

**SRM Institute of Science and Technology  
NCR Campus, Modinagar, Ghaziabad**

**BACHELOR OF TECHNOLOGY**

*in*

**COMPUTER SCIENCE AND ENGINEERING**



**DATABASE MANAGEMENT SYSTEMS (18CSC303J)**

NAME	:	Vatsal Goel
REG. NO.	:	RA1911028030012
DEGREE/BRANCH/SECTION	:	B.TECH/CSE/C
YEAR/SEMESTER	:	3rd/6 <sup>th</sup>
SESSION	:	JAN- 2022 - MAY 2022

**SRM Institute of Science and Technology**  
**NCR Campus, Modinagar**

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Reg. No.	R	A	1	9	1	1	0	2	8	0	3	0	0	1	2
----------	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

**BONAFIDE CERTIFICATE**

It is to be certified that the bonafide practical record submitted by **Vatsal Goel** of 6<sup>th</sup> Semester/ 3<sup>rd</sup> Year for Bachelor of Technology degree in the Department of Computer Science and Engineering, NCR Campus, SRM Institute of Science and Technology, has been done for the course Database Management Systems(18CSC303J), during the academic year session JAN- 2022 – MAY 2022.

**Head of the Department**

---

DR. R. P. MAHAPATRA(HOD)  
Professor  
Department of CSE

**Lab In-charge**

---

DR. ANNA ALPHY  
Assistant Professor  
Department of CSE

*Submitted for the University Examination held on \_\_\_\_\_.*

---

Examiner 1

---

Examiner 2

# **INDEX**

<b>Exp. No.</b>	<b>Name of Experiment</b>	<b>Page No.</b>	<b>Date</b>	<b>Signature</b>
1.	SQL Data Definition Language Commands on sample exercise			
2.	SQL Data Manipulation Language Commands			
3.	SQL Data Control Language Commands and Transaction control commands to the sample exercises			
4.	Inbuilt functions in SQL on sample			
5.	Construct a ER Model for the application to be constructed to a database			
6.	Nested Queries on sample exercise			
7.	Join Queries on sample exercise			
8.	Set operators & Views			
9.	PL/SQL Conditional and Iterative Statements			
10.	PL/SQL Procedures on sample exercise			
11.	PL/SQL Functions			
12.	PL/SQL Cursors			
13.	PL/SQL Exception Handling			
14.	PL/SQL Trigger			
15.	Frame and exercise the appropriate PL/SQL Cursors and Exceptional Handling for the project			

Exp No:-1

Date:-

## Experiment:-DDL Commands

Aim:- SQL data definition:-

- Create command
- Alter command
- Drop
- Truncate

Theory:-

- o DDL changes the structure of the table like creating a table, deleting a table, altering a table, etc.
- o 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)  
)
```

OUTPUT:-

The screenshot shows the MySQL Workbench interface. At the top, there's a toolbar with various icons for database management. Below the toolbar is a 'Result Grid' window. Inside the grid, there's a table structure with four columns: 'id', 'name', 'occupation', and 'age'. A single row is present with all values set to 'NULL'. The bottom of the window shows the table name 'employee\_table1' and an 'Output' section. On the right side of the interface, there's a vertical sidebar labeled 'Result Grid'.

	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;
```

The screenshot shows the MySQL Workbench interface. In the top query editor, the command `DROP TABLE employee_table;` is run. Below it, the command `CREATE TABLE employee_table (id int NOT NULL AUTO_INCREMENT, name varchar(45) NOT NULL, occupation varchar(35), age int);` is run. The output pane shows the execution details for both commands.

```

1 18:27:26 CREATE TABLE employee_table( id int NOT NULL AUTO_INCREMENT, name varchar(45) NOT NULL, occupation varchar(35), age int)
2 18:27:34 SELECT * FROM first1.employee_table LIMIT 0,1000
3 18:43:16 DROP TABLE employee_table

```

**c. 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.

### QUERY:-

```

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

```

The screenshot shows the MySQL Workbench interface. The command `ALTER TABLE employee_table MODIFY occupation varchar(35) AFTER age;` is run. The output pane shows the execution details for this command.

```

4 19:01:27 SELECT * FROM first1.employee_table LIMIT 0,1000
5 19:04:11 ALTER TABLE employee_table MODIFY occupation varchar(35) AFTER age
6 19:04:14 SELECT * FROM first1.employee_table LIMIT 0,1000
7 19:04:54 ALTER TABLE employee_table MODIFY occupation varchar(35) AFTER age
8 19:04:56 SELECT * FROM first1.employee_table LIMIT 0,1000

```

MySQL Workbench

File Edit View Query Database Server Tools Scripting Help

INNER JOIN employee\_table employee\_table employee\_table

1 • SELECT \* FROM first.employee\_table;

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

Result Grid | Filter Rows | Edit | Export/Import | Wrap Cell Content |

Table: employee\_table

Columns:

- id** int AI PK
- name** varchar(255)
- occupation** varchar(255)
- age** int

Action Output

Time Action Message Duration / Fetch

1 19:34:11 ALTER TABLE employee\_table ADD id int AUTO\_INCREMENT AFTER name 0 rows affected 0.000 sec / 0.000 sec

2 19:34:11 ALTER TABLE employee\_table ADD occupation varchar(255) AFTER name 0 rows affected 0.000 sec / 0.000 sec

3 19:34:11 SELECT \* FROM first.employee\_table LIMIT 0, 1000 10 rows returned 0.000 sec / 0.000 sec

4 19:34:11 ALTER TABLE employee\_table MODIFY occupation varchar(255) AFTER age 0 rows affected 0.000 sec / 0.000 sec

5 19:34:11 ALTER TABLE employee\_table MODIFY occupation varchar(255) AFTER id 0 rows affected 0.000 sec / 0.000 sec

6 19:34:11 SELECT \* FROM first.employee\_table LIMIT 0, 1000 10 rows returned 0.000 sec / 0.000 sec

7 19:34:11 ALTER TABLE employee\_table MODIFY occupation varchar(255) AFTER age 0 rows affected 0.000 sec / 0.000 sec

8 19:34:11 SELECT \* FROM first.employee\_table LIMIT 0, 1000 10 rows returned 0.000 sec / 0.000 sec

## ALTER TABLE-ADD COLUMN

Query:-

ALTER table employee  
ADD city VARCHAR(255);

MySQL Workbench

File Edit View Query Database Server Tools Scripting Help

Query 1 employee employee

1 • SELECT \* FROM first.employee;

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

Result Grid | Filter Rows | Edit | Export/Import | Wrap Cell Content |

Table: employee

Columns:

- id** int AI PK
- name** varchar(255)
- occupation** varchar(255)
- age** int

Action Output

Time Action Message Duration / Fetch

1 22:06:55 ALTER table employee ADD city VARCHAR(255) 0 rows affected 0.001 sec / 0.000 sec

2 22:06:59 SELECT \* FROM first.employee LIMIT 0, 1000 5 rows returned 0.000 sec / 0.000 sec

## ALTER TABLE-DROP COLUMN:-

Query:-

ALTER table employee  
DROP column occupation;

The screenshot shows the MySQL Workbench interface. On the left, the 'Schemas' tree view shows the 'test' schema with its tables, views, and other database objects. The 'Tables' section under 'test' includes 'employee'. In the center, the 'Query Editor' tab is active, displaying the query: 'SELECT \* FROM test.employee;'. Below the query editor is a 'Result Grid' showing the data from the 'employee' table:

	id	name	age	city
1	Mark Andre	22	2000	London
2	Satish Khan	22	2000	London
3	Hema Khan	17	2000	London
4	Gurdyal Singh	20	2000	London
5	John Abraham	20	2000	London

At the bottom of the interface, the status bar displays the message: '6 22:12:48 ALTER table employee DROP column occupation'.

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

### QUERY:-

```
TRUNCATE TABLE employee_table;
```

The screenshot shows the MySQL Workbench interface after executing a TRUNCATE operation. The 'Output' pane at the bottom displays the command: 'TRUNCATE TABLE employee\_table1;'. The 'Result Grid' pane shows the 'employee\_table1' table structure with four columns: 'id', 'name', 'occupation', and 'age'. All four columns have their values set to 'NULL'.

	id	name	occupation	age
*	NULL	NULL	NULL	NULL

Exp No:-2

Date:-

## **EXPERIMENT:-DML Commands**

**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

STEP 1: Start

STEP 2: Create the table with its essential attributes. STEP 3: Insert the record into table

STEP 4: Update the existing records into the table STEP 5: Delete the records in to the table.

STEP 6: Select the records

#### **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);
```

The screenshot shows the MySQL Workbench interface. In the left sidebar, under the 'Schemas' section, the 'first1' schema is selected, and the 'Tables' section shows the 'employee\_table'. The main pane displays the results of the following SQL query:

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

The result grid shows the following data:

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

Below the result grid, the 'Action Output' pane shows the following log entries:

- 1 19:11:17 CREATE TABLE employee\_table( id int NOT NULL AUTO\_INCREMENT, name varchar(45) NOT NULL... 0 row(s) affected Duration / Fetch 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 Duration / Fetch 0.000 sec
- 3 19:11:25 SELECT \* FROM first1.employee\_table LIMIT 0,1000 5 row(s) returned Duration / Fetch 0.000 sec / 0.000 sec
- 4 19:15:11 SELECT \* FROM first1.employee\_table LIMIT 0,1000 5 row(s) returned Duration / Fetch 0.016 sec / 0.000 sec

## 2. Update Command:-

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

The screenshot shows the MySQL Workbench interface. In the left sidebar, under the 'Schemas' section, the 'first1' schema is selected, and the 'Tables' section shows the 'employee\_table'. The main pane displays the results of the following SQL query:

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

The result grid shows the following data:

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

Below the result grid, the 'Action Output' pane shows the following log entries:

- 1 19:11:17 CREATE TABLE employee\_table( id int NOT NULL AUTO\_INCREMENT, name varchar(45) NOT NULL... 0 row(s) affected Duration / Fetch 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 Duration / Fetch 0.000 sec
- 3 19:11:25 SELECT \* FROM first1.employee\_table LIMIT 0,1000 5 row(s) returned Duration / Fetch 0.000 sec / 0.000 sec
- 4 19:15:11 SELECT \* FROM first1.employee\_table LIMIT 0,1000 5 row(s) returned Duration / Fetch 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 Duration / Fetch 0.000 sec
- 6 19:21:04 SELECT \* FROM first1.employee\_table LIMIT 0,1000 5 row(s) returned Duration / Fetch 0.000 sec / 0.000 sec

### 3.Select Command:-

The screenshot shows the MySQL Workbench interface. In the top navigation bar, 'INNER JOIN' is selected between 'employee\_table' and 'employee\_table'. Below the navigation bar, a query window displays the following SQL statement:

