

## Module - II

### SQL - Structured Query Language

*Lecture - 6*

## Nested Subqueries

## SQL - Nested Subqueries

### ➤ Nested Subqueries

- Set Membership
- Set Comparison
- Test for Empty Relations
- Test for the Absence of Duplicate Tuples
- Subqueries in the From Clause
- The With Clause
- Scalar Subqueries
- Scalar Without a From Clause

# 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**.
- The nesting can be done in the following SQL query

```
select  $A_1, A_2, \dots, A_n$   
from  $r_1, r_2, \dots, r_m$   
where  $P$ 
```

# Nested Subqueries

- **From clause:**  $r_i$  can be replaced by any valid subquery
- **Where clause:**  $P$  can be replaced with an expression of the form:

$B$  `<operation>` (subquery)

$B$  is an attribute and `<operation>` to be defined later.

- **Select clause:**  
 $A_i$  can be replaced by a **subquery** that generates a single value.

# Nested Subqueries

- **From clause:**  $r_i$  can be replaced by any valid subquery
- **Where clause:**  $P$  can be replaced with an expression of the form:

$B$  `<operation>` (subquery)

$B$  is an attribute and `<operation>` to be defined later.

- **Select clause:**  
 $A_i$  can be replaced by a **subquery** that generates a single value.

# Set Membership Operations

- `in`
- `not in`

# Set Membership

- Find courses offered in Fall 2017 and in Spring 2018

```
select distinct course_id
from section
where semester = 'Fall' and year= 2017 and
       course_id in (select course_id
                       from section
                       where semester =
                           'Spring' and year= 2018);
```

# Set Membership

- Find courses offered in Fall 2017 but not in Spring 2018

```
select distinct course_id
from section
where semester = 'Fall' and year= 2017 and
       course_id not in (select course_id
                          from section
                          where semester =
                          'Spring' and year= 2018);
```



# Set Membership

- Name all instructors whose name is neither “Mozart” nor Einstein”

```
select distinct name
```

```
from instructor
```

```
where name not in ('Mozart', 'Einstein')
```

# Set Membership

- 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);
```

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

# Set Comparison Operations

- some
- all

## Set Comparison - "some" Clause

- 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';
```

# Set Comparison - "some" Clause

- Same query using **> some** clause

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

# Definition of "some" Clause

- $F \langle \text{comp} \rangle \text{ some } r \Leftrightarrow \exists t \in r \text{ such that } (F \langle \text{comp} \rangle t)$

Where  $\langle \text{comp} \rangle$  can be:  $<$ ,  $\leq$ ,  $>$ ,  $=$ ,  $\neq$

$(5 < \text{some } \begin{array}{|c|} \hline 0 \\ \hline 5 \\ \hline 6 \\ \hline \end{array}) = \text{true}$  (read: 5 < some tuple in the relation)

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

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

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

$(= \text{ some}) \equiv \text{in}$

However,  $(\neq \text{ some}) \not\equiv \text{not in}$

# Set Comparison - "all" Clause

- 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

- $F \langle \text{comp} \rangle \mathbf{all} \ r \Leftrightarrow \forall t \in r \ (F \langle \text{comp} \rangle t)$

$$(5 \langle \mathbf{all} \begin{array}{|c|} \hline 0 \\ \hline 5 \\ \hline 6 \\ \hline \end{array} \rangle) = \text{false}$$

$$(5 \langle \mathbf{all} \begin{array}{|c|} \hline 6 \\ \hline 10 \\ \hline \end{array} \rangle) = \text{true}$$

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

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

$(\neq \mathbf{all}) \equiv \mathbf{not\ in}$

However,  $(= \mathbf{all}) \neq \mathbf{in}$



# Test for Empty Relations

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

## Use of “exists” Clause

- Yet another way of specifying the query “Find all courses taught in both the Fall 2017 semester and in the Spring 2018 semester”.

```
select course_id
from section as S
where semester = 'Fall' and year = 2017 and
      exists (select *
              from section as T
              where semester = 'Spring' and year=
                2018 and S.course_id = T.course_id);
```

- **Correlation name** – variable **S** in the **outer query**
- **Correlated subquery** – the **inner query**

# Use of “not exists” Clause

- 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) );
```

- First nested query lists all courses offered in Biology
- Second nested query lists all courses a particular student took

# Use of “not exists” Clause

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

# Test for Absence of Duplicate Tuples

- The **unique** construct tests whether a **subquery** has any **duplicate tuples** in its result.
- The **unique** construct evaluates to “**true**” if a given **subquery** contains **no duplicates** .

# Test for Absence of Duplicate Tuples

- Find all courses that were offered at most once in 2017

```
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 = 2017);
```

# Subqueries in the Form Clause

- SQL allows a **subquery** expression to be used in the **from** clause.
- **Example:** *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)
where avg_salary > 42000;
```

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

# Subqueries in the Form 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 relation** whose definition is available only to the query in which the **with** clause occurs.
- **Example:** *Find all departments with the maximum budget*

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

# Complex Queries using With Clause

- **Example:** *Find all departments where the total salary is greater than the average of the total salary at 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

- **Scalar subquery** is one which is used where a **single value** is expected.
- **Example:** *List all departments along with the number of instructors in each department*

```
select dept_name,  
       (select count(*)  
        from instructor  
        where department.dept_name =  
          instructor.dept_name)  
       as num_instructors  
from department;
```

- **Runtime error** if **subquery** returns more than one result tuple.