Module 3 (Part 2)

Outline

- Data Control Commands
- Views in SQL
- Triggers
- Assertions
- Security and Authorization in SQL

Introduction to DCL

- DCL stands for Data Control Language.
- DCL is used to control user access in a database.
- This command is related to the security issues.
- Using DCL command, it allows or restricts the user from accessing data in database schema.
- DCL commands are as follows,
  - GRANT
  - REVOKE
- It is used to grant or revoke access permissions from any database user.

GRANT COMMAND

- **GRANT command** gives user's access privileges to the database.
- This command allows specified users to perform specific tasks.

Syntax:

GRANT <privilege list>
ON <relation name or view name>
TO <user/role list>;

GRANT ALL ON
employee
TO ABC;
[WITH GRANT OPTION]

REVOKE COMMAND

- **REVOKE command** is used to cancel previously granted or denied permissions.
- This command withdraw access privileges given with the GRANT command.
- It takes back permissions from user.

Syntax:

REVOKE <privilege list>
ON <relation name or view name>
FROM <user name>;

REVOKE UPDATE
ON employee
FROM ABC;

Difference between GRANT and REVOKE command.

GRANT	REVOKE
GRANT command allows a user to perform certain activities on the database.	**REVOKE command** disallows a user to perform certain activities.
It grants access privileges for database objects to other users.	It revokes access privileges for database objects previously granted to other users.

Example:
 	
GRANT privilege_name
ON object_name
TO
 	
{
user_name|PUBLIC|role_name
}
 	
[WITH GRANT OPTION];
	Example:
	
	REVOKE privilege_name
	ON object_name
	
	FROM
	{
	user_name|PUBLIC|role_name
	}

Introduction to Views

- View is a virtual table that contains rows and columns, just like a real table.
- It is used to hide complexity of query from user.
- A virtual table does not exist physically, it is created by a SQL statement that joins one or more tables.

Views in SQL

- In SQL, a view is a **virtual table** containing the records of one or more tables based on SQL statement executed.
- Just like a real table, view contains rows and columns.
- You can add SQL functions, WHERE, and JOIN statements to a view and present the data as if the data were coming from one single table
- The changes made in the table get automatically reflected in original table and vice versa

Views in SQL

Purpose of View:

- View is very useful in maintaining the security of database
- Consider a base table employee having following data

Emp_id	Emp_name	Salary	Address
E1	Kunal	8000	Camp
E2	Jay	7000	Tilak Road
E3	Radha	9000	Somwar Peth
E4	Sagar	7800	Warje
E5	Supriya	6700	LS Road

Views in SQL

1. Now just consider we want to give this table to any user but **don't want to show him salaries** of all the employees

In that we can create view from this table which will contain only the part of base table which we wish to show to user

Emp_id	Emp_name	Address
E1	Kunal	Camp
E2	Jay	Tilak Road
E3	Radha	Somwar Peth
E4	Sagar	Warje
E5	Supriya	LS Road

Views in SQL

2. Also in multiuser system, it may be possible that more than one user may want to update the data of some table

   - Consider two users A and B want to update the employee table.

     In such case we can give views to both these users

   - These users will make changes in their respective views, and their respective changes are done in the base table automatically

1. Creating View

Student_Detail

STU_ID	NAME	ADDRESS
1	Stephan	Delhi
2	Kathrin	Noida
3	David	Ghaziabad
4	Alina	Gurugram

- A view can be created using the **CREATE VIEW** statement.
- We can create a view from a single table or multiple tables.

1. Creating View

Syntax:

```
CREATE VIEW view_name AS
SELECT column1, column2.....
FROM table_name
WHERE condition;
```

Query:

```
CREATE VIEW DetailsView AS
SELECT NAME, ADDRESS
FROM Student_Details
WHERE STU_ID < 4;
```

```
SELECT * FROM DetailsView;
```

Output:

NAME	ADDRESS
Stephan	Delhi
Kathrin	Noida
David	Ghaziabad

1. Creating View

Creating View from multiple tables

Student_Detail

STU_ID	NAME	ADDRESS
1	Stephan	Delhi
2	Kathrin	Noida
3	David	Ghaziabad
4	Alina	Gurugram

1. Creating View

Student_Marks

STU_ID	NAME	MARKS	AGE
1	Stephan	97	19
2	Kathrin	86	21
3	David	74	18
4	Alina	90	20
5	John	96	18

1. Creating View

Query:

```
CREATE VIEW MarksView AS
SELECT Student_Detail.NAME, Student_Detail.ADDRESS, Student_Marks.MARKS
FROM Student_Detail, Student_Mark
WHERE Student_Detail.NAME = Student_Marks.NAME;
```

```
SELECT * FROM MarksView;
```

