<u>Exp No:-1</u>

<u>Aim:-</u> SQL Data Definition Language Commands on sample exercise

SQL data definition:
Create command
Alter command
Drop
Truncate

<u>Theory</u>:-

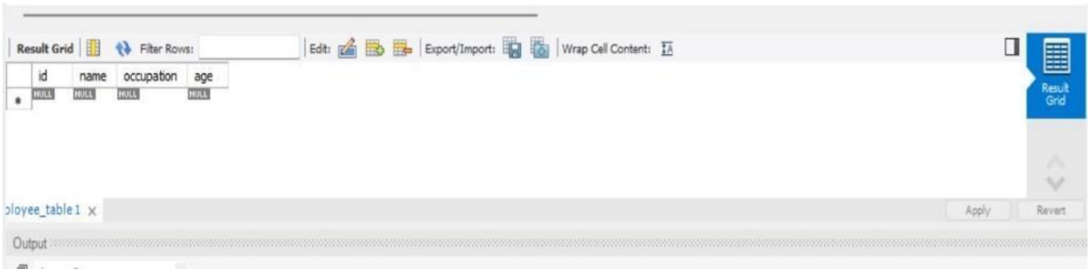
DDL changes the structure of the table like creating a table, deleting a table, altering a table, etc.
All the command of DDL are auto-committed that means it permanently save all the changes in the database.

a. CREATE: It is used to create a new table in the database.

QUERY:-

```
CREATE TABLE employee_table( id int
   NOT NULL AUTO_INCREMENT, name
   varchar(45) NOT NULL, occupation
   varchar(35) NOT NULL, age
   int NOT NULL, PRIMARY
   KEY (id)
);
```

<u>OUTPUT</u>:-

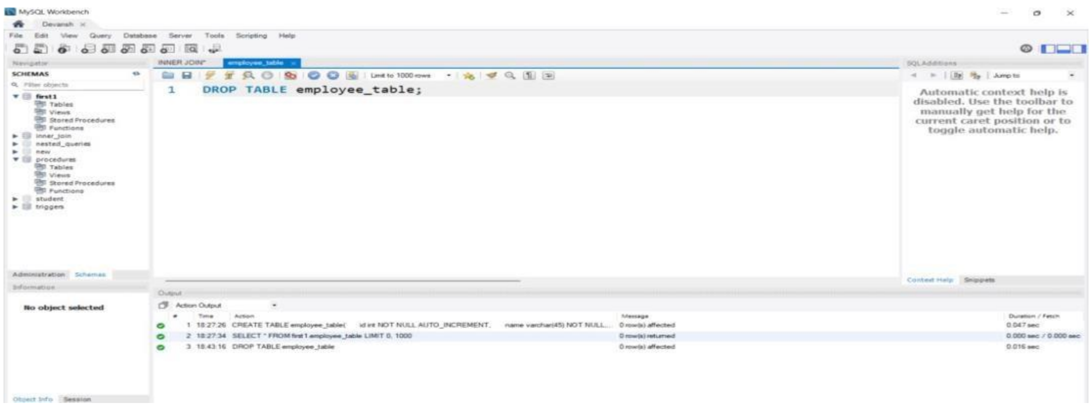


id	name	occupation	age
NULL	NULL	NULL	NULL

B .DROP: It is used to delete both the structure and record stored in the table.

QUERY:

```
DROP TABLE employee_table;
```



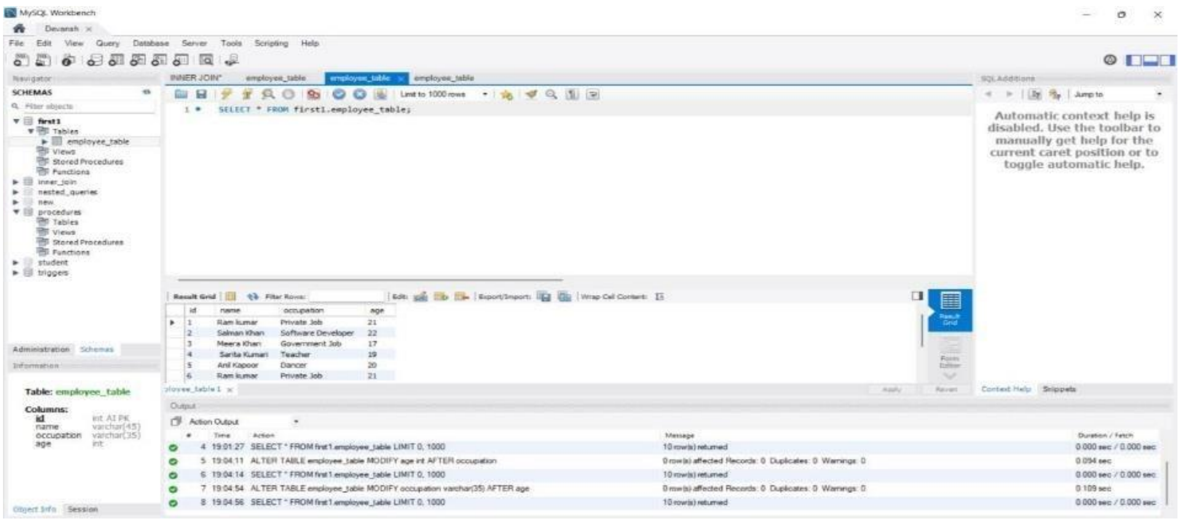
```
DROP TABLE employee_table;
```

```
#  Time      Action                                                                                   Message              Duration / Fetch
1  18:27:26  CREATE TABLE employee_table(    id int NOT NULL AUTO_INCREMENT,    name varchar(45) NOT NULL...  0 row(s) affected    0.047 sec
2  18:27:34  SELECT * FROM first1.employee_table LIMIT 0, 1000                                        0 row(s) returned    0.000 sec / 0.000 sec
3  18:43:16  DROP TABLE employee_table                                                                0 row(s) affected    0.016 sec
```

ALTER : It is used to alter the structure of the database. This change could be either to modify the characteristics of an existing attribute or probably to add a new attribute.

QUERIES:

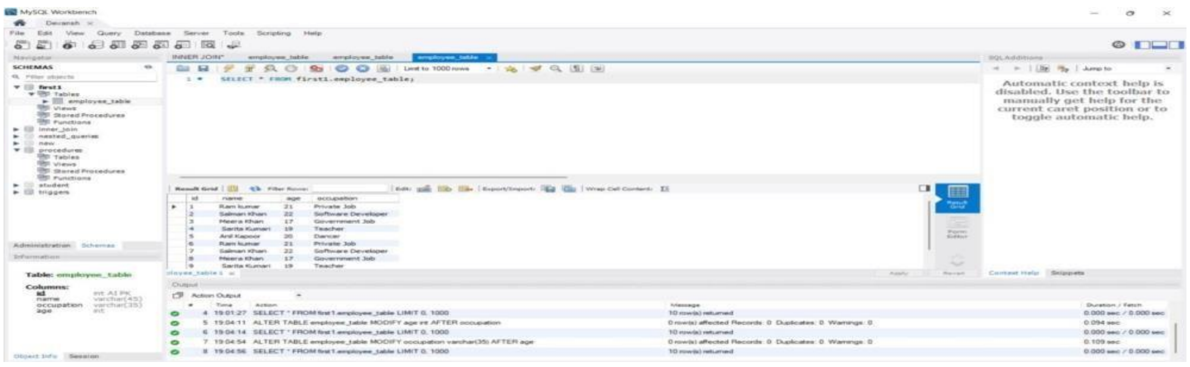
ALTER TABLE employee_table
MODIFY occupation varchar(35)
AFTER age;



```
SELECT * FROM first1.employee_table;
```

id	name	occupation	age
1	Ram kumar	Private Job	21
2	Salman Khan	Software Developer	22
3	Meera Khan	Government Job	17
4	Sarita Kumari	Teacher	19
5	Anil Kapoor	Dancer	20
6	Ram kumar	Private Job	21

```
#  Time      Action                                                              Message                                                   Duration / Fetch
4  19:01:27  SELECT * FROM first1.employee_table LIMIT 0, 1000                   10 row(s) returned                                        0.000 sec / 0.000 sec
5  19:04:11  ALTER TABLE employee_table MODIFY age int AFTER occupation         0 row(s) affected Records: 0  Duplicates: 0  Warnings: 0   0.094 sec
6  19:04:14  SELECT * FROM first1.employee_table LIMIT 0, 1000                   10 row(s) returned                                        0.000 sec / 0.000 sec
7  19:04:54  ALTER TABLE employee_table MODIFY occupation varchar(35) AFTER age  0 row(s) affected Records: 0  Duplicates: 0  Warnings: 0   0.109 sec
8  19:04:56  SELECT * FROM first1.employee_table LIMIT 0, 1000                   10 row(s) returned                                        0.000 sec / 0.000 sec
```



```
SELECT * FROM first1.employee_table;
```

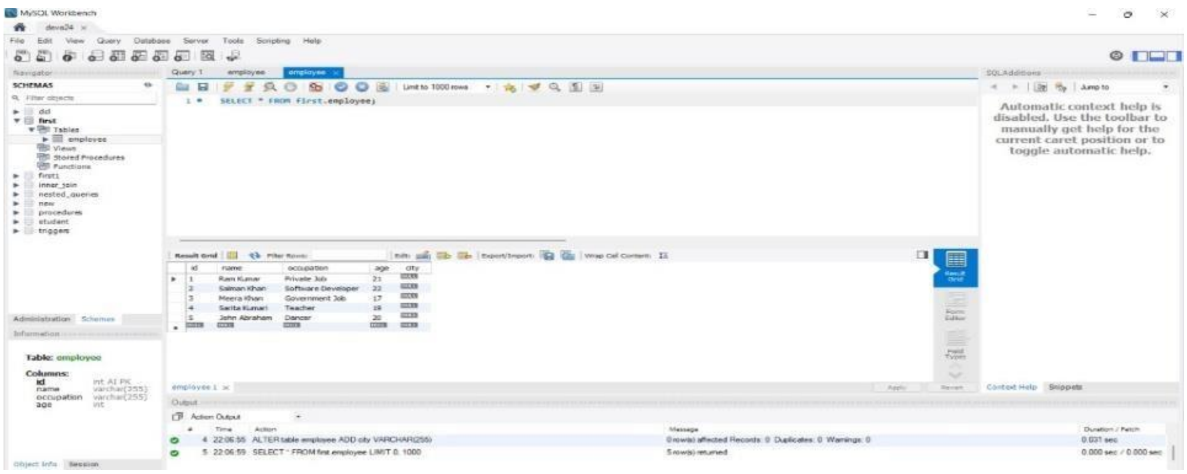
id	name	age	occupation
1	Ram kumar	21	Private Job
2	Salman Khan	22	Software Developer
3	Meera Khan	17	Government Job
4	Sarita Kumari	19	Teacher
5	Anil Kapoor	20	Dancer
6	Ram kumar	21	Private Job
7	Salman Khan	22	Software Developer
8	Meera Khan	17	Government Job
9	Sarita Kumari	19	Teacher

```
#  Time      Action                                                              Message                                                   Duration / Fetch
4  19:01:27  SELECT * FROM first1.employee_table LIMIT 0, 1000                   10 row(s) returned                                        0.000 sec / 0.000 sec
5  19:04:11  ALTER TABLE employee_table MODIFY age int AFTER occupation         0 row(s) affected Records: 0  Duplicates: 0  Warnings: 0   0.094 sec
6  19:04:14  SELECT * FROM first1.employee_table LIMIT 0, 1000                   10 row(s) returned                                        0.000 sec / 0.000 sec
7  19:04:54  ALTER TABLE employee_table MODIFY occupation varchar(35) AFTER age  0 row(s) affected Records: 0  Duplicates: 0  Warnings: 0   0.109 sec
8  19:04:56  SELECT * FROM first1.employee_table LIMIT 0, 1000                   10 row(s) returned                                        0.000 sec / 0.000 sec
```

ALTER TABLE-ADD COLUMN

QUERIES:

```
ALTER table employee
ADD city
VARCHAR(255);
```



```
SELECT * FROM first.employee;
```

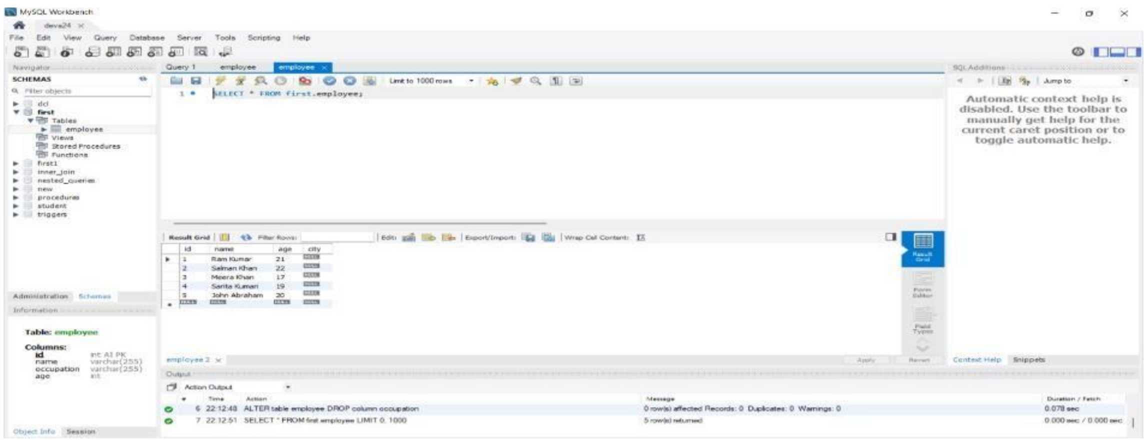
id	name	occupation	age	city
1	Ram Kumar	Private Job	21	NULL
2	Salman Khan	Software Developer	22	NULL
3	Meera Khan	Government Job	17	NULL
4	Sarita Kumari	Teacher	19	NULL
5	John Abraham	Dancer	20	NULL
NULL	NULL	NULL	NULL	NULL

```
#  Time      Action                                    Message                                                    Duration / Fetch
4  22:06:55  ALTER table employee ADD city VARCHAR(255) 0 row(s) affected Records: 0  Duplicates: 0  Warnings: 0  0.031 sec
5  22:06:59  SELECT * FROM first.employee LIMIT 0, 1000 5 row(s) returned                                         0.000 sec / 0.000 sec
```

ALTER TABLE-DROP COLUMN:

QUERIES:

```
ALTER table employee
DROP column
occupation;
```



```
SELECT * FROM first.employee;
```

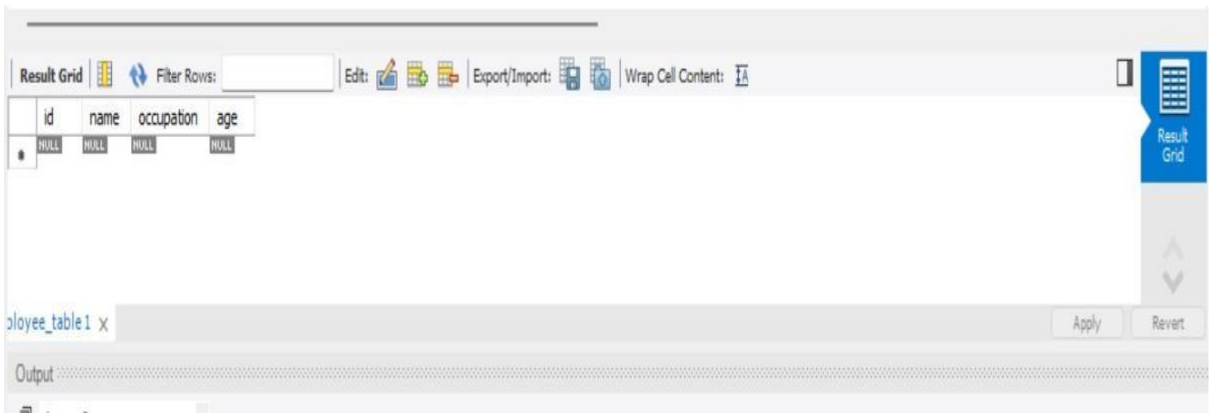
id	name	age	city
1	Ram Kumar	21	NULL
2	Salman Khan	22	NULL
3	Meera Khan	17	NULL
4	Sarita Kumari	19	NULL
5	John Abraham	20	NULL
NULL	NULL	NULL	NULL

```
#  Time      Action                                        Message                                                    Duration / Fetch
6  22:12:48  ALTER table employee DROP column occupation   0 row(s) affected Records: 0  Duplicates: 0  Warnings: 0  0.078 sec
7  22:12:51  SELECT * FROM first.employee LIMIT 0, 1000    5 row(s) returned                                         0.000 sec / 0.000 sec
```

TRUNCATE: It is used to delete all the rows from the table and free the space containing the table.

QUERIES:

```
TRUNCATE TABLE employee_table;
```



```
id    name  occupation  age
NULL  NULL  NULL        NULL
```

ployee_table1

RESULT: The DDL commands in RDBMS are implemented successfully and output was verified.

Exp No:-2

Data Manipulation Language (DML)

Aim:- Execute DML Commands

Theory:- DML commands are the most frequently used SQL commands and is used to query and manipulate the existing database objects. Some of the commands are Insert, Select, Update, Delete.

Insert Command: This is used to add one or more rows to a table. The values are separated by commas and the data types char and date are enclosed in apostrophes. The values must be entered in the same order as they are defined.

Select Command: It is used to retrieve information from the table. It is generally referred to as querying the table. We can either display all columns in a table or only specify column from the table.

Update Command: It is used to alter the column values in a table. A single column may be updated or more than one column could be updated.

Delete command: After inserting row in a table we can also delete them if required. The delete command consists of a from clause followed by an optional where clause.

PROCEDURE:-

1. Goto Start and click on programs.
2. Point to Oracle, then Ora81 and click on SQLPlus.
3. Then a pop-up menu will appear to authenticate you as user. Type 'scott' as username, 'tiger' as password and 'orcl' as the host.
4. Now SQLPlus window will get opened for working with various SQL commands in it.

SYNTAX AND DEFINITION:

DML commands:

1. Command: SELECT
   Syntax:
   ```
   SELECT[ALL | DISTINCT] select list
   FROM table_name1[,…table_nameN]
   [JOIN join_condition]
   [WHERE search_condition]
   ```

2. Command: INSERT
   Syntax:
   ```
   INSERT INTO[[database_name]owner]{table_name|view_name}
    [(column_list)]{[DEFAULT]VALUES|VALUES(value[…])|SELECT
   Statement}
   ```

3. Command: UPDATE

Syntax:

```
    UPDATE table_name
    SET     column_name=expression     [,…n]
WHERE search_condition
```

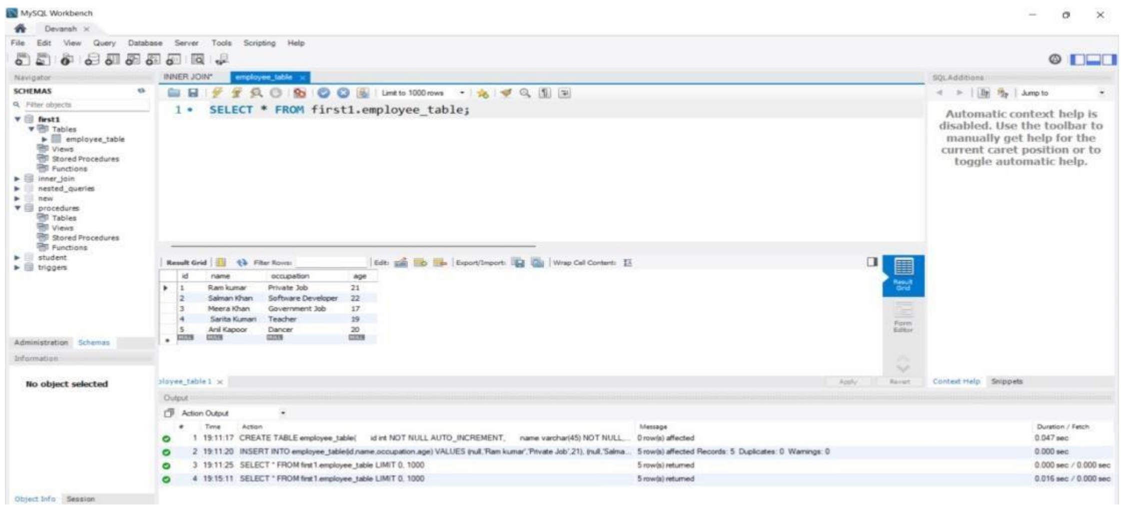
4. Command: DELETE

   Syntax:

   ```
   DELETE[FROM] table_name WHERE search_condition]
   ```

1. INSERT Command:-

```
INSERT INTO
employee_table(id,name,occupation,age) VALUES
(null,'Ram kumar','Private Job',21),
(null,'Salman Khan','Software
Developer',22), (null,'Meera
Khan','Government Job',17), (null,' Sarita
Kumari','Teacher',19), (null,'Anil
Kapoor','Dancer',20);
```



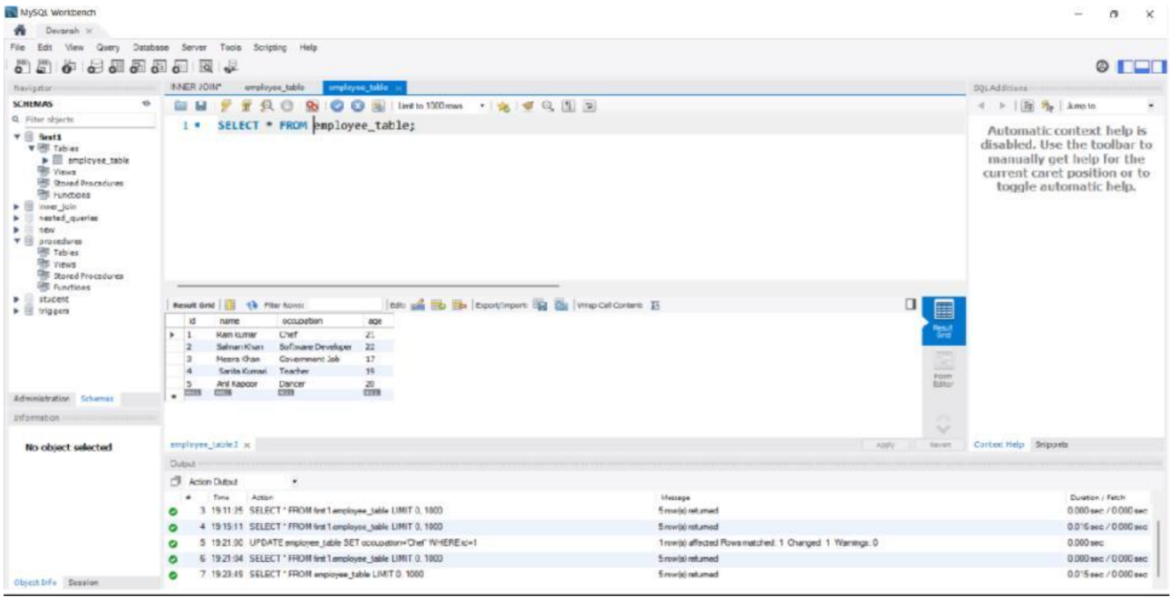
```
SELECT * FROM first1.employee_table;
```

id	name	occupation	age
1	Ram kumar	Private Job	21
2	Salman Khan	Software Developer	22
3	Meera Khan	Government Job	17
4	Sarita Kumari	Teacher	19
5	Anil Kapoor	Dancer	20
NULL	NULL	NULL	NULL

```
#  Time      Action                                                                                                     Message                                                 Duration / Fetch
1  19:11:17  CREATE TABLE employee_table(     id int NOT NULL AUTO_INCREMENT,     name varchar(45) NOT NULL...            0 row(s) affected                                       0.047 sec
2  19:11:20  INSERT INTO employee_table(id,name,occupation,age) VALUES (null,'Ram kumar','Private Job',21), (null,'Salma...  5 row(s) affected Records: 5  Duplicates: 0  Warnings: 0  0.000 sec
3  19:11:25  SELECT * FROM first1.employee_table LIMIT 0, 1000                                                            5 row(s) returned                                       0.000 sec / 0.000 sec
4  19:15:11  SELECT * FROM first1.employee_table LIMIT 0, 1000                                                            5 row(s) returned                                       0.016 sec / 0.000 sec
```

2. Update Command:-

```
UPDATE
employee_table
occupation='Chef'
WHERE id=1;
```



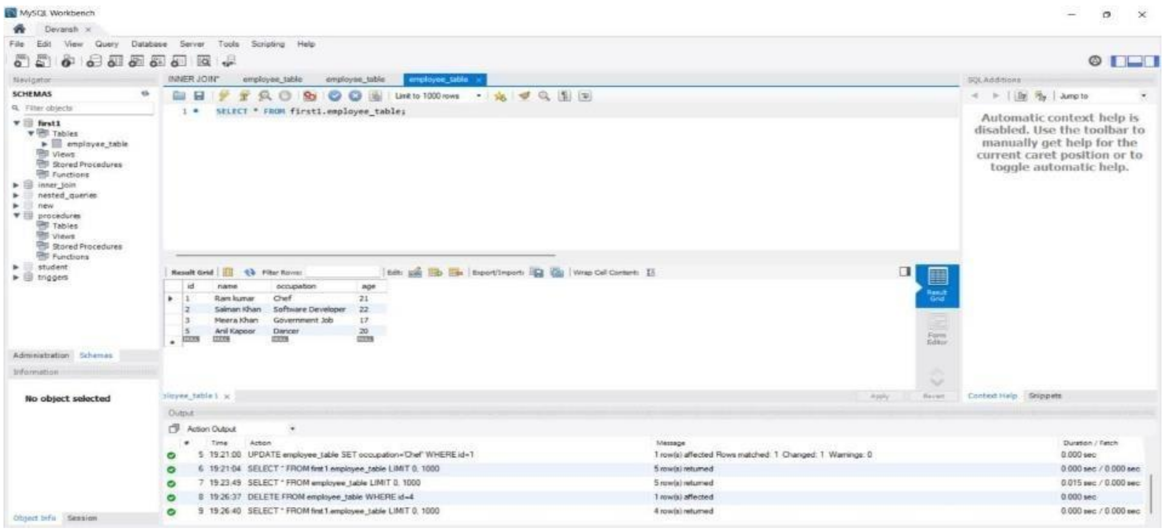
```
SELECT * FROM employee_table;
```

id	name	occupation	age
1	Ram kumar	Chef	21
2	Salman Khan	Software Developer	22
3	Meera Khan	Government Job	17
4	Sarita Kumari	Teacher	19
5	Anil Kapoor	Dancer	20
NULL	NULL	NULL	NULL

```
#  Time      Action                                                          Message                                                    Duration / Fetch
3  19:11:25  SELECT * FROM first1.employee_table LIMIT 0, 1000               5 row(s) returned                                          0.000 sec / 0.000 sec
4  19:15:11  SELECT * FROM first1.employee_table LIMIT 0, 1000               5 row(s) returned                                          0.016 sec / 0.000 sec
5  19:21:00  UPDATE employee_table SET occupation='Chef' WHERE id=1          1 row(s) affected Rows matched: 1  Changed: 1  Warnings: 0 0.000 sec
6  19:21:04  SELECT * FROM first1.employee_table LIMIT 0, 1000               5 row(s) returned                                          0.000 sec / 0.000 sec
7  19:23:49  SELECT * FROM employee_table LIMIT 0, 1000                      5 row(s) returned                                          0.015 sec / 0.000 sec
```

