Nested Subqueries

- A subquery is a select-from-where expression that is nested within another query.
- A common use of subqueries is to perform tests for set membership, set comparisons, and set cardinality.
- Ex : find names of students who have not taken any CS course !
- Nesting possible in select, from, where, …

Example Query

- Find courses offered in Fall 2009 and in Spring 2010

```
select distinct course_id
from section
where semester = ’ Fall’  and year= 2009 and
          course_id in (select course_id
                        from section
                        where semester = ’ Spring’
                        and year= 2010);
```

□ **SQL also provides ‘not in’**

□ **compare a value using ‘some’ or ‘all’ with a set**

Set Comparison

- Find names of instructors with salary greater than that of some (at least one) instructor in the Biology department.

```
select name
from instructor
where salary > some (select salary
                     from instructor
                     where dept_name = ’ Biology’ );
```

Definition of some Clause

- v <comp-op> some *r*
  where <comp-op> can be: $<$, $\leq$, $>$, $=$, $\neq$

$$(5 < \textbf{some} \begin{array}{|c|}\hline 0\\ \hline 5\\ \hline 6\\ \hline\end{array}) = \text{true}$$

(read: 5 < some tuple in the relation)

$$(5 < \textbf{some} \begin{array}{|c|}\hline 0\\ \hline 5\\ \hline\end{array}) = \text{false}$$

$$(5 = \textbf{some} \begin{array}{|c|}\hline 0\\ \hline 5\\ \hline\end{array}) = \text{true}$$

$$(5 \neq \textbf{some} \begin{array}{|c|}\hline 0\\ \hline 5\\ \hline\end{array}) = \text{true (since } 0 \neq 5)$$

(= **some**) $\equiv$ **in**

However, ($\neq$ **some**) $\not\equiv$ **not in**

Definition of all Clause

- v <comp-op> all *r*

$$(5 < \textbf{all} \begin{array}{|c|}\hline 0 \\ \hline 5 \\ \hline 6 \\ \hline \end{array}) = \text{false}$$

$$(5 < \textbf{all} \begin{array}{|c|}\hline 6 \\ \hline 10 \\ \hline \end{array}) = \text{true}$$

$$(5 = \textbf{all} \begin{array}{|c|}\hline 4 \\ \hline 5 \\ \hline \end{array}) = \text{false}$$

$$(5 \neq \textbf{all} \begin{array}{|c|}\hline 4 \\ \hline 6 \\ \hline \end{array}) = \text{true (since } 5 \neq 4 \text{ and } 5 \neq 6\text{)}$$

(≠ **all**) ≡ **not in**

However, (= **all**) ≢ **in**

Test for Empty Relations

- The exists construct returns the value true if the argument subquery is nonempty.
- exists $r \Leftrightarrow r \neq \emptyset$ gives true
- not exists $r \Leftrightarrow r = \emptyset$ gives true

Correlation Variables

- Inner query refers to attributes from relations in the outer query
- Get courses from Fall 2009 which also ran in Spring 2010

```
  select course_id
   from section as S
   where semester = ’Fall’ and year= 2009 and
          exists (select *
                  from section as T
                  where semester = ’Spring’
                       and year =          2010
                      and S.course_id=   T.course_id);
```

- Correlated subquery
- Uses correlation name or correlation variable

Not Exists

- Find all students who have taken all courses offered in the Biology department.

```
select distinct S.ID, S.name
from student as S
where not exists ( (select course_id
                    from course
                    where dept_name = ’ Biology’ )
                 except
                    (select T.course_id
                     from takes as T
                     where S.ID = T.ID));
```

Biology courses

This student’s courses

- **Note that $X – Y = Ø \Leftrightarrow X \subseteq Y$**
- ***Note:* Cannot write this query using = all and its variants**

? All courses done by a student are from Biology
-- none studied by her is from non-Biology

Test for Absence of Duplicate Tuples

- The unique construct tests whether a subquery has any duplicate tuples in its result.
    - (Evaluates to “true” on an empty set)
- Find all courses that were offered at most once in 2009

```
select T.course_id
from course as T
where unique (select R.course_id
              from section as R
              where T.course_id= R.course_id
                    and R.year = 2009);
```