NAME	ADDRESS	MARKS
Stephan	Delhi	97
Kathrin	Noida	86
David	Ghaziabad	74
Alina	Gurugram	90

2. Updating View

- Update query is used to update the records of view
- Updation in view reflects the original table also means same changes will be made in the original table

```
UPDATE < view_name > SET<column1>=<value1>,<column2>=<value2>,...
WHERE <condition>;
```

AGENT_CODE	AGENT_NAME	WORKING_AREA	COMMISSION	PHONE_NO
A007	Ramasundar	Bangalore	0.15	077-25814763
A003	Alex	London	0.18	075-12458969
A008	Alford	New York	0.12	044-25874365
A011	Ravi Kumar	Bangalore	0.15	077-45625874
A010	Santakumar	Chennai	0.14	007-22388644
A012	Lucida	San Jose	0.12	044-52981425

2. Updating View

```
UPDATE agentview
SET commission=.13
WHERE working_area='London';
```

AGENT_CODE	AGENT_NAME	WORKING_AREA	COMMISSION	PHONE_NO
A007	Ramasundar	Bangalore	0.15	077-25814763
A003	Alex	London	0.13	075-12458969
A008	Alford	New York	0.12	044-25874365
A011	Ravi Kumar	Bangalore	0.15	077-45625874
A010	Santakumar	Chennai	0.14	007-22388644
A012	Lucida	San Jose	0.12	044-52981425

2. Updating View

- In case of view containing joins between multiple tables, only insertion and updation in the view is allowed, deletion is not allowed
- Data modification is **not allowed** in the view which is based on union queries
- Data modification is **not allowed** in the view where GROUPBY or DISTINCT statements are used
- In view, text and image columns **can’t be modified**

Regarding view updates

- View with single defining table is updatable only if view attributes contains primary key of the base relation
- View defined on multiple tables using joins are generally not updatable
- Views defined using grouping and aggregate functions are not updatable.

3. Dropping View

- Drop query is used to delete a view

Syntax

```
DROP VIEW view_name;
```

- **Example**:

```
DROP VIEW MarksView;
```

Types of Views

- There are **two** types of Views:

1. **Simple View**

- The views which are based on **only one table** called as Simple view.
- Allow to perform DML (Data Manipulation Language) operations with some restrictions.
- Query defining simple view **cannot have** any join or grouping condition.

Types of Views

2. Complex View

- The views which are based on **more than one table** called as complex view.
- Do not allow DML operations to be performed.
- Query defining complex view can have join or grouping condition.

Difference: Simple and Complex View

Simple View	Complex View
Contains only one single base table or is created from only one table.	Contains more than one base table or is created from more than one table.
We cannot use group functions like MAX(), COUNT(), etc.	We can use group functions.
INSERT, DELETE and UPDATE are directly possible on a simple view.	We cannot apply INSERT, DELETE and UPDATE on complex view directly.
Example: CREATE VIEW Employee AS SELECT Empid, Empname FROM Employee WHERE Empid = '030314';	**Example:** CREATE VIEW EmployeeByDepartment AS SELECT e.emp_id, d.dept_id, e.emp_name FROM Employee e, Department d WHERE e.dept_id=d.dept_id;

Advantages of View

- View provides data security for the same base tables.
- It allows different users to view the same data in different ways at the same time.
- It is used to represent additional information like derived columns.
- It is used to hide complex queries.
- It presents a consistent, unchanged image of the database structure, even if the tables are split or renamed.
- It does not allow direct access to the tables of the data dictionary.

Disadvantages of View

- We cannot use DML operations on View, if there is more than one table.
- When table is dropped view becomes inactive.
- View is a database object, so it occupies the space.
- Without table, view will not work.
- Updating is possible for simple view but not for complex views, they are read only type of views.

Triggers

- A trigger is a **procedure** that is automatically invoked by the DBMS in response to specific alteration database or a table in database.
- Triggers are stored in database as a simple database object.
- A database that has a set of associated triggers is called an **active database.**
- A database trigger enables DBA (Database Administrators) to create additional relationships between separate databases.

Triggers

- **Need/Purpose of Triggers**
  - To generate data automatically
  - Validate input data
  - Replicate data to different files to achieve data consistency

Components of Trigger (E-C-A Model)

- **Event (E):** SQL statement that causes the trigger to fire (or activate). This event may be insert, update or delete operation database table.
- **Condition (C)** : A condition that must be satisfied for execution of trigger.
- **Action (A)** : This is code or statement that execute when triggering condition is satisfied and trigger is activated on database table.