```
1 • SELECT * FROM employee_table;
```

The results grid shows the following data:

	id	name	occupation	age
1	1	Ram kumar	Chef	21
2	2	Salmahan	Software Developer	22
3	3	Meera Khan	Government Job	17
4	4	Santa Kumari	Teacher	29
5	5	Anil Kapoor	Dancer	20

Below the results grid, the 'Output' pane shows the execution history:

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;

The screenshot shows the MySQL Workbench interface. In the top navigation bar, 'INNER JOIN' is selected between 'employee\_table' and 'employee\_table'. Below the navigation bar, a query window displays the following SQL statement:

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

The results grid shows the following data:

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

Below the results grid, the 'Output' pane shows the execution history:

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;

The screenshot shows the MySQL Workbench interface. In the top-left pane, the Navigator displays the schema 'frst1' with its tables, including 'employee\_table'. The central pane contains the SQL editor with the following query:

```
1 • SELECT * FROM employee_table
2 WHERE age>17;
```

The Result Grid shows the following data:

ID	Name	Occupation	Age
1	Ram Kumar	Private Job	21
2	Salman Khan	Software Developer	22
4	Santa Kumar	Teacher	19
5	Anil Kapoor	Dancer	20

The bottom pane shows the Action Output log:

Time	Action	Message	Duration / Fetch
1 19:35:59	CREATE TABLE employee_table( id int NOT NULL AUTO_INCREMENT, name varchar(45) NOT NULL, occupation varchar(45) NOT NULL, age int NOT NULL ) ENGINE=InnoDB DEFAULT CHARSET=utf8;	0 rows affected	0.031 sec
2 19:36:09	INSERT INTO employee_table(id,name,occupation,age) VALUES (null,'Ran kumar','Private Job',21), (null,'Salma...','Software Dev...','22');	5 rows(5) affected Records: 5 Duplicates: 0 Warnings: 0	0.000 sec
3 19:36:17	SELECT * FROM frst1.employee_table LIMIT 0, 1000	5 rows(5) returned	0.000 sec / 0.000 sec
4 19:37:21	SELECT * FROM employee_table WHERE age>17 LIMIT 0, 1000	4 rows(4) returned	0.016 sec / 0.000 sec

### 2. SELECT With ORDER BY:-

SELECT \* FROM employee\_table  
WHERE age>17  
ORDER BY name;

The screenshot shows the MySQL Workbench interface. In the top-left pane, the Navigator displays the schema 'frst1' with its tables, including 'employee\_table'. The central pane contains the SQL editor with the following query:

```
1 • SELECT * FROM employee_table
2 WHERE age>17
3 ORDER BY name;
```

The Result Grid shows the following data:

ID	Name	Occupation	Age
4	Santa Kumar	Teacher	19
5	Anil Kapoor	Dancer	20
1	Ram Kumar	Private Job	21
2	Salman Khan	Software Developer	22

The bottom pane shows the Action Output log:

Time	Action	Message	Duration / Fetch
1 19:35:59	CREATE TABLE employee_table( id int NOT NULL AUTO_INCREMENT, name varchar(45) NOT NULL, occupation varchar(45) NOT NULL, age int NOT NULL ) ENGINE=InnoDB DEFAULT CHARSET=utf8;	0 rows affected	0.031 sec
2 19:36:09	INSERT INTO employee_table(id,name,occupation,age) VALUES (null,'Ran kumar','Private Job',21), (null,'Salma...','Software Dev...','22');	5 rows(5) affected Records: 5 Duplicates: 0 Warnings: 0	0.000 sec
3 19:36:17	SELECT * FROM frst1.employee_table LIMIT 0, 1000	5 rows(5) returned	0.000 sec / 0.000 sec
4 19:37:21	SELECT * FROM employee_table WHERE age>17 LIMIT 0, 1000	4 rows(4) 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 rows(4) returned	0.016 sec / 0.000 sec

### **3. SELECT WITH DISTINCT:-**

#### **QUERY:-**

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

The screenshot shows the MySQL Workbench interface. In the top navigation bar, 'Devarsh' is selected. The 'Query' tab is active, displaying the query: '1 • SELECT DISTINCT age FROM employee\_table;'. The results are shown in a 'Result Grid' table:

age
22
19
20

On the right side of the interface, there is a note: 'Automatic context help is disabled. Use the toolbar to manually get help for the current caret position or to toggle automatic help.' Below the results, the 'Object Info' and 'Session' tabs are visible.

### **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.

## QUERY:-

```
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');
```

The screenshot shows the MySQL Workbench interface with the 'Devarash' database selected. In the Navigator pane, under the 'first1' schema, the 'Tables' section shows the 'employee' table. The SQL editor pane contains the query: `SELECT * FROM first1.employee;`. The Result Grid pane displays the following data:

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

The Output pane shows the execution history:

#	Time	Action	Message	Duration / Fetch
4	20:56:44	INSERT INTO employee(id,name,age,gender) VALUES (1,'Akash',21,'M'), (2,'Rahul',22,'M'), (3,'Devansh',18,'M'), (4,'Amit',19,'M'), (5,'Akash',24,'M')	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

Exp No:-3

Date:-

## **EXPERIMENT:-DCL Commands And TCL Commands**

**Aim:-** SQL data control language and transaction control Language

**Theory:-** 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

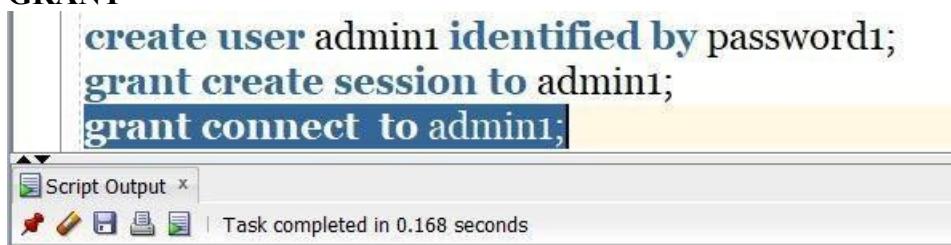
### **GRANT**

```
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;
```

### **GRANT**



A screenshot of the Oracle SQL Developer interface. The main area shows the following SQL code:

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

Below the code, a status bar indicates "Task completed in 0.168 seconds".

Grant succeeded.

### **REVOKE**



A screenshot of the Oracle SQL Developer interface. The main area shows the following SQL code:

```
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;
```

Below the code, a status bar indicates "Task completed in 0.056 seconds".

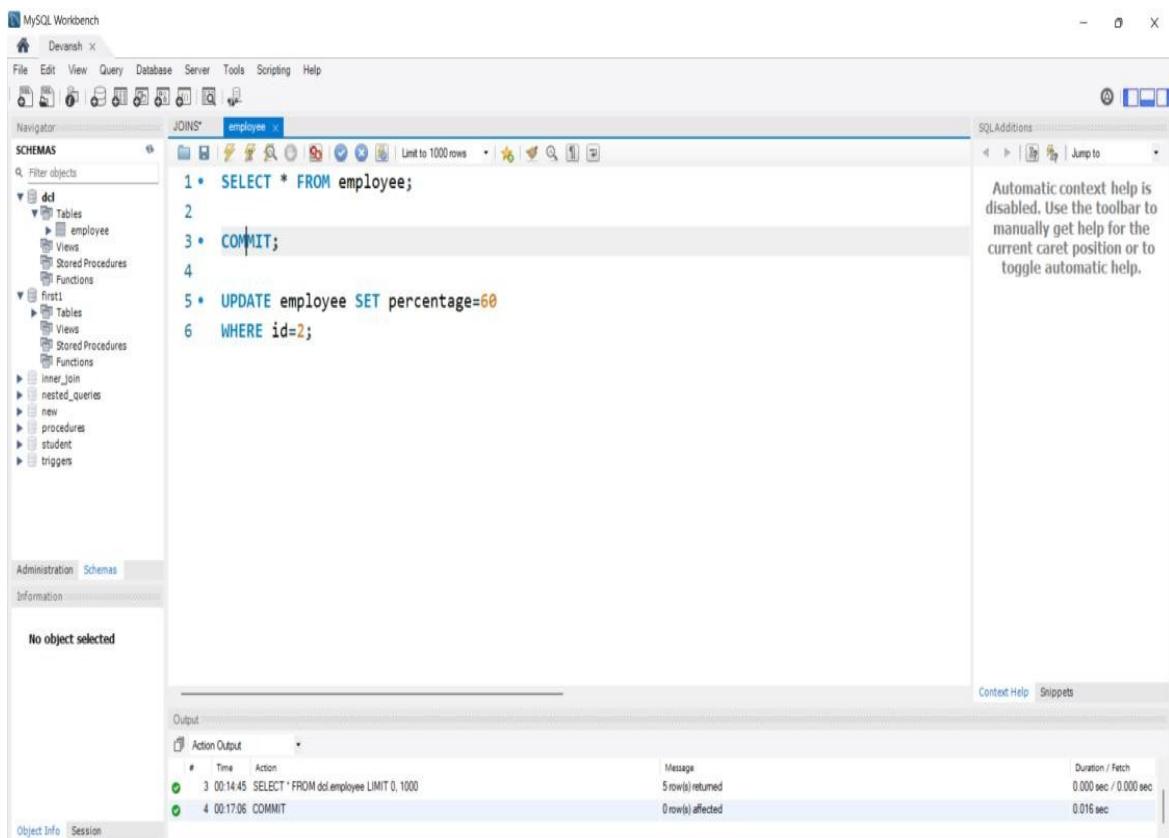
Revoke succeeded.

## 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.

### 1. 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.



The screenshot shows the MySQL Workbench interface. The query editor contains the following SQL code:

```
1 * SELECT * FROM employee;
2
3 * COMMIT;
4
5 * UPDATE employee SET percentage=60
6 WHERE id=2;
```

The Output pane shows the results of the executed statements:

#	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

The screenshot shows the MySQL Workbench interface with the following details:

- Navigator:** Shows the schema `dc1` with a table `employee` selected.
- SQL Editor:** Contains the following SQL code:
 

```

1 • SELECT * FROM employee;
2
3 • COMMIT;
4
5 • UPDATE employee SET percentage=60
6 WHERE id=2;
```
- Result Grid:** Displays the `employee` table with 5 rows:
 

ID	Name	Percentage	Age	Gender	City
1	Ram Kumar	45	21	M	Agra
2	Abhay Kumar	60	22	M	Bhopal
3	Santa Kumari	59	25	F	Delhi
4	Juh Chawla	65	29	F	Mumbai
5	John Abraham	76	32	M	Chennai
- Action Output:** Shows the execution log:
 

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

The screenshot shows the MySQL Workbench interface with the following details:

- Navigator:** Shows the schema `dc1` with a table `employee` selected.
- SQL Editor:** Contains the following SQL code:
 

```

1 • SELECT * FROM employee;
2 • COMMIT;
3 • UPDATE employee SET percentage=78
4 WHERE id=2;
5 • ROLLBACK;
```
- Result Grid:** Displays the `employee` table with 5 rows:
 

ID	Name	Percentage	Age	Gender	City
1	Ram Kumar	45	21	M	Agra
2	Abhay Kumar	78	22	M	Bhopal
3	Santa Kumari	59	25	F	Delhi
4	Juh Chawla	65	29	F	Mumbai
5	John Abraham	76	32	M	Chennai
- Action Output:** Shows the execution log:
 

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

## 2.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.

The screenshot shows the MySQL Workbench interface with a transaction log and a results grid.

**SQL Editor:**

```

1 • SELECT * FROM employee;
2 • COMMIT;
3 • UPDATE employee SET percentage=78
4 WHERE id=2;
5 • ROLLBACK;

```

**Result Grid:**

ID	Name	Percentage	Age	Gender	City
1	Ram Kumar	45	21	M	Agra
2	Akshay Kumar	60	22	M	Bhopal
3	Sarita Kumar	59	25	F	Delhi
4	Juh Chawla	65	29	F	Mumbai
5	John Abraham	76	32	M	Chennai

**Action Output:**

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

The screenshot shows the MySQL Workbench interface with a transaction log and a results grid.

**SQL Editor:**

```

1 • SELECT * FROM employee;
2 • COMMIT;
3 • UPDATE employee SET percentage=78
4 WHERE id=2;
5 • ROLLBACK;

```

**Result Grid:**

ID	Name	Percentage	Age	Gender	City
1	Ram Kumar	45	21	M	Agra
2	Akshay Kumar	60	22	M	Bhopal
3	Sarita Kumar	59	25	F	Delhi
4	Juh Chawla	65	29	F	Mumbai
5	John Abraham	76	32	M	Chennai

**Action Output:**

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

### 3.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;

```

The screenshot shows the MySQL Workbench interface with a script output and a query result.

