EX: NO: 7 **DCL AND TCL COMMANDS**

AIM:

To create and execute complex transactions and realize DCL and TCL commands.

PERFORM THE FOLLOWING:

1. Create table.
2. Perform DCL and TCL commands.
3. Execute different user privileges.
4. Report the answers.

DCL (Data Control Language):

DCL includes commands such as GRANT and REVOKE which mainly deal with the rights, permissions, and other controls of the database system.

- **GRANT:** This command gives users access privileges to the database.
  GRANT SELECT, UPDATE ON MY_TABLE TO SOME_USER, ANOTHER_USER;
- **REVOKE:** This command withdraws the user's access privileges given by using the GRANT command.
  REVOKE SELECT, UPDATE ON MY_TABLE FROM USER1, USER2;

TCL (Transaction Control Language):

Transactions group a set of tasks into a single execution unit. Each transaction begins with a specific task and ends when all the tasks in the group successfully complete. If any of the tasks fail, the transaction fails. Therefore, a transaction has only two results: success or failure.

COMMIT: Commits a Transaction.

```
COMMIT;
```

ROLLBACK: Rollbacks a transaction in case of any error occurs.

```
ROLLBACK to SAVEPOINT_NAME;
```

SAVEPOINT: Sets a save point within a transaction.

```
SAVEPOINT SAVEPOINT_NAME;
```

TCL Exercises:

```
mysql> select * from employee;
+-------+--------+--------+------------+
| empid | name   | salary | dob        |
+-------+--------+--------+------------+
|  1001 | pragya |  10000 | 2001-02-28 |
|  1002 | anu    |  20000 | 2002-05-28 |
|  1003 | bob    |  30000 | 2000-01-18 |
+-------+--------+--------+------------+
3 rows in set (0.17 sec)
```

START TRANSACTION:

```
mysql> start transaction;
Query OK, 0 rows affected (0.00 sec)
```

CREATE SAVEPOINT:

```
mysql> savepoint s1;
Query OK, 0 rows affected (0.01 sec)

mysql> insert into employee values(1004,'vini',40000,'1999-06-13');
Query OK, 1 row affected (0.10 sec)
mysql> select * from employee;
+-------+--------+--------+------------+
| empid | name   | salary | dob        |
+-------+--------+--------+------------+
|  1001 | pragya |  10000 | 2001-02-28 |
|  1002 | anu    |  20000 | 2002-05-28 |
|  1003 | bob    |  30000 | 2000-01-18 |
|  1004 | vini   |  40000 | 1999-06-13 |
+-------+--------+--------+------------+
4 rows in set (0.17 sec)
```

CREATE updated SAVEPOINT:

```
mysql> savepoint upd;
Query OK, 0 rows affected (0.00 sec)
```

UPDATE TABLE

```
mysql> update employee set name='anupriya' where empid=1002;
Query OK, 1 row affected (0.02 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> select * from employee;
+-------+----------+--------+------------+
| empid | name     | salary | dob        |
+-------+----------+--------+------------+
|  1001 | pragya   |  10000 | 2001-02-28 |
|  1002 | anupriya |  20000 | 2002-05-28 |
|  1003 | bob      |  30000 | 2000-01-18 |
|  1004 | vini     |  40000 | 1999-06-13 |
+-------+----------+--------+------------+
4 rows in set (0.00 sec)
```

CREATE delete SAVEPOINT:

```
mysql> savepoint del;
Query OK, 0 rows affected (0.00 sec)
```

DELETE ROWS FROM TABLE:

```
mysql> delete from employee where empid=1003;
Query OK, 1 row affected (0.01 sec)
```

```
mysql> select * from employee;
+-------+----------+--------+------------+
| empid | name     | salary | dob        |
+-------+----------+--------+------------+
|  1001 | pragya   |  10000 | 2001-02-28 |
|  1002 | anupriya |  20000 | 2002-05-28 |
|  1004 | vini     |  40000 | 1999-06-13 |
+-------+----------+--------+------------+
3 rows in set (0.00 sec)
```

ROLLBACK TO DELETED SAVEPOINT:

```
mysql> rollback to del;
Query OK, 0 rows affected (0.00 sec)

mysql> select * from employee;
+-------+----------+--------+------------+
| empid | name     | salary | dob        |
+-------+----------+--------+------------+
|  1001 | pragya   |  10000 | 2001-02-28 |
|  1002 | anupriya |  20000 | 2002-05-28 |
|  1003 | bob      |  30000 | 2000-01-18 |
|  1004 | vini     |  40000 | 1999-06-13 |
+-------+----------+--------+------------+
4 rows in set (0.00 sec)
```

ROLLBACK TO UPDATE SAVEPOINT:

```
mysql> rollback to upd;
Query OK, 0 rows affected (0.00 sec)
```

```
mysql> select * from employee;
+-------+--------+--------+------------+
| empid | name   | salary | dob        |
+-------+--------+--------+------------+
|  1001 | pragya |  10000 | 2001-02-28 |
|  1002 | anu    |  20000 | 2002-05-28 |
|  1003 | bob    |  30000 | 2000-01-18 |
|  1004 | vini   |  40000 | 1999-06-13 |
+-------+--------+--------+------------+
```

ROLLBACK TO S1 SAVEPOINT:

```
mysql> rollback to s1;
Query OK, 0 rows affected (0.01 sec)

mysql> select * from employee;
+-------+--------+--------+------------+
| empid | name   | salary | dob        |
+-------+--------+--------+------------+
|  1001 | pragya |  10000 | 2001-02-28 |
|  1002 | anu    |  20000 | 2002-05-28 |
|  1003 | bob    |  30000 | 2000-01-18 |
+-------+--------+--------+------------+
3 rows in set (0.17 sec)
```

COMMIT TRANSACTION:

```
mysql> commit;
Query OK, 0 rows affected (0.16 sec)
```

DCL (Data Control Language)

Administrator Login:

C:\Windows\system32>cd C:\Program Files\MySQL\MySQL Server 8.0\bin

C:\Program Files\MySQL\MySQL Server 8.0\bin>mysql -u root -p

Enter password: *****(mysql)

Welcome to the MySQL monitor. Commands end with ; or \g.