Subqueries in the From Clause

- Its result treated like a table
- Find the average salaries of those departments where the average salary is greater than $42,000.

```
 select dept_name, avg_salary
from (select dept_name, avg (salary) as avg_salary
      from instructor
      group by dept_name)
where avg_salary > 42000;
```

- Note that we do not need to use the having clause

- Another way to write above query

```
select dept_name, avg_salary
from (select dept_name, avg (salary)
      from instructor
      group by dept_name)
      as dept_avg (dept_name,  avg_salary)
where avg_salary > 42000;
```

With Clause

- The with clause provides a way of defining a temporary 'virtual' table whose definition is available only to this query.
- Find all departments with the maximum budget

```
with max_budget (value) as
    (select max(budget)
     from department)
select budget
from department, max_budget
where department.budget = max_budget.value;
```

Supported by most database systems, with minor syntax variations

Complex Queries using With Clause

- allowing decomposing complex queries in steps
- Find all departments having total salary greater than the average of the total salary of all departments

```
with dept _total (dept_name, value) as
        (select dept_name, sum(salary)
         from instructor
         group by dept_name),
dept_total_avg(value) as
       (select avg(value)
       from dept_total)
select dept_name
from dept_total, dept_total_avg
where dept_total.value >= dept_total_avg.value;
```

Scalar Subquery

- subquery producing a single value

```
select name
from instructor
where  salary * 10  >
    (select budget  from department
     where department.dept_name = instructor.dept_name)
```

- Runtime error if subquery returns more than one result tuple

Modification of the Database

- Deletion of tuples from a given relation
- Insertion of new tuples
- Updating values in some tuples

```
delete from R
where <condition>

    insert into R              insert into R
      values (….)                select ….
                        -- columns being initialized can be listed as R(A,B…)

update <R>
set <attribute> = expression
where  ……
```

Deletion …

- Delete all instructors

```
delete from instructor
```

- Delete *instructors* from Finance dept

```
delete from instructor
where dept_name= ’Finance’;
```

- Delete *instructors* located in the Watson building (need to refer to another table

```
delete from instructor
where dept_name in
       (select dept_name
        from department
        where building = ’Watson’);
```

Deletion (Cont.)

- Delete all instructors whose salary is less than the average salary of instructors (self reference)

delete from ***instructor***
where *salary* < (**select avg** (*salary*) **from** ***instructor*****)**;

- **Problem: as we delete tuples from deposit, the average salary changes**
- **Solution used in SQL:**

1. **Compute avg salary and find all tuples to delete**
2. **Next, delete all tuples found above**
   (without recomputing avg or retesting the tuples)

Insertion

- Add a new *course*

```
 insert into course
     values (’CS-437’, ’DB Systems’, ’Comp. Sci.’, 4);
```

- or equivalently

```
insert into course (course_id, title, dept_name, credits)
      values (’CS-437’, ’DB Systems’, ’Comp. Sci.’, 4);
```

Insertion (Cont.)

- Add all instructors to the *student* with tot_creds as 0

```
insert into student
     select ID, name, dept_name, 0
    from   instructor
```

- The select from where statement is evaluated fully before any of its results are inserted into the relation (otherwise queries like

  ```
  insert into table1 select * from table1
  ```

  would cause problems (infinite tuples will be inserted !)

Modification of the Database – Updates

- Increase salaries of instructors whose salary is over $100,000 by 3%, and all others receive a 5% raise
  - Write two update statements:

```
update instructor
   set salary = salary * 1.03
   where salary > 100000;
update instructor
    set salary = salary * 1.05
    where salary <= 100000;
```

  - The order is important
  - Can be done better using the case statement (next slide)

Case Statement for Conditional Updates

- Same query as before but with case statement

```
update instructor
    set salary = case
            when salary <= 100000 then salary * 1.05
            else salary * 1.03
            end
```

Updates with Scalar Subqueries

- Recompute and update tot_creds value for all students by summing credits of courses taken and passed

```
update student S
set tot_cred = ( select sum(credits)
                 from takes natural join course
                where S.ID= takes.ID and
                      takes.grade <> 'F' and
                      takes.grade is not null);
```

- Sets *tot_creds* to null for students who have not taken any course
- Instead of sum(*credits*), use:

```
case
   when sum(credits) is not null then sum(credits)
   else 0
end
```