứa dữ liệu trong bảng của bạn.
- **Thiết kế cơ sở dữ liệu vật lý**
  Thiết kế cơ sở dữ liệu vật lý bao gồm thiết kế bảng, thiết kế chỉ mục, thiết kế bảng truy vấn được cụ thể hóa và thiết kế của các đối tượng cơ sở dữ liệu vật lý khác. Thiết kế chỉ mục có một tác động quan trọng đến việc lựa chọn đường dẫn truy cập. Như đã được đề cập trong bài trước, với truy cập bảng riêng lẻ, có hai kiểu phương thức truy cập: quét vùng bảng (TBSCAN) và quét chỉ mục (IXSCAN). Các quá trình quét chỉ mục thường là cách hiệu quả nhất để truy cập dữ liệu, đặc biệt là khi bảng lớn, nhưng số các hàng đủ điều kiện lại nhỏ.
- **Câu lệnh SQL**
  Chính câu lệnh SQL cũng ảnh hưởng đến việc lựa chọn đường dẫn truy cập. Ví dụ, các biến vị ngữ được mã hóa không đúng có thể ngăn không cho trình tối ưu hóa sử dụng quét chỉ mục ngay cả khi chỉ mục có sẵn. Ngoài ra, trước khi chọn đường dẫn truy cập, trước tiên trình tối ưu hóa thực hiện một loạt các chuyển đổi truy vấn để tăng số các đường dẫn truy cập có sẵn. Nếu câu lệnh SQL bị mã hóa sai, thật khó chuyển đổi các truy vấn với trình tối ưu hóa, có ít tùy chọn có sẵn hơn để chọn một đường dẫn truy cập tối ưu.
- **Các xem xét khác để chọn đường dẫn truy cập**
  Ngoài việc tự xem xét số liệu thống kê danh mục, thiết kế cơ sở dữ liệu vật lý và câu lệnh

SQL, trình tối ưu hóa DB2 cũng xem xét mô hình bộ xử lý trung tâm, số lượng các bộ xử lý trung tâm, kích thước nhóm bộ đệm, kích thước nhóm RID và các thiết lập tài nguyên hệ thống khác. Ví dụ, đường dẫn truy cập có thể thay đổi từ một hệ thống này sang một hệ thống khác nếu chúng có các kích thước nhóm bộ đệm khác nhau, ngay cả khi tất cả số liệu thống kê danh mục giống hệt nhau.

Trình tối ưu hóa DB2 là toàn diện và khá mạnh. Nếu trình tối ưu hóa DB2 đang hoạt động, thì tại sao cần điều chỉnh truy vấn? Có hai lý do trả lời cho câu hỏi này:

- **Trình tối ưu hóa DB2 không biết tất cả.**
  Mặc dù trình tối ưu hóa DB2 có rất nhiều thông tin nhờ đó để bố trí kế hoạch của nó, nó không thể biết những gì không tồn tại. Ví dụ, trình tối ưu hóa không biết các đặc điểm của dữ liệu trừ khi bạn đã chạy RUNSTATS để điền số liệu thống kê có liên quan vào danh mục đó. Ngoài ra, không thể biết được một số mục cho đến thời gian chạy. Ví dụ, trình tối ưu hóa không biết được các giá trị của các biến hoặc các dấu tham số chủ (nếu chúng được chứa trong truy vấn) cho đến khi thực hiện truy vấn.
- **Trình tối ưu hóa DB2 không kiểm soát tất cả.**
  Như đã đề cập ở trên, thiết kế cơ sở dữ liệu vật lý, câu lệnh SQL và các giá trị thiết lập tài nguyên hệ thống tác động đến cách trình tối ưu hóa lựa chọn đường dẫn truy cập tốt nhất, nhưng cả hai cơ sở dữ liệu lẫn thiết kế truy vấn đều là các nhiệm vụ đang nằm ngoài sự kiểm soát của trình tối ưu hóa DB2. Đây là nơi mà các DBA và những người phát triển đóng một vai trò quan trọng trong việc trợ giúp hoặc gây thiệt hại cho hiệu năng SQL.

Mục đích của việc điều chỉnh truy vấn là cung cấp đầu vào có thể tốt nhất cho trình tối ưu hóa sao cho trình tối ưu hóa có thể chọn đường dẫn truy cập tốt nhất. Điều này liên quan đến nỗ lực từ cả hai những người phát triển ứng dụng và các DBA.

Đối với những người phát triển ứng dụng:

- **Làm theo các hướng dẫn và các tiêu chuẩn mã hóa SQL.**
  Bạn cần tuân theo các hướng dẫn và các tiêu chuẩn mã hóa SQL khi bạn viết các câu lệnh SQL của mình. Ví dụ, viết các biến vị ngữ chỉ mục có khả năng hoặc các biến vị ngữ giai đoạn 1 và tránh viết các truy vấn không có các biến vị ngữ nối (còn được gọi là *phép nối Đề-các*).
- **Khai thác các tùy chọn kết buộc REOPT một cách đúng đắn.**
  Đối với các câu lệnh SQL có các biến, trình tối ưu hóa sử dụng một hệ số bộ lọc mặc định để xác định đường dẫn truy cập tốt nhất tại thời điểm kết buộc. Trong một số trường hợp, đường dẫn truy cập không thực hiện tốt trong thời gian chạy nếu câu lệnh đó có chứa các biến máy chủ, các dấu tham số, hoặc các đăng ký đặc biệt. Bạn có thể sử dụng các tùy chọn kết buộc REOPT để *tối ưu hóa lại* đường dẫn truy cập hoặc tại thời điểm kết buộc hoặc trong thời gian chạy.

Đối với các nhà quản trị cơ sở dữ liệu (DBA):

- **Thu thập số liệu thống kê đầy đủ và chính xác.**
  Số liệu thống kê không đầy đủ hoặc không chính xác dẫn đến các ước tính giá không

chính xác cho các đường dẫn truy cập ứng cử viên và là lý do phổ biến nhất làm cho việc lựa chọn các đường dẫn truy cập không hiệu quả. Trong khi đó, việc thu thập và làm mới tất cả số liệu thống kê sẽ tiêu tốn quá nhiều tài nguyên không cần thiết. Căn cứ vào số lượng các hoạt động INSERT, UPDATE và DELETE và các thay đổi trong các bản phân phối dữ liệu, bạn cần thu thập số liệu thống kê thường xuyên và với việc tiêu thụ tài nguyên tối thiểu.

- **Tối ưu hóa thiết kế chỉ mục.**
  Bạn cần thiết kế các chỉ mục để hỗ trợ truy cập hiệu quả với các biến vị ngữ cục bộ và các biến vị ngữ nối. Bạn cũng có thể cần thiết kế các chỉ mục để tránh sắp xếp dữ liệu và cung cấp chỉ mục chỉ để truy cập.
- **Điều chỉnh toàn bộ ứng dụng.**
  Để đảm bảo hiệu năng tốt của ứng dụng, điều cần thiết là điều chỉnh toàn bộ ứng dụng này. Nỗ lực cần thiết để điều chỉnh toàn bộ ứng dụng, bằng cách đánh giá tất cả các câu lệnh riêng lẻ, có ưu thế hơn. Ngoài ra, việc cải thiện hiệu năng trên một câu lệnh có thể đi ngược lại hiệu năng của các câu lệnh khác trong ứng dụng. Vì vậy, điều rất quan trọng là điều chỉnh toàn bộ ứng dụng, còn được gọi là điều chỉnh tải công việc. Bài này sẽ tập trung vào điều chỉnh một truy vấn đơn, phần tiếp theo của loạt bài này sẽ mở rộng phương pháp luận trong bài này để giới thiệu điều chỉnh tải công việc một cách chi tiết.

Bài này mô tả một phương pháp luận để hiểu các vấn đề về hiệu năng truy vấn tiềm năng và cách giải quyết những vấn đề tiềm năng đó. Việc sử dụng Optim Query Tuner làm cho quá trình này đơn giản hơn.

Phương pháp luận điều chỉnh truy vấn

Tổng quan về phương pháp luận điều chỉnh truy vấn

Để thực hiện điều chỉnh truy vấn, trước tiên bạn cần hiểu những gì bạn muốn điều chỉnh, trong trường hợp này đó là chính truy vấn đó và trình tối ưu hóa lựa chọn kế hoạch truy cập hiện tại của truy vấn đó, rồi tìm ra cách để điều chỉnh truy vấn đó.

Dựa trên ý tưởng này, bạn sẽ thực hiện các nhiệm vụ sau để điều chỉnh truy vấn đầy đủ, bạn có thể thực hiện truy vấn đó từ bên trong Query Tuner:

- Định dạng truy vấn vấn đề để làm cho việc đọc và hiểu logic truy vấn dễ dàng hơn.
- Chú thích truy vấn vấn đề với số liệu thống kê có liên quan để hiểu rõ hơn những gì trình tối ưu hóa DB2 đang sử dụng cho các đánh giá của nó.
- Phân tích kế hoạch truy cập truy vấn để hiển thị trực quan các lựa chọn mà trình tối ưu hóa thực hiện khi truy cập dữ liệu.
- Thực hiện phân tích số liệu thống kê để đảm bảo rằng trình tối ưu hóa DB2 luôn có số liệu thống kê phổ biến nhất và số liệu thống kê cần thiết nhất.
- Thực hiện phân tích biến vị ngữ để xem liệu các biến vị ngữ có khả năng chọn lọc không.
- Thực hiện phân tích chỉ mục để đảm bảo rằng các chỉ mục thích hợp tồn tại để giúp tránh các lần quét bảng không cần thiết.

Trong các phần tiếp theo, sử dụng câu lệnh SQL trong Liệt kê 1 làm một mẫu để giải thích từng nhiệm vụ điều chỉnh truy vấn riêng một cách chi tiết. Như bạn có thể tưởng tượng, có một ít sự phụ thuộc lẫn nhau giữa các nhiệm vụ này. Ví dụ, việc thay đổi số liệu thống kê, được thu thập, có thể có nhiều khả năng ảnh hưởng đến các kết quả phân tích biến vị ngữ. Ngoài ra, bạn có thể cần lặp lại thông qua một hoặc nhiều nhiệm vụ này vài lần cho đến khi giải quyết được một vấn đề hiệu năng cụ thể.

Liệt kê 1. Truy vấn mẫu được sử dụng trong bài này

```
SELECT CCUS.CUST_FIRST_NAME
     , CCUS.CUST_LAST_NAME
     , CINT.CUST_INTEREST_RANK
     , CILO.CUST_INTERST
FROM CUST_CUSTOMER AS CCUS
     , CUST_INTEREST_LOOKUP AS CILO
     , CUST_INTEREST AS CINT
WHERE ( CCUS.CUST_CITY = 'Singapore'
        AND CCUS.CUST_PROV_STATE = 'Singapore'
        AND CCUS.CUST_CODE IN (
                        SELECT COHE.CUST_CODE
                        FROM CUST_ORDER_HEADER AS COHE
                             , CUST_ORDER_STATUS AS COST
                        WHERE ( COHE.CUST_ORDER_DATE
                                    >='2009-01-01 00:00:00.001'
                                AND COST.CUST_ORDER_STATUS IN ( 'Shipped',
                                    'Back-ordered', 'In-process' )
                                AND COHE.CUST_ORDER_STATUS_CODE
                                   = COST.CUST_ORDER_STATUS_CODE
                              )
                    )
        AND CCUS.CUST_CODE = CINT.CUST_CODE
        AND CINT.CUST_INTEREST_CODE = CILO.CUST_INTEREST_CODE
      )
ORDER BY CCUS.CUST_LAST_NAME ASC
         , CCUS.CUST_FIRST_NAME ASC
         , CINT.CUST_INTEREST_RANK ASC
```

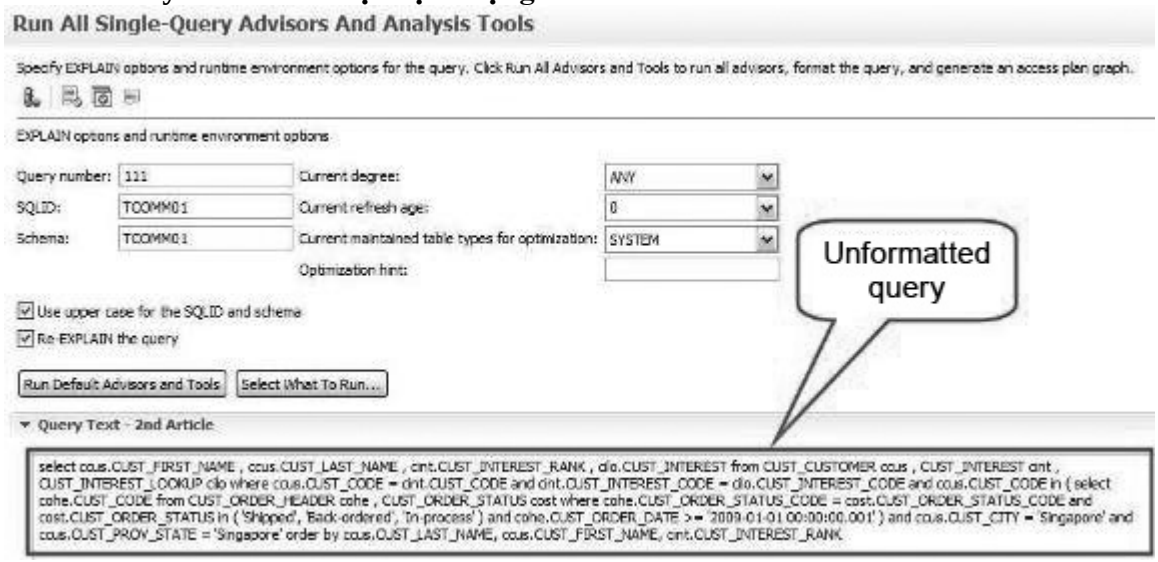
Định dạng truy vấn vấn đề

Trước khi điều chỉnh một truy vấn, bạn cần hiểu các khía cạnh sau về truy vấn vấn đề.

- *Các ngữ nghĩa* của truy vấn: Cần truy cập những bảng nào trong truy vấn đó? Sử dụng các loại biến vị ngữ nào trên mỗi bảng được tham khảo? Sử dụng các loại biến vị ngữ nào để nối các bảng được tham khảo?
- *Đường dẫn truy cập* của các truy vấn: Cách truy cập các bảng? Toàn bộ bảng có được quét hoặc có được truy cập bằng một chỉ mục không? Nếu có một chỉ mục, thì đó là chỉ mục hay các chỉ mục nào? Chuỗi nối và phương pháp nối là gì?

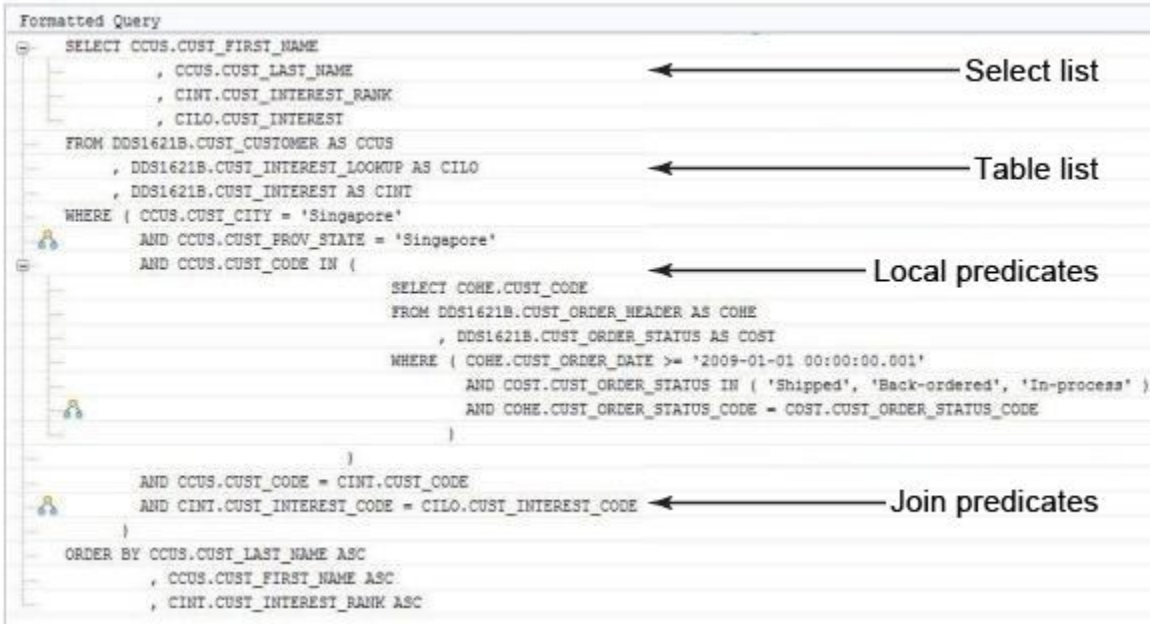
Như bạn có thể thấy trong Hình 2, truy vấn vấn đề chưa được định dạng ban đầu khó đọc và hiểu.

Hình 2. Truy vấn chưa được định dạng



Optim Query Tuner có thể định dạng truy vấn vấn đề, cung cấp một điểm khởi đầu tốt để phân tích. Trong truy vấn đã định dạng, mỗi tham khảo bảng, mỗi tham khảo cột trong mệnh đề SELECT và mỗi biến vị ngữ được hiển thị trên một dòng riêng của nó. Đối với truy vấn mẫu trong bài này, truy vấn đã định dạng được hiển thị trong Hình 3.

Hình 3. Truy vấn đã định dạng



Như bạn có thể tưởng tượng, với SQL phức tạp dài dòng, việc định dạng đơn giản truy vấn có thể tiết kiệm hàng giờ đồng hồ cho DBA. Bây giờ rất dễ tìm ra các bảng nào được truy cập, có bao nhiêu bảng trong truy vấn, và các bảng này được nối như thế nào. Các truy vấn đã định dạng cung cấp cho bạn khả năng sau:

- Tìm hiểu các bộ phận của truy vấn một cách chi tiết hơn, như các khung nhìn và các truy vấn con được tham khảo, bằng cách mở rộng và thay đổi các phần của một SQL phức tạp.
- Dễ dàng thấy cách truy cập một bảng cụ thể trong SQL. Khi bạn nhấn vào bất kỳ dòng nào trong truy vấn đã định dạng, các dòng khác của truy vấn đó có chứa các tham khảo cột hoặc bảng của cùng một bảng cũng được đánh dấu.
- Tùy chỉnh thứ tự định dạng của các biến vị ngữ theo các tiêu chí khác nhau như các biến vị ngữ cục bộ hoặc các biến vị ngữ nối, và các tham khảo bảng.

