Experiment No.9
Views and Triggers
Date of Performance:27/3/25
Date of Submission:28/3/25

Aim: Views and Triggers

Objective: Views can join and simplify multiple **tables** into a single virtual table A database trigger is procedural code that is automatically executed in response to certain events on a particular table or view in a database. The trigger is mostly used for maintaining the integrity of the information on the database. For example, when a new record (representing a new worker) is added to the employees table, new records should also be created in the tables of the taxes, vacations and salaries.

Theory:

VIEWS

A view is nothing more than a SQL statement that is stored in the database with an associated name. A view is actually a composition of a table in the form of a predefined SQL query.

A view can contain all rows of a table or select rows from a table. A view can be created from one or many tables which depends on the written SQL query to create a view.

Views, which are a type of virtual tables allow users to do the following −

- Structure data in a way that users or classes of users find natural or intuitive.
- Restrict access to the data in such a way that a user can see and (sometimes) modify exactly what they need and no more.
- Summarize data from various tables which can be used to generate reports.

Creating Views

Database views are created using the **CREATE VIEW** statement. Views can be created from a single table, multiple tables or another view.

To create a view, a user must have the appropriate system privilege according to the specific implementation.

The basic **CREATE VIEW** syntax is as follows −

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

You can include multiple tables in your SELECT statement in a similar way as you use them in a normal SQL SELECT query.

TIGGERS:

Triggers are stored programs, which are automatically executed or fired when some events occur. Triggers are, in fact, written to be executed in response to any of the following events :

- A database manipulation (DML) statement (DELETE, INSERT, or UPDATE)
- A database definition (DDL) statement (CREATE, ALTER, or DROP).
- A database operation (SERVERERROR, LOGON, LOGOFF, STARTUP, or SHUTDOWN).

Triggers can be defined on the table, view, schema, or database with which the event is associated.

Benefits of Triggers
Triggers can be written for the following purposes −

- Generating some derived column values automatically
- Enforcing referential integrity
- Event logging and storing information on table access
- Auditing
- Synchronous replication of tables
- Imposing security authorizations
- Preventing invalid transactions

Creating Triggers

The syntax for creating a trigger is −

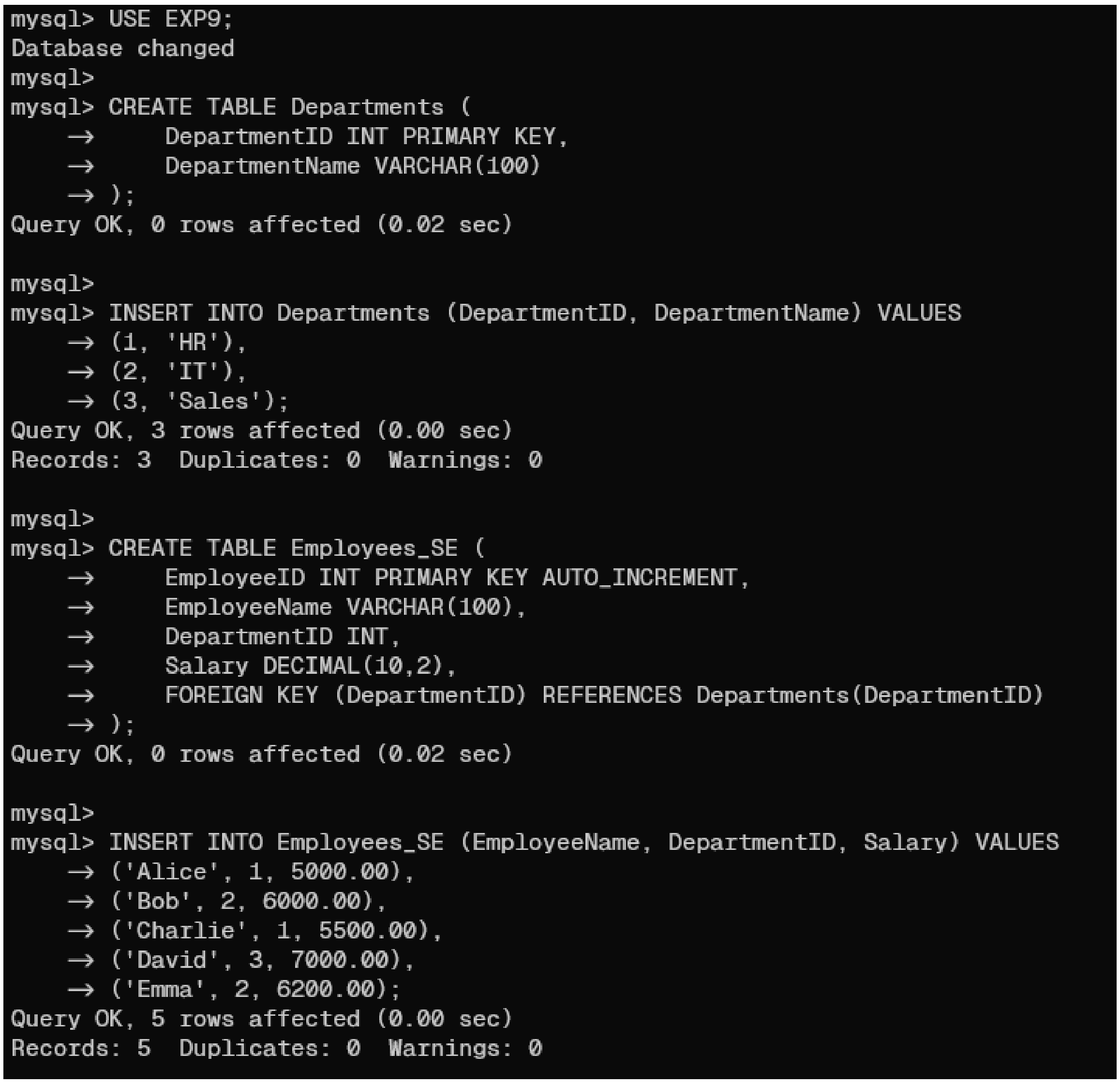
```
CREATE [OR REPLACE ] TRIGGER trigger_name
{BEFORE | AFTER | INSTEAD OF }
{INSERT [OR] | UPDATE [OR] | DELETE}
[OF col_name]
ON table_name
[REFERENCING OLD AS o NEW AS n]
[FOR EACH ROW]
WHEN (condition)
DECLARE
  Declaration-statements
BEGIN
```

```
  Executable-statements
EXCEPTION
  Exception-handling-statements
END;
```