3. Delete command:-

DELETE FROM
employee_table WHERE
id=4;



```
SELECT * FROM first1.employee_table;
```

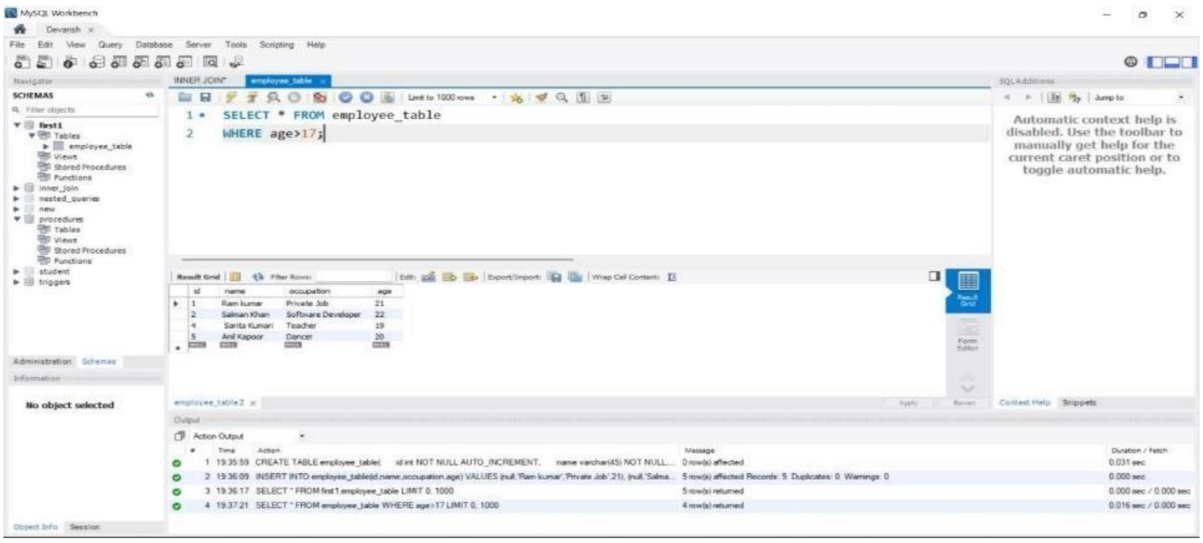
id	name	occupation	age
1	Ram kumar	Chef	21
2	Salman Khan	Software Developer	22
3	Meera Khan	Government Job	17
5	Anil Kapoor	Dancer	20
NULL	NULL	NULL	NULL

```
#  Time      Action                                                          Message                                                    Duration / Fetch
5  19:21:00  UPDATE employee_table SET occupation='Chef' WHERE id=1          1 row(s) affected Rows matched: 1  Changed: 1  Warnings: 0 0.000 sec
6  19:21:04  SELECT * FROM first1.employee_table LIMIT 0, 1000               5 row(s) returned                                          0.000 sec / 0.000 sec
7  19:23:49  SELECT * FROM employee_table LIMIT 0, 1000                      5 row(s) returned                                          0.015 sec / 0.000 sec
8  19:26:37  DELETE FROM employee_table WHERE id=4                           1 row(s) affected                                          0.000 sec
9  19:26:40  SELECT * FROM first1.employee_table LIMIT 0, 1000               4 row(s) returned                                          0.000 sec / 0.000 sec
```

To Implement Basic Select statements

1. SELECT with WHERE Clause:-

SELECT * FROM
employee_table
WHERE age>17;



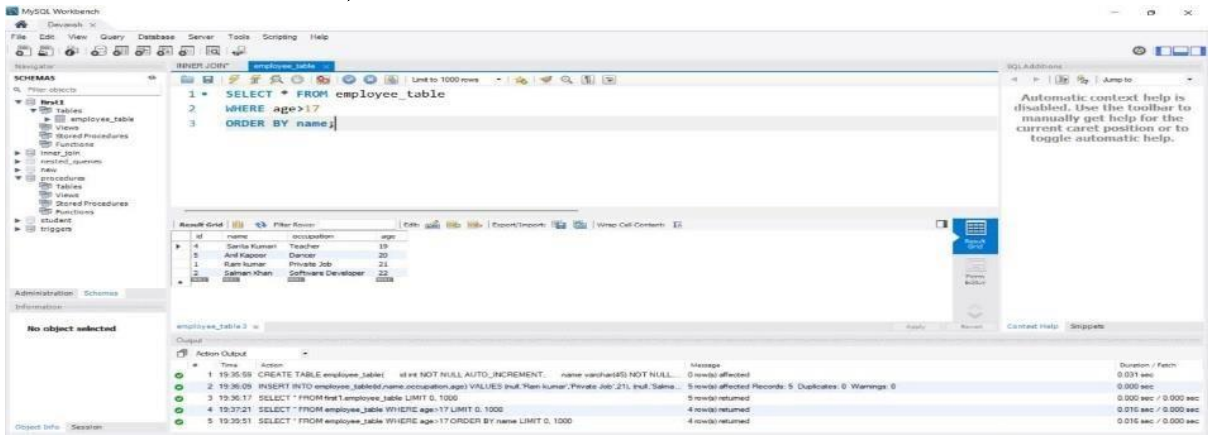
```
SELECT * FROM employee_table
WHERE age>17;
```

id	name	occupation	age
1	Ram kumar	Private Job	21
2	Salman Khan	Software Developer	22
4	Sarita Kumari	Teacher	19
5	Anil Kapoor	Dancer	20

```
#  Time      Action                                                                                         Message                                                    Duration / Fetch
1  19:35:59  CREATE TABLE employee_table(    id int NOT NULL AUTO_INCREMENT,    name varchar(45) NOT NULL...   0 row(s) affected                                          0.031 sec
2  19:36:09  INSERT INTO employee_table(id,name,occupation,age) VALUES (null,'Ram kumar','Private Job',21), (null,'Salma...   5 row(s) affected Records: 5  Duplicates: 0  Warnings: 0   0.000 sec
3  19:36:17  SELECT * FROM first1.employee_table LIMIT 0, 1000                                              5 row(s) returned                                          0.000 sec / 0.000 sec
4  19:37:21  SELECT * FROM employee_table WHERE age>17 LIMIT 0, 1000                                        4 row(s) returned                                          0.016 sec / 0.000 sec
```

2. SELECT With ORDER BY:-

SELECT * FROM
employee_table
WHERE age>17
ORDER BY name;



```
SELECT * FROM employee_table
WHERE age>17
ORDER BY name;
```

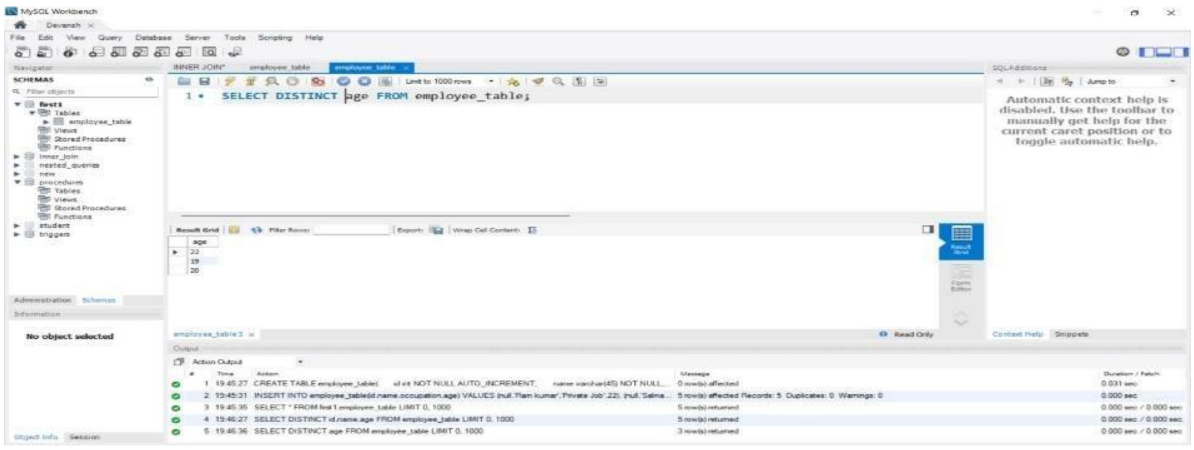
id	name	occupation	age
4	Sarita Kumari	Teacher	19
5	Anil Kapoor	Dancer	20
1	Ram kumar	Private Job	21
2	Salman Khan	Software Developer	22

```
#  Time      Action                                                                                         Message                                                    Duration / Fetch
1  19:35:59  CREATE TABLE employee_table(    id int NOT NULL AUTO_INCREMENT,    name varchar(45) NOT NULL...   0 row(s) affected                                          0.031 sec
2  19:36:09  INSERT INTO employee_table(id,name,occupation,age) VALUES (null,'Ram kumar','Private Job',21), (null,'Salma...   5 row(s) affected Records: 5  Duplicates: 0  Warnings: 0   0.000 sec
3  19:36:17  SELECT * FROM first1.employee_table LIMIT 0, 1000                                              5 row(s) returned                                          0.000 sec / 0.000 sec
4  19:37:21  SELECT * FROM employee_table WHERE age>17 LIMIT 0, 1000                                        4 row(s) returned                                          0.016 sec / 0.000 sec
5  19:39:51  SELECT * FROM employee_table WHERE age>17 ORDER BY name LIMIT 0, 1000                          4 row(s) returned                                          0.016 sec / 0.000 sec
```

SELECT WITH DISTINCT:-

QUERIES:

SELECT DISTINCT age FROM employee_table;



```
SELECT DISTINCT age FROM employee_table;
```

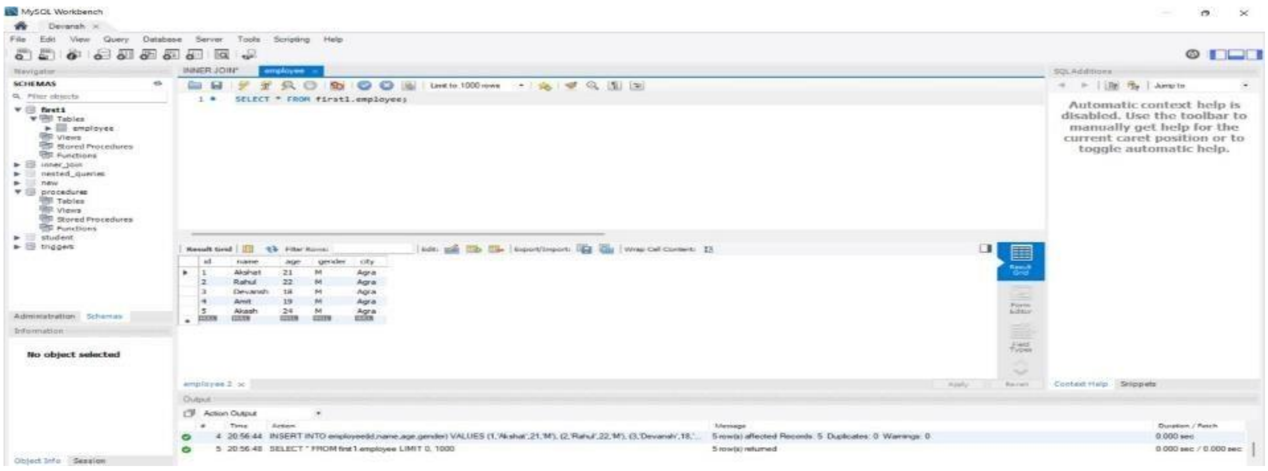
```
age
22
19
20
```

```
#  Time      Action                                                                                     Message                                                 Duration / Fetch
1  19:45:27  CREATE TABLE employee_table(    id int NOT NULL AUTO_INCREMENT,    name varchar(45) NOT NULL...  0 row(s) affected                                       0.031 sec
2  19:45:31  INSERT INTO employee_table(id,name,occupation,age) VALUES (null,'Ram kumar','Private Job',22), (null,'Salma...  5 row(s) affected Records: 5  Duplicates: 0  Warnings: 0  0.000 sec
3  19:45:35  SELECT * FROM first1.employee_table LIMIT 0, 1000                                          5 row(s) returned                                       0.000 sec / 0.000 sec
4  19:46:27  SELECT DISTINCT id,name,age FROM employee_table LIMIT 0, 1000                              5 row(s) returned                                       0.000 sec / 0.000 sec
5  19:46:36  SELECT DISTINCT age FROM employee_table LIMIT 0, 1000                                      3 row(s) returned                                       0.000 sec / 0.000 sec
```

To Execute Constraints in MySQL Theory:
The constraint in MySQL is used to specify the rule that allows or restricts what values/data will be stored in the table. They provide a suitable method to ensure data accuracy and integrity inside the table. It also helps to limit the type of data that will be inserted inside the table. If any interruption occurs between the constraint and data action, the action is failed.

QUERIES:

```
CREATE TABLE employee( id INT NOT
NULL UNIQUE, name VARCHAR(50)
NOT NULL, age INT NOT NULL
CHECK(age>=18), gender
VARCHAR(10)NOT NULL,
city VARCHAR(10) NOT NULL DEFAULT 'Agra'
);

INSERT INTO employee(id,name,age,gender)
VALUES
(1,'Akshat',21,'M'),
(2,'Rahul',22,'M'),
(3,'Devansh',18,'M'),
(4,'Amit',19,'M'),
(5,'Akash',24,'M');
```



```
SELECT * FROM first1.employee;
```

id	name	age	gender	city
1	Akshat	21	M	Agra
2	Rahul	22	M	Agra
3	Devansh	18	M	Agra
4	Amit	19	M	Agra
5	Akash	24	M	Agra
NULL	NULL	NULL	NULL	NULL

```
#  Time      Action                                                                                   Message                                                     Duration / Fetch
4  20:56:44  INSERT INTO employee(id,name,age,gender) VALUES (1,'Akshat',21,'M'), (2,'Rahul',22,'M'), (3,'Devansh',18,'...   5 row(s) affected Records: 5  Duplicates: 0  Warnings: 0   0.000 sec
5  20:56:48  SELECT * FROM first1.employee LIMIT 0, 1000                                              5 row(s) returned                                           0.000 sec / 0.000 sec
```

RESULT: The DML commands in RDBMS are implemented successfully and output is verified

Exp No:-3

DCL Commands and TCL Commands

Aim:- SQL data control language and transaction control Language
DCL (Data Control Language) includes commands like GRANT and REVOKE, which are useful to give "rights & permissions." Other permission controls parameters of the database system.

Examples of DCL commands:-

Commands that come under DCL:-

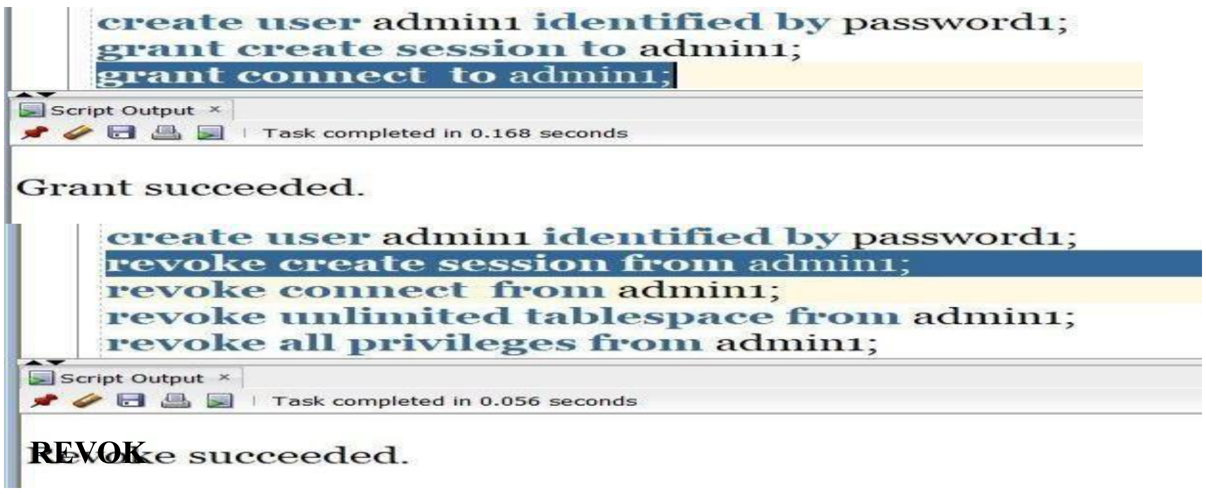
- Grant
- Revoke

GRANT

create user admin1 identified by
password1; grant create session to
admin1;
grant connect to admin1;

REVOKE

revoke create session from
admin1; revoke connect from
admin1; revoke create table
from admin1;



```
create user admin1 identified by password1;
grant create session to admin1;
grant connect  to admin1;
```

Script Output ×
Task completed in 0.168 seconds

```
Grant succeeded.
```

```
create user admin1 identified by password1;
revoke create session from admin1;
revoke connect  from admin1;
revoke unlimited tablespace from admin1;
revoke all privileges from admin1;
```

Script Output ×
Task completed in 0.056 seconds

```
Revoke succeeded.
```

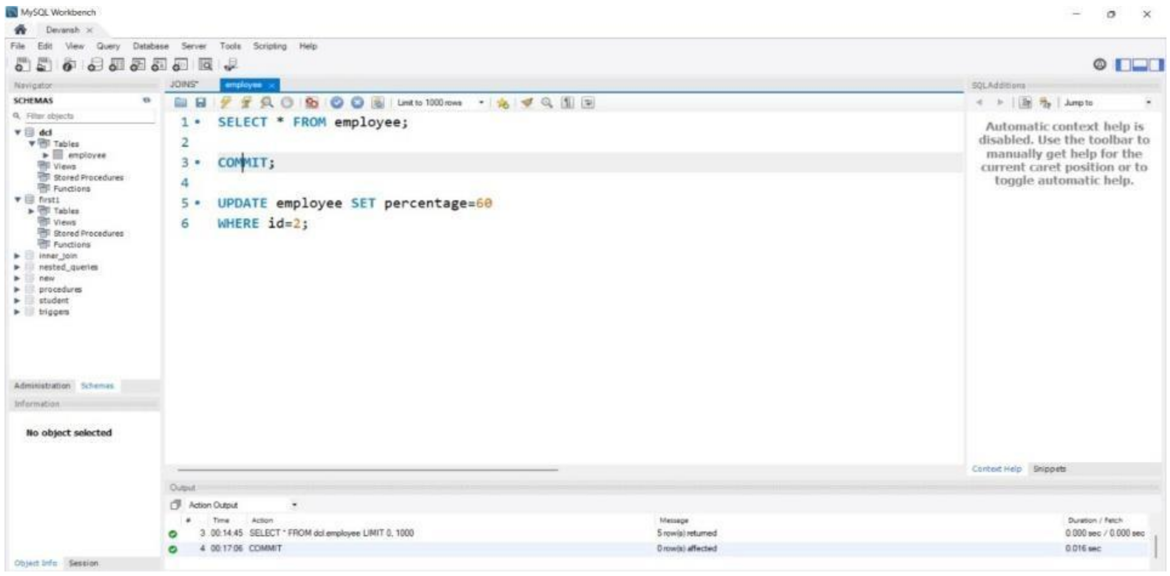
REVOK

TCL Commands in SQL

- In SQL, TCL stands for Transaction control language.
- A single unit of work in a database is formed after the consecutive execution of commands is known as a transaction.
- There are certain commands present in SQL known as TCL commands that help the user manage the transactions that take place in a database.
- COMMIT,ROLLBACK and SAVEPOINT are the most commonly used TCL commands in SQL.

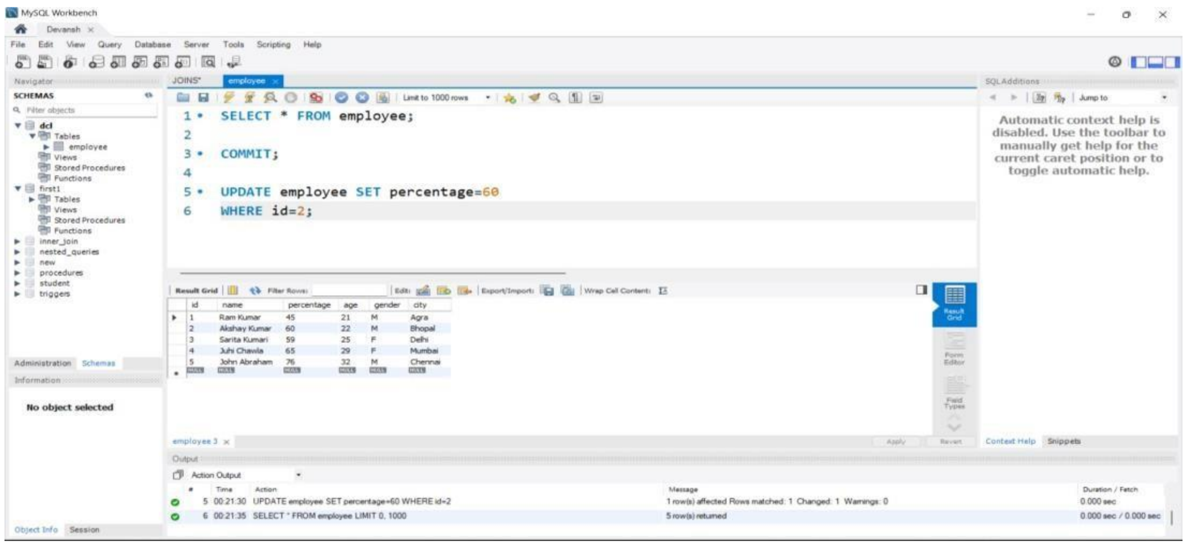
COMMIT:-

COMMIT command in SQL is used to save all the transaction-related changes permanently to the disk. Whenever DDL commands such as INSERT, UPDATE and DELETE are used, the changes made by these commands are permanent only after closing the current session. So before closing the session, one can easily roll back the changes made by the DDL commands. Hence, if we want the changes to be saved permanently to the disk without closing the session, we will use the commit command.



```
SELECT * FROM employee;

COMMIT;

UPDATE employee SET percentage=60
WHERE id=2;
```

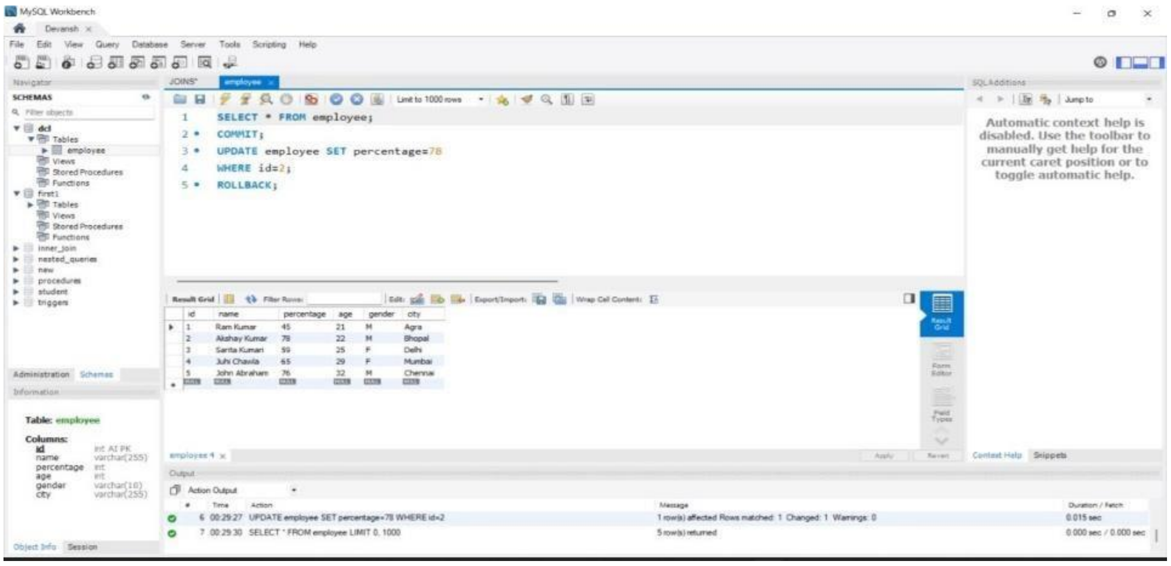
```
#  Time      Action                                        Message              Duration / Fetch
3  00:14:45  SELECT * FROM dcl.employee LIMIT 0, 1000      5 row(s) returned    0.000 sec / 0.000 sec
4  00:17:06  COMMIT                                        0 row(s) affected    0.016 sec
```



```
SELECT * FROM employee;

COMMIT;

UPDATE employee SET percentage=60
WHERE id=2;
```

id	name	percentage	age	gender	city
1	Ram Kumar	45	21	M	Agra
2	Akshay Kumar	60	22	M	Bhopal
3	Sarita Kumari	59	25	F	Delhi
4	Juhi Chawla	65	29	F	Mumbai
5	John Abraham	76	32	M	Chennai
NULL	NULL	NULL	NULL	NULL	NULL

```
#  Time      Action                                          Message                                              Duration / Fetch
5  00:21:30  UPDATE employee SET percentage=60 WHERE id=2    1 row(s) affected Rows matched: 1 Changed: 1 Warnings: 0   0.000 sec
6  00:21:35  SELECT * FROM employee LIMIT 0, 1000            5 row(s) returned                                    0.000 sec / 0.000 sec
```

