



NCR CAMPUS, MODINAGAR

(A Constituent of SRM University, Chennai T.N.)

Delhi-Meerut Road, Sikari Kalan, Modinagar – 201204, GHAZIABAD (U.P.)

Database Management System Lab
(18CSC303J)

Lab Record
(Jan-May 2023)

Name of Student :
Reg. No. :
Degree/Branch : B. Tech / CSE
Semester/Section : 6th Sem, (CSE-J)
Course Code : 18CSC303J
Course Title : Database Management System
Faculty In charge. : Ms. Anjali Malik

SRM IST, DELHI-NCR CAMPUS, MODINAGAR

Department Of Computer Science and Engineering

REGISTRATION NO.

R	A	2	0	1	1	0	0	3	0	3	0			
---	---	---	---	---	---	---	---	---	---	---	---	--	--	--

BONAFIDE CERTIFICATE

It is to be certified that the bonfide practical record submitted by _____ of 6th semester for Bachelor of Technology degree in the Department of Computer Science and Engineering, Delhi-NCR Campus, SRM IST has been done for the course **Database Management System Lab (18CSC303J)** during the academic semester session January 2023 – May 2023.

Dr. R. P. Mahapatra

Head of the Department
Computer Science & Engg.

Ms. Anjali Malik

Assistant Professor
Computer Science & Engg.

Submitted for the University Examination held on _____

Examiner 1

Examiner 2

INDEX

Exp. No.	Name of Experiment	Page No.	Date	Signature
01.				
02.				
03.				
04.				
05.				
06.				
07.				
08.				
09.				
10.				
11.				

12.				
13.				
14.				
15.				

EXPERIMENT 1

AIM-SQL data definition: - write at least four queries for each

- Create
- Alter
- Modify
- Drop
- Rename

CREATE

1. create table student(Stu_name varchar(20),S_id int,d_o_b, varchar(10));
2. create table subject(Sub_name varchar(10),Sub_id int);
3. create table class_level(counsellor varchar(10), level_num int);
4. create table class_rep(CR_name varchar(10), section char);

ALTER

1. create table student(Stu_name varchar(20),S_id int,d_o_b varchar(10));
alter table student drop column d_o_b;
2. create table subject(Sub_name varchar(10), Sub_id int);
alter table subject drop column Sub_id;
3. create table class_level(counsellor varchar(10), level_num int);
alter table class level drop column level_num;
4. create table class_rep(CR_name varchar(10), section char);
alter table class_rep drop column section;

DROP

1. create table student(Stu_name varchar(20),S_id int,d_o_b varchar(10));
drop table student;
2. create table subject(Sub_name varchar(10),Sub_id int);
drop table subject;
3. create table class_level(counsellor varchar(10), level_num int);
drop table class_level;
4. create table class_rep(CR_name varchar(10),section char);
drop table class_rep;

RENAME

1. create table student(Stu_name varchar(20),S_id int,d_o_b varchar(10));
rename table student to student_details;
2. create table subject(Sub_name varchar(10), Sub_id int);
rename table student to student_details;
3. create table class_level(counsellor varchar(10), level_num int);
rename class_level to class_details;
4. create table class_rep(CR_name varchar(10), section char);
rename class_rep to CR details.

OUTPUT

CREATE

The screenshot shows a SQL interface with two main sections: Input and Output. In the Input section, four CREATE TABLE statements are entered:

```
CREATE TABLE student(Stu_name varchar(20),S_id int,d_o_b varchar(10));
CREATE TABLE subject(Sub_name varchar(10),Sub_id int);
CREATE TABLE class_level(counsellor varchar(10),level_num int);
CREATE TABLE class_rep(CR_name varchar(10),section char);
```

In the Output section, it says "SQL query successfully executed. However, the result set is empty." To the right, under "Available Tables", there are four tables listed with their columns and empty data rows:

- Class_level**: counsellor, level_num
empty
- Class_rep**: CR_name, section
empty
- Student**: Stu_name, S_id, d_o_b
empty
- Subject**: Sub_name, Sub_id
empty

ALTER

The screenshot shows a SQL interface with two main sections: Input and Output. In the Input section, several ALTER TABLE statements are entered, primarily involving dropping columns from the previously created tables:

```
CREATE TABLE student(Stu_name varchar(20),S_id int,d_o_b varchar(10));
ALTER TABLE student DROP COLUMN d_o_b;
CREATE TABLE subject(sub_name varchar(10),sub_id int);
ALTER TABLE subject DROP COLUMN Sub_id;
CREATE TABLE class_level(counsellor varchar(10),level_num int);
ALTER TABLE class_level DROP COLUMN level_num;
CREATE TABLE class_rep(CR_name varchar(10),section char);
ALTER TABLE class_rep DROP COLUMN section;
```

In the Output section, it says "SQL query successfully executed. However, the result set is empty." To the right, under "Available Tables", the same four tables are listed with their columns and empty data rows, identical to the previous screenshot:

- Class_level**: counsellor, level_num
empty
- Class_rep**: CR_name, section
empty
- Student**: Stu_name, S_id, d_o_b
empty
- Subject**: Sub_name, Sub_id
empty

DROP

Input

```
CREATE TABLE student(Stu_name varchar(20),S_id int,d_o_b varchar(10));  
ALTER TABLE student DROP COLUMN d_o_b;  
CREATE TABLE subject(Sub_name varchar(10),Sub_id int);  
ALTER TABLE subject DROP COLUMN Sub_id;  
CREATE TABLE class_level(counsellor varchar(10),level_num int);  
ALTER TABLE class_level DROP COLUMN level_num;  
CREATE TABLE class_rep(CR_name varchar(10),section char);  
ALTER TABLE class_rep DROP COLUMN section;
```

Output

SQL query successfully executed. However, the result set is empty.

Available Tables

Class_level
counsellor
empty

Class_rep
CR_name
empty

Student
Stu_name
S_id
empty

Subject
Sub_name
empty

RENAME

Input

```
CREATE TABLE Student(Stu_name varchar(20),S_id int,d_o_b varchar(10));  
ALTER TABLE Student RENAME TO Student_details;  
CREATE TABLE Subject(Sub_name varchar(10),Sub_id int);  
ALTER TABLE Subject RENAME TO Subject_details;  
CREATE TABLE class_level(counsellor varchar(10),level_num int);  
ALTER TABLE Class_level RENAME TO Class_details;  
CREATE TABLE class_rep(CR_name varchar(10),section char);  
ALTER TABLE Class_rep RENAME TO CR_details;
```

Output

SQL query successfully executed. However, the result set is empty.

