Program 1

Create a table called Employee & execute the following. Employee(EMPNO,ENAME,JOB, MANAGER\_NO, SAL, COMMISSION)

1. Create a user and grant all permissions to theuser.
2. Insert the any three records in the employee table contains attributes EMPNO,ENAME JOB, MANAGER\_NO, SAL, COMMISSION and use rollback.

Check the result.

1. Add primary key constraint and not null constraint to the employee table.
2. Insert null values to the employee table and verify the result. Solution:

Lets login with the **root** account as shown below. Create a database **COMPANY** and switch to it using the **USE** command.

$ sudo mysql -u root

mysql> CREATE DATABASE COMPANY;

Query OK, 1 row affected (0.14 sec)

mysql> USE COMPANY;

Database changed

# Creating the Employee Table

Within the Database

create a table

as follows. Use the

**TABLES;** command to confirm that the table was indeed created.

**SHOW**

**Employee**

**COMPANY**

mysql> CREATE TABLE COMPANY.Employee (

-> EMPNO INT,

-> ENAME VARCHAR(255),

-> JOB VARCHAR(255),

-> MANAGER\_NO INT,

-> SAL DECIMAL(10, 2),

-> COMMISSION DECIMAL(10, 2)

-> );

Query OK, 0 rows affected (0.91 sec)

mysql> SHOW TABLES;

+ +

| Tables\_in\_COMPANY |

+ +

| Employee |

+ +

1 row in set (0.00 sec)

You can verify the structure of this newly created Employee table using the DESC command.

mysql> DESC COMPANY.Employee;

+------------+---------------+------+-----+---------+ +

| Field | Type | Null | Key | Default | Extra |

+------------+---------------+------+-----+---------+ +

| EMPNO | int | YES | | NULL | |

| ENAME | varchar(255) | YES | | NULL | |

| JOB | varchar(255) | YES | | NULL | |

| MANAGER\_NO | int | YES | | NULL | |

| SAL | decimal(10,2) | YES | | NULL | |

| COMMISSION | decimal(10,2) | YES | | NULL | |

+------------+---------------+------+-----+---------+ +

6 rows in set (0.00 sec)

# Create a User and Grant Permissions

mysql> CREATE USER IF NOT EXISTS 'dbuser'@'localhost' IDENTIFIED BY 'T0p5E(RET';

mysql> GRANT ALL PRIVILEGES ON COMPANY.Employee TO 'dbuser'@'localhost';

Now logout and login with the new account credentials. Press Command to login with new user account is shown below.

**Ctrl+D**

to logout.

$ mysql -u dbuser -p Enter password:

Welcome to the MySQL monitor. Commands end with ; or \g. Your MySQL connection id is 11

Server version: 8.0.37 MySQL Community Server - GPL

Copyright (c) 2000, 2024, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its affiliates. Other names may be trademarks of their respective owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql>

Now you have successfully logged with your new account. Change the current

**COMMIT**

**USE**

**COMPANY**

database to

**ROLLBACK**

database using

command. Now we will illustrate how

to records and also the

**insert**

and

facilities.

-- Change the current database to COMPANY

mysql> USE COMPANY;

Database changed

mysql> SELECT \* FROM Employee;

Query OK, 0 rows affected (0.00 sec)

-- START A TRANSACTION mysql> START TRANSACTION;

Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO Employee (EMPNO, ENAME, JOB, MANAGER\_NO, SAL, COMMISSION)

-> VALUES (1, 'Kavana Shetty', 'Manager', NULL, 5000.00, 1000.00);

Query OK, 1 row affected (0.00 sec)

-- COMMIT DATABASE, db CONTENTS ARE WRITTEN TO THE DISK

mysql> COMMIT;

Query OK, 0 rows affected (0.06 sec)

-- DISPLAY TABLE CONTENTS

mysql> SELECT \* FROM Employee;

+-------+---------------+---------+------------+---------+ +

| EMPNO | ENAME | JOB | MANAGER\_NO | SAL | COMMISSION |

+-------+---------------+---------+------------+---------+ +

| 1 | Kavana Shetty | Manager | NULL | 5000.00 | 1000.00 |

+-------+---------------+---------+------------+---------+ +

1 row in set (0.00 sec)

-- START ANOTHER TRANSACTION mysql> START TRANSACTION;

-- INSERT MORE RECORDS

mysql> INSERT INTO Employee (EMPNO, ENAME, JOB, MANAGER\_NO, SAL, COMMISSION)

VALUES (2, 'Ram Charan', 'Developer', 1, 4000.00, NULL);

mysql> INSERT INTO Employee (EMPNO, ENAME, JOB, MANAGER\_NO, SAL, COMMISSION)

VALUES (3, 'Honey Singh', 'Salesperson', 2, 3000.00, 500.00);

mysql> SELECT \* FROM Employee;