Trở lại với truy vấn đã định dạng được hiển thị trong Hình 3, bạn có thể thấy như sau:

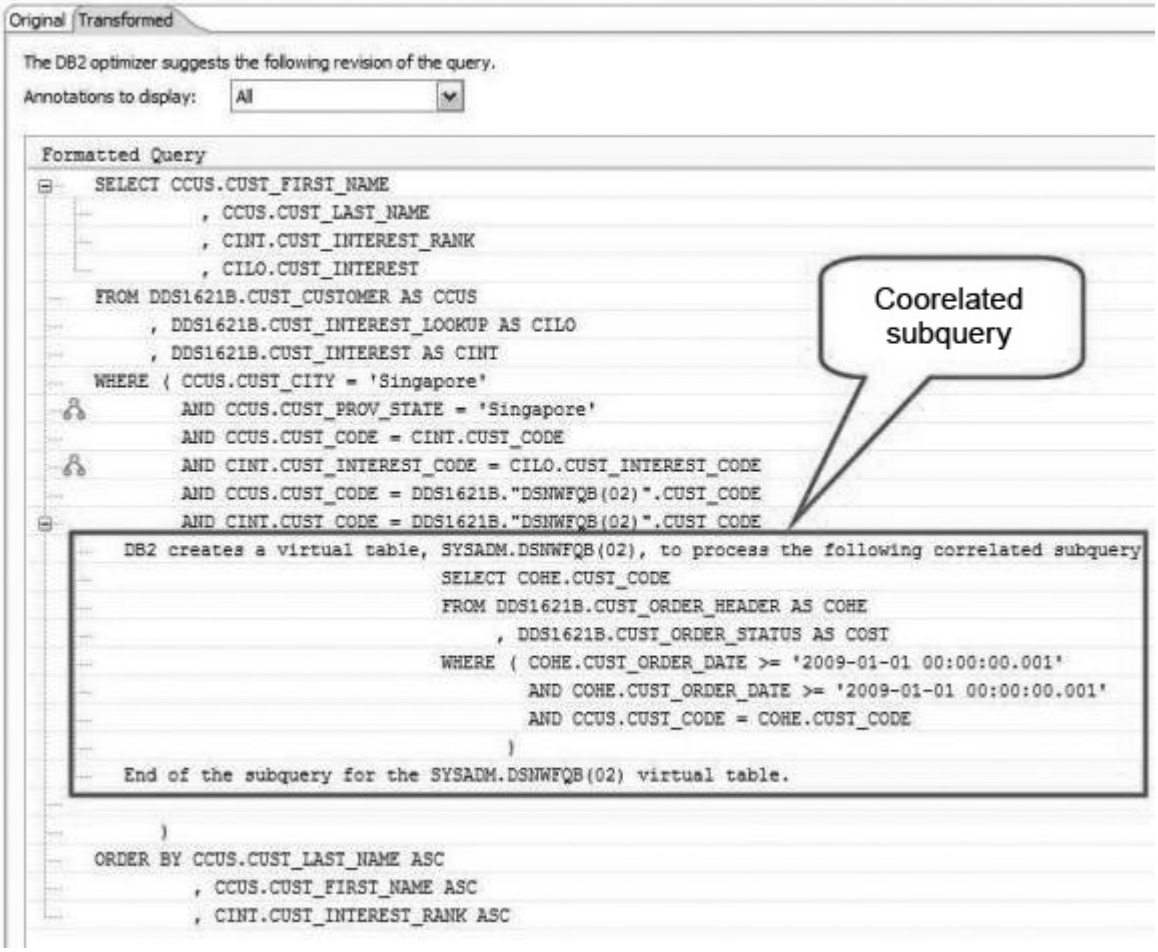
- Truy vấn truy cập vào ba bảng sau: CUST_CUSTOMER, CUST_INTEREST và CUST_INTEREST_LOOKUP để nhận được tên khách hàng đủ điều kiện và thông tin quan tâm.
- Ba bảng được nối với các biến vị ngữ bằng nhau.
- Có ba biến vị ngữ trên bảng CUST_CUSTOMER. Hai biến vị ngữ đầu tiên là các biến vị ngữ bằng nhau đơn giản (CCUS.CUST_CITY = 'Singapore', CCUS.CUST_PROV_STATE = 'Singapore'). Biến vị ngữ thứ ba là một biến vị ngữ IN-list, trong đó có một truy vấn con không tương quan:
    - Truy vấn con truy cập các bảng CUST_ORDER_HEADER và CUST_ORDER_STATUS để nhận được mã khách hàng đủ điều kiện, và hai bảng đó được nối với một biến vị ngữ nối bằng nhau.
    - Có một biến vị ngữ cục bộ phạm vi trên CUST_ORDER_HEADER và một biến vị ngữ cục bộ IN-list trên CUST_ORDER_STATUS.
- Không có biến vị ngữ cục bộ trên hai bảng khác (CUST_INTEREST và CUST_INTEREST_LOOKUP).
- Kết quả được sắp xếp theo tên và sự quan tâm của khách hàng.

Query Tuner cũng làm cho việc phát hiện ra nơi trình tối ưu hóa DB2 đã chuyển đổi một truy vấn trở nên dễ dàng. Xin nhắc lại, các phép chuyển đổi là các điều chỉnh mà trình tối ưu hóa DB2 thực hiện với một truy vấn để cố gắng cải thiện hiệu năng của truy vấn, ví dụ nó có thể thêm một biến vị ngữ cho closure bắc cầu (closure là một hàm hạng nhất có các biến tự do bị kết buột trong một môi trường từ vựng của một ngôn ngữ) để làm cho việc đánh giá chuỗi nối dễ dàng.

Để minh họa, dựa vào một biến vị ngữ ví dụ như A.CUSTNO BETWEEN ? AND ? AND C.CUSTNO = A.CUSTNO, DB2 có thể tìm thấy rằng biến vị ngữ C.CUSTNO BETWEEN ? AND ? cũng phải là đúng, vì vậy phép chuyển đổi truy vấn DB2 có thể thêm biến vị ngữ đó để cho phép nó xem xét chỉ mục khác.

Đối với ví dụ mẫu trong bài này, truy vấn đã chuyển đổi như trong Hình 4.

Hình 4. Truy vấn đã chuyển đổi

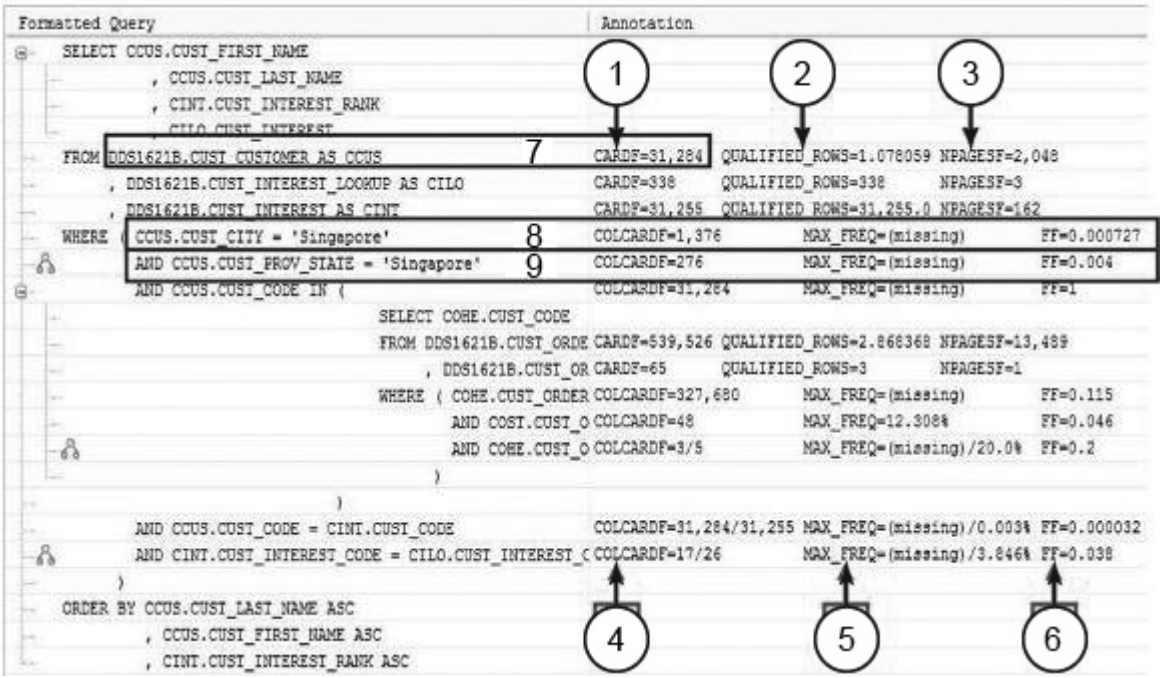


Như bạn thấy, trình tối ưu hóa tạo một bảng ảo để xử lý truy vấn con IN-list. Ngoài ra, truy vấn con không tương quan đã được chuyển đổi thành một truy vấn con tương quan. Đây là một sự tối ưu hóa đã được giới thiệu trong DB2 cho z/OS V9.1, cho phép DB2 tối ưu hóa toàn bộ một truy vấn chứ không chỉ là một số khối truy vấn độc lập. Khi tối ưu hóa toàn bộ một truy vấn, DB2 có thể xem xét hiệu quả của một khối truy vấn trên một khối truy vấn khác, và có thể xem xét các khối truy vấn sắp xếp lại để xác định một đường dẫn truy vấn tối ưu.

Chú thích truy vấn vấn đề

Cùng với các biến vị ngữ SQL đã định dạng và các tham khảo bảng, Query Tuner (Trình điều chỉnh truy vấn) gồm các chú thích về số liệu thống kê danh mục và các thông tin ước tính giá có liên quan, ví dụ như cardinality (số các yếu tố trong một tập) và các hàng đủ điều kiện đã đánh giá, như thể hiện trong Hình 5. Việc có sẵn thông tin này một cách dễ dàng có thể giúp các DBA tăng tốc độ phân tích và giảm thời gian chết với các tình huống khẩn cấp.

Hình 5. Truy vấn vấn đề có chú thích



Đối với mỗi tham khảo bảng trong truy vấn, Query Tuner thêm các chú thích sau:

- CARDF (xem số 1 trong Hình 5): cardinality bảng, biểu thị tổng số hàng trong bảng. CUST_CUSTOMER là bảng lớn nhất (31.284), trong khi CUST_INTEREST_LOOKUP là bảng nhỏ nhất (338).
- QUALIFIED_ROWS (số 2 trong Hình 5): Số các hàng đủ điều kiện sau khi áp dụng các biến vị ngữ cục bộ trên bảng đó. Mặc dù bảng CUST_CUSTOMER và CUST_INTEREST hầu như có cùng một cardinality bảng, các hàng đủ điều kiện với bảng CUST_CUSTOMER chỉ hơn 1, có nghĩa là có các biến vị ngữ cục bộ có khả năng chọn lọc cao trên bảng đó. Trái ngược lại, với bảng CUST_INTEREST số các hàng đủ điều kiện là giống như cardinality (31.255). Điều này cho biết rằng không có quá trình lọc nào trên bảng đó. Điều này có thể giải thích được vì trên bảng CUST_CUSTOMER có hai biến vị ngữ cục bộ trong khi trên bảng CUST_INTEREST không có các biến vị ngữ cục bộ nào.
- NPAGESF (số 3 trong Hình 5): Tổng số các trang trên đó các hàng của bảng này xuất hiện.

Với mỗi biến vị ngữ trong truy vấn, Query Tuner thêm các chú thích số liệu thống kê cho các cột được tham khảo trong biến vị ngữ đó và cũng thêm ước tính giá cho biến vị ngữ đó:

- COLCARDF (xem số 4 trong Hình 5): cardinality cột, biểu thị số lượng ước tính của các giá trị khác nhau trong cột. Nếu biến vị ngữ đó chứa nhiều hơn một cột, thì tách rời cardinality cột cho từng cột đã tham khảo bằng dấu gạch chéo ngược ("/") theo đúng thứ tự xuất hiện các cột trong biến vị ngữ.

- MAX_FREQ (số 5 trong Hình 5): tần số tối đa của tất cả các giá trị cột có thể. Tần số cho một giá trị cột cụ thể là tỷ lệ phần trăm của các hàng trong bảng có chứa các giá trị cụ thể cho một cột. Ví dụ, nếu có năm giá trị khác nhau cho một cột (COLCARDF = 5), và nếu dữ liệu được phân bố đồng đều, MAX_FREQ bằng khoảng 20% vì mỗi giá trị cột khác nhau điền vào 20% các hàng của bảng. Nếu cardinality cột là 5 và MAX_FREQ vẫn còn cách xa hơn 20%, có nghĩa là dữ liệu trong bảng phân bố không đều trên cột đó. Tóm lại, có sự không đối xứng dữ liệu trên cột.
- FF (xem số 6 trong Hình 5): Hệ số lọc với biến vị ngữ. Hệ số lọc là một số giữa 0 và 1 đánh giá tỷ lệ phần trăm của các hàng trong một bảng có biến vị ngữ là đúng. Hệ số lọc cho biết biến vị ngữ có khả năng chọn lựa như thế nào. Biến vị ngữ càng có khả năng chọn lọc nhiều hơn, thì biến vị ngữ sẽ càng được áp dụng sớm hơn.

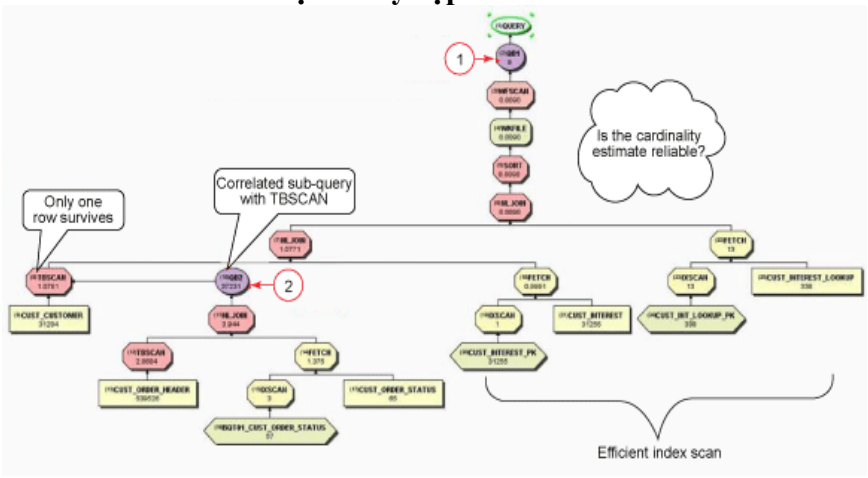
Phân tích kế hoạch truy cập truy vấn

Query Tuner cung cấp một sự hiển thị trực quan về quá trình xử lý mà máy chủ dữ liệu của bạn sử dụng để chạy một truy vấn. Sự hiển thị trực quan này được gọi là biểu đồ kế hoạch truy cập. Từ biểu đồ kế hoạch truy cập, bạn có thể thấy những lựa chọn nào mà trình tối ưu đã thực hiện đối với cách sẽ xử lý truy vấn và lý do cơ bản cho những lựa chọn đó. Biểu đồ gồm các nút biểu diễn các bảng, các chỉ mục, các hoạt động và dữ liệu được trả về. Các nút được bố trí và được kết nối bởi các liên kết cho biết dòng chảy của quá trình này. Biểu đồ được đọc từ trái sang phải, từ dưới lên trên. Và mỗi nút được chú thích bằng số liệu thống kê, các giá ước tính, thông tin về độ chọn lọc và v.v.. được sử dụng để xác định dòng chảy của kế hoạch truy cập.

Việc hiểu biết về kế hoạch truy cập là quan trọng để hiểu biết và tác động đến hiệu năng, cũng như để ổn định hiệu năng. Hãy tham khảo bài trước trong loạt bài này để biết thêm thông tin về đọc và hiểu các đường dẫn truy cập.

Hình 6 là biểu đồ đường dẫn truy cập được Query Tuner tạo cho truy vấn mẫu trong bài này.

Hình 6. Biểu đồ kế hoạch truy cập



(Xem ảnh lớn hơn của Hình 6.)

Từ biểu đồ kế hoạch truy cập trong Hình 6, bạn có thể thấy như sau:

- Truy vấn này chứa hai khối truy vấn, QB1 (xem số 1 trong Hình 6) và QB2 (xem số 2 trong Hình 6). QB2 biểu diễn truy vấn con IN-list trong khi QB1 là truy vấn con chính.
- QB2 được nối với bảng CUST_CUSTOMER trong khối truy vấn bên ngoài QB1, có nghĩa là truy vấn con IN-list đã được chuyển đổi làm truy vấn con tương quan, mặc dù nó là một truy vấn con không tương quan trong truy vấn ban đầu.
- Có thể tóm tắt kế hoạch truy cập với QB1 như sau: TBSCAN(CUST_CUSTOMER) NLJ ISCAN(CUST_INTEREST) NLJ ISCAN(CUST_INTEREST_LOOKUP)
    - 3 bảng trong QB1 được nối bằng phép nối vòng lặp lồng nhau (NLJ)
    - 3 bảng trong QB1 được nối trong chuỗi nối sau: CUST_CUSTOMER -> CUST_INTEREST -> CUST_INTEREST_LOOKUP
    - Truy cập bảng CUST_CUSTOMER bằng cách quét bảng trong khi truy cập bảng CUST_INTEREST và CUST_INTEREST_LOOKUP bằng quét chỉ mục.
- Có thể tóm tắt kế hoạch truy cập với QB2 như sau: TBSCAN(CUST_ORDER_HEADER) NLJ ISCAN(CUST_ORDER_STATUS).
    - 2 bảng QB2 được nối bằng phép nối vòng lặp lồng nhau (NLJ)
    - 2 bảng QB2 được nối trong chuỗi nối sau: CUST_ORDER_HEADER -> CUST_ORDER_STATUS
    - Truy cập bảng CUST_ORDER_HEADER bằng quét bảng trong khi truy cập CUST_ORDER_STATUS bằng quét chỉ mục.

Từ biểu đồ kế hoạch truy cập trong Hình 6, bạn có thể thực hiện một số phân tích hiệu năng ban đầu:

- Việc truy cập vào các bảng bên trong (CUST_INTEREST và CUST_INTEREST_LOOKUP) bằng quét chỉ mục. Đó là một kế hoạch truy cập hiệu quả hợp lý.
- Việc truy cập vào bảng hàng đầu của truy vấn con bên ngoài (CUST_CUSTOMER) và truy vấn con bên trong (CUST_ORDER_HEADER) bằng quét bảng, có thể có một vấn đề tiềm năng:
    - Cardinality bảng với bảng CUST_CUSTOMER bằng khoảng 30000. Tuy nhiên, vì bảng này là một bảng hàng đầu và sẽ được truy cập bằng quét bảng chỉ một lần, bảng này có thể gây ra một số vấn đề hiệu năng, nhưng bảng này không phải là một thảm họa.
    - Truy vấn con bên trong là một truy vấn con tương quan. Tùy thuộc vào có bao nhiêu hàng đủ điều kiện được trả về từ bảng bên ngoài (CUST_CUSTOMER), truy vấn con này có thể được truy cập nhiều lần. Từ hoặc chú thích trong Hình 5 hoặc biểu đồ kế hoạch truy cập trong Hình 6, bạn có thể thấy rằng các hàng đủ điều kiện được đánh giá cho bảng CUST_CUSTOMER là 1; nói cách khác, trình tối ưu hóa cho rằng sẽ chỉ quét bảng CUST_ORDER_HEADER một lần. Dựa vào cardinality bảng xấp xỉ là 50000; bảng này sẽ không phải là một thảm họa về hiệu năng.

Như vậy đến nay, bạn chắc chắn rằng bảng CUST_CUSTOMER sẽ được quét chỉ một lần vì nó là một bảng hàng đầu trong truy vấn con bên ngoài. Tuy nhiên bạn không chắc chắn liệu bảng

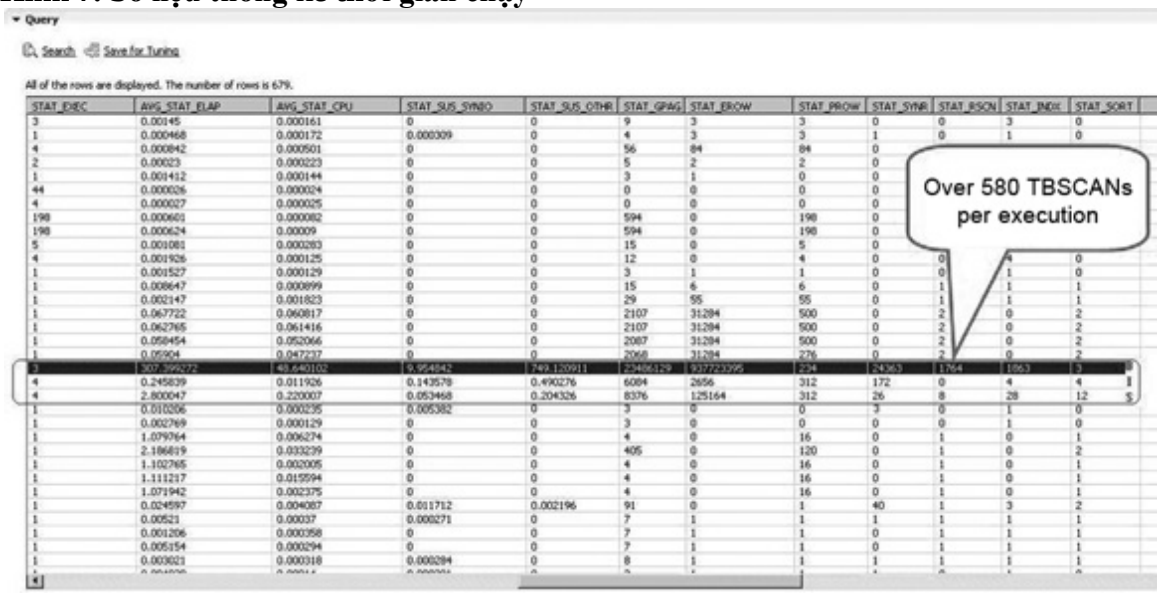
CUST_ORDER_HEADER sẽ được quét chỉ một lần không, do các hàng đủ điều kiện từ bảng CUST_CUSTOMER được tính toán với số liệu thống kê và các biến vị ngữ hiện có:

- Cardinality bảng với bảng CUST_CUSTOMER là 31284 (xem số 7 trong Hình 5).
- Có hai biến vị ngữ cục bộ trên bảng này có khả năng chọn lọc cao:
  - CCUS.CUST_CITY = 'Singapore', FF=0.00727 (xem số 8 trong Hình 5)
  - CCUS.CUST_PROV_STATE = 'Singapore', FF=0.004 (xem số 9 trong Hình 5)
- Vì vậy các hàng đủ điều kiện được trình tối ưu hóa đánh giá xấp xỉ bằng 31284*0,00727*0,004 = 1.