Triggers

Trigger syntax

```
CREATE [OR REPLACE] TRIGGER <Trigger Name>
[<ENABLE | DISABLE>]
<BEFORE|AFTER>
<INSERT|UPDATE | DELETE>
ON <Table_ Name>
        [FOR EACH ROW]
DECLARE
        <Variable_ Definitions>
BEGIN
        <TriggerCode> ;
END;
```

Triggers

- **Trigger Parameters**

OR REPLACE	If trigger is already present then drop and recreate the trigger
< Trigger Name>	Name of trigger to be created.
BEFORE	Indicates that trigger is to be fired before the triggering event occurs
AFTER	Indicates that trigger is to be fired After the triggering event occurs
INSERT	Indicates that trigger is to be fired whenever insert statement adds a row to table
UPDATE	Indicates that trigger is to be fired whenever Update statement modifies a row in a table
DELETE	Indicates that trigger is to be fired whenever delete statement removes a row from table

Triggers

- **Trigger Parameters**

FOR EACH ROW	Trigger will be fired only once for each row.
WHEN	Contains condition that must be satisfied to execute trigger
<trigger code>	Code to be executed whenever triggering event occurs

Trigger Types

1. **Row level Triggers**

- A row level trigger is fired each time the table is affected by the triggering statement
- For example, if an UPDATE statement changes multiple rows in a table, a row trigger is fired once for each row affected by the UPDATE statement
- If a triggering statement do not affect any row then a row trigger will not run only.
- IF **FOR EACH ROW** clause is written that means trigger is row level trigger.

Trigger Types

2. **Statement level triggers**

- A statement level trigger is fired only once on behalf of the triggering statement, irrespective of the number of rows in the table that are affected by the triggering statement.
- This trigger executes once even if no rows are affected.
- For example, if a DELETE statement deletes several rows from a table, a statement-level DELETE trigger fired only one time

Trigger Classification

- The classification of trigger is based on various parameters:

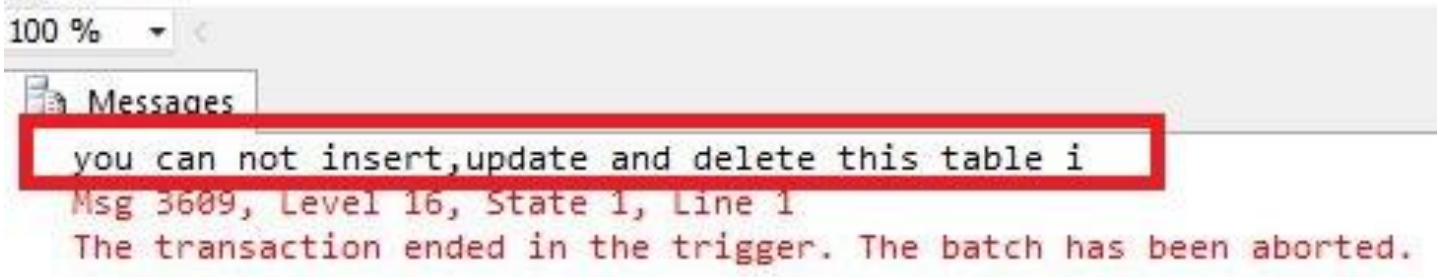
1. **Classification based on the timing**
   a) BEFORE Trigger: This trigger is fired before the occurrence of specified event.
   b) AFTER Trigger: This trigger is fired after the occurrence of specified event.
2. **Classification based on the level**
   a) STATEMENT level Trigger : This trigger is fired for once for the specified event statement.
   b) ROW level Trigger : This trigger is fired for all the records which are affected in the specified event. (only for DML)

DML Trigger

- The DML triggers are fired in response to DML (Data Manipulation Language) command events that start with Insert, Update, and Delete.
- Like insert_table, Update_view and Delete_table.

```
create trigger deep
on emp
for
insert,update,delete
as
print'you can not insert,update and delete this table i'
rollback;
```

DML Trigger

- When we insert, update, or delete a table in a database, then the following message appears,
- 

Trigger Example: Row Level Trigger

- Creating a trigger on employee table whenever a new employee added, a comment is written in EmpLog table

```
CREATE OR REPLACE TRIGGER AutoRecruit
AFTER INSERT ON EMP
FOR EACH ROW
BEGIN
        INSERT into EmpLog values(“Employee inserted”);
END;
```

Trigger created

Trigger Example

mysql>INSERT into EMP (1, ‘abc’, ‘manager’, 30000);

1 row created

mysql> SELECT * from EmpLog;

STATUS

Employee inserted

Trigger Example: Row Level Trigger

- There comes a new student add him to CS

```
mysql>
CREATE TRIGGER CSAutoRecruit
AFTER INSERT ON Student
FOR EACH ROW
BEGIN
        INSERT into Recruit(1,'abc','CS');
END
```

Trigger Example:2

```
CREATE TABLE Student(
studentID INT NOT NULL AUTO_INCREMENT,
FName VARCHAR(20),
LName VARCHAR(20),
Address VARCHAR(30),
City VARCHAR(15),
Marks INT,
PRIMARY KEY(studentID)
);
```

Trigger Example:2

- ***Before Insert***

```
CREATE TRIGGER calculate
before INSERT
ON student
FOR EACH ROW
SET new.marks = new.marks+100;
```

- Here when we insert data into the student table automatically the trigger will be invoked.
- The trigger will add 100 to the marks column into the student column.