**Script Output:**

```

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;

```

**Query Result:**

Task completed in 1.3 seconds

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

Script Output x | Query Result x  
✖️ ✎ | Task completed in 0.122 seconds

Savepoint created.

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

Script Output x | Query Result x  
✖️ ✎ | Task completed in 0.507 seconds

Savepoint created.

Exp No:-4

Date:-

## **EXPERIMENT:-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;
```

The screenshot shows the MySQL Workbench interface. The SQL editor tab contains the following query:

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

The results grid displays the following data:

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

The status bar at the bottom right indicates the duration of the query execution.

### **2. LOWER():-**

```
SELECT id, LOWER(name) AS Name ,percentage
```

```
FROM employee;
```

```

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

```

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

Action Output

#	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():-

#### QUERY:-

```

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

```

```

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

```

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

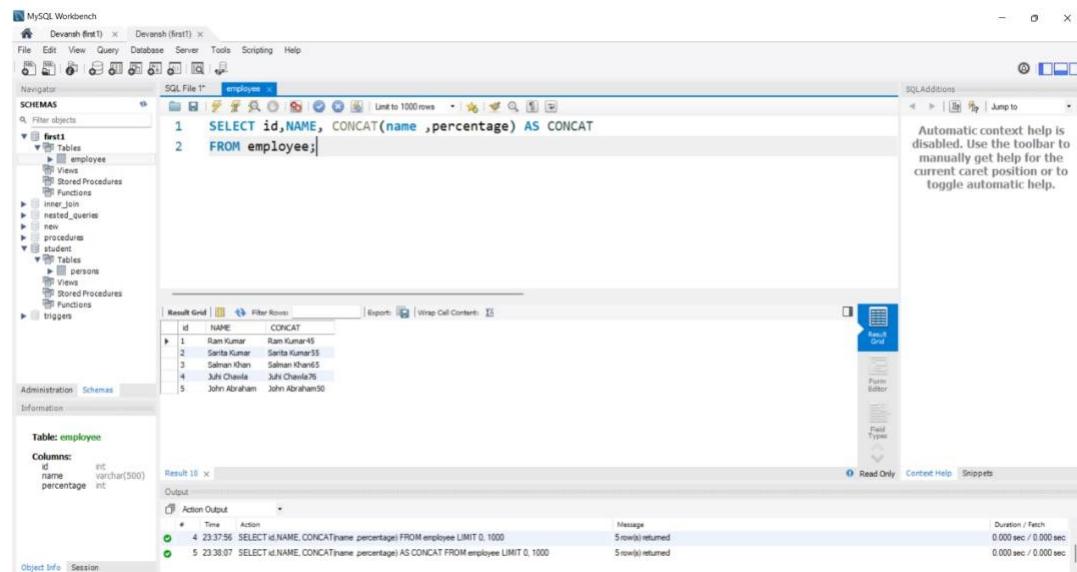
Action Output

#	Time	Action	Message	Duration / Fetch
2	23:32:25	SELECT id,NAME, character_length(name)AS "CHARACTER LENGTH" ,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

#### 4. CONCAT():-

##### QUERY:-

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



The screenshot shows the MySQL Workbench interface with the following details:

- Navigator:** Shows the schema `first1` with tables `employee` and `student`.
- SQL Editor:** Contains the SQL query:

```
1 SELECT id,NAME, CONCAT(name ,percentage) AS CONCAT  
2 FROM employee;
```
- Result Grid:** Displays the results of the query:

	id	NAME	CONCAT
1		Ram Kumar	Ram Kumar45
2		Santa Kumar	Santa Kumar55
3		Salman Khan	Salman Khan5
4		Juli Chawla	Juli Chawla76
5		John Abraham	John Abraham30
- Output:** Shows the execution log:

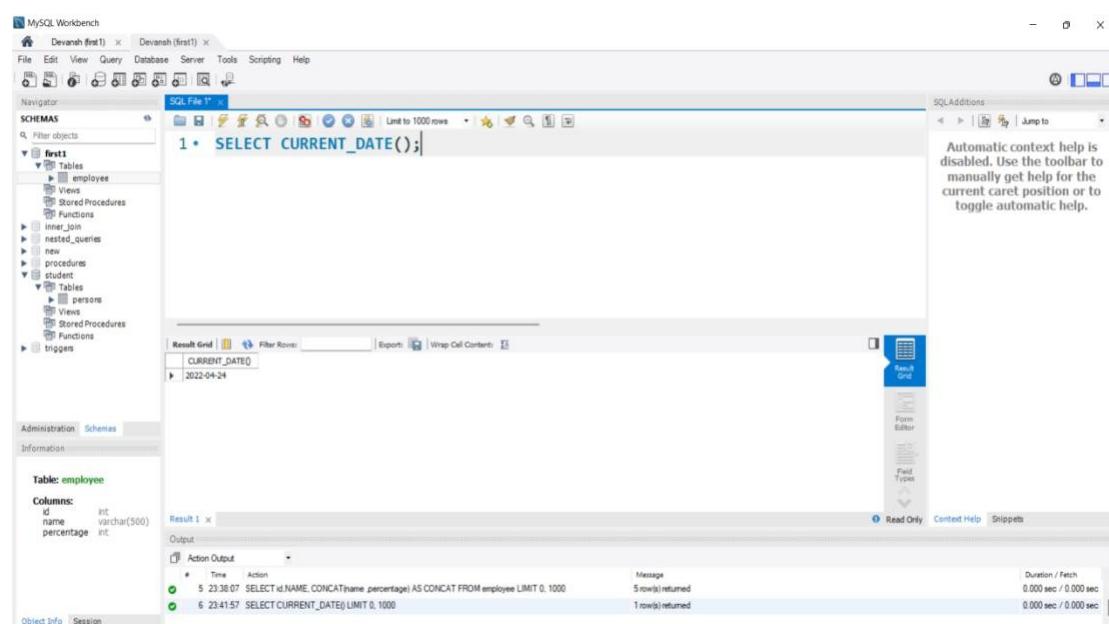
Action	Time	Message	Duration / Fetch
SELECT id,NAME, CONCAT(name ,percentage) FROM employee LIMIT 0, 1000	4 23:37:56	5 row(s) returned	0.000 sec / 0.000 sec
SELECT id,NAME, CONCAT(name ,percentage) AS CONCAT FROM employee LIMIT 0, 1000	4 23:37:07	5 row(s) returned	0.000 sec / 0.000 sec

#### DATE AND TIME FUNCTIONS:-

##### 1.CURRENT\_DATE():-

##### QUERY:-

```
SELECT CURRENT_DATE();
```



The screenshot shows the MySQL Workbench interface with the following details:

- Navigator:** Shows the schema `first1` with tables `employee` and `student`.
- SQL Editor:** Contains the SQL query:

```
1 • SELECT CURRENT_DATE();
```
- Result Grid:** Displays the results of the query:

CURRENT_DATE()
2022-04-24
- Output:** Shows the execution log:

Action	Time	Message	Duration / Fetch
SELECT CURRENT_DATE();	5 23:38:07	1 row(s) returned	0.000 sec / 0.000 sec
SELECT CURRENT_DATE();	6 23:41:57	1 row(s) returned	0.000 sec / 0.000 sec

## 2. NOW():-

### QUERY:-

```
SELECT NOW();
```

The screenshot shows the MySQL Workbench interface with the query `SELECT NOW();` executed. The result grid displays the current timestamp: `2022-04-24 23:44:53`. The output pane shows the message `1 row(s) returned`.

## 3 DATE():-

### QUERY:-

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

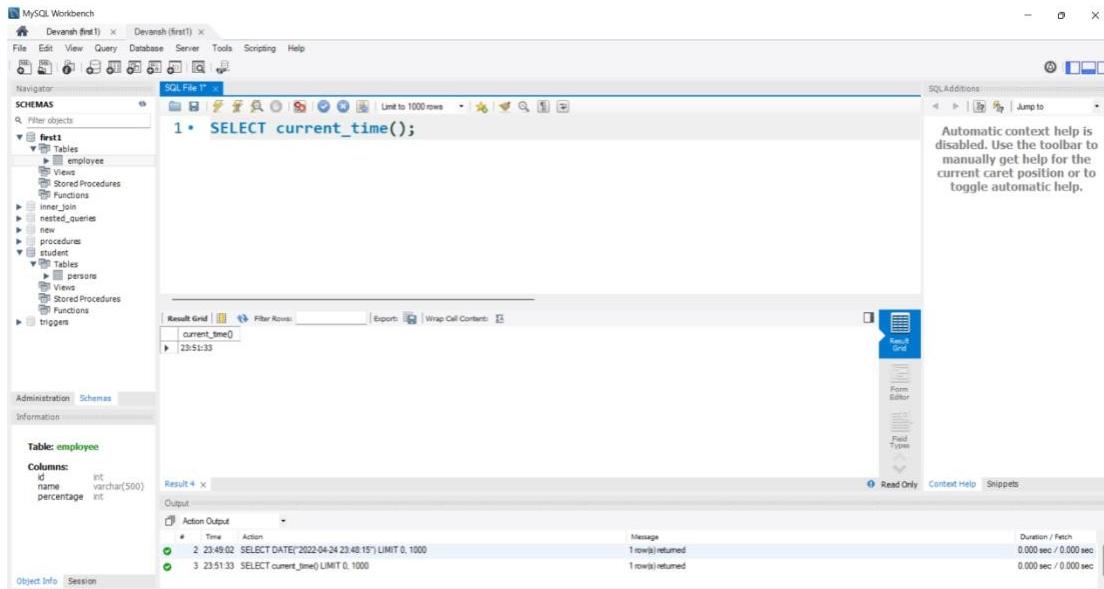
The screenshot shows the MySQL Workbench interface with the query `SELECT DATE("2022-04-24 23:48:15");` executed. The result grid displays the date part of the timestamp: `2022-04-24`. The output pane shows the message `1 row(s) returned`.

## TIME FUNCTIONS:-

### 1.Current\_time():-

#### QUERY:-