Điều gì sẽ xảy ra nếu các hàng đủ điều kiện được đánh giá từ bảng CUST_CUSTOMER là sai? Ví dụ, điều gì sẽ xảy ra nếu hai biến vị ngữ cục bộ không được trình tối ưu hóa đánh giá bằng khả năng chọn lựa? Điều này có thể gây ra các vấn đề hiệu năng nghiêm trọng do đã có thể truy cập bảng CUST_ORDER_HEADER nhiều lần.

Một cách để xác nhận hợp lệ nghẽn cổ chai hiệu năng đáng ngờ là nhờ xem xét số liệu thống kê thời gian chạy của truy vấn đó, có thể thu được bằng cách bật chức năng theo vết hiệu năng trên IFCID 318. Một lựa chọn khác là sử dụng Query Tuner để bắt giữ các câu lệnh từ bộ nhớ truy cập nhanh (cache) câu lệnh và để xem thông tin thời gian chạy của câu lệnh đó, như trong Hình 7.

Hình 7. Số liệu thống kê thời gian chạy



(Xem ảnh lớn hơn của Hình 7.)

Dòng được đánh dấu (kết thúc với một chữ "B" trong Hình 7) cho thấy thông tin thời gian chạy với truy vấn mẫu trong bài này. Như bạn có thể thấy, truy vấn này được thực hiện ba lần, và thời gian trôi qua trung bình khoảng 307 giây, rất chậm. Tổng số lần quét bảng (STAT_RSCN) khoảng 1764, tức là, nhiều hơn 580 lần quét bảng (1764/3) cho mỗi lần thực hiện. Điều này còn cách xa hơn so với những gì có thể được đánh giá từ biểu đồ kế hoạch truy cập với khoảng 2 làn

quét (một với bảng CUST_CUSTOMER, và một với bảng CUST_ORDER_HEADER). Điều này khẳng định thêm nghi ngờ của chúng ta là các hàng đủ điều kiện theo số đã đánh giá từ bảng CUST_CUSTOMER còn cách xa thực tế.

Một cách khác để xác nhận hợp lệ điều này là ban hành một truy vấn như sau để tính giá trị thực của các hàng đủ điều kiện.

Liệt kê 2. Truy vấn đếm

```
SELECT COUNT(*)
FROM CUST_CUSTOMER AS CCUS
WHERE CCUS.CUST_CITY = 'Singapore' AND CCUS.CUST_PROV_STATE = 'Singapore'
```

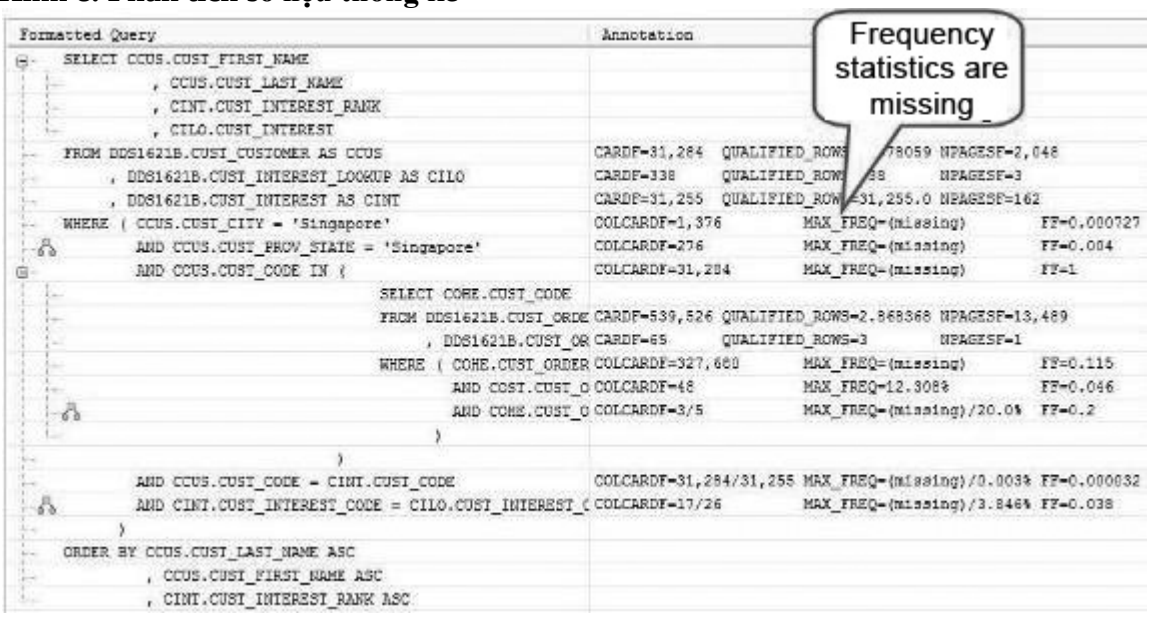
Kết quả của truy vấn count (đếm) nói trên cho thấy rằng có khoảng 588 hàng đủ điều kiện từ bảng CUST_CUSTOMER. Nói cách khác, trình tối ưu hóa đánh giá quá cao độ chọn lọc của các biến vị ngữ cục bộ trên bảng này. Trong các phần sau của bài này, bạn sẽ phân tích truy vấn vấn đề theo các quan điểm về số liệu thống kê, biến vị ngữ và chỉ mục, sau đó chỉ ra tại sao lại xảy ra đánh giá quá cao, cũng như làm thế nào để giải quyết nó.

Thực hiện phân tích số liệu thống kê

Từ thông tin chú thích truy vấn, bạn có thể dễ dàng xem các loại số liệu thống kê nào có sẵn, và các loại số liệu thống kê nào còn thiếu. Với truy vấn trong bài này, số liệu thống kê cơ bản của các bảng, các cột và các chỉ mục tham khảo được thu thập, ví dụ như cardinality bảng, cardinality cột, v.v..

Mặt khác, một số số liệu thống kê phân phối, như tần số cột, chưa bao giờ được thu thập. Ví dụ, như trong Hình 8, MAX_FREQ với cột CUST_CITY trong biến vị ngữ `CCUS.CUST_CITY = 'Singapore'` được thể hiện là còn thiếu. Số liệu thống kê còn thiếu có thể làm cho trình tối ưu hóa đánh giá quá cao hoặc đánh giá quá thấp độ chọn lọc của một biến vị ngữ, và cuối cùng chọn một đường dẫn truy cập không hiệu quả.

Hình 8. Phân tích số liệu thống kê



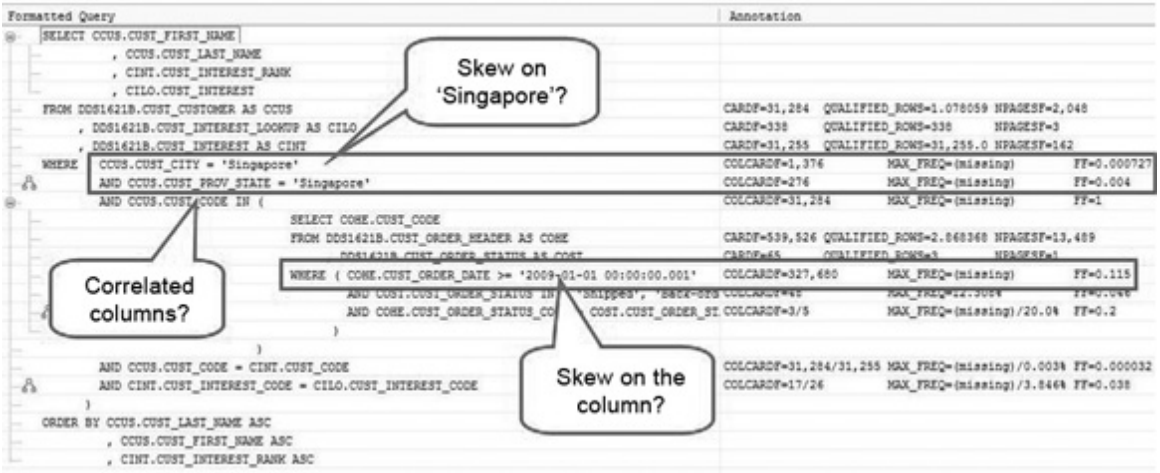
Một vấn đề tương tự có thể xảy ra ngay cả khi số liệu thống kê được thu thập từ trước nhưng đã lâu không được làm mới đến mức bây giờ chúng đã quá hạn. Điều này đặc biệt đúng nếu dữ liệu đã thay đổi đáng kể từ lần thu thập số liệu thống kê gần nhất. Có một thuộc tính RUNSTATS TIMESTAMP trên nút bảng hoặc nút chỉ mục của biểu đồ kế hoạch truy cập cho biết thời gian cuối cùng đã thu thập số liệu thống kê. Lưu ý, bạn có thể quy định một ngưỡng theo các mức ưu tiên của Query Tuner để định nghĩa xem số liệu thống kê cũ thế nào trước khi chúng được coi là quá hạn. Theo mặc định, nếu số liệu thống kê cũ quá 1 năm thì sẽ được xác định là quá hạn.

Nếu thu thập số liệu thống kê khác nhau tại các thời điểm khác nhau, thì có thể dẫn đến một tình huống ở đó số liệu thống kê không phù hợp với nhau trên DB2 cho z/OS. Ví dụ, nếu thu thập số liệu thống kê mức bảng trên một bảng cụ thể, rồi chèn một số các hàng đáng kể vào bảng này, thì việc thu thập số liệu thống kê với các chỉ mục này chỉ có thể kết thúc với các cột chỉ mục có cardinality cột lớn hơn cardinality bảng. Số liệu thống kê không phù hợp cũng có thể đánh lừa trình tối ưu hóa khiến cho trình này chọn một đường dẫn truy cập xấu.

Thực hiện phân tích biến vị ngữ

Trong khi đó, bạn cũng có thể thực hiện phân tích biến vị ngữ để tìm xem có bất kỳ vấn đề tiềm năng nào không trong khung nhìn chú thích, như thể hiện trong Hình 9.

Hình 9. Phân tích biến vị ngữ



(Xem ảnh lớn hơn của Hình 9.)

Theo thông tin chú thích trong hình này, tham khảo bảng CUST_CUSTOMER (CCUS) có hai biến vị ngữ cục bộ có khả năng chọn lọc cao, với hệ số lọc tương ứng là 0,00727 và 0,004.

Vì không thu thập số liệu thống kê tần số nào (MAX_FREQ=(missing)), nên hệ số lọc của mỗi biến vị ngữ được tính toán dựa trên giả định rằng dữ liệu được phân bố đồng đều trên các cột. Ví dụ, cardinality cột của CUST_CITY là 1376, do đó hệ số lọc của biến vị `CCUS.CUST_CITY='Singapore'` được tính là 1/1376=0,00727.

Người ta có thể đặt bao nhiêu lòng tin vào đánh giá về độ chọn lọc của biến vị ngữ? Nếu dữ liệu không đối xứng trên các cột, ví dụ nếu phần lớn dữ liệu có một tên thành phố là `Singapore`, thì độ chọn lọc hiện tại có thể được đánh giá quá cao. Để có được một đánh giá chính xác hơn, bạn cần phải thu thập số liệu thống kê tần số trên các cột CUST_CITY và CUST_PROV_STATE của bảng CUST_CUSTOMER.

Vấn đề tiềm năng khác là biến vị ngữ cục bộ trên tham khảo bảng CUST_ORDER_HEADER (COHE) trong truy vấn con bên trong (xem trong Hình 9). Vì nó là một biến vị ngữ phạm vi và không có sẵn số liệu thống kê tần số và hoành đồ, nên tính toán hệ số lọc bằng thông tin của số liệu thống kê cơ bản là HIGH2KEY và LOW2KEY. Việc đánh giá có thể không chính xác nếu có dữ liệu không đối xứng trên cột. Để nhận được một đánh giá chính xác hơn, bạn cần thu thập số liệu thống kê hoành đồ trên cột CUST_ORDER_DATE của bảng CUST_ORDER_STATUS_CODE.

Bây giờ bạn đã phân tích độ chọn lọc của các biến vị ngữ cục bộ, chúng ta hãy xem độ chọn lọc bảng.

Với ba bảng trong truy vấn con bên ngoài, không có biến vị ngữ cục bộ nào trên bảng CUST_INTEREST_LOOKUP (CILO) và CUST_INTEREST (CInt); do đó nếu cả hai bảng đó đã được xử lý như bảng hàng đầu, thì độ chọn lọc bảng bằng không, rất xấu.

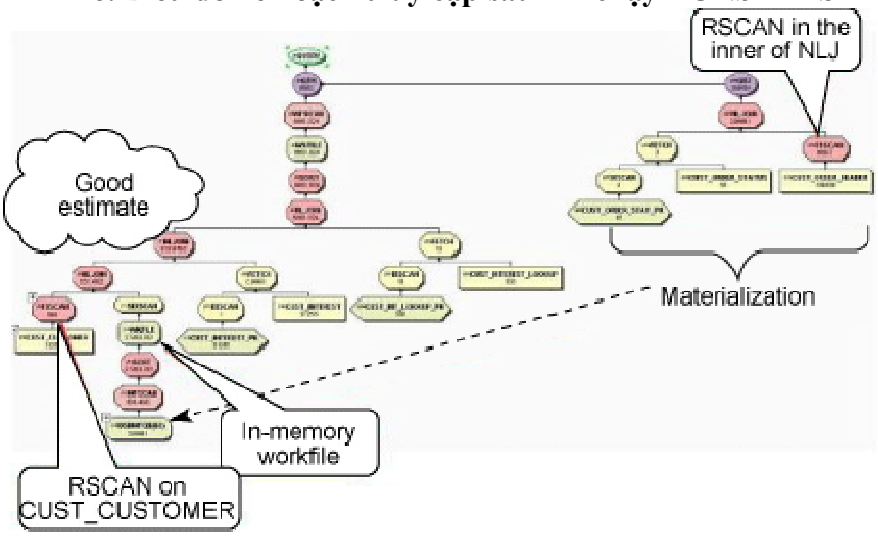
Nếu xử lý bảng CUST_CUSTOMER (CCUS) như là bảng hàng đầu, dựa vào hai biến vị ngữ cục bộ, độ chọn lọc bảng của CUST_CUSTOMER là 0,00727 * 0,004, và các hàng đủ điều kiện của bảng CUST_CUSTOMER xấp xỉ bằng 1 (31284 * 0,00727 * 0,004). Điều này có vẻ có độ chọn lọc cao, nhưng giá trị này được tính với giả định rằng hai cột (CUST_CITY, CUST_PROV_STATE) không tương quan với nhau, là không đúng sự thật. Vì Singapore là một trong những thành phố trực thuộc nước Singapore, nên để nhận được một đánh giá chính xác về độ chọn lọc bảng, bạn cần phải thu thập số liệu thống kê về cardinality và tần số của nhóm cột (CUST_CITY, CUST_PROV_STATE).

Sau khi xử lý bảng CUST_CUSTOMER, có thể coi biến vị ngữ nối `CCUS.CUST_CODE = CINT.CUST_CODE` là một biến vị ngữ cục bộ trên bảng CUST_INTEREST (CINT). Căn cứ vào cardinality bảng CUST_INTEREST xấp xỉ bằng 31255, bạn có thể thấy rằng với mỗi hàng đủ điều kiện từ bảng CUST_CUSTOMER, chỉ có một bản ghi phù hợp trong CUST_INTEREST. Nghĩa là, bảng CUST_INTEREST có thể có độ chọn lọc cao thông qua biến vị ngữ nối.

Sau khi xử lý bảng CUST_CUSTOMER và CUST_INTEREST, có thể coi biến vị ngữ nối `CINT.CUST_INTEREST_CODE = CILO.CUST_INTEREST_CODE` là một biến vị ngữ cục bộ trên bảng CUST_INTEREST_LOOKUP (CILO). Dựa vào cardinality bảng của bảng CUST_INTEREST_LOOKUP xấp xỉ bằng 338, bạn có thể thấy rằng với mỗi hàng đủ điều kiện của bảng CUST_INTEREST, có khoảng 13 (338/26) bản ghi phù hợp trong bảng CUST_INTEREST_LOOKUP, đó là một mức chọn lọc khá tốt.

Với số liệu thống kê và phân tích biến vị ngữ ở trên, bạn có một ý niệm rõ ràng về cần thu thập số liệu thống kê nào. Sau khi thu thập số liệu thống kê đó, bạn sẽ nhận được biểu đồ kế hoạch truy cập mới được thể hiện trong Hình 10.

Hình 10. Biểu đồ kế hoạch truy cập sau khi chạy RUNSTATS

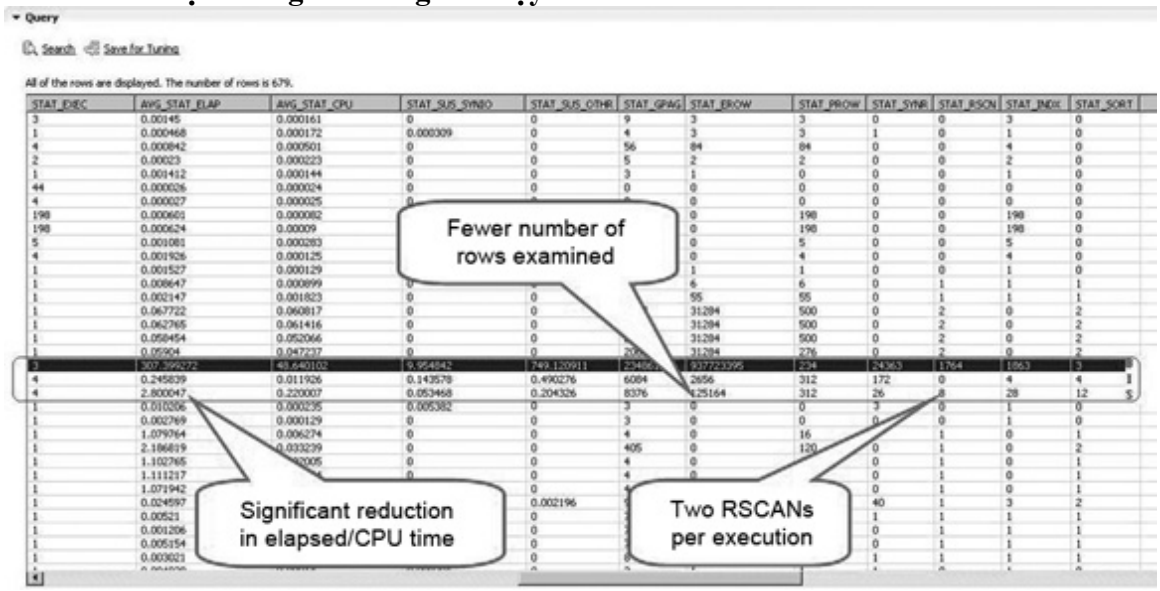


Từ biểu đồ kế hoạch truy cập mới này, bạn có thể thấy như sau:

- Do đã thu thập số liệu thống kê trên nhóm cột (CUST_CITY và CUST_PROV_STATE) trên bảng CUST_CUSTOMER, bây giờ trình tối ưu hóa DB2 biết số hàng chính xác đủ điều kiện sau khi lọc cục bộ là 590 thay vì 1.
- Chuỗi nối của truy vấn con bên trong được thay đổi, chuỗi ban đầu là CUST_ORDER_HEADER-> CUST_ORDER_STATUS, now it is CUST_ORDER_STATUS -> CUST_ORDER_HEADER; vì số liệu thống kê hoành đồ trên cột CUST_ORDER_DATE của bảng CUST_ORDER_HEADER đã được thu thập.
- Với các hàng đủ điều kiện chính xác từ CUST_CUSTOMER, truy vấn con bên trong được xử lý như là truy vấn con không tương quan, được cụ thể hóa vào một tệp làm việc và sau đó được truy cập bằng quét chỉ mục rời rạc trong bộ nhớ.

Dòng được kết thúc bằng một chữ "S" trong Hình 11 cho thấy thông tin thời gian chạy của truy vấn mẫu sau khi chạy RUNSTATS. Thời gian trôi qua trung bình xấp xỉ 2,8 giây, một sự cải tiến hiệu năng đáng kể so với thời gian đó trước khi chạy RUNSTATS (khoảng 307 giây).

Hình 11. Số liệu thống kê thời gian chạy sau RUNSTATS



(Xem ảnh lớn hơn của Hình 11.)

Cũng như trong Hình 11, bạn có thể thấy rằng vẫn còn có hai lần quét bảng cho mỗi lần thực hiện truy vấn này. Từ biểu đồ kế hoạch truy cập trong Hình 10, bạn có thể dễ dàng tìm ra hai lần quét bảng đến từ bảng CUST_CUSTOMER và CUST_ORDER_HEADER tương ứng. Bạn sẽ thấy cách có thể sửa chữa việc này nhờ việc phân tích chỉ mục trong phần tiếp theo.

Thực hiện phân tích chỉ mục

Optim Query Tuner cung cấp các báo cáo theo định dạng HTML và định dạng văn bản có chứa thông tin về các bảng, các chỉ mục, và các biến vị ngữ có liên quan với một câu lệnh SQL cụ thể. Với truy vấn mẫu trong bài này, Hình 12 cho thấy báo cáo chỉ mục, mô tả các chỉ mục hiện có của các bảng tham khảo.