Trigger Example:2

After Insert

- To use this variant we need one more table i.e, Percentage where the trigger will store the results. Use the below code to create the Percentage Table.

```
create table Final_mark(
per int );
```

```
CREATE TRIGGER total_mark
after insert
ON student
FOR EACH ROW
insert into Final_mark values(new.marks);
```

Here when we insert data to the table, *total_mark trigger* will store the result in the Final_mark table.

Trigger Advantages

- Triggers are useful for enforcing referential integrity, which preserves the defined relationships between tables when you add, update or delete the rows in those tables.
- Make sure that a column is filled with default information.
- After finding that the new information is inconsistent with the database, raise an error that will cause the entire transaction to roll back.

Trigger Disadvantages

- **Invisible from client applications –** Basically MySQL triggers are invoked and executed invisible from the client applications hence it is very much difficult to figure out what happens in the database layer.
- **Impose load on server –** Triggers can impose a high load on the database server.
- **Not recommended for high velocity of data –** Triggers are not beneficial for use with high-velocity data i.e. the data when a number of events per second are high. It is because in case of high-velocity data the triggers get triggered all the time.

Assertions

- An **assertion** is a predicate expressing a condition we wish the database to always satisfy.
- Domain constraints, functional dependency and referential integrity are special forms of assertion.
- Where a constraint cannot be expressed in these forms, we use an assertion, e.g.
  - Ensuring the sum of loan amounts for each branch is less than the sum of all account balances at the branch.
  - Ensuring every loan customer keeps a minimum of $1000 in an account.
- **Syntax**
  - **create assertion** assertion-name **check** predicate

Q. Ensuring the sum of loan amounts for each branch is less than the sum of all account balances at the branch.

```
create assertion sum-constraint check
    (not exists (select * from branch
            where (select sum)amount) from loan
                where (loan.bname = branch.bname >=
                    (select sum)amount) from account
                        where (account.bname = branch.bname)))
```

Ensuring every loan customer keeps a minimum of $1000 in an account.

create assertion *balance-constraint* **check**

(**not exists** (**select * from** *loan L*

(**where not exists** (**select ***

from *borrower B, depositor D, account A*

where *L.loan# = B.loan#* **and** *B.cname = D.cname* **and** *D.account# = A.account#* **and** *A.balance* >= 1000)))

- When an assertion is created, the system tests it for validity. If the assertion is valid, any further modification to the database is allowed only if it does not cause that assertion to be violated.
- This testing may result in significant overhead if the assertions are complex. Because of this, the **assert** should be used with great care.
- Some system developer omits support for general assertions or provides specialized form of assertions that are easier to test.

Security and Authorization

- Authentication
  - Authentication is the process of confirming that a user logs in only in accordance with the rights to perform the activities he is authorized to perform. User authentication can be performed at operating system level or database level itself.
- type of security authentication process
  - Based on Operating System authentications.
  - Lightweight Directory Access Protocol (LDAP)

- Authorization
  - access the Database and its functionality within the database system, which is managed by the Database manager.
- permissions available for authorization
  - Primary permission: Grants the authorization ID directly.
  - Secondary permission: Grants to the groups and roles if the user is a member
  - Public permission: Grants to all users publicly.
  - Context-sensitive permission: Grants to the trusted context role.

- Authorization can be given to users based on the categories
  - System-level authorization
  - System administrator [SYSADM]
  - System Control [SYSCTRL]
  - System maintenance [SYSMAINT]
  - System monitor [SYSMON]

- Authorities provide of control over instance-level functionality. Authority provide to group privileges, to control maintenance and authority operations. For instance, database and database objects.
  - Database-level authorization
  - Security Administrator [SECADM]
  - Database Administrator [DBADM]
  - Access Control [ACCESSCTRL]
  - Data access [DATAACCESS]
  - SQL administrator. [SQLADM]
  - Workload management administrator [WLMADM]
  - Explain [EXPLAIN]

- Authorities provide controls within the database.
  - Object-Level Authorization: Object-Level authorization involves verifying privileges when an operation is performed on an object.
  - Content-based Authorization: User can have read and write access to individual rows and columns on a particular table using Label-based access Control [LBAC].