Advanced SQL – Views, Security, and Transactions

UBCO Master of Data Science – DATA 540

Views

A ***view*** is a named query that is defined in the database.

- To the user, a view appears just like any other table and can be present in any SQL query where a table is present.

A view may either be:

- *virtual* - produced by a SQL query on demand.
- *materialized* - the view is stored as a derived table that is updated when the tables that it refers to are updated.

Creating Views

Views are created using the **`CREATE VIEW`** command:

```
CREATE VIEW viewName [(col1,col2,...,colN)]
AS selectStatement
```

Notes:

- Select statement can be any SQL query and is called the ***defining query*** of the view.
- It is possible to rename the columns when defining the view, otherwise they default to the names produced by the query.

Views Example

Create a view that has only the employees of department `'D2'`:

```
CREATE VIEW empD2
AS SELECT * FROM emp WHERE dno = 'D2';
```

Create a view that only shows the employee number, title, and name:

```
CREATE VIEW staff (Number, Name, Title)
AS SELECT eno, ename, title FROM emp;
```

- The first example is a *horizontal view* because it only contains a subset of the rows.
- The second example is a *vertical view* because it only contains a subset of the columns.

Removing Views

Views are removed using the `DROP VIEW` command:

```
DROP VIEW viewName [RESTRICT|CASCADE]
```

Notes:

- `RESTRICT` will not delete a view if other views are dependent on it.
- `CASCADE` deletes the view and all dependent views.

Advantages and Disadvantages of Views

Advantages:

- Data independence - allows base relations to change without affecting users.
- Security - views can be used to limit access to certain data to certain users.
- Easier querying - using views can often reduce the complexity of some queries.
- Convenience/Customization - users only see the parts of the database that they have interest and access to.

Disadvantages:

- Updatable views are not always supported and are restrictive.
- Performance - views add increased overhead: during query parsing and especially if they are materialized.

Views Practice Questions

Creates views that:

- 1) Show only employees that have title `'EE'`.
- 2) List only projects that have someone working on them.
- 3) List only the employees that manage another employee.
- 4) List the department number, name, and average employee salary in the department.
- 5) Show all employees but only lists their number, name, and title.
- 6) Show all employee and `workson` information for employee `'E1'`.

SQL Security

Security in SQL is based on ***authorization identifiers***, ***ownership***, and ***privileges***.

An authorization identifier (or user id) is associated with each user. Normally, a password is also associated with a authorization identifier. Every SQL statement performed by the DBMS is executed on behalf of some user.

The authorization identifier is used to determine which database objects the user has access to.

Whenever a user creates an object, the user is the owner of the object and initially is the only one with the ability to access it.

SQL Privileges

Privileges give users the right to perform operations on database objects. The set of privileges are:

- `SELECT` - the user can retrieve data from table
- `INSERT` - the user can insert data into table
- `UPDATE` - the user can modify data in the table
- `DELETE` - the user can delete data (rows) from the table
- `REFERENCES` - the ability to reference columns of a named table in integrity constraints
- `USAGE` - the ability to use domains, character sets, and translations (i.e. other database objects besides tables)

Notes:

- `INSERT` and `UPDATE` can be restricted to certain columns.
- When a user creates a table, they become the owner and have full privileges on the table.

SQL GRANT Command

The **GRANT** command gives privileges on database objects to users.

```
GRANT {privilegeList | ALL [PRIVILEGES]}
   ON ObjectName
   TO {AuthorizationIdList | PUBLIC}
   [WITH GRANT OPTION]
```

The privilege list is one or more of the following privileges:

```
SELECT [(columnName [,...]]
DELETE
INSERT [(columnName [,...]]
UPDATE [(columnName [,...]]
REFERENCES [(columnName [,...]]
USAGE
```

SQL GRANT Command (2)

The `ALL PRIVILEGES` keyword grants all privileges to the user except the ability to grant privileges to other users.

The `PUBLIC` keyword grants access to all users (present and future) of the database.

