# CSCB20
# Introduction to Databases and Web Application

## Week 4 - SQL and Introduction to Web Development

Dr. Purva R Gawde

# Topics covered last week

- Creating tables, setting constraints…
- Inserting and updating tables
- More query commands
- Creating views
- Joins
  - Left
  - Right
  - Full
- More on NULL values

# Topics covered this week

- SQL
  - Case
  - Subquery
- Introduction to Web development
  - Frontend
  - Backend
  - Intro to HTML

# Case

- If you have a list of values and you want to select any one of them based on some conditions.

- CASE expression will evaluate these list of conditions for all the values. If the condition is true, it will return that value.

# SQLite Simple Case Example

- Simple CASE
- Query to group students with either ‘Science’ or ‘Other’ department group

```
SELECT
  name,
  CASE dept_name
     WHEN 'Biology' THEN 'Science'
     WHEN 'Comp. Sci.' THEN 'Science'
     WHEN 'Physics' THEN 'Science'
     WHEN 'Elec. Eng.' THEN 'Science'
    ELSE 'Other'
  END dept_group
FROM student NATURAL JOIN department;
```

# SQLite Searched Case

- Evaluates a list of expressions to decide the result.
- Searched CASE

```
CASE
     WHEN bool_expression_1 THEN result_1
     WHEN bool_expression_2 THEN result_2
     [ ELSE result_else ]
END
```

# SQLite Searched Case Example 1

```
SELECT
  name,
  CASE
    WHEN tot_cred < 100 THEN '<100'
    WHEN tot_cred > 100 THEN '>100'
    ELSE tot_cred
  END tot_cred
FROM Student;
```

# SQLite Searched Case Example 2

```
SELECT
    course_id,
    sec_id,
    building,
    room_number,
    CASE
        WHEN room_number IS NULL THEN 'Room Unknown'
        WHEN room_number < 200 THEN 'Small Room'
        WHEN room_number >= 200 THEN 'Large Room'
        ELSE 'Other'
    END AS room_category
FROM section;
```

# Subqueries

- A subquery is a select-from-where expression that is nested within another query.
- Where can a subquery go?
  - In SELECT or WHERE or FROM clause
  - Most common clause for subquery is WHERE

```
SELECT column_1
FROM table_1
WHERE column_1 = (
   SELECT column_1
   FROM table_2
);
```

# Subquery in WHERE clause

Set Membership, Set comparison, Test for Empty relations, Test for duplicate tuples

# Subqueries - set membership

- Find all the courses taught in the both the Fall 2017 and Spring 2018 semesters.
  - ```
    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);
    ```
    ```

# Subqueries - set membership

- Find all the courses taught in Fall 2017 but not in  Spring 2018 semesters.
  - ```
    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);
    ```
- Also works for following query
- Select the names of instructors whose names are neither “Mozart” or “Einstein”.

```
SELECT DISTINCT name
FROM instructor
WHERE name NOT IN ('Mozart', 'Einstein');
```
```

# Subqueries - set membership

- find the total number of (distinct) students who have taken course sections taught by the instructor with ID 10101.
- This nested query is called a subquery.

```
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');
```

# Subqueries - Test for Empty Relations

- **EXISTS**: a feature for testing whether a subquery has any tuples in its result
- Find all courses taught in both the Fall 2017 semester and in the Spring 2018 semester

```
SELECT DISTINCT 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** from an outer query (S in the above query), can be used in a subquery in the where clause
- A subquery that uses a correlation name from an outer query is called a **correlated subquery.**

# Subqueries - Test for Empty Relations

- Find the names of all instructors whose salary is greater than 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';
```

- Let's rewrite this query using subquery..

```
SELECT I.name
FROM instructor as I
WHERE EXISTS (SELECT salary
              FROM instructor as S
              WHERE S.dept_name = 'Biology' AND I.salary > S.salary);
```

# Subqueries - Test for Empty Relations

- **NOT EXISTS**: test for the nonexistence of tuples in a subquery
- Find all students who have taken all courses offered in the “Physics” department

```
SELECT S.ID, S.name
FROM Student as S
WHERE NOT EXISTS
     (SELECT course_id
     FROM course
     WHERE dept_name = 'Physics'
     EXCEPT
     SELECT T.course_id
     FROM takes as T
     WHERE S.ID = T.ID);
```

- **Correlation name** from an outer query (S in the above query), can be used in a subquery in the where clause
- A subquery that uses a correlation name from an outer query is called a **correlated subquery.**

# Subqueries - Test for the Absence of Duplicate Tuples

- **UNIQUE:** Construct to return the value true if the argument subquery contains no duplicate tuples
- NOT supported in sqlite
- Find all courses that were offered at most once in 2017
- Alternative way:

```
SELECT T.course_id
FROM course AS T
WHERE 1>= (SELECT count(R.course_id)
          FROM section AS R
          WHERE T.course_id = R.course_id and R.year = 2017);
```

# Subqueries - Test for the Absence of Duplicate Tuples

- **NOT UNIQUE**
- NOT supported in sqlite
- Find all courses that were offered at least twice in 2017

```
SELECT T.course_id
FROM course AS T
WHERE 1 < (SELECT count(R.course_id)
              FROM section AS R
              WHERE T.course_id = R.course_id and R.year = 2017);
```

# Subquery in FROM clause

# Subqueries - 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)
WHERE avg_salary > 42000;
```

- We can give subquery result relation a name

```
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;
```

# Subqueries - In the From Clause

- **Find the maximum across all departments of the total of all instructors' salaries in each department.**

```
SELECT max(tot_salary)
FROM (SELECT dept_name, sum(salary) AS tot_salary
     FROM instructor
     GROUP BY dept_name) AS dept_total;
```

- Subqueries in from clause **cannot use correlation variables** from other relations in the same from clause

# Subquery Example

- Using subquery in the FROM clause example (**myuni.db**)

```
SELECT
  s.name, a.mark
FROM student AS s
INNER JOIN
(
   SELECT StudentId, mark
   FROM test AS t
   INNER JOIN mark AS m ON t.TestId = m.TestId
) AS a ON s.StudentId = a.StudentId;
```

- s.name is selected from the main query that gives the name of students and
- a.Mark is selected from the subquery; that gives marks obtained by each of these students

# Subquery Example

- Departments' names that don't exist in the Students table (**myuni.db**)
- Using subquery in the WHERE clause example

```
SELECT dept_name
FROM Department AS d
WHERE NOT EXISTS (SELECT DepartmentId
                  FROM Student AS s
                  WHERE d.DepartmentId = s.DepartmentId);
```

# Subquery in SELECT statement

# Subqueries - Scalar Subqueries

- SQL allows subqueries to occur wherever an expression returning a value is permitted, provided the subquery returns only one tuple containing a single attribute; such sub- queries are called **scalar subqueries.**

```
SELECT dept_name,
(SELECT count(*)
 FROM instructor
 WHERE department.dept_name = instructor.dept_name)
AS num_instructors
FROM department;
```

- The subquery in this example is guaranteed to return only a single value since it has a count(*) aggregate without a group by.
- correlation variables : attributes of relations in the from clause of the outer query, such as `department.dept_name`