Hình 12. Bản ghi chỉ mục

lock:1 Plan:1					
ne	Correlation Name				
.CUST_CUSTOMER	CCUS				
	Index Name	Index Only	ONE_FETCH	Equal Unique	GB_OB_DIST
	DDS1621B.BQT01_CUST_CUSTOMER	N	N	N	N
	Key Column Name	Key Order	COLCARDF	MCARDF	FF
	CUST_AGE	Ascending	54	54	
	Index Name	Index Only	ONE_FETCH	Equal Unique	GB_OB_DIST
	DDS1621B.BQT02_CUST_CUSTOMER	N	N	N	N
	Key Column Name	Key Order	COLCARDF	MCARDF	FF
	CUST_PROV_STATE_CODE	Ascending	117	117	
	Index Name	Index Only	ONE_FETCH	Equal Unique	GB_OB_DIST
	DDS1621B.BQT03_CUST_CUSTOMER	N	N	N	N
	Key Column Name	Key Order	COLCARDF	MCARDF	FF
	GENDER_CODE	Ascending	3	3	
	MARITAL_STATUS_CODE	Ascending	4	-1	
	CUST_AGE	Ascending	54	228	
	Index Name	Index Only	ONE_FETCH	Equal Unique	GB_OB_DIST
	DDS1621B.IDX_CUST_CUSTOMER	N	N	N	N
	Key Column Name	Key Order	COLCARDF	MCARDF	FF
	CUST_CODE	Ascending	31284	31284	0.000032
lock:1 Plan:2					
ne	Correlation Name				
.CUST_INTEREST	CINT				
	Index Name	Index Only	ONE_FETCH	Equal Unique	GB_OB_DIST
	DDS1621B.CUST_INTEREST_PK	N	N	N	N

Để thuận tiện cho việc thảo luận, các chỉ mục hiện có được tóm tắt trong Bảng 1.

Bảng 1. Các chỉ mục hiện có

Tên bảng	Tên chỉ mục	Cột chỉ mục
CUST_CUSTOMER	BQT01_CUST_CUSTOMER BQT02_CUST_CUSTOMER BQT03_CUST_CUSTOMER IDX_CUST_CUSTOMER	CUST_AGE CUST_PROV_STATE_CODE GENDER_CODE, MARITAL_STATUS_CODE, CUST_AGE CUST_CODE
CUST_INTEREST	CUST_INTEREST_PK	CUST_CODE, CUST_INTEREST_CODE
CUST_INTEREST_LOOKUP	CUST_INT_LOOKUP_PK	CUST_INTEREST_CODE, CUST_INTEREST_LANGUAGE
CUST_ORDER_HEADER	BQT01_CUST_ORDER_HEADER	CRDT_METHOD_CODE

CUST_ORDER_STATUS	CUST_ORDER_HEADER_PK	CUST_ORDER_NUMBER
	BQT01_CUST_ORDER_STATUS	CUST_ORDER_STATUS, CUST_ORDER_STATUS_LANGUAGE
	CUST_ORDER_STAT_PK	CUST_ORDER_STATUS_CODE, CUST_ORDER_STATUS_LANGUAGE

Bây giờ chúng ta hãy xem xét các biến vị ngữ với bảng CUST_CUSTOMER khi được trích xuất từ truy vấn đã định dạng trong Hình 3:

- CCUS.CUST_CITY = 'Singapore'
- CCUS.CUST_PROV_STATE = 'Singapore'
- CCUS.CUST_CODE = CINT.CUST_CODE

Dựa vào các chỉ mục hiện có trong Bảng 1 và biến vị ngữ ở trên, chỉ có biến vị ngữ nối (CCUST.CUST_CODE = CINT.CUST_CODE) có thể được lợi từ chỉ mục hiện có `IDX_CUST_CUSTOMER`. Hai biến vị ngữ cục bộ không thể được lợi từ việc phối hợp hoặc kiểm tra các lần quét chỉ mục từ bất kỳ các chỉ mục hiện có nào.

Để hỗ trợ cả hai biến vị ngữ cục bộ và các biến vị ngữ nối với các lần quét chỉ mục, bạn cần tạo một chỉ mục trên các cột CUST_CITY, CUST_PROV_STATE và CUST_CODE.

Đối với bảng CUST_INTEREST và CUST_INTEREST_LOOKUP, mỗi bảng có một biến vị ngữ nối, và chỉ mục hiện tại có thể hỗ trợ một lần quét chỉ mục phù hợp trên các biến vị ngữ nối như được hiển thị dưới đây.

- CCUS.CUST_CODE = CINT.CUST_CODE
- CINT.CUST_INTEREST_CODE = CILO.CUST_INTEREST_CODE

Đối với bảng CUST_ORDER_HEADER, hai trong số các chỉ mục hiện có không hỗ trợ các biến vị ngữ cục bộ và nối trên bảng, như được cho thấy dưới đây.

- COHE.CUST_ORDER_DATE >= '2009-01-01 00:00:00.001'
- COHE.CUST_ORDER_STATUS_CODE = COST.CUST_ORDER_STATUS_CODE

Để hỗ trợ cả hai biến vị ngữ cục bộ và biến vị ngữ nối với một lần quét chỉ mục, bạn cần tạo một chỉ mục trên các cột CUST_ORDER_DATE và CUST_ORDER_STATUS_CODE. Ngoài ra, để có được hiệu năng tốt hơn với chỉ mục chỉ truy cập, bạn cũng có thể thêm CUST_CODE làm một cột chính. Khi cho rằng bảng này là một phần của truy vấn con IN-list, nếu CUST_CODE là cột chính đầu tiên, thì trình tối ưu hóa DB2 có thể xem xét việc tối ưu hóa truy vấn chung để truy cập truy vấn con như là truy vấn con đã tương quan.

Đối với bảng CUST_ORDER_STATUS, chỉ mục đầu tiên là BQT01_CUST_ORDER_STATUS hỗ trợ truy cập chỉ mục cho biến vị ngữ cục bộ, nhưng không thể hỗ trợ cho biến vị ngữ nối. Chỉ mục thứ hai CUST_ORDER_STAT_PK có thể hỗ trợ truy cập chỉ mục cho biến vị ngữ nối nhưng lại không thể hỗ trợ cho biến vị ngữ cục bộ:

- COST.CUST_ORDER_STATUS IN ('Shipped', 'Back-ordered', 'In-process')
- COHE.CUST_ORDER_STATUS_CODE = COST.CUST_ORDER_STATUS_CODE

Để đạt được hiệu năng tốt hơn trên bảng CUST_ORDER_STATUS, bạn cần tạo ra một chỉ mục với cả hai cột CUST_ORDER_STATUS_CODE và CUST_ORDER_STATUS.

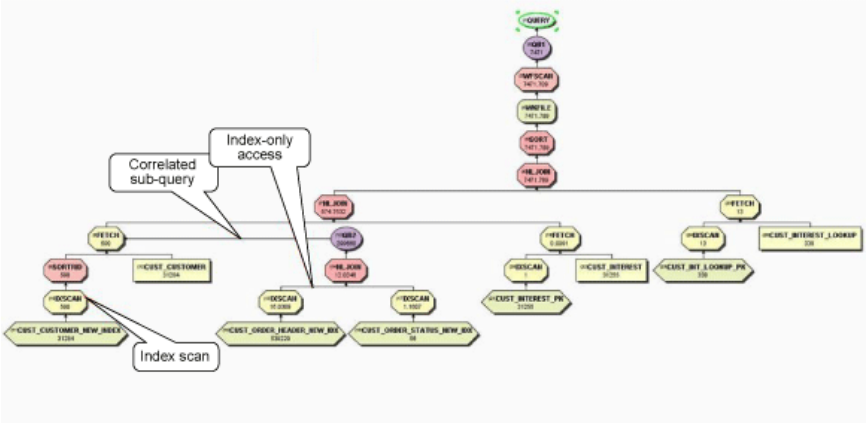
Dựa vào phân tích chỉ mục ở trên, chúng ta đề xuất ba chỉ mục sau để đạt được hiệu năng tốt hơn:

Bảng 2. Các chỉ mục mới

Tên bảng	Tên chỉ mục	Cột chỉ mục
CUST_CUSTOMER	CUST_CUSTOMER_NEW_INDEX	CUST_CITY, CUST_PROV_STATE, CUST_CODE
CUST_ORDER_HEADER	CUST_ORDER_HEADER_NEW_IDX	CUST_CODE, CUST_ORDER_DATE, CUST_ORDER_STATUS_CODE
CUST_ORDER_STATUS	CUST_ORDER_STATUS_NEW_IDX	CUST_ORDER_STATUS_CODE, CUST_ORDER_STATUS

Sau khi tạo ba chỉ mục mới, bạn sẽ nhận được biểu đồ kế hoạch truy cập mới được thể hiện trong Hình 13.

Hình 13. Biểu đồ kế hoạch truy cập sau khi tạo các chỉ mục

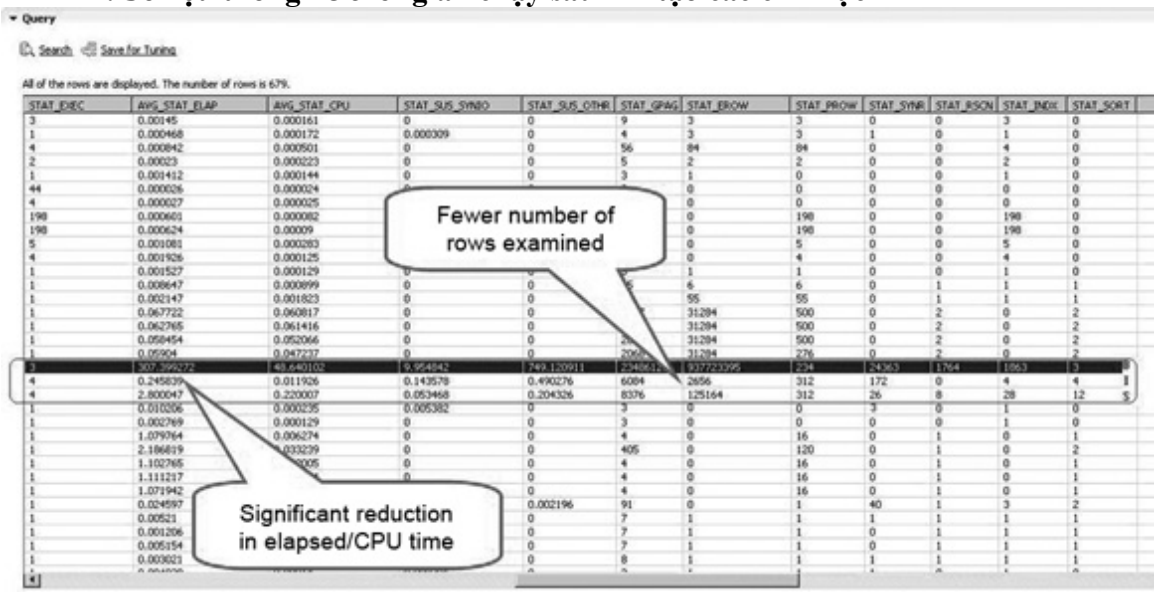


(Xem ảnh lớn hơn của Hình 13.)

Với các chỉ mục mới được tạo ra, truy cập tất cả các bảng bằng các lần quét chỉ mục. Thay đổi lại truy vấn con thành truy vấn con tương quan, và truy cập hai bảng trong truy vấn con bằng cách sử dụng chỉ mục cung cấp hiệu năng tốt nhất.

Dòng được kết thúc với một chữ "I" trong Hình 14 cho thấy thông tin thời gian chạy của truy vấn mẫu sau khi tạo các chỉ mục mới. Thời gian trôi qua trung bình là khoảng 0,246 giây, một sự cải thiện hơn nữa so với hiệu năng sau khi chạy RUNSTATS (khoảng 2,8 giây).

Hình 14. Số liệu thống kê thời gian chạy sau khi tạo các chỉ mục



(Xem ảnh lớn hơn của Hình 14.)

Với sự phân tích từng bước một trên số liệu thống kê, biến vị ngữ, và chỉ mục, hiệu năng của truy vấn vấn đề được cải thiện đáng kể. Bảng 2 hiển thị một bản tóm tắt về các cải thiện hiệu năng. Hàng 1 cho thấy số liệu thống kê thời gian chạy trước khi điều chỉnh. Hàng 2 cho thấy số liệu thống kê sau khi thu thập số liệu thống kê có liên quan, và hàng 3 cho thấy các kết quả sau khi tạo các chỉ mục đã đề xuất.

Bảng 3. Bảng 3. So sánh hiệu năng

Nhiệm vụ điều chỉnh	Đếm số lần thực hiện	STAT_GPAG	STAT_ELAP	STAT_CPU	AVG_STAT_ELAP	AVG_STAT_CPU
Trước khi điều chỉnh	3	23486129	922,197815	145,920307	307,399272	48,640102
Thu thập số liệu thống kê	4	8376	11,200186	0,88002	2,800047	0,220007
Tạo các chỉ mục	4	6084	0,983357	0,047704	0,245839	0,011926

Tăng tốc độ điều chỉnh truy vấn bằng Optim Query Tuner

Cho đến nay, bạn đã thấy cách bạn có thể phân tích và điều chỉnh một truy vấn vấn đề với sự hỗ trợ của các khả năng Optim Query Tuner, ví dụ như chú thích truy vấn, biểu đồ kế hoạch truy cập, và v.v. Mặc dù các công cụ này rất trực quan và thân thiện với người dùng, bạn vẫn cần có đủ kiến thức cơ bản về trình tối ưu hóa DB2 và đường dẫn truy cập SQL để tận dụng đầy đủ lợi thế của các hàm mạnh mẽ do các công cụ này cung cấp. Tất nhiên, bạn cũng cần dành một số thời gian cho việc phân tích.

Tuy nhiên, trong nhiều trường hợp, bạn không cần thực hiện điều chỉnh truy vấn bằng thủ công. Optim Query Tuner cung cấp một loạt các trình tư vấn để tự động hóa việc điều chỉnh truy vấn bằng cách đưa ra các khuyến cáo, mà bạn có thể xem xét và thực hiện trực tiếp từ sản phẩm này để giải quyết một vấn đề về hiệu năng.

- Trình tư vấn số liệu thống kê: các khuyến cáo để thu thập số liệu thống kê cần thiết để làm cho việc lựa chọn đường dẫn truy cập dễ dàng.
- Trình tư vấn truy vấn: các khuyến cáo để viết lại truy vấn có hiệu quả tốt hơn.
- Trình tư vấn chỉ mục: các khuyến cáo về các chỉ mục có thể giúp cải thiện hiệu năng.

Khuyến cáo: Thông thường, nó đề xuất bạn chạy các trình tư vấn theo thứ tự sau:

1. Chạy trình tư vấn thống kê và hành động ngay để tạo số liệu thống kê hiện tại.
2. Chạy trình tư vấn truy vấn và viết lại SQL nếu cần thiết và có thể.
3. Chạy trình tư vấn chỉ mục và tạo các chỉ mục mới hoặc điều chỉnh các chỉ mục hiện có. Số liệu thống kê chính xác và đầy đủ sẽ giúp đưa ra lời khuyên về đường dẫn truy cập tốt, và tất nhiên không nên thực hiện trình tư vấn chỉ mục mà không có số liệu thống kê tốt.

Các phần sau nghiên cứu chi tiết hơn về các trình tư vấn nói trên.

Thực hiện trình tư vấn số liệu thống kê để cải thiện chất lượng và thu thập số liệu thống kê

Như đã đề cập ở phần đầu của bài này, số liệu thống kê cơ sở dữ liệu là những bằng chứng dựa vào đó để trình tối ưu hóa đưa ra các quyết định về các kế hoạch truy cập. Do đó, nếu số liệu thống kê không chính xác, quá hạn, hay mâu thuẫn nhau, thì trình tối ưu hóa sẽ tạo ra các đánh giá không chính xác cho các giá của các bước trong một kế hoạch truy vấn, dẫn đến hiệu năng kém.

Trong DB2, lệnh `RUNSTATS TABLE ALL INDEX ALL` thu thập tất cả số liệu thống kê như nhau, mà nhiều số liệu thống kê đó có thể không cần thiết cho việc cải thiện hiệu năng truy vấn. Đồng thời, tiện ích này không thu thập số liệu thống kê chính nào đó, ví dụ như số liệu thống kê nhiều cột và phân tán. Thường có các tương quan dữ liệu giữa các cột. Ví dụ, có một sự tương quan mạnh mẽ giữa các cột CUST_CITY và CUST_PROV_STATE với truy vấn mẫu trong bài này. Việc thu thập số liệu thống kê của các cột riêng biệt có thể không đủ để cung cấp thông tin cần thiết, vì vậy bạn cần thu thập số liệu thống kê của nhóm cột.

Nhờ đưa ra khuyến cáo RUNSTATS nên trình tư vấn số liệu thống kê Query Tuner làm cho việc thu thập số liệu thống kê cần thiết trở nên dễ dàng hơn bằng cách xác định trạng thái của số liệu thống kê có vấn đề như sau.

- Số liệu thống kê còn thiếu: Khi vẫn còn thiếu số liệu thống kê, trình tối ưu hóa dùng các giá trị mặc định để xác định các giá, có thể hoàn toàn không chính xác.
- Số liệu thống kê mâu thuẫn nhau: Số liệu thống kê mâu thuẫn nhau có thể làm cho trình tối ưu hóa lấy ra các giá ước tính sai và đưa ra quyết định sai về đánh giá kế hoạch truy cập. Có thể xảy ra số liệu thống kê mâu thuẫn nhau khi những người dùng thu thập số liệu thống kê một phần, ví dụ như thu thập số liệu thống kê bảng và chỉ mục riêng biệt, tại các thời điểm khác nhau.
- Số liệu thống kê quá hạn: Số liệu thống kê cũ có thể không còn biểu diễn trạng thái hiện tại của bảng nữa.

Trình tư vấn số liệu thống kê cũng đánh giá tầm quan trọng tương đối của lời khuyên và cung cấp hai kiểu khuyến cáo sau.

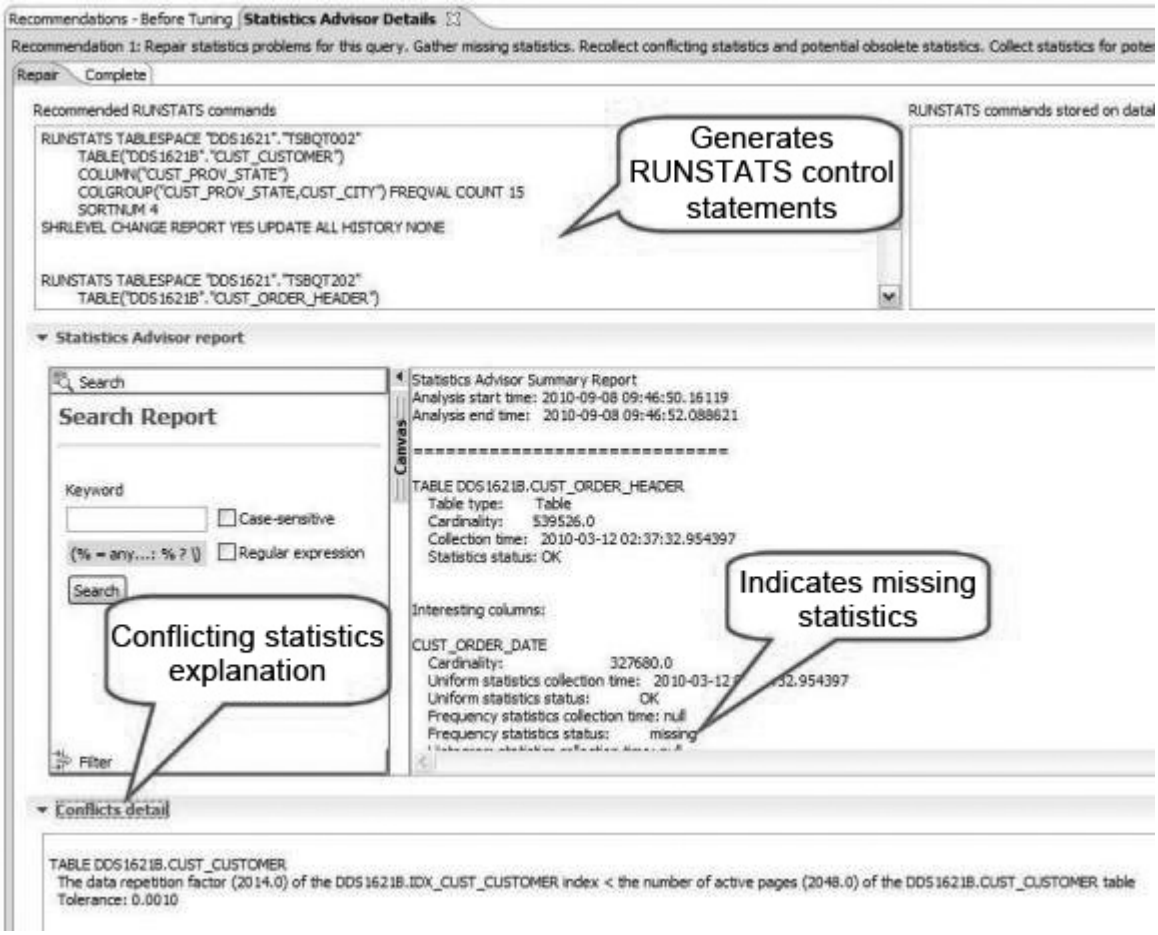
- Sửa chữa: Các khuyến cáo của kiểu này cho biết số liệu thống kê quan trọng vẫn còn thiếu, hoặc có các mâu thuẫn tồn tại giữa số liệu thống kê. Khuyến cáo này bao gồm một lệnh `RUNSTATS` mà bạn có thể chạy để nắm bắt và sửa chữa số liệu thống kê có liên quan.