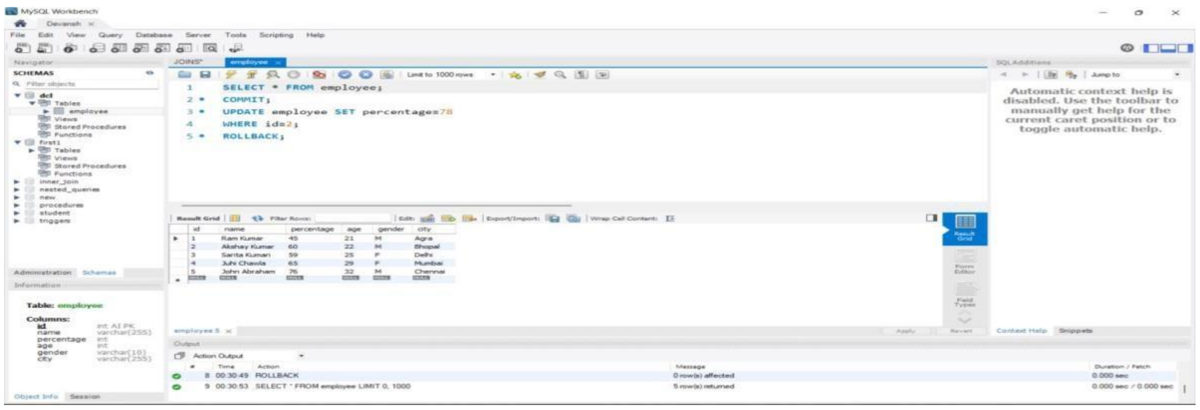


```sql
SELECT * FROM employee;
COMMIT;
UPDATE employee SET percentage=78
WHERE id=2;
ROLLBACK;
```

id	name	percentage	age	gender	city
1	Ram Kumar	45	21	M	Agra
2	Akshay Kumar	78	22	M	Bhopal
3	Sarita Kumari	59	25	F	Delhi
4	Juhi Chawla	65	29	F	Mumbai
5	John Abraham	76	32	M	Chennai

```
#  Time      Action                                          Message                                                      Duration / Fetch
6  00:29:27  UPDATE employee SET percentage=78 WHERE id=2    1 row(s) affected Rows matched: 1  Changed: 1  Warnings: 0   0.015 sec
7  00:29:30  SELECT * FROM employee LIMIT 0, 1000            5 row(s) returned                                            0.000 sec / 0.000 sec
```

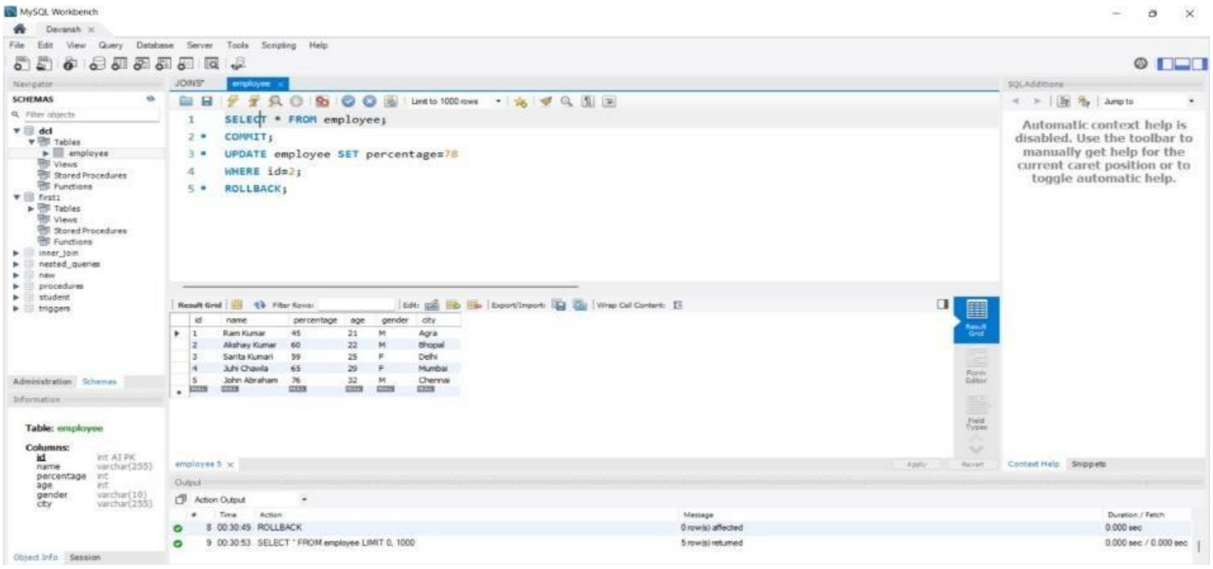
ROLLBACK:
While carrying a transaction, we must create savepoints to save different parts of the transaction. According to the user's changing requirements, he/she can roll back the transaction to different savepoints. Consider a scenario: We have initiated a transaction followed by the table creation and record insertion into the table. After inserting records, we have created a savepoint INS. Then we executed a delete query, but later we thought that mistakenly we had removed the useful record. Therefore in such situations, we have an option of rolling back our transaction. In this case, we have to roll back our transaction using the ROLLBACK command to the savepoint INS, which we have created before executing the DELETE query.



```sql
SELECT * FROM employee;
COMMIT;
UPDATE employee SET percentage=78
WHERE id=2;
ROLLBACK;
```

id	name	percentage	age	gender	city
1	Ram Kumar	45	21	M	Agra
2	Akshay Kumar	60	22	M	Bhopal
3	Sarita Kumari	59	25	F	Delhi
4	Juhi Chawla	65	29	F	Mumbai
5	John Abraham	76	32	M	Chennai

```
#  Time      Action                                  Message              Duration / Fetch
8  00:30:49  ROLLBACK                                0 row(s) affected    0.000 sec
9  00:30:53  SELECT * FROM employee LIMIT 0, 1000    5 row(s) returned    0.000 sec / 0.000 sec
```



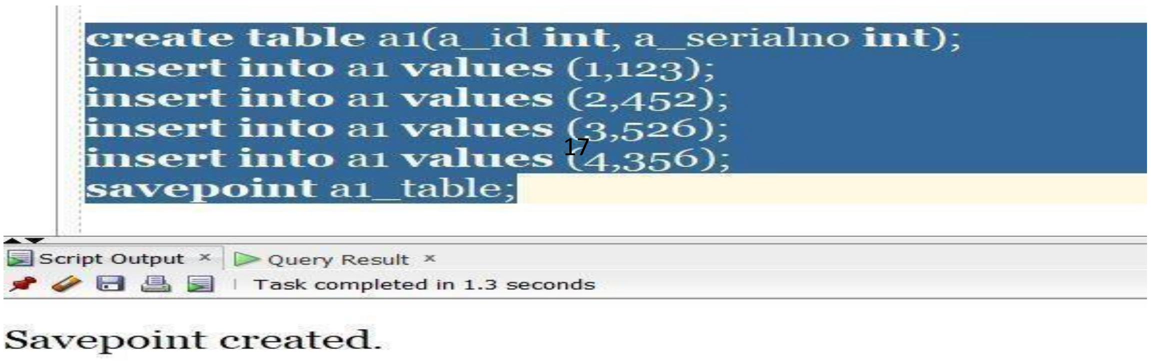
```sql
SELECT * FROM employee;
COMMIT;
UPDATE employee SET percentage=78
WHERE id=2;
ROLLBACK;
```

id	name	percentage	age	gender	city
1	Ram Kumar	45	21	M	Agra
2	Akshay Kumar	60	22	M	Bhopal
3	Sarita Kumari	59	25	F	Delhi
4	Juhi Chawla	65	29	F	Mumbai
5	John Abraham	76	32	M	Chennai

```
#  Time      Action                                  Message              Duration / Fetch
8  00:30:49  ROLLBACK                                0 row(s) affected    0.000 sec
9  00:30:53  SELECT * FROM employee LIMIT 0, 1000    5 row(s) returned    0.000 sec / 0.000 sec
```

SAVEPOINT:

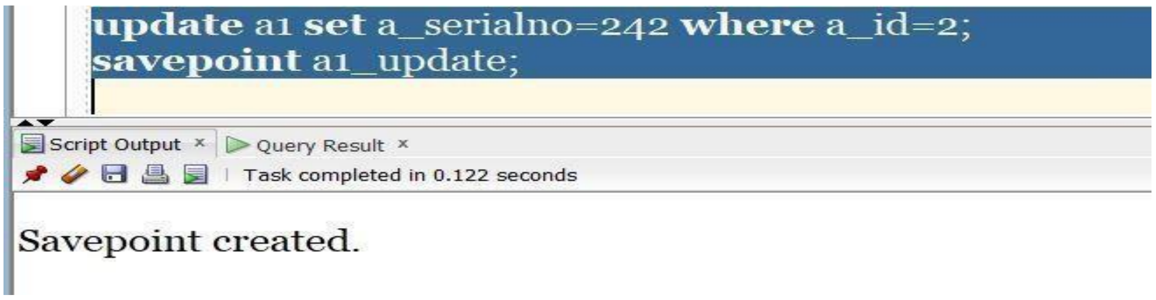
We can divide the database operations into parts. For example, we can consider all the insert related queries that we will execute consecutively as one part of the transaction and the delete command as the other part of the transaction. Using the SAVEPOINT command in SQL, we can save these different parts of the same transaction using different names. For example, we can save all the insert related queries with the savepoint named INS. To save all the insert related queries in one savepoint, we have to execute the SAVEPOINT query followed by the savepoint name after finishing the insert command execution.



```
create table a1(a_id int, a_serialno int);
insert into a1 values (1,123);
insert into a1 values (2,452);
insert into a1 values (3,526);
insert into a1 values (4,356);
savepoint a1_table;
```

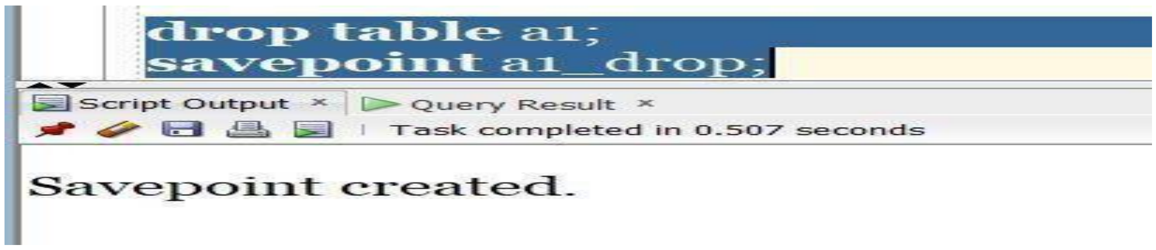
```
Savepoint created.
```

17



```
update a1 set a_serialno=242 where a_id=2;
savepoint a1_update;
```

```
Savepoint created.
```



```
drop table a1;
savepoint a1_drop;
```

```
Savepoint created.
```

RESULT: The DCL commands in RDBMS are implemented successfully and output is verified.

Exp No:-4
Inbuilt Functions

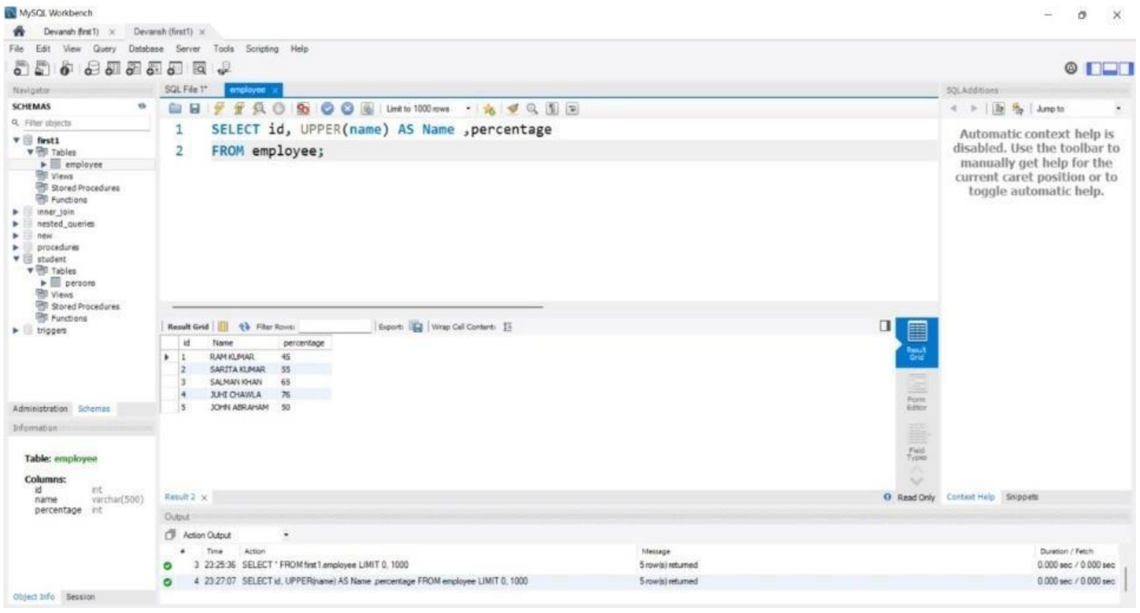
Aim:- To perform In-Built Functions In SQL.

To Execute String, Date,time function

Theory:- MySQL has many built-in functions.This reference contains string, numeric, date, and some advanced functions in MySQL.

1. **UPPER():-**

SELECT id, UPPER(name) AS Name ,percentage FROM employee;



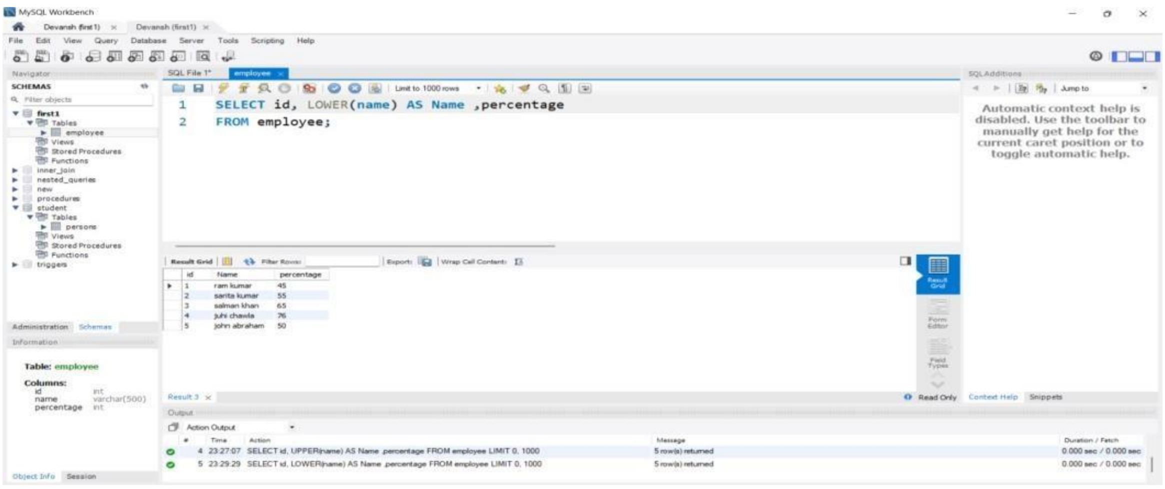
```
SELECT id, UPPER(name) AS Name ,percentage
FROM employee;
```

id	Name	percentage
1	RAM KUMAR	45
2	SARITA KUMAR	55
3	SALMAN KHAN	65
4	JUHI CHAWLA	76
5	JOHN ABRAHAM	50

```
#  Time      Action                                                                  Message            Duration / Fetch
3  23:25:36  SELECT * FROM first1.employee LIMIT 0, 1000                             5 row(s) returned  0.000 sec / 0.000 sec
4  23:27:07  SELECT id, UPPER(name) AS Name ,percentage FROM employee LIMIT 0, 1000  5 row(s) returned  0.000 sec / 0.000 sec
```

2. LOWER():-

SELECT id, LOWER(name) AS Name ,percentage FROM employee;



```
SELECT id, LOWER(name) AS Name ,percentage
FROM employee;
```

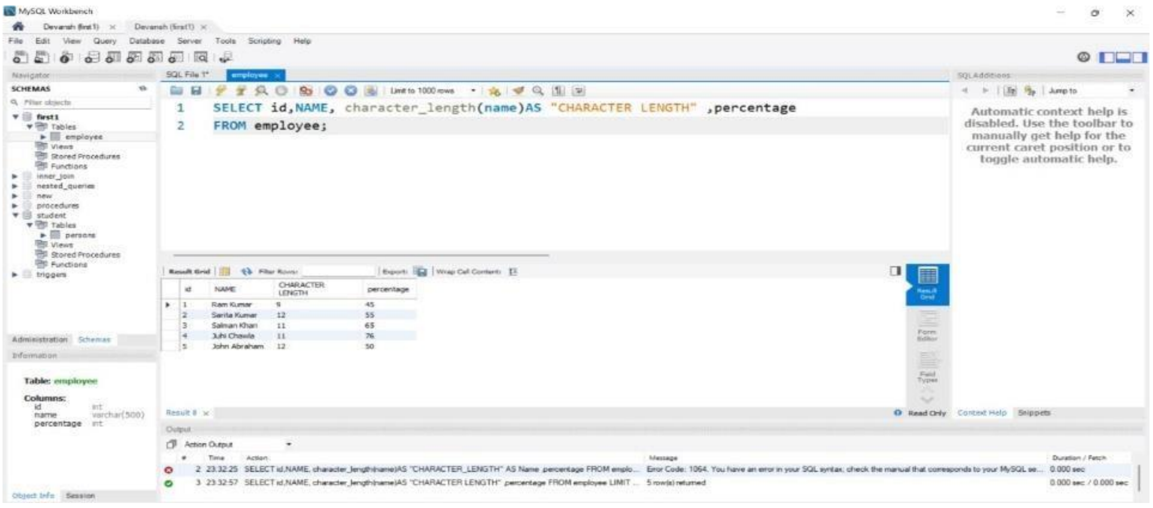
id	Name	percentage
1	ram kumar	45
2	sarita kumar	55
3	salman khan	65
4	juhi chawla	76
5	john abraham	50

```
#  Time      Action                                                                  Message            Duration / Fetch
4  23:27:07  SELECT id, UPPER(name) AS Name ,percentage FROM employee LIMIT 0, 1000  5 row(s) returned  0.000 sec / 0.000 sec
5  23:29:29  SELECT id, LOWER(name) AS Name ,percentage FROM employee LIMIT 0, 1000  5 row(s) returned  0.000 sec / 0.000 sec
```

3. CHARACTER_LENGTH():-

<u>QUERY</u>:-

SELECT id,NAME, character_length(name)AS "CHARACTER
LENGTH" ,percentage
FROM employee;



```
SELECT id,NAME, character_length(name)AS "CHARACTER LENGTH" ,percentage
FROM employee;
```

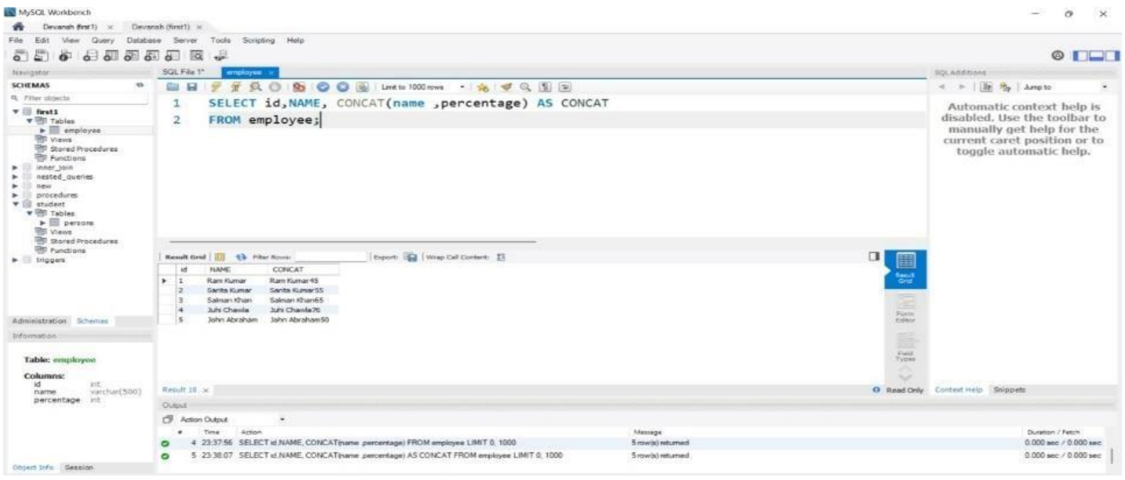
id	NAME	CHARACTER LENGTH	percentage
1	Ram Kumar	9	45
2	Sarita Kumar	12	55
3	Salman Khan	11	65
4	Juhi Chawla	11	76
5	John Abraham	12	50

```
#  Time      Action                                                                                  Message                                                                                   Duration / Fetch
2  23:32:25  SELECT id,NAME, character_length(name)AS "CHARACTER_LENGTH" AS Name ,percentage FROM emplo...  Error Code: 1064. You have an error in your SQL syntax; check the manual that corresponds to your MySQL se...  0.000 sec
3  23:32:57  SELECT id,NAME, character_length(name)AS "CHARACTER LENGTH" ,percentage FROM employee LIMIT ...  5 row(s) returned                                                                         0.000 sec / 0.000 sec
```

CONCAT():-

<u>QUERY</u>:-

SELECT id,NAME, CONCAT(name ,percentage) AS CONCAT FROM employee;



```
SELECT id,NAME, CONCAT(name ,percentage) AS CONCAT
FROM employee;
```

id	NAME	CONCAT
1	Ram Kumar	Ram Kumar45
2	Sarita Kumar	Sarita Kumar55
3	Salman Khan	Salman Khan65
4	Juhi Chawla	Juhi Chawla76
5	John Abraham	John Abraham50

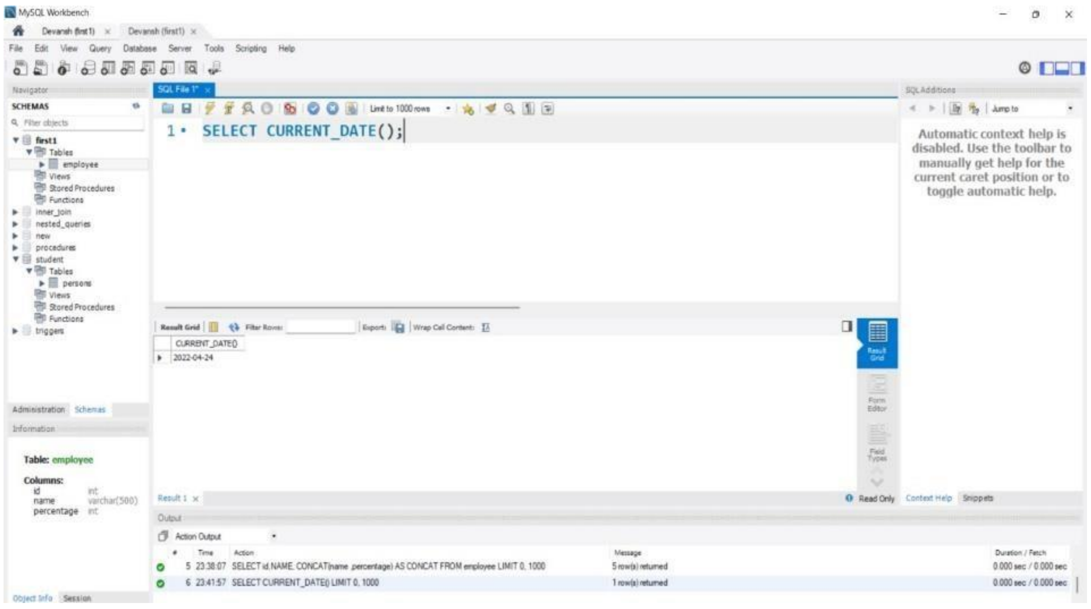
```
#  Time      Action                                                                  Message            Duration / Fetch
4  23:37:56  SELECT id,NAME, CONCAT(name ,percentage) FROM employee LIMIT 0, 1000            5 row(s) returned  0.000 sec / 0.000 sec
5  23:38:07  SELECT id,NAME, CONCAT(name ,percentage) AS CONCAT FROM employee LIMIT 0, 1000  5 row(s) returned  0.000 sec / 0.000 sec
```

DATE AND TIME FUNCTIONS:-

1.CURRENT_DATE():-

QUERY:-

SELECT CURRENT_DATE();



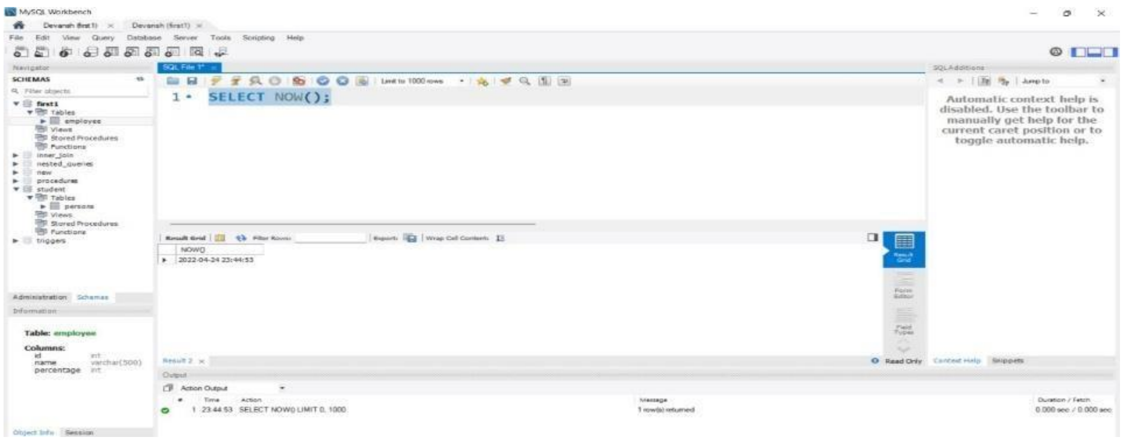
```
1 • SELECT CURRENT_DATE();
```

```
CURRENT_DATE()
2022-04-24
```

```
#  Time      Action                                                                        Message             Duration / Fetch
5  23:38:07  SELECT id,NAME, CONCAT(name ,percentage) AS CONCAT FROM employee LIMIT 0, 1000  5 row(s) returned    0.000 sec / 0.000 sec
6  23:41:57  SELECT CURRENT_DATE() LIMIT 0, 1000                                          1 row(s) returned   0.000 sec / 0.000 sec
```

2. NOW():-

QUERY:-
SELECT NOW();