Your MySQL connection id is 19

Server version: 8.0.32 MySQL Community Server - GPL

Create User:

mysql> create user 'anu'@'localhost' identified by 'dbms';

Query OK, 0 rows affected (0.12 sec)

mysql> select user(); **//Current User**

```
+----------------+
| user()         |
+----------------+
| root@localhost |
+----------------+
1 row in set (0.01 sec)
```

mysql> select user from mysql.user; **//already Login user**

```
+------------------+
| user             |
+------------------+
| mysql.infoschema |
| mysql.session    |
| mysql.sys        |
| myuser           |
| anu              |
+------------------+
9 rows in set (0.00 sec)
```

mysql> use employee; **// Create and use Database**

Database changed

Granting Privileges on user:

```
mysql> GRANT SELECT, INSERT, UPDATE ON Employee.emp TO 'subi'@'localhost';
Query OK, 0 rows affected (0.48 sec)
```

Revoking Privileges on user:

```
mysql>  REVOKE UPDATE ON employee.EMP FROM 'subi'@'localhost';
Query OK, 0 rows affected (0.12 sec)
```

Changing to new user in mysql command prompt:

```
mysql> system mysql -u subi -p
Enter password: ****
```

```
mysql> use employee;                    //use Database
Database changed
```

```
mysql> SHOW GRANTS FOR 'subi'@'localhost';       //Show all grants of user
+-----------------------------------------------------------------------+
| Grants for subi@localhost                                             |
+-----------------------------------------------------------------------+
| GRANT USAGE ON *.* TO `subi`@`localhost`                              |
| GRANT SELECT, INSERT, UPDATE ON `employee`.`emp` TO `subi`@`localhost` |
+-----------------------------------------------------------------------+
2 rows in set (0.00 sec)
```

```
mysql> select * from emp;               // accepting select  Persmission
+-------+--------+-------------+-----------+
| EMPNO | ENAME  | DESIGNATION | SALARY    |
+-------+--------+-------------+-----------+
|   100 | vinoth | xxx         | 223000.00 |
+-------+--------+-------------+-----------+
1 row in set (0.00 sec)
```

mysql> insert into emp values(111,'ram','ap',224353); // **accepting insert grant**

Query OK, 1 row affected (0.10 sec)

mysql> update emp set designation='analyst' where designation='xxx'; // **accepting update grant**

Query OK, 1 row affected (0.08 sec)

Rows matched: 1 Changed: 1 Warnings: 0

mysql> delete from emp where empno=101; // **denying delete grant**

ERROR 1142 (42000): DELETE command denied to user 'subi'@'localhost' for table 'emp'

// after revoking update permission denying update permission

mysql> update emp set empno=110 where empno=100;

ERROR 1142 (42000): UPDATE command denied to user 'subi'@'localhost' for table 'emp'

//To Drop user

```
mysql> drop user 'ajay'@'localhost';
Query OK, 0 rows affected (0.19 sec)

mysql> select user, host from mysql.user;
+------------------+-----------+
| user             | host      |
+------------------+-----------+
| mysql.infoschema | localhost |
| mysql.session    | localhost |
| mysql.sys        | localhost |
| root             | localhost |
+------------------+-----------+
4 rows in set (0.00 sec)
```

Result:

Thus the TCL and DCL commands were executed and verified successfully.

EX: NO: 8 **TRIGGERS**

AIM:

To develop and execute a Trigger for Before and After update, Delete, Insert operations on a table

Trigger:

A Trigger is a stored procedure that defines an action that the database automatically takes when some database-related event such as Insert, Update or Delete occur.

Types Of Triggers:

The various types of triggers are as follows

Before: Before triggers are fired before the DML statement is actually executed.
After: After triggers are fired after the DML statement has finished execution.
For each row: It specifies that the trigger fires once per row.
For each statement: This is the default trigger that is invoked. It specifies that the trigger fires once per statement.

Variables used in triggers

- :new
- :old

These two variables retain the new and old values of the column updated in the database. The values in these variables can be used in the database triggers for data manipulation.

Snytax:

CREATE TRIGGER <trigger name> <trigger time > <trigger event> ON <table name> FOR EACH ROW <trigger body>;

Using MySQL Triggers

Every trigger associated with a table has a unique name and function based on two factors:

1. **Time**. **BEFORE** or **AFTER** a specific row event.

2. **Event**. **INSERT**, **UPDATE** or **DELETE**.