Available Tables

Class_details
counsellor
level_num
empty

CR_details
CR_name
section
empty

Student_details
Stu_name
S_id
d_o_b
empty

Subject_details
Sub_name
Sub_id
empty

EXPERIMENT 2

AIM- SQL data manipulation: At least four queries for each

1. Insert Command
2. Update Command
3. Delete Command
4. Merge Command
5. Select

INSERT

1. create table student(Stu_name varchar(20),S_id int,d_o_b varchar(10));
insert into student values ('Vishal', 1,'22-12-2000'), ('Alpha',2,'12-11-2000'), ('Priyanshu',3,'30-05-2000'), ('Priyanshi',4,'28-02-2000');
2. create table subject(Sub_name varchar(10), Sub_id int);
insert into subject
values('maths',234),(computer',303), ('database',236), ('dsa',235);
3. create table class_level(counsellor varchar(10), level_num int);
insert into class_level values('Ms Anjali', 10), ('Mr Navin', 11), ('Mr Manas', 12);
4. create table class_rep(CR_name varchar(10), section char);

UPDATE

1. update student set Stu_name='Alice' where S_id=2;
2. update subject set Sub_id=304 where Sub_name='computer';
3. update class_level set level_num-5 where counsellor='Mr Sahu ';
4. update class_rep set CR_name='Harsh' where section='A';

DELETE

1. delete from student where S_id=4;
2. delete from subject where Sub_id-236;
3. delete from class_level where counsellor='Mrs Babita';
4. delete from class_rep where CR_name='Anu';

MERGE

```
create table products (product_id int product_name
varchar(100), rate int);
insert into products values (1, 'tea', 10),(2, 'coffee', 20),(3, 'muffin',
30),(4, 'biscuit', 40);
create table updated_products (product_id int, product_name
```

```

varchar(100), rate int );
insert into updated_products values (1, 'tea', 10),(2, 'coffee', 20),(3,
'muffin', 30),(5, 'pizza', 60);
select * from products;
select * from updated_products;
merge into products t using
updated_products s on (t.product_id =
s.product_id)
"
when matched then
update set t.product_name = s.product_name,
t.rate = s.rate when not matched then
insert(t.product_name,t.rate) values
(s.product_name, s.rate); select * from products;

```

SELECT

1. select from students;
2. select from subject;
3. select * from class_rep;
4. select from class_level;

OUTPUT

INSERT

The screenshot shows a SQL editor interface with the following details:

- Input:** Contains the SQL code for creating four tables and inserting data into them.
- Available Tables:** Shows four tables: counselor, Class_level, Class_rep, and Student.
- counselor:** Contains 3 rows with columns counselor and level_num.
- Class_level:** Contains 3 rows with columns CR_name and section.
- Student:** Contains 3 rows with columns Stu_name, S_id, and d_o_b.
- Subject:** Contains 2 rows with columns Sub_name and Sub_id.

```

CREATE TABLE Student(Stu_name varchar(20),S_id int,d_o_b varchar(10));
INSERT INTO Student VALUES ('Vishal',1,'09-06-2002'),('Alpha',2,'01-01-2002'),
('Priyanshu',3,'02-01-2002');

CREATE TABLE Subject(Sub_name varchar(10),Sub_id int);

INSERT INTO Subject VALUES ('DBMS',1),('Kotlin',2),('React',4);

CREATE TABLE Class_level(counsellor varchar(10),level_num int);

INSERT INTO Class_level VALUES ('MS. Anjali',10),('MR. Navin',8),('MR. Manas',7);

CREATE TABLE Class_rep(CR_name varchar(10),section char);

INSERT INTO Class_rep VALUES ('Malik', 'J'), ('Akash', 'H'), ('Prashant', 'I');

```

Output: Displays the message "SQL query successfully executed. However, the result set is empty."

UPDATE

Input

```
UPDATE Student SET Stu_name='Priyansh' WHERE S_id=2;
UPDATE Subject SET Sub_id=3 WHERE Sub_name='React';
UPDATE Class_level SET level_num=8 WHERE counsellor='MR. Manas';
UPDATE Class_rep SET CR_name='Harsh' WHERE section='I';
```

Run SQL

Available Tables

counsellor	level_num
MS. Anjali	10
MR. Navin	8
MR. Manas	8

CR_name	section
Malik	J
Akash	H
Harsh	I

Output

SQL query successfully executed. However, the result set is empty.

Stu_name	S_id	d_o_b
Vishal	1	09-06-2002
Priyansh	2	01-01-2002
Priyanshu	3	02-01-2002

Sub_name	Sub_id
DBMS	1
Kotlin	2
React	3

DELETE

Input

```
DELETE FROM Student WHERE S_id=3;
DELETE FROM Subject WHERE Sub_id='React';
DELETE FROM Class_level WHERE level_num=8;
DELETE FROM Class_rep WHERE CR_name='Harsh';
```

Run SQL

Available Tables

counsellor	level_num
MS. Anjali	10

CR_name	section
Malik	J
Akash	H

Output

SQL query successfully executed. However, the result set is empty.

Stu_name	S_id	d_o_b
Vishal	1	09-06-2002
Priyansh	2	01-01-2002

Sub_name	Sub_id
DBMS	1
Kotlin	2
React	3

MERGE

db<>fiddle SQL Server 2022 no sample DB run markdown

By using db<>fiddle, you agree to license everything you submit by [Creative Commons CC0](#).



```
1 create table product(product_id int, product_name varchar(20), rate int);
2 insert into product values(1, 'Tea', 22), (2, 'Coffee', 30), (3, 'Muffin', 67);
3
4 create table updated_products (product_id int, product_name varchar(100), rate int);
5 insert into updated_products values (1, 'tea', 10), (2, 'coffee', 20), (3, 'muffin', 30);
6
7 v merge into product t
8 using updated_products s on t.product_id = s.product_id
9 when matched then
10 update set
11   t.product_name = s.product_name,
12   t.rate = s.rate
13 when not matched then
14 insert (product_id, product_name, rate) values (s.product_id, s.product_name, s.rate);
15 select *from product;
```

product_id	product_name	rate
1	tea	10
2	coffee	20
3	muffin	30



SELECT

Input

```
SELECT * FROM Student;
SELECT * FROM Subject;
SELECT * FROM Class_rep;
SELECT * FROM Class_level;
```

Available Tables

Class_level	
counsellor	level_num
MS. Anjali	10

Class_rep	
CR_name	section
Malik	J
Akash	H

Student		
Stu_name	S_id	d_o_b
Vishal	1	09-06-2002
Priyansh	2	01-01-2002

Subject	
Sub_name	Sub_id
DBMS	1
Kotlin	2
React	3

Output

Stu_name	S_id	d_o_b
Vishal	1	09-06-2002
Priyansh	2	01-01-2002

Sub_name	Sub_id
DBMS	1
Kotlin	2
React	3

CR_name	section
Malik	J

Input

```

SELECT * FROM Student;
SELECT * FROM Subject;
SELECT * FROM class_rep;
SELECT * FROM Class_level;

```

Output

Sub_name	Sub_id
DBMS	1
Kotlin	2
React	3

CR_name	section
Malik	J
Akash	H

counsellor	level_num
MS. Anjali	10

Available Tables

Class_level	
counsellor	level_num
MS. Anjali	10

Class_rep	
CR_name	section
Malik	J
Akash	H

Student		
Stu_name	s_id	d.o.b
Vishal	1	09-06-2002
Priyansh	2	01-01-2002

Subject	
Sub_name	Sub_id
DBMS	1
Kotlin	2
React	3

EXPERIMENT 3

AIM – SQL data control language and transaction control Language. At least three queries for each-

- a) Grant
- b) Revoke
- c) Commit
- d) Rollback
- e) Savepoint

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;

COMMIT

- grant connect to admin1;
- revoke connect from admin1;
- commit;

ROLLBACK

- create table subject(Sub_name varchar(10),Sub_id int);
insert into subject values('maths',234), ('computer',303), ('database',236);
select * from subject;
delete from subject where sub_id>235;
rollback;
- create table class_rep(CR_name varchar(10),section char);

```
insert into class_rep values ('Sarthak','A'), ('Sachin','B'), ('Priya','C');
delete from class_rep where section='C';
rollback;
```

- create table class_level(counsellor
varchar(10),level
num int);
insert into class_level values('Ms
Ruchika',10), ('Mrs Babita', 11). ('Ms
Rani', 12);
delete from class level where
level_num>10;
rollback;