```
SELECT current_time();
```

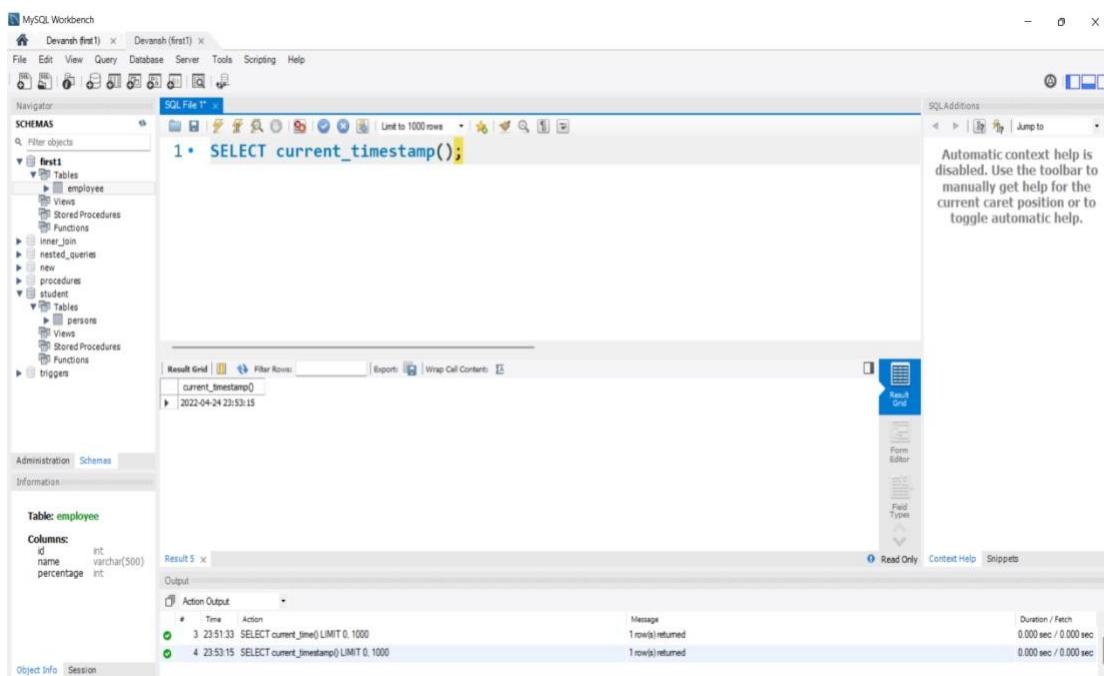


The screenshot shows the MySQL Workbench interface. A query window titled "SQL File 1" contains the command "1 • SELECT current\_time();". The results are displayed in a "Result Grid" table with one row, showing the value "23:51:33". Below the main window, the "Object Info" tab is selected, showing details for the "employee" table.

### 2 Current\_timestamp():-

#### QUERY:-

```
SELECT current_timestamp();
```

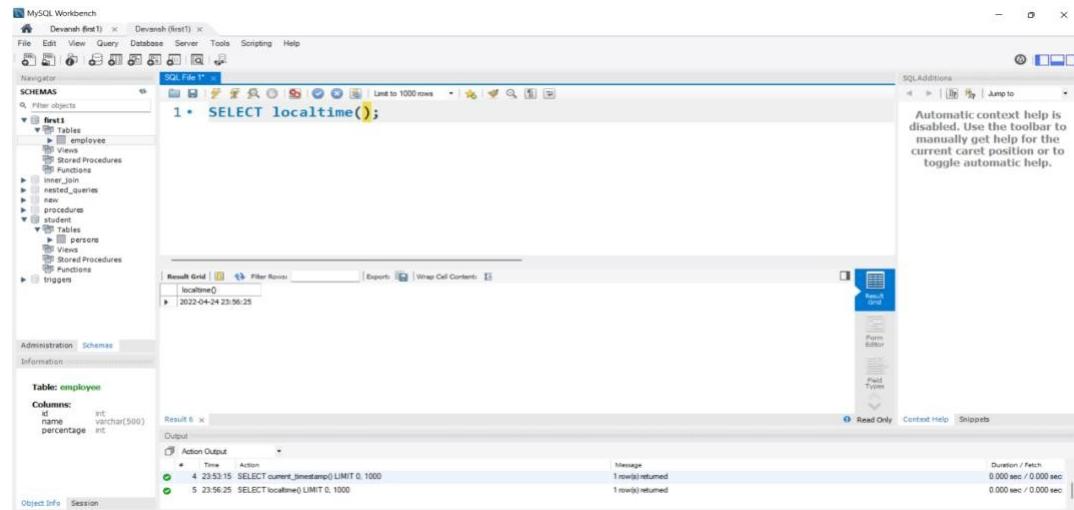


The screenshot shows the MySQL Workbench interface. A query window titled "SQL File 1" contains the command "1 • SELECT current\_timestamp();". The results are displayed in a "Result Grid" table with one row, showing the value "2022-04-24 23:53:15". Below the main window, the "Object Info" tab is selected, showing details for the "employee" table.

### 3. locatime():-

#### QUERY:-

```
SELECT locatime();
```



```
1 • SELECT locatime();
```

Result Grid	Filter Rows	Exports	Wrap Cell Content
localtime	2022-04-24 23:56:25		

#### To Execute Aggregate Functions.

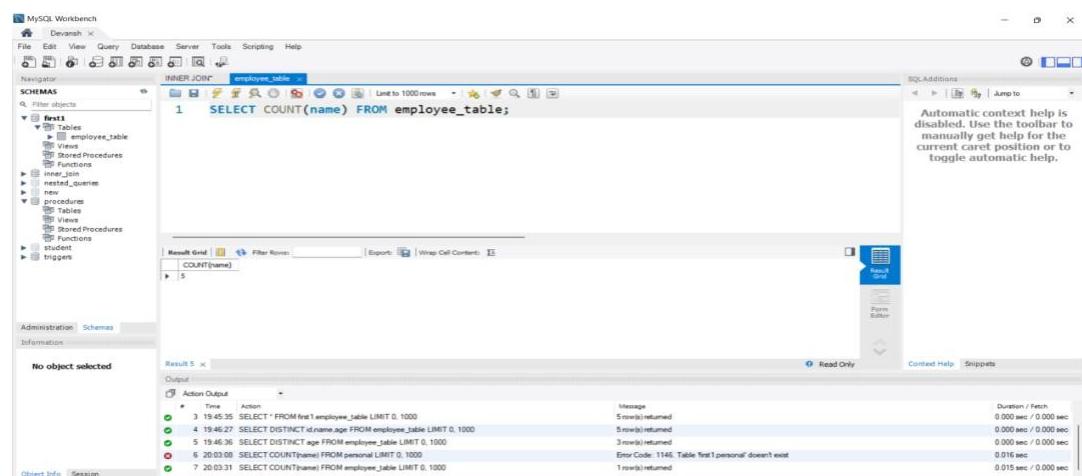
**Theory**:-SQL aggregation function is used to perform the calculations on multiple rows of a single column of a table. It returns a single value. It is also used to summarize the data.

#### 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;
```



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

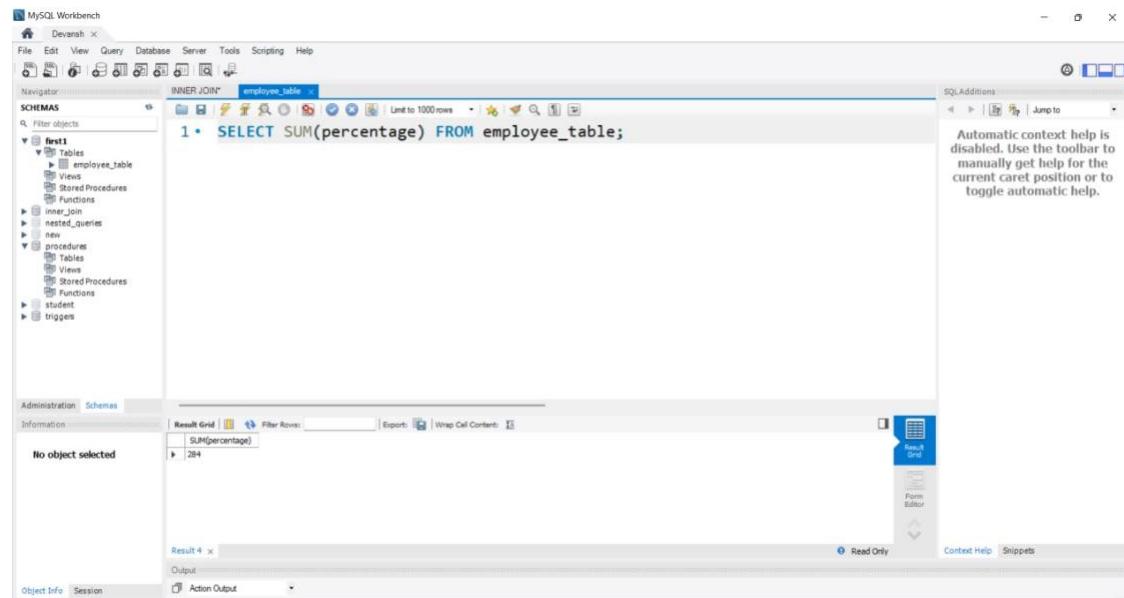
Result Grid	Filter Rows	Exports	Wrap Cell Content
COUNT(name)	5		