- Hoàn thành: Các khuyến cáo của kiểu này bao gồm cả các khuyến cáo Sửa chữa cũng như số liệu thống kê khác, có thể được làm mới cho mục đích bảo trì. Khuyến cáo này cung cấp một công việc `RUNSTATS` mà bạn có thể muốn chạy theo định kỳ trong chu kỳ bảo trì của bạn để duy trì sự lành mạnh của số liệu thống kê có liên quan.

Khi bạn tìm đến lời khuyên của một khuyến cáo cụ thể, như thể hiện trong Hình 15, bạn sẽ thấy rằng trình tư vấn số liệu thống kê tạo một lệnh `RUNSTATS` mà bạn có thể sử dụng để thu thập hoặc sửa chữa số liệu thống kê. Bạn có thể lưu chúng để thực hiện sau, hoặc chạy chúng trực tiếp từ máy khách Query Tuner nếu bạn có ủy quyền thích hợp. Lưu ý, để thực hiện các khuyến cáo `RUNSTATS` từ Query Tuner, thủ tục đã lưu `SYSPROC.DSNUTILU` cần có sẵn để dùng và có khả năng hoạt động ở phía máy chủ.

Bạn cũng có thể thấy lời giải thích sau khuyến cáo đó. Trong Hình 15, bạn có thể thấy rằng trình tư vấn đã tìm thấy một số cá thể mâu thuẫn hoặc thiếu số liệu thống kê cũng như đưa ra lời giải thích về lý do trình tư vấn cho rằng chúng đang mâu thuẫn nhau.

Hình 15. Các khuyến cáo của trình tư vấn số liệu thống kê



Bạn đã trải qua cách cải thiện chất lượng số liệu thống kê có nghĩa là bạn đang cung cấp cho trình tối ưu hóa DB2 dữ liệu chính xác mà dựa vào đó để đưa ra các quyết định cơ bản về việc tối

ưu hóa kế hoạch truy cập và do đó có khả năng cải thiện hiệu năng và giảm số lượng CPU. Tuy nhiên, có lợi ích khác nữa. Nhờ chỉ thu thập số liệu thống kê cần thiết, bạn có thể tránh được việc thu thập số liệu thống kê không liên quan đẩy chi phí hoạt động của CPU lên cao và tạo ra tải công việc không cần thiết bên trong các cửa sổ bảo trì có giới hạn. Nói cách khác, trình tư vấn số liệu thống kê có thể giúp cải thiện chất lượng số liệu thống kê cũng như nâng cao hiệu quả của việc thu thập số liệu thống kê.

Thực hiện trình tư vấn truy vấn để cải thiện thiết kế truy vấn

Trước khi lựa chọn đường dẫn truy cập, trình tối ưu hóa DB2 chuyển đổi câu lệnh SQL thành các dạng giống hệt nhau về ngữ nghĩa, ví dụ như bằng cách sử dụng sắp xếp đẩy xuống của biến vị ngữ hoặc closure bắc cầu của biến vị ngữ. Kết quả là, nó có thể cải thiện các đường dẫn truy cập có khả năng. Ngược lại, Query Tuner giúp người viết và người điều chỉnh các truy vấn nhận biết các lỗi và các trường hợp sơ suất bằng cách thực hiện các đề xuất ràng buộc truy vấn hơn nữa, tăng cường sử dụng chỉ mục, và giảm các lần đọc dữ liệu. Trình tư vấn truy vấn tìm kiếm các cơ hội trong một truy vấn để làm như sau.

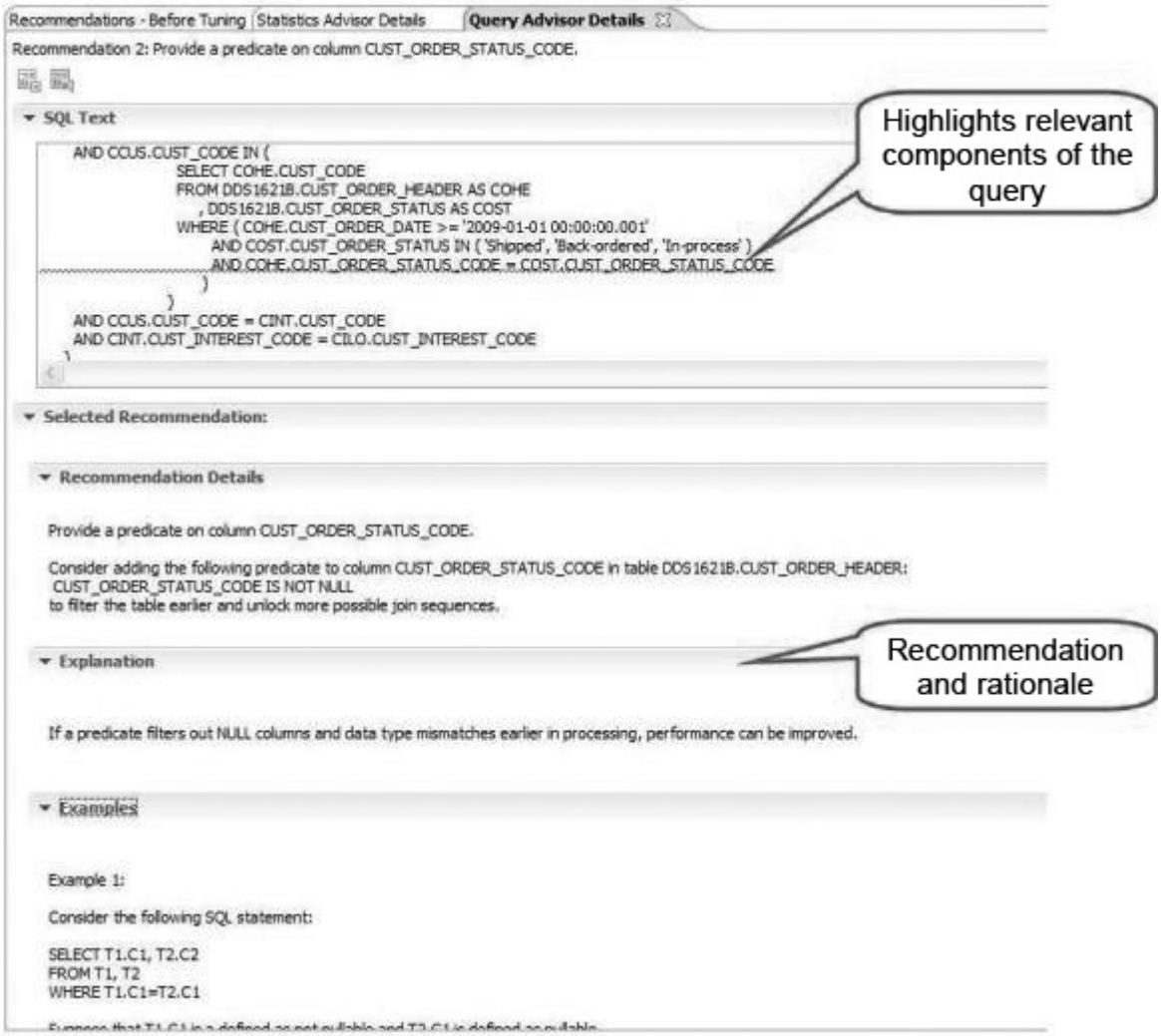
- Giảm thiểu số các trang chỉ mục và các hàng dữ liệu phải đọc. Ví dụ, bạn có thể giảm thiểu việc đọc các hàng bằng cách có các biến vị ngữ trong truy vấn có thể xác định các hàng cần thiết từ chỉ riêng chỉ mục ấy.
- Giảm thiểu các hoạt động sắp xếp. Ví dụ, mệnh đề ORDER BY hay GROUP BY có cần thiết trong truy vấn hay không, hoặc có thể giải quyết chúng thông qua truy cập chỉ mục không.

Cụ thể, trình tư vấn truy vấn kiểm tra như sau.

- Các biến vị ngữ nối còn thiếu, nhưng chỉ khi một khóa ngoài được định nghĩa.
- Các biến vị ngữ Giai đoạn 2 có thể cải thiện hiệu năng nếu viết lại như Giai đoạn 1 hoặc có thể tạo chỉ mục. Xem phần Tài nguyên để biết một liên kết đến Trung tâm Thông tin của DB2 cho z/OS nơi bạn có thể tìm thêm thông tin về biến vị ngữ Giai đoạn 1 và Giai đoạn 2.
- Các biến vị ngữ Giai đoạn 1 có thể cải thiện hiệu năng nếu viết lại là chỉ mục có khả năng.
- Các biến vị ngữ cục bộ bổ sung không được DB2 cung cấp tự động do DB2 có thể cung cấp closure bắc cầu của biến vị ngữ.
- Các biến vị ngữ được đẩy xuống đến một biểu thức bảng lồng nhau hoặc khung nhìn cụ thể hóa mà không thay đổi kết quả, và đã không thực hiện tự động bằng DB2.
- Các biến vị ngữ đã bổ sung một mệnh đề WHERE phức tạp, có chứa `OR`, `AND`, và các dấu ngoặc đơn `()`. Điều này có thể cải thiện hiệu năng mà không thay đổi kết quả.
- Việc sử dụng `SELECT *` có thể được thay thế bằng danh sách cột cụ thể.

Trong Hình 16, bạn có thể xem các chi tiết của một khuyến cáo của trình tư vấn truy vấn cùng với lý do cơ bản cho khuyến cáo đó. Giao diện người dùng (UI) tự động làm nổi bật phần truy vấn liên quan đến khuyến cáo đó. Theo cách này, các DBA hoặc những người phát triển tìm hiểu về điều chỉnh truy vấn khi họ xem khuyến cáo.

Hình 16. Các khuyến cáo của trình tư vấn truy vấn



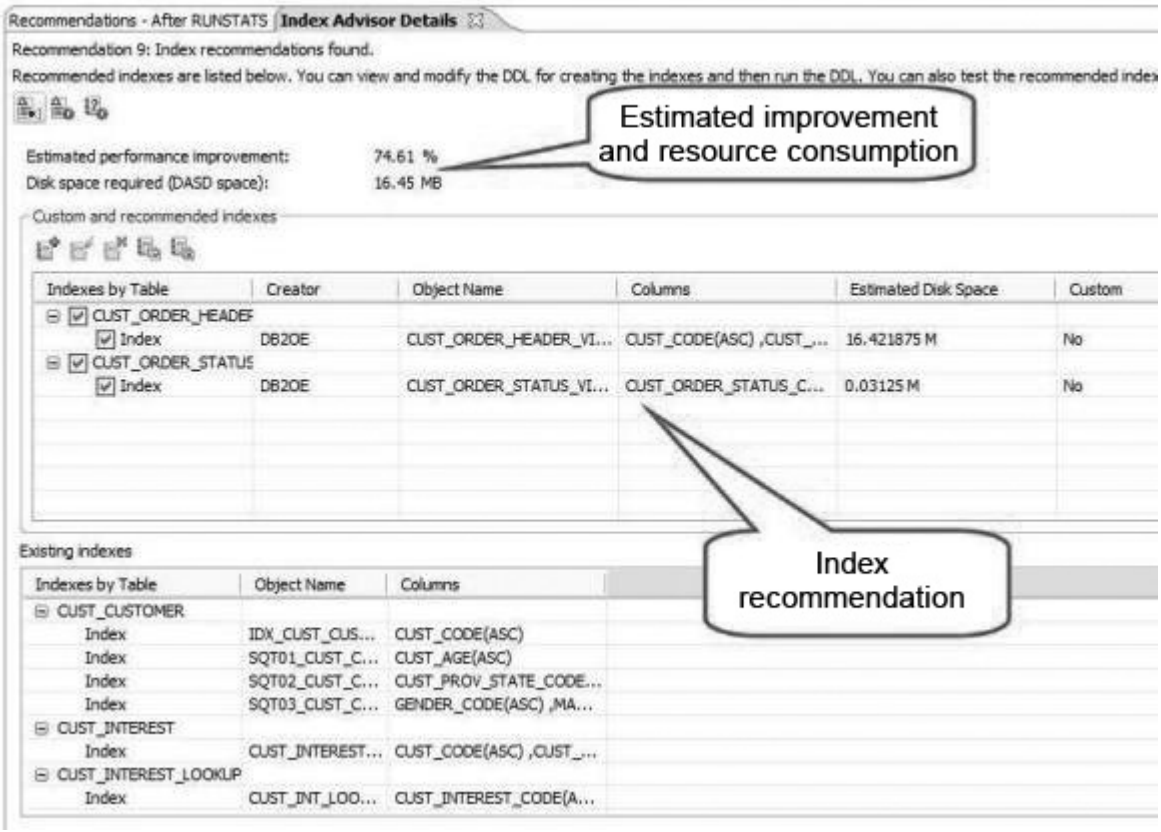
Thực hiện trình tư vấn chỉ mục để nâng cao hiệu quả truy vấn

Query Tuner cũng cung cấp lời khuyên về chỉ mục. Nó phân tích truy vấn và khuyến cáo các chỉ mục bổ sung thường mang lại lợi ích cho truy cập truy vấn. Trình tư vấn chỉ mục có thể khuyến cáo các chỉ mục vì những lý do sau.

- Các khóa ngoài vẫn chưa định nghĩa các chỉ mục.
- Các chỉ mục sẽ cung cấp lọc chỉ mục và/hoặc kiểm tra câu lệnh SQL.
- Các chỉ mục sẽ cung cấp truy cập chỉ theo chỉ mục cho câu lệnh SQL.
- Các chỉ mục có thể giúp tránh các sắp xếp.

Hình 17 cho thấy các khuyến cáo chỉ mục cho truy vấn mẫu trong bài này, cùng với sự cải thiện hiệu năng đã đánh giá và yêu cầu vùng DASD. Nó cũng tạo ra DDL cần thiết để tạo các chỉ mục, và cung cấp cho bạn khả năng để chạy chúng ngay lập tức, giả sử bạn có ủy quyền thích hợp, hoặc lưu chúng để sau này xem xét và thực hiện.

Hình 17. Các khuyến cáo của trình tư vấn chỉ mục



Bạn cũng có thể tùy chỉnh các khuyến cáo và tạo DDL. Ví dụ, bạn có thể quy định một số tối đa cho số các cột có thể là một phần của một khóa chỉ mục và thay đổi Creator ID (Mã định danh của trình tạo) được sử dụng khi tạo DDL chỉ mục. Bạn cũng có thể quy định các giá trị FREEPAGE, PCTFREE, và CLUSTERRATIO được thừa nhận cho các chỉ mục mới, và dù kích thước trang lá chỉ mục có thể vượt quá 4 KB hay không.

Bạn cũng có thể thử nghiệm các chỉ mục được khuyến cáo cùng với các chỉ mục mà bạn đề xuất với "phân tích what-if " trước khi triển khai. Nó cho phép bạn áp dụng các ràng buộc khác nhau cho trình tư vấn chỉ mục và xem cách các ràng buộc đó thay đổi các khuyến cáo. "Phân tích what-if " sử dụng một chỉ mục ảo trong các vỏ bọc để đánh giá hiệu quả của các chỉ mục. Nhờ đó, bạn có thể chọn tạo các chỉ mục có thể giúp cung cấp cho bạn hiệu năng mà bạn cần.

Kết luận

Bài này đã giới thiệu một phương pháp luận để điều chỉnh các truy vấn riêng rẽ, và đã sử dụng một truy vấn mẫu để giải thích từng bước của phương pháp luận này một cách chi tiết. Bài này bao gồm cách hiểu truy vấn vấn đề tốt hơn bằng cách sử dụng các phương tiện trợ giúp trực quan do định dạng truy vấn, ghi chú, các báo cáo, và biểu đồ kế hoạch truy cập cung cấp. Bạn đã học cách phân tích và điều chỉnh truy vấn vấn đề từ quan điểm của số liệu thống kê, các biến vị ngữ, và các chỉ mục. Bạn cũng đã được trải qua các cách để tăng tốc độ điều chỉnh truy vấn bằng cách sử dụng các trình tư vấn Query Tuner.

Bài tiếp theo của loạt bài này sẽ mở rộng thảo luận đến phạm vi của *tải truy vấn*, và giới thiệu lý do cần làm việc này, với một ví dụ để giới thiệu điều chỉnh tải công việc một cách chi tiết.

Hãy khởi đầu nhanh chóng với DB2 9 pureXML, Phần 3: Truy vấn dữ liệu XML của DB2 bằng SQL

Cynthia M. Saracco, Kiến trúc giải pháp cao cấp, IBM

Tóm tắt: Bản phát hành DB2 9 của IBM mô tả sự hỗ trợ mới quan trọng để lưu trữ, quản lý và truy vấn dữ liệu XML, được gọi là pureXML. Trong bài này, hãy tìm hiểu cách truy vấn dữ liệu được lưu trữ trong các cột XML bằng cách sử dụng SQL và SQL/XML. Các bài viết tiếp theo trong loạt bài này sẽ minh họa cách truy vấn dữ liệu XML bằng XQuery, một ngôn ngữ mới được DB2 hỗ trợ.

Lưu ý: Được viết lần đầu vào năm 2006, bài viết này luôn được cập nhật để theo kịp với các thay đổi sản phẩm trong phiên bản 9.5 và 9.7 của DB2 .

Mặc dù kiến trúc lai của DB2 biểu hiện một sự chuyển hướng quan trọng so với bản phát hành trước, việc tìm hiểu những khả năng XML mới của nó không phải là một quá trình tẻ nhạt. Nếu bạn đã quen với SQL, thì bạn có thể áp dụng ngay các kỹ năng của mình để làm việc với dữ liệu XML được lưu trữ nguyên gốc trong DB2. Hãy xem cách làm trong bài viết này.

Các tính năng XML trong DB2 9 bao gồm việc quản lý lưu trữ mới, lập chỉ mục và hỗ trợ ngôn ngữ truy vấn. Trong bài này, hãy tìm hiểu cách truy vấn dữ liệu trong các cột XML của DB2 bằng cách sử dụng SQL hoặc SQL với các phần mở rộng XML (SQL/XML). Các bài viết tiếp theo trong loạt bài này sẽ bàn về sự hỗ trợ mới của DB2 cho XQuery, một chuẩn công nghiệp mới nổi, và tìm hiểu khi nào nó có thể có ích nhất.

Bạn có thể ngạc nhiên khi biết DB2 cũng hỗ trợ các truy vấn song ngữ -- đó là, các truy vấn kết hợp các biểu thức trong cả hai SQL và XQuery. Ngôn ngữ (hoặc cách

kết hợp các ngôn ngữ) nào mà bạn nên sử dụng phụ thuộc vào các yêu cầu ứng dụng của bạn, cũng như kỹ năng của bạn. Việc kết hợp các phần tử của cả hai ngôn ngữ truy vấn vào một truy vấn không khó như bạn tưởng. Và làm như vậy có thể cung cấp cho bạn các khả năng mạnh mẽ để tìm kiếm và tích hợp dữ liệu SQL truyền thống và dữ liệu XML.

Cơ sở dữ liệu mẫu

Các truy vấn trong bài viết này sẽ truy cập vào các bảng mẫu được tạo ra trong bài "Hãy khởi đầu nhanh chóng với DB2 9 pureXML, Phần 2" (developerWorks, 03. 2006). Nhìn lướt qua, các bảng mẫu "items" (các mặt hàng) và "clients" (các khách hàng) được định nghĩa như sau:

Liệt kê 1. Các định nghĩa bảng

```
create table items (
id              int primary key not null,
brandname       varchar(30),
itemname        varchar(30),
sku             int,
srp             decimal(7,2),
comments        xml
```

)

```
create table clients(
id              int primary key not null,
name            varchar(50),
status          varchar(10),
contactinfo     xml
)
```

Dữ liệu XML mẫu có trong cột "items.comments" được chỉ ra trong Liệt kê 2, trong khi dữ liệu XML mẫu trong cột "clients.contactinfo" được hiển thị trong Liệt kê 3. Các ví dụ truy vấn tiếp theo sẽ tham chiếu các phần tử cụ thể trong một hoặc cả hai tài liệu XML này.