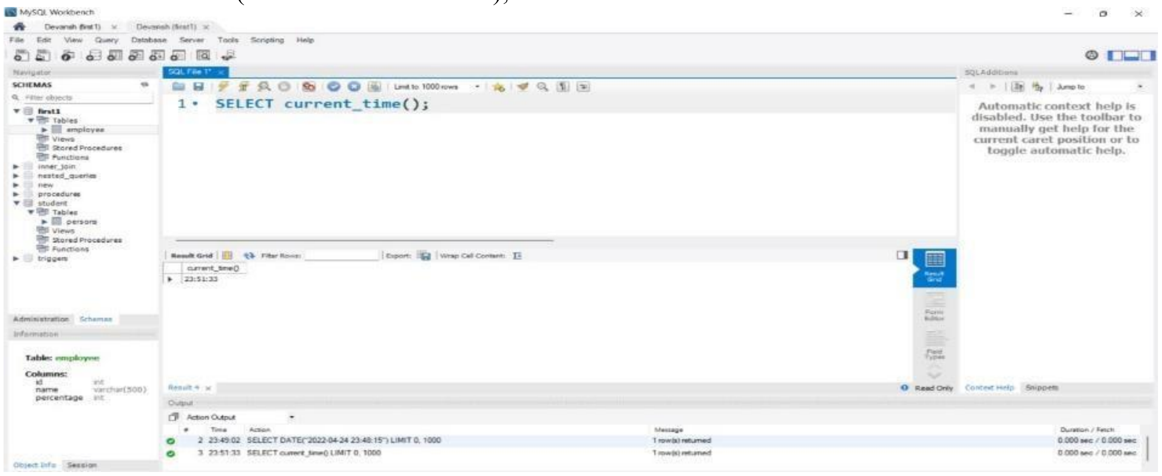
```
1 • SELECT NOW();
```

```
NOW()
2022-04-24 23:44:53
```

```
#  Time      Action                      Message            Duration / Fetch
1  23:44:53  SELECT NOW() LIMIT 0, 1000  1 row(s) returned  0.000 sec / 0.000 sec
```

3.DATE():-
_QUERY:-

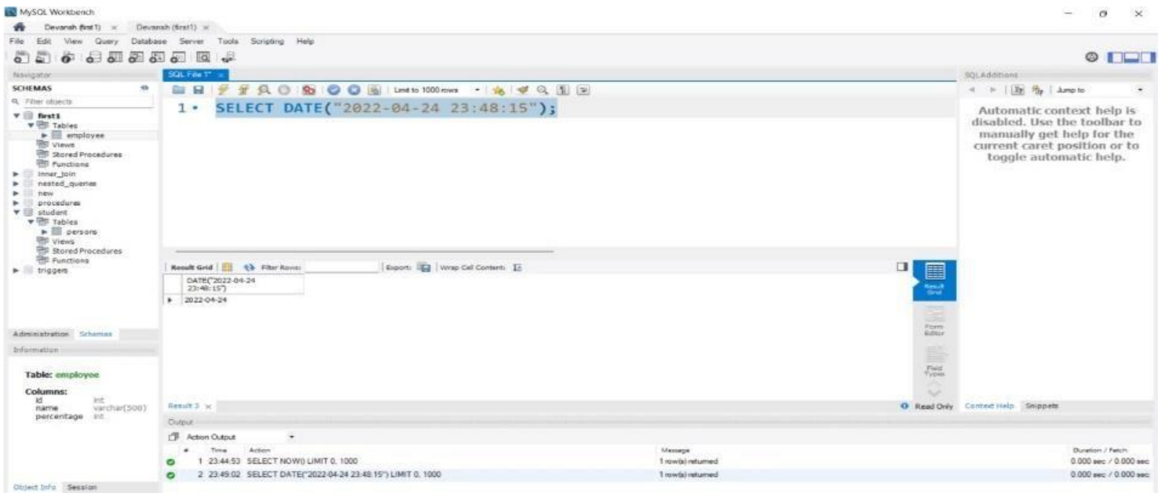
SELECT DATE("2022-04-24 23:48:15");



```
1 • SELECT current_time();
```

```
current_time()
23:51:33
```

```
#  Time      Action                                         Message            Duration / Fetch
2  23:49:02  SELECT DATE("2022-04-24 23:48:15") LIMIT 0, 1000  1 row(s) returned  0.000 sec / 0.000 sec
3  23:51:33  SELECT current_time() LIMIT 0, 1000            1 row(s) returned  0.000 sec / 0.000 sec
```



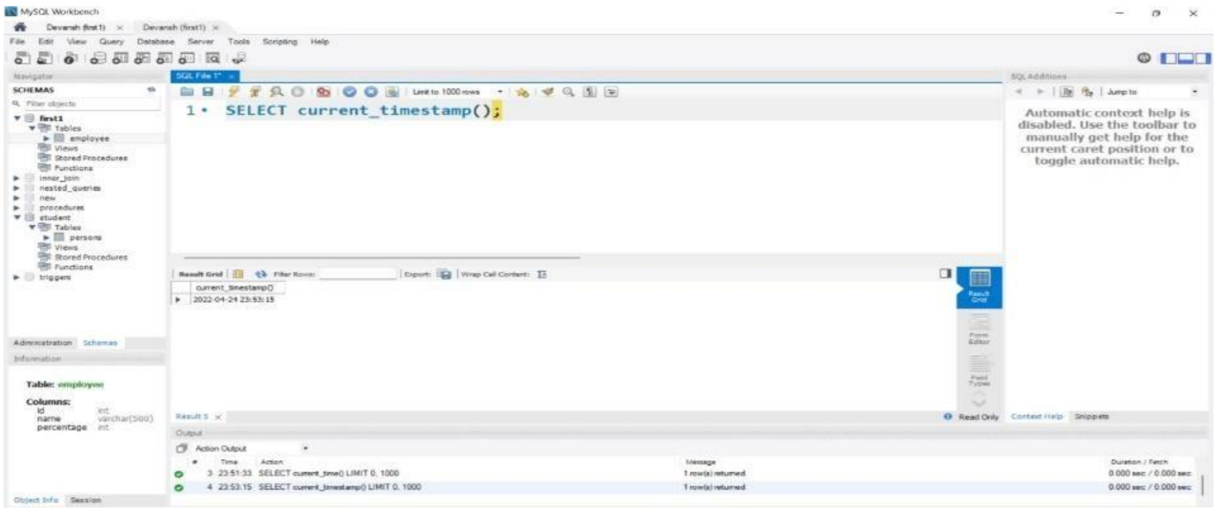
```
SELECT DATE("2022-04-24 23:48:15");
```

```
DATE("2022-04-24
23:48:15")
2022-04-24
```

```
#  Time      Action                                               Message             Duration / Fetch
1  23:44:53  SELECT NOW() LIMIT 0, 1000                           1 row(s) returned   0.000 sec / 0.000 sec
2  23:49:02  SELECT DATE("2022-04-24 23:48:15") LIMIT 0, 1000     1 row(s) returned   0.000 sec / 0.000 sec
```

TIME FUNCTIONS:-

1.<u>Current_time()</u>:- <u>QUERY</u>:-
SELECT current_time();
2<u> Current_timestamp():</u>-
<u>QUERY</u>:-
SELECT current_timestamp();



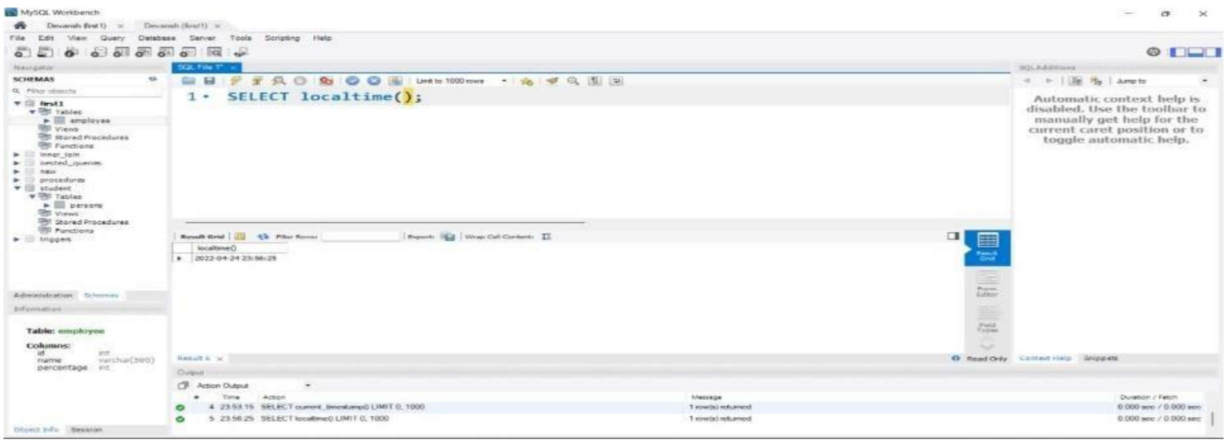
```
SELECT current_timestamp();
```

```
current_timestamp()
2022-04-24 23:53:15
```

```
#  Time      Action                                          Message             Duration / Fetch
3  23:51:33  SELECT current_time() LIMIT 0, 1000             1 row(s) returned   0.000 sec / 0.000 sec
4  23:53:15  SELECT current_timestamp() LIMIT 0, 1000        1 row(s) returned   0.000 sec / 0.000 sec
```

3. localtime():-

<u>QUERY</u>:-
SELECT localtime();



```
SELECT localtime();
```

```
localtime()
2022-04-24 23:56:25
```

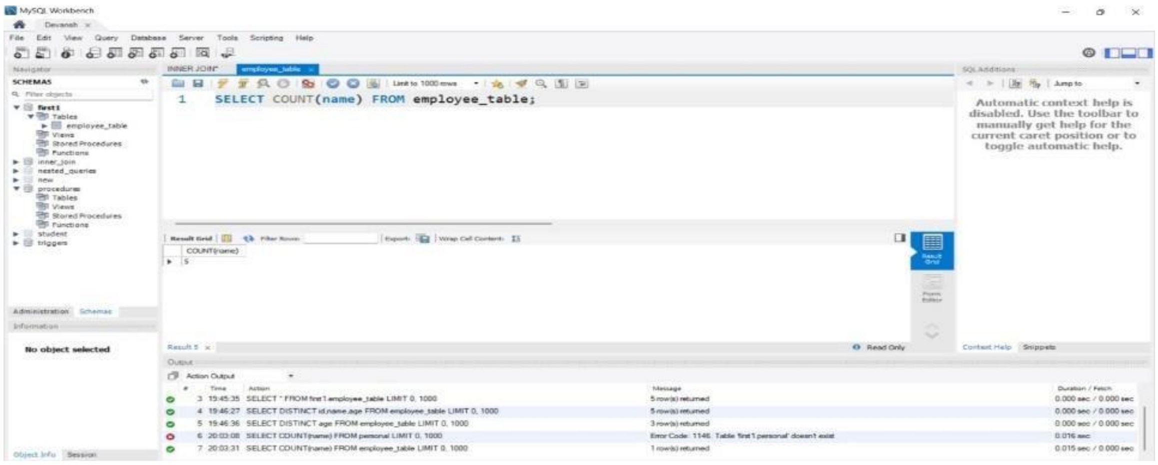
```
#  Time      Action                                        Message            Duration / Fetch
4  23:53:15  SELECT current_timestamp() LIMIT 0, 1000      1 row(s) returned  0.000 sec / 0.000 sec
5  23:56:25  SELECT localtime() LIMIT 0, 1000              1 row(s) returned  0.000 sec / 0.000 sec
```

To Execute Aggregate Functions.: SQL aggregation function is used to perform the calculations on multiple rows of a single column of a table. It returns a single value.It is also used to summarize the data.

1. **COUNT FUNCTION:** COUNT function is used to Count the number of rows in a database table. It can work on both numeric and non-numeric data types.

QUERY:

SELECT COUNT(name) FROM employee_table;



```
SELECT COUNT(name) FROM employee_table;
```

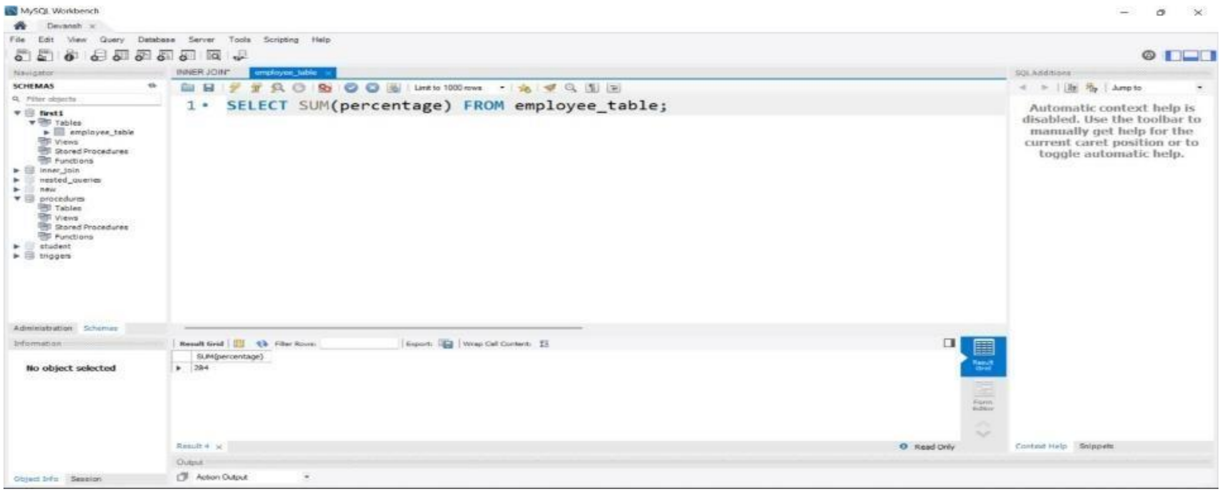
```
COUNT(name)
5
```

```
#  Time      Action                                                      Message                                          Duration / Fetch
3  19:45:35  SELECT * FROM first1.employee_table LIMIT 0, 1000           5 row(s) returned                                0.000 sec / 0.000 sec
4  19:46:27  SELECT DISTINCT id,name,age FROM employee_table LIMIT 0, 1000  5 row(s) returned                             0.000 sec / 0.000 sec
5  19:46:36  SELECT DISTINCT age FROM employee_table LIMIT 0, 1000       3 row(s) returned                                0.000 sec / 0.000 sec
6  20:03:08  SELECT COUNT(name) FROM personal LIMIT 0, 1000              Error Code: 1146. Table 'first1.personal' doesn't exist  0.016 sec
7  20:03:31  SELECT COUNT(name) FROM employee_table LIMIT 0, 1000        1 row(s) returned                                0.015 sec / 0.000 sec
```

2. **SUM Function: Sum function is used to calculate the sum of all selected columns. It works on numeric fields only.**

QUERY:-

SELECT SUM(percentage) FROM employee_table;



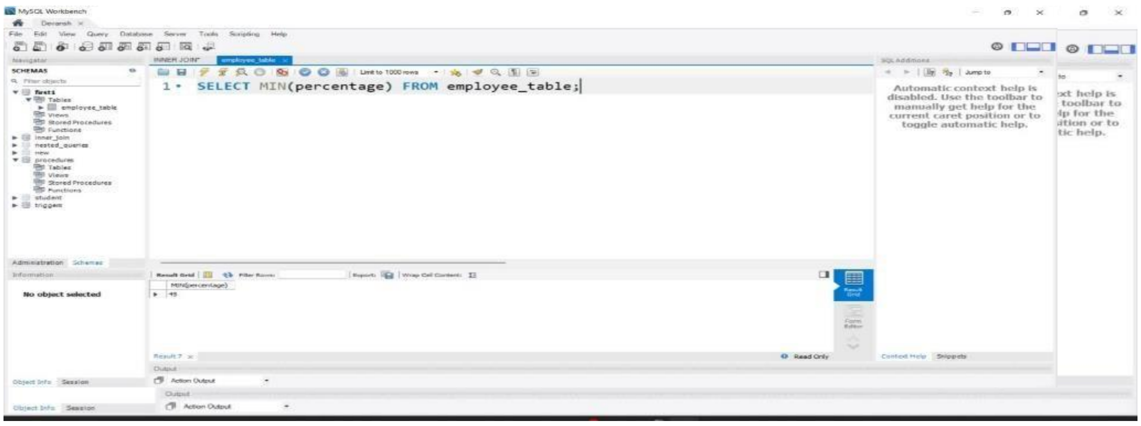
```
SELECT SUM(percentage) FROM employee_table;
```

```
SUM(percentage)
284
```

3. AVG function: The AVG function is used to calculate the average value of the numeric type.

<u>QUERY:</u>-

SELECT AVG(percentage) FROM employee_table;



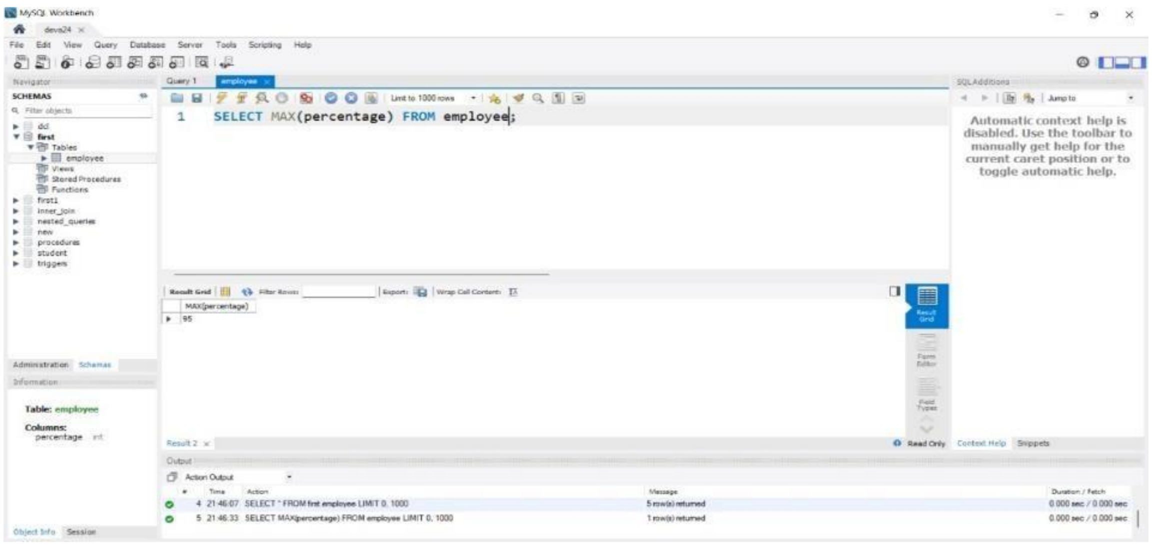
```
SELECT MIN(percentage) FROM employee_table;
```

```
MIN(percentage)
45
```

4. MAX Function: MAX function is used to find the maximum value of a certain column. This function determines the largest value of all selected values of a column.

<u>QUERY:-</u>

- SELECT MAX(percentage) FROM employee;



```
SELECT MAX(percentage) FROM employee;
```

```
MAX(percentage)
95
```

```
#  Time      Action                                               Message            Duration / Fetch
4  21:46:07  SELECT * FROM first.employee LIMIT 0, 1000           5 row(s) returned  0.000 sec / 0.000 sec
5  21:46:33  SELECT MAX(percentage) FROM employee LIMIT 0, 1000   1 row(s) returned  0.000 sec / 0.000 sec
```

Automatic context help is disabled. Use the toolbar to manually get help for the current caret position or to toggle automatic help.

Table: employee

Columns:
percentage int

5. **MIN Function**: MIN function is used to find the minimum value of a certain column. This function determines the smallest value of all selected values of a column.

QUERY:-

SELECT MIN(percentage) FROM employee_table;

RESULT: The inbuilt function commands in RDBMS are implemented successfully and output is verified.

Exp No:-5

Construct a ER Model for the application to be constructed to a Database

Aim:-Construct a e- r model for the application to be constructed to a Database:

E-R model for Property Management Database Project

E-R model for Water Supply Management System

E-R model for Home renting system database

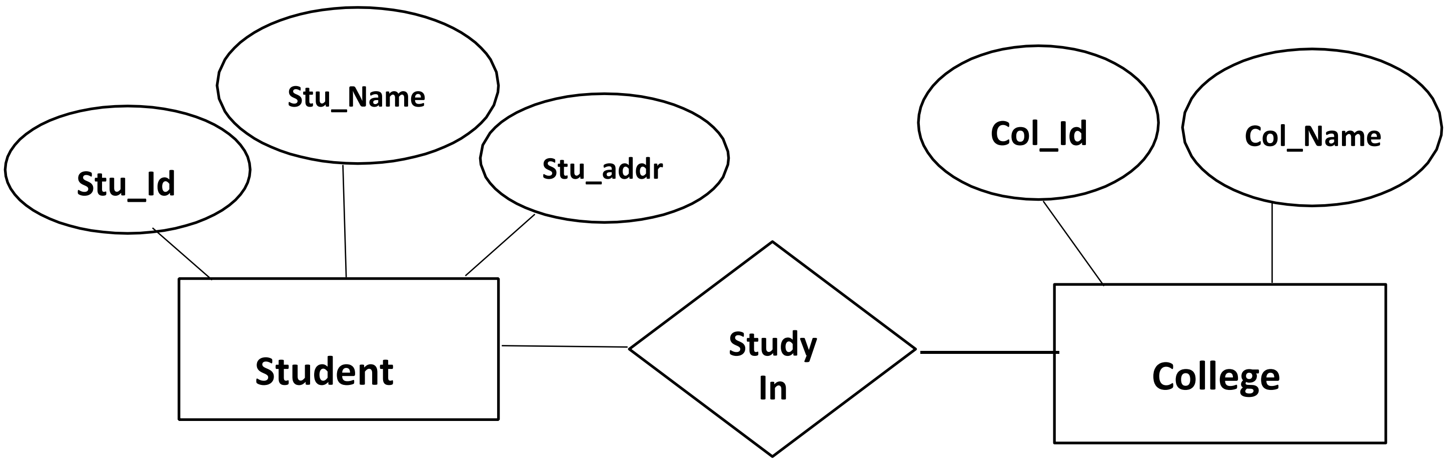
E-R model for Complaint management system database

E-R model for Employee performance review system database

E-R model for Employee track and report system database

Theory:-

ER model stands for an Entity-Relationship model. It is a high-level data model. This model is used to define the data elements and relationship for a specified system.It develops a conceptual design for the database. It also develops a very simple and easy to design view of data.In ER modeling, the database structure is portrayed as a diagram called an entity-relationship diagram.



EXP NO:-6
Nested Queries on sample exercise

Aim:- To Execute Subqueries

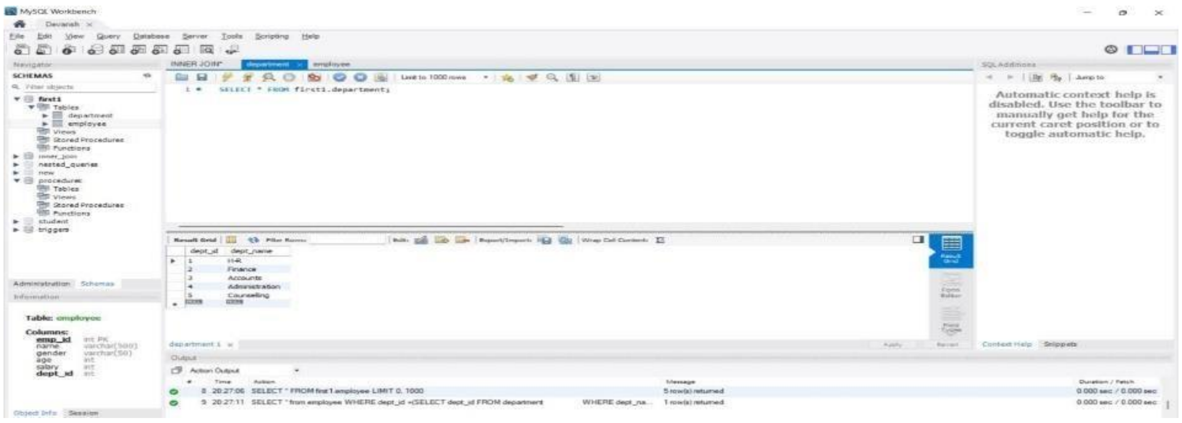
Theory:-

A subquery in MySQL is a query, which is nested into another SQL query and embedded with SELECT, INSERT, UPDATE or DELETE statement along with the various operators. We can also nest the subquery with another subquery. A subquery is known as the inner query, and the query that contains subquery is known as the outer query. The inner query executed first gives the result to the outer query, and then the main/outer query will be performed. MySQL allows us to use subquery anywhere, but it must be closed within parenthesis. All subquery forms and operations supported by the SQL standard will be supported in MySQL also.