Delete Triggers

To delete a trigger, use the **DROP TRIGGER** statement:

DROP TRIGGER<trigger name>;

Alternatively, use:

DROP TRIGGER IF EXISTS<trigger name>;

Procedure:

1. Create a table called *person* with *name* and *age* for columns.

```
CREATE TABLE person (name varchar(45), age int);
```

Insert sample data into the table:

```
INSERT INTO person VALUES ('Matthew', 25), ('Mark', 20);
```

Select the table to see the result:

```
SELECT * FROM person;
```

```
mysql> CREATE TABLE person (name varchar(45), age int);
Query OK, 0 rows affected (0.14 sec)

mysql> INSERT INTO person VALUES ('Matthew', 25), ('Mark', 20);
Query OK, 2 rows affected (0.00 sec)
Records: 2  Duplicates: 0  Warnings: 0

mysql> SELECT * FROM person;
+---------+------+
| name    | age  |
+---------+------+
| Matthew |   25 |
| Mark    |   20 |
+---------+------+
2 rows in set (0.00 sec)
```

2. Create a table called *average_age* with a column called *average*:

```
CREATE TABLE average_age (average double);
```

Insert the average age value into the table:

```
INSERT INTO average_age SELECT AVG(age) FROM person;
```

Select the table to see the result:

```
SELECT * FROM average_age;
```

```
mysql> CREATE TABLE average_age (average double);
Query OK, 0 rows affected (0.20 sec)

mysql> INSERT INTO average_age SELECT AVG(age) FROM person;
Query OK, 1 row affected (0.01 sec)
Records: 1  Duplicates: 0  Warnings: 0

mysql> SELECT * FROM average_age;
+---------+
| average |
+---------+
|    22.5 |
+---------+
1 row in set (0.00 sec)
```

3. Create a table called *person_archive* with *name*, *age*, and *time* columns:

```
CREATE TABLE person_archive (
name varchar(45),
age int,
time timestamp DEFAULT NOW());
```

```
mysql> CREATE TABLE person_archive (
    -> name varchar(45),
    -> age int,
    -> time timestamp DEFAULT NOW());
Query OK, 0 rows affected (0.28 sec)
```

Note: The function **NOW()** records the current time.

Create a BEFORE INSERT Trigger

To create a **BEFORE INSERT** trigger, use:

```
CREATE TRIGGER <trigger name> BEFORE INSERT
ON <table name>
FOR EACH ROW
<trigger body>;
```

The **BEFORE INSERT** trigger gives control over data modification before committing into a database table..

BEFORE INSERT Trigger Example

Create a **BEFORE INSERT** trigger to check the age value before inserting data into the *person* table:

```
delimiter //
CREATE TRIGGER person_bi BEFORE INSERT
ON person
FOR EACH ROW
IF NEW.age < 18 THEN
SIGNAL SQLSTATE '50001' SET MESSAGE_TEXT = 'Person must be older
than 18.';
END IF; //
delimiter ;
```

```
mysql> delimiter //
mysql> CREATE TRIGGER person_bi BEFORE INSERT
    -> ON person
    -> FOR EACH ROW
    -> IF NEW.age < 18 THEN
    -> SIGNAL SQLSTATE '50001' SET MESSAGE_TEXT = 'Person must be older than 18.';
    -> END IF; //
Query OK, 0 rows affected (0.17 sec)

mysql> delimiter ;
```

Inserting data activates the trigger and checks the value of *age* before committing the information:

```
INSERT INTO person VALUES ('John', 14);
```

```
INSERT INTO person VALUES ('John', 14);
1644 (50001): Person must be older than 18.
```

The console displays the descriptive error message. The data does not insert into the table because of the failed trigger check.

Create an AFTER INSERT Trigger

Create an **AFTER INSERT** trigger with:

```
CREATE TRIGGER <trigger name> AFTER INSERT
ON <table name>
FOR EACH ROW
<trigger body>;
```

The **AFTER INSERT** trigger is useful when the entered row generates a value needed to update another table.

AFTER INSERT Trigger Example

Inserting a new row into the *person* table does not automatically update the average in the *average_age* table. Create an **AFTER INSERT** trigger on the *person* table to update the *average_age* table after insert:

```
delimiter //
CREATE TRIGGER person_ai AFTER INSERT
ON person
FOR EACH ROW
UPDATE average_age SET average = (SELECT AVG(age) FROM person); //
delimiter ;
```

```
mysql> delimiter //
mysql> CREATE TRIGGER person_ai AFTER INSERT
    -> ON person
    -> FOR EACH ROW
    -> UPDATE average_age SET average = (SELECT AVG(age) FROM person); //
Query OK, 0 rows affected (0.30 sec)

mysql> delimiter ;
```

Inserting a new row into the *person* table activates the trigger:

```
INSERT INTO person VALUES ('John', 19);
```

```
mysql> INSERT INTO person VALUES ('John', 19);
Query OK, 1 row affected (0.00 sec)

mysql> SELECT * FROM person;
+---------+------+
| name    | age  |
+---------+------+
| Matthew |   25 |
| Mark    |   20 |
| John    |   19 |
+---------+------+
3 rows in set (0.00 sec)

mysql> SELECT * FROM average_age;
+--------------+
| average      |
+--------------+
| 21.333333333 |
+--------------+
1 row in set (0.00 sec)
```

The data successfully commits to the *person* table and updates the *average_age* table with the correct average value.

Create a BEFORE UPDATE Trigger

Make a **BEFORE UPDATE** trigger with:

```
CREATE TRIGGER <trigger name> BEFORE UPDATE
ON <table name>
FOR EACH ROW
<trigger body>;
```

The **BEFORE UPDATE** triggers go together with the **BEFORE INSERT** triggers. If any restrictions exist before inserting data, the limits should be there before updating as well.

