CSCD 327: Relational Database Systems

Subqueries

Instructor: Dr. Dan Li

Nested Subqueries

- SQL provides a mechanism for the nesting of subqueries.
- A **subquery** is a **select-from-where** expression that is nested within another query.
- In the most common uses, a subquery produces a single column of data as its query results.
- Subqueries are most frequently used in the WHERE clause of a SQL statement.
- A common use of subqueries is to perform tests for set membership, set comparisons, and set cardinality.

Set Membership

- 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);
```

This provides a way to implement INTERSECT.

- Find courses offered in Fall 2009 but not in Spring 2010

```
select distinct course_id
from section
where semester = 'Fall' and year= 2009 and
        course_id  not in (select course_id
                           from section
                           where semester = 'Spring' and year= 2010);
```

This provides a way to implement EXCEPT.

Example Query

- Find the total number of (distinct) students who have taken course sections taught by the instructor with *ID* 10101

```
select count (distinct ID)
from takes
where (course_id, sec_id, semester, year) in
                    (select course_id, sec_id, semester, year
                   from teaches
                   where teaches.ID= 10101);
```

This provides a way to avoid using JOIN.

Note: Above query can be written in a much simpler manner. The formulation above is simply to illustrate SQL features.

Set Comparison (=, <>, <, <=, >, >=)

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

```
select distinct T.name
from instructor as T, instructor as S
where T.salary > S.salary and S.dept_name = 'Biology';
```

Same query using > **some** clause

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

By default, the comparison using > means greater than some, so the keyword **some** can be omitted.
Any is the same as **some**.

Definition of Some Clause

- F <comp> **some** $r \Leftrightarrow \exists\ t \in r$ such that (F <comp> t)
  Where <comp> can be: $<,\ \leq,\ >,\ =,\ \neq$

(5 < **some** | 0 / 5 / 6 |) = true (read: 5 < some tuple in the relation)

(5 < **some** | 0 / 5 |) = false

(5 = **some** | 0 / 5 |) = true

(5 ≠ **some** | 0 / 5 |) = true (since 0 ≠ 5)

(= **some**) is the same as **in**
However, (≠ **some**) is not the same as **not in**

Example Query

- Find the names of all instructors whose salary is greater than the salary of **all** instructors in the Biology department.

```
select name
from instructor
where salary > all (select salary
                    from instructor
                    where dept_name = 'Biology');
```

Definition of all Clause

- F <comp> **all** $r \Leftrightarrow \forall\ t \in r$ (F <comp> *t)*

(5 < **all** | 0, 5, 6 |) = false

(5 < **all** | 6, 10 |) = true

(5 = **all** | 4, 5 |) = false

(5 ≠ **all** | 4, 6 |) = true (since 5 ≠ 4 and 5 ≠ 6)

(≠ **all**) is the same as **not in**
However, (= **all**) is not the same as **in**

Existence Test

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

Correlation Variables

- Yet another way of specifying the query "Find all courses taught in both the Fall 2009 semester and in the Spring 2010 semester"

```
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);
```

This provides another way to implement INTERSECT.

- **Correlated subquery**
- **Correlation name** or **correlation variable*, course_id* in this example.**

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'
                         and course_id not in
                   ( select T.course_id
                     from takes as T
                     where S.ID = T.ID));
```

X

Not in

Y

Note that $X - Y = \emptyset \Leftrightarrow X \subseteq Y$

Note: Cannot write this query using = **all** and its variants

Subqueries in the FROM Clause

- SQL allows a subquery expression to be used in the **from** clause
- Find the average instructors' 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
) AS dept_avg
WHERE avg_salary >42000
```

dept_name	avg_salary
Biology	72000.000000
Comp. Sci.	77333.333333
Elec. Eng.	80000.000000
Finance	85000.000000
History	61000.000000
Physics	91000.000000

MySQL required each derived table to have its own name, as defined "dept_avg".

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

Subqueries in the FROM Clause (Cont.)

- And yet another way to write it: **lateral** clause

```
select name, salary, avg_salary
 from instructor I1,
        lateral (select avg(salary) as avg_salary
                 from instructor I2
                 where I2.dept_name= I1.dept_name);
```

- Lateral clause permits later part of the **from** clause (after the lateral keyword) to access correlation variables from the earlier part.
- Note: lateral is part of the SQL standard, but is not supported on many database systems (e.g. MySQL); some databases such as SQL Server offer alternative syntax

Subqueries in the HAVING Clause

- *List the salespeople whose average order size for products manufactured by ACI is at least as big as that salesperson's overall average order size.*

```
SELECT NAME, AVG(AMOUNT)
  FROM SALESREPS, ORDERS
 WHERE EMPL_NUM = REP
   AND MFR = 'ACI'
 GROUP BY NAME, EMPL_NUM
HAVING AVG(AMOUNT) >= (SELECT AVG(AMOUNT)
                         FROM ORDERS
                        WHERE REP = EMPL_NUM);
```

```
NAME          AVG(AMOUNT)
----------- ------------
Bill Adams      $7,865.40
Sue Smith      $15,000.00
Tom Snyder     $22,500.00
```

Temporary Table

- Find all departments with the maximum budget

```
create temporary table max_budget
        (select max(budget)  as value
         from department)
select dept_name, budget
from department, max_budget
where department.budget = max_budget.value;
```

Temporary tables are just like regular tables, except they exist only for the current session, and are dropped when the session ends.

Scalar Subquery

- Scalar subquery is one which is used where a single value is expected
- Find the budget and the total salary in each department
- E.g. **select** *dept_name*, budget,
  (**select sum(salary**)
  from *instructor*
  where *department*.*dept_name* = *instructor*.*dept_name*)
  as *total_salary*
  from *department*;

  Scalar subquery in SELECT clause
- Runtime error if subquery returns more than one result tuple