SAVEPOINT

- create table a1(a_id int, a_serialno int);
insert into a1 values
(1,123), (2,452),
(3,526);
savepoint a1_table;
- update a1 set
a_serialno=242 where
a_id=2;
savepoint a1_update;
- drop table a1;
savepoint a1_drop;

OUTPUT

GRANT **REVOKE**

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

Script Output X
Task completed in 0.168 seconds

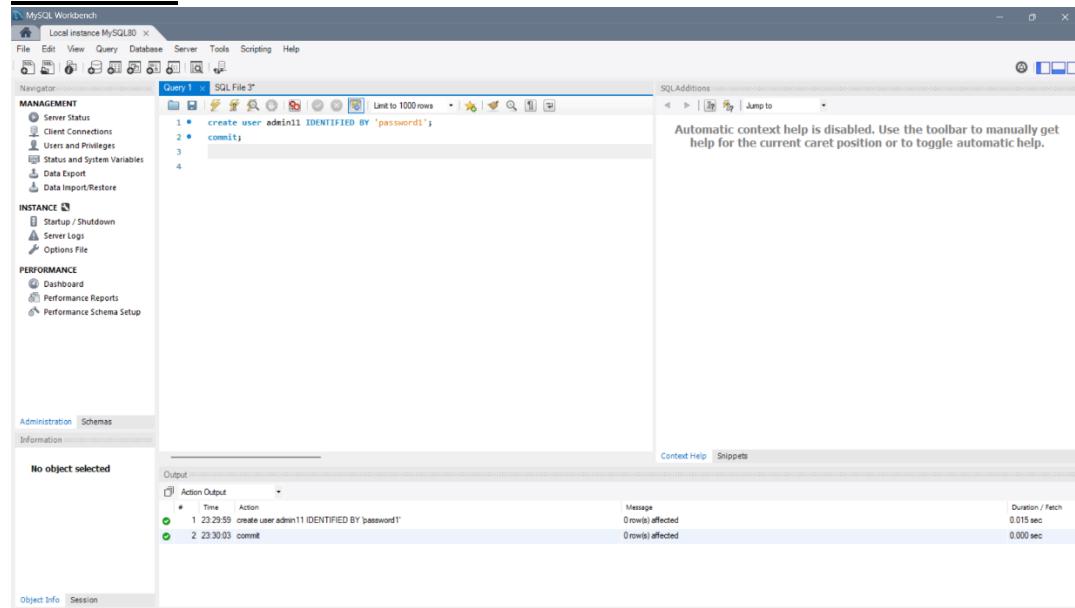
Grant succeeded.

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

Script Output X
Task completed in 0.056 seconds

Revoke succeeded.

COMMIT



ROLLBACK

```
MySQL> delete from subject where sun_id > 235;
ERROR 1054 (42S22): Unknown column 'sun_id' in 'where clause'
mysql> rollback;
Query OK, 0 rows affected (0.00 sec)

mysql> select* from subject;
+-----+-----+
| sub_name | Sub_id |
+-----+-----+
| Maths    | 234   |
| Computer | 303   |
| Database | 236   |
+-----+-----+
3 rows in set (0.00 sec)

mysql> delete from subject where Sub_id > 235;
Query OK, 2 rows affected (0.01 sec)

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

mysql> select *from subject;
+-----+-----+
| sub_name | Sub_id |
+-----+-----+
| Maths    | 234   |
+-----+-----+
1 row in set (0.00 sec)

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

mysql> |
```

SAVEPOINT

```
MySQL Workbench
Local instance MySQL80 x
File Edit View Query Database Server Tools Scripting Help
Navigator: Schemas
SCHEMAS
Filter objects
sys
visha
vishal
Tables
class_level
class_reo
product
student
subject
subject_details
updated_products
Views
Stored Procedures
Functions
Query 1 x
1 • create table a1(a_id int,a_serialno int);
2 • insert into a1 values(1,123);
3 • insert into a1 values(2,452);
4 • insert into a1 values(3,526);
5 • insert into a1 values(4,356);
6 • savepoint a1_table;
7
8
SQLAdditions: Automatic context help is disabled. Use the toolbar to manually get help for the current caret position or to toggle automatic help.
Administration Schemas
Information
Table: subject
Columns:
sub_name  varchar(20)
Sub_id    int
Object Info Session
Output: Action Output
# Time Action
1 10:20:34 create table a1(a_id int,a_serialno int)
2 10:21:04 create table a1(a_id int,a_serialno int)
3 10:22:16 insert into a1 values(1,123)
4 10:22:20 insert into a1 values(2,452)
5 10:22:45 insert into a1 values(3,526)
6 10:22:55 savepoint a1_table
Message
Error Code: 1046: No database selected. Select the default DB to be used by double-clicking its name in the Schemas tab.
Duration / Fetch
0.000 sec
0.015 sec
0.000 sec
0.000 sec
0.016 sec
0.000 sec
```

MySQL Workbench

Local instance MySQL80

File Edit View Query Database Server Tools Scripting Help

Navigator: Schemas

SCHEMAS: sys, visha, vishal

Tables: class_level, class_rep, product, student, subject, subject_details, updated_products

Views, Stored Procedures, Functions

Query 1

```

1 • set sql_safe_updates=0;
2 • update a1 set a_serialno=242 where a_id=2;
3 • savepoint a1_update;
4
5

```

SQLAdditions: SQLAdditions

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

Output: Action Output

#	Time	Action	Message	Duration / Fetch
1	10:45:37	set sql_safe_updates=0	0 row(s) affected	0.000 sec
2	10:45:37	update a1 set a_serialno=242 where a_id=2;	1 row(s) affected Rows matched: 1 Changed: 1 Warnings: 0	0.015 sec
3	10:45:37	savepoint a1_update	0 row(s) affected	0.000 sec

Context Help Snippets

Object Info Session

MySQL Workbench

Local instance MySQL80

File Edit View Query Database Server Tools Scripting Help

Navigator: Schemas

SCHEMAS: sys, visha, vishal

Tables: class_level, class_rep, product, student, subject, subject_details, updated_products

Views, Stored Procedures, Functions

Query 1

```

1 • drop table a1;
2 • savepoint a1_drop;

```

SQLAdditions: SQLAdditions

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

Output: Action Output

#	Time	Action	Message	Duration / Fetch
1	11:18:51	drop table a1	0 row(s) affected	0.016 sec
2	11:18:51	savepoint a1_drop	0 row(s) affected	0.000 sec

Context Help Snippets

Object Info Session

EXPERIMENT-4

AIM - Inbuilt functions in SQL:

- a) NVL functions
- b) Case function
- c) String functions
- d) Date functions
- e) Aggregation functions
- f Numeric function
- g) Case manipulation function
- h) Character manipulation function

Case Function

- select stu_name,
- case
- stu_id = 1 then 'good'
- when 4 then 'bad'
- else 'ok' end as student_type from student;
- select Marks,
- case Marks>90 then 'Excellent' when marks > 80 then 'Good' else 'Average' end as subject_type from subject;

String Function

- select ASCII(stu_name) as ASCII_Name from dual;
- select character_length('Vishal dubey');
- SELECT CHARACTER_LENGTH(stu_name) as Alias_Name from student_details;

Date Function

- select current_date from dual;
- select current_timestamp from dual;
- select round(date '2017-07-16','MM') as rounded_off from dual;