Liệt kê 2. Tài liệu XML mẫu được lưu trữ trong cột "comments" của bảng "items"

```
<Comments>
```

<Comment>
<CommentID>133</CommentID>
<ProductID>3926</ProductID>
<CustomerID>8877</CustomerID>
<Message>Heels on shoes wear out too quickly.</Message>
<ResponseRequested>No</ResponseRequested>
</Comment>
<Comment>
<CommentID>514</CommentID>
<ProductID>3926</ProductID>
<CustomerID>3227</CustomerID>
<Message>Where can I find a supplier in San Jose?</Message>
<ResponseRequested>Yes</ResponseRequested>
</Comment>
</Comments>

Liệt kê 3. Tài liệu XML mẫu được lưu trữ trong cột "contactinfo" của bảng "clients"

```
<Client>
        <Address>
                <street>5401 Julio Ave.</street>
                <city>San Jose</city>
                <state>CA</state>
                <zip>95116</zip>
        </Address>
        <phone>
                <work>4084630000</work>
                <home>4081111111</home>
                <cell>4082222222</cell>
        </phone>
        <fax>4087776666</fax>
        <email>love2shop@yahoo.com</email>
```

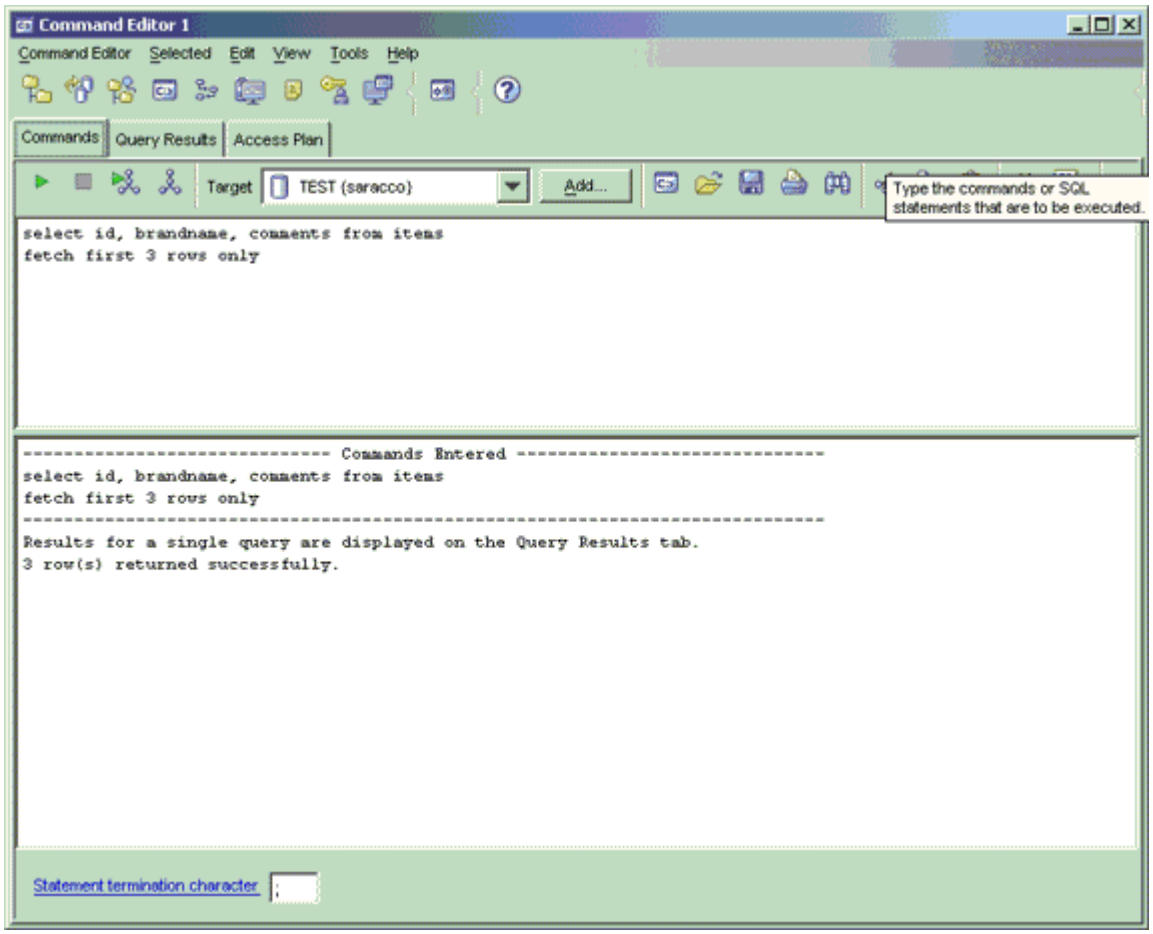
</Cleint>

Môi trường truy vấn

Thiết kế tất cả các truy vấn trong bài viết này được ban hành tương tác với nhau, bạn có thể thực hiện qua bộ xử lý dòng lệnh DB2 hoặc Trình soạn thảo lệnh của DB2 (DB2 Command Editor) của Trung tâm điều khiển DB2 (DB2 Control Center). Các ảnh màn hình và các tập lệnh trong bài viết này tập trung vào cái sau. (IBM Data Studio và IBM Optim Development Studio cũng đi kèm với một bàn làm việc của Nhà phát triển dựa trên Eclipse có thể giúp các lập trình viên xây dựng các truy vấn bằng đồ họa. Tuy nhiên, bài viết này không bàn về các vấn đề phát triển ứng dụng hoặc Development Studio).

Để sử dụng DB2 Command Editor, hãy khởi chạy Control Center và chọn **Tools** > **Command Editor**. Một cửa sổ tương tự như Hình 1 sẽ xuất hiện. Gõ các truy vấn của bạn vào ô bên trên, nhấn vào mũi tên màu xanh lá cây ở góc trên bên trái để chạy chúng và xem kết quả của bạn ở ô bên dưới hoặc trong thẻ "Query results" (Các kết quả truy vấn) riêng.

Hình 1. DB2 Command Editor, có thể được khởi chạy từ DB2 Control Center



Các truy vấn chỉ dùng SQL

Ngay cả khi hiểu biết của bạn về SQL bị hạn chế, bạn chỉ cần một chút nỗ lực để truy vấn dữ liệu XML. Ví dụ, truy vấn sau đây chọn tất cả các nội dung của bảng "clients", bao gồm thông tin XML được lưu trữ trong cột "contactinfo":

Liệt kê 4. Câu lệnh SELECT đơn giản

```
select * from clients
```

Tất nhiên, bạn có thể viết thêm nhiều truy vấn SQL có chọn lựa hơn kết hợp các phép chiếu và các phép hạn chế quan hệ. Truy vấn sau lấy ra các mã định danh ID, các tên và thông tin liên hệ cho tất cả khách hàng có trạng thái "Vàng" (Gold). Lưu ý rằng "contactinfo" chứa dữ liệu XML, trong khi hai cột khác không chứa:

Liệt kê 5. Câu lệnh SELECT đơn giản với phép chiếu và phép hạn chế

```
select id, name, contactinfo

from clients

where status = 'Gold'
```

Và, như bạn có thể mong đợi, bạn có thể tạo các khung nhìn dựa trên các truy vấn như vậy, như đã thấy ở đây với "khung nhìn vàng":

Liệt kê 6. Tạo một khung nhìn có chứa một cột XML

```
create view goldview as

select id, name, contactinfo

from clients

where status = 'Gold'
```

Thật không may, có rất nhiều thứ mà bạn không thể làm được chỉ với SQL. Các câu lệnh SQL thuần túy cho phép bạn lấy toàn bộ các tài liệu XML (như bạn đã thấy), nhưng bạn không thể xác định các biến vị ngữ truy vấn dựa trên XML và bạn không thể lấy một phần các tài liệu XML hoặc các giá trị phần tử cụ thể từ một tài liệu XML. Nói cách khác, bạn không thể chiếu, hạn chế, nối, tổng hợp hoặc xếp thứ tự các đoạn của các tài liệu XML khi sử dụng SQL thuần túy. Ví dụ, bạn không thể lấy chỉ các địa chỉ email của các khách hàng Vàng của bạn hoặc các tên của các khách hàng sống trong vùng có mã vùng bưu điện "95116". Để biểu diễn các kiểu truy vấn này, bạn cần sử dụng SQL với các phần mở rộng XML (SQL/XML), XQuery hoặc kết hợp cả hai.

Phần tiếp theo tìm hiểu một số tính năng cơ bản của SQL/XML. Và trong một bài viết tiếp theo, hãy tìm hiểu cách viết XQuery cũng như cách kết hợp XQuery với SQL.

Các truy vấn SQL/XML

Như tên ngụ ý, SQL/XML được thiết kế làm cầu nối giữa thế giới SQL và XML. Nó đã phát triển như là một phần của nỗ lực tiêu chuẩn SQL và bây giờ bao gồm các đặc tả để nhúng các biểu thức XPath hoặc XQuery trong các câu lệnh SQL. XPath là một ngôn ngữ dùng để chuyển hướng các tài liệu XML để tìm các phần tử hay các thuộc tính. XQuery bao gồm sự hỗ trợ cho XPath.

Điều quan trọng cần lưu ý là các biểu thức XQuery (và XPath) phân biệt chữ hoa và chữ thường. Ví dụ, XQuery tham chiếu phần tử XML "zip" sẽ không áp dụng cho các phần tử XML có tên là "ZIP" hoặc "Zip". Đôi khi trường hợp phân biệt chữ hoa và chữ thường gây khó khăn cho các lập trình viên SQL phải nhớ, trong khi cú pháp truy vấn SQL cho phép họ sử dụng "zip", "ZIP" và "Zip" để nói đến cùng một tên cột.

DB2 9 mô tả nhiều hơn 15 hàm SQL/XML, cho phép bạn tìm kiếm dữ liệu cụ thể trong các tài liệu XML, chuyển đổi dữ liệu quan hệ thành dữ liệu XML, chuyển đổi dữ liệu XML thành dữ liệu quan hệ và thực hiện các nhiệm vụ có ích khác. Bài viết này không trình bày toàn bộ sự phong phú của SQL/XML. Tuy nhiên, nó xem xét một số thách thức truy vấn thông thường và làm thế nào để các hàm SQL/XML chủ yếu có thể giải quyết những thách thức này.

Các kết quả "hạn chế" dựa trên các giá trị phần tử XML

Các lập trình viên SQL thường viết các truy vấn để hạn chế các hàng được trả về từ DBMS dựa trên một số điều kiện. Ví dụ, truy vấn SQL trong Liệt kê 5 hạn chế các hàng được lấy từ bảng "clients" để chỉ bao gồm những khách hàng nào có trạng thái "Vàng". Trong trường hợp này, trạng thái của khách hàng được bắt giữ trong một cột SQL VARCHAR. Nhưng điều gì sẽ xảy ra nếu bạn muốn hạn chế việc tìm kiếm của mình dựa trên một số điều kiện áp dụng cho dữ liệu trong một

cột XML? Hàm XMLExists của SQL/XML cung cấp một phương tiện để làm điều này.

XMLExists cho phép bạn chuyển hướng đến một phần tử trong tài liệu XML của bạn và thử nghiệm với một điều kiện cụ thể. Khi được quy định như một phần của mệnh đề WHERE, XMLExists hạn chế các kết quả trả về với chỉ các hàng nào có chứa một tài liệu XML có giá trị phần tử XML cụ thể (nói cách khác, ở đó giá trị cụ thể đánh giá là "đúng").

Hãy xem xét một vấn đề truy vấn mẫu đã nêu ở trên. Hãy tưởng tượng rằng bạn cần xác định vị trí các tên của tất cả các khách hàng sống trong vùng có một mã vùng bưu điện cụ thể. Nhớ lại, bảng "clients" lưu trữ các địa chỉ của các khách hàng (bao gồm các mã vùng bưu điện) trong một cột XML. (Xem Liệt kê 3.) Khi sử dụng XMLExists, bạn có thể tìm kiếm cột XML theo mã vùng bưu điện đích và hạn chế tập kết quả trả về cho phù hợp. Truy vấn SQL/XML sau đây trả về các tên của các khách hàng sống trong vùng có mã vùng bưu điện 95116:

Liệt kê 7. Hạn chế các kết quả dựa trên một giá trị phần tử XML

```
select name from clients

where xmlexists('$c/Client/Address[zip="95116"]'

passing clients.contactinfo as "c")
```

Dòng đầu tiên là một mệnh đề SQL quy định rằng bạn chỉ cần lấy thông tin trong cột "name" của bảng "clients". Mệnh đề WHERE gọi hàm XMLExists, quy định một biểu thức XPath nhắc DB2 chuyển hướng đến phần tử "zip" và kiểm tra một giá trị là 95116. Mệnh đề "$c/Client/Address" chỉ ra đường dẫn trong hệ thống phân cấp tài liệu XML ở nơi DB2 có thể xác định vị trí phần tử "zip". Khi sử dụng dữ liệu có khả năng truy cập từ nút "$c" (mà chúng tôi sẽ giải thích ngay), DB2 sẽ chuyển hướng thông qua phần tử "Client" (Khách hàng) đến phần tử con "Address" (Địa chỉ) của nó để kiểm tra mã vùng bưu điện (giá trị "zip"). Dòng cuối cùng giải quyết giá trị của "$c": đó là cột "contactinfo" của bảng "clients". Vì vậy, DB2 kiểm tra dữ liệu XML được chứa trong cột "contactinfo", chuyển hướng từ phần tử gốc "Client" đến phần tử con "Address" rồi tới "zip" và xác định xem khách hàng có sống trong vùng có mã vùng bưu điện đích không. Nếu có, hàm XMLExists đánh giá là "true" và DB2 trả về tên của khách hàng liên quan đến hàng đó.

Một lỗi phổ biến liên quan đến việc tạo biến vị ngữ truy vấn XMLExists, như trong Liệt kê 8.

Liệt kê 8. Cú pháp không đúng với việc hạn chế các kết quả dựa vào một giá trị phần tử XML

```
select name from clients

where xmlexists('$c/Client/Address/zip="95116" '

passing clients.contactinfo as "c")
```

Trong khi truy vấn này sẽ thực hiện thành công, nó sẽ không hạn chế các kết quả với các khách hàng đang sống trong vùng có mã bưu điện là 95116. (Điều này là do ngữ nghĩa được quy định trong tiêu chuẩn; nó không phải là duy nhất với DB2). Để hạn chế các kết quả với các khách hàng sống trong vùng có mã bưu điện là 95116, bạn cần sử dụng cú pháp được hiện thị trong Liệt kê 7.

Bạn có thể muốn biết cách đặt một truy vấn hạn chế dữ liệu XML trong một ứng dụng. Trong khi bài viết này không bàn về các chủ đề phát triển ứng dụng cụ thể, nó chỉ đưa ra một ví dụ Java đơn giản sử dụng một dấu tham số trong một câu lệnh SQL/XML để hạn chế kết quả theo thông tin về các khách hàng sống trong vùng có một mã vùng bưu điện cụ thể.

"Chiếu" các giá trị phần tử XML

Bây giờ hãy xem xét một tình huống hơi khác một chút, trong đó bạn muốn chiếu các giá trị XML vào tập kết quả trả về của bạn. Nói cách khác, chúng ta muốn lấy một hay nhiều giá trị phần tử từ các tài liệu XML của chúng ta. Có nhiều cách để làm điều này. Trước tiên hãy sử dụng hàm XMLQuery để lấy một giá trị cho một phần tử rồi sử dụng hàm XMLTable để lấy các giá trị cho nhiều phần tử và ánh xạ chúng vào các cột của một tập kết quả SQL.

Hãy xem xét cách giải quyết một vấn đề đã nêu ở trên: cách tạo ra một bản ghi liệt kê các địa chỉ email của khách hàng Vàng. Truy vấn sau đây trong Liệt kê 9 gọi hàm XMLQuery để hoàn thành nhiệm vụ này:

Liệt kê 9. Lấy thông tin email cho các khách hàng có đủ điều kiện

```
select xmlquery('$c/Client/email'

passing contactinfo as "c")

from clients

where status = 'Gold'
```

Dòng đầu tiên quy định rằng bạn muốn trả về các giá trị cho phần tử con "email" của phần tử gốc "Client". Các dòng thứ hai và thứ ba cho thấy nơi DB2 có thể tìm thấy thông tin này -- trong cột "contactinfo" của bảng "clients". Dòng thứ tư bổ sung thêm cho truy vấn của bạn để cho biết rằng bạn chỉ quan tâm đến các địa chỉ email của các khách hàng Vàng. Truy vấn này sẽ trả về một tập các phần tử và các giá trị XML. Ví dụ, nếu bạn đã có 500 khách hàng Vàng, mỗi khách hàng có một địa chỉ email, kết quả đầu ra của bạn sẽ là một tập kết quả một cột có 500 hàng, như trong Liệt kê 10:

Liệt kê 10. Kết quả đầu ra mẫu của truy vấn trước đó

```
1

-----------------------------------------
```

<email>user5976@anyprovider.com</email>

. . .

<email>someID@yahoo.com</email>

Nếu bạn có nhiều địa chỉ email của các khách hàng Vàng riêng biệt, bạn có thể muốn chỉ thị cho DB2 chỉ trả về địa chỉ chính (có nghĩa là, địa chỉ email đầu tiên được tìm thấy trong tài liệu "contactinfo" của khách hàng). Bạn có thể sửa đổi biểu thức XPath ở dòng đầu tiên của truy vấn của bạn để làm như vậy:

Liệt kê 11. Lấy địa chỉ email đầu tiên của từng khách hàng có đủ điều kiện

```
select xmlquery('$c/Client/email[1]'
passing contactinfo as "c")
from clients
where status = 'Gold'
```

Cuối cùng, nếu bạn thiếu địa chỉ email của một số khách hàng Vàng, bạn có thể phải viết một truy vấn để loại trừ các kết quả bằng không (null) khỏi tập kết quả đó. Để thực hiện điều này, hãy sửa đổi truy vấn trước đó bằng cách thêm một biến vị ngữ khác vào mệnh đề WHERE để kiểm tra chỗ còn thiếu thông tin email. Bạn đã quen với hàm SQL/XML cho phép bạn làm điều đó -- đó là hàm XMLExists. Liệt kê 12 cho thấy cách bạn có thể viết lại truy vấn trước đó để lọc ra bất kỳ hàng nào có các khách hàng Vàng thiếu địa chỉ email trong thông tin liên hệ của họ (được lưu trữ như XML):

Liệt kê 12. Lấy địa chỉ email đầu tiên của mỗi khách hàng có đủ điều kiện mà ít nhất chúng ta có một địa chỉ email của họ

```
select xmlquery('$c/Client/email[1]'

passing contactinfo as "c")

from clients

where status = 'Gold'

and xmlexists('$c/Client/email' passing contactinfo as "c")
```

Bây giờ hãy xem xét một tình huống hơi khác một chút, trong đó bạn cần lấy nhiều giá trị phần tử XML. Hàm XMLTable tạo ra kết quả dạng bảng từ dữ liệu được lưu trữ trong các cột XML và khá có ích để cung cấp cho các lập trình viên một khung nhìn "quan hệ" của dữ liệu XML. Giống như XMLExists và

XMLQuery, hàm XMLTable làm cho DB2 chuyển hướng qua hệ thống phân cấp tài liệu XML để định vị dữ liệu quan tâm. Tuy nhiên, hàm XMLTable cũng bao gồm các mệnh đề để ánh xạ dữ liệu XML đích vào các cột của tập kết quả của các kiểu dữ liệu SQL.

Hãy xem xét truy vấn sau đây (Liệt kê 13), nó chiếu các cột của cả dữ liệu quan hệ lẫn dữ liệu XML được lưu trữ trong bảng "items". (Xem Liệt kê 2 để xem xét lại bảng "items"). Các mã định danh ý kiến, các mã định danh khách hàng và các thông báo được lưu trữ trong các tài liệu XML trong cột "comments". Các tên mặt hàng được lưu trữ trong cột VARCHAR của SQL.