QUERY:-

```
CREATE TABLE department( dept_id INT
        PRIMARY         KEY, dept_name
VARCHAR(50)
);

INSERT INTO department
VALUES
(1,'H-R'),
(2,'Finance'),
(3,'Accounts'),
(4,'Administration'),
(5,'Counselling');


CREATE TABLE employee( emp_id
INT PRIMARY KEY,
name VARCHAR(500),
gender VARCHAR(50),
age INT, salary INT,
dept_id INT,
FOREIGN KEY(dept_id) REFERENCES
department(dept_id)
);

INSERT INTO employee
VALUES
(1,'Ali','M',23,24000,3),
(2,'Anup','M',24,25000,4),
(3,'Akshay','M',22,22000,1),
(4,'Akshat','M',21,65000,2),
(5,'Rahul','M',23, 22000,4);
```

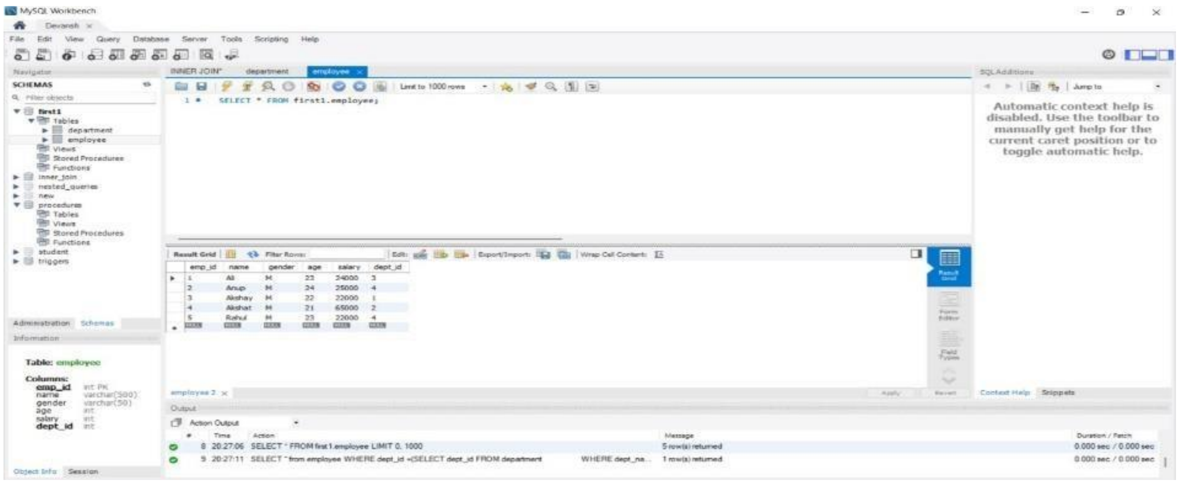
```
-- THIS IS NESTED QUERY-- SELECT
* from employee
WHERE dept_id =(SELECT dept_id FROM department WHERE dept_name='H-R');
```



```
SELECT * FROM first1.department;
```

dept_id	dept_name
1	H-R
2	Finance
3	Accounts
4	Administration
5	Counselling
NULL	NULL

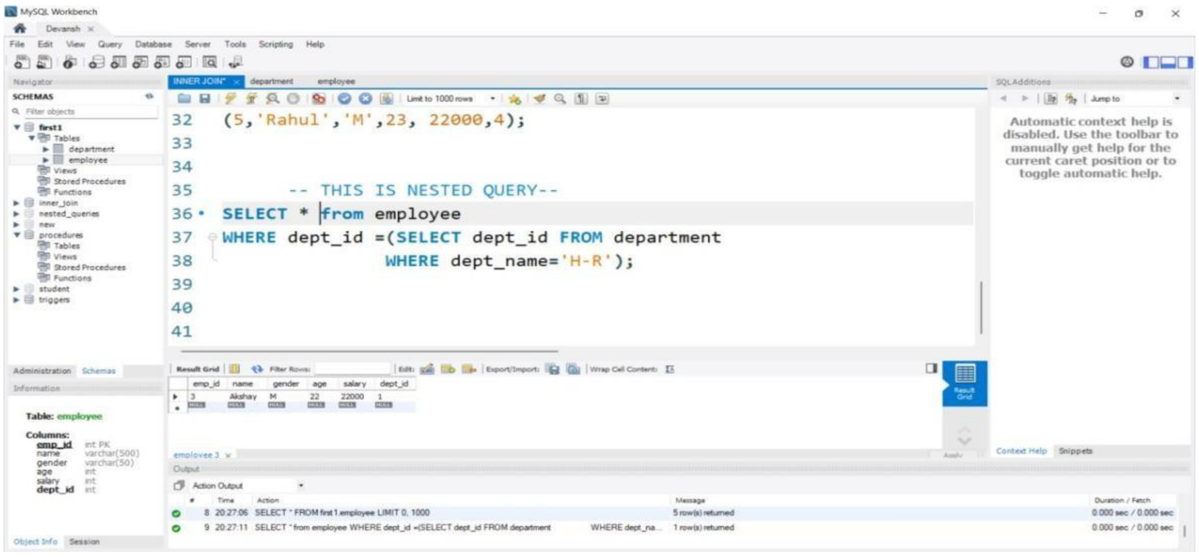
```
#  Time      Action                                                        Message            Duration / Fetch
8  20:27:06  SELECT * FROM first1.employee LIMIT 0, 1000                   5 row(s) returned  0.000 sec / 0.000 sec
9  20:27:11  SELECT * from employee WHERE dept_id =(SELECT dept_id FROM department    WHERE dept_na...  1 row(s) returned  0.000 sec / 0.000 sec
```



```
SELECT * FROM first1.employee;
```

emp_id	name	gender	age	salary	dept_id
1	Ali	M	23	24000	3
2	Anup	M	24	25000	4
3	Akshay	M	22	22000	1
4	Akshat	M	21	65000	2
5	Rahul	M	23	22000	4
NULL	NULL	NULL	NULL	NULL	NULL

```
#  Time      Action                                                        Message            Duration / Fetch
8  20:27:06  SELECT * FROM first1.employee LIMIT 0, 1000                   5 row(s) returned  0.000 sec / 0.000 sec
9  20:27:11  SELECT * from employee WHERE dept_id =(SELECT dept_id FROM department    WHERE dept_na...  1 row(s) returned  0.000 sec / 0.000 sec
```



```
(5,'Rahul','M',23, 22000,4);



        -- THIS IS NESTED QUERY--
SELECT * from employee
WHERE dept_id =(SELECT dept_id FROM department
                       WHERE dept_name='H-R');
```

emp_id	name	gender	age	salary	dept_id
3	Akshay	M	22	22000	1
NULL	NULL	NULL	NULL	NULL	NULL

```
#  Time      Action                                                        Message            Duration / Fetch
8  20:27:06  SELECT * FROM first1.employee LIMIT 0, 1000                   5 row(s) returned  0.000 sec / 0.000 sec
9  20:27:11  SELECT * from employee WHERE dept_id =(SELECT dept_id FROM department    WHERE dept_na...  1 row(s) returned  0.000 sec / 0.000 sec
```

Subqueries:-

CREATING TABLE FOR SUBQUERY

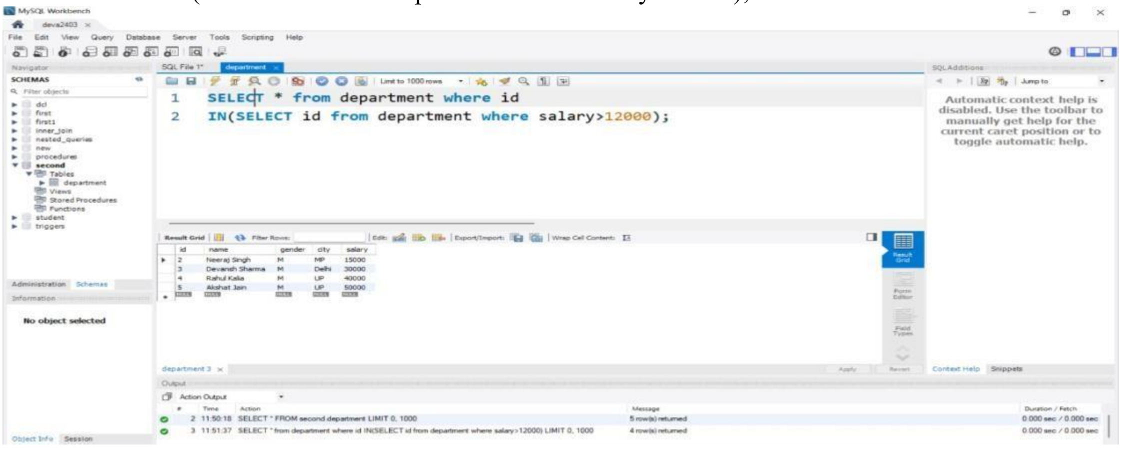
CREATE TABLE department(id INT primary key, name

varchar(100) NOT NULL,
gender varchar(50) NOT NULL,
city varchar(20) NOT NULL,
salary int NOT NULL
);

INSERT INTO department(id,name,gender,city,salary)
VALUES
(1,'Ram Kumar','M','Rajasthan',12000),
(2,'Neeraj Singh','M','MP',15000),
(3,'Devansh Sharma','M','Delhi',30000),
(4,'Rahul Kalia','M','UP',40000),
(5,'Akshat Jain','M','UP',50000);

QUERY 1:-

SELECT * from department where id
IN(SELECT id from department where salary>12000);



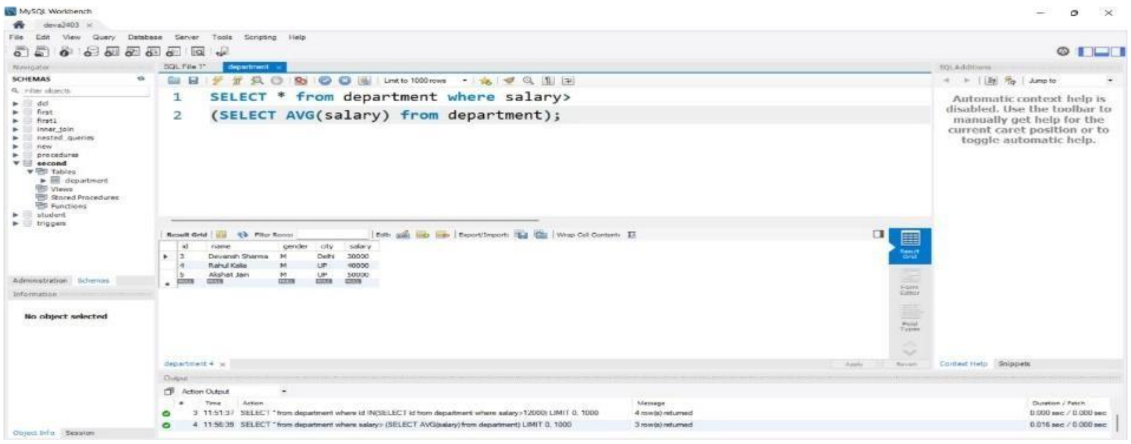
```
SELECT * from department where id
IN(SELECT id from department where salary>12000);
```

id	name	gender	city	salary
2	Neeraj Singh	M	MP	15000
3	Devansh Sharma	M	Delhi	30000
4	Rahul Kalia	M	UP	40000
5	Akshat Jain	M	UP	50000
NULL	NULL	NULL	NULL	NULL

```
#  Time      Action                                                                                     Message             Duration / Fetch
2  11:50:18  SELECT * FROM second.department LIMIT 0, 1000                                              5 row(s) returned   0.000 sec / 0.000 sec
3  11:51:37  SELECT * from department where id IN(SELECT id from department where salary>12000) LIMIT 0, 1000   4 row(s) returned   0.000 sec / 0.000 sec
```

QUERY 1:-

SELECT * from department where salary>
(SELECT AVG(salary) from department);



```
SELECT * from department where salary>
(SELECT AVG(salary) from department);
```

id	name	gender	city	salary
3	Devansh Sharma	M	Delhi	30000
4	Rahul Kalia	M	UP	40000
5	Akshat Jain	M	UP	50000
NULL	NULL	NULL	NULL	NULL

```
#  Time      Action                                                                                     Message             Duration / Fetch
3  11:51:37  SELECT * from department where id IN(SELECT id from department where salary>12000) LIMIT 0, 1000   4 row(s) returned   0.000 sec / 0.000 sec
4  11:56:39  SELECT * from department where salary> (SELECT AVG(salary) from department) LIMIT 0, 1000   3 row(s) returned   0.016 sec / 0.000 sec
```

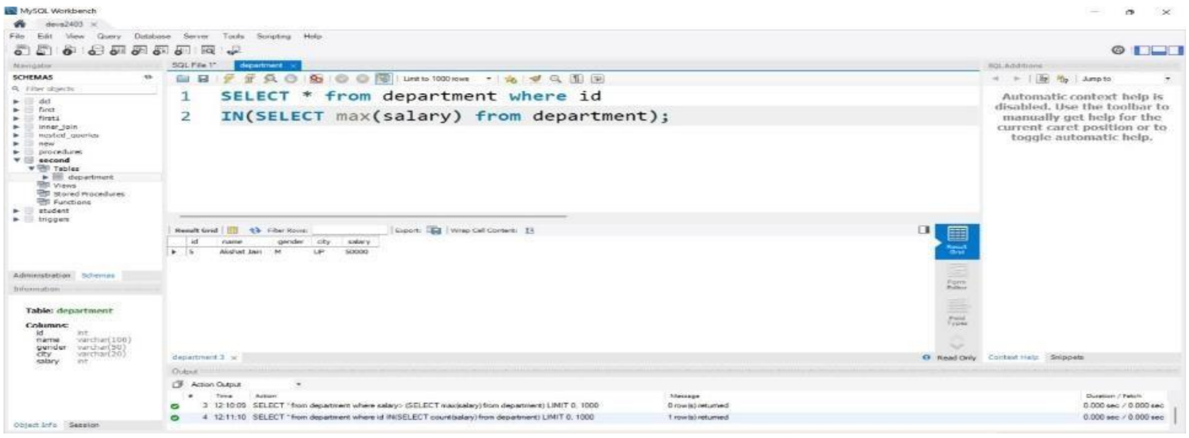
Find the name and salary of the employee with maximum salary:-

CREATING TABLE
CREATE TABLE department(id
INT,
name varchar(100),
gender
varchar(50), city

```
 varchar(20), salary
 int
 );
INSERT INTO department(id,name,gender,city,salary)
VALUES
(1,'Ram Kumar','M','Rajasthan',12000),
(2,'Neeraj Singh','M','MP',15000),
(3,'Devansh Sharma','M','Delhi',30000),
(4,'Rahul Kalia','M','UP',40000),
(5,'Akshat Jain','M','UP',50000);
```

QUERY:-

```
SELECT * from department where id
IN(SELECT max(salary) from department);
```



```
SELECT * from department where id
IN(SELECT max(salary) from department);
```

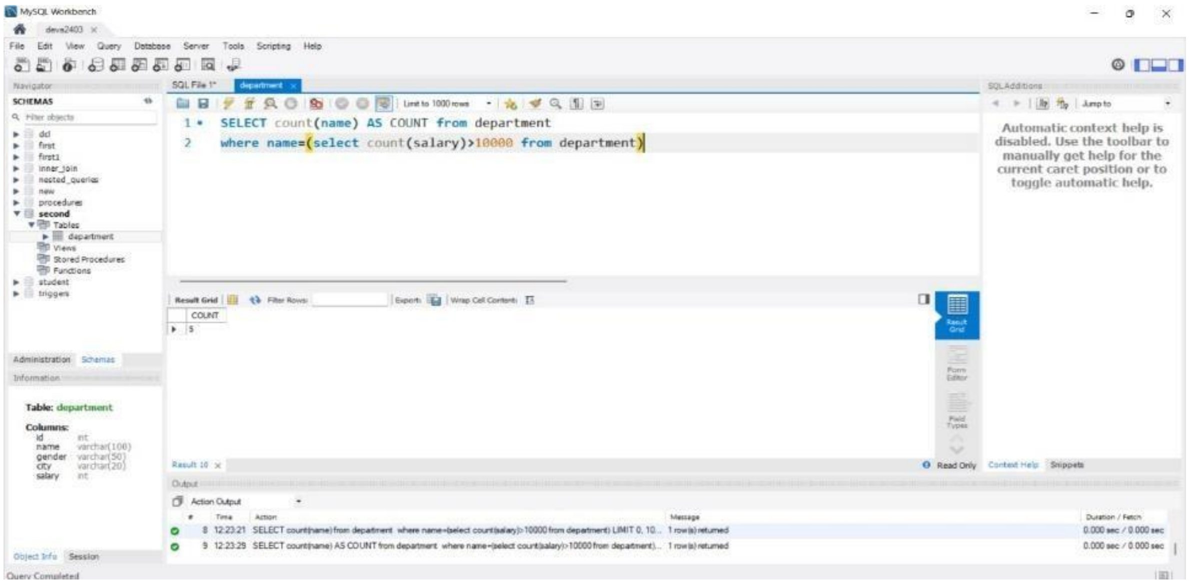
	id	name	gender	city	salary
▶	5	Akshat Jain	M	UP	50000

```
#  Time      Action                                                                                  Message              Duration / Fetch
3  12:10:09  SELECT * from department where salary> (SELECT max(salary) from department) LIMIT 0, 1000  0 row(s) returned    0.000 sec / 0.000 sec
4  12:11:10  SELECT * from department where id IN(SELECT count(salary) from department) LIMIT 0, 1000  1 row(s) returned    0.000 sec / 0.000 sec
```

Find the count of employees for each job so that at least two of the employees had salary greater than 10000:-

QUERY:-

```
SELECT count(name) AS COUNT from department where
name=(select count(salary)>10000 from department)
```



```
SELECT count(name) AS COUNT from department
where name=(select count(salary)>10000 from department)
```

	COUNT
▶	5

```
#  Time      Action                                                                                          Message            Duration / Fetch
8  12:23:21  SELECT count(name) from department  where name=(select count(salary)>10000 from department) LIMIT 0, 10...  1 row(s) returned  0.000 sec / 0.000 sec
9  12:23:29  SELECT count(name) AS COUNT from department  where name=(select count(salary)>10000 from department)...  1 row(s) returned  0.000 sec / 0.000 sec
```

RESULT: The program for nested queries in RDBMS are implemented successfully and output is verified.

Exp No:-7
Join Queries

Aim:-To Execute Joins in MySQL

Theory:

MySQL JOINS are used with SELECT statement. It is used to retrieve data from multiple tables. It is performed whenever you need to fetch records from two or more tables.

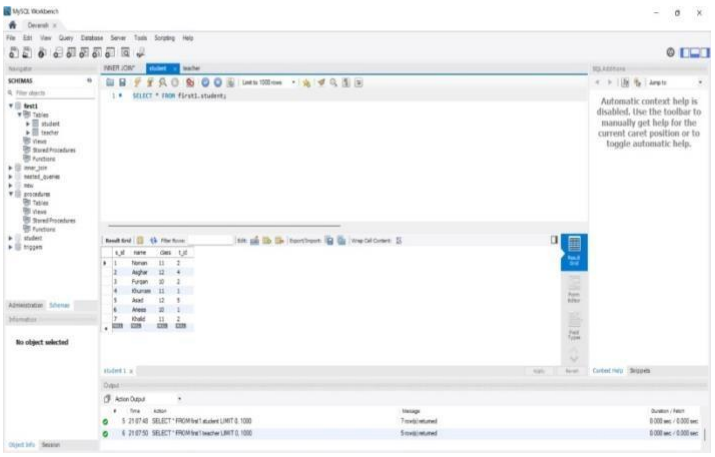
There are three types of MySQL joins:

MySQL INNER JOIN (or sometimes called simple join)
MySQL LEFT OUTER JOIN (or sometimes called LEFT JOIN)
MySQL RIGHT OUTER JOIN (or sometimes called RIGHT JOIN)

MySQL Inner JOIN (Simple Join):
The MySQL INNER JOIN is used to return all rows from multiple tables where the join condition is satisfied. It is the most common type of join.

QUERY:-

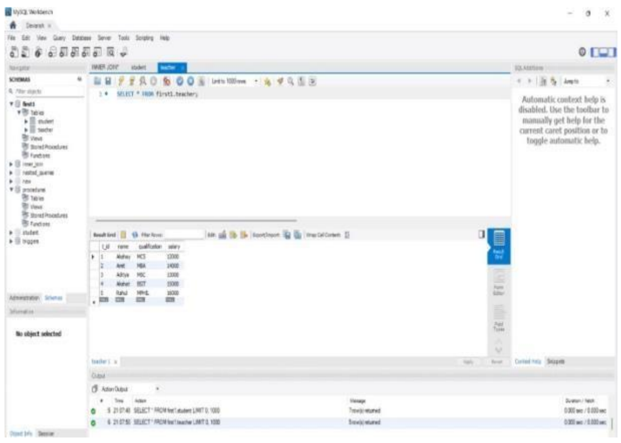
```
CREATE TABLE teacher( t_id int primary key,
name varchar(50) not null, qualification varchar(50) not null, salary int not null
);

INSERT INTO teacher VALUES (1,'Akshay','MCS',12000),
(2,'Amit','MBA',14000),
(3,'Aditya','MSC',13000),
(4,'Akshat','BSIT',15000),
(5,'Rahul','MPHIL',16000);

CREATE TABLE student( s_id int primary key,
name varchar(50) not null, class int not null,
t_id int not null
);

INSERT INTO student VALUES (1,'Noman',11,2),
(2,'Asghar',12,4),
(3,'Furqan',10,2),
(4,'Khurram',11,1),
(5,'Asad',12,5),
(6,'Anees',10,1),
(7,'Khalid',11,2);


-- INNER JOIN QUERY--

SELECT t.t_id,t.name,t.qualification,s.name,s.class FROM teacher t
INNER JOIN student s ON t.t_id= s.t_id ORDER BY t_id,t.name;
```



```
SELECT * FROM first1.student;
```

s_id	name	class	t_id
1	Noman	11	2
2	Asghar	12	4
3	Furqan	10	2
4	Khurram	11	1
5	Asad	12	5
6	Anees	10	1
7	Khalid	11	2

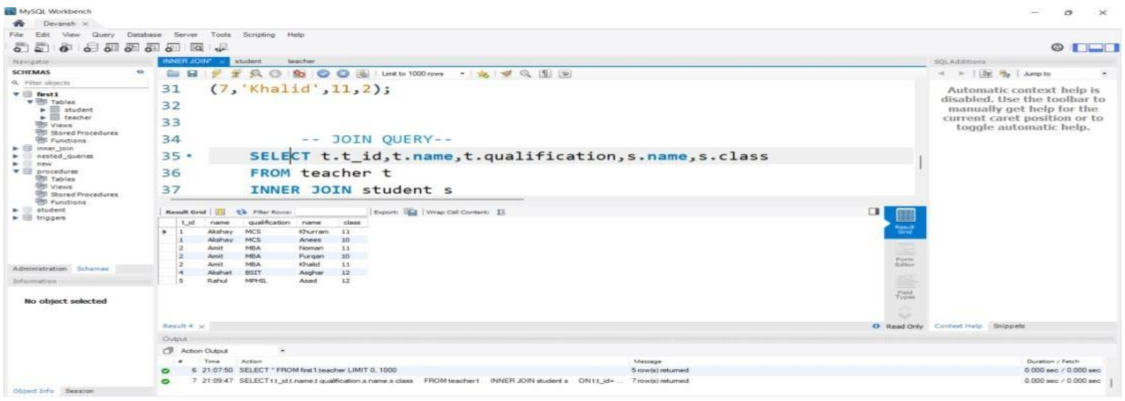
```
5 21:07:40 SELECT * FROM first1.student LIMIT 0, 1000    7 row(s) returned
6 21:07:50 SELECT * FROM first1.teacher LIMIT 0, 1000    5 row(s) returned
```



```
SELECT * FROM first1.teacher;
```

t_id	name	qualification	salary
1	Akshay	MCS	12000
2	Amit	MBA	14000
3	Aditya	MSC	13000
4	Akshat	BSIT	15000
5	Rahul	MPHIL	16000

```
5 21:07:40 SELECT * FROM first1.student LIMIT 0, 1000    7 row(s) returned
6 21:07:50 SELECT * FROM first1.teacher LIMIT 0, 1000    5 row(s) returned
```



```
(7,'Khalid',11,2);


        -- JOIN QUERY--
    SELECT t.t_id,t.name,t.qualification,s.name,s.class
    FROM teacher t
    INNER JOIN student s
```

t_id	name	qualification	name	class
1	Akshay	MCS	Khurram	11
1	Akshay	MCS	Anees	10
2	Amit	MBA	Noman	11
2	Amit	MBA	Furqan	10
2	Amit	MBA	Khalid	11
4	Akshat	BSIT	Asghar	12
5	Rahul	MPHIL	Asad	12

```
6 21:07:50 SELECT * FROM first1.teacher LIMIT 0, 1000    5 row(s) returned    0.000 sec / 0.000 sec
7 21:09:47 SELECT t.t_id,t.name,t.qualification,s.name,s.class    FROM teacher t    INNER JOIN student s    ON t.t_id= ...    7 row(s) returned    0.000 sec / 0.000 sec
```

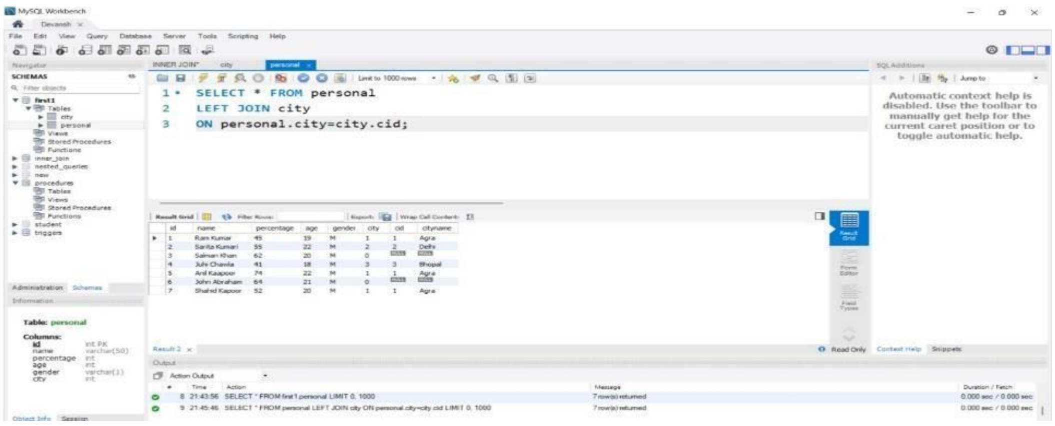
MySQL Left Outer Join:
The LEFT OUTER JOIN returns all rows from the left hand table specified in the ON condition and only those rows from the other table where the join condition is fulfilled.