BEFORE UPDATE Trigger Example

If there is an age restriction for the *person* table before inserting data, the age restriction should also exist before updating information. Without the **BEFORE UPDATE** trigger, the age check trigger is easy to avoid. Nothing restricts editing to a faulty value.

Add a **BEFORE UPDATE** trigger to the *person* table with the same body as the **BEFORE INSERT** trigger:

```
delimiter //
CREATE TRIGGER person_bu BEFORE UPDATE
ON person
FOR EACH ROW
IF NEW.age < 18 THEN
SIGNAL SQLSTATE '50002' SET MESSAGE_TEXT = 'Person must be older than 18.';
END IF; //
delimiter ;
```

```
mysql> delimiter ;
mysql> delimiter //
mysql> CREATE TRIGGER person_bu BEFORE UPDATE
    -> ON person
    -> FOR EACH ROW
    -> IF NEW.age < 18 THEN
    -> SIGNAL SQLSTATE '50002' SET MESSAGE_TEXT = 'Person must be older than 18';
    -> END IF; //
Query OK, 0 rows affected (0.36 sec)

mysql> delimiter ;
```

Updating an existing value activates the trigger check:

```
UPDATE person SET age = 17 WHERE name = 'John';
```

```
mysql> UPDATE person SET age = 17 WHERE name = 'John';
ERROR 1644 (50001): Person must be over the age of 18.
```

Updating the *age* to a value less than 18 displays the error message, and the information does not update.

Create an AFTER UPDATE Trigger

Use the following code block to create an **AFTER UPDATE** trigger:

```
CREATE TRIGGER <trigger name> AFTER UPDATE
ON <table name>
FOR EACH ROW
<trigger body>;
```

The **AFTER UPDATE** trigger helps keep track of committed changes to data. Most often, any changes after inserting information also happen after updating data.

AFTER UPDATE Trigger Example

Any successful updates to the *age* data in the table *person* should also update the intermediate average value calculated in the *average_age* table.

Create an **AFTER UPDATE** trigger to update the *average_age* table after updating a row in the *person* table:

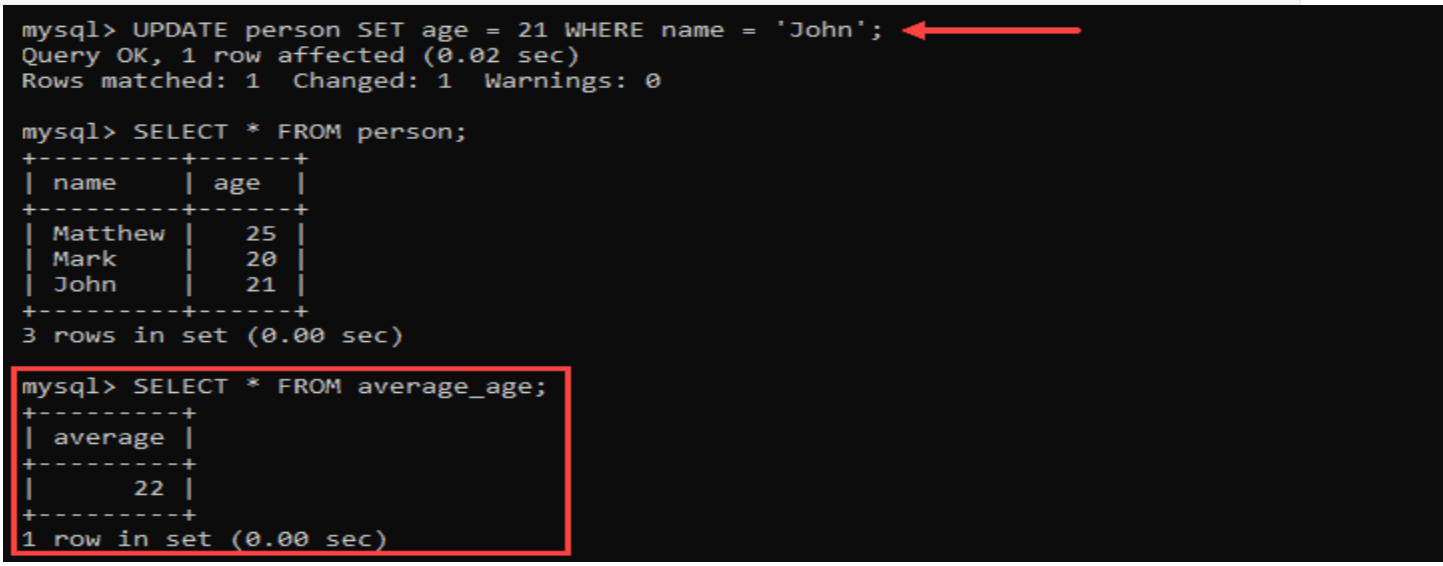
```
delimiter //
CREATE TRIGGER person_au AFTER UPDATE
ON person
FOR EACH ROW
UPDATE average_age SET average = (SELECT AVG(age) FROM person); //
delimiter ;
```

```
mysql> delimiter //
mysql> CREATE TRIGGER person_au AFTER UPDATE
    -> ON person
    -> FOR EACH ROW
    -> UPDATE average_age SET average = (SELECT AVG(age) FROM person); //
Query OK, 0 rows affected (0.93 sec)

mysql> delimiter ;
```

Updating existing data changes the value in the *person* table:

```
UPDATE person SET age = 21 WHERE name = 'John';
```



Updating the table *person* also updates the average in the *average_age* table.