+-------+---------------+-------------+------------+---------+ +

| EMPNO | ENAME | JOB | MANAGER\_NO | SAL | COMMISSION |

+-------+---------------+-------------+------------+---------+ +

| 1 | Kavana Shetty | Manager | NULL | 5000.00 | 1000.00 |

| 2 | Ram Charan | Developer | 1 | 4000.00 | NULL |

| 3 | Honey Singh | Salesperson | 2 | 3000.00 | 500.00 |

+-------+---------------+-------------+------------+---------+ +

3 rows in set (0.00 sec)

mysql> DELETE FROM Employee where ENAME = 'Kavana Shetty'; Query OK, 1 row affected (0.00 sec)

mysql> SELECT \* FROM Employee;

+-------+-------------+-------------+------------+---------+ +

| EMPNO | ENAME | JOB | MANAGER\_NO | SAL | COMMISSION |

+-------+-------------+-------------+------------+---------+ +

| 2 | Ram Charan | Developer | 1 | 4000.00 | NULL |

| 3 | Honey Singh | Salesperson | 2 | 3000.00 | 500.00 |

+-------+-------------+-------------+------------+---------+ +

2 rows in set (0.00 sec)

-- ROLLBACK 2 INSERTS AND 1 DELETE OPERATIONS

mysql> ROLLBACK;

Query OK, 0 rows affected (0.06 sec)

mysql> SELECT \* FROM Employee;

+-------+---------------+---------+------------+---------+ +

| EMPNO | ENAME | JOB | MANAGER\_NO | SAL | COMMISSION |

+-------+---------------+---------+------------+---------+ +

| 1 | Kavana Shetty | Manager | NULL | 5000.00 | 1000.00 |

+-------+---------------+---------+------------+---------+ +

1 row in set (0.00 sec)

You can now see how the rollback operation can be used above.

# Adding Constraints

## Add Primary Key Constraint

-- Add Primary Key Constraint mysql> ALTER TABLE Employee

-> ADD CONSTRAINT pk\_employee PRIMARY KEY (EMPNO);

Query OK, 0 rows affected (1.65 sec)

-- verify primary key constraint mysql> DESC Employee;

+------------+---------------+------+-----+---------+ +

| Field | Type | Null | Key | Default | Extra |

+------------+---------------+------+-----+---------+ +

| EMPNO | int | NO | PRI | NULL | |

| ENAME | varchar(255) | YES | | NULL | |

| JOB | varchar(255) | YES | | NULL | |

| MANAGER\_NO | int | YES | | NULL | |

| SAL | decimal(10,2) | YES | | NULL | |

| COMMISSION | decimal(10,2) | YES | | NULL | |

+------------+---------------+------+-----+---------+ +

6 rows in set (0.00 sec)

mysql> INSERT INTO Employee (EMPNO, ENAME, JOB, MANAGER\_NO, SAL, COMMISSION)

-> VALUES (1, 'Ranjan', 'Manager', NULL, 5000.00, 1000.00);

ERROR 1062 (23000): Duplicate entry '1' for key 'Employee.PRIMARY'

Since EMPNO field is the primary key it cannot have duplicate values, hence we see that the insert operation fails when provided with a duplicate value.

## Add Not Null Constraint

-- Add Not Null Constraints mysql> ALTER TABLE Employee

-> MODIFY ENAME VARCHAR(255) NOT NULL,

-> MODIFY JOB VARCHAR(255) NOT NULL,

-> MODIFY SAL DECIMAL(10, 2) NOT NULL;

Query OK, 0 rows affected (1.08 sec)

mysql> INSERT INTO Employee (EMPNO, ENAME, JOB, MANAGER\_NO, SAL, COMMISSION)

-> VALUES (4, 'Ranjan', 'Manager', NULL, 5000.00, 1000.00);

Query OK, 1 row affected (0.16 sec)

mysql>

mysql> SELECT \* FROM Employee;

+-------+---------------+---------+------------+---------+ +

| EMPNO | ENAME | JOB | MANAGER\_NO | SAL | COMMISSION |

+-------+---------------+---------+------------+---------+ +

| 1 | Kavana Shetty | Manager | NULL | 5000.00 | 1000.00 |

| 4 | Ranjan | Manager | NULL | 5000.00 | 1000.00 |

+-------+---------------+---------+------------+---------+ +

2 rows in set (0.00 sec)

mysql> INSERT INTO Employee (ENAME, JOB, MANAGER\_NO, SAL, COMMISSION)

-> VALUES (NULL, 'Tester', NULL, 3500.00, NULL);

ERROR 1048 (23000): Column 'ENAME' cannot be null

We just illustrated as to how to add not null constraint to the Employee table. We

see that the first insert doesn’t violate null constraint, however the second insert does violate null constraint as ENAME field cannot be null.