Aggregation Function

- create table stud(id int,regno int,name varchar(10),year int,marks int);
insert into stud values(1,23, 'Vishal',3,18),(2,24,'Naman',3,20),(3,25,'Shalini',5,19),(7,26, 'Aneesh',3,20);
select * from stud;
- select year,avg(marks) from stud group by year;
- select year,max(marks) from stud group by year having avg(marks)>19;
- select count(marks) from stud;

Numeric Functions

- select floor(4.1234) from dual;
- select log(10,1000) as log_1000_base10 from dual;
- select sin(3.141592653589793238) from dual;

Case Manipulation Function

- select * from student where lower(stu_name)= 'Priyanshu';
- select * from subject where upper(sub_name)='MATHS';
- select lower('Saturday') as lower_val from dual;

Character Manipulation Function

- select instr('this is anupriya', 'anu') location from dual;
- select concat(concat('Anu','priya'),' Johri')as sysdba_name from dual;
- select trim(leading'0' from '00120341523413000')as trimmed from dual;

NVL Function

```
select nvl (100, 200) from dual;  
select nvl (null,200) from dual;  
create table hello (id int, code int);  
insert into hello(code) values(12);  
insert into hello values(1,12);  
insert into hello(code)values(15);
```

```
insert into hello(code) values(25);
select * from hello;
select nvl(id,code)from hello where code>20;
select nvl(id,code) from hello;
```

OUTPUT

CASE FUNCTION

```
mysql> select stu_name,
    -> case
    -> when stu_id = 1 then 'good'
    -> when stu_id = 4 then 'bad'
    -> else 'ok'
    -> end as student_type
    -> from student_details;
+-----+-----+
| stu_name | student_type |
+-----+-----+
| Vishal Dubey | good
| Naman Gururani | good
| Aneesh Tripathi | ok
| Priyanshu Kushwaha | bad
+-----+-----+
4 rows in set (0.00 sec)

mysql> select marks,
    -> case
    -> when marks > 90 then 'Excellent'
    -> when marks > 80 then 'Good'
    -> else 'Average'
    -> end as student_type
    -> from student_details;
+-----+-----+
| marks | student_type |
+-----+-----+
| 80 | Average
| 90 | Good
| 95 | Excellent
| 91 | Excellent
+-----+-----+
4 rows in set (0.00 sec)

mysql>
```

STRING FUNCTION

```
| MySQL 8.0 Command Line Cli | + - |  
mysql> select *from student_details;  
+-----+-----+-----+-----+  
| stu_name | stu_id | d_o_b | Marks |  
+-----+-----+-----+-----+  
| Vishal Dubey | 1 | 2002-06-09 | 80 |  
| Naman Gunurani | 1 | 2002-06-23 | 90 |  
| Aneesh Tripathi | 3 | 2001-04-07 | 95 |  
| Priyanshu Kushwaha | 4 | 2001-12-30 | 91 |  
+-----+-----+-----+-----+  
4 rows in set (0.00 sec)  
  
mysql> SELECT ASCII(stu_name) as ASCII_Name FROM student_details;  
+-----+  
| ASCII_Name |  
+-----+  
| 86 |  
| 78 |  
| 65 |  
| 80 |  
+-----+  
4 rows in set (0.00 sec)  
  
mysql> select character_length('Vishal Dubey');  
+-----+  
| character_length('Vishal Dubey') |  
+-----+  
| 12 |  
+-----+  
1 row in set (0.00 sec)  
  
mysql> SELECT CHAR_LENGTH(stu_name) as Alias_Name FROM student_details;  
+-----+  
| Alias_Name |  
+-----+  
| 12 |  
| 14 |  
| 15 |  
| 18 |  
+-----+  
4 rows in set (0.00 sec)  
  
mysql>
```

DATE FUNCTION

```
| MySQL 8.0 Command Line Cli | + - |  
mysql> select current_date from dual;  
+-----+  
| current_date |  
+-----+  
| 2023-03-27 |  
+-----+  
1 row in set (0.00 sec)  
  
mysql> select current_timestamp from dual;  
+-----+  
| current_timestamp |  
+-----+  
| 2023-03-27 18:43:32 |  
+-----+  
1 row in set (0.00 sec)  
  
mysql> select round(date '2017-07-16','MM') as rounded_off from dual;  
+-----+  
| rounded_off |  
+-----+  
| 20170716 |  
+-----+  
1 row in set, 1 warning (0.00 sec)  
  
mysql> |
```

AGGREGATION FUNCTION

```
MySQL 8.0 Command Line Cli + ▾
mysql> select *from stud;
+----+-----+-----+----+-----+
| id | regno | name  | year | marks |
+----+-----+-----+----+-----+
| 1  | 23   | Vishal | 3   | 18  |
| 2  | 24   | Naman  | 3   | 20  |
| 3  | 25   | Shalini | 5   | 19  |
| 7  | 26   | Aneesh | 3   | 20  |
+----+-----+-----+----+-----+
4 rows in set (0.00 sec)

mysql> select year,avg(marks) from stud group by year;
+-----+-----+
| year | avg(marks) |
+-----+-----+
| 3    | 19.3333  |
| 5    | 19.0000  |
+-----+-----+
2 rows in set (0.00 sec)

mysql> select year,max(marks) from stud group by year having avg(marks)>19;
+-----+-----+
| year | max(marks) |
+-----+-----+
| 3    | 20         |
+-----+-----+
1 row in set (0.00 sec)

mysql> select count(marks) from stud;
+-----+
| count(marks) |
+-----+
|        4     |
+-----+
1 row in set (0.00 sec)

mysql>
```

NUMERIC FUNCTION

```
MySQL 8.0 Command Line Cli + ▾
mysql> select floor(4.1234) from dual;
+-----+
| floor(4.1234) |
+-----+
|        4      |
+-----+
1 row in set (0.00 sec)

mysql> select log(10,10000) from dual;
+-----+
| log(10,10000) |
+-----+
|        4      |
+-----+
1 row in set (0.00 sec)

mysql> select sin(3.14159265358) from dual;
+-----+
| sin(3.14159265358) |
+-----+
| 0.000000000979317720293495 |
+-----+
1 row in set (0.00 sec)

mysql>
```

CASE MANIPULATION FUNCTION

```
MySQL 8.0 Command Line Cli  +  ×
mysql> select *from stud where lower(name)='Vishal';
+----+-----+-----+-----+
| id | regno | name  | year | marks |
+----+-----+-----+-----+
|  1 |    23 | Vishal |   3 |    18 |
+----+-----+-----+-----+
1 row in set (0.00 sec)

mysql> select *from stud where upper(name)='Aneesh';
+----+-----+-----+-----+
| id | regno | name  | year | marks |
+----+-----+-----+-----+
|  7 |    26 | Aneesh |   3 |    20 |
+----+-----+-----+-----+
1 row in set (0.00 sec)

mysql> select lower('Saturday') as lower_val from dual;
+-----+
| lower_val |
+-----+
| saturday |
+-----+
1 row in set (0.00 sec)

mysql> |
```