The `WITH GRANT OPTION` allows users to grant privileges to other users. A user can only grant privileges that they themselves hold.

GRANT Examples

Allow all users to query the `dept` relation:

```
GRANT SELECT ON dept TO PUBLIC;
```

Only allow users `Manager` and `Director` to access and change `Salary` in `emp`:

```
GRANT SELECT, UPDATE(salary) ON emp TO Manager,Director;
```

Allow the `Director` full access to `proj` and the ability to grant privileges to other users:

```
GRANT ALL PRIVILEGES ON proj TO Director
    WITH GRANT OPTION;
```

Required Privileges Example

What privileges are required for this statement:

```
UPDATE emp SET salary=salary*1.1
WHERE eno IN (
    SELECT eno FROM workson WHERE hours > 30)
```

Answer:

```
SELECT on emp

UPDATE(salary) on emp

SELECT on workson
```

Required Privileges Question

Question: What are the required privileges for this statement?

```
DELETE FROM dept WHERE dno NOT IN
    (SELECT dno FROM workson)
```

A) DELETE, SELECT

B) DELETE on dept, DELETE on workson

C) DELETE on dept, SELECT on workson

D) DELETE on dept

E) DELETE on dept, SELECT on workson, SELECT on dept

Required Privileges Question (2)

Question: What are the required privileges for this statement?

```
INSERT INTO workson (eno,pno) VALUES ('E5','P5')
```

A) `INSERT on workson`

B) `INSERT`

C) `INSERT, SELECT`

D) `INSERT on workson, UPDATE on workson`

E) `none`

GRANT Question

Question: **True or False:** A user `WITH GRANT OPTION` can grant a privilege that they do not hold themselves.

A) true

B) false

GRANT Question (2)

Question: **True or False:** A users may be granted the same privilege on the same object from multiple users.

A) true

B) false

SQL REVOKE Command

The REVOKE command is used to remove privileges on database objects from users.

```
REVOKE [GRANT OPTION FOR]{privilegeList | ALL [PRIVILEGES]}
   ON ObjectName
   FROM {AuthorizationIdList | PUBLIC} {RESTRICT|CASCADE}
```

Notes:

- ALL PRIVILEGES removes all privileges on object.
- GRANT OPTION FOR removes the ability for users to pass privileges on to other users (not the privileges themselves).
- RESTRICT - REVOKE fails if privilege has been passed to other users.
- CASCADE - removes any privileges and objects created using the revoked privileges including those passed on to others.

SQL `REVOKE` Command (2)

Privileges are granted to an object to a user *from* a specific user. If a user then revokes their granting of privileges, that only applies to that user.

Example:

- User A grants all privileges to user B on table T.
- User B grants all privileges to user C on table T.
- User E grants `SELECT` privilege to user C on table T.
- User C grants all privileges to user D on table T.
- User A revokes all privileges on table T from B (using `cascade`).
- This causes all privileges to be removed for user C as well, except the `SELECT` privilege which was granted by user E.
- User D now has only the `SELECT` privilege as well.

`REVOKE` Examples

Do not allow public (general) users to query the `dept` relation:

```
REVOKE SELECT ON dept FROM PUBLIC;
```

Remove all privileges from user `Joe` on `emp` table:

```
REVOKE ALL PRIVILEGES ON emp FROM Joe;
```

REVOKE Question

Question: User *A* executes:

```
GRANT SELECT, UPDATE ON T TO B WITH GRANT OPTION;
```

then User B executes:

```
GRANT SELECT, UPDATE ON T TO C;
```

then User A executes:

```
REVOKE GRANT OPTION FOR SELECT, UPDATE ON T FROM B;
```

True or False: User *C* loses `SELECT` on *T*.

A) true

B) false

Security Practice Questions

Use `GRANT` and `REVOKE` to provide the desired access for each of these cases. You may also need views. Assume that you are the DBA who has full privileges (owner) of all objects currently in the database.

1) Allow all users read-only access to the `dept` relation.