Program 2

Create a table called Employee that contain attributes EMPNO,ENAME,JOB, MGR,SAL & execute the following.

1. Add a column commission with domain to the Employee table.
2. Insert any five records into the table.
3. Update the column details of job
4. Rename the column of Employ table using alter command.
5. Delete the employee whose Empno is 105. Solution:

# Creating the Employee Table

mysql> CREATE DATABASE COMPANY02;

Query OK, 1 row affected (0.16 sec)

mysql> USE COMPANY02;

Database changed

mysql> CREATE TABLE Employee (

-> EMPNO INT,

-> ENAME VARCHAR(255),

-> JOB VARCHAR(255),

-> MGR INT,

-> SAL DECIMAL(10, 2)

-> );

Query OK, 0 rows affected (0.48 sec)

mysql> SHOW TABLES;

+ +

| Tables\_in\_COMPANY02 |

+ +

| Employee |

+ +

1 row in set (0.00 sec)

mysql> DESC Employee;

+-------+---------------+------+-----+---------+ +

| Field | Type | Null | Key | Default | Extra |

+-------+---------------+------+-----+---------+ +

| EMPNO | int | YES | | NULL | |

| ENAME | varchar(255) | YES | | NULL | |

| JOB | varchar(255) | YES | | NULL | |

| MGR | int | YES | | NULL | |

| SAL | decimal(10,2) | YES | | NULL | |

+-------+---------------+------+-----+---------+ +

5 rows in set (0.00 sec)

# Adding a Column (Commission) to the Employee Table

mysql> ALTER TABLE Employee

-> ADD COLUMN COMMISSION DECIMAL(10, 2);

Query OK, 0 rows affected (0.37 sec)

mysql> DESC Employee;

+------------+---------------+------+-----+---------+ +

| Field | Type | Null | Key | Default | Extra |

+------------+---------------+------+-----+---------+ +

| EMPNO | int | YES | | NULL | |

| ENAME | varchar(255) | YES | | NULL | |

| JOB | varchar(255) | YES | | NULL | |

| MGR | int | YES | | NULL | |

| SAL | decimal(10,2) | YES | | NULL | |

| COMMISSION | decimal(10,2) | YES | | NULL | |

+------------+---------------+------+-----+---------+ +

6 rows in set (0.00 sec)

**ALTER**

We have added a column shown above.

**COMMISSION**

using the

command, which is

# Inserting 5 Records into the Employee Table

mysql> INSERT INTO Employee (EMPNO, ENAME, JOB, MGR, SAL, COMMISSION)

-> VALUES

-> (101, 'Radha Bai', 'Manager', NULL, 5000.00, 1000.00),

-> (102, 'Krishna Kumar', 'Developer', 101, 4000.00, NULL),

-> (103, 'Abdul Sattar', 'Salesperson', 102, 3000.00, 500.00),

-> (104, 'Bob Johnson', 'Accountant', 101, 4500.00, NULL),

-> (105, 'Amartya Sen', 'HR Manager', 101, 4800.00, 800.00);

Query OK, 5 rows affected (0.12 sec) Records: 5 Duplicates: 0 Warnings: 0

mysql> SELECT \* FROM Employee;

+-------+---------------+-------------+------+---------+ +

| EMPNO | ENAME | JOB | MGR | SAL | COMMISSION |

+-------+---------------+-------------+------+---------+ +

| 101 | Radha Bai | Manager | NULL | 5000.00 | 1000.00 |

| 102 | Krishna Kumar | Developer | 101 | 4000.00 | NULL |

| 103 | Abdul Sattar | Salesperson | 102 | 3000.00 | 500.00 |

| 104 | Bob Johnson | Accountant | 101 | 4500.00 | NULL |

| 105 | Amartya Sen | HR Manager | 101 | 4800.00 | 800.00 |

+-------+---------------+-------------+------+---------+ +

5 rows in set (0.00 sec)

# Updating Column Details (JOB) in the Employee Table

mysql> UPDATE Employee

-> SET JOB = 'Senior Developer'

-> WHERE EMPNO = 102;

Query OK, 1 row affected (0.09 sec)

Rows matched: 1 Changed: 1 Warnings: 0

mysql> SELECT \* FROM Employee;

+-------+---------------+------------------+------+---------+ +

| EMPNO | ENAME | JOB | MGR | SAL | COMMISSION |

+-------+---------------+------------------+------+---------+ +

| 101 | Radha Bai | Manager | NULL | 5000.00 | 1000.00 |

| 102 | Krishna Kumar | Senior Developer | 101 | 4000.00 | NULL |

| 103 | Abdul Sattar | Salesperson | 102 | 3000.00 | 500.00 |

| 104 | Bob Johnson | Accountant | 101 | 4500.00 | NULL |

| 105 | Amartya Sen | HR Manager | 101 | 4800.00 | 800.00 |

+-------+---------------+------------------+------+---------+ +

5 rows in set (0.00 sec)