<u>QUERY</u>:-

```
CREATE TABLE city(
cid INT NOT NULL AUTO_INCREMENT, cityname VARCHAR(50) NOT NULL, PRIMARY KEY(cid)
);

INSERT INTO city(cityname) VALUES
('Agra'),
('Delhi'),
('Bhopal'),
('Jaipur'),
('Noida');

CREATE TABLE personal( id INT NOT NULL, name
VARCHAR(50) NOT NULL, percentage INT NOT
NULL, age INT NOT NULL, gender VARCHAR(1) NOT
NULL, city INT NOT NULL,
PRIMARY KEY(id),
FOREIGN KEY(city) REFERENCES City(cid)
);

INSERT INTO personal(id,name,percentage,age,gender,city) VALUES
(1,'Ram Kumar',45,19,"M",1),
(2,'Sarita Kumari',55,22,"M",2),
(3,'Salman Khan',62,20,"M",1),
(4,'Juhi Chawla',41,18,"M",3),
(5,'Anil Kaapoor',74,22,"M",1),
```

(6,'John Abraham',64,21,"M",2),
(7,'Shahid Kapoor',52,20,"M",1);

SELECT * FROM personal LEFT JOIN city
ON personal.city=city.cid;

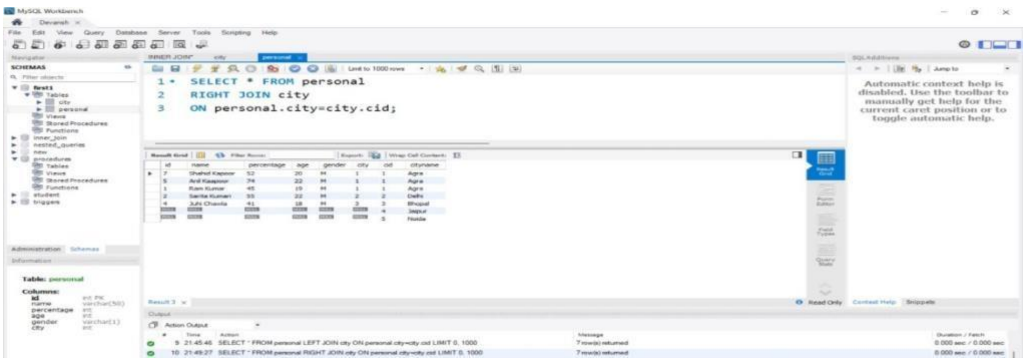


```
SELECT * FROM personal
LEFT JOIN city
ON personal.city=city.cid;
```

id	name	percentage	age	gender	city	cid	cityname
1	Ram Kumar	45	19	M	1	1	Agra
2	Sarita Kumari	55	22	M	2	2	Delhi
3	Salman Khan	62	20	M	0	NULL	NULL
4	Juhi Chawla	41	18	M	3	3	Bhopal
5	Anil Kaapoor	74	22	M	1	1	Agra
6	John Abraham	64	21	M	0	NULL	NULL
7	Shahid Kapoor	52	20	M	1	1	Agra

```
8  21:43:56  SELECT * FROM first1.personal LIMIT 0, 1000                                      7 row(s) returned   0.000 sec / 0.000 sec
9  21:45:46  SELECT * FROM personal LEFT JOIN city ON personal.city=city.cid LIMIT 0, 1000     7 row(s) returned   0.000 sec / 0.000 sec
```

MySQL Right Outer Join:
The MySQL Right Outer Join returns all rows from the RIGHT-hand table specified in the ON condition and only those rows from the other table where he join condition is fulfilled.



```
SELECT * FROM personal
RIGHT JOIN city
ON personal.city=city.cid;
```

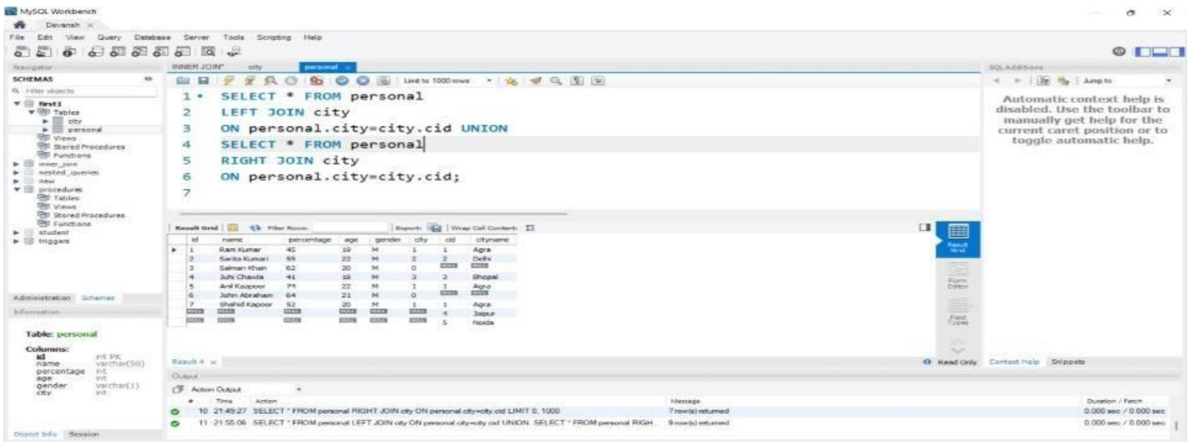
id	name	percentage	age	gender	city	cid	cityname
7	Shahid Kapoor	52	20	M	1	1	Agra
5	Anil Kaapoor	74	22	M	1	1	Agra
1	Ram Kumar	45	19	M	1	1	Agra
2	Sarita Kumari	55	22	M	2	2	Delhi
4	Juhi Chawla	41	18	M	3	3	Bhopal
NULL	NULL	NULL	NULL	NULL	NULL	4	Jaipur
NULL	NULL	NULL	NULL	NULL	NULL	5	Noida

```
9   21:45:46  SELECT * FROM personal LEFT JOIN city ON personal.city=city.cid LIMIT 0, 1000    7 row(s) returned   0.000 sec / 0.000 sec
10  21:49:27  SELECT * FROM personal RIGHT JOIN city ON personal.city=city.cid LIMIT 0, 1000   7 row(s) returned   0.000 sec / 0.000 sec
```

SQL FULL OUTER JOIN Keyword:

The FULL OUTER JOIN keyword returns all records when there is a match in left (table1) or right (table2) table records.

QUERY:-
SELECT * FROM personal LEFT JOIN city
ON personal.city=city.cid UNION SELECT * FROM personal RIGHT JOIN city ON personal.city=city.cid;



```
SELECT * FROM personal
LEFT JOIN city
ON personal.city=city.cid UNION
SELECT * FROM personal
RIGHT JOIN city
ON personal.city=city.cid;
```

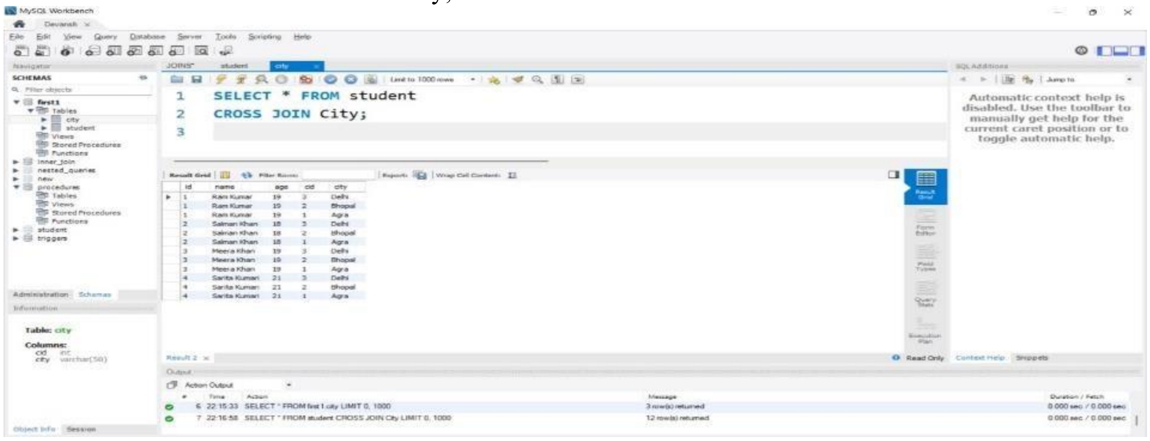
id	name	percentage	age	gender	city	cid	cityname
1	Ram Kumar	45	19	M	1	1	Agra
2	Sarita Kumari	55	22	M	2	2	Delhi
3	Salman Khan	62	20	M	0	NULL	NULL
4	Juhi Chawla	41	18	M	3	3	Bhopal
5	Anil Kaapoor	74	22	M	1	1	Agra
6	John Abraham	64	21	M	0	NULL	NULL
7	Shahid Kapoor	52	20	M	1	1	Agra
NULL	NULL	NULL	NULL	NULL	NULL	4	Jaipur
NULL	NULL	NULL	NULL	NULL	NULL	5	Noida

```
10  21:49:27  SELECT * FROM personal RIGHT JOIN city ON personal.city=city.cid LIMIT 0, 1000          7 row(s) returned   0.000 sec / 0.000 sec
11  21:55:06  SELECT * FROM personal LEFT JOIN city ON personal.city=city.cid UNION  SELECT * FROM personal RIGH   9 row(s) returned   0.000 sec / 0.000 sec
```

SQL CROSS JOIN Keyword:
The CROSS JOIN keyword returns all records from both tables (table1 and table2).

<u>QUERY</u>:

SELECT * FROM student CROSS JOIN City;



```
SELECT * FROM student
CROSS JOIN City;
```

id	name	age	cid	city
1	Ram Kumar	19	3	Delhi
1	Ram Kumar	19	2	Bhopal
1	Ram Kumar	19	1	Agra
2	Salman Khan	18	3	Delhi
2	Salman Khan	18	2	Bhopal
2	Salman Khan	18	1	Agra
3	Meera Khan	19	3	Delhi
3	Meera Khan	19	2	Bhopal
3	Meera Khan	19	1	Agra
4	Sarita Kumari	21	3	Delhi
4	Sarita Kumari	21	2	Bhopal
4	Sarita Kumari	21	1	Agra

```
#  Time      Action                                                  Message             Duration / Fetch
6  22:15:33  SELECT * FROM first1.city LIMIT 0, 1000                 3 row(s) returned   0.000 sec / 0.000 sec
7  22:16:58  SELECT * FROM student CROSS JOIN City LIMIT 0, 1000     12 row(s) returned  0.000 sec / 0.000 sec
```

<u>RESULT</u>: The program for join queries in RDBMS are implemented successfully and output is verified.

Exp No:-8
Set Operators & Views

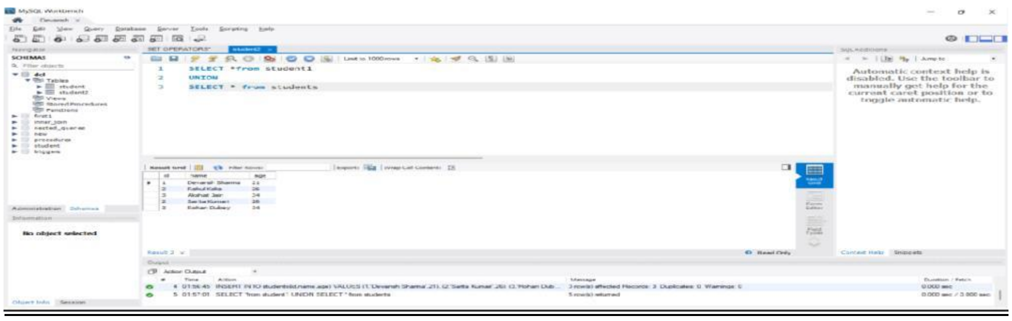
Aim:- To execute Set Operators and Views in Mysql

Theory:- SQL supports few Set operations which can be performed on the table data. These are used to get meaningful results from data stored in the table, under different special conditions.

SET OPERATORS:- Union
Union all
Intersect
Minus

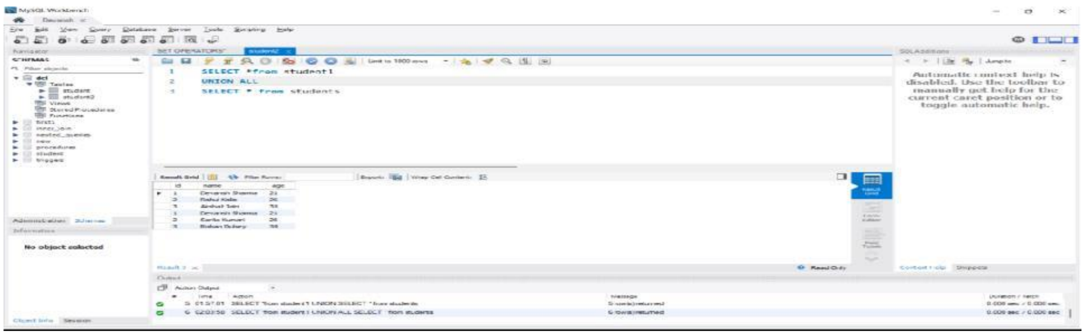
CREATING TABLES FOR SET OPERATORS(UNION and UNION ALL):-
```
CREATE TABLE student1( id INT, name
VARCHAR(255),
age INT
);

INSERT INTO student1(id,name,age) VALUES
(1,'Devansh Sharma',21),
(2,'Rahul Kalia',26),
(3,'Akshat Jain',34);

CREATE TABLE students( id INT, name
VARCHAR(255),
age INT
);

INSERT INTO students(id,name,age) VALUES
(1,'Devansh Sharma',21),
(2,'Sarita Kumari',26),
(3,'Rohan Dubey',34);
```

Union:- Query:-
```
SELECT *from student1 UNION
 SELECT * from students
```


```
SELECT *from student1
UNION
SELECT * from students
```
```
id  name            age
1   Devansh Sharma  21
2   Rahul Kalia     26
3   Akshat Jain     34
2   Sarita Kumari   26
3   Rohan Dubey     34
```
```
4  01:56:45  INSERT INTO students(id,name,age) VALUES (1,'Devansh Sharma',21), (2,'Sarita Kumari',26), (3,'Rohan Dub...  3 row(s) affected Records: 3  Duplicates: 0  Warnings: 0   0.000 sec
5  01:57:01  SELECT *from student1 UNION SELECT * from students                                                       5 row(s) returned                                          0.000 sec / 3.800 sec
```

Union All:-

```
SELECT *from student1 UNION ALL
SELECT * from students
```



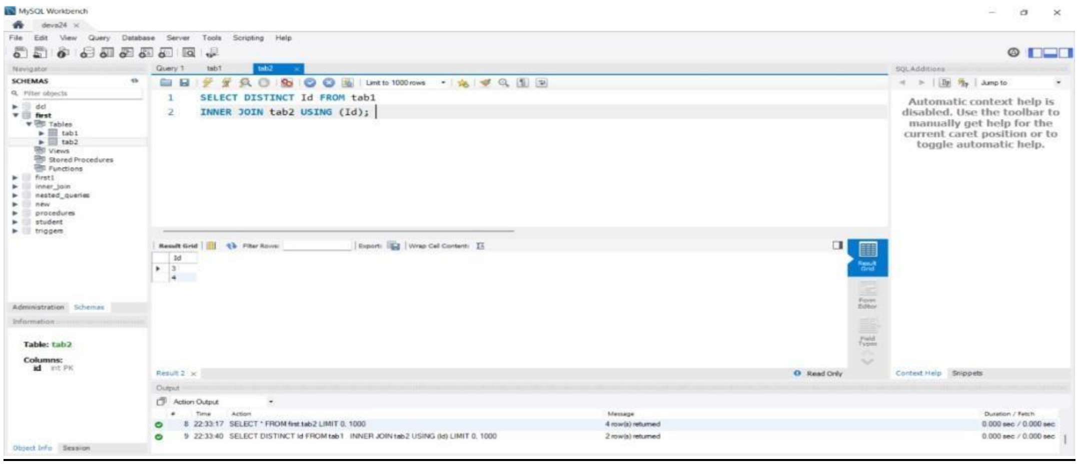
```
SELECT *from student1
UNION ALL
SELECT * from students
```

```
5  01:57:01  SELECT *from student1 UNION SELECT * from students
6  02:03:58  SELECT *from student1 UNION ALL SELECT * from students   6 row(s) returned
```

INTERSECT:-   Query:-
FOR CREATING TABLE
CREATE TABLE tab1 (Id INT PRIMARY KEY
);
INSERT INTO tab1 VALUES (1), (2), (3), (4);

CREATE TABLE tab2 (id INT PRIMARY KEY
);
INSERT INTO tab2 VALUES (3), (4), (5), (6);

FOR EXECUTION
SELECT DISTINCT Id FROM tab1 INNER JOIN tab2 USING (Id);



```
SELECT DISTINCT Id FROM tab1
INNER JOIN tab2 USING (Id);
```

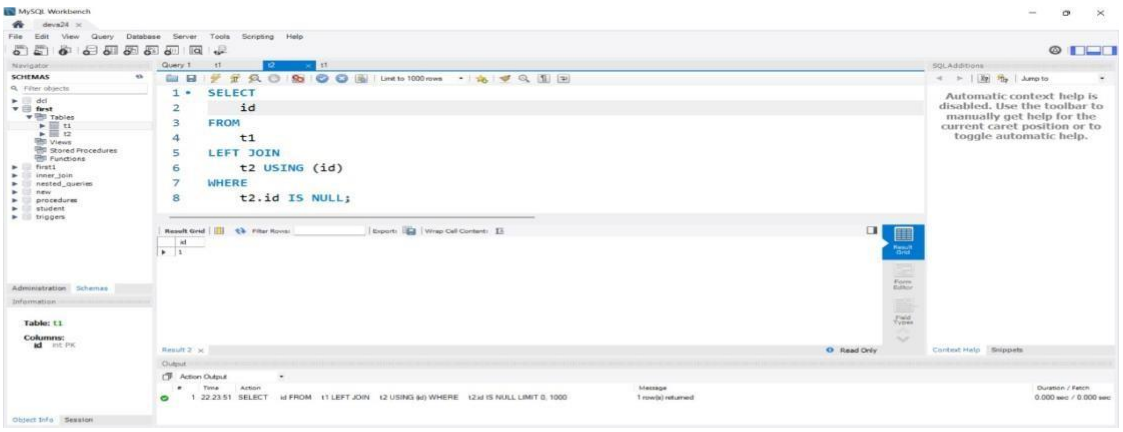
```
Id
3
4
```

```
#  Time      Action                                                                Message             Duration / Fetch
8  22:33:17  SELECT * FROM first.tab2 LIMIT 0, 1000                                4 row(s) returned   0.000 sec / 0.000 sec
9  22:33:40  SELECT DISTINCT Id FROM tab1  INNER JOIN tab2 USING (Id) LIMIT 0, 1000  2 row(s) returned   0.000 sec / 0.000 sec
```

MINUS:-Query:

FOR CREATING TABLE CREATE
TABLE t1 (
id INT PRIMARY KEY
);
CREATE TABLE t2 (
id INT PRIMARY KEY
);
INSERT INTO t1 VALUES (1),(2),(3); INSERT INTO t2 VALUES (2),(3),(4);

FOR EXECUTION
SELECT id FROM
t1
LEFT JOIN t2 USING
(id) WHERE
t2.id IS NULL;



```
SELECT
    id
FROM
    t1
LEFT JOIN
    t2 USING (id)
WHERE
    t2.id IS NULL;
```

```
id
1
```

```
#  Time      Action                                                               Message            Duration / Fetch
1  22:23:51  SELECT id FROM t1 LEFT JOIN t2 USING (id) WHERE t2.id IS NULL LIMIT 0, 1000   1 row(s) returned  0.000 sec / 0.000 sec
```

VIEWS:-

Theory:-

In SQL, a view is a virtual table based on the result-set of an SQL statement.

A view contains rows and columns, just like a real table. The fields in a view are fields from one or more real tables in the database.

You can add SQL statements and functions to a view and present the data as if the data were coming from one single table.

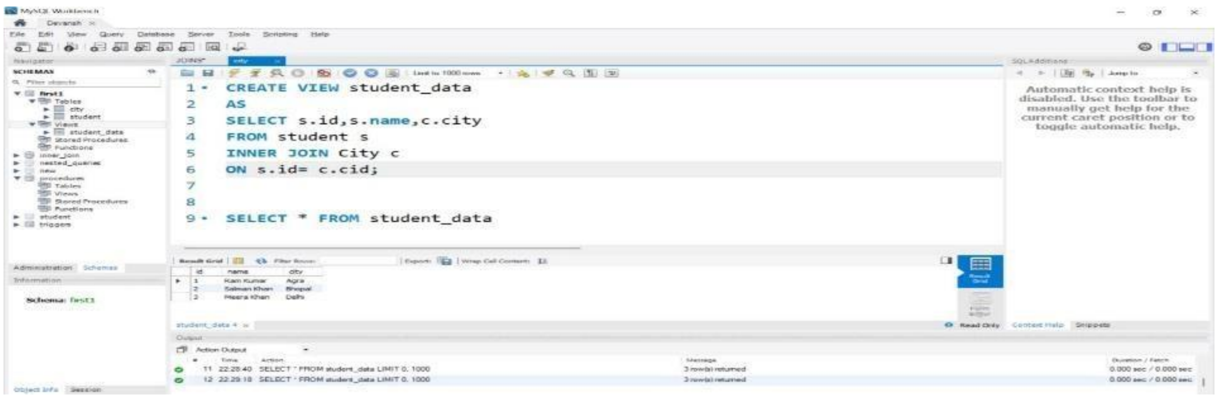
A view is created with the CREATE VIEW statement.

QUERY:-

CREATE VIEW student_data AS
SELECT s.id,s.name,c.city FROM student s
INNER JOIN City c

ON s.id= c.cid;

SELECT * FROM student_data



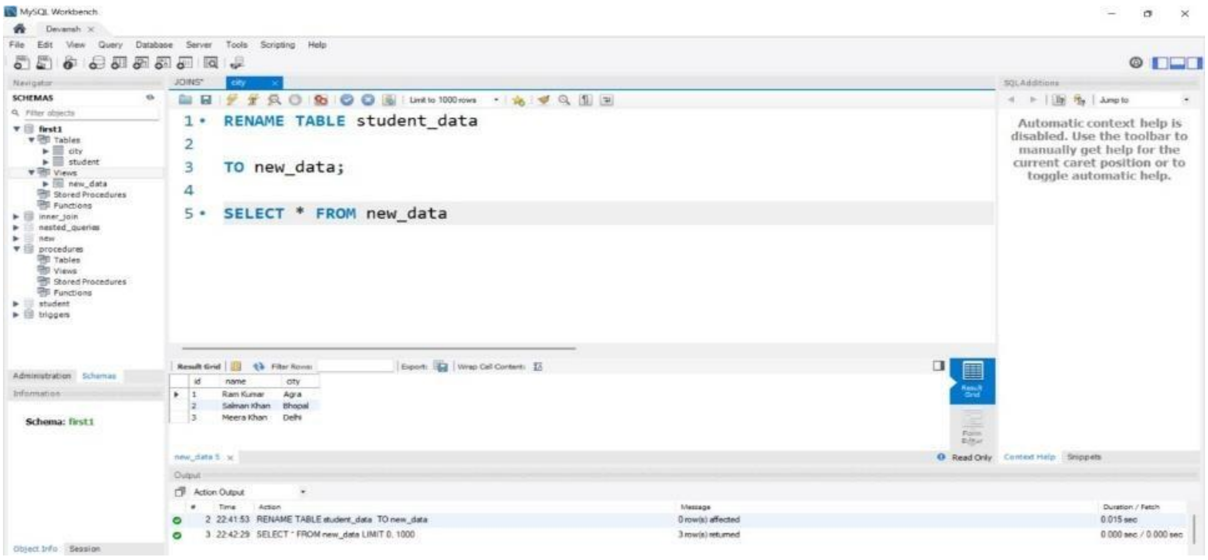
```
CREATE VIEW student_data
AS
SELECT s.id,s.name,c.city
FROM student s
INNER JOIN City c
ON s.id= c.cid;


SELECT * FROM student_data
```

```
id  name         city
1   Ram Kumar    Agra
2   Salman Khan  Bhopal
3   Meera Khan   Delhi
```

```
#   Time      Action                                      Message            Duration / Fetch
11  22:28:40  SELECT * FROM student_data LIMIT 0, 1000    3 row(s) returned  0.000 sec / 0.000 sec
12  22:29:18  SELECT * FROM student_data LIMIT 0, 1000    3 row(s) returned  0.000 sec / 0.000 sec
```

IF we rename the existing view then we use RENAME Command:- RENAME
TABLE student_data
TO new_data;

SELECT * FROM new_data



```
RENAME TABLE student_data

TO new_data;

SELECT * FROM new_data
```

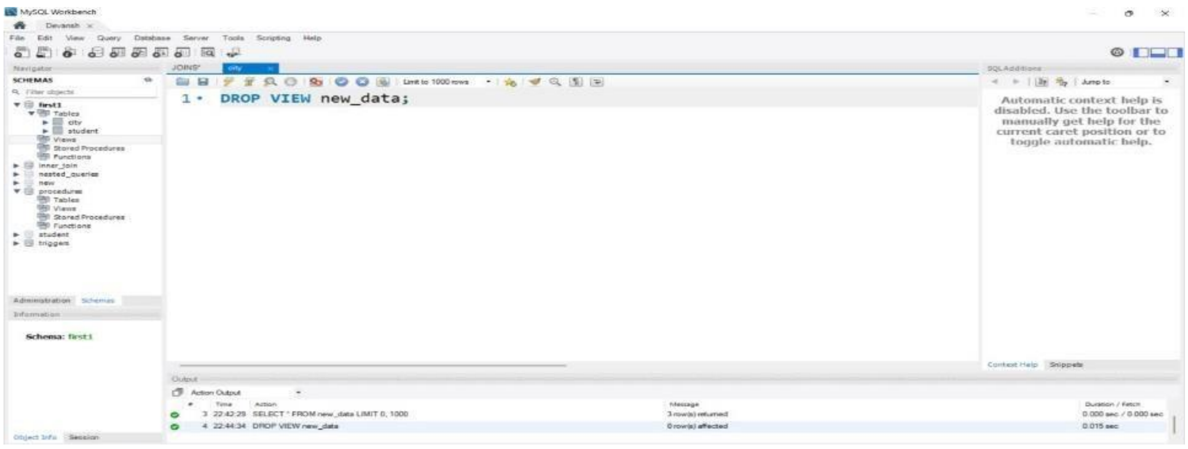
```
id  name         city
1   Ram Kumar    Agra
2   Salman Khan  Bhopal
3   Meera Khan   Delhi
```

```
#  Time      Action                                   Message              Duration / Fetch
2  22:41:53  RENAME TABLE student_data TO new_data    0 row(s) affected    0.015 sec
3  22:42:29  SELECT * FROM new_data LIMIT 0, 1000     3 row(s) returned    0.000 sec / 0.000 sec
```

IF we DELETE the VIEW then we use DROP VIEW:-

<u>QUERY</u>:-

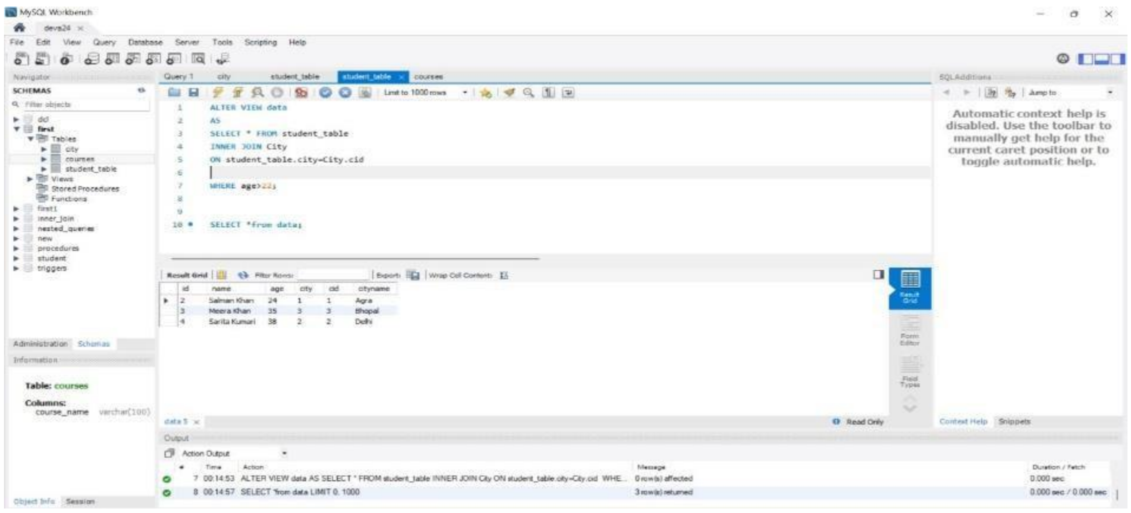
DROP VIEW new_data;



```
DROP VIEW new_data;
```

```
#  Time      Action                                   Message              Duration / Fetch
3  22:42:29  SELECT * FROM new_data LIMIT 0, 1000     3 row(s) returned    0.000 sec / 0.000 sec
4  22:44:34  DROP VIEW new_data                       0 row(s) affected    0.015 sec
```

ALTER IN VIEWS <u>QUERY</u>:- ALTER
VIEW data AS
SELECT * FROM student_table INNER JOIN City
ON student_table.city=City.cid WHERE age>22;
SELECT *from data;



```
ALTER VIEW data
AS
SELECT * FROM student_table
INNER JOIN City
ON student_table.city=City.cid

WHERE age>22;


SELECT *from data;
```

id	name	age	city	cid	cityname
2	Salman Khan	24	1	1	Agra
3	Meera Khan	35	3	3	Bhopal
4	Sarita Kumari	38	2	2	Delhi

#	Time	Action	Message	Duration / Fetch
7	00:14:53	ALTER VIEW data AS SELECT * FROM student_table INNER JOIN City ON student_table.city=City.cid WHE...	0 row(s) affected	0.000 sec
8	00:14:57	SELECT *from data LIMIT 0, 1000	3 row(s) returned	0.000 sec / 0.000 sec

Result: The program for Set Operators & Views in RDBMS is implemented successfully and output is verified.