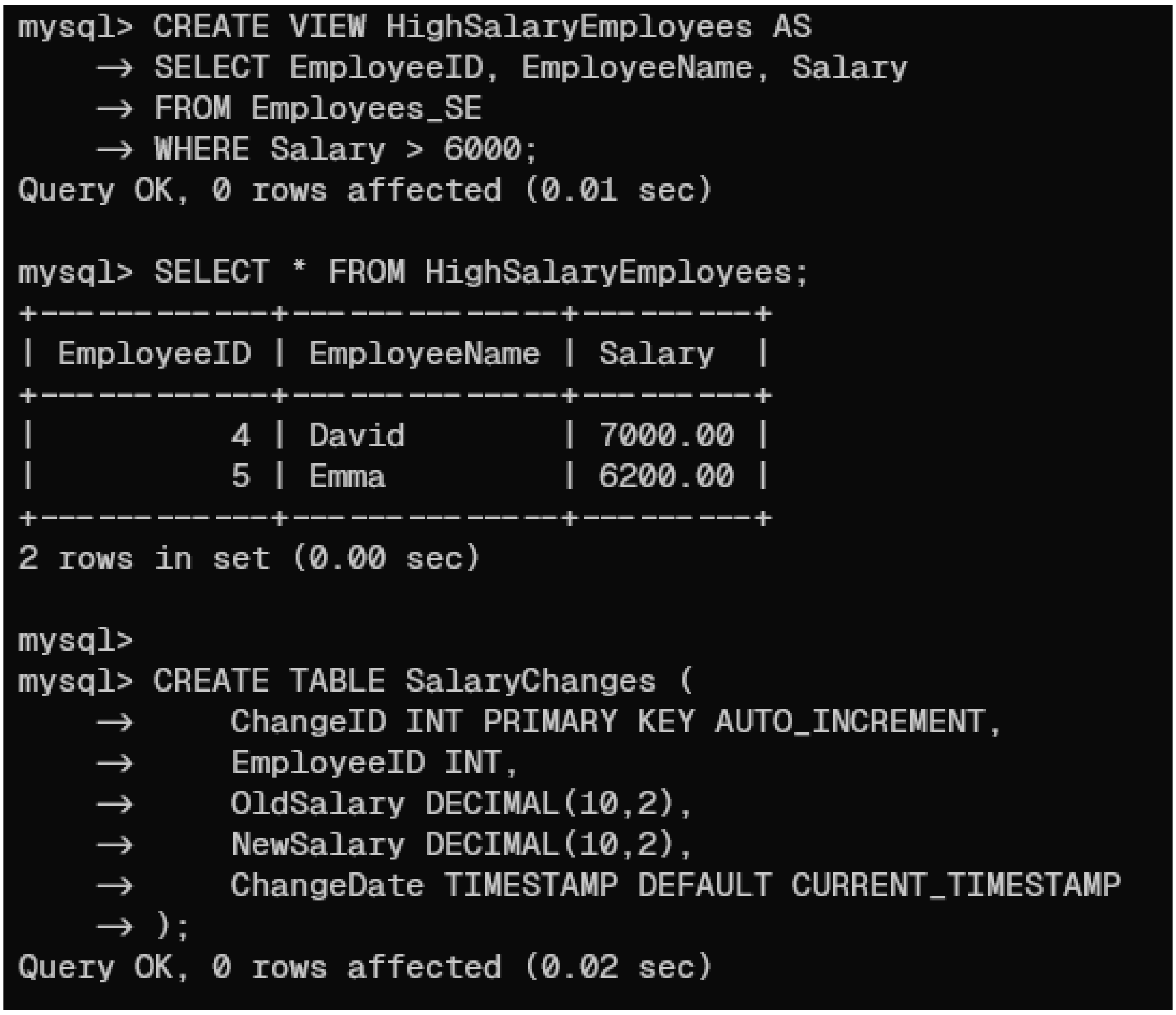
Where,

- CREATE [OR REPLACE] TRIGGER trigger_name − Creates or replaces an existing trigger with the *trigger_name*.
- {BEFORE | AFTER | INSTEAD OF} − This specifies when the trigger will be executed. The INSTEAD OF clause is used for creating trigger on a view.
- {INSERT [OR] | UPDATE [OR] | DELETE} − This specifies the DML operation.
- [OF col_name] − This specifies the column name that will be updated.
- [ON table_name] − This specifies the name of the table associated with the trigger.