Renaming a Column in the Employee Table

To rename the **MGR** column to **MANAGER\_ID**:

mysql> ALTER TABLE Employee

-> CHANGE COLUMN MGR MANAGER\_ID INT;

Query OK, 0 rows affected (0.30 sec) Records: 0 Duplicates: 0 Warnings: 0

mysql> DESC Employee;

+------------+---------------+------+-----+---------+ +

| Field | Type | Null | Key | Default | Extra |

+------------+---------------+------+-----+---------+ +

| EMPNO | int | YES | | NULL | |

| ENAME | varchar(255) | YES | | NULL | |

| JOB | varchar(255) | YES | | NULL | |

| MANAGER\_ID | int | YES | | NULL | |

| SAL | decimal(10,2) | YES | | NULL | |

| COMMISSION | decimal(10,2) | YES | | NULL | |

+------------+---------------+------+-----+---------+ +

6 rows in set (0.00 sec)

# Deleting a Specific Employee (EMPNO = 105) from the Employee Table

mysql> DELETE FROM Employee

-> WHERE EMPNO = 105;

Query OK, 1 row affected (0.14 sec)

mysql> SELECT \* FROM Employee;

+-------+---------------+------------------+------------+---------+ +

| EMPNO | ENAME | JOB | MANAGER\_ID | SAL | COMMISSION |

+-------+---------------+------------------+------------+---------+ +

| 101 | Radha Bai | Manager | NULL | 5000.00 | 1000.00 |

| 102 | Krishna Kumar | Senior Developer | 101 | 4000.00 | NULL |

| 103 | Abdul Sattar | Salesperson | 102 | 3000.00 | 500.00 |

| 104 | Bob Johnson | Accountant | 101 | 4500.00 | NULL |

+-------+---------------+------------------+------------+---------+ +

4 rows in set (0.00 sec)

Program 3

Queries using aggregate functions(COUNT,AVG,MIN,MAX,SUM),Group by,Orderby. Employee(E\_id, E\_name, Age, Salary)

1. Create Employee table containing all Records E\_id, E\_name, Age, Salary.
2. Count number of employee names from employeetable
3. Find the Maximum age from employee table.
4. Find the Minimum age from employeetable.
5. Find salaries of employee in Ascending Order.
6. Find grouped salaries of employees Solution:

# Creating the Employee Table

mysql> CREATE DATABASE COMPANY03;

Query OK, 1 row affected (0.09 sec)

mysql> USE COMPANY03;

Database changed

mysql> CREATE TABLE Employee (

-> E\_id INT PRIMARY KEY,

-> E\_name VARCHAR(255),

-> Age INT,

-> Salary DECIMAL(10, 2)

-> );

Query OK, 0 rows affected (1.00 sec)

mysql> DESC Employee;

+--------+---------------+------+-----+---------+ +

| Field | Type | Null | Key | Default | Extra |

+--------+---------------+------+-----+---------+ +

| E\_id | int | NO | PRI | NULL | |

| E\_name | varchar(255) | YES | | NULL | |

| Age | int | YES | | NULL | |

| Salary | decimal(10,2) | YES | | NULL | |

+--------+---------------+------+-----+---------+ +

4 rows in set (0.00 sec)

# Populating the Employee Table with 12 Records

mysql> INSERT INTO Employee (E\_id, E\_name, Age, Salary)

-> VALUES

-> (1, 'Samarth', 30, 50000.00),

-> (2, 'Ramesh Kumar', 25, 45000.00),

-> (3, 'Seema Banu', 35, 60000.00),

-> (4, 'Dennis Anil', 28, 52000.00),

-> (5, 'Rehman Khan', 32, 58000.00),

-> (6, 'Pavan Gowda', 40, 70000.00),

-> (7, 'Shruthi Bhat', 27, 48000.00),

-> (8, 'Sandesh Yadav', 29, 51000.00),

-> (9, 'Vikram Acharya', 33, 62000.00),

-> (10, 'Praveen Bellad', 26, 46000.00),

-> (11, 'Sophia Mary', 31, 55000.00),

-> (12, 'Darshan Desai', 34, 63000.00);

Query OK, 12 rows affected (0.14 sec) Records: 12 Duplicates: 0 Warnings: 0

mysql> SELECT \* FROM Employee;