Exp No:-9

Conditional & Iterative Statements

AIM – PL/SQL conditional and iterative statements.

Theory:- A program that supports a user interface may run a main loop that waits for, and then processes, user keystrokes (this doesn't apply to stored programs, however).Many mathematical algorithms can be implemented only by loops in computer programs.When processing a file, a program may loop through each record in the file and perform computations.A database program may loop through the rows returned by a SELECT statement.

QUERY:

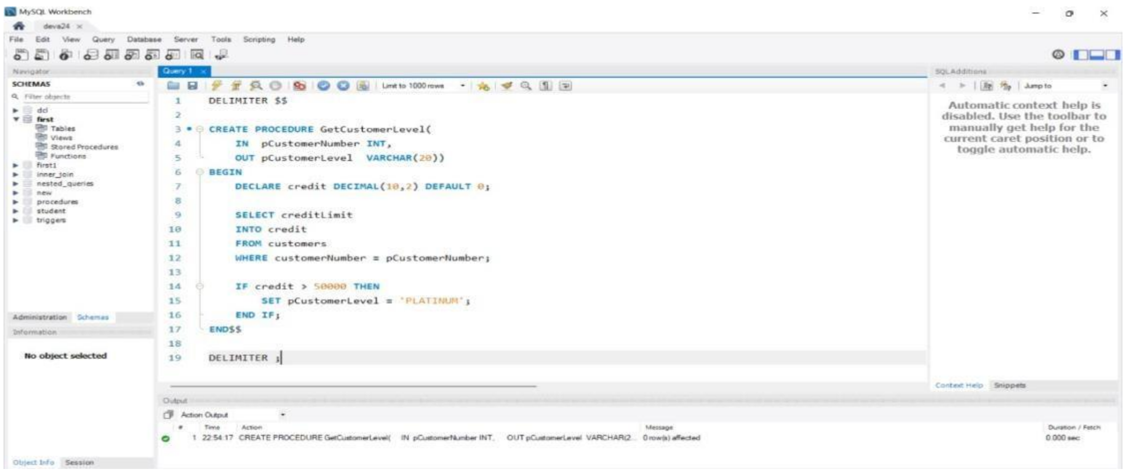
a) Conditional Statement

FOR CREATING PROCEDURE

```
DELIMITER $$
CREATE
PROCEDURE
GetCustomerLevel(
   IN pCustomerNumber INT,
   OUT pCustomerLevel
VARCHAR(20)) BEGIN DECLARE
   credit
DECIMAL(10,2) DEFAULT 0;
SELECT
   creditLimit INTO
   credit
   FROM
   customers
   WHERE
customerNumber              =
pCustomerNumber; IF credit
>    50000    THEN    SET
pCustomerLevel =
   'PLATINUM'; END IF;
END$$
DELIMIT
ER ;
```

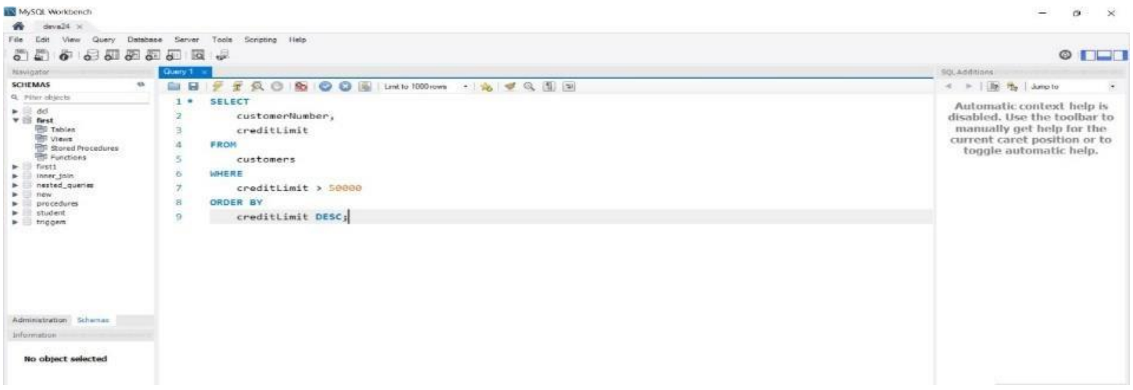
FOR EXECUTION

```
SELECT customerNum
   ber,
   creditLimit
FROM custo
mers
WHERE
   creditLimit >
50000 ORDER BY creditLimit
   DESC;
```

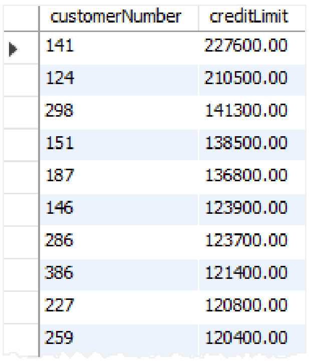


```
DELIMITER $$

CREATE PROCEDURE GetCustomerLevel(
    IN  pCustomerNumber INT,
    OUT pCustomerLevel  VARCHAR(20))
BEGIN
    DECLARE credit DECIMAL(10,2) DEFAULT 0;

    SELECT creditLimit
    INTO credit
    FROM customers
    WHERE customerNumber = pCustomerNumber;

    IF credit > 50000 THEN
        SET pCustomerLevel = 'PLATINUM';
    END IF;
END$$

DELIMITER ;
```

```
#  Time      Action                                                                                Message            Duration / Fetch
1  22:54:17  CREATE PROCEDURE GetCustomerLevel(   IN pCustomerNumber INT,   OUT pCustomerLevel VARCHAR(2...   0 row(s) affected   0.000 sec
```



```
SELECT
    customerNumber,
    creditLimit
FROM
    customers
WHERE
    creditLimit > 50000
ORDER BY
    creditLimit DESC;
```



customerNumber	creditLimit
141	227600.00
124	210500.00
298	141300.00
151	138500.00
187	136800.00
146	123900.00
286	123700.00
386	121400.00
227	120800.00
259	120400.00

Iterative Statement

CREATE TABLE calendars(id
INT AUTO_INCREMENT,
fulldate DATE UNIQUE, day TINYINT NOT NULL, month
TINYINT NOT NULL,
quarter TINYINT NOT NULL, year INT NOT NULL, PRIMARY KEY(id)
);

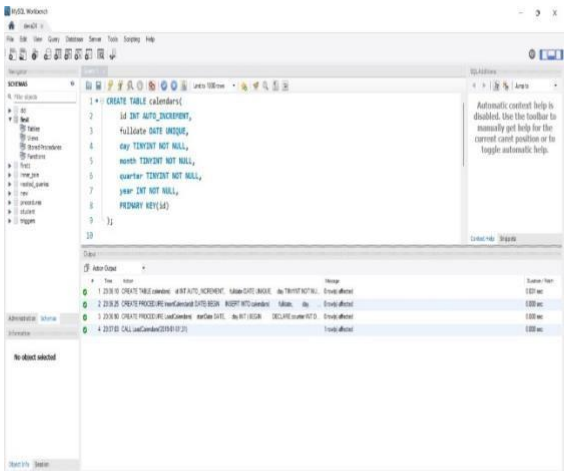
FOR CREATING PROCEDURE

DELIMITER $$
CREATE PROCEDURE InsertCalendar(dt DATE)
BEGIN
INSERT INTO calendars(fulldate,
day, month, quarter, year
) VALUES(
dt,
EXTRACT(DAY FROM dt), EXTRACT(MONTH FROM dt), EXTRACT(QUARTER FROM dt), EXTRACT(YEAR FROM dt)
);
END$$ DELIMITER ;

DELIMITER $$

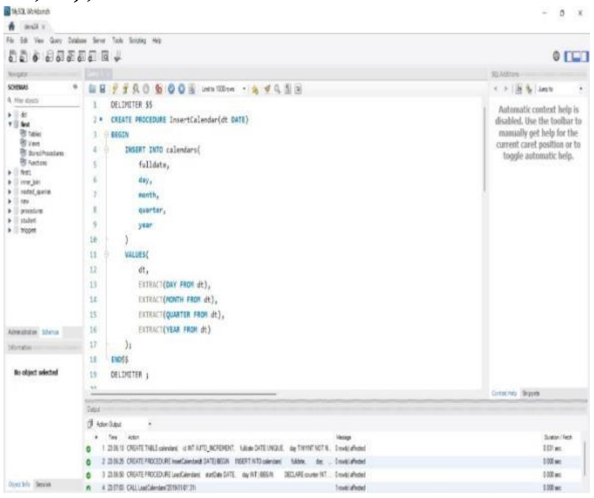
CREATE PROCEDURE LoadCalendars(startDate DATE,
day INT) BEGIN
DECLARE counter INT DEFAULT 1; DECLARE
dt DATE DEFAULT
startDate;
WHILE counter <= day DO CALL InsertCalendar(dt); SET counter = counter + 1; SET dt =
DATE_ADD(dt,INTERVAL 1 day); END WHILE;
END$$ DELIMITER ;

FOR EXECUTION: CALL LoadCalendars('2019-01-01',31);

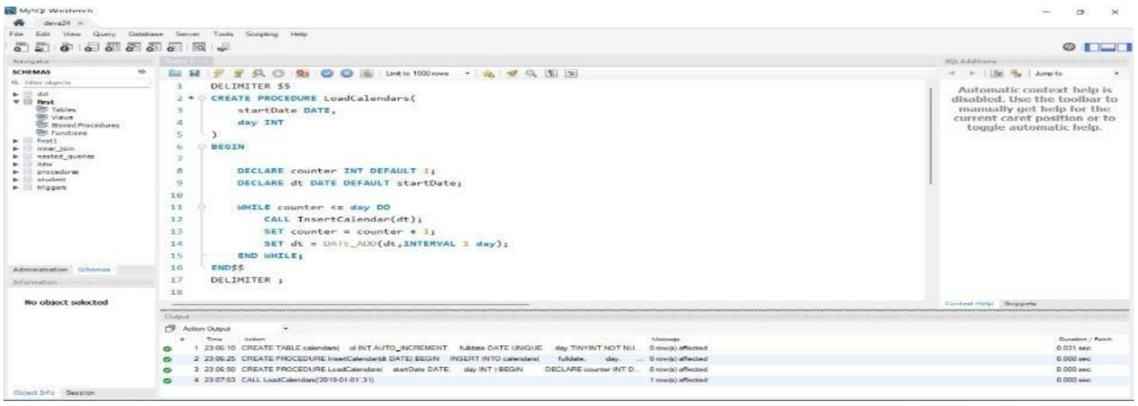


```
CREATE TABLE calendars(
    id INT AUTO_INCREMENT,
    fulldate DATE UNIQUE,
    day TINYINT NOT NULL,
    month TINYINT NOT NULL,
    quarter TINYINT NOT NULL,
    year INT NOT NULL,
    PRIMARY KEY(id)
);
```

```
1 23:06:10 CREATE TABLE calendars(   id INT AUTO_INCREMENT,   fulldate DATE UNIQUE,   day TINYINT NOT NU...   0 row(s) affected   0.031 sec
2 23:06:25 CREATE PROCEDURE InsertCalendar(dt DATE) BEGIN   INSERT INTO calendars(   fulldate,   day,   ...   0 row(s) affected   0.000 sec
3 23:06:50 CREATE PROCEDURE LoadCalendars(   startDate DATE,   day INT ) BEGIN   DECLARE counter INT D...   0 row(s) affected   0.000 sec
4 23:07:03 CALL LoadCalendars('2019-01-01',31)   1 row(s) affected   0.000 sec
```

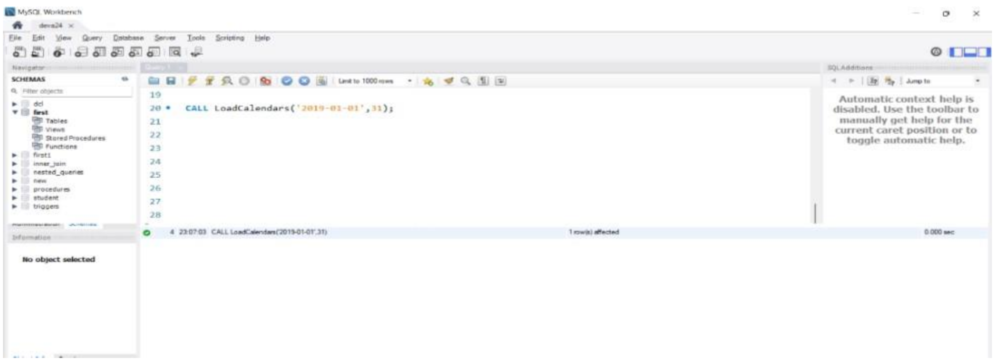


```
DELIMITER $$
CREATE PROCEDURE InsertCalendar(dt DATE)
BEGIN
    INSERT INTO calendars(
        fulldate,
        day,
        month,
        quarter,
        year
    )
    VALUES(
        dt,
        EXTRACT(DAY FROM dt),
        EXTRACT(MONTH FROM dt),
        EXTRACT(QUARTER FROM dt),
        EXTRACT(YEAR FROM dt)
    );
END$$
DELIMITER ;
```



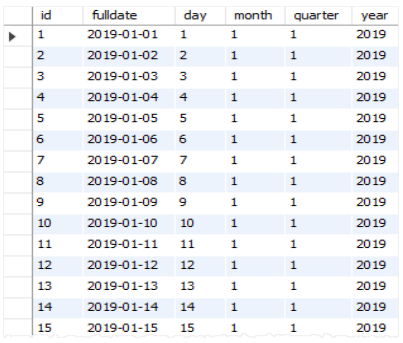
```
DELIMITER $$
CREATE PROCEDURE LoadCalendars(
    startDate DATE,
    day INT
)
BEGIN

    DECLARE counter INT DEFAULT 1;
    DECLARE dt DATE DEFAULT startDate;

    WHILE counter <= day DO
        CALL InsertCalendar(dt);
        SET counter = counter + 1;
        SET dt = DATE_ADD(dt,INTERVAL 1 day);
    END WHILE;
END$$
DELIMITER ;
```

```
#  Time      Action                                                                                              Message             Duration / Fetch
1  23:06:10  CREATE TABLE calendars(   id INT AUTO_INCREMENT    fulldate DATE UNIQUE    day TINYINT NOT NU...   0 row(s) affected   0.031 sec
2  23:06:25  CREATE PROCEDURE InsertCalendar(dt DATE) BEGIN   INSERT INTO calendars(   fulldate,   day,   ...   0 row(s) affected   0.000 sec
3  23:06:50  CREATE PROCEDURE LoadCalendars(   startDate DATE    day INT ) BEGIN    DECLARE counter INT D...   0 row(s) affected   0.000 sec
4  23:07:03  CALL LoadCalendars('2019-01-01',31)                                                            1 row(s) affected   0.000 sec
```



```
CALL LoadCalendars('2019-01-01',31);
```

```
4  23:07:03  CALL LoadCalendars('2019-01-01',31)    1 row(s) affected    0.000 sec
```



id	fulldate	day	month	quarter	year
1	2019-01-01	1	1	1	2019
2	2019-01-02	2	1	1	2019
3	2019-01-03	3	1	1	2019
4	2019-01-04	4	1	1	2019
5	2019-01-05	5	1	1	2019
6	2019-01-06	6	1	1	2019
7	2019-01-07	7	1	1	2019
8	2019-01-08	8	1	1	2019
9	2019-01-09	9	1	1	2019
10	2019-01-10	10	1	1	2019
11	2019-01-11	11	1	1	2019
12	2019-01-12	12	1	1	2019
13	2019-01-13	13	1	1	2019
14	2019-01-14	14	1	1	2019
15	2019-01-15	15	1	1	2019

RESULT: The program for PL/SQL conditional and iterative statements in RDBMS is implemented successfully and output is verified.

Exp No:-10

PL/SQL Procedures on sample exercise.

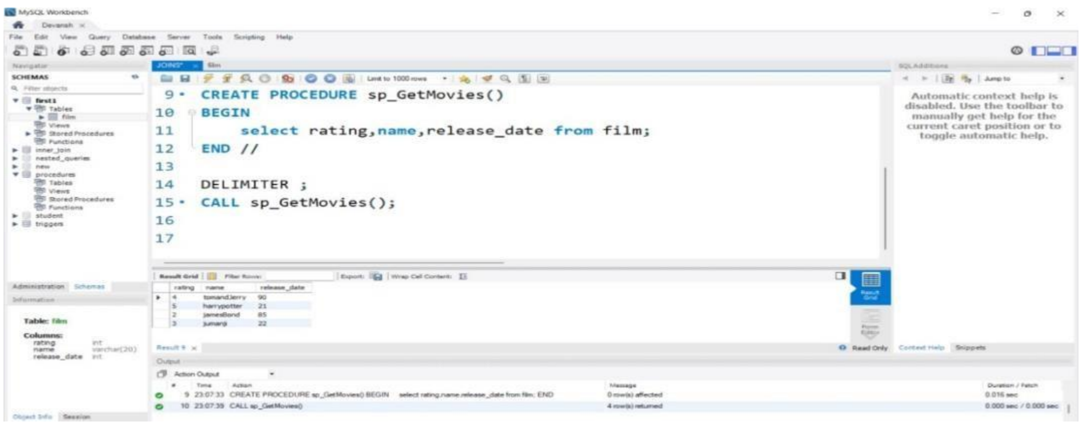
Aim:- To Execute Procedure in MySql

Theory:- A procedure (often called a stored procedure) is a collection of pre- compiled SQL statements stored inside the database. It is a subroutine or a subprogram in the regular computing language. A procedure always contains a name, parameter lists, and SQL statements. We can invoke the procedures by using triggers, other procedures and applications such as Java, Python, PHP, etc. It was first introduced in MySQL version 5. Presently, it can be supported by almost all relational database systems.

If we consider the enterprise application, we always need to perform specific tasks such as database cleanup, processing payroll, and many more on the database regularly. Such tasks involve multiple SQL statements for executing each task. This process might easy if we group these tasks into a single task. We can fulfill this requirement in MySQL by creating a stored procedure in our database.

Query:-

```
create table film(rating int ,name varchar(20),release_date int); insert into film values(4,'tomandJerry',90);
insert into film values(5,'harrypotter',21); insert into film values(2,'jamesBond',85); insert into film values(3,'jumanji',22);

DELIMITER //
CREATE PROCEDURE sp_GetMovies() BEGIN
select rating,name,release_date from film; END //

DELIMITER ;

CALL sp_GetMovies();
```



```
CREATE PROCEDURE sp_GetMovies()
BEGIN
    select rating,name,release_date from film;
END //

DELIMITER ;
CALL sp_GetMovies();
```

```
rating  name         release_date
4       tomandJerry  90
5       harrypotter  21
2       jamesBond    85
3       jumanji      22
```

```
#   Time      Action                                                                          Message             Duration / Fetch
9   23:07:33  CREATE PROCEDURE sp_GetMovies() BEGIN  select rating,name,release_date from film; END   0 row(s) affected   0.016 sec
10  23:07:39  CALL sp_GetMovies()                                                             4 row(s) returned   0.000 sec / 0.000 sec
```

RESULT: The program for PL/SQL Procedures on sample exercise in RDBMS is implemented successfully and output is verified.

Exp No:-11

PL/SQL Functions

AIM - PL/SQL functions

Theory:-

A function can be used as a part of SQL expression i.e. we can use them with select/update/merge commands. One most important characteristic of a function is that unlike procedures, it must return a value.

QUERY

CREATING TABLE FOR FUNCTION

```
CREATE TABLE employee( emp_id
INT,
fname     varchar(50),     lname
varchar(50),
start_date date
);
INSERT INTO
employee(emp_id,fname,lname,start_date) VALUES
(1,'Michael','Smith','2001-06-22'),
(2,'Susan', 'Barker','2002-09-12'),
(3,'Robert','Tvler','2000-02-09'),
(4,'Susan','Hawthorne','2002-04-24');
```

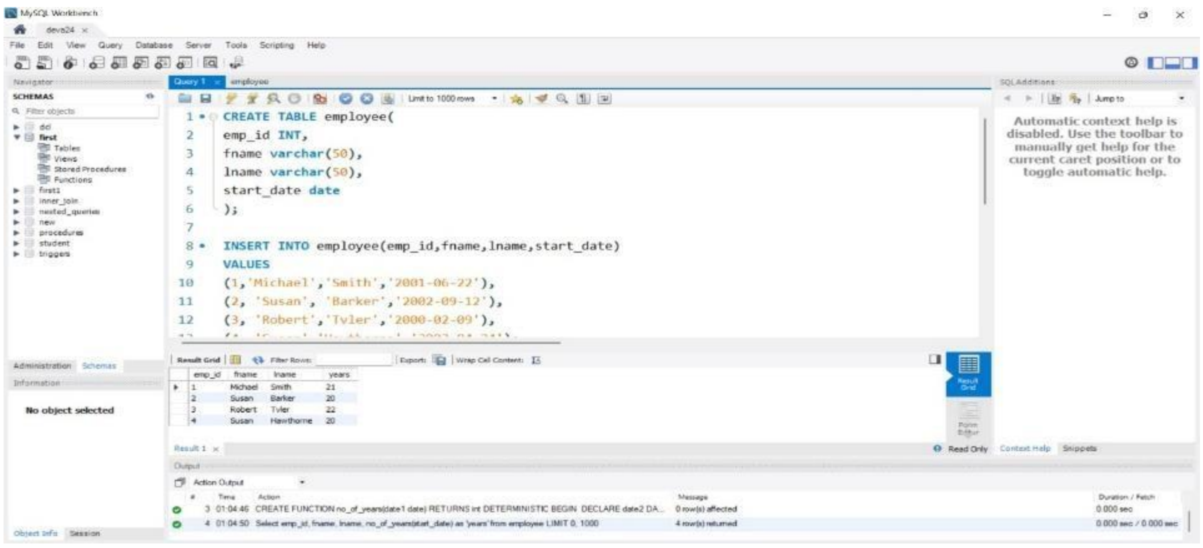
CREATING FUNCTION

```
DELIMITER //

CREATE FUNCTION no_of_years(date1 date) RETURNS int DETERMINISTIC BEGIN
 DECLARE date2 DATE; Select current_date()into date2; RETURN year(date2)-year(date1); END // DELIMITER
;
```

CALLING FUNCTION

```
Select emp_id, fname, lname, no_of_years(start_date) as 'years' from employee;
```

OUTPUT:-

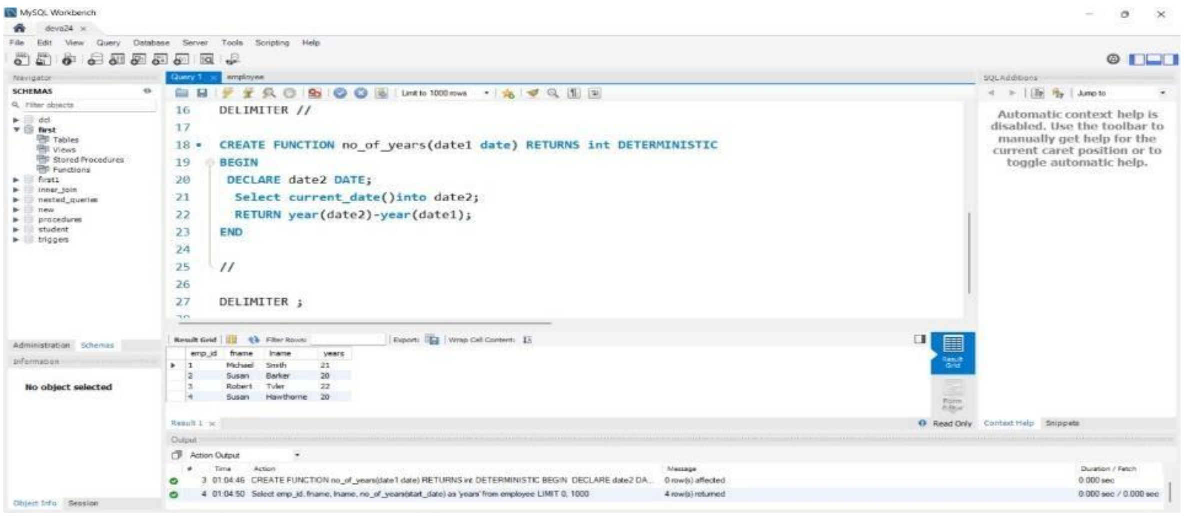


```
CREATE TABLE employee(
emp_id INT,
fname varchar(50),
lname varchar(50),
start_date date
);

INSERT INTO employee(emp_id,fname,lname,start_date)
VALUES
(1,'Michael','Smith','2001-06-22'),
(2, 'Susan', 'Barker','2002-09-12'),
(3, 'Robert','Tvler','2000-02-09'),
```

```
emp_id  fname    lname      years
1       Michael  Smith      21
2       Susan    Barker     20
3       Robert   Tvler      22
4       Susan    Hawthorne  20
```

```
#  Time      Action                                                                                  Message              Duration / Fetch
3  01:04:46  CREATE FUNCTION no_of_years(date1 date) RETURNS int DETERMINISTIC BEGIN DECLARE date2 DA...  0 row(s) affected    0.000 sec
4  01:04:50  Select emp_id, fname, lname, no_of_years(start_date) as 'years' from employee LIMIT 0, 1000  4 row(s) returned    0.000 sec / 0.000 sec
```

Automatic context help is disabled. Use the toolbar to manually get help for the current caret position or to toggle automatic help.



```
DELIMITER //

CREATE FUNCTION no_of_years(date1 date) RETURNS int DETERMINISTIC
BEGIN
 DECLARE date2 DATE;
  Select current_date()into date2;
  RETURN year(date2)-year(date1);
END

//

DELIMITER ;
```

emp_id	fname	lname	years
1	Michael	Smith	21
2	Susan	Barker	20
3	Robert	Tyler	22
4	Susan	Hawthorne	20

```
#  Time      Action                                                                                    Message            Duration / Fetch
3  01:04:46  CREATE FUNCTION no_of_years(date1 date) RETURNS int DETERMINISTIC BEGIN  DECLARE date2 DA...  0 row(s) affected  0.000 sec
4  01:04:50  Select emp_id, fname, lname, no_of_years(start_date) as 'years' from employee LIMIT 0, 1000   4 row(s) returned  0.000 sec / 0.000 sec
```

RESULT: The program for PL/SQL PL/SQL Functions in RDBMS is implemented successfully and output is verified.