## **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;
```



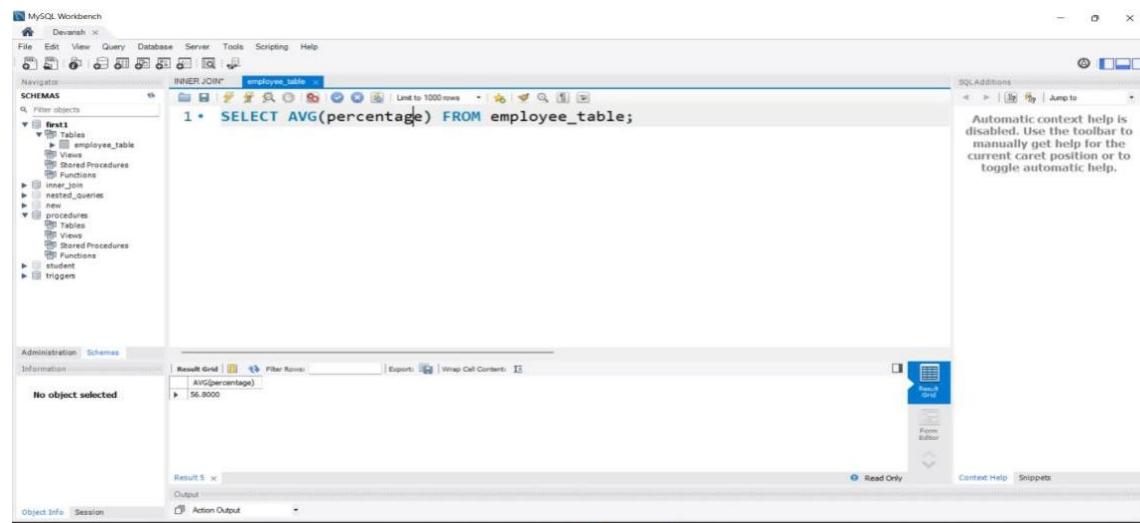
The screenshot shows the MySQL Workbench interface. In the top-left pane, the Navigator displays the schema 'first1' with a single table 'employee\_table'. The main query editor window contains the SQL command: `1 • SELECT SUM(percentage) FROM employee_table;`. The results pane below shows a single row with the value '284' under the column 'SUM(percentage)'. The status bar at the bottom indicates 'Result 4 x' and 'Read Only'.

## **3. AVG function:-**

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

### **QUERY:-**

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



The screenshot shows the MySQL Workbench interface. In the top-left pane, the Navigator displays the schema 'first1' with a single table 'employee\_table'. The main query editor window contains the SQL command: `1 • SELECT AVG(percentage) FROM employee_table;`. The results pane below shows a single row with the value '56.8000' under the column 'AVG(percentage)'. The status bar at the bottom indicates 'Result 5 x' and 'Read Only'.

#### **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.

#### **QUERY:-**

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

The screenshot shows the MySQL Workbench interface. In the top-left, the Navigator pane displays the schema 'test' with a single table 'employee'. The main area contains a query editor with the following SQL statement:

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

The results are shown in two tabs: 'Result Grid' and 'Result 2'. The 'Result Grid' tab shows a single row with the value '95' under the heading 'MAX(percentage)'. The 'Result 2' tab shows the execution log:

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

#### **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;
```

The screenshot shows the MySQL Workbench interface. In the top-left, the Navigator pane displays the schema 'test' with a single table 'employee\_table'. The main area contains a query editor with the following SQL statement:

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

The results are shown in two tabs: 'Result Grid' and 'Result 7'. The 'Result Grid' tab shows a single row with the value '45' under the heading 'MIN(percentage)'. The 'Result 7' tab shows the execution log:

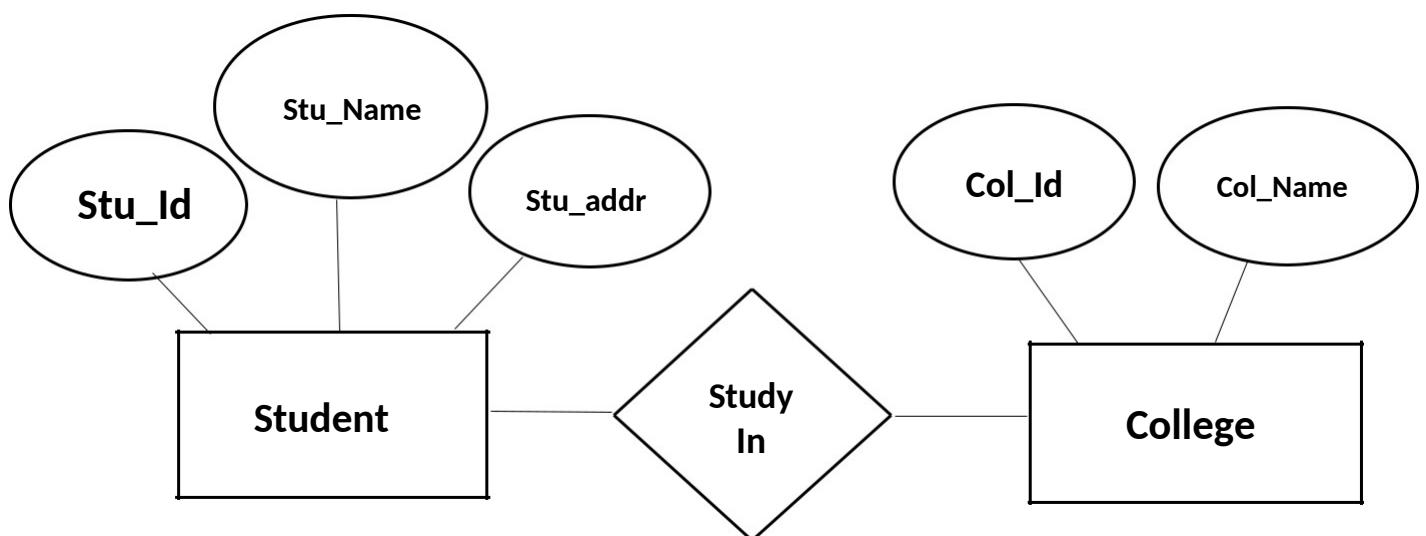
#	Time	Action	Message	Duration / Fetch
1	21:46:07	SELECT * FROM INNER JOIN `first_employee` ON `first_employee`.`id` = `employee_table`.`id` LIMIT 0, 1000	5 rows  returned	0.000 sec / 0.000 sec
2	21:46:33	SELECT MIN(percentage) FROM employee_table LIMIT 0, 1000	1 rows  returned	0.000 sec / 0.000 sec

## **EXPERIMENT:-ER Model**

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

- a) E-R model for Property Management Database Project
- b) E-R model for Water Supply Management System
- c) E-R model for Home renting system database
- d) E-R model for Complaint management system database
- e) E-R model for Employee performance review system database
- f) 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

Date:-

## **EXPERIMENT:-Nested Queries**

**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');
```

The screenshot shows the MySQL Workbench interface with the following details:

- Schemas:** The current schema is "first1".
- Tables:** The "department" and "employee" tables are visible.
- Query Editor:** The query being run is:

```
INNER JOIN department x employee  
1 • SELECT * FROM first1.departments
```
- Result Grid:** The result of the query shows department data:

dept_id	dept_name
1	H-R
2	Finance
3	Accounts
4	Administration
5	Counseling
- Output Panel:** Shows the execution log:

#	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

The screenshot shows the MySQL Workbench interface with the following details:

- Schemas:** The current schema is "first1".
- Tables:** The "department" and "employee" tables are visible.
- Query Editor:** The query being run is:

```
INNER JOIN department x employee  
1 • SELECT * FROM first1.employee;
```
- Result Grid:** The result of the query shows employee data:

emp_id	name	gender	age	salary	dept_id
1	Ali	M	23	24000	3
2	Anup	M	24	25000	4
3	Akhay	M	22	22000	1
4	Akshat	M	21	65000	2
5	Rahul	M	23	22000	4
- Output Panel:** Shows the execution log:

#	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

The screenshot shows the MySQL Workbench interface with a query editor window. The SQL code is:

```

INNER JOIN department employee
32 (5,'Rahul','M',23, 22000,4);
33
34
35 -- THIS IS NESTED QUERY--
36 • SELECT * |from employee
37 WHERE dept_id =(SELECT dept_id FROM department
38 WHERE dept_name='H-R');
39
40
41

```

The result grid shows one row of data:

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

The status bar at the bottom right indicates the duration of the query execution.

## 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);

```

```

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

+----+-----+-----+-----+
| id | name | gender | city  |
+----+-----+-----+-----+
| 2  | Meenal Singh | M    | MP   |
| 3  | Devansh Sharma | M    | Delhi |
| 4  | Rahul Kala | M    | UP   |
| 5  | Akash Jan | M    | UP   |
+----+-----+-----+-----+

```

Action Output

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 INSELECT 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);

```

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

+----+-----+-----+-----+
| id | name | gender | city  |
+----+-----+-----+-----+
| 3  | Devansh Sharma | M    | Delhi |
| 4  | Rahul Kala | M    | UP   |
| 5  | Akash Jan | M    | UP   |
+----+-----+-----+-----+

```

Action Output

Time	Action	Message	Duration / Fetch
3 11:51:37	SELECT * from department where salary>(SELECT AVG(salary) from department) 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);

```

The screenshot shows the MySQL Workbench interface with a query editor window. The query is:

```

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

```

The results grid shows one row:

	id	name	gender	city	salary
1	5	Akshat Jain	M	UP	50000

The status bar at the bottom indicates the following activity:

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

The screenshot shows the MySQL Workbench interface with a query editor window titled "SQL File 1" containing the following code:

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

The "Result Grid" pane shows the output:

COUNT
5

The "Output" pane displays the execution log:

Action Output
8 12:23:21 SELECT count(name) from department where name=(select count(salary)>10000 from department) LIMIT 0, 10... 1 row(s) returned
9 12:23:25 SELECT count(name) AS COUNT from department where name=(select count(salary)>10000 from department)... 1 row(s) returned

Exp No:-7

Date:-

## **EXPERIMENT:-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)

### **1.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;
```

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

t_id	name	qualification	salary
1	Akshay	MCS	12000
2	Amit	MBA	14000
3	Aditya	BCS	13000
4	Akash	BTST	15000
5	Rahul	MPHL	16000

```

MySQL Workbench - Devansh.x

File Edit View Query Database Server Tools Help
INNER JOIN student teacher
31 (7,'Khalid',11,2);
32
33
34 -- JOIN QUERY--
35 * SELECT t.t_id,t.name,t.qualification,s.name,s.class
36 FROM teacher t
37 INNER JOIN student s

```

No object selected

Result Grid | Filter Rows: | Export: | Wrap Cell Content: | Read Only | Form Editor | Field Types |

t_id	name	qualification	name	class
1	Ashay	MCS	Khurram	11
1	Ashay	MCS	Anees	10
2	Amit	MBA	Noman	11
2	Amit	MBA	Furqan	10
4	Akash	BST	Khalid	11
4	Akash	BST	Asghar	12
5	Rahul	MPHL	Asad	12

Action Output

Time	Action	Message	Duration / Fetch
6 21:07:50	SELECT * FROM first1teacher 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=s.t_id	7 row(s) returned	0.000 sec / 0.000 sec

## 2. 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.

### QUERY:-

```

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 Kapoor',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;
```

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

### 3.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 the join condition is fulfilled.

The screenshot shows the MySQL Workbench interface with the following details:

- Schemas:** Shows the 'hrv1' schema with tables 'city' and 'personal'.
- Query Editor:** Displays the query: `1 • SELECT * FROM personal  
2 RIGHT JOIN city  
3 ON personal.city=city.cid;`
- Result Grid:** Shows the results of the query, listing 7 rows from the 'personal' table and 5 rows from the 'city' table. The columns are: id, name, percentage, age, gender, city, cid, and cityname.
- Action Output:** Shows the execution time and message: "7 rows(s) returned".

