IS5102
Database Management Systems
Lecture 11: Advanced SQL

Alexander Konovalov
alexander.konovalov@st-andrews.ac.uk

(with thanks to Susmit Sarkar)

2021

Before

- Defining Table Structure
- Querying Tables
- Modifying Data
- Orderings and Aggregates
- Integrity Constraints
- Views

This week: Advanced SQL

- Nested queries (subqueries)
- Join Expressions
- Authorisation
- Functions
- Triggers

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
- A common use of subqueries is to perform tests for set membership, set comparisons, and set cardinality

Find names of instructors with salary greater than that of **some** (at least one) instructor in the Physics and Astronomy department

This is an example demonstrated earlier:

```
SELECT DISTINCT T.instr_name
  FROM instructor AS T, instructor AS S
 WHERE T.salary > S.salary
   AND S.dept_id = 'PHYS';
```

In MySQL (but not in SQLite) the same result could be achieved using SOME clause

```
SELECT instr_name
  FROM instructor
 WHERE salary > SOME(SELECT salary
                       FROM instructor
                      WHERE dept_id = 'PHYS');
```

Find names of all instructors whose salary is greater than the salary of **all** instructors in the Physics and Astronomy department

```
SELECT instr_name
  FROM instructor
 WHERE salary > ALL(SELECT salary
                      FROM instructor
                     WHERE dept_id = 'PHYS');
```

Subqueries in the From Clause

SQL also 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 £31,000.

```
SELECT dept_id, avg_salary
  FROM (SELECT dept_id, avg (salary) as avg_salary
          FROM instructor
      GROUP BY dept_id)
 WHERE avg_salary > 31000;
```

Note that we do not need to use the HAVING clause

Scalar subquery is one which is used where a single value is expected

```
SELECT dept_id,
       (SELECT COUNT(*)
          FROM instructor
         WHERE department.dept_id = instructor.dept_id)
    AS num_instructors
  FROM department;
```

Runtime error if subquery returns more than one result tuple

- **Join condition** – defines **which tuples** in the two relations match, and **what attributes** are present in the result of the join.
- **Join type** – defines **how** tuples in each relation that do not match any tuple in the other relation (based on the join condition) are treated.

Course

course_id	*title*	*dept_name*	*credits*
MT5753	Statistical Modelling	Statistics	20
CS5012	Language & Computation	Comp.Sci	15
CS5010	Artificial Intelligence	Comp.Sci	15

Prereq

course_id	*prereq_id*
CS5012	CS5010
MT5753	MT5700
IS5120	IS5102

Observe:

Prereq information missing for CS5010

Course information missing for IS5120

```
SELECT *
  FROM course NATURAL JOIN prereq
```

course_id	*title*	*dept_name*	*credits*	*prereq_id*
MT5753	Statistical Modelling	Statistics	20	MT5700
CS5012	Language & Computation	Comp.Sci	15	CS5010

Outer Join

- An **extension** of the join operation that avoids loss of information.
- Computes the join and then adds tuples from one relation that does not match tuples in the other relation to the result of the join.
- Uses `NULL` values.

```
SELECT *
  FROM course NATURAL LEFT OUTER JOIN prereq
```

course_id	*title*	*dept_name*	*credits*	*prereq_id*
MT5753	Statistical Modelling	Statistics	20	MT5700
CS5012	Language & Computation	Comp.Sci	15	CS5010
CS5010	Artificial Intelligence	Comp.Sci	15	NULL

```
SELECT *
  FROM course NATURAL RIGHT OUTER JOIN prereq
```

course_id	*title*	*dept_name*	*credits*	*prereq_id*
MT5753	Statistical Modelling	Statistics	20	MT5700
CS5012	Language & Computation	Comp. Sci	15	CS5010
IS5120	NULL	NULL	NULL	IS5102

Note: Right Outer Join is not supported by SQLite

```
SELECT *
  FROM course NATURAL FULL OUTER JOIN prereq
```

course_id	*title*	*dept_name*	*credits*	*prereq_id*
MT5753	Statistical Modelling	Statistics	20	MT5700
CS5012	Language & Computation	Comp.Sci	15	CS5010
CS5010	Artificial Intelligence	Comp.Sci	15	NULL
IS5120	NULL	NULL	NULL	IS5102

Note: Full Outer Join is not supported by SQLite

- Some SQL implementations (but not SQLite) support **Discretionary Access Control**
  - User given access rights on database objects
  - Users gain certain privileges when they create an object and can pass these rights on at their discretion
- Mechanisms based on authorisation identifiers and ownership

Levels of authorization on parts of the database:

- **Read** – allows reading, but not modification of data.
- **Insert** – allows insertion of new data, but not modification of existing data.
- **Update** – allows modification, but not deletion of data.
- **Delete** – allows deletion of data.

- The GRANT statement is used to confer authorization

```
GRANT <privilege_list>
   ON <relation name or view name>
   TO <user_list>
```

- <user_list> can be one of:
  - a user-id
  - public, which allows all valid users the privilege granted
  - a role (more on this later)
- Granting a privilege on a view does not imply granting any privileges on the underlying relations.
- The grantor of the privilege must already hold the privilege on the specified item (or be the database administrator).

Privileges in SQL

- `SELECT`: allows read access to relation, or the ability to query using the view
  Example: grant users U1, U2, and U3 the select authorization on the instructor relation:

```
GRANT SELECT
   ON instructor
   TO U1, U2, U3
```

- `INSERT`: the ability to insert tuples
- `UPDATE`: the ability to update using the SQL update statement
- `DELETE`: the ability to delete tuples.
- `ALL PRIVILEGES`: used as a short form for all the allowable privileges

SQL Authorisation Example

Give the user with authorisation identifier `Manager` all privileges on the `Staff` table and allow their delegation

```
GRANT ALL PRIVILEGES
  ON Staff
  TO Manager
  WITH GRANT OPTION
```

SQL Authorisation Example 2

Give users `Personnel` and `Director` the privileges of `SELECT` and `UPDATE` on the column `salary` of the `Staff` table.

```
GRANT SELECT, UPDATE (salary)
   ON Staff
   TO Personnel, Director
```

The REVOKE statement is used to revoke authorization.

```
REVOKE <privilege_list>
    ON <relation name or view name>
  FROM <user_list>
```

Example:

```
REVOKE SELECT
    ON branch
  FROM U1, U2, U3
```

<privilege-list> may be ALL to revoke all privileges the revokee may hold.

- Creating Roles

```
CREATE ROLE instructor;
GRANT instructor TO Alexander;
```

- Privileges can be granted to roles:

```
GRANT SELECT ON takes_course TO instructor;
```

- Roles can be granted to users, as well as to other roles

```
CREATE ROLE teaching_assistant;
GRANT teaching_assistant TO instructor;
```

instructor inherits all privileges of teaching_assistant

- Chain of Roles

```
CREATE ROLE head;
GRANT instructor TO head;
GRANT head TO Ian;
```

- Chapter 4 and 5, Database System Concepts, 6th Ed. Silberschatz, Korth and Sudarshan
- Chapter 7, Database Systems, Connolly Begg

Useful URLs:
http://www.w3schools.com/sql/default.asp