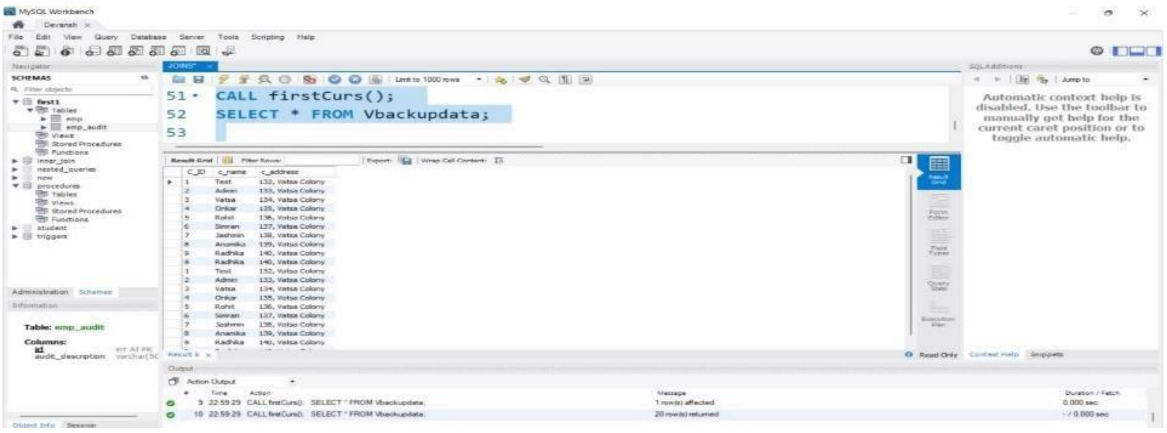
Exp No:-12
PL/SQL Cursors

Aim:- To Execute Cursors in MySql

Theory:- A cursor allows you to iterate a set of rows returned by a query and process each row individually. MySQL cursor is read-only, non-scrollable and asensitive. Read-only: you cannot update data in the underlying table through the cursor.

QUERY:-

```
CREATE TABLE GetVatsaCursor(
C_ID INT PRIMARY KEY AUTO_INCREMENT,
c_name VARCHAR(50), c_address VARCHAR(200)); CREATE TABLE Vbackupdata( C_ID INT, c_name
VARCHAR(50), c_address VARCHAR(200));
INSERT INTO GetVatsaCursor(c_name, c_address) VALUES('Test', '132, Vatsa Colony'),
('Admin', '133, Vatsa Colony'),
('Vatsa', '134, Vatsa Colony'),
('Onkar', '135, Vatsa Colony'), ('Rohit',
'136, Vatsa Colony'),
('Simran', '137, Vatsa Colony'), ('Jashmin', '138, Vatsa Colony'), ('Anamika', '139, Vatsa Colony'), ('Radhika', '140,
Vatsa Colony'); SELECT * FROM GetVatsaCursor; SELECT * FROM Vbackupdata; delimiter //
CREATE PROCEDURE firstCurs() BEGIN
DECLARE d INT DEFAULT 0; DECLARE c_id INT;
DECLARE c_name, c_address VARCHAR(20);
DECLARE Get_cur CURSOR FOR SELECT * FROM GetVatsaCursor; DECLARE CONTINUE HANDLER FOR
SQLSTATE '02000'
SET d = 1;
DECLARE CONTINUE HANDLER FOR SQLSTATE '23000' SET d = 1;
OPEN Get_cur; lbl: LOOP
IF d = 1 THEN
LEAVE lbl;
            END IF;
            IF NOT d = 1 THEN
            FETCH Get_cur INTO c_id, c_name, c_address;
            INSERT INTO Vbackupdata VALUES(c_id, c_name, c_address); END
            IF;
            END LOOP; CLOSE
            Get_cur;
            END;
            CALL firstCurs();
            SELECT * FROM Vbackupdata
```



```
CALL firstCurs();
SELECT * FROM Vbackupdata;
```

C_ID	c_name	c_address
1	Test	132, Vatsa Colony
2	Admin	133, Vatsa Colony
3	Vatsa	134, Vatsa Colony
4	Onkar	135, Vatsa Colony
5	Rohit	136, Vatsa Colony
6	Simran	137, Vatsa Colony
7	Jashmin	138, Vatsa Colony
8	Anamika	139, Vatsa Colony
9	Radhika	140, Vatsa Colony
9	Radhika	140, Vatsa Colony
1	Test	132, Vatsa Colony
2	Admin	133, Vatsa Colony
3	Vatsa	134, Vatsa Colony
4	Onkar	135, Vatsa Colony
5	Rohit	136, Vatsa Colony
6	Simran	137, Vatsa Colony
7	Jashmin	138, Vatsa Colony
8	Anamika	139, Vatsa Colony
9	Radhika	140, Vatsa Colony

```
#   Time      Action                                          Message              Duration / Fetch
9   22:59:29  CALL firstCurs(); SELECT * FROM Vbackupdata;    1 row(s) affected    0.000 sec
10  22:59:29  CALL firstCurs(); SELECT * FROM Vbackupdata;    20 row(s) returned   - / 0.000 sec
```

RESULT: The program for **PL/SQL Cursors** in RDBMS is implemented successfully and output is verified.

Exp No:-13

PL/SQL Exception Handling

Aim- PL/SQL exception handling.

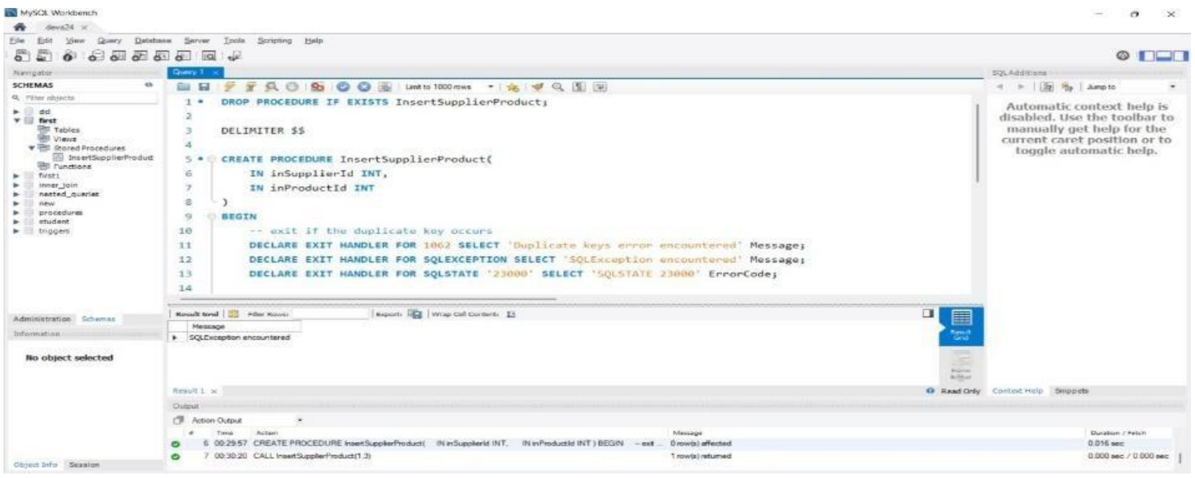
Theory:-

When an error occurs inside a stored procedure, it is important to handle it appropriately, such as continuing or exiting the current code block's execution, and issuing a meaningful error message.MySQL provides an easy way to define handlers that handle from general conditions such as warnings or exceptions to specific conditions e.g., specific error codes.

QUERY:-

```
DROP PROCEDURE IF EXISTS InsertSupplierProduct; DELIMITER $$
CREATE PROCEDURE InsertSupplierProduct( IN inSupplierId INT,
    IN inProductId INT
)
BEGIN

    -- exit if the duplicate key occurs

    DECLARE EXIT HANDLER FOR 1062 SELECT

'Duplicate keys error encountered' Message; DECLARE EXIT HANDLER
FOR SQLEXCEPTION

SELECT 'SQLException encountered' Message; DECLARE EXIT HANDLER
FOR SQLSTATE '23000' SELECT 'SQLSTATE 23000' ErrorCode;

    -- insert a new row into the SupplierProducts

    INSERT                    INTO                    SupplierProducts(supplierId,productId)
    VALUES(inSupplierId,inProductId); -- return the products supplied by the supplier id
    SELECT COUNT(*)

                    FROM SupplierProducts

                    WHERE supplierId = inSupplierId;
                    END$$
                    DELIMITER ;
```

OUTPUT



```
DROP PROCEDURE IF EXISTS InsertSupplierProduct;

DELIMITER $$

CREATE PROCEDURE InsertSupplierProduct(
    IN inSupplierId INT,
    IN inProductId INT
)
BEGIN
    -- exit if the duplicate key occurs
    DECLARE EXIT HANDLER FOR 1062 SELECT 'Duplicate keys error encountered' Message;
    DECLARE EXIT HANDLER FOR SQLEXCEPTION SELECT 'SQLException encountered' Message;
    DECLARE EXIT HANDLER FOR SQLSTATE '23000' SELECT 'SQLSTATE 23000' ErrorCode;
```

Message
SQLException encountered

```
#  Time      Action                                                                           Message              Duration / Fetch
6  00:29:57  CREATE PROCEDURE InsertSupplierProduct(  IN inSupplierId INT,  IN inProductId INT ) BEGIN  -- exit ...  0 row(s) affected  0.016 sec
7  00:30:20  CALL InsertSupplierProduct(1,3)                                                  1 row(s) returned    0.000 sec / 0.000 sec
```

RESULT: The program for **PL/SQL EXCEPTION handling** in RDBMS is implemented successfully and output is verified.

Exp No:-14
PL/SQL Trigger

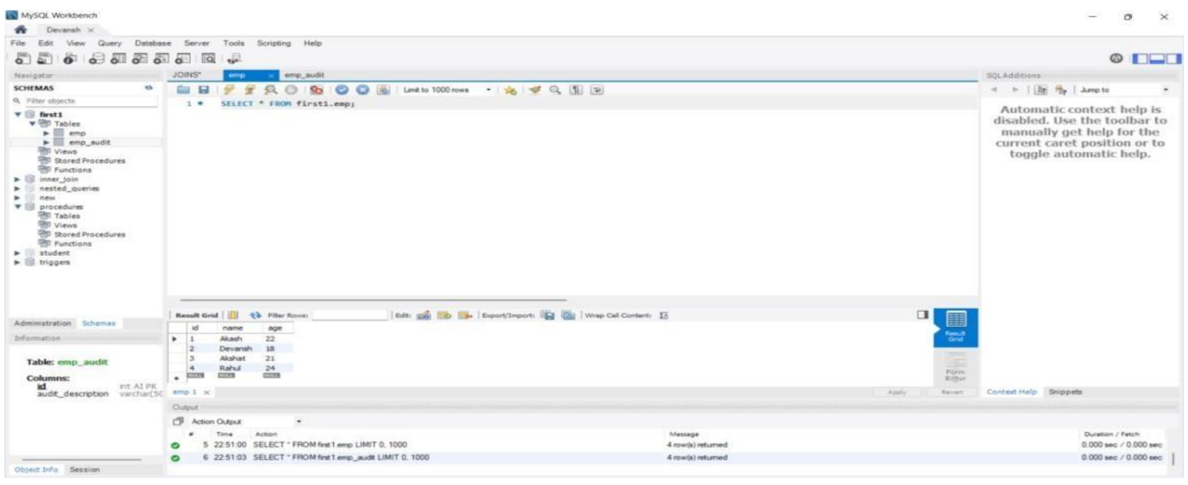
Aim:- To Execute Triggers in MYsql

Theory:- A trigger in MySQL is a set of SQL statements that reside in a system catalog. It is a special type of stored procedure that is invoked automatically in response to an event. Each trigger is associated with a table, which is activated on any DML statement such as INSERT, UPDATE, or DELETE.

A trigger is called a special procedure because it cannot be called directly like a stored procedure. The main difference between the trigger and procedure is that a trigger is called automatically when a data modification event is made against a table. In contrast, a stored procedure must be called explicitly.

QUERY:-

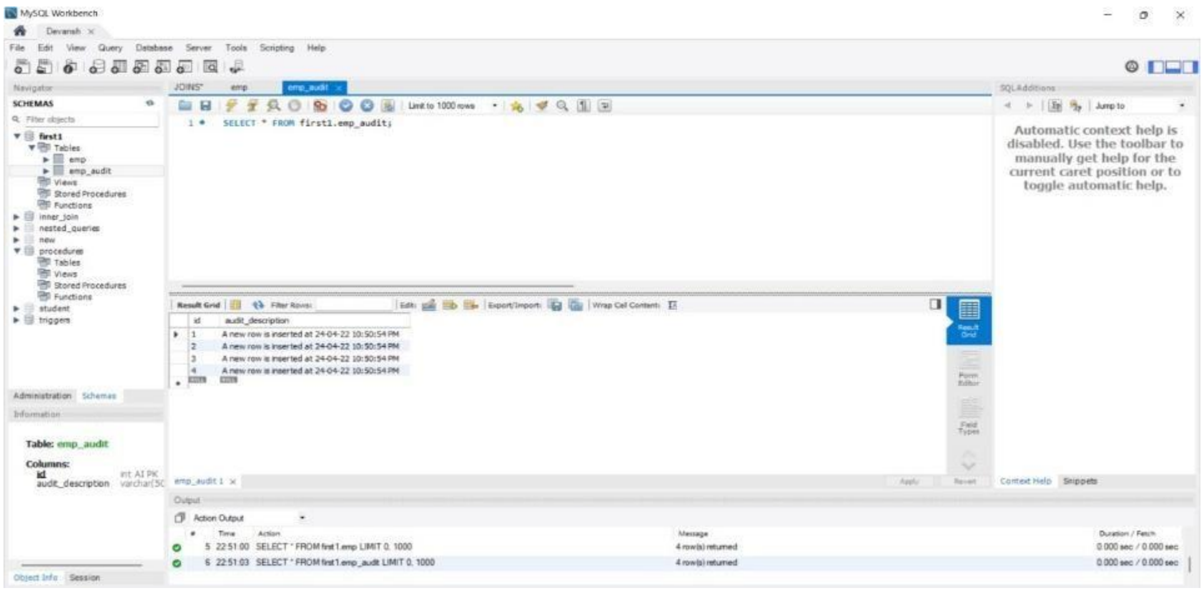
```
CREATE  TABLE  emp(  id
INT      PRIMARY      KEY
AUTO_INCREMENT,
name  VARCHAR(50),  age
INT
);
CREATE TABLE emp_audit(
id INT PRIMARY KEY AUTO_INCREMENT,
audit_description VARCHAR(500)
);
DELIMITER //
CREATE TRIGGER
tr_AfterInsetEmp AFTER INSERT
ON emp
FOR EACH
ROW BEGIN
INSERT INTO emp_audit

VALUES(null,concat('newrow',date_format(now(),'%d-%m-%y %h:%i:%s %p'))); END//
DELIMITER ; INSERT INTO emp VALUES
(null,'Akash',22),
(null,'Devansh',18),
(null,'Akshat',21),
(null,'Rahul',24);
```



```
SELECT * FROM first1.emp;
```

id	name	age
1	Akash	22
2	Devansh	18
3	Akshat	21
4	Rahul	24

```
#  Time      Action                                        Message            Duration / Fetch
5  22:51:00  SELECT * FROM first1.emp LIMIT 0, 1000        4 row(s) returned  0.000 sec / 0.000 sec
6  22:51:03  SELECT * FROM first1.emp_audit LIMIT 0, 1000  4 row(s) returned  0.000 sec / 0.000 sec
```



```
SELECT * FROM first1.emp_audit;
```

```
id  audit_description
1   A new row is inserted at 24-04-22 10:50:54 PM
2   A new row is inserted at 24-04-22 10:50:54 PM
3   A new row is inserted at 24-04-22 10:50:54 PM
4   A new row is inserted at 24-04-22 10:50:54 PM
```

Table: emp_audit

Columns:
id int AI PK
audit_description varchar(5C

```
#  Time      Action                                          Message            Duration / Fetch
5  22:51:00  SELECT * FROM first1.emp LIMIT 0, 1000          4 row(s) returned  0.000 sec / 0.000 sec
6  22:51:03  SELECT * FROM first1.emp_audit LIMIT 0, 1000    4 row(s) returned  0.000 sec / 0.000 sec
```

MySQL Workbench
File Edit View Query Database Server Tools Scripting Help
Automatic context help is disabled. Use the toolbar to manually get help for the current caret position or to toggle automatic help.

RESULT: The program for **PL/SQL trigger** in RDBMS is implemented successfully and output is verified.

Exp No:-15

Frame and Execute PL/SQL Cursor & Exceptional Handling

AIM –Frame and execute all queries for a project: Home renting system database

INTRODUCTION:-The Home Rental System is Searching in Based on the Apartment House for rent in metropolitan cities. The Home Rental System is Based on the Owners and the Customers. The Owner is updated on the Apartment details, and rent details. The Customer is details about the Room space, Room rent and the Address Details also.

PROBLEM STATEMENT

There is no properly allocate home and the system is not easily arranges according to their user interest. And also the home rental management system almost is done through the manual system. The administrative system doesn't have the facility to make home rental
management system through online and the most time the work done through illegal intermediate personwithout awareness of the administrative and this make more complex and more cost to find home for the customer. This leads to customer in to more trouble, cost, dishonest and time wastage.

The problem found in the current system:

- Complexity of finding home is not easy and more tedious. And also Extra money to find home.
- The system needs more human power.
- The user cannot get information about home when they need.
- There is too match time consumption find home

OBJECTIVE:

The main objective of the system is to develop online home rental management system for wolkite city

Specific objectives

In order to attain the general objective, the following are the list of specific objectives:

- To facilitate home record keeping for who wants home and for administrative management system.
- Prepare an online home rental system for the home finders
- To reduce the travel costs and other unnecessary expenses of the buyer.
- Reduce the role of the broker, thus, providing protection from frauds related to papers.

MODULES

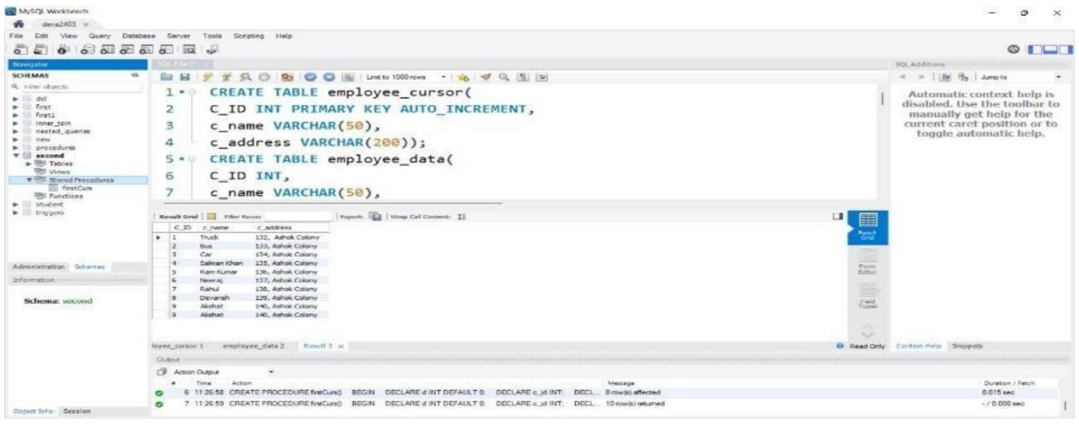
- Tenant
- Owner
- Contract
- Payment

OPERATION:

First will be the login page where the User will login using their ID and Password.
Next the user will enter the details like the requirements of the house he needs and contact details The other ebd user who wants to rent their house can also give their details He can also search the houses available in a particular area.
Whoever has the required spcifications of the house can contact using the details provided.
There is also a chatting option if they want to communicate with each other. There is also an option of video calling. After providing the details in the website, a contract will be generated which they can download.

OUTPUT:-

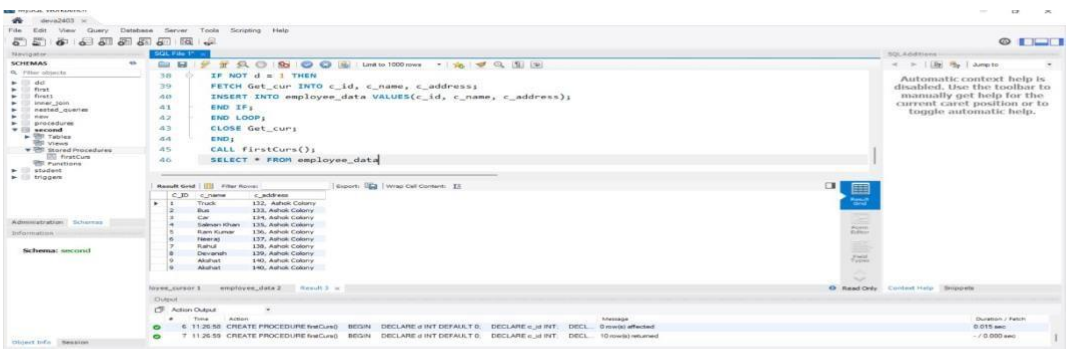
a) Cursor Output



```
CREATE TABLE employee_cursor(
C_ID INT PRIMARY KEY AUTO_INCREMENT,
c_name VARCHAR(50),
c_address VARCHAR(200));
CREATE TABLE employee_data(
C_ID INT,
c_name VARCHAR(50),
```

C_ID	c_name	c_address
1	Truck	132, Ashok Colony
2	Bus	133, Ashok Colony
3	Car	134, Ashok Colony
4	Salman Khan	135, Ashok Colony
5	Ram Kumar	136, Ashok Colony
6	Neeraj	137, Ashok Colony
7	Rahul	138, Ashok Colony
8	Devansh	139, Ashok Colony
9	Akshat	140, Ashok Colony
9	Akshat	140, Ashok Colony

```
#  Time      Action                                                                          Message              Duration / Fetch
6  11:26:58  CREATE PROCEDURE firstCurs()  BEGIN  DECLARE d INT DEFAULT 0;  DECLARE c_id INT;  DECL...  0 row(s) affected    0.015 sec
7  11:26:59  CREATE PROCEDURE firstCurs()  BEGIN  DECLARE d INT DEFAULT 0;  DECLARE c_id INT;  DECL...  10 row(s) returned   - / 0.000 sec
```

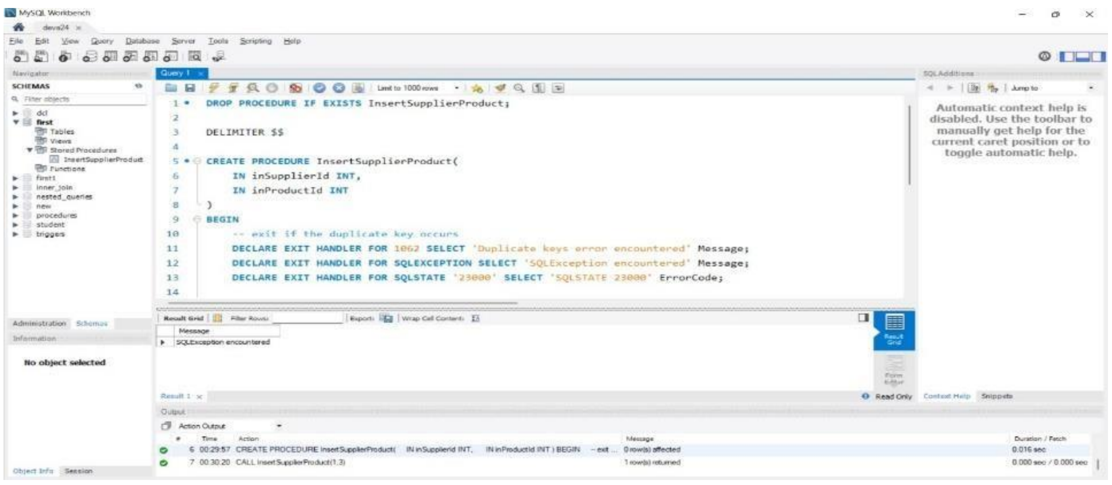


```
IF NOT d = 1 THEN
FETCH Get_cur INTO c_id, c_name, c_address;
INSERT INTO employee_data VALUES(c_id, c_name, c_address);
END IF;
END LOOP;
CLOSE Get_cur;
END;
CALL firstCurs();
SELECT * FROM employee_data
```

C_ID	c_name	c_address
1	Truck	132, Ashok Colony
2	Bus	133, Ashok Colony
3	Car	134, Ashok Colony
4	Salman Khan	135, Ashok Colony
5	Ram Kumar	136, Ashok Colony
6	Neeraj	137, Ashok Colony
7	Rahul	138, Ashok Colony
8	Devansh	139, Ashok Colony
9	Akshat	140, Ashok Colony
9	Akshat	140, Ashok Colony

```
#  Time      Action                                                                          Message              Duration / Fetch
6  11:26:58  CREATE PROCEDURE firstCurs()  BEGIN  DECLARE d INT DEFAULT 0;  DECLARE c_id INT;  DECL...  0 row(s) affected    0.015 sec
7  11:26:59  CREATE PROCEDURE firstCurs()  BEGIN  DECLARE d INT DEFAULT 0;  DECLARE c_id INT;  DECL...  10 row(s) returned   - / 0.000 sec
```

b) Exception Handling



```
DROP PROCEDURE IF EXISTS InsertSupplierProduct;

DELIMITER $$

CREATE PROCEDURE InsertSupplierProduct(
    IN inSupplierId INT,
    IN inProductId INT
)
BEGIN
    -- exit if the duplicate key occurs
    DECLARE EXIT HANDLER FOR 1062 SELECT 'Duplicate keys error encountered' Message;
    DECLARE EXIT HANDLER FOR SQLEXCEPTION SELECT 'SQLException encountered' Message;
    DECLARE EXIT HANDLER FOR SQLSTATE '23000' SELECT 'SQLSTATE 23000' ErrorCode;

```

Message
SQLException encountered

```
#  Time      Action                                                                                  Message             Duration / Fetch
6  00:29:57  CREATE PROCEDURE InsertSupplierProduct(  IN inSupplierId INT,  IN inProductId INT ) BEGIN  -- exit ...  0 row(s) affected   0.016 sec
7  00:30:20  CALL InsertSupplierProduct(1,3)                                                           1 row(s) returned   0.000 sec / 0.000 sec
```

RESULT: The program for Frame and Execute PL/SQL Cursor & Exceptional Handling in RDBMS is implemented successfully and output is verified.