Databases Design. Introduction to SQL

LECTURE 11

Nested queries

The Complete Select Statement

```
SELECT attribute(s)
FROM table(s)

[WHERE selection condition(s)]
[GROUP BY condition]
[HAVING selection condition]
[ORDER BY condition]
```

Nested Queries

Nested query (Subquery or **Inner query)** is a query within another SQL query.

Subquery is used to return data that will be used in the main query as a condition to further restrict the data to be retrieved.

Subqueries allow to express a selection condition using a tradition SELECT-FROM-WHERE statement.

Nested Queries

Subqueries can be used with the SELECT, INSERT, UPDATE, and DELETE statements along with the operators like =, <, >, >=, <=, IN, etc.

There are a few rules that subqueries must follow:

- Subqueries must be enclosed within parentheses – (…).
- Subqueries that return more than one row can only be used with multiple value operators such as the IN operator.

Nested Queries

- Subqueries are most frequently used with the SELECT statement. For example, the basic syntax for using subquery in WHERE clause:

```
SELECT attribute(s)
FROM table(s)
WHERE attribute OPERATOR
    (SELECT attribute(s)
    FROM table(s)
    [WHERE] );
```

Nested Queries

- There are three types of subqueries: scalar, row, table. Scalar subquery returns a single column and a single row (a single value).

- Example: Return the first and last name of the student who has group's name = ‘CSSE-01‘.

```
SELECT fname, lname
FROM Students
WHERE group_id =
      (SELECT group_id
      FROM Groups
      WHERE name=‘CSSE-01’);
```

Subqueries with comparison operators

- Using of comparison operators is possible only when a result of a subquery is one value (one field).

Operator	Description
<	less than
>	greater than
<=	less than or equal to
>=	greater than or equal to
=	equal
<> or !=	not equal

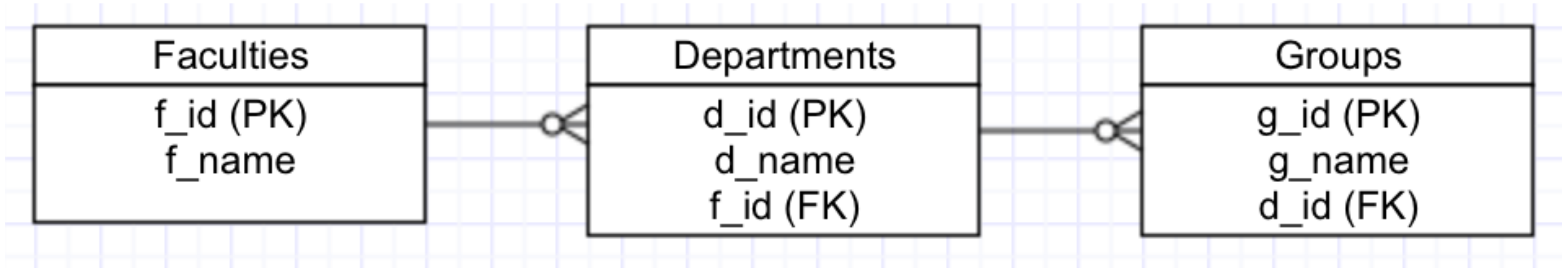
Subqueries with comparison operators

```
SELECT fname, lname, gpa
FROM Students
WHERE gpa > (SELECT avg(gpa)
                         FROM Students);
```

- -

```
SELECT fname, lname, gpa
FROM Students
WHERE
gpa > (SELECT avg(gpa) FROM Students)
AND
gpa < (SELECT max(gpa) FROM Students);
```

Subqueries with comparison operators



```
SELECT f_name
FROM   Faculties
WHERE f_id = (
    SELECT          f_id
    FROM Departments
    WHERE d_id = (
        SELECT d_id
        FROM Groups
        WHERE g_name = ‘CSSE - 01’));
```

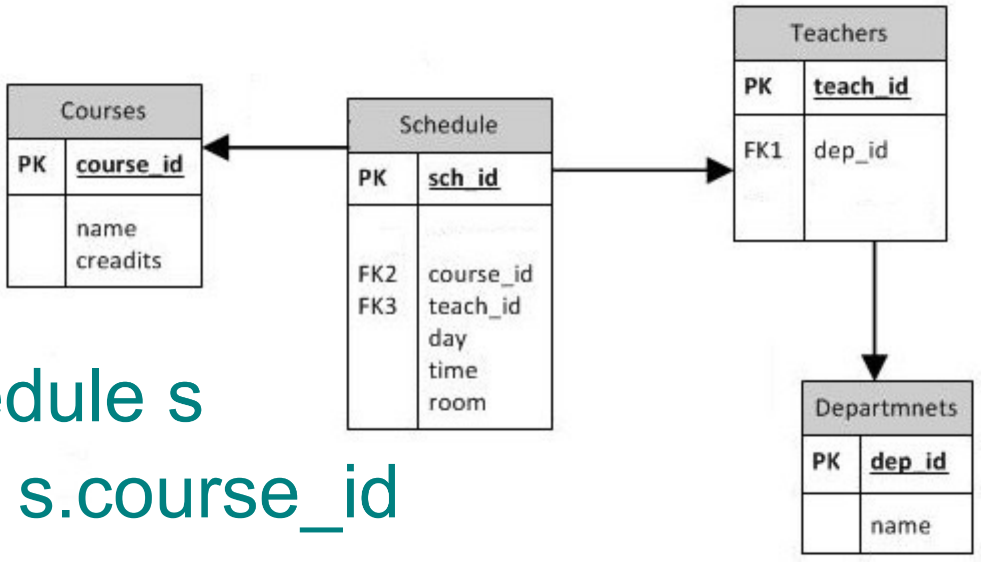
Set Membership

- The IN and NOT IN operators can be used to test simple set membership.