#### 4.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;
```

The screenshot shows the MySQL Workbench interface with the following details:

- Schemas:** Shows the 'hrv1' schema with tables 'city' and 'personal'.
- Query Editor:** Displays the query: `1 • SELECT * FROM personal  
2 LEFT JOIN city  
3 ON personal.city=city.cid UNION  
4 • SELECT * FROM personal  
5 RIGHT JOIN city  
6 ON personal.city=city.cid;`
- Result Grid:** Shows the results of the query, listing 9 rows from the 'personal' table and 5 rows from the 'city' table. The columns are: id, name, percentage, age, gender, city, cid, and cityname.
- Action Output:** Shows the execution time and message: "7 rows(s) returned" and "9 rows(s) returned".

#### 5.SQL CROSS JOIN Keyword

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

## QUERY:-

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

The screenshot shows the MySQL Workbench interface with the following details:

- Navigator:** Shows the database schema with tables `student` and `city` under the `first1` schema.
- SQL Editor:** Displays the query:

```
1  SELECT * FROM student  
2  CROSS JOIN City;  
3
```
- Result Grid:** Shows the result of the query, displaying 12 rows of data from the cross join of `student` and `city`.

ID	Name	Age	City ID	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	Santa Kumari	21	3	Delhi
4	Santa Kumari	21	2	Bhopal
4	Santa Kumari	21	1	Agra
- Output:** Shows the execution log with two entries:

#	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

Exp No:-8

Date:-

## **EXPERIMENT:-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:-**

- a) Union
- b) Union all
- c) Intersect
- d) 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);
```

#### **a) Union:-**

##### **Query:-**

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

The screenshot shows the MySQL Workbench interface with the following details:

- Schemas:** The schema is set to 'student'.
- Queries:** The query entered is:
 

```
1 SELECT * from student1
2 UNION
3 SELECT * from students
```
- Result Grid:** The result grid displays the following data:
 

	id	name	age
1	1	Devarsh Sharma	21
2	2	Rahul Kella	26
3	3	Akash Jain	34
4	2	Santa Kumari	26
5	3	Rohan Dubey	34
- Action Output:** The output shows two rows of actions:
 

Time	Action	Message	Duration / Fetch
01:59:45	INSERT INTO student(id,name,age) VALUES (1,'Devarsh Sharma',21), (2,'Santa Kumari',26), (3,'Rohan Dubey',34)	3 row(s) affected Records: 3 Duplicates: 0 Warnings: 0	0.000 sec
01:57:01	SELECT * from student1 UNION SELECT * from students	5 row(s) returned	0.000 sec / 0.000 sec

### b) Union All:-

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

The screenshot shows the MySQL Workbench interface with the following details:

- Schemas:** The schema is set to 'student'.
- Queries:** The query entered is:
 

```
1 SELECT *from student1
2 UNION ALL
3 SELECT * from students
```
- Result Grid:** The result grid displays the following data:
 

	id	name	age
1	1	Devarsh Sharma	21
2	2	Rahul Kella	26
3	3	Akash Jain	34
4	1	Devarsh Sharma	21
5	2	Santa Kumari	26
6	3	Rohan Dubey	34
- Action Output:** The output shows two rows of actions:
 

Time	Action	Message	Duration / Fetch
01:57:01	SELECT *from student1 UNION SELECT * from students	5 row(s) returned	0.000 sec / 0.000 sec
02:03:58	SELECT *from student1 UNION ALL SELECT * from students	6 row(s) returned	0.000 sec / 0.000 sec

### c) 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);

```

MySQL Workbench screenshot showing the execution of a query. The 'Query 1' tab contains the SQL code: 'SELECT DISTINCT Id FROM tab1 INNER JOIN tab2 USING (Id);'. The 'Result Grid' shows the output: 'id' with values 3 and 4. The 'Result 2' tab shows the execution log with two entries: '8 22:33:17 SELECT \* FROM first.tab2 LIMIT 0, 1000' and '9 22:33:40 SELECT DISTINCT Id FROM tab1 INNER JOIN tab2 USING (id) LIMIT 0, 1000'. The log indicates 4 rows returned for the first query and 2 rows returned for the second query.

### **d) 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;

```

The screenshot shows the MySQL Workbench interface. In the top-left, the Navigator pane displays the schema 'first' with tables 't1' and 't2'. The main area contains a SQL editor window with the following query:

```

1 • SELECT
2     id
3   FROM
4     t1
5 LEFT JOIN
6     t2 USING (id)
7 WHERE
8     t2.id IS NULL;

```

The results grid below shows a single row with the value '1' in the 'id' column.

At the bottom, the Output pane shows the execution log:

Action	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
```

The screenshot shows the MySQL Workbench interface. In the left sidebar, under the 'Schemas' section for 'first1', there is a 'Views' folder containing a single view named 'student\_data'. The main query editor window displays the SQL code for creating this view:

```
1 • CREATE VIEW student_data
2 AS
3   SELECT s.id,s.name,c.city
4   FROM student s
5   INNER JOIN City c
6   ON s.id= c.cid;
7
8
9 • SELECT * FROM student_data
```

Below the code, the 'Result Grid' shows the data from the 'student\_data' view:

ID	Name	City
1	Ram Kumar	Agra
2	Salman Khan	Bhopal
3	Meera Khan	Delhi

The 'Output' pane at the bottom shows the execution log:

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

The screenshot shows the MySQL Workbench interface. In the left sidebar, under the 'Schemas' section for 'first1', there is a 'Views' folder containing a view named 'new\_data'. The main query editor window displays the SQL code for renaming the view:

```
1 • RENAME TABLE student_data
2
3 TO new_data;
4
5 • SELECT * FROM new_data
```

Below the code, the 'Result Grid' shows the data from the 'new\_data' view:

ID	Name	City
1	Ram Kumar	Agra
2	Salman Khan	Bhopal
3	Meera Khan	Delhi

The 'Output' pane at the bottom shows the execution log:

Action	Time	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:-

### **QUERY:-**

DROP VIEW new\_data;

The screenshot shows the MySQL Workbench interface with the following details:

- File Bar:** File, Edit, View, Query, Database, Server, Tools, Scripting, Help.
- Navigator:** Shows the schema structure under 'first1'. It includes 'Tables' (city, student), 'Views', 'Stored Procedures', 'Functions', 'inter\_join', 'nested\_queries', 'new', 'procedures', and 'student'.
- SQL Editor:** The query '1 • DROP VIEW new\_data;' is entered.
- Output Window:** Shows the execution log:

#	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

### QUERY:-

ALTER VIEW data

AS

SELECT \* FROM student\_table

INNER JOIN City

ON student\_table.city=City.cid

WHERE age>22;

SELECT \*from data;

The screenshot shows the MySQL Workbench interface with a query editor window. The query is:

```
1 ALTER VIEW data
2 AS
3 SELECT * FROM student_table
4 INNER JOIN City
5 ON student_table.city=City.cid
6 |
7 WHERE age>22;
8
9
10 *  SELECT *from data;
```

The results grid displays the following data:

ID	Name	Age	City	CID	CityName
1	Salman Khan	24	1	1	Agra
2	Meera Khan	35	3	3	Bhopal
3	Sarita Kumari	38	2	2	Delhi

The status bar at the bottom shows the output of the last two statements:

- 7 00:14:53 ALTER VIEW data AS SELECT \* FROM student\_table INNER JOIN City ON student\_table.city=City.cid WHERE age>22; 0 row(s) affected Duration / Fetch 0.000 sec / 0.000 sec
- 8 00:14:57 SELECT \*from data LIMIT 0, 1000 3 row(s) returned Duration / Fetch 0.000 sec / 0.000 sec

Exp No:-9

Date:-

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

##### **FOR EXECUTION**

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

MySQL Workbench

File Edit View Query Database Server Tools Scripting Help

Navigator: deva24

Query 1:

```

1  DELIMITER $$

2  CREATE PROCEDURE GetCustomerLevel(
3      IN pCustomerNumber INT,
4      OUT pCustomerLevel VARCHAR(20))
5  BEGIN
6      DECLARE credit DECIMAL(10,2) DEFAULT 0;
7
8      SELECT creditLimit
9          INTO credit
10         FROM customers
11        WHERE customerNumber = pCustomerNumber;
12
13      IF credit > 50000 THEN
14          SET pCustomerLevel = 'PLATINUM';
15      END IF;
16
17  END$$

18
19  DELIMITER ;

```

Output:

Action	Time	Action	Message	Duration / Fetch
CREATE PROCEDURE	1 22:54:17	CREATE PROCEDURE GetCustomerLevel( IN pCustomerNumber INT, OUT pCustomerLevel VARCHAR(20))	0 rows(s) affected	0.000 sec

Contact Help Snippets

MySQL Workbench

File Edit View Query Database Server Tools Scripting Help

Navigator: deva24

Query 1:

```

1 *  SELECT
2     customerNumber,
3     creditLimit
4   FROM
5     customers
6   WHERE
7     creditLimit > 50000
8   ORDER BY
9     creditLimit DESC;

```

Output:

	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

**b)Iterative Statement**

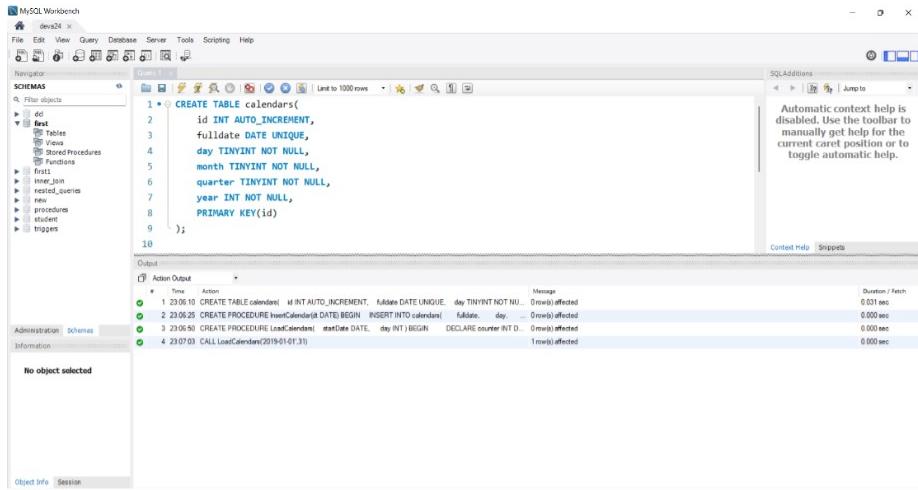
```
CREATE TABLE calendars(
    id INT AUTO_INCREMENT,
    fulldate DATE UNIQUE,
    day TINYINT NOT NULL,
    month TINYINT NOT NULL,
    quarter TINYINT NOT NULL,
    year INT NOT NULL,
    PRIMARY KEY(id)
);
```

**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);

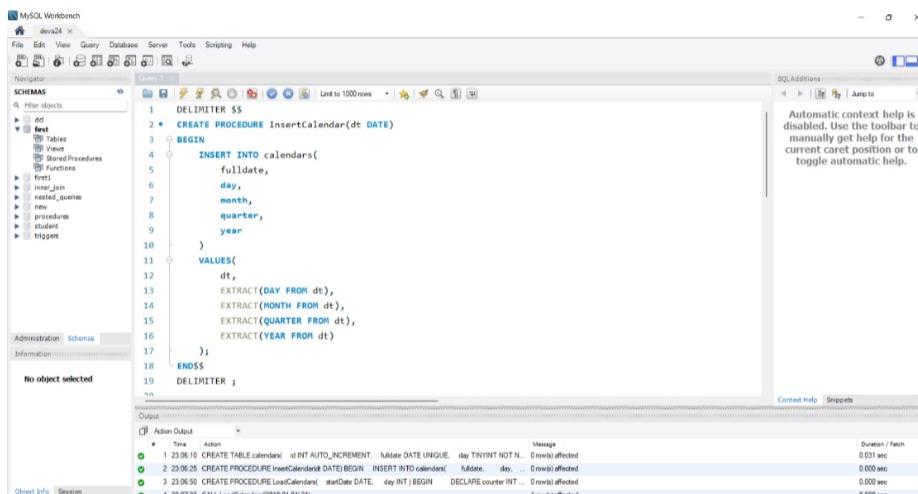