+------+----------------+------+ +

| E\_id | E\_name | Age | Salary |

+------+----------------+------+ +

| 1 | Samarth | 30 | 50000.00 |

| 2 | Ramesh Kumar | 25 | 45000.00 |

| 3 | Seema Banu | 35 | 60000.00 |

| 4 | Dennis Anil | 28 | 52000.00 |

| 5 | Rehman Khan | 32 | 58000.00 |

| 6 | Pavan Gowda | 40 | 70000.00 |

| 7 | Shruthi Bhat | 27 | 48000.00 |

| 8 | Sandesh Yadav | 29 | 51000.00 |

| 9 | Vikram Acharya | 33 | 62000.00 |

| 10 | Praveen Bellad | 26 | 46000.00 |

| 11 | Sophia Mary | 31 | 55000.00 |

| 12 | Darshan Desai | 34 | 63000.00 |

+------+----------------+------+ +

12 rows in set (0.00 sec)

# Count Number of Employee Names

mysql> SELECT COUNT(E\_name) AS TotalEmployees

-> FROM Employee;

+ +

| TotalEmployees |

+ +

| 12 |

+ +

1 row in set (0.00 sec)

# Find the Maximum Age

mysql> SELECT MAX(Age) AS MaxAge

-> FROM Employee;

+ +

| MaxAge |

+ +

| 40 |

+ +

1 row in set (0.01 sec)

# Find the Minimum Age

mysql> SELECT MIN(Age) AS MinAge

-> FROM Employee;

+ +

| MinAge |

+ +

| 25 |

+ +

1 row in set (0.00 sec)

# Find Salaries of Employees in Ascending Order

mysql> SELECT E\_name, Salary

-> FROM Employee

-> ORDER BY Salary ASC;

+----------------+ +

| E\_name | Salary |

+----------------+ +

| Ramesh Kumar | 45000.00 |

| Praveen Bellad | 46000.00 |

| Shruthi Bhat | 48000.00 |

| Samarth | 50000.00 |

| Dennis Anil | 52000.00 |

| Sandesh Yadav | 52000.00 |

| Sophia Mary | 55000.00 |

| Rehman Khan | 58000.00 |

| Seema Banu | 62000.00 |

| Vikram Acharya | 62000.00 |

| Darshan Desai | 63000.00 |

| Pavan Gowda | 70000.00 |

+----------------+ +

12 rows in set (0.00 sec)

# Find Grouped Salaries of Employees

mysql> SELECT Salary, COUNT(\*) AS EmployeeCount

-> FROM Employee

-> GROUP BY Salary;

+----------+ +

| Salary | EmployeeCount |

+----------+ +

| 50000.00 | 1 |

|  |  |
| --- | --- |
| | 45000.00 | | 1 | |
| | 62000.00 | | 2 | |
| | 52000.00 | | 2 | |
| | 58000.00 | | 1 | |
| | 70000.00 | | 1 | |
| | 48000.00 | | 1 | |
| | 46000.00 | | 1 | |
| | 55000.00 | | 1 | |

| 63000.00 | 1 |

+----------+ +

10 rows in set (0.00 sec)

In these queries:





counts the number of non-NULL values in the

finds the maximum age among the employees.

column.

* **MIN(Age)** finds the minimum age among the employees.

**MAX(Age)**

**E\_name**

**COUNT(E\_name)**



sorts the employees based on their salaries in ascending

order.

1. groups employees by their salaries and counts the number of employees for each salary.

**GROUP BY Salary**

**ORDER BY Salary ASC**

Program 4

Create a row level trigger for the customers table that would fire for INSERT or UPDATE or DELETE operations performed on the CUSTOMERS table. This trigger will display the salary difference between the old & new Salary.

CUSTOMERS(ID,NAME,AGE,ADDRESS,SALARY)

Solution:

# Create the

First, create the

**CUSTOMERS**

# Table

table with the specified columns:

**CUSTOMERS**

mysql> CREATE DATABASE COMPANY04;

Query OK, 1 row affected (0.14 sec)

mysql> USE COMPANY04;

Database changed

mysql> CREATE TABLE CUSTOMERS (

-> ID INT PRIMARY KEY AUTO\_INCREMENT,

-> NAME VARCHAR(255),

-> AGE INT,

-> ADDRESS VARCHAR(255),

-> SALARY DECIMAL(10, 2)

-> );

Query OK, 0 rows affected (0.49 sec)

To achieve the desired functionality of capturing changes on **INSERT**, **UPDATE**,

or operations and displaying the salary difference in MySQL, you’ll need to

**DELETE**

create separate row-level triggers for each operation (**INSERT**, **UPDATE**, **DELETE**). These

triggers will capture the **OLD** and **NEW** values of the column and display the

**SALARY**

salary difference when an INSERT, UPDATE, or DELETE operation occurs.Here’s how you can do it:

# Create Trigger for INSERT Operation

-- INSERT TRIGGER DELIMITER //

CREATE TRIGGER after\_insert\_salary\_difference AFTER INSERT ON CUSTOMERS

FOR EACH ROW BEGIN

SET @my\_sal\_diff = CONCAT('salary inserted is ', NEW.SALARY);