- IN and NOT IN are typically used in subqueries in WHERE. For example,

```
WHERE … IN (SELECT …);
WHERE … NOT IN (SELECT …);
```

Set Membership: example



```
SELECT c.name
FROM Course c, Schedule s
WHERE c.course_id = s.course_id
AND s.teach_id IN (
        SELECT t.teach_id
        FROM Teachers t, Department d
        WHERE t.dep_id=d.dep_id
        AND d.name='CET');
```

Another query with the identical result

```
SELECT c.name
FROM Courses c, Schedule s, Teachers t,
   Department d
WHERE c.course_id = s.course_id
   AND s.teach_id = t.teach_id
   AND t.dep_id=d.dep_id
   AND d.name=‘CET’;
```

EXISTS and NOT EXISTS

- SQL allows testing the emptiness of a subquery’s result using the EXISTS and NOT EXISTS keywords.

- The EXISTS keyword tests if a result is not empty. NOT EXISTS tests if a result is empty.

- If it returns at least one row, the result of EXISTS is "true"; if the subquery returns no rows, the result of EXISTS is "false".

EXISTS: example

```
SELECT fname, name
FROM Students
WHERE EXISTS  (
   SELECT *
   FROM Students
   WHERE group_id = 1);
```

ANY

... attribute OPERATOR ANY (subquery)

- The ANY operator compares the value to each value returned by the subquery. Therefore ANY keyword (which must follow a comparison operator) returns TRUE if the comparison is TRUE for ANY of the values in the column that the subquery returns.
- "IN" is equivalent to "= ANY".

ANY: example

```
SELECT *
FROM Students s
WHERE s. bdate < ANY (
    SELECT t.bdate
    FROM Teachers t);
```

ALL

... attribute OPERATOR ALL (subquery)

- The ALL operator compares value to every value returned by the subquery. The result of ALL is true if all rows yield true. The result is false if any false result is found.
- "NOT IN" is equivalent to "<> ALL".

ALL: example

```
SELECT *
FROM Students s
WHERE s. bdate < ALL (
    SELECT t.bdate
    FROM Teachers t);
```

Subquery in FROM (1)

- A subquery can also be found in SELECT or FROM clauses.

```
SELECT num_of_stud
FROM (
  SELECT group_id, count(*)
                              AS num_of_stud
  FROM Students
  GROUP BY group_id) StudNum;
```

HAVING vs Subquery

- Example with HAVING from the last lecture:

```
SELECT group_id, count(*)
FROM Students
GROUP BY group_id
HAVING count(*) > 20;
```

- The same result with subquery:

```
SELECT *
 FROM ( SELECT group_id, count(*) AS num_of_stud
             FROM Students
             GROUP BY group_id) StudNum
 WHERE num_of_stud > 20;
```

Subqueries in FROM (2)

```
SELECT *
FROM
(SELECT count(*) FROM students) students,
(SELECT count(*) FROM teachers) teachers;
```

count	count
…	… .

Subquery in INSERT (1)

- Subqueries also can be used with INSERT statements. The INSERT statement uses the data returned from the subquery to insert into another table.

- Example:

```
INSERT INTO Teachers (fname, lname)
      (SELECT  fname, lname
       FROM Students
      WHERE stud_id = 01);
```

Subquery in INSERT (2)

```
INSERT INTO Teachers (teach_id, fname,
lname, dep_id)
VALUES (1, ‘…’,’…’,
                (SELECT dep_id
                FROM Departments
                WHERE dep_name=‘CET’));
```

Subquery in DELETE

- The subquery can be used in conjunction with the DELETE statement.

```
DELETE
FROM Students
WHERE group_id =
    (SELECT group_id
    FROM Groups
    WHERE name=‘CSSE-01’);
```

Subquery in UPDATE

- The subquery can be used in conjunction with the UPDATE statement. Either single or multiple columns in a table can be updated when using a subquery with the UPDATE statement.

```
UPDATE Students
SET group_id=group_id + 1
WHERE group_id =
    (SELECT group_id
    FROM Groups
    WHERE name = ‘CSSE-01’);
```

Books

- Connolly, Thomas M. Database Systems: A Practical Approach to Design, Implementation, and Management / Thomas M. Connolly, Carolyn E. Begg.- United States of America: Pearson Education
- Garcia-Molina, H. Database system: The Complete Book / Hector Garcia-Molina.- United States of America: Pearson Prentice Hall
- Sharma, N. Database Fundamentals: A book for the community by the community / Neeraj Sharma, Liviu Perniu.- Canada

- www.postgresql.org/docs/manuals/
- www.postgresql.org/docs/books/