

---

Introduction to Databases, Fall 2003  
IT University of Copenhagen

**Lecture 6, part II: More on SQL**

September 30, 2003

Lecturer: Rasmus Pagh

---

# — Today's lecture - part II —

---

- The SQL environment: Schemas, catalogs, and privileges.
- Subqueries in SQL
- Views in SQL

# — What you should remember from previously —

In this lecture I will assume that you remember:

- The SQL concepts needed to solve the first hand-in:
  - Projection and selection using `SELECT-FROM-WHERE`.
  - `SELECT-FROM-WHERE` involving multiple relations.

---

**Next: The SQL environment: Schemas, catalogs, and privileges.**

---

# — Organization of relations —

---

Relations in an RDBMS are often organized in a structured way:

- Relations are organized into **schemas**.
- Schemas are organized into **catalogs**.
- Catalogs are organized into **clusters**.

SQL only defines standard syntax for defining schemas:

```
CREATE SCHEMA <name of schema>  
<list of schema elements, e.g., table declarations>
```

In Oracle, schemas correspond to users, and a slightly different syntax is used. Catalogs and clusters are not used as described above.

## — Referring to schema elements —

---

We can refer to an element of a given schema using the syntax:

`<name of schema>.<name of schema element>`

If no schema name is given, it is assumed that element names refer to the **current schema**.

### Examples:

- If the schema `pagh` has a relation `Movie`, it can be referred to as `pagh.Movie`.
- In Oracle, the user `pagh` does not need to write “`pagh.`” in front of `Movie`, because `pagh` is the current schema.

## — Authorization in SQL —

---

Because databases often have many users, not all of which are allowed to do any database operation, SQL has an **authorization** system.

Every user (called a **module** in case the user is a program) has certain rights, or **privileges**, to access and modify database elements. The basic privileges are:

SELECT, INSERT, DELETE, UPDATE

It is possible to have privileges for certain attributes in a relation, e.g., a secretary might have the UPDATE(address, city) privilege for relation with customer information.

# — Granting privileges —

---

## Basics of managing privileges:

- If a user defines a new schema, she has all possible privileges for the tables (and other database elements) of that schema.
- Users may **grant** (“copy”) privileges of their own to other users.
- Being able to grant a privilege is a special privilege in itself that can be passed on.

## Syntax for granting privileges:

```
GRANT <privilege list> ON <database element>  
TO <user list> [WITH GRANT OPTION]
```

## — Revoking privileges —

---

Granted privileges can be withdrawn (or **revoked**) by a user at any time.

Basics of revoking privileges:

- Privileges given without the GRANT OPTION can simply be removed.
- Otherwise we would like to revoke any privilege in the database that is *only possible because of the privilege that was revoked*.
- What happens when revoking is *independent* on the order in which privileges were given.

Syntax for revoking privileges:

```
REVOKE <privilege list> ON <database element>  
FROM <user list> CASCADE
```

## — Grant diagrams

---

To control revoking, the DBMS maintains a **grant diagram** (also called an **authorization graph**) with:

- One node for each privilege of each user.
- Arrows showing which privileges and users are behind each privilege.

Grant privileges due to ownership of a database element are indicated by \*\*, and other grant privileges are indicated by \*.

[Figure 8.26 shown on slide]

[Figure 8.27 shown on slide]

[Figure 8.28 shown on slide]

[Figure 8.29 shown on slide]

## — Problem session (5 minutes) —

---

Consider the grant diagram of Fig. 8.26. Which privileges does Sisko have after the changes caused by user janeway running the following three SQL commands?

```
REVOKE SELECT ON Studio FROM picard CASCADE;  
REVOKE INSERT ON Studio FROM kirk CASCADE;  
GRANT INSERT(name) ON Studio TO kirk WITH GRANT OPTION;
```

What if the two last commands were swapped?

---

**Next: Subqueries in SQL**

---

# — Subqueries

---

Until now, you have seen SQL queries of the form

```
SELECT <list of attributes>  
FROM <list of relations>  
WHERE <condition>
```

What we haven't used is that:

- In any place where a relation is allowed, we may put an SQL query (a **subquery**) computing a relation.
- In any place where an atomic value is allowed, we may put an SQL subquery computing a relation with one attribute and one tuple.

## — Subqueries in FROM clauses —

---

Instead of just relations, we may use SQL queries in the FROM clause of SELECT-FROM-WHERE.

If we need a name for referring to the relation computed by the subquery, a tuple variable is used.

Subqueries should always be surrounded by parentheses.

## — Subqueries producing scalar values —

When a query produces a relation with one attribute and one tuple, it can be used in any place where we can put an atomic (or **scalar**) value.

### **Semantics:**

In places where an atomic value is expected, SQL regards a relation instance containing one atomic value  $x$  to be the same as the value  $x$ .

If such a subquery does not result in exactly one tuple, it is a **run-time error**, and the SQL query cannot be completed.

## — Subqueries in conditions —

---

One common use of subqueries is in the `WHERE` part of `SELECT-FROM-WHERE`. There are several operators in SQL that apply to a relation  $R$  and produce a boolean result:

- `EXISTS R` is true if and only if  $R$  is not empty.
- `s IN R` is true if and only if  $s$  is a tuple in  $R$ .

If  $R$  is **unary** (has just one attribute):

- `s > ALL R` is true if and only if  $s$  is greater than *all* values in  $R$ .
- `s > ANY R` is true if and only if  $s$  is greater than *some* value in  $R$ .

...and similarly for other comparison operators (`<`, `>=`, `<=`, `<>`).

## — Correlated subqueries —

---

Sometimes a subquery depends on (is **correlated** with) tuple variables/reasons of the surrounding SELECT-FROM-WHERE.

### **Semantics:**

The query is evaluated once for each **binding** of tuple variables in the surrounding SELECT-FROM-WHEREs.

### **Scoping rule:**

In case several tuple variables/reasons have the same name  $x$ , an occurrence of  $x$  refers to the *closest* such tuple variable/reason.

## — Problem session (5 minutes) —

---

What does the below SQL query compute?

```
SELECT title, year
FROM Movie
WHERE EXISTS (SELECT *
              FROM Movie M2
              WHERE Movie.year = year + 1
              AND EXISTS (SELECT *
                        FROM Movie M3
                        WHERE M2.year = year + 1));
```

**Tip:** Read from inside out.

---

**Next: Views in SQL**

---

# Views

---

**Views** are queries that have been given a name.

Syntax for declaring a view:

```
CREATE VIEW <name of view> AS <SQL query>
```

We may use the name of a view in SQL expressions, as a *shorthand* for the corresponding query.

## — Properties of views —

---

- Views are elements of the database schema, just like relation schemas.
- Privileges to access a view are handled just like privileges for relations.
- The privileges to perform the query must be held by the user who *defines* the view, but not necessarily by users accessing the view.
- Sufficiently simple views can be modified, meaning that the the modifications are passed on to the underlying relations.

## — Most important points in this lecture —

As a minimum, you should after this week:

- Understand the mechanism for granting and revoking privileges in SQL.
- Be able to define and use views in SQL.
- Be able to understand and form SQL expressions using several levels of subqueries.

## Next time

---

Next week we will study **relational algebra**, the theoretical basis of SQL queries.

- Relational algebra on sets.
- Relational algebra on bags.
- ... and the relation to SQL queries.