Create a BEFORE DELETE Trigger

To create a **BEFORE DELETE** trigger, use:

```
CREATE TRIGGER <trigger name> BEFORE DELETE
ON <table name>
FOR EACH ROW
<trigger body>;
```

The **BEFORE DELETE** trigger is essential for security reasons. If a parent table has any children attached, the trigger helps block deletion and prevents orphaned tables. The trigger also allows archiving data before deletion.

BEFORE DELETE Trigger Example

Archive deleted data by creating a **BEFORE DELETE** trigger on the table *person* and insert the values into the *person_archive* table:

```
delimiter //
```

```
CREATE TRIGGER person_bd BEFORE DELETE
ON person
FOR EACH ROW
INSERT INTO person_archive (name, age)
VALUES (OLD.name, OLD.age); //
delimiter ;
```

```
mysql> delimiter //
mysql> CREATE TRIGGER person_bd BEFORE DELETE
    -> ON person
    -> FOR EACH ROW
    -> INSERT INTO person_archive (name, age)
    -> VALUES (OLD.name, OLD.age); //
Query OK, 0 rows affected (0.33 sec)

mysql> delimiter ;
```

Deleting data from the table *person* archives the data into the *person_archive* table before deleting:

```
DELETE FROM person WHERE name = 'John';
```

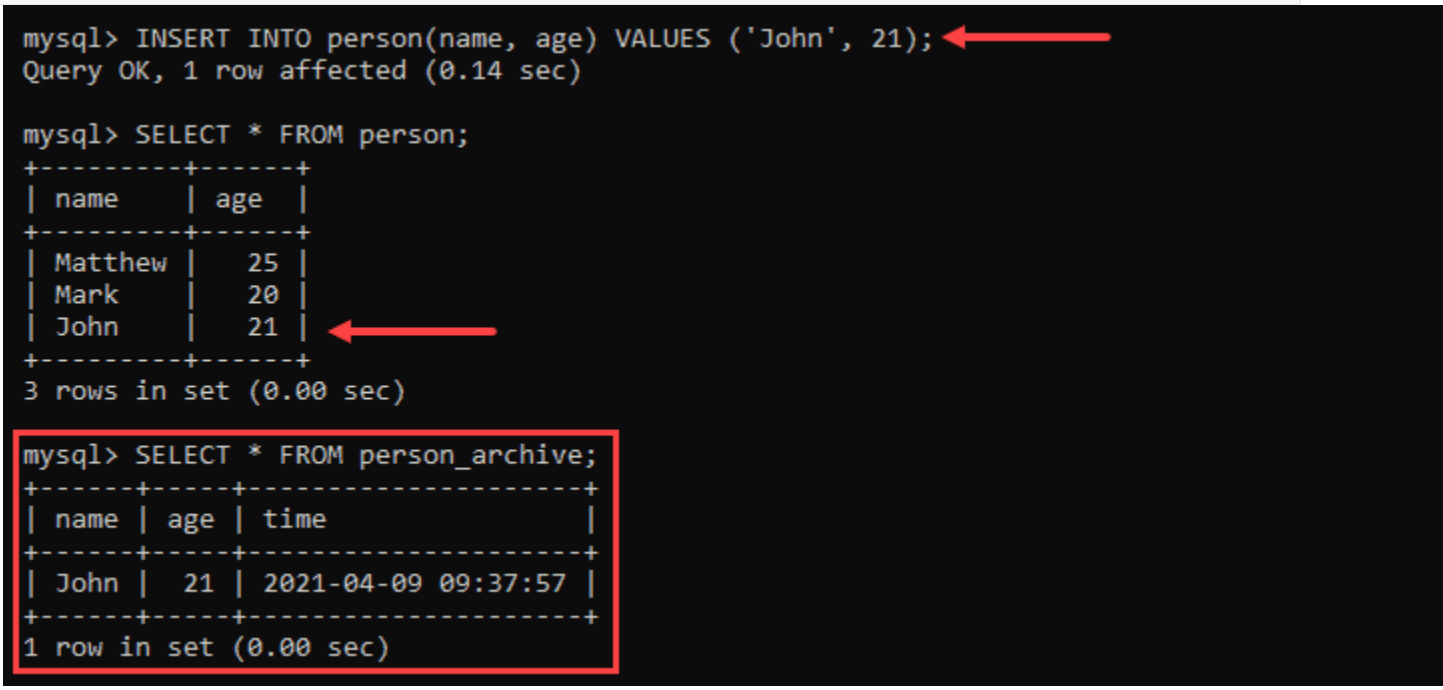
```
mysql> DELETE FROM person WHERE name = 'John';
Query OK, 1 row affected (0.03 sec)

mysql> SELECT * FROM person;
+---------+------+
| name    | age  |
+---------+------+
| Matthew |   25 |
| Mark    |   20 |
+---------+------+
2 rows in set (0.00 sec)

mysql> SELECT * FROM person_archive;
+------+-----+---------------------+
| name | age | time                |
+------+-----+---------------------+
| John |  21 | 2021-04-09 09:37:57 |
+------+-----+---------------------+
1 row in set (0.00 sec)
```

Inserting the value back into the *person* table keeps the log of the deleted data in the *person_archive* table:

```
INSERT INTO person VALUES ('John', 21);
```



The **BEFORE DELETE** trigger is useful for logging any table change attempts.