```

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)
);
    
```

Action Output

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



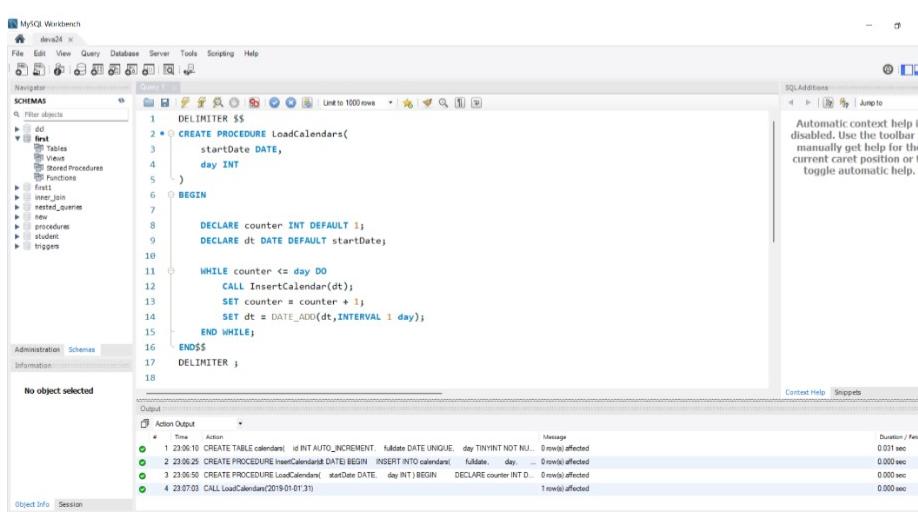
```

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 ;
    
```

Action Output

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



```

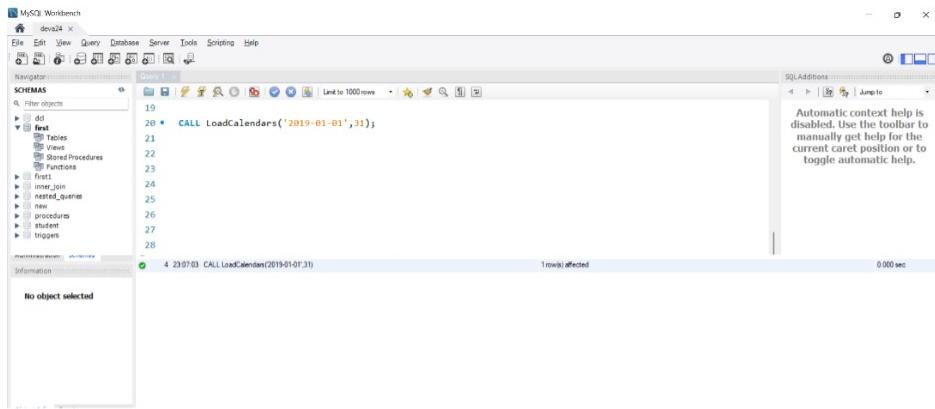
DELIMITER $$

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 ;
    
```

Action Output

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

Exp No:-10

Date:-

## **EXPERIMENT:-Procedures**

**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 be 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();
```

MySQL Workbench

Devarsh X

File Edit View Query Database Server Tools Scripting Help

**JOINS** × film

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

```

9 • CREATE PROCEDURE sp_GetMovies()
10 BEGIN
11     select rating,name,release_date from film;
12 END //
13
14 DELIMITER ;
15 • CALL sp_GetMovies();
16
17

```

Result Grid | Filter Rows: Export: Wrap Cell Content: |

Table: film

Columns:

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

Result 9 × | Read Only | Context Help | Skip to |

Action Output |

#	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

Object Info Session

Exp No:-11

Date:-

## **EXPERIMENT:-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','Tylor','2000-02-09'),  
(4,'Susan','Hawthorne','2002-04-24');
```

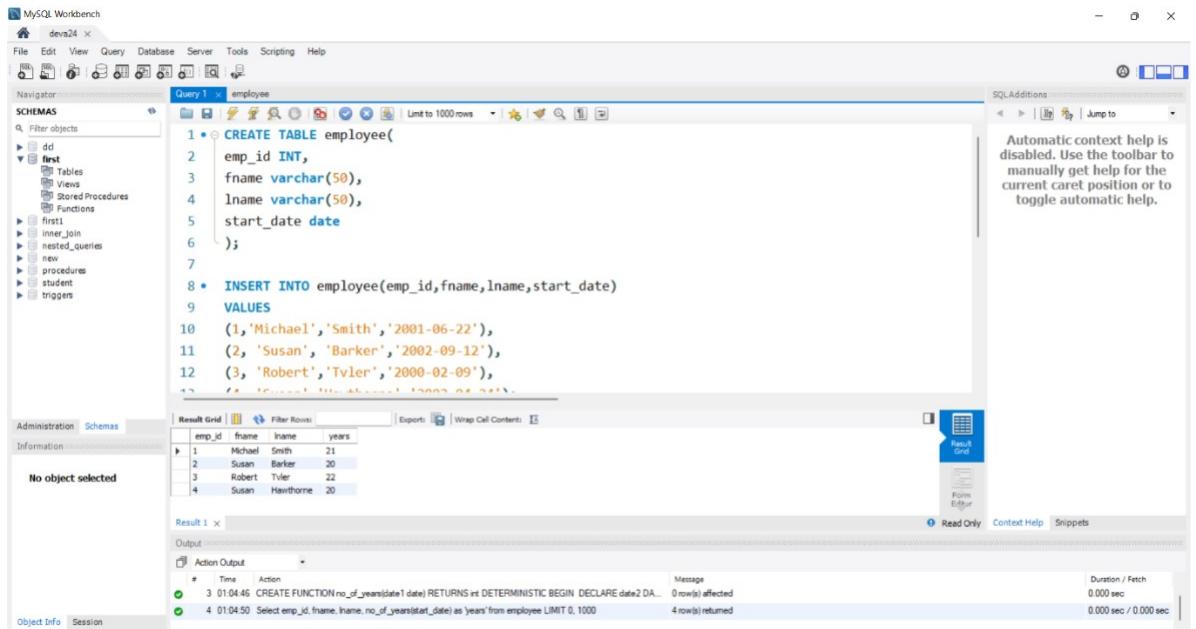
#### **CREATING FUNCTION**

```
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:-



The screenshot shows the MySQL Workbench interface with the database 'deva24' selected. In the Navigator pane, under the 'Tables' section of the 'first' schema, there is a new table named 'employee'. The 'Query 1' tab contains the following SQL code:

```

1 • CREATE TABLE employee(
2     emp_id INT,
3     fname varchar(50),
4     lname varchar(50),
5     start_date date
6 );
7
8 • INSERT INTO employee(emp_id,fname,lname,start_date)
9     VALUES
10    (1,'Michael','Smith','2001-06-22'),
11    (2,'Susan','Barker','2002-09-12'),
12    (3,'Robert','Tylor','2000-02-09'),
13    (4,'Susan','Hawthorne','2000-06-22');

```

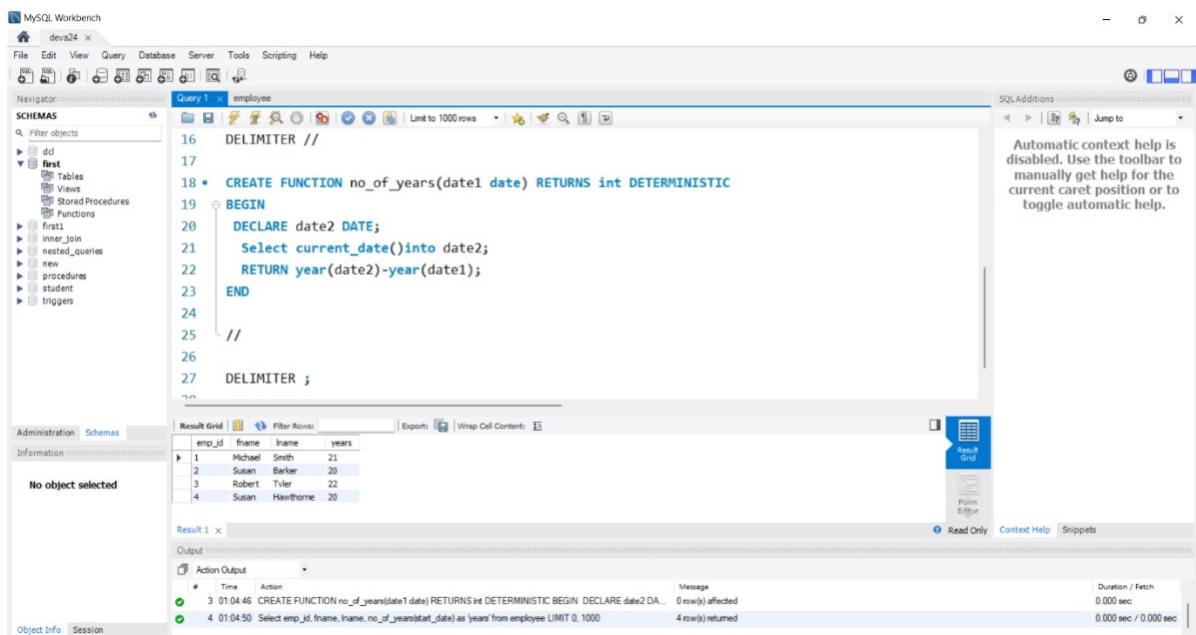
The 'Result Grid' pane shows the data inserted into the 'employee' table:

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

The 'Output' pane displays the execution log:

- Action Output
 

#	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



The screenshot shows the MySQL Workbench interface with the database 'deva24' selected. In the Navigator pane, under the 'Functions' section of the 'first' schema, there is a new function named 'no\_of\_years'. The 'Query 1' tab contains the following SQL code:

```

16 DELIMITER //
17
18 • CREATE FUNCTION no_of_years(date1 date) RETURNS int DETERMINISTIC
19 BEGIN
20     DECLARE date2 DATE;
21     Select current_date()into date2;
22     RETURN year(date2)-year(date1);
23 END
24
25 //
26
27 DELIMITER ;

```

The 'Result Grid' pane shows the data inserted into the 'employee' table:

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

The 'Output' pane displays the execution log:

- Action Output
 

#	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

Exp No:-12

Date:-

## **EXPERIMENT:-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

```

The screenshot shows the MySQL Workbench interface with the following details:

- File Menu:** File, Edit, View, Query, Database, Server, Tools, Scripting, Help.
- Toolbar:** Standard MySQL icons for opening, saving, and executing queries.
- Navigator:** Shows the schema structure of the database, including the `first1` schema which contains tables like `emp` and `emp_audit`.
- SQL Editor:** Contains two queries:
  - Line 51: `CALL firstCurs();`
  - Line 52: `SELECT * FROM Vbackupdata;`
- Result Grid:** Displays the output of the second query, showing 20 rows of data from the `Vbackupdata` table. The columns are `C_ID`, `c_name`, and `c_address`. The data includes entries such as:
 

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	Sirwan	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	Sirwan	137, Vatsa Colony
7	Jashmin	138, Vatsa Colony
8	Anamika	139, Vatsa Colony
9	Radhika	140, Vatsa Colony
- Output Window:** Shows the execution log with two entries:
 

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

Exp No:-13

Date:-

## **EXPERIMENT:-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

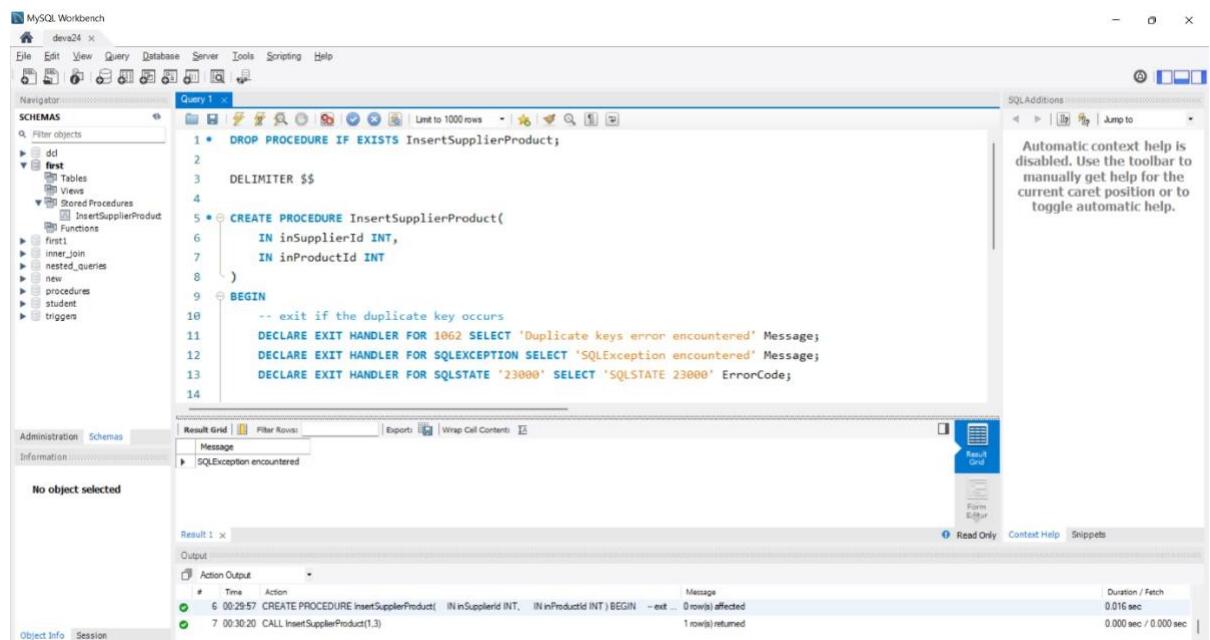
WHERE supplierId = inSupplierId;

END$$

DELIMITER ;

```

## OUTPUT



The screenshot shows the MySQL Workbench interface with the following details:

- Schemas:** The current schema is 'first'. The 'Stored Procedures' section contains a single procedure named 'InsertSupplierProduct'.
- Query Editor:** The SQL code for the procedure is displayed:
 

```

1 • DROP PROCEDURE IF EXISTS InsertSupplierProduct;
2
3 DELIMITER $$*
4
5 • CREATE PROCEDURE InsertSupplierProduct(
6     IN inSupplierId INT,
7     IN inProductId INT
8 )
9 BEGIN
10    -- exit if the duplicate key occurs
11    DECLARE EXIT HANDLER FOR 1062 SELECT 'Duplicate keys error occurred' Message;
12    DECLARE EXIT HANDLER FOR SQLEXCEPTION SELECT 'SQLException encountered' Message;
13    DECLARE EXIT HANDLER FOR SQLSTATE '23000' SELECT 'SQLSTATE 23000' ErrorCode;
14
      
```
- Result Grid:** The 'Message' column shows the error message: "SQLException encountered".
- Output:** The log shows two entries:
 

#	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)	0.000 sec / 0.000 sec	

Exp No:-14

Date:-

## **EXPERIMENT:-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);
```

MySQL Workbench

Devarsh X

File Edit View Query Database Server Tools Scripting Help

Navigator JOINs emp emp\_audit

SCHEMAS first1 Tables emp emp\_audit Views Stored Procedures Functions inner\_join nested\_queries new procedures Tables Views Stored Procedures Functions student triggers

SQLAdditions

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

Result Grid | Filter Rows | Edits | Export/Imports | Wrap Cell Content | Result Grid | Form Editor | Context Help | Snippets

Table: emp\_audit

Columns:

- id** int AI PK
- audit\_description** varchar(50)

Object Info Session

Action Output

#	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

Devarsh X

File Edit View Query Database Server Tools Scripting Help

Navigator JOINs emp emp\_audit

SCHEMAS first1 Tables emp emp\_audit Views Stored Procedures Functions inner\_join nested\_queries new procedures Tables Views Stored Procedures Functions student triggers

SQLAdditions

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

Result Grid | Filter Rows | Edits | Export/Imports | Wrap Cell Content | Result Grid | Form Editor | Context Help | Snippets

Table: emp\_audit

Columns:

- id** int AI PK
- audit\_description** varchar(50)

Object Info Session

Action Output

#	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

Exp No:-15

Date:-

## **Experiment:-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 much 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**

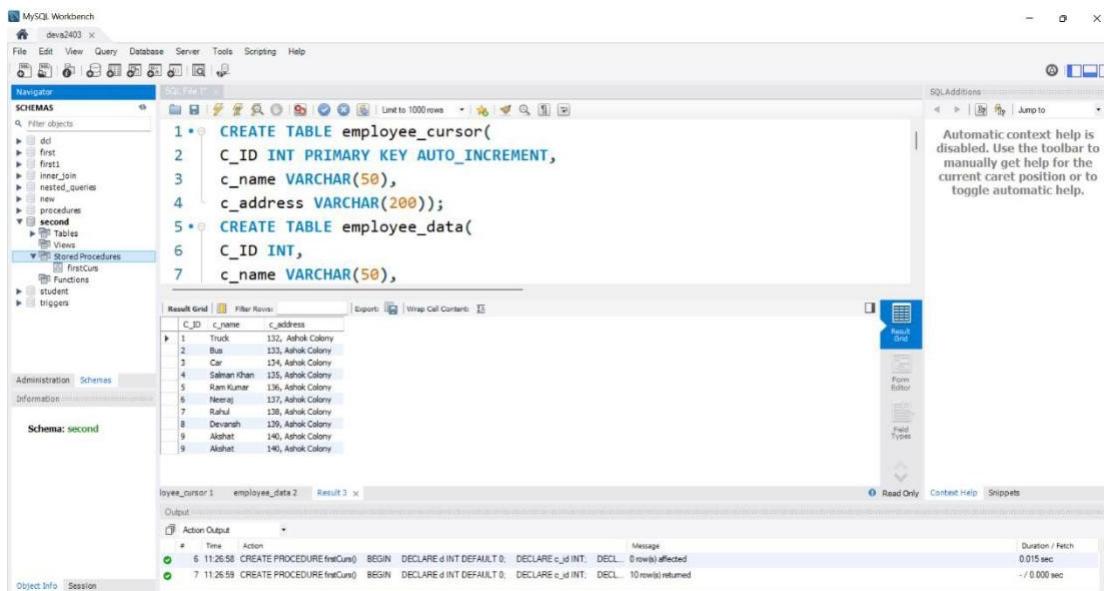
- Tenant
- Owner
- Contract
- Payment

## **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 specifications 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**



The screenshot shows the MySQL Workbench interface with the following details:

- Schemas:** The current schema is "second".
- Tables:** Two tables are shown: "employee\_cursor" and "employee\_data".
- Script Editor:** The SQL code for creating the tables is displayed:

```
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),
```
- Result Grid:** A table titled "Result Grid" displays the data from the "employee\_cursor" table:

C_ID	c_name	c_address
1	Truck	132, Ashok Colony
2	Bus	133, Ashok Colony
3	Car	124, Ashok Colony
4	Kalben Khan	132, Ashok Colony
5	Ram Kumar	136, Ashok Colony
6	Neeraj	137, Ashok Colony
7	Rahul	130, Ashok Colony
8	Devansh	139, Ashok Colony
9	Akshat	140, Ashok Colony
9	Akshat	140, Ashok Colony
- Output:** The "Action Output" section shows the execution log for the queries:

#	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

MySQL Workbench

File Edit View Query Database Server Tools Scripting Help

Navigator: Schemas

Schema: second

SQL File 1: x

```

38 IF NOT d = 1 THEN
39   FETCH Get_cur INTO c_id, c_name, c_address;
40   INSERT INTO employee_data VALUES(c_id, c_name, c_address);
41 END IF;
42 END LOOP;
43 CLOSE Get_cur;
44 END;
45 CALL firstCurs();
46 SELECT * FROM employee_data;

```

Result Grid | Filter Rows: | Export: | Wrap Cell Content: |

c_id	c_name	c_address
1	Truck	132, Ashok Colony
2	Bus	133, Ashok Colony
3	Car	134, Ashok Colony
4	Sukhen Kumar	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	Akash	140, Ashok Colony
9	Akash	140, Ashok Colony

employee\_cursor1 employee\_data 2 Result 3 x

Action Output

#	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 (invCur)	BEGIN DECLARE d INT DEFAULT 0; DECLARE c_id INT; DECL 10 row(s) returned	/ 0.000 sec

SQLAdditions | < > | Jump to

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

## b)Exception Handling

MySQL Workbench

File Edit View Query Database Server Tools Scripting Help

Navigator: Schemas

No object selected

Query 1: x

```

1 • DROP PROCEDURE IF EXISTS InsertSupplierProduct;
2
3 DELIMITER $$

5 • CREATE PROCEDURE InsertSupplierProduct(
6   IN inSupplierId INT,
7   IN inProductId INT
8 )
9 BEGIN
10   -- exit if the duplicate key occurs
11   DECLARE EXIT HANDLER FOR 1062 SELECT 'Duplicate keys error encountered' Message;
12   DECLARE EXIT HANDLER FOR SQLEXCEPTION SELECT 'SQLException encountered' Message;
13   DECLARE EXIT HANDLER FOR SQLSTATE '23000' SELECT 'SQLSTATE 23000' ErrorCode;
14

```

Result Grid | Filter Rows: | Export: | Wrap Cell Content: |

Message

SQLException encountered

Result 1: x

Action Output

#	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