CHARACTER MAINIPULATION FUNCTION

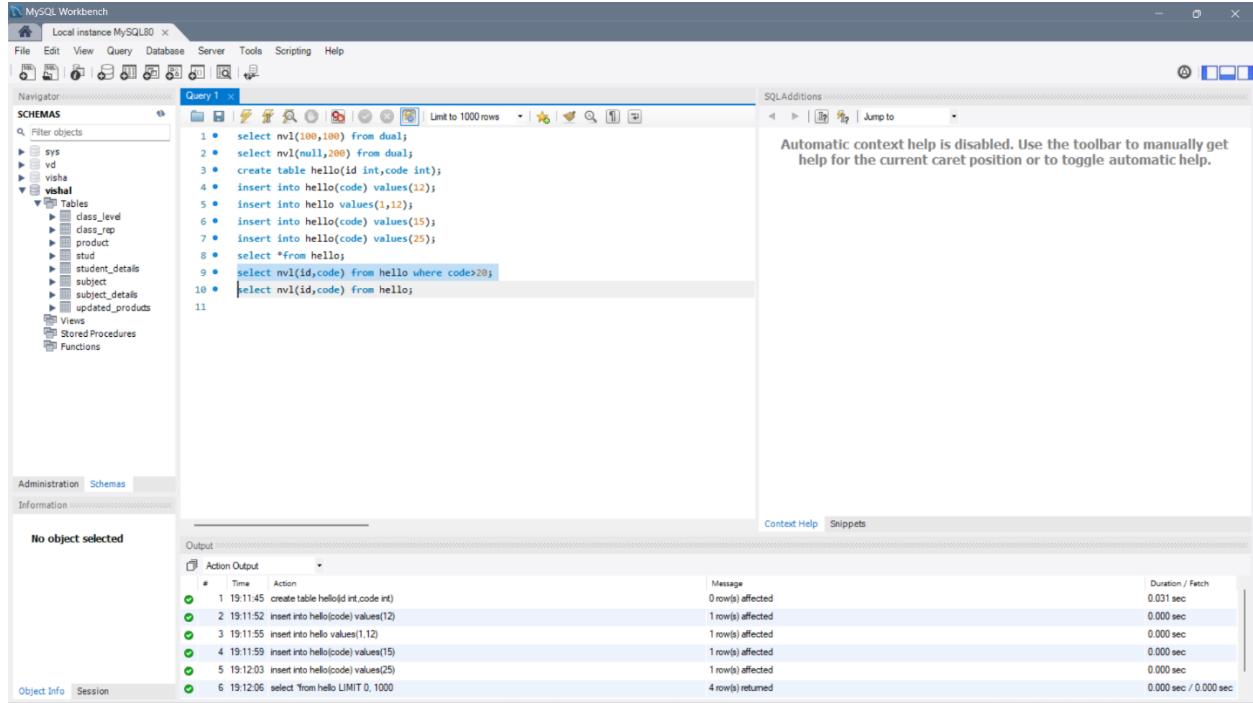
```
MySQL 8.0 Command Line Cli  +  ×
mysql> select instr('this is anupriya','anu') location from dual;
+-----+
| location |
+-----+
|      9 |
+-----+
1 row in set (0.00 sec)

mysql> select concat(concat('Anu','Priya'),'Johri') as sysdba_name from dual;
+-----+
| sysdba_name |
+-----+
| AnuPriyaJohri |
+-----+
1 row in set (0.00 sec)

mysql> select trim(leading'0' from '00120345548900900') as trimmed from dual;
+-----+
| trimmed   |
+-----+
| 120345548900900 |
+-----+
1 row in set (0.01 sec)

mysql> |
```

NVL FUNCTION



The screenshot shows the MySQL Workbench interface with a query editor window titled "Query 1". The code entered is:

```
1 • select nvl(100,100) from dual;
2 • select nvl(null,200) from dual;
3 • create table hello(id int,code int);
4 • insert into hello(code) values(12);
5 • insert into hello values(1,12);
6 • insert into hello(code) values(15);
7 • insert into hello(code) values(25);
8 • select *from hello;
9 • select nvl(id,code) from hello where code>20;
10 • select nvl(id,code) from hello;
11
```

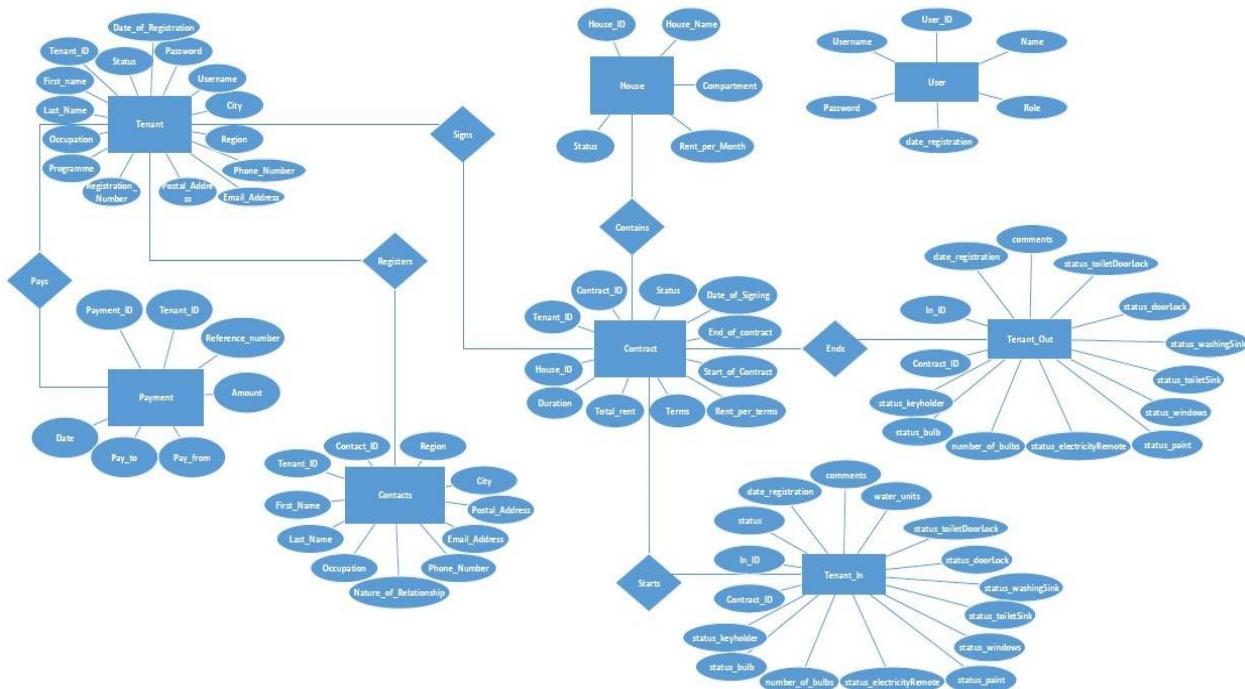
The output pane shows the results of the executed statements. The log table "Action Output" contains the following entries:

#	Time	Action	Message	Duration / Fetch
1	19:11:45	create table hello(id int,code int)	0 row(s) affected	0.031 sec
2	19:11:52	insert into hello(code) values(12)	1 row(s) affected	0.000 sec
3	19:11:55	insert into hello values(1,12)	1 row(s) affected	0.000 sec
4	19:11:59	insert into hello(code) values(15)	1 row(s) affected	0.000 sec
5	19:12:03	insert into hello(code) values(25)	1 row(s) affected	0.000 sec
6	19:12:06	select *from hello LIMIT 0, 1000	4 row(s) returned	0.000 sec / 0.000 sec

EXPERIMENT-05

AIM- Construct an 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



EXPERIMENT-06

AIM- Nested queries:

- a) Copying rows from another table
- b) Updating two columns with a sub query
- c) Deleting rows based on another table
- d) Using a subquery in an insert statement
- e) Creating table using subquery