END;//

DELIMITER ;

# Create Trigger for UPDATE Operation

-- UPDATE TRIGGER DELIMITER //

CREATE TRIGGER after\_update\_salary\_difference AFTER UPDATE ON CUSTOMERS

FOR EACH ROW BEGIN

DECLARE old\_salary DECIMAL(10, 2); DECLARE new\_salary DECIMAL(10, 2);

SET old\_salary = OLD.SALARY; SET new\_salary = NEW.SALARY;

SET @my\_sal\_diff = CONCAT('salary difference after update is ', NEW.SALARY - OLD.SALARY);

END;//

DELIMITER ;

# Create Trigger for DELETE Operation

-- DELETE TRIGGER DELIMITER //

CREATE TRIGGER after\_delete\_salary\_difference AFTER DELETE ON CUSTOMERS

FOR EACH ROW BEGIN

SET @my\_sal\_diff = CONCAT('salary deleted is ', OLD.SALARY);

END;//

DELIMITER ;

# Testing the Trigger:

Once the triggers are created, you can perform **INSERT**, **UPDATE**, or

operations

**DELETE**

on the triggers.

**CUSTOMERS**

For example:

table to observe the salary difference messages generated by the

mysql> -- test INSERT TRIGGER

mysql> INSERT INTO CUSTOMERS (NAME, AGE, ADDRESS, SALARY)

-> VALUES ('Shankara', 35, '123 Main St', 50000.00);

Query OK, 1 row affected (0.14 sec)

mysql>

mysql> SELECT @my\_sal\_diff AS SAL\_DIFF;

+ +

| SAL\_DIFF |

+ +

| salary inserted is 50000.00 |

+ +

1 row in set (0.00 sec)

mysql> -- test UPDATE TRIGGER mysql> UPDATE CUSTOMERS

-> SET SALARY = 55000.00

-> WHERE ID = 1;

Query OK, 1 row affected (0.13 sec)

Rows matched: 1 Changed: 1 Warnings: 0

mysql> SELECT @my\_sal\_diff AS SAL\_DIFF;

+ +

| SAL\_DIFF |

+ +

| salary difference after update is 5000.00 |

+ +

1 row in set (0.00 sec)

mysql> -- test DELETE TRIGGER mysql> DELETE FROM CUSTOMERS

-> WHERE ID = 1;

Query OK, 1 row affected (0.13 sec)

mysql>

mysql> SELECT @my\_sal\_diff AS SAL\_DIFF;

+ +

| SAL\_DIFF |

+ +

| salary deleted is 55000.00 |

+ +

1 row in set (0.00 sec)

Each operation (**INSERT**, **UPDATE**, **DELETE**) will trigger the respective trigger (**after\_insert\_salary\_difference**, **after\_update\_salary\_difference**, **lary\_difference**), which will display the salary change or difference associated with that operation.

**after\_delete\_sa**

By using separate triggers for each operation and utilizing the **OLD** and **NEW** keywords appropriately within the trigger bodies, you can effectively capture and handle

**CUSTOMERS**

**SALARY**

changes to the

column in the

table in MySQL. You can adjust the

trigger logic and message formatting as needed based on your specific requirements.

Program 5

Create cursor for Employee table & extract the values from the table. Declare the variables

,Open the cursor & extrct the values from the cursor. Close the cursor. Employee(E\_id, E\_name, Age, Salary)

Solution:

# Creating the Employee Table and insert few records

CREATE DATABASE COMPANY05;

USE COMPANY05;

CREATE TABLE Employee ( E\_id INT,

E\_name VARCHAR(255), Age INT,

Salary DECIMAL(10, 2)

);

INSERT INTO Employee (E\_id, E\_name, Age, Salary) VALUES

(1, 'Samarth', 30, 50000.00),

(2, 'Ramesh Kumar', 25, 45000.00),

(3, 'Seema Banu', 35, 62000.00),

(4, 'Dennis Anil', 28, 52000.00),

(5, 'Rehman Khan', 32, 58000.00);

# Create a Stored Procedure with Cursor

To create a cursor for the table, extract values using the cursor, and then

**Employee**

close the cursor in MySQL, you’ll need to use stored procedures that support cursor operations.