Liệt kê 13. Lấy nhiều phần tử XML và chuyển đổi mỗi phần tử đó thành một kiểu dữ liệu SQL truyền thống

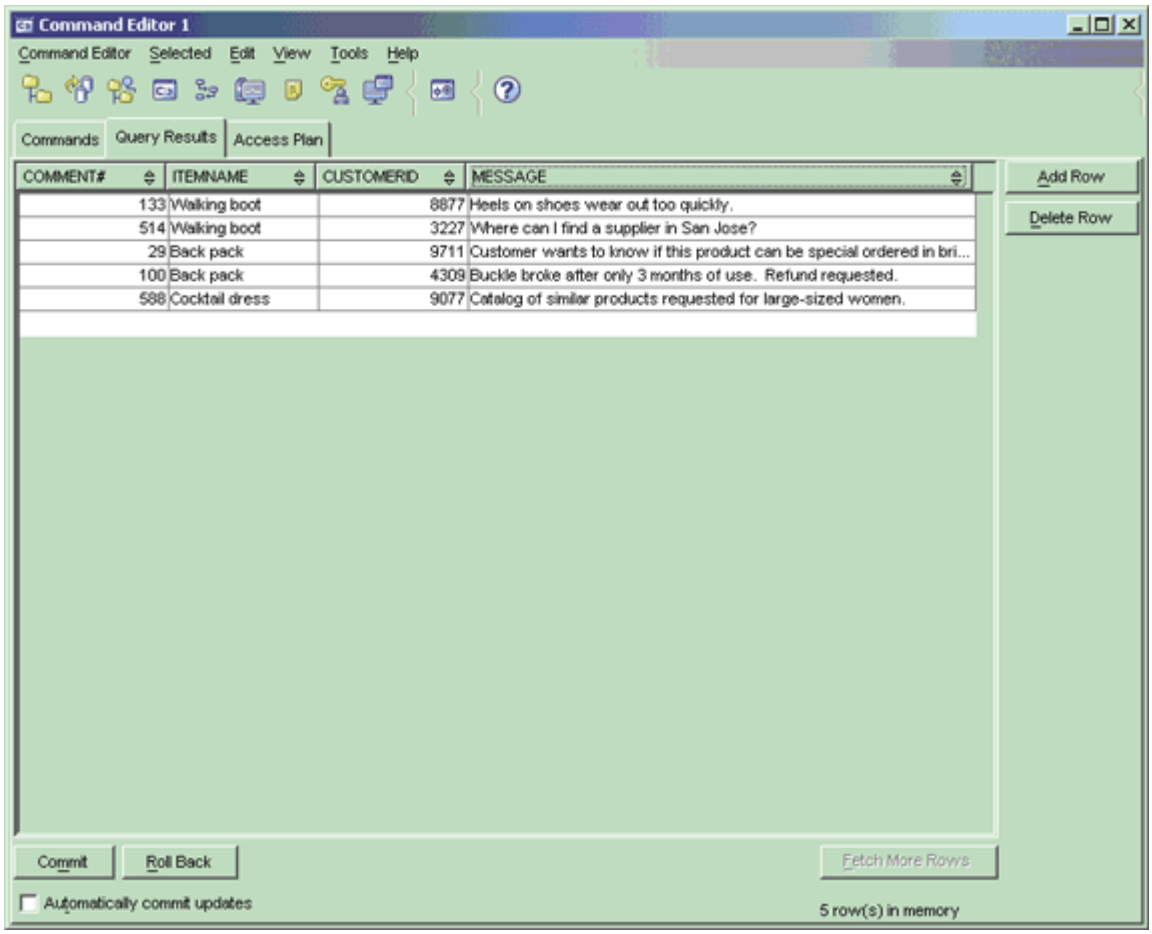
```
select t.Comment#, i.itemname, t.CustomerID, Message from items i,

xmltable('$c/Comments/Comment' passing i.comments as "c"

columns Comment# integer path 'CommentID',

        CustomerID integer path 'CustomerID',

        Message varchar(100) path 'Message') as t
```

Dòng đầu tiên quy định các cột có trong tập kết quả của bạn. Các cột được bao quanh bằng các dấu ngoặc kép và được thêm tiền tố với biến "t" dựa vào các giá trị

phần tử XML, như các dòng truy vấn tiếp theo cho biết. Dòng thứ hai gọi hàm XMLTable để xác định cột XML của DB2 chứa dữ liệu đích ("i.comments") và đường dẫn trong các tài liệu XML của cột nơi mà các phần tử cần quan tâm được định vị (trong phần tử con "Comment" của phần tử gốc "Comments"). Mệnh đề "columns", bao trùm các dòng từ 3 đến 5, nhận biết các phần tử XML cụ thể sẽ được ánh xạ vào các cột kết quả đầu ra trong tập kết quả SQL, đã quy định trên dòng 1. Phần của ánh xạ này có quy định các kiểu dữ liệu mà các giá trị của phần tử XML sẽ được chuyển đổi theo các kiểu dữ liệu đó. Trong ví dụ này, tất cả dữ liệu XML được chuyển đổi theo kiểu dữ liệu SQL truyền thống.

Hình 2 cho thấy các kết quả mẫu khi chạy truy vấn này. Như bạn thấy, đầu ra là một tập kết quả SQL đơn giản. Lưu ý rằng các tên cột đã được chuyển thành chữ hoa -- một sự kiện thông thường với SQL.

Hình 2. Kết quả đầu ra mẫu từ truy vấn khi sử dụng hàm XMLTABLE



Nếu muốn, bạn có thể sử dụng hàm XMLTable để tạo ra các tập kết quả cũng có các cột XML. Ví dụ, câu lệnh sau đây tạo ra một tập kết quả tương tự tập kết quả trước đó, trừ dữ liệu "Message" được chứa trong một cột XML chứ không phải cột VARCHAR của SQL.

Liệt kê 14. Lấy nhiều phần tử XML và chuyển đổi chúng sang các kiểu dữ liệu SQL truyền thống hoặc dữ liệu XML

```
select t.Comment#, i.itemname, t.CustomerID, Message from items i,

xmltable('$c/Comments/Comment' passing i.comments as "c"

columns Comment# integer path 'CommentID',

        CustomerID integer path 'CustomerID',

        Message XML by ref path 'Message') as t
```

Tạo các khung nhìn quan hệ của dữ liệu XML

Như bạn có thể tưởng tượng, các hàm SQL/XML có thể được sử dụng để định nghĩa các khung nhìn. Điều này đặc biệt có ích nếu bạn muốn đưa cho các lập trình viên ứng dụng SQL của bạn một mô hình quan hệ về dữ liệu XML nguyên gốc của bạn.

Việc tạo ra một khung nhìn quan hệ trên dữ liệu trong một cột XML không phức tạp hơn nhiều so với việc chiếu các giá trị phần tử XML. Bạn chỉ cần viết một câu lệnh SQL/XML SELECT gọi hàm XMLTable và sử dụng hàm này làm cơ sở cho định nghĩa khung nhìn của bạn. Ví dụ sau đây trong Liệt kê 15 tạo ra một khung nhìn dựa vào thông tin trong các cột XML và không- XML của bảng "items". (Nó tương tự như truy vấn trong Liệt kê 13.)

Liệt kê 15. Tạo ra một khung nhìn, dựa vào kết quả đầu ra của XMLTABLE

```
create view commentview(itemID, itemname, commentID, message, mustrespond)
as

select i.id, i.itemname, t.CommentID, t.Message, t.ResponseRequested from items
i,

xmltable('$c/Comments/Comment' passing i.comments as "c"

columns CommentID integer path 'CommentID',

        Message varchar(100) path 'Message',

        ResponseRequested varchar(100) path 'ResponseRequested') as t;
```

Mặc dù rất dễ tạo ra các khung nhìn quan hệ trên dữ liệu cột XML, bạn nên xem xét cẩn thận cách sử dụng chúng nếu bạn không dùng phiên bản V9.7. Trước phiên bản V9.7, DB2 đã không sử dụng các chỉ mục cột XML khi đã đưa ra các truy vấn dựa vào các khung nhìn như vậy. Vì thế, nếu bạn đã lập chỉ mục cho phần tử ResponseRequested và đã đưa ra một truy vấn SQL hạn chế các kết quả của cột "mustrespond" theo một giá trị nhất định, thì DB2 sẽ đọc tất cả các tài liệu XML và tìm kiếm giá trị "ResponseRequested" thích hợp. Điều này sẽ làm chậm hiệu năng thời gian chạy, trừ khi bạn có ít dữ liệu. Vì vậy, hãy cẩn thận ở đây cho đến khi bạn nâng cấp lên phiên bản V9.7, khi DB2 sẽ sử dụng các chỉ mục XML dựa vào các biến vị ngữ SQL.

Nối dữ liệu XML và dữ liệu quan hệ

Đến bây giờ, bạn có thể tự hỏi về việc nối dữ liệu XML với dữ liệu không-XML (ví dụ, dữ liệu quan hệ dựa trên các kiểu SQL truyền thống). DB2 cho phép bạn làm điều này bằng một câu lệnh SQL/XML đơn. Trong khi có những cách khác để tạo nên các kết nối như vậy, tùy thuộc vào lược đồ cơ sở dữ liệu của bạn và các yêu cầu khối lượng công việc, chúng ta sẽ trình bày một ví dụ ở đây. Và bạn có thể ngạc nhiên khi biết rằng bạn đã biết đủ về SQL/XML để nhận làm việc này.

Hãy nhớ lại rằng cột XML trong bảng "items" có chứa một phần tử "CustomerID". Cột này có thể dùng làm một khóa nối cho cột "id" dựa trên-số nguyên trong bảng "clients". Vì vậy, nếu bạn muốn có một bản ghi các tên và trạng thái các khách hàng đã góp ý kiến về một hoặc nhiều sản phẩm của bạn, bạn sẽ phải nối các giá trị phần tử XML từ một bảng có các giá trị số nguyên SQL tới bảng khác. Và một cách để thực hiện việc này là sử dụng hàm XMLExists, như trong Liệt kê 16:

Liệt kê 16. Nối dữ liệu XML và dữ liệu không-XML

```
select clients.name, clients.status from items, clients

where xmlexists('$c/Comments/Comment[CustomerID=$p]'

passing items.comments as "c", clients.id as "p")
```

Dòng đầu tiên xác định các cột SQL có trong tập kết quả truy vấn và các bảng nguồn được tham chiếu trong truy vấn. Dòng thứ hai có mệnh đề kết nối của bạn. Ở đây, hàm XMLExists xác định xem giá trị "CustomerID" trong một nguồn đích

có bằng một giá trị lấy ra từ một nguồn đích khác không. Dòng thứ ba quy định các nguồn này: nguồn đầu tiên là cột XML "comments" trong bảng "items" và nguồn thứ hai là cột "id" số nguyên trong bảng "clients". Như vậy, nếu các khách hàng đã góp ý kiến về mặt hàng bất kỳ và thông tin về khách hàng này có sẵn trong bảng "clients", thì biểu thức XMLExists sẽ đánh giá là "true" (đúng) và tên và thông tin trạng thái của khách hàng sẽ có trong bản ghi đó.

Sử dụng biểu thức "FLWOR" trong SQL/XML

Mặc dù chúng ta đã chỉ thảo luận một vài hàm, SQL/XML cung cấp nhiều khả năng mạnh mẽ để truy vấn dữ liệu XML và tích hợp dữ liệu đó với dữ liệu quan hệ. Thật vậy, bạn đã thấy một số ví dụ về cách làm điều đó, nhưng chúng ta sẽ bàn một thêm một chút nữa ở đây.

Cả hai hàm XMLExists và XMLQuery đều cho phép bạn kết hợp XQuery vào SQL. Ví dụ trước của chúng ta cho biết cách sử dụng các hàm này với các biểu thức XPath đơn giản để chuyển hướng đến một phần của một tài liệu XML cần quan tâm. Bây giờ hãy xem xét một ví dụ đơn giản trong đó bạn bao gồm XQuery trong các truy vấn SQL của mình.

XQueries có thể chứa một số hoặc tất cả các mệnh đề sau: "for," "let," "where," "order by" và "return." Nhìn chung, chúng tạo thành các biểu thức FLWOR (được đọc như từ flower). Các lập trình viên SQL có thể nhận thấy thật là thuận tiện để kết hợp các XQuery vào các danh sách SELECT của họ để trích xuất (hoặc chiếu) các đoạn của các tài liệu XML vào các tập kết quả của họ. Và trong khi đây không phải là cách duy nhất có thể sử dụng hàm XMLQuery, thì nó là một kịch bản mà bài viết này trình bày. (Một bài viết sau trong loạt bài này sẽ bàn sâu hơn về XQuery).

Hãy tưởng tượng rằng bạn muốn lấy các tên và các địa chỉ email chính của các khách hàng "Vàng" của bạn. Theo một số khía cạnh, nhiệm vụ này tương tự như

một nhiệm vụ mà chúng ta đã thực hiện ở trên (xem Liệt kê 11), khi chúng ta tìm hiểu cách chiếu các giá trị phần tử XML. Ở đây, bạn chuyển qua XQuery (bằng các mệnh đề "for" và "return") làm đầu vào cho hàm XMLQuery :

Liệt kê 17. Lấy dữ liệu XML bằng cách sử dụng các mệnh đề "for" và "return" của XQuery

```
select name, xmlquery('for $e in $c/Client/email[1] return $e'

passing contactinfo as "c")

from clients

where status = 'Gold'
```

Dòng đầu tiên quy định rằng các tên khách hàng và kết quả đầu ra từ hàm XMLQuery sẽ được chứa trong tập kết quả. Dòng thứ hai cho biết rằng phần tử con "email" đầu tiên của phần tử "Client" là được trả về. Dòng thứ ba nhận biết nguồn dữ liệu XML của chúng ta -- cột "contactinfo". Dòng 4 cho chúng ta biết cột này nằm trong bảng "clients". Cuối cùng, dòng thứ năm cho biết rằng chỉ có các khách hàng "Vàng" cần được chúng ta quan tâm.

Vì ví dụ này rất đơn giản, bạn có thể viết truy vấn tương tự ở đây. Để thay thế, bạn có thể viết truy vấn tương tự theo cách gọn hơn nhiều so với cách bạn đã làm ở trên:

Liệt kê 18. Viết lại truy vấn ở trên theo một cách gọn hơn

```
select name, xmlquery('$c/Client/email[1]'

passing contactinfo as "c")

from clients

where status = 'Gold'
```

Tuy nhiên, mệnh đề return của XQuery cho phép bạn chuyển đổi kết quả đầu ra XML khi cần. Ví dụ, bạn có thể trích xuất các giá trị phần tử email và xuất bản các giá trị này dưới dạng HTML. Truy vấn sau đây sẽ tạo ra một tập kết quả trong đó địa chỉ email đầu tiên của mỗi khách hàng Vàng được trả về như là một đoạn HTML.

Liệt kê 19. Lấy và chuyển đổi dữ liệu XML thành HTML

```
select xmlquery('for $e in $c/Client/email[1]/text()

return <p>{$e}</p>'
```

```
passing contactinfo as "c")

from clients

where status = 'Gold'
```

Dòng đầu tiên cho biết bạn đã quan tâm đến việc biểu diễn văn bản của địa chỉ email đầu tiên của các khách hàng có đủ điều kiện. Dòng thứ hai quy định rằng thông tin này được bao quanh bằng các thẻ của đoạn HTML trước khi trả về. Cụ thể là, các dấu móc nhọn ({ }) chỉ thị cho DB2 đánh giá biểu thức kèm theo (trong trường hợp, "$e") thay vì xử lý nó như là một chuỗi bằng chữ. Nếu bạn bỏ qua các dấu móc nhọn đó, DB2 sẽ trả về một tập kết quả có chứa " $e " cho mọi bản ghi khách hàng có đủ điều kiện.

Xuất bản dữ liệu quan hệ như dữ liệu XML

Cho đến nay, chúng ta đã tập trung vào các cách để truy vấn, trích xuất hoặc chuyển đổi dữ liệu chứa trong một cột XML của DB2. Và, như bạn đã thấy, các khả năng này đều có sẵn thông qua SQL/XML.

SQL/XML cũng cung cấp các tính năng tiện dụng khác. Trong số đó là khả năng chuyển đổi hoặc xuất bản dữ liệu quan hệ như là dữ liệu XML. Bài viết này chỉ trình bày ba hàm SQL/XML liên quan sau: XMLElement, XMLAgg và XMLForest.

Hàm XMLElement cho phép bạn chuyển đổi dữ liệu được lưu giữ trong các cột SQL truyền thống thành các đoạn XML. Do đó, bạn có thể xây dựng các phần tử XML (có hoặc không có các thuộc tính XML) từ cơ sở dữ liệu SQL của mình. Ví dụ sau đây lồng thêm cách sử dụng hàm XMLElement của nó để tạo ra một loạt

các phần tử mặt hàng, mỗi phần tử lại chứa các phần tử con cho các giá trị mã định danh ID, tên thương hiệu và mã hàng trong kho ("sku") thu được từ bảng "items":

Liệt kê 20. Sử dụng hàm XMLElement để xuất bản dữ liệu quan hệ như dữ liệu XML

```
select xmlelement (name "item",
        xmlelement (name "id", id),
        xmlelement (name "brand", brandname),
        xmlelement (name "sku", sku) ) from items
where srp < 100
```

Chạy truy vấn này sẽ tạo ra một kết quả tương tự như:

Liệt kê 21. Kết quả đầu ra mẫu từ truy vấn trước đó

```
<item>
```

```
  <id>4272</id>
  <brand>Classy</brand>
  <sku>981140</sku>
</item>
. . .
<item>
  <id>1193</id>
  <brand>Natural</brand>
  <sku>557813</sku>
</item>
```

Bạn có thể kết hợp hàm XMLElement với các hàm xuất bản SQL/XML để xây dựng và nhóm các giá trị XML với nhau, lồng chúng vào các hệ thống phân cấp như mong muốn. Ví dụ trong Liệt kê 22 sử dụng hàm XMLElement để tạo các phần tử customerList mà nội dung của chúng được nhóm lại theo các giá trị trong cột "status" (trạng thái). Đối với mỗi bản ghi "customerList", hàm XMLAgg trả về một chuỗi các phần tử khách hàng, mỗi chuỗi gồm các phần tử con dựa vào cột "name" và "status" của chúng ta. Hơn nữa, bạn sẽ thấy rằng các giá trị phần tử khách hàng được sắp xếp thứ tự theo tên của khách hàng.

Liệt kê 22. Tổng hợp và nhóm dữ liệu

```
select xmlelement(name "customerList",

xmlagg (xmlelement (name "customer",

xmlforest (name as "fullName", status as "status") )

order by name ) )

from clients

group by status
```

Hãy giả sử bảng "clients" của chúng ta chứa ba giá trị "trạng thái" riêng: "Gold" (Vàng), "Silver" (Bạc) và "Standard" (Chuẩn). Việc chạy truy vấn trước đó sẽ làm cho DB2 trả về ba phần tử customerList, mỗi phần tử có thể chứa nhiều phần tử con khách hàng để chứa thêm tên và thông tin trạng thái. Vì vậy, kết quả đầu ra sẽ xuất hiện tương tự như:

Liệt kê 23. Kết quả đầu ra mẫu từ truy vấn trước đó

```
<customerList>
  <customer>
    <fullName>Chris Bontempo</fullname>
    <status>Gold</status>
  </customer>
  <customer>
    <fullName>Ella Kimpton</fullName>
    <status>Gold</status>
  </customer>
. . .
</customerList>
<customerList>
  <customer>
    <fullName>Lisa Hansen</fullName>
    <status>Silver</status>
  </customer>
. . .
```

```
</customerList>

<customerList>

  <customer>

    <fullName>Rita Gomez</fullName>

    <status>Standard</status>

  </customer>

. . .

</customerList>
```

Các hoạt động cập nhật và xóa

Mặc dù trọng tâm của bài viết này là về tìm kiếm và lấy ra dữ liệu được lưu trữ trong các cột XML bằng cách sử dụng SQL, nhưng cũng đáng bỏ chút thời gian xem xét hai nhiệm vụ phổ biến khác: đó là cập nhật và xoá dữ liệu trong các cột XML.

DB2 9 cho phép những người dùng cập nhật và xóa dữ liệu XML bằng các câu lệnh SQL và SQL/XML. Thật vậy, vì dự thảo ban đầu của tiêu chuẩn XQuery không đề cập đến những vấn đề này, nên những người dùng của DB2 đã phải dựa vào SQL để thực hiện các nhiệm vụ này. Tuy nhiên, W3C đã đang tiếp tục đưa

vào một Phương tiện cập nhật XQuery (XQuery Update Facility), được triển khai thực hiện trong DB2 phiên bản 9.5. Việc bổ sung XQuery Update Facility (ban đầu gọi là TRANSFORM-PHÉP BIẾN ĐỔI) đã làm đơn giản hóa đáng kể việc cập nhật các thuộc tính và các phần tử trong một tài liệu XML, cũng như đã thiết lập một tiêu chuẩn để làm điều đó. Bây giờ XQuery Update Facility hiện đang trong Trạng thái giới thiệu ứng cử viên (Candidate Recommendation Status).

Cập nhật dữ liệu XML

Trong khi DB2 9 cho phép bạn cập nhật một cột XML bằng một câu lệnh UPDATE của SQL hoặc thông qua việc sử dụng một thủ tục đã lưu do hệ thống cung cấp (DB2XMLFUNCTIONS.XMLUPDATE), với DB2 phiên bản 9.5, có thể sử dụng XQuery Update Facility mới. Điều này cho phép cập nhật, chèn, xóa và tạo một phần tử hoặc thuộc tính mới trong một tài liệu XML hiện có mà không cần tạo lại toàn bộ tài liệu. Phương tiện Cập nhật cũng có thể được sử dụng để sửa đổi nhiều nút trong giao dịch tương tự.