COPYING ROWS FROM ANOTHER TABLE

CREATING TABLES

```
create table empl(name varchar(10),id int,salary int, holiday int, dept varchar(10));
create table admin_emp(name varchar(10),id int, email varchar(20));
insert into empl
values('Aneesh', 1,23412,12,MGT),('Vishal',2,43322,10,'Shalini'), ('Riya',4,65000,
15, 'RND'),
insert into admin_emp
values('Naman', 5,'naman21@gmail.com'), ('Manas',6,'Manascool@gmail.com'),
('Garvit',7,'garvithot2@gmail.com'), ('suraj',8,'surajboom@gmail.com');
select * from empl;
select from admin_emp;
```

COPYING ROWS FROM ANOTHER TABLE

```
insert into empl (name,id)select name,id from admin_emp where id>6;
select * from empl;
```

UPDATING TWO COLUMNS WITH A SUBQUERY

```
update empl set salary=70000, holiday=11 where id in(select id
from admin_emp where name='Garvit');
update empl set salary=60000, holiday=11 where id in(select id
from admin_emp where name='Suraj');
select * from empl;
```

DELETING ROWS BASED ON ANOTHER TABLE

```
delete from emp1 where id in(select id from admin_emp where id>=6);
select * from emp1;
```

USING SUBQUERY TO INSERT STATEMENT

```
insert into emp1(id)(select id from admin_emp where id>=6);
select * from emp1;
```

CREATING TABLE USING SUBQUERY

```
create table emp_salary as select id,salary from emp1 where id >=1
and id<=4; select * from emp_salary;
```

OUTPUT

COPYING ROWS FROM ANOTHER TABLE

The screenshot shows the MySQL Workbench interface with a query editor and a results grid.

Query Editor:

```
5 •      select* from emp1;
6 •      select * from admin_emp;
7
8      # Copying Rows From Another Table
9 •      insert into emp1(name,id) select name ,id from admin_emp where id>6 ;
10 •     select * from emp1;
11
12
100%  28:10 |
```

Result Grid:

	name	id	salary	holiday	dept
1	Aneesh	1	25412	12	MGT
2	Vishal	2	43326	13	ADMIN
3	Lokesh	3	43326	14	ADMIN
4	Riya	4	65000	15	R&D
5	Garvit	7	NUL	NUL	NUL
6	Suraj	8	NUL	NUL	NUL

Action Output:

Time	Action
1 10:36:09	select * from admin_emp LIMIT 0, 1000
2 10:38:20	insert into emp1(name,id) select name ,id from admin_emp where id>6
3 10:38:23	select * from emp1 LIMIT 0, 1000
4 10:40:02	select * from emp1 LIMIT 0, 1000

Response:

```
4 row(s) returned
2 row(s) affected Records
6 row(s) returned
6 row(s) returned
```

UPDATING TWO COLUMNS WITH A SUBQUERY

The screenshot shows a MySQL Workbench interface with a query editor and results grid. The query is:

```
11
12      # Updating Two Columns With a Subquery
13
14 •      set sql_safe_updates = 0;
15 •      update emp1 set salary =70000,holiday =11 where id in(select id from admin_emp where name = "Garvit");
16 •      update emp1 set salary =60000,holiday =11 where id in(select id from admin_emp where name = "Suraj");
17 •      select *from emp1;
```

The results grid shows the emp1 table with 8 rows. The last two rows, corresponding to Garvit and Suraj, have their salary and holiday values updated.

name	id	salary	holiday	dept
Aneesh	1	23412	12	MGT
Vishal	2	23322	10	ADMIN
Shalini	3	40000	14	ADMIN
Riya	4	65000	15	RND
Garvit	7	70000	11	NILL
Suraj	8	60000	11	NILL

Action Output:

Time	Action	Response
1 10:48:09	update emp1 set salary =70000,holiday =11 where id in(select id from...	0 row(s) affected Rows matched: 1 Changed: 0 Warnings: 0
2 10:48:13	update emp1 set salary =60000,holiday =11 where id in(select id from...	0 row(s) affected Rows matched: 1 Changed: 0 Warnings: 0
3 10:48:15	select *from emp1 LIMIT 0, 1000	6 row(s) returned

DELETING ROWS BASED ON ANOTHER TABLE

The screenshot shows a MySQL Workbench interface with a query editor and results grid. The query is:

```
16 •      update emp1 set salary =60000,holiday =11 where id in(select id from admin_emp where name = "Suraj");
17 •      select *from emp1;
18
19      # Deleting Rows Based On Another Table
20
21 •      delete from emp1 where id in(select id from admin_emp where id>=6);
22 •      select * from emp1;
```

The results grid shows the emp1 table with 7 rows. The row for Suraj (id 8) has been deleted.

name	id	salary	holiday	dept
Aneesh	1	23412	12	MGT
Vishal	2	23322	10	ADMIN
Shalini	3	40000	14	ADMIN
Riya	4	65000	15	RND

Action Output:

Time	Action	Response
1 10:48:09	update emp1 set salary =70000,holiday =11 where id in(select id from...	0 row(s) affected Rows matched: 1 Changed: 0 Warnings: 0
2 10:48:13	update emp1 set salary =60000,holiday =11 where id in(select id from...	0 row(s) affected Rows matched: 1 Changed: 0 Warnings: 0
3 10:48:15	select *from emp1 LIMIT 0, 1000	6 row(s) returned
4 10:49:38	delete from emp1 where id in(select id from admin_emp where id>=6)	2 row(s) affected
5 10:49:38	select * from emp1 LIMIT 0, 1000	4 row(s) returned

USING SUBQUERY TO INSERT STATEMENT

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

```
25  
26 •      insert into emp1(id)select id from admin_emp where id>=6;  
27 •  
100%  68:26 |  
Result Grid  Filter Rows: Q Search Export:  
name  id  salary  holiday  dept  
Aneesh 1  23412  12  MGT  
Vishal 2  43322  10  ADMIN  
Shalini 3  40000  14  ADMIN  
Riya 4  65000  15  RND  
NULL 6  NULL  NULL  NULL  
NULL 7  NULL  NULL  NULL  
NULL 8  NULL  NULL  NULL  
emp1 7 |
```

The Action Output pane shows the execution log:

Time	Action	Response
1 10:48:09	update emp1 set salary =70000,holiday =11 where id in(select id from...	0 row(s) affected Rows matched: 1 Changed: 0 Warnings: 0
2 10:48:13	update emp1 set salary =60000,holiday =11 where id in(select id from...	0 row(s) affected Rows matched: 1 Changed: 0 Warnings: 0
3 10:48:15	select * from emp1 LIMIT 0, 1000	6 row(s) returned
4 10:49:38	delete from emp1 where id in(select id from admin_emp where id>=6)	2 row(s) affected
5 10:49:38	select * from emp1 LIMIT 0, 1000	4 row(s) returned
6 10:50:59	insert into emp1(id)select id from admin_emp where id>=6	3 row(s) affected Records: 3 Duplicates: 0 Warnings: 0
7 10:50:59	select * from emp1 LIMIT 0, 1000	7 row(s) returned

CREATING TABLE USING SUBQUERY

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

```
26 •      insert into emp1(id)select id from admin_emp where id>=6;  
27 •      select * from emp1;  
28  
29      # Creating Table Using Subquery  
30  
31 •      create table emp_salary as select id,salary from emp1 where id>=1 and id<=4 ;  
32 •      select * from emp_salary;
```

The Result Grid shows the data in the emp_salary table:

id	salary
1	23412
2	43322
3	40000
4	65000

The Action Output pane shows the execution log:

Time	Action	Response
1 10:48:09	update emp1 set salary =70000,holiday =11 where id in(select id from...	0 row(s) affected Rows matched: 1 Changed: 0 Warnings: 0
2 10:48:13	update emp1 set salary =60000,holiday =11 where id in(select id from...	0 row(s) affected Rows matched: 1 Changed: 0 Warnings: 0
3 10:48:15	select *from emp1 LIMIT 0, 1000	6 row(s) returned
4 10:49:38	delete from emp1 where id in(select id from admin_emp where id>=6)	2 row(s) affected
5 10:49:38	select * from emp1 LIMIT 0, 1000	4 row(s) returned
6 10:50:59	insert into emp1(id)select id from admin_emp where id>=6	3 row(s) affected Records: 3 Duplicates: 0 Warnings: 0
7 10:50:59	select * from emp1 LIMIT 0, 1000	7 row(s) returned
8 10:52:39	create table emp_salary as select id,salary from emp1 where id>=1 and...	4 row(s) affected Records: 4 Duplicates: 0 Warnings: 0
9 10:52:39	select * from emp_salary LIMIT 0, 1000	4 row(s) returned

EXPERIMENT-07

AIM-Join queries:

- a) Equi-join
- b) Non Equi-join
- c) Outer join
- d) Left outer join
- e) Right outer join
- f) Self join

CREATE DATABASE

```
create table stud(id int,regno int,name varchar(10),year int,marks int);
insert into stud values(1,23, 'Vishal',3,18), (2,24,'Naman',3,20),
(3,25,'Shalini',5,19), (7,26, 'Priyanshu',2,20), (8,27, 'Aneesh',3,20);
select * from stud;
create table subject_details(id int,subject_name varchar(10), year int);
insert into subject_details values(1,'maths',3), (2,'eng',3), (3,'CD',5), (4,'dbms',5),
(5,'COA',3), (6,'CN',5), (7,'OS',3);
select * from subject_details;
```

EQUI JOIN

```
select * from stud,subject_details where
subject_details.year = stud.year;
```

NON EQUI JOIN

```
select * from stud,subject_details where stud.marks>18;
```

OUTER JOIN

```
select * from stud,subject_details where subject_details.id = stud.id;
```

LEFT OUTER JOIN

```
select * from stud LEFT OUTER JOIN subject_details on stud.id=
subject_details.id;
```

RIGHT OUTER JOIN

```
select * from stud RIGHT OUTER JOIN subject_details on stud.id=
subject_details.id;
```

SELF JOIN

- select a.id,a.name,b.id,b.name from stud a, stud b where a.id=byear;
- select a.id,a.subject_name,b.id,b.subject_name from subject_details a, subject_details b where a.id=b.year;

OUTPUT

CREATE DATABASE

```
MySQL 8.0 Command Line Cli x + v
mysql> select *from stud;
+---+---+---+---+
| id | regno | name | year |
+---+---+---+---+
| 1 | 23 | Vishal | 3 | 18 |
| 2 | 24 | Naman | 3 | 20 |
| 3 | 25 | Shalini | 5 | 19 |
| 7 | 26 | Aneesh | 3 | 20 |
| 5 | 27 | Priyanshu | 2 | 19 |
+---+---+---+---+
5 rows in set (0.00 sec)

mysql> select *from subject_details;
+---+---+---+
| id | subject_name | year |
+---+---+---+
| 1 | Maths | 3 |
| 2 | English | 3 |
| 3 | CD | 5 |
| 4 | DBMS | 5 |
| 5 | COA | 3 |
| 6 | CN | 5 |
| 7 | OS | 3 |
+---+---+---+
7 rows in set (0.00 sec)

mysql>
```

EQUI JOIN

```
MySQL 8.0 Command Line Cli x + v
mysql> select *from stud,subject_details where subject_details.year=stud.year;
+---+---+---+---+---+---+---+
| id | regno | name | year | marks | id | subject_name | year |
+---+---+---+---+---+---+---+
| 7 | 26 | Aneesh | 3 | 20 | 1 | Maths | 3 |
| 2 | 24 | Naman | 3 | 20 | 1 | Maths | 3 |
| 1 | 23 | Vishal | 3 | 18 | 1 | Maths | 3 |
| 7 | 26 | Aneesh | 3 | 20 | 2 | English | 3 |
| 2 | 24 | Naman | 3 | 20 | 2 | English | 3 |
| 1 | 23 | Vishal | 3 | 18 | 2 | English | 3 |
| 3 | 25 | Shalini | 5 | 19 | 3 | CD | 5 |
| 3 | 25 | Shalini | 5 | 19 | 4 | DBMS | 5 |
| 7 | 26 | Aneesh | 3 | 20 | 5 | COA | 3 |
| 2 | 24 | Naman | 3 | 20 | 5 | COA | 3 |
| 1 | 23 | Vishal | 3 | 18 | 5 | COA | 3 |
| 3 | 25 | Shalini | 5 | 19 | 6 | CN | 5 |
| 7 | 26 | Aneesh | 3 | 20 | 7 | OS | 3 |
| 2 | 24 | Naman | 3 | 20 | 7 | OS | 3 |
| 1 | 23 | Vishal | 3 | 18 | 7 | OS | 3 |
+---+---+---+---+---+---+---+
15 rows in set (0.00 sec)

mysql>
```

NON EQUI JOIN

```
MySQL 8.0 Command Line Cli + v
mysql> select *from stud,subject_details where stud.marks>18;
+---+---+---+---+---+---+---+
| id | regno | name   | year | marks | id   | subject_name | year |
+---+---+---+---+---+---+---+
| 5  | 27   | Priyanshu | 2    | 19   | 1   | Maths        | 3   |
| 7  | 26   | Aneesh   | 3    | 20   | 1   | Maths        | 3   |
| 3  | 25   | Shalini  | 5    | 19   | 1   | Maths        | 3   |
| 2  | 24   | Naman    | 3    | 20   | 1   | Maths        | 3   |
| 5  | 27   | Priyanshu | 2    | 19   | 2   | English      | 3   |
| 7  | 26   | Aneesh   | 3    | 20   | 2   | English      | 3   |
| 3  | 25   | Shalini  | 5    | 19   | 2   | English      | 3   |
| 2  | 24   | Naman    | 3    | 20   | 2   | English      | 3   |
| 5  | 27   | Priyanshu | 2    | 19   | 3   | CD           | 5   |
| 7  | 26   | Aneesh   | 3    | 20   | 3   | CD           | 5   |
| 3  | 25   | Shalini  | 5    | 19   | 3   | CD           | 5   |
| 2  | 24   | Naman    | 3    | 20   | 3   | CD           | 5   |
| 5  | 27   | Priyanshu | 2    | 19   | 4   | DBMS         | 5   |
| 7  | 26   | Aneesh   | 3    | 20   | 4   | DBMS         | 5   |
| 3  | 25   | Shalini  | 5    | 19   | 4   | DBMS         | 5   |
| 2  | 24   | Naman    | 3    | 20   | 4   | DBMS         | 5   |
| 5  | 27   | Priyanshu | 2    | 19   | 5   | COA          | 3   |
| 7  | 26   | Aneesh   | 3    | 20   | 5   | COA          | 3   |
| 3  | 25   | Shalini  | 5    | 19   | 5   | COA          | 3   |
| 2  | 24   | Naman    | 3    | 20   | 5   | COA          | 3   |
| 5  | 27   | Priyanshu | 2    | 19   | 6   | CN           | 5   |
| 7  | 26   | Aneesh   | 3    | 20   | 6   | CN           | 5   |
| 3  | 25   | Shalini  | 5    | 19   | 6   | CN           | 5   |
| 2  | 24   | Naman    | 3    | 20   | 6   | CN           | 5   |
| 5  | 27   | Priyanshu | 2    | 19   | 7   | OS           | 3   |
| 7  | 26   | Aneesh   | 3    | 20   | 7   | OS           | 3   |
| 3  | 25   | Shalini  | 5    | 19   | 7   | OS           | 3   |
| 2  | 24   | Naman    | 3    | 20   | 7   | OS           | 3   |
+---+---+---+---+---+---+---+
28 rows in set (0.00 sec)

mysql>
```