DELIMITER //

CREATE PROCEDURE fetch\_employee\_data() BEGIN

-- Declare variables to store cursor values DECLARE emp\_id INT;

DECLARE emp\_name VARCHAR(255);

DECLARE emp\_age INT;

DECLARE emp\_salary DECIMAL(10, 2);

-- Declare a cursor for the Employee table DECLARE emp\_cursor CURSOR FOR

SELECT E\_id, E\_name, Age, Salary FROM Employee;

-- Declare a continue handler for the cursor DECLARE CONTINUE HANDLER FOR NOT FOUND

SET @finished = 1;

-- Open the cursor OPEN emp\_cursor;

-- Initialize a variable to control cursor loop SET @finished = 0;

FETCH emp\_cursor INTO emp\_id, emp\_name, emp\_age, emp\_salary;

IF @finished = 1 THEN

SELECT CONCAT('Employee ID: ', emp\_id, ', Name: ', emp\_name, ', Age: ', emp\_age, ', Salary: ',

;

In this stored procedure (

):

**fetch\_employee\_data**

DELIMITER ;

END//

CLOSE emp\_cursor

-- Close the cursor

END LOOP;

emp\_salary) AS Employee\_Info;

-- Output or process each row (for demonstration, print the values)

END IF;

LEAVE cursor\_loop;

-- Check if no more rows to fetch

-- Fetch the next row from the cursor into variables

cursor\_loop: LOOP

-- Loop through the cursor results



We declare variables ( retrieved from the cursor.

,

,

,

) to store values



A cursor (

the **Employee** table.

) is declared to select **E\_id**, **E\_name**, **Age**, and

from



We declare a continue handler (**CONTINUE HANDLER**) for handle the end of cursor data.

condition to

* The cursor is opened (**OPEN emp\_cursor**), and a loop (**cursor\_loop**) is used to fetch each row from the cursor.
* We fetch values into the variables and process them within the loop (for demonstration, we print the values using a statement).





The loop continues until all rows are fetched (**@finished = 1**).

Finally, the cursor is closed (

).

**CLOSE emp\_cursor**

**emp\_salary**

**emp\_age**

**emp\_name**

**emp\_id**

**emp\_cursor**

**SELECT**

**NOT FOUND**

**Salary**

3. Execute the Stored Procedure

Once the stored procedure fetch and process data from the

is created, you can execute it to

table:

+ +

+ +

| Employee ID: 1, Name: Samarth, Age: 30, Salary: 50000.00 |

+ +

1 row in set (0.07 sec)

+ +

|

| Employee\_Info

mysql> CALL fetch\_employee\_data();

**Employee**

**fetch\_employee\_data**

| Employee\_Info |

+ +

| Employee ID: 2, Name: Ramesh Kumar, Age: 25, Salary: 45000.00 |

+ +

1 row in set (0.07 sec)

+ +

| Employee\_Info |

+ +

| Employee ID: 3, Name: Seema Banu, Age: 35, Salary: 62000.00 |

+ +

1 row in set (0.07 sec)

+ +

| Employee\_Info |

+ +

| Employee ID: 4, Name: Dennis Anil, Age: 28, Salary: 52000.00 |

+ +

1 row in set (0.07 sec)

+ +

| Employee\_Info |

+ +

| Employee ID: 5, Name: Rehman Khan, Age: 32, Salary: 58000.00 |

+ +

1 row in set (0.07 sec)

Query OK, 0 rows affected (0.07 sec)

1. The stored procedure (**emp\_id**, **emp\_name**, **emp\_age**,



A cursor ( and **Salary**.

declares variables

) to store values retrieved from the cursor.

) is declared for the table to select **E\_id**, **E\_name**, **Age**,



The cursor is opened (**OPEN emp\_cursor**), and the row from the cursor into the declared variables.

statement retrieves the first

1. A loop processes each row fetched by the cursor (**SQLSTATE() =**





**'00000'** checks for successful fetching).

Within the loop, you can perform operations or output the values of each row.

The

statement closes the cursor after processing all rows.

**CLOSE**

**emp\_salary**

**fetch\_employee\_data**

**WHILE**

**emp\_cursor**

**FETCH**

**Employee**

This example demonstrates how to create and use a cursor in MySQL to extract values from the table row by row. Adjust the cursor query and processing logic based on your table structure and desired operations.

**Employee**

Program 6

Write a PL/SQL block of code using parameterized Cursor, that will merge the data available in the newly created table N\_RollCall with the data available in the table O\_RollCall. If the

data in the first table already exist in the second table then that data should be skipped. Solution:

To accomplish this task in MySQL, we can use a stored procedure with a parameterized cursor to merge data from one table (**N\_RollCall**) into another table (**O\_RollCall**) while skipping existing data. We’ll iterate through the records

of and insert them into only if they do not already exist.