2) Allow the user `Accounting` to read `emp` records and update the `salary` of `emp`.

3) Allow user `Davis` all privileges on a view containing the employees in Dept `D2` including the ability to grant privileges.

4) Allow user `Smith` to only see their employee record (`eno='E3'`) and `WorksOn` information. (need a view)

5) `Davis` is replaced as head of Dept. `D2`, so only leave him query access to employees in `D2`.

Transaction Management Overview

The database system must ensure that the data is always ***consistent***.
Two challenges in preserving consistency:

- 1) Must handle ***failures*** of various kinds (hardware, crashes).
- 2) Must support ***concurrent execution*** of multiple transactions and guarantee that concurrency does not cause inconsistency.

A ***transaction*** is an ***atomic*** program that executes on the database and preserves the consistency of the database.

- The input to a transaction is a consistent database AND the output of the transaction must also be a consistent database.
- A transaction must execute completely or not at all.

Transaction Management - Motivating Example

Consider a person who wants to transfer $50 from a savings account with balance $1000 to a checking account with current balance $250.

- 1) At the ATM, the person starts the process by telling the bank to remove $50 from the savings account.
- 2) The $50 is removed from the savings account by the bank.
- 3) Before the customer can tell the ATM to deposit the $50 in the checking account, the ATM "crashes."

Where has the $50 gone?

It is lost if the ATM did not support transactions!
The customer wanted the withdraw and deposit to both happen in one step, or neither action to happen.

ACID Properties

To preserve integrity, transactions have the following properties:

- ***Atomicity*** **-** Either all operations of the transaction are properly reflected in the database or none are.
- ***Consistency*** **-** Execution of a transaction in isolation preserves the consistency of the database.
- ***Isolation*** **-** Although multiple transactions may execute concurrently, each transaction must be unaware of other concurrently executing transactions.
- ***Durability*** **-** After a transaction successfully completes, the changes it has made to the database persist, even if there are system failures.

ACID Properties

Question: Two transactions running at the same time can see each other's updates. What ACID property is violated?

A) atomicity

B) consistency

C) isolation

D) durability

E) none of them

Transaction Definition in SQL

In SQL, a transaction begins implicitly.

A transaction in SQL ends by:

- **Commit** accepts updates of current transaction.
- **Rollback** aborts current transaction and discards its updates. Failures may also cause a transaction to be aborted.

In programming with databases you can often set the connection to **auto-commit** which means a commit is done after every statement.

Recursive Queries in SQL

General form:

```
WITH RECURSIVE tableName(attr1,attr2,..attrN) AS
   <SELECT query that defines recursive relation>
<SELECT query that uses recursive relation>
```

Example: Return all employees supervised by 'J. Jones'.

```
WITH RECURSIVE supervises(supId,empId) AS
   (SELECT supereno, eno FROM emp)
   UNION
   (SELECT S1.supId, S2.empId
   FROM supervises S1, supervises S2
   WHERE S1.empId = S2.supId)
SELECT E1.ename FROM supervises, emp AS E, emp AS E2
WHERE supervises.supId = E2.eno and E2.ename = 'J. Jones'
      and supervises.empId = E1.ename;
```

Conclusion

Views define virtual relations over base relations. This is useful to:

- reduce query complexity
- improve performance (especially if view is materialized)
- provide different user views of the database
- allow security to be enforced on subsets of the schema

SQL security is enforced using `GRANT`/`REVOKE`.

Relational databases support transactions and the ACID properties.

`WITH RECURSIVE` syntax is used to write recursive SQL queries.

Objectives

Views:

- define views using `CREATE VIEW` from a high-level description
- list advantages and disadvantages of views

Security:

- be able to use `GRANT`/`REVOKE` syntax
- list the types of privileges and know when to use them
- given a SQL command, explain what privileges are required for it to execute
- explain how views are useful for security

Transactions:

- Define: transaction
- List ACID properties.
- Be aware of WITH RECURSIVE for recursive SQL queries.

THE UNIVERSITY OF BRITISH COLUMBIA