Ví dụ, nếu bạn muốn ban hành một câu lệnh UPDATE để thay đổi địa chỉ e-mail về thông tin liên hệ của một khách hàng cụ thể, thì bạn đơn giản phải cung cấp địa chỉ e-mail mới.

Hãy xem xét câu lệnh sau đây:

Liệt kê 24. Câu lệnh UPDATE mẫu

update clients

```
set contactinfo = xmlquery( '

        copy $new := $CONTACTINFO

        modify do replace value of $new/client/email with
"newemail@someplace.com"

        return  $new' )

where id = 3227;
```

Các mệnh đề của phương tiện XQuery Update yêu cầu "copy $new", "modify do replace of $new" và "return $new". Bạn có thể tìm hiểu thêm về cú pháp chính xác và các tùy chọn trong phần Tài nguyên dưới đây. Chúng tôi đã đưa cả trang web với các đặc tả XQuery cũng như một bài viết của developerWorks cung cấp thêm chi tiết về XQuery Update Facility (Phương tiện cập nhật XQuery).

Xóa dữ liệu XML

Xóa các hàng chứa các cột XML là một quá trình đơn giản. Câu lệnh DELETE của SQL cho phép bạn nhận biết (hoặc hạn chế) các hàng bạn muốn xóa thông qua một mệnh đề WHERE. Mệnh đề này có thể có các biến vị ngữ đơn giản để nhận biết các giá trị các cột không-XML hoặc các hàm SQL/XML để nhận biết các giá trị phần tử XML được chứa trong các cột XML.

Ví dụ, đây là cách bạn có thể xóa tất cả các thông tin khách hàng có mã định danh khách hàng 3227:

Liệt kê 25. Xóa dữ liệu của một khách hàng cụ thể

```
delete from clients

where id = 3227
```

Bạn có nhớ cách hạn chế các câu lệnh SELECT của SQL để trả về chỉ các hàng của các khách hàng sống trong vùng có mã bưu điện 95116 không? Nếu có, bạn có thể dễ dàng áp dụng hiểu biết đó để xóa các hàng theo dõi các khách hàng đó. Đây là cách để làm điều này bằng hàm XMLExists:

Liệt kê 26. Xóa dữ liệu của các khách hàng trong một mã vùng cụ thể

```
delete from clients

where xmlexists('$c/Client/Address[zip="95116"]'

passing clients.contactinfo as "c");
```

Lập chỉ mục

Cuối cùng, cần lưu ý rằng bạn có thể tạo các chỉ mục XML chuyên dụng để tăng tốc truy cập dữ liệu được lưu trữ trong các cột XML. Vì đây là một bài viết giới

thiệu và dữ liệu mẫu nhỏ, nên bài viết này không trình bày chủ đề đó ở đây. Tuy nhiên, trong các môi trường sản xuất, việc định nghĩa các chỉ mục phù hợp có thể rất quan trọng để đạt được hiệu năng tối ưu. Phần Tài nguyên của bài viết này có thể giúp bạn tìm hiểu thêm về công nghệ lập chỉ mục mới của DB2.

Tóm tắt

Bài viết này đã trình bày nhiều về nền tảng, làm nổi bật một số khía cạnh quan trọng của SQL/XML và cách bạn có thể sử dụng nó để truy vấn dữ liệu trong các cột XML. Chắc chắn có nhiều thứ mà bạn có thể làm với các hàm SQL và SQL/XML hơn những thứ mà chúng ta đã thảo luận ở đây. Bài viết này bao gồm một ví dụ Java đơn giản để minh họa cách bạn có thể sử dụng các dấu tham số với SQL/XML để truy vấn dữ liệu trong các cột XML. Chúng ta sẽ thảo luận các vấn đề phát triển ứng dụng chi tiết hơn trong một bài viết tương lai. Tuy nhiên, bài viết tiếp theo sẽ tìm hiểu một số khía cạnh thú vị về XQuery, một ngôn ngữ truy vấn mới được DB2 9 hỗ trợ.

Lời cảm ơn

Cảm ơn George Lapis, Matthias Nicola, Sriram Padmanabhan, Gary Robinson, Hardeep Singh, và Bert Van der Linden vì sự giúp đỡ của họ cho bài viết này.

Mục lục

- Cơ sở dữ liệu mẫu
- Môi trường truy vấn
- Các truy vấn chỉ dùng SQL
- Các truy vấn SQL/XML
- Các hoạt động cập nhật và xóa
- Tóm tắt

Truy vấn dữ liệu XML từ một bảng với kiểu dữ liệu XML

Mục đích của bài này nhằm hướng dẫn các quản trị viên cơ sở dữ liệu Microsoft SQL Server trong việc:

• Tạo giản đồ XML (XML Shema).
• Tạo một bảng với kiểu dữ liệu XML.
• Nhập file XML vào bảng với kiểu dữ liệu XML.
• Truy vấn file XML.
• Truy vấn file XML và đưa ra kết quả, tương tự như kết quả trả ra từ các lệnh Transact SQL Statement.

Bước 1

Đầu tiên, tạo một file C:\XML\Customer1.XML như bên dưới. File XML này chứa dữ liệu liên quan đến một khách hàng.

```
<?xml version="1.0" standalone="yes"?>
<Customer>
  <CustomerLogInfo>
    <Date>2007-03-31T06:40:38.0000000-05:00</Date>
    <user>james.brewer</user>
    <Userid>1AE</Userid>
    <ls>A-Accessible</ls>
    <eqtid>761</eqtid>
    <es>Stopped</es>
    <tp>30</tp>
  </CustomerLogInfo>
  <CustomerLogInfo>
    <Date>2007-03-31T06:40:38.0000000-05:00</Date>
    <user>james.brewer</user>
    <Userid>1AE</Userid>
    <ls>Not-Accessible</ls>
    <eqtid>870</eqtid>
    <es>Stopped</es>
    <tp>30</tp>
  </CustomerLogInfo>
  <CustomerLogInfo>
    <Date>2007-03-31T06:40:38.0000000-05:00</Date>
    <user>james.brewer</user>
    <Userid>1AE</Userid>
    <ls>A-Accessible</ls>
    <eqtid>97F</eqtid>
    <es>Started</es>
    <tp>30</tp>
  </CustomerLogInfo>
</Customer>
```

Bước 2

Tạo một cơ sở dữ liệu (CSDL) và một tập hợp XML Schema như bên dưới:

```
USE [master]
GO
/****** Object:  Database [XMLTest]    Script Date: 04/17/2007 01:49:43
******/
IF  EXISTS (SELECT name FROM sys.databases WHERE name = N'XMLTest')
DROP DATABASE [XMLTest]
go
create database XMLTest
go
use XMLTest
go
Create XML Schema Collection XMLTrack
as
N'<xs:schema
attributeFormDefault="unqualified"
elementFormDefault="qualified"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
<xs:element name="Customer">
<xs:complexType>
<xs:sequence>
<xs:element maxOccurs="unbounded" name="CustomerLogInfo">
<xs:complexType>
<xs:sequence>
<xs:element name="Date" type="xs:string" />
<xs:element name="user" type="xs:string" />
<xs:element name="Userid" type="xs:string" />
<xs:element name="ls" type="xs:string" />
<xs:element name="eqtid" type="xs:string" />
<xs:element name="es" type="xs:string" />
<xs:element name="tp" type="xs:int" />
</xs:sequence>
</xs:complexType>
</xs:element>
</xs:sequence>
</xs:complexType>
</xs:element>
</xs:schema>'
go
```

Chú ý: update tập hợp Schema dựa trên dữ liệu của riêng bạn trong file XML.

Bước 3

Tạo một bảng với kiểu dữ liệu XML:

```
USE [XMLTest]
GO
/****** Object:  Table [dbo].[XMLFiles]    Script Date: 04/17/2007
02:07:52 ******/
IF  EXISTS (SELECT * FROM sys.objects WHERE object_id =
```

```
OBJECT_ID(N'[dbo].[XMLFiles]') AND type in (N'U'))
DROP TABLE [dbo].[XMLFiles]

create table XMLFiles(Fileid int identity(1,1),
ImportedDate datetime constraint xmldatestamp default getdate(),
Filename varchar(500),
data xml (XMLTrack))
```

Bước 4

Nhập file XML vừa tạo (C:\XML\Customer1.XML), sử dụng hàm ***openrowset*** như bên dưới:

```
USE [XMLTest]
go
INSERT INTO XMLFiles(Filename,DATA)
SELECT 'Customer1' a,*
FROM OPENROWSET( BULK 'C:\XML\Customer1.xml' ,SINGLE_CLOB)
as mytable
go
```

Chú ý: Từ khoá SINGLE_BLOB sẽ nhập toàn bộ file XML cho cột có kiểu dữ liệu XML.

Bước 5

Truy vấn bảng XMLFiles, sử dụng các thao tác SQL như bên dưới:

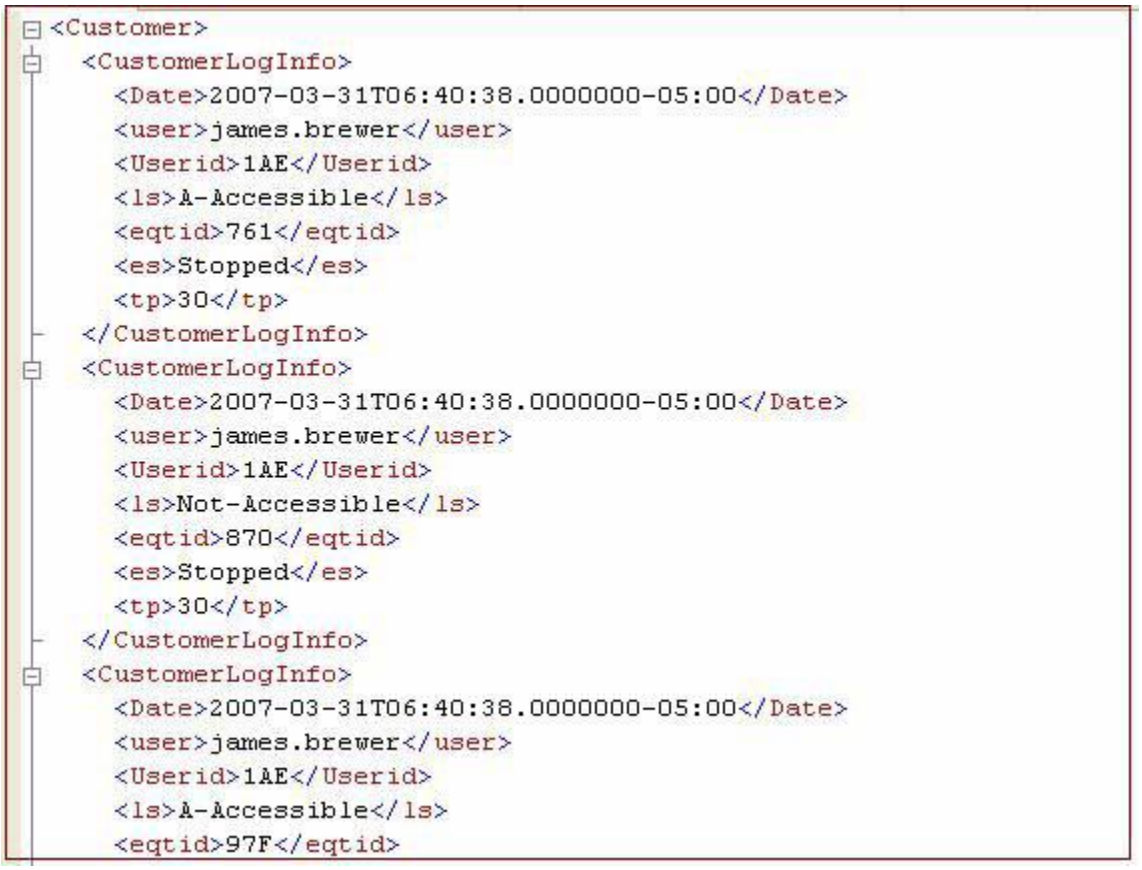
```
USE [XMLTest]
go
select * from XMLFiles where FileId=1
go
```

Lệnh này sẽ đưa ra các kết quả sau:

Hình 1.0

Khi kích chuột lên dữ liệu, nó cũng hiển thị dữ liệu XML (Hình 1.1).

```
<Customer>
<CustomerLogInfo>
<Date>2007-03-31T06:40:38.0000000-05:00</Date>
<user>james.brewer</user>
<Userid>1AE</Userid>
<ls>A-Accessible</ls>
<eqtid>761</eqtid>
<es>Stopped</es>
<tp>30</tp>
</CustomerLogInfo>
<CustomerLogInfo>
<Date>2007-03-31T06:40:38.0000000-05:00</Date>
<user>james.brewer</user>
<Userid>1AE</Userid>
<ls>Not-Accessible</ls>
<eqtid>870</eqtid>
<es>Stopped</es>
<tp>30</tp>
</CustomerLogInfo>
<CustomerLogInfo>
<Date>2007-03-31T06:40:38.0000000-05:00</Date>
<user>james.brewer</user>
<Userid>1AE</Userid>
<ls>A-Accessible</ls>
<eqtid>97F</eqtid>
<es>Started</es>
<tp>30</tp>
</CustomerLogInfo>
</Customer>
```



Hình 1.1

Bước 6

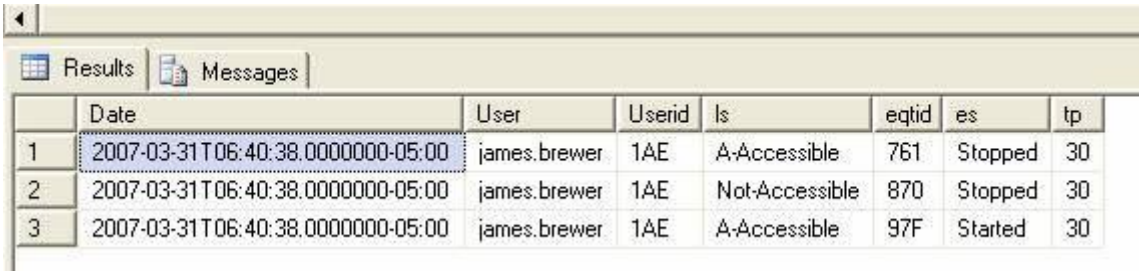
Bây giờ, truy vấn dữ liệu XML từ bảng để đưa ra một giao dịch SQL như tập hợp kết quả. Thực thi chương trình XQuery như bên dưới:

```
SELECT
ref.value ('Date', 'nvarchar(364)') as [Date],
ref.value ('user', 'nvarchar(364)') as [User],
ref.value ('Userid', 'nvarchar(364)') as [Userid],
ref.value ('ls', 'nvarchar(364)') as [ls],
ref.value ('eqtid', 'nvarchar(364)') as [eqtid],
ref.value ('es', 'nvarchar(364)') as [es],
ref.value ('tp', 'nvarchar(364)') as [tp]
FROM   XMLFiles CROSS APPLY Data.nodes ('//Customer/CustomerLogInfo')
R(ref)
where Fileid=1
```

Chương trình sẽ cho kết quả như Hình 1.2:

```
Date,User,Userid,ls,eqtid,es,tp
2007-03-31T06:40:38.0000000-05:00,james.brewer,1AE,A-
Accessible,761,Stopped,30
2007-03-31T06:40:38.0000000-05:00,james.brewer,1AE,Not-
Accessible,870,Stopped,30
```

```
2007-03-31T06:40:38.0000000-05:00,james.brewer,1AE,A-
Accessible,97F,Started,30
```



Results | Messages

	Date	User	Userid	ls	eqtid	es	tp
1	2007-03-31T06:40:38.0000000-05:00	james.brewer	1AE	A-Accessible	761	Stopped	30
2	2007-03-31T06:40:38.0000000-05:00	james.brewer	1AE	Not-Accessible	870	Stopped	30
3	2007-03-31T06:40:38.0000000-05:00	james.brewer	1AE	A-Accessible	97F	Started	30

Hình 1.2

Bước 7

Bây giờ lặp lại bước 4 và nhập lại dữ liệu.

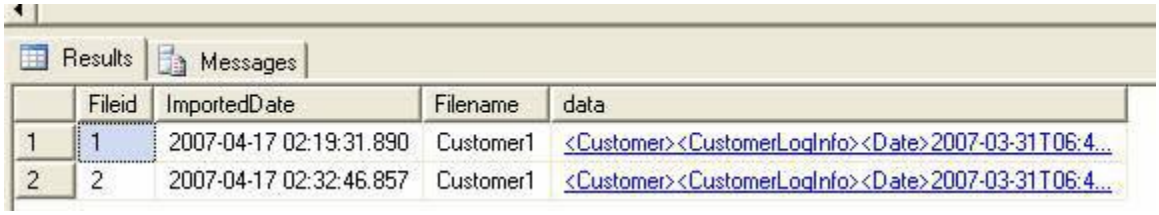
```
USE [XMLTest]
go
INSERT INTO XMLFiles(Filename,DATA)
SELECT 'Customer1' a,*
FROM OPENROWSET( BULK 'C:\XML\Customer1.xml' ,SINGLE_CLOB)
as mytable
go
```

Bước 8

Truy vấn bảng như bên dưới:

```
USE [XMLTest]
go
select * from XMLFiles
go
```

Kết quả là:



Results | Messages

	Fileid	ImportedDate	Filename	data
1	1	2007-04-17 02:19:31.890	Customer1	<Customer><CustomerLogInfo><Date>2007-03-31T06:4...
2	2	2007-04-17 02:32:46.857	Customer1	<Customer><CustomerLogInfo><Date>2007-03-31T06:4...

Hình 1.3

Bước 9

Để hiển thị tất cả dữ liệu từ cả hai hàng, chúng ta có thể viết truy vấn như bên dưới:

```
SELECT
ref.value ('Date', 'nvarchar(364)') as [Date],
```

```
ref.value ('user', 'nvarchar(364)') as [User],
ref.value ('Userid', 'nvarchar(364)') as [Userid],
ref.value ('ls', 'nvarchar(364)') as [ls],
ref.value ('eqtid', 'nvarchar(364)') as [eqtid],
ref.value ('es', 'nvarchar(364)') as [es],
ref.value ('tp', 'nvarchar(364)') as [tp]
FROM   XMLFiles CROSS APPLY Data.nodes ('//Customer/CustomerLogInfo')
R(ref)
where Fileid=1
union all
SELECT
ref.value ('Date', 'nvarchar(364)') as [Date],
ref.value ('user', 'nvarchar(364)') as [User],
ref.value ('Userid', 'nvarchar(364)') as [Userid],
ref.value ('ls', 'nvarchar(364)') as [ls],
ref.value ('eqtid', 'nvarchar(364)') as [eqtid],
ref.value ('es', 'nvarchar(364)') as [es],
ref.value ('tp', 'nvarchar(364)') as [tp]
FROM   XMLFiles CROSS APPLY Data.nodes ('//Customer/CustomerLogInfo')
R(ref)
where Fileid=2
```

Kết quả có dạng như Hình 1.4.

```
Date,User,Userid,ls,eqtid,es,tp
2007-03-31T06:40:38.0000000-05:00,james.brewer,1AE,A-
Accessible,761,Stopped,30
2007-03-31T06:40:38.0000000-05:00,james.brewer,1AE,Not-
Accessible,870,Stopped,30
2007-03-31T06:40:38.0000000-05:00,james.brewer,1AE,A-
Accessible,97F,Started,30
2007-03-31T06:40:38.0000000-05:00,james.brewer,1AE,A-
Accessible,761,Stopped,30
2007-03-31T06:40:38.0000000-05:00,james.brewer,1AE,Not-
Accessible,870,Stopped,30
2007-03-31T06:40:38.0000000-05:00,james.brewer,1AE,A-
Accessible,97F,Started,30
```

Chú ý: Nếu bạn đang có kế hoạch hiển thị tất cả dữ liệu từ từng hàng có kiểu dữ liệu XML, bạn có thể tạo một thủ tục lưu trữ với bảng tạm thời hoặc con trỏ…

Results | Messages

	Date	User	Userid	ls	eqtid	es	tp
1	2007-03-31T06:40:38.0000000-05:00	james.brewer	1AE	A-Accessible	761	Stopped	30
2	2007-03-31T06:40:38.0000000-05:00	james.brewer	1AE	Not-Accessible	870	Stopped	30
3	2007-03-31T06:40:38.0000000-05:00	james.brewer	1AE	A-Accessible	97F	Started	30
4	2007-03-31T06:40:38.0000000-05:00	james.brewer	1AE	A-Accessible	761	Stopped	30
5	2007-03-31T06:40:38.0000000-05:00	james.brewer	1AE	Not-Accessible	870	Stopped	30
6	2007-03-31T06:40:38.0000000-05:00	james.brewer	1AE	A-Accessible	97F	Started	30

Hình 1.4