Create an AFTER DELETE Trigger

Make an **AFTER DELETE** trigger with:

```
CREATE TRIGGER <trigger name> AFTER DELETE
ON <table name>
FOR EACH ROW
<trigger body>;
```

The **AFTER DELETE** triggers maintain information updates that require the data row to disappear before making the updates.

AFTER DELETE Trigger Example

Create an **AFTER DELETE** trigger on the table *person* to update the *average_age* table with the new information:

```
delimiter //
CREATE TRIGGER person_ad AFTER DELETE
ON person
FOR EACH ROW
UPDATE average_age SET average = (SELECT AVG(person.age) FROM person); //
delimiter ;
```

```
mysql> delimiter //
mysql> CREATE TRIGGER person_ad AFTER DELETE
    -> ON person
    -> FOR EACH ROW
    -> UPDATE average_age SET average = (SELECT AVG(age) FROM person); //
Query OK, 0 rows affected (0.25 sec)

mysql> delimiter ;
```

Deleting a record from the table *person* updates the *average_age* table with the new average:

```
mysql> DELETE FROM person WHERE name = 'John';
Query OK, 1 row affected (0.01 sec)

mysql> SELECT * FROM person;
+---------+------+
| name    | age  |
+---------+------+
| Matthew |   25 |
| Mark    |   20 |
+---------+------+
2 rows in set (0.00 sec)

mysql> SELECT * FROM average_age;
+---------+
| average |
+---------+
|    22.5 |
+---------+
1 row in set (0.00 sec)
```

Without the **AFTER DELETE** trigger, the information does not update automatically.

Result:

Thus the Trigger for Before and After update, Delete, Insert operations were executed successfully.

.

EX: NO: 9 VIEWS AND INDEXES

AIM:

To create view and index for database tables with a large number of records.

.

VIEWS

OBJECTIVE:

- Views Helps to encapsulate complex query and make it reusable.
- Provides user security on each view - it depends on your data policy security.
- Using view to convert units - if you have a financial data in US currency, you can
- Create view to convert them into Euro for viewing in Euro currency.

PROCEDURE

STEP 1: Start
STEP 2: Create the table with its essential attributes.
STEP 3: Insert attribute values into the table.

STEP 4: Create the view from the above created table.
STEP 5: Execute different Commands and extract information from the View.
STEP 6: Stop

SQL COMMANDS

1. COMMAND NAME: **CREATE VIEW**
COMMAND DESCRIPTION: **CREATE VIEW** command is used to define a view.

2. COMMAND NAME: **INSERT IN VIEW**
COMMAND DESCRIPTION: **INSERT** command is used to insert a new row into the view.

3. COMMAND NAME: **DELETE IN VIEW**
COMMAND DESCRIPTION: **DELETE** command is used to delete a row from the view.

4. COMMAND NAME: **UPDATE OF VIEW**
COMMAND DESCRIPTION: **UPDATE** command is used to change a value in a tuple without changing all values in the tuple.

5. COMMAND NAME: **DROP OF VIEW**
COMMAND DESCRIPTION: **DROP** command is used to drop the view table

COMMANDS EXECUTION

CREATION OF TABLE

mysql> create table employee(empid integer,emp_name varchar(20),deptname varchar(10),dept_no integer,DOJ date);
Query OK, 0 rows affected (0.39 sec)
5 rows in set (0.00 sec)

DESCRIPTION OF TABLE:

```
mysql> DESC EMPLOYEE;
+-----------+-------------+------+-----+---------+-------+
| Field     | Type        | Null | Key | Default | Extra |
+-----------+-------------+------+-----+---------+-------+
| emp_name  | varchar(20) | YES  |     | NULL    |       |
| emp_id    | int         | YES  |     | NULL    |       |
| dept_name | varchar(10) | YES  |     | NULL    |       |
| dept_no   | int         | YES  |     | NULL    |       |
| DOJ       | date        | YES  |     | NULL    |       |
+-----------+-------------+------+-----+---------+-------+
```

DISPLAY VIEW:

mysql>SELECT * FROM EMPLOYEE;

```
+----------+--------+------------+---------+------------+
| emp_name | emp_id | dept_name  | dept_no | DOJ        |
+----------+--------+------------+---------+------------+
| ravi     |    124 | project    |       3 | 2020-05-03 |
| vijay    |    110 | developing |       2 | 2020-07-23 |
| jay      |    112 | HR         |       1 | 2018-07-01 |
+----------+--------+------------+---------+------------+
3 rows in set (0.00 sec)
```

SYNTAX FOR CREATION OF VIEW

MYSQL> CREATE <VIEW> <VIEW NAME> AS SELECT<COLUMN_NAME_1>, <COLUMN_NAME_2> FROM <TABLE NAME>;

CREATION OF VIEW

mysql> create view empview as select emp_name,empid,deptname,dept_no from employee;
Query OK, 0 rows affected (0.09 sec)

DESCRIPTION OF VIEW

MYSQL> DESC EMPVIEW;

```
+-----------+-------------+------+-----+---------+-------+
| Field     | Type        | Null | Key | Default | Extra |
+-----------+-------------+------+-----+---------+-------+
| emp_name  | varchar(20) | YES  |     | NULL    |       |
| emp_id    | int         | YES  |     | NULL    |       |
| dept_name | varchar(10) | YES  |     | NULL    |       |
| dept_no   | int         | YES  |     | NULL    |       |
+-----------+-------------+------+-----+---------+-------+
4 rows in set (0.00 sec)
```