**O\_RollCall**

**N\_RollCall**

# Create the Tables

First, let’s create the

**N\_RollCall**

and

tables with similar structure:

**O\_RollCall**

CREATE DATABASE ROLLCALL;

USE ROLLCALL;

-- Create N\_RollCall table CREATE TABLE N\_RollCall (

student\_id INT PRIMARY KEY, student\_name VARCHAR(255), birth\_date DATE

);

-- Create O\_RollCall table with common data CREATE TABLE O\_RollCall (

student\_id INT PRIMARY KEY, student\_name VARCHAR(255), birth\_date DATE

);

# Add Sample Records to both tables

Let’s insert some sample data into the

**O\_RollCall**

table:

mysql> -- Insert common data into O\_RollCall

mysql> INSERT INTO O\_RollCall (student\_id, student\_name, birth\_date)

-> VALUES

-> (1, 'Shivanna', '1995-08-15'),

-> (3, 'Cheluva', '1990-12-10');

Query OK, 2 rows affected (0.17 sec) Records: 2 Duplicates: 0 Warnings: 0

Let’s insert some sample data into the common with **O\_RollCall**:

**N\_RollCall**

table, including records that are

mysql> -- Insert sample records into N\_RollCall

mysql> INSERT INTO N\_RollCall (student\_id, student\_name, birth\_date)

-> VALUES

-> (1, 'Shivanna', '1995-08-15'), -- Common record with O\_RollCall

-> (2, 'Bhadramma', '1998-03-22'),

-> (3, 'Cheluva', '1990-12-10'), -- Common record with O\_RollCall

-> (4, 'Devendra', '2000-05-18'),

-> (5, 'Eshwar', '1997-09-03');

Query OK, 5 rows affected (0.21 sec) Records: 5 Duplicates: 0 Warnings: 0

# Define the Stored Procedure

Next, let’s define the

**merge\_rollcall\_data**

stored procedure to merge records

from into **O\_RollCall**, skipping existing records:

**N\_RollCall**

DELIMITER //

CREATE PROCEDURE merge\_rollcall\_data() BEGIN

DECLARE done INT DEFAULT FALSE; DECLARE n\_id INT;

DECLARE n\_name VARCHAR(255);

DECLARE n\_birth\_date DATE;

-- Declare cursor for N\_RollCall table DECLARE n\_cursor CURSOR FOR

SELECT student\_id, student\_name, birth\_date FROM N\_RollCall;

-- Declare handler for cursor

DECLARE CONTINUE HANDLER FOR NOT FOUND

SET done = TRUE;

-- Open the cursor OPEN n\_cursor;

-- Start looping through cursor results cursor\_loop: LOOP

-- Fetch data from cursor into variables

FETCH n\_cursor INTO n\_id, n\_name, n\_birth\_date;

-- Check if no more rows to fetch IF done THEN

LEAVE cursor\_loop; END IF;

-- Check if the data already exists in O\_RollCall IF NOT EXISTS (

SELECT 1

FROM O\_RollCall

WHERE student\_id = n\_id

) THEN

-- Insert the record into O\_RollCall

INSERT INTO O\_RollCall (student\_id, student\_name, birth\_date) VALUES (n\_id, n\_name, n\_birth\_date);

END IF; END LOOP;

-- Close the cursor CLOSE n\_cursor;

|  |  |  |
| --- | --- | --- |
| END// |  | |
|  | | |
| DELIMITER ; | |  |

1. The stored procedure through the records of the

uses a cursor (**n\_cursor**) to iterate table.

1. Inside the cursor loop (**cursor\_loop**), each record (**n\_id**, **n\_name**, ) from is fetched and checked against the table.
2. If the record does not already exist in it is inserted into **O\_RollCall**.
3. The cursor loop continues until all records from processed.
4. The cursor is then closed (**CLOSE n\_cursor**).

(checked using **NOT EXISTS**),

have been

**O\_RollCall**

**n\_date**

**N\_RollCall**

**O\_RollCall**

**N\_RollCall**

**N\_RollCall**

**merge\_rollcall\_data**

# Execute the Stored Procedure

Finally, execute the stored procedure to merge records

**O\_RollCall**

**N\_RollCall**

**merge\_rollcall\_data**

from

into

while skipping existing records:

mysql> CALL merge\_rollcall\_data(); Query OK, 0 rows affected (0.87 sec)

# Verify Records in

**O\_RollCall**

After executing the procedure, verify the records in the

table to confirm

**O\_RollCall**

that new records from records have been skipped:

**N\_RollCall**

have been inserted, while existing common

mysql> -- Select all records from O\_RollCall mysql> SELECT \* FROM O\_RollCall;

+------------+--------------+ +

| student\_id | student\_name | birth\_date |

+------------+--------------+ +

| 1 | Shivanna | 1995-08-15 |<-- Common record, not duplicated

| 2 | Bhadramma | 1998-03-22 |<-- New record from N\_RollCall

| 3 | Cheluva | 1990-12-10 |<-- Common record, not duplicated

| 4 | Devendra | 2000-05-18 |<-- New record from N\_RollCall

| 5 | Eshwar | 1997-09-03 |<-- New record from N\_RollCall

+------------+--------------+ +

5 rows in set (0.00 sec)

.