OUTER JOIN

```
MySQL 8.0 Command Line Cli + v
mysql> select *from stud,subject_details where subject_details.id=stud.id;
+---+---+---+---+---+---+---+
| id | regno | name   | year | marks | id   | subject_name | year |
+---+---+---+---+---+---+---+
| 1  | 23   | Vishal | 3    | 18   | 1   | Maths        | 3   |
| 2  | 24   | Naman  | 3    | 20   | 2   | English      | 3   |
| 3  | 25   | Shalini | 5    | 19   | 3   | CD           | 5   |
| 5  | 27   | Priyanshu | 2    | 19   | 5   | COA          | 3   |
| 7  | 26   | Aneesh  | 3    | 20   | 7   | OS           | 3   |
+---+---+---+---+---+---+---+
5 rows in set (0.00 sec)

mysql>
```

LEFT OUTER JOIN

```
MySQL 8.0 Command Line Cli + v
mysql> select *from stud left outer join subject_details on stud.id=subject_details.id;
+----+-----+-----+-----+-----+-----+-----+
| id | regno | name   | year  | marks | id    | subject_name | year |
+----+-----+-----+-----+-----+-----+-----+
| 1  | 23   | Vishal | 3     | 18   | 1    | Maths        | 3   |
| 2  | 24   | Naman  | 3     | 20   | 2    | English       | 3   |
| 3  | 25   | Shalini | 5     | 19   | 3    | CD           | 5   |
| 7  | 26   | Aneesh | 3     | 20   | 7    | OS           | 3   |
| 5  | 27   | Priyanshu | 2   | 19   | 5    | COA          | 3   |
+----+-----+-----+-----+-----+-----+-----+
5 rows in set (0.00 sec)

mysql> |
```

RIGHT OUTER JOIN

```
MySQL 8.0 Command Line Cli + v
mysql> select *from stud right outer join subject_details on stud.id=subject_details.id;
+----+-----+-----+-----+-----+-----+-----+
| id | regno | name   | year  | marks | id    | subject_name | year |
+----+-----+-----+-----+-----+-----+-----+
| 1  | 23   | Vishal | 3     | 18   | 1    | Maths        | 3   |
| 2  | 24   | Naman  | 3     | 20   | 2    | English       | 3   |
| 3  | 25   | Shalini | 5     | 19   | 3    | CD           | 5   |
| NULL | NULL | NULL   | NULL  | NULL | 4    | DBMS         | 5   |
| 5  | 27   | Priyanshu | 2   | 19   | 5    | COA          | 3   |
| NULL | NULL | NULL   | NULL  | NULL | 6    | CN           | 5   |
| 7  | 26   | Aneesh | 3     | 20   | 7    | OS           | 3   |
+----+-----+-----+-----+-----+-----+-----+
7 rows in set (0.00 sec)

mysql> |
```

SELF JOIN

```
MySQL> select a.id,a.name,b.id,b.name from stud a, stud b where a.id = b.year;
+---+-----+---+-----+
| id | name   | id | name   |
+---+-----+---+-----+
| 3 | Shalini | 1 | Vishal |
| 3 | Shalini | 2 | Naman  |
| 5 | Priyanshu | 3 | Shalini |
| 3 | Shalini | 7 | Aneesh |
| 2 | Naman  | 5 | Priyanshu |
+---+-----+---+-----+
5 rows in set (0.00 sec)

MySQL> select a.id,a.subject_name,b.id,b.subject_name from subject_details a, subject_details b where a.id = b.year;
+---+-----+---+-----+
| id | subject_name | id | subject_name |
+---+-----+---+-----+
| 3 | CD           | 1 | Maths      |
| 3 | CD           | 2 | English    |
| 5 | COA          | 3 | CD         |
| 5 | COA          | 4 | DBMS       |
| 3 | CD           | 5 | COA        |
| 5 | COA          | 6 | CN         |
| 3 | CD           | 7 | OS         |
+---+-----+---+-----+
7 rows in set (0.00 sec)

mysql>
```

EXPERIMENT-08

AIM - Set operators and views:

- a) Union
- b) Union all
- c) Intersect
- d) Minus
- e) Creation of views
- f) Modifications of views
- g) Deletion of view
- h) Rownum query

UNION

```
select * from  
A_table union  
select * from B_table;
```

UNION ALL

```
select * from A_table union  
all  
select * from B_table;
```

INTERSECT

```
select * from A_table Pp  
intersect  
select * from B_table;
```

MINUS

```
select* from A  
minus  
select* from B_table;
```

CREATION OF VIEWS

```
create view academics as  
select stu_name from  
student;
```

MODIFICATION OF VIEWS

alter view academics compile;

DELETION OF VIEW

drop view academics;

ROWNUM QUERY

select from stud where rownum<10;

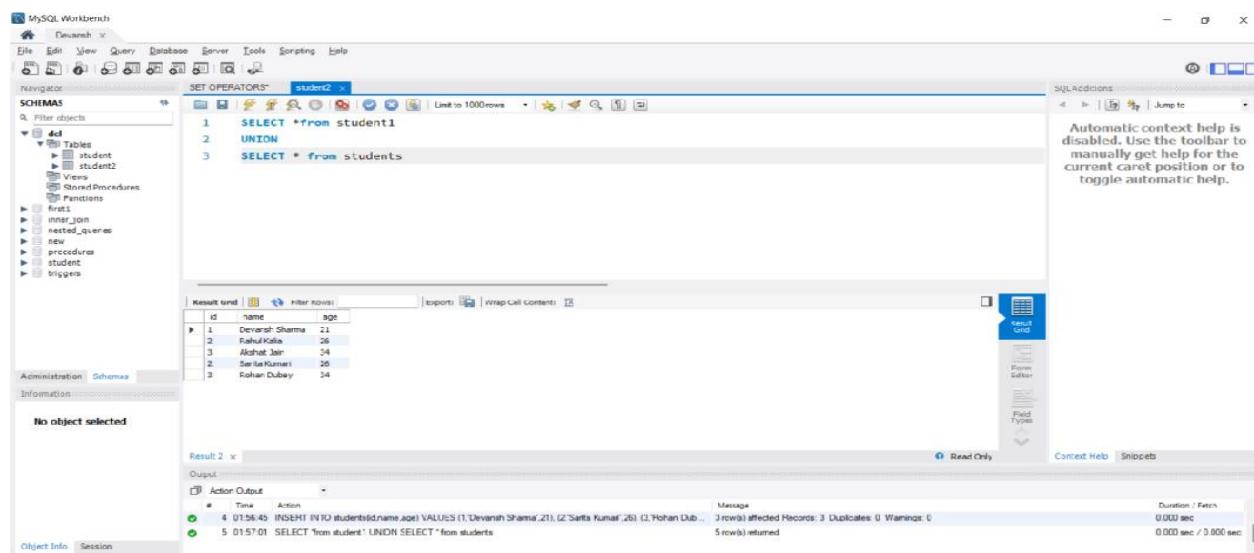
select from stud where rownum<4 order by name,

update stud set id-rownum;

select * from stud;

OUTPUT