DISPLAY VIEW:

SQL> SELECT * FROM EMPVIEW;

```
+----------+--------+------------+---------+
| emp_name | emp_id | dept_name  | dept_no |
+----------+--------+------------+---------+
| ravi     |    124 | project    |       3 |
| vijay    |    110 | developing |       2 |
| jay      |    112 | HR         |       1 |
+----------+--------+------------+---------+
3 rows in set (0.00 sec)
```

INSERTION INTO VIEW

INSERT STATEMENT:
SYNTAX:

MYSQL>INSERT INTO <VIEW_NAME> (COLUMN NAME1,...)VALUES(VALUE1,....);

mysql> INSERT INTO EMPVIEW VALUES('SRI',120,'HR',1);
Query OK, 1 row affected (0.04 sec)
SQL> SELECT * FROM EMPVIEW;

```
+----------+--------+------------+---------+
| emp_name | emp_id | dept_name  | dept_no |
+----------+--------+------------+---------+
| ravi     |    124 | project    |       3 |
| vijay    |    110 | developing |       2 |
| jay      |    112 | HR         |       1 |
| SRI      |    120 | HR         |       1 |
+----------+--------+------------+---------+
4 rows in set (0.00 sec)
```

DELETION OF VIEW:

DELETE STATEMENT:

SYNTAX:

SQL> DELETE <VIEW_NMAE>WHERE <COLUMN NMAE> ='VALUE';
mysql> DELETE FROM EMPVIEW WHERE EMP_NAME='SRI';
Query OK, 1 row affected (0.30 sec)
SQL> SELECT * FROM EMPVIEW;

```
+----------+--------+------------+---------+
| emp_name | emp_id | dept_name  | dept_no |
+----------+--------+------------+---------+
| ravi     |    124 | project    |       3 |
| vijay    |    110 | developing |       2 |
| jay      |    112 | HR         |       1 |
+----------+--------+------------+---------+
3 rows in set (0.01 sec)
```

UPDATE STATEMENT:

SYNTAX:

MYSQL>UPDATE <VIEW_NAME> SET< COLUMN NAME> = <COLUMN NAME>
<VIEW> WHERE <COLUMNNAME>=VALUE;

mysql> UPDATE EMPVIEW SET EMP_NAME='SRIVIJAY' WHERE EMP_ID=110;
Query OK, 1 row affected (0.05 sec)
Rows matched: 1 Changed: 1 Warnings: 0

```
mysql> SELECT * FROM EMPVIEW;
+----------+--------+------------+---------+
| emp_name | emp_id | dept_name  | dept_no |
+----------+--------+------------+---------+
| ravi     |    124 | project    |       3 |
| SRIVIJAY |    110 | developing |       2 |
| jay      |    112 | HR         |       1 |
+----------+--------+------------+---------+
3 rows in set (0.00 sec)
```

DROP A VIEW:

SYNTAX:
MYSQL> DROP VIEW <VIEW_NAME>

EXAMPLE:

mysql> DROP VIEW EMPVIEW;

Query OK, 0 rows affected (0.08 sec)

INDEX

- The CREATE INDEX statement is used to create indexes in tables.
- Indexes allow the database application to find data fast; without reading the whole table.
- An index can be created in a table to find data more quickly and efficiently.
- The users cannot see the indexes, they are just used to speed up searches/queries.

Note: Updating a table with indexes takes more time than updating a table without (because the indexes also need an update). So you should only create indexes on columns (and tables) that will be frequently searched against.

CREATE INDEX

Syntax

Creates an index on a table. Duplicate values are allowed:

CREATE INDEX index_name ON table_name (column_name);

mysql> CREATE TABLE PERSON(FIRSTNAME VARCHAR(20),LASTNAME VARCHAR(20),DEPTNO INTEGER,DEPTNAME VARCHAR(20));

Query OK, 0 rows affected (0.21 sec)

mysql> CREATE INDEX IDX_PNAME ON PERSON(LASTNAME,FIRSTNAME);
Query OK, 0 rows affected (0.30 sec)
Records: 0 Duplicates: 0 Warnings: 0

To access the query plan for the SELECT query, execute the following:

After executing the above statement, the index is created successfully. Now, run the below statement to see how MySQL internally performs this query.

mysql> EXPLAIN SELECT FIRSTNAME,LASTNAME FROM PERSON WHERE DEPTNAME='CSE';

```
mysql> EXPLAIN SELECT FIRSTNAME,LASTNAME FROM PERSON WHERE DEPTNAME='CSE';
+----+-------------+--------+------------+------+---------------+------+---------+------+------+----------+-------------+
| id | select_type | table  | partitions | type | possible_keys | key  | key_len | ref  | rows | filtered | Extra       |
+----+-------------+--------+------------+------+---------------+------+---------+------+------+----------+-------------+
|  1 | SIMPLE      | PERSON | NULL       | ALL  | NULL          | NULL | NULL    | NULL |    2 |    50.00 | Using where |
+----+-------------+--------+------------+------+---------------+------+---------+------+------+----------+-------------+
1 row in set, 1 warning (0.14 sec)
```