- [REFERENCING OLD AS o NEW AS n] − This allows you to refer new and old values for various DML statements, such as INSERT, UPDATE, and DELETE.
- [FOR EACH ROW] − This specifies a row-level trigger, i.e., the trigger will be executed for each row being affected. Otherwise the trigger will execute just once when the SQL statement is executed, which is called a table level trigger.
- WHEN (condition) − This provides a condition for rows for which the trigger would fire. This clause is valid only for row-level triggers.

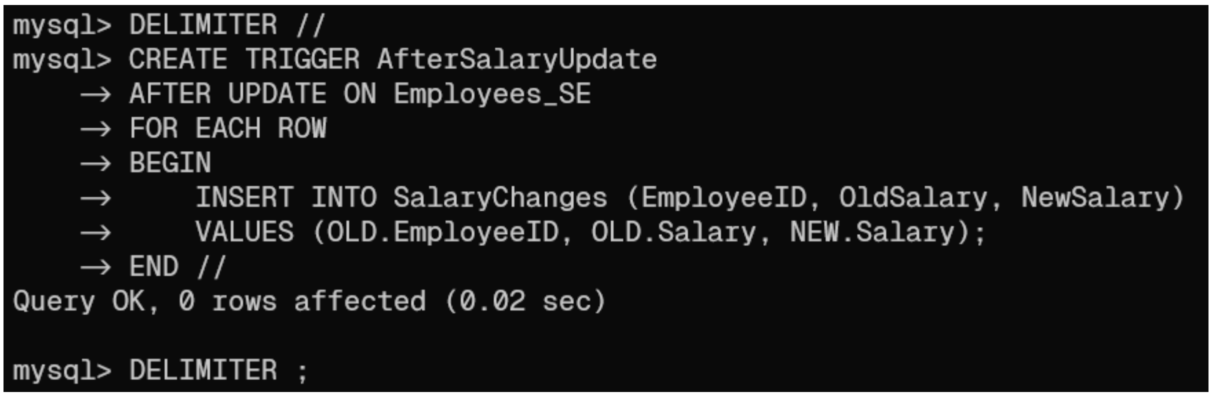
Implementation:



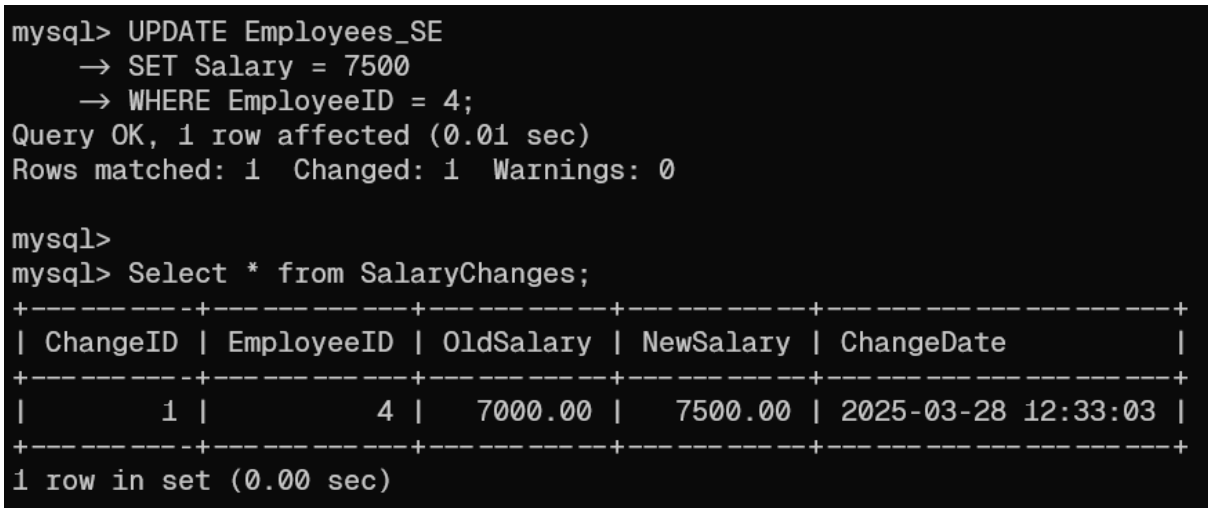
```
mysql> USE EXP9;
Database changed
mysql>
mysql> CREATE TABLE Departments (
    ->     DepartmentID INT PRIMARY KEY,
    ->     DepartmentName VARCHAR(100)
    -> );
Query OK, 0 rows affected (0.02 sec)

mysql>
mysql> INSERT INTO Departments (DepartmentID, DepartmentName) VALUES
    -> (1, 'HR'),
    -> (2, 'IT'),
    -> (3, 'Sales');
Query OK, 3 rows affected (0.00 sec)
Records: 3  Duplicates: 0  Warnings: 0

mysql>
mysql> CREATE TABLE Employees_SE (
    ->     EmployeeID INT PRIMARY KEY AUTO_INCREMENT,
    ->     EmployeeName VARCHAR(100),
    ->     DepartmentID INT,
    ->     Salary DECIMAL(10,2),
    ->     FOREIGN KEY (DepartmentID) REFERENCES Departments(DepartmentID)
    -> );
Query OK, 0 rows affected (0.02 sec)

mysql>
mysql> INSERT INTO Employees_SE (EmployeeName, DepartmentID, Salary) VALUES
    -> ('Alice', 1, 5000.00),
    -> ('Bob', 2, 6000.00),
    -> ('Charlie', 1, 5500.00),
    -> ('David', 3, 7000.00),
    -> ('Emma', 2, 6200.00);
Query OK, 5 rows affected (0.00 sec)
Records: 5  Duplicates: 0  Warnings: 0
```



```
mysql> CREATE VIEW HighSalaryEmployees AS
    -> SELECT EmployeeID, EmployeeName, Salary
    -> FROM Employees_SE
    -> WHERE Salary > 6000;
Query OK, 0 rows affected (0.01 sec)

mysql> SELECT * FROM HighSalaryEmployees;
+------------+--------------+---------+
| EmployeeID | EmployeeName | Salary  |
+------------+--------------+---------+
|          4 | David        | 7000.00 |
|          5 | Emma         | 6200.00 |
+------------+--------------+---------+
2 rows in set (0.00 sec)

mysql>
mysql> CREATE TABLE SalaryChanges (
    ->     ChangeID INT PRIMARY KEY AUTO_INCREMENT,
    ->     EmployeeID INT,
    ->     OldSalary DECIMAL(10,2),
    ->     NewSalary DECIMAL(10,2),
    ->     ChangeDate TIMESTAMP DEFAULT CURRENT_TIMESTAMP
    -> );
Query OK, 0 rows affected (0.02 sec)
```



```
mysql> DELIMITER //
mysql> CREATE TRIGGER AfterSalaryUpdate
    -> AFTER UPDATE ON Employees_SE
    -> FOR EACH ROW
    -> BEGIN
    ->     INSERT INTO SalaryChanges (EmployeeID, OldSalary, NewSalary)
    ->     VALUES (OLD.EmployeeID, OLD.Salary, NEW.Salary);
    -> END //
Query OK, 0 rows affected (0.02 sec)

mysql> DELIMITER ;
```



```
mysql> UPDATE Employees_SE
    -> SET Salary = 7500
    -> WHERE EmployeeID = 4;
Query OK, 1 row affected (0.01 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql>
mysql> Select * from SalaryChanges;
+----------+------------+-----------+-----------+---------------------+
| ChangeID | EmployeeID | OldSalary | NewSalary | ChangeDate          |
+----------+------------+-----------+-----------+---------------------+
|        1 |          4 |   7000.00 |   7500.00 | 2025-03-28 12:33:03 |
+----------+------------+-----------+-----------+---------------------+
1 row in set (0.00 sec)
```

Conclusion: The experiment successfully demonstrated the use of SQL views and triggers to enhance database functionality. Views allowed efficient data structuring and simplified access to multiple tables, enabling intuitive data representation and secure access controls. Triggers provided automated responses to database events, ensuring integrity, synchronization, and security while supporting advanced features like derived column generation and transaction validation. These implementations significantly contribute to the efficient management and operation of relational databases.