Minggu 7
SQL: Views and Access Control

Learning Outcomes

Pada akhir pertemuan ini, diharapkan Mahasiswa dapat Mendemonstrasikan penggunaan views dan access control statements in SQL Standard (C3)

Outline Materi

- Purpose of views.
- How to create and delete views using SQL.
- How the DBMS performs operations on views.
- Under what conditions views are updatable.
- Advantages and disadvantages of views.
- How the ISO transaction model works.
- How to use the GRANT and REVOKE statements as a level of security.

Views

View

Dynamic result of one or more relational operations operating on base relations to produce another relation.

- **Virtual relation that does not necessarily actually exist in the database but is produced upon request, at time of request.**

Views

- Contents of a view are defined as a query on one or more base relations.
- With view resolution, any operations on view are automatically translated into operations on relations from which it is derived.
- With view materialization, the view is stored as a temporary table, which is maintained as the underlying base tables are updated.

SQL - CREATE VIEW

CREATE VIEW ViewName [(newColumnName [,...])]

AS subselect

[WITH [CASCADED | LOCAL] CHECK OPTION]

- **Can assign a name to each column in view.**
- **If list of column names is specified, it must have same number of items as number of columns produced by *subselect*.**
- **If omitted, each column takes name of corresponding column in *subselect*.**

SQL - CREATE VIEW

- List must be specified if there is any ambiguity in a column name.
- The *subselect* is known as the defining query.
- WITH CHECK OPTION ensures that if a row fails to satisfy WHERE clause of defining query, it is not added to underlying base table.
- Need SELECT privilege on all tables referenced in subselect and USAGE privilege on any domains used in referenced columns.

Example 6.3 - Create Horizontal View

Create view so that manager at branch B003 can only see details for staff who work in his or her office.

```
CREATE VIEW Manager3Staff
AS    SELECT *
      FROM Staff
      WHERE branchNo = 'B003';
```

Table 6.3 Data for view Manager3Staff.

staffNo	fName	lName	position	sex	DOB	salary	branchNo
SG37	Ann	Beech	Assistant	F	10-Nov-60	12000.00	B003
SG14	David	Ford	Supervisor	M	24-Mar-58	18000.00	B003
SG5	Susan	Brand	Manager	F	3-Jun-40	24000.00	B003

Example 6.4 - Create Vertical View

Create view of staff details at branch B003 excluding salaries.

```
CREATE VIEW Staff3
AS SELECT staffNo, fName, lName, position, sex
   FROM Staff
   WHERE branchNo = 'B003';
```

Table 6.4 Data for view Staff3.

staffNo	fName	lName	position	sex
SG37	Ann	Beech	Assistant	F
SG14	David	Ford	Supervisor	M
SG5	Susan	Brand	Manager	F

Example 6.5 - Grouped and Joined Views

Create view of staff who manage properties for rent, including branch number they work at, staff number, and number of properties they manage.

```
CREATE VIEW StaffPropCnt (branchNo, staffNo, cnt)
AS SELECT s.branchNo, s.staffNo, COUNT(*)
    FROM Staff s, PropertyForRent p
    WHERE s.staffNo = p.staffNo
    GROUP BY s.branchNo, s.staffNo;
```

Example 6.3 - Grouped and Joined Views

Table 6.5 Data for view StaffPropCnt.

branchNo	staffNo	cnt
B003	SG14	1
B003	SG37	2
B005	SL41	1
B007	SA9	1

SQL - DROP VIEW

DROP VIEW ViewName [RESTRICT | CASCADE]

- **Causes definition of view to be deleted from database.**
- **For example:**

DROP VIEW Manager3Staff;

SQL - DROP VIEW

- **With CASCADE, all related dependent objects are deleted; i.e. any views defined on view being dropped.**
- **With RESTRICT (default), if any other objects depend for their existence on continued existence of view being dropped, command is rejected.**

View Resolution

Count number of properties managed by each member at branch B003.

```
SELECT staffNo, cnt
FROM StaffPropCnt
WHERE branchNo = 'B003'
ORDER BY staffNo;
```

View Resolution

(a) View column names in SELECT list are translated into their corresponding column names in the defining query:

```
SELECT s.staffNo As staffNo, COUNT(*) As
  cnt
```

(b) View names in FROM are replaced with corresponding FROM lists of defining query:

```
FROM Staff s, PropertyForRent p
```

View Resolution

(c) WHERE from user query is combined with WHERE of defining query using AND:

```
WHERE s.staffNo = p.staffNo AND
  branchNo = 'B003'
```

(d) GROUP BY and HAVING clauses copied from defining query:

```
GROUP BY s.branchNo, s.staffNo
```

(e) ORDER BY copied from query with view column name translated into defining query column name

```
ORDER BY s.staffNo
```

View Resolution

(f) Final merged query is now executed to produce the result:

```
SELECT s.staffNo, COUNT(*)
FROM staff s, PropertyForRent p
WHERE s.staffNo = p.staffNo AND
                  branchNo = 'B003'
GROUP BY s.branchNo, s.staffNo
ORDER BY s.staffNo;
```

Restrictions on Views

SQL imposes several restrictions on creation and use of views.

(a) If column in view is based on an aggregate function:

- Column may appear only in SELECT and ORDER BY clauses of queries that access view.
- Column may not be used in WHERE nor be an argument to an aggregate function in any query based on view.

Restrictions on Views

- For example, following query would fail:

```
SELECT COUNT(cnt)
FROM StaffPropCnt;
```

- Similarly, following query would also fail:

```
SELECT *
FROM StaffPropCnt
WHERE cnt > 2;
```

Restrictions on Views

(b) Grouped view may never be joined with a base table or a view.

- **For example, StaffPropCnt view is a grouped view, so any attempt to join this view with another table or view fails.**

View Updatability

- **All updates to base table reflected in all views that encompass base table.**
- **Similarly, may expect that if view is updated then base table(s) will reflect change.**

View Updatability

- However, consider again view StaffPropCnt.
- If we tried to insert record showing that at branch B003, SG5 manages 2 properties:

```
INSERT INTO StaffPropCnt
VALUES ('B003', 'SG5', 2);
```

- Have to insert 2 records into PropertyForRent showing which properties SG5 manages. However, do not know which properties they are; i.e. do not know primary keys!

View Updatability

- **If change definition of view and replace count with actual property numbers:**

```
CREATE VIEW StaffPropList (branchNo,
                           staffNo, propertyNo)
AS SELECT s.branchNo, s.staffNo,
 p.propertyNo
    FROM Staff s, PropertyForRent p
    WHERE s.staffNo = p.staffNo;
```

View Updatability

- **Now try to insert the record:**

```
INSERT INTO StaffPropList
VALUES ('B003', 'SG5', 'PG19');
```

- **Still problem, because in PropertyForRent all columns except postcode/staffNo are not allowed nulls.**
- **However, have no way of giving remaining non-null columns values.**

View Updatability

- **ISO specifies the views that must be updatable in system that conforms to standard.**
- **A view is updatable if and only if:**

  - DISTINCT is not specified.

  - Every element in SELECT list of defining query is a column name and no column appears more than once.

  - FROM clause specifies only one table, excluding any views based on a join, union, intersection or difference.

  - No nested SELECT referencing outer table.

  - No GROUP BY or HAVING clause.

  - Also, every row added through view must not violate integrity constraints of base table.

Updatable View

For view to be updatable, DBMS must be able to trace any row or column back to its row or column in the source table.

WITH CHECK OPTION

- **Rows exist in a view because they satisfy WHERE condition of defining query.**
- **If a row changes and no longer satisfies condition, it disappears from the view.**
- **New rows appear within view when insert/update on view cause them to satisfy WHERE condition.**
- **Rows that enter or leave a view are called *migrating rows*.**
- **WITH CHECK OPTION prohibits a row migrating out of the view.**

WITH CHECK OPTION

- **LOCAL/CASCADED apply to view hierarchies.**
- **With LOCAL, any row insert/update on view and any view directly or indirectly defined on this view must not cause row to disappear from view unless row also disappears from derived view/table.**
- **With CASCADED (default), any row insert/ update on this view and on any view directly or indirectly defined on this view must not cause row to disappear from the view.**

Example 6.6 - WITH CHECK OPTION

```
CREATE VIEW Manager3Staff
AS      SELECT *
        FROM Staff
        WHERE branchNo = 'B003'
WITH CHECK OPTION;
```

- Cannot update branch number of row B003 to B002 as this would cause row to migrate from view.
- Also cannot insert a row into view with a branch number that does not equal B003.

Example 6.6 - WITH CHECK OPTION

- If Manager3Staff is defined not on Staff directly but on another view of Staff:

```
CREATE VIEW LowSalary
 AS SELECT * FROM Staff WHERE salary > 9000;
CREATE VIEW HighSalary
 AS SELECT * FROM LowSalary
      WHERE salary > 10000
 WITH LOCAL CHECK OPTION;
CREATE VIEW Manager3Staff
 AS SELECT * FROM HighSalary
      WHERE branchNo = 'B003';
```

Example 6.6 - WITH CHECK OPTION

```
UPDATE Manager3Staff
SET salary = 9500
WHERE staffNo = 'SG37';
```

- Update would fail: although update would cause row to disappear from HighSalary, row would not disappear from LowSalary.
- However, if update tried to set salary to 8000, update would succeed as row would no longer be part of LowSalary.

Example 6.6 - WITH CHECK OPTION

- If HighSalary had specified WITH CASCADED CHECK OPTION, setting salary to 9500 or 8000 would be rejected because row would disappear from HighSalary.
- To prevent anomalies like this, each view should be created using WITH CASCADED CHECK OPTION.

Advantages of Views

- Data independence
- Currency
- Improved security
- Reduced complexity
- Convenience
- Customization
- Data integrity

Disadvantages of Views

- **Update restriction**
- **Structure restriction**
- **Performance**

View Materialization

- View resolution mechanism may be slow, particularly if view is accessed frequently.
- View materialization stores view as temporary table when view is first queried.
- Thereafter, queries based on materialized view can be faster than recomputing view each time.
- Difficulty is maintaining the currency of view while base tables(s) are being updated.

View Maintenance

- **View maintenance aims to apply only those changes necessary to keep view current.**
- **Consider following view:**

```
CREATE VIEW StaffPropRent(staffNo)
AS SELECT DISTINCT staffNo
     FROM PropertyForRent
     WHERE branchNo = 'B003' AND
           rent > 400;
```

Table 6.8 Data for view StaffPropRent.

staffNo
SG37
SG14

View Materialization

- If insert row into PropertyForRent with rent £400 then view would be unchanged.
- If insert row for property PG24 at branch B003 with staffNo = SG19 and rent = 550, then row would appear in materialized view.
- If insert row for property PG54 at branch B003 with staffNo = SG37 and rent = 450, then no new row would need to be added to materialized view.
- If delete property PG24, row should be deleted from materialized view.
- If delete property PG54, then row for PG37 should not be deleted (because of existing property PG21).

Transactions

- **SQL defines transaction model based on COMMIT and ROLLBACK.**
- **Transaction is logical unit of work with one or more SQL statements guaranteed to be atomic with respect to recovery.**
- **An SQL transaction automatically begins with a *transaction-initiating* SQL statement (e.g., SELECT, INSERT).**
- **Changes made by transaction are not visible to other concurrently executing transactions until transaction completes.**

Transactions

- **Transaction can complete in one of four ways:**

  - COMMIT ends transaction successfully, making changes permanent.

  - ROLLBACK aborts transaction, backing out any changes made by transaction.

  - For programmatic SQL, successful program termination ends final transaction successfully, even if COMMIT has not been executed.

  - For programmatic SQL, abnormal program end aborts transaction.

Transactions

- New transaction starts with next transaction-initiating statement.
- SQL transactions cannot be nested.
- SET TRANSACTION configures transaction:

```
SET TRANSACTION
  [READ ONLY | READ WRITE] |
  [ISOLATION LEVEL READ UNCOMMITTED |
  READ COMMITTED|REPEATABLE READ
  |SERIALIZABLE ]
```

Immediate and Deferred Integrity Constraints

- Do not always want constraints to be checked immediately, but instead at transaction commit.
- Constraint may be defined as INITIALLY IMMEDIATE or INITIALLY DEFERRED, indicating mode the constraint assumes at start of each transaction.
- In former case, also possible to specify whether mode can be changed subsequently using qualifier [NOT] DEFERRABLE.
- Default mode is INITIALLY IMMEDIATE.

Immediate and Deferred Integrity Constraints

- **SET CONSTRAINTS statement used to set mode for specified constraints for current transaction:**

  SET CONSTRAINTS

  {ALL | constraintName [, . . .]}

  {DEFERRED ¦ IMMEDIATE}

Access Control - Authorization Identifiers and Ownership

- Authorization identifier is normal SQL identifier used to establish identity of a user. Usually has an associated password.
- Used to determine which objects user may reference and what operations may be performed on those objects.
- Each object created in SQL has an owner, as defined in AUTHORIZATION clause of schema to which object belongs.
- Owner is only person who may know about it.

Privileges

- **Actions user permitted to carry out on given base table or view:**

SELECT Retrieve data from a table.

INSERT Insert new rows into a table.

UPDATE Modify rows of data in a table.

DELETE Delete rows of data from a table.

REFERENCES Reference columns of named table in integrity constraints.

USAGE Use domains, collations, character sets, and translations.

Privileges

- Can restrict INSERT/UPDATE/REFERENCES to named columns.
- Owner of table must grant other users the necessary privileges using GRANT statement.
- To create view, user must have SELECT privilege on all tables that make up view and REFERENCES privilege on the named columns.

GRANT

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

- ***PrivilegeList*** **consists of one or more of above privileges separated by commas.**
- **ALL PRIVILEGES grants all privileges to a user.**

GRANT

- **PUBLIC allows access to be granted to all present and future authorized users.**
- ***ObjectName* can be a base table, view, domain, character set, collation or translation.**
- **WITH GRANT OPTION allows privileges to be passed on.**

Example 6.7/8 - GRANT

Give Manager full privileges to Staff table.

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

Give users Personnel and Director SELECT and UPDATE on column salary of Staff.

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

Example 6.9 - GRANT Specific Privileges to PUBLIC

Give all users SELECT on Branch table.

```
GRANT SELECT
ON Branch
TO PUBLIC;
```

REVOKE

- **REVOKE takes away privileges granted with GRANT.**

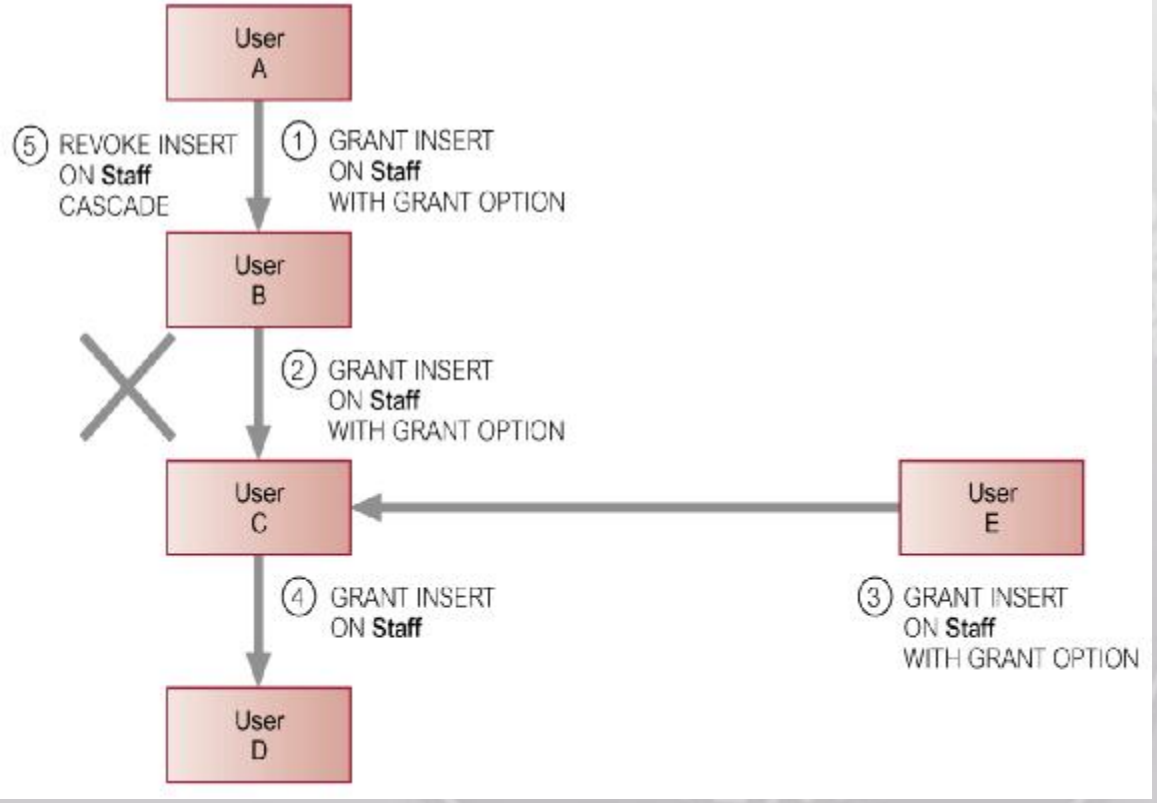
```
REVOKE [GRANT OPTION FOR]
  {PrivilegeList | ALL PRIVILEGES}
ON ObjectName
FROM {AuthorizationIdList | PUBLIC}
   [RESTRICT | CASCADE]
```

- **ALL PRIVILEGES refers to all privileges granted to a user by user revoking privileges.**

REVOKE

- **GRANT OPTION FOR allows privileges passed on via WITH GRANT OPTION of GRANT to be revoked separately from the privileges themselves.**
- **REVOKE fails if it results in an abandoned object, such as a view, unless the CASCADE keyword has been specified.**
- **Privileges granted to this user by other users are not affected.**

REVOKE



Example 6.10/11 - REVOKE Specific Privileges

Revoke privilege SELECT on Branch table from all users.

```
REVOKE SELECT
ON Branch
FROM PUBLIC;
```

Revoke all privileges given to Director on Staff table.

```
REVOKE ALL PRIVILEGES
ON Staff
FROM Director;
```