SHOW INDEXES:

```
+--------+------------+----------+--------------+-------------+-----------+-------------+----------+--------+------+------------+---------+---------------+---------+------------+
| Table  | Non_unique | Key_name | Seq_in_index | Column_name | Collation | Cardinality | Sub_part | Packed | Null | Index_type | Comment | Index_comment | Visible | Expression |
+--------+------------+----------+--------------+-------------+-----------+-------------+----------+--------+------+------------+---------+---------------+---------+------------+
| person |          1 | IDX_PNAME |            1 | LASTNAME    | A         |           2 |     NULL |   NULL | YES  | BTREE      |         |               | YES     | NULL       |
| person |          1 | IDX_PNAME |            2 | FIRSTNAME   | A         |           2 |     NULL |   NULL | YES  | BTREE      |         |               | YES     | NULL       |
+--------+------------+----------+--------------+-------------+-----------+-------------+----------+--------+------+------------+---------+---------------+---------+------------+
2 rows in set (0.04 sec)
```

DROPPING INDEXES

mysql> DROP INDEX IDX_PNAME ON PERSON;

Query OK, 0 rows affected (0.12 sec)

Records: 0 Duplicates: 0 Warnings: 0

mysql> show indexes from person;

Empty set (0.00 sec)

Result:

Thus the views and indexes created on person table was executed successfully.

EX: NO: 10 CREATION OF XML DATABASE AND VALIDATION USING XML SCHEMA

AIM:

To Create a XML database and validation using XML schema.

Procedure:

1. Search liquid xml validator from google.

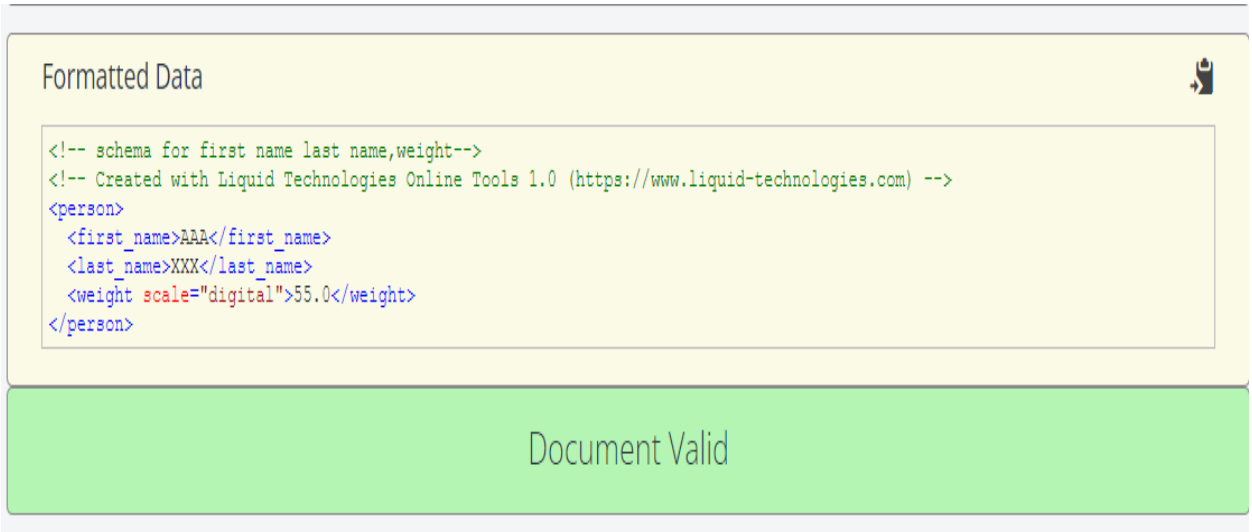
```
  <!-- schema for first name last name,weight-->
<person>
  <first_name>AAA</first_name>
  <last_name>XXX</last_name>
  <weight scale="digital">55.0</weight>
</person>
```

```
<?xml version="1.0">
```

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XML Schema">
  <xs:element name="person">
            <xs:complexType>
                    <xs:sequence>
                              <xs:element name="first_name" type="xs:string/>
                              <xs:element name="last_name" type="xs:string/>
                              <xs:element name="weight">
                                        <xs:complexType>
                                        <xs:simpleContent>
                                                  <xs:extension base="xs:float">
                                                            <xs:attribute name="scale"
type="xs:string"/>
                                                  </xs:extension>
                                        </xs:simpleContent>
                              </xs:complexType>
                              </xs:element>
                    </xs:sequence>
            </xs:complexType>
  </xs:element>
</xs:schema>
```

2. Open xml validator link from google.

2.Select xml formatter.



3. Select XML TO XSD
   Copy the formatted text there.

4. SELECT XML VALIDATOR XSD

Copy the generated text.

5. **Last step select** XML Validator (XSD)

6. Copy first few lines of code in first box and the generated code in the second box.

XML data to validate

```xml
<!-- schema for first name last name,weight-->
<person>
    <first_name>AAA</first_name>
    <last_name>XXX</last_name>
    <weight scale="digital">55.0</weight>
</person>

```

XML schema (XSD) data

```xml
<?xml version="1.0" encoding="utf-8"?>
<!-- Created with Liquid Technologies Online Tools 1.0 (https://www.liquid-technologies.com) -->
<xs:schema attributeFormDefault="unqualified" elementFormDefault="qualified" xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="person">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="first_name" type="xs:string" />
        <xs:element name="last_name" type="xs:string" />
        <xs:element name="weight">
          <xs:complexType>
            <xs:simpleContent>
              <xs:extension base="xs:decimal">
                <xs:attribute name="scale" type="xs:string" use="required" />
```

Validate

Document Valid

RESULT:

Thus the creation of XML database and validation using XML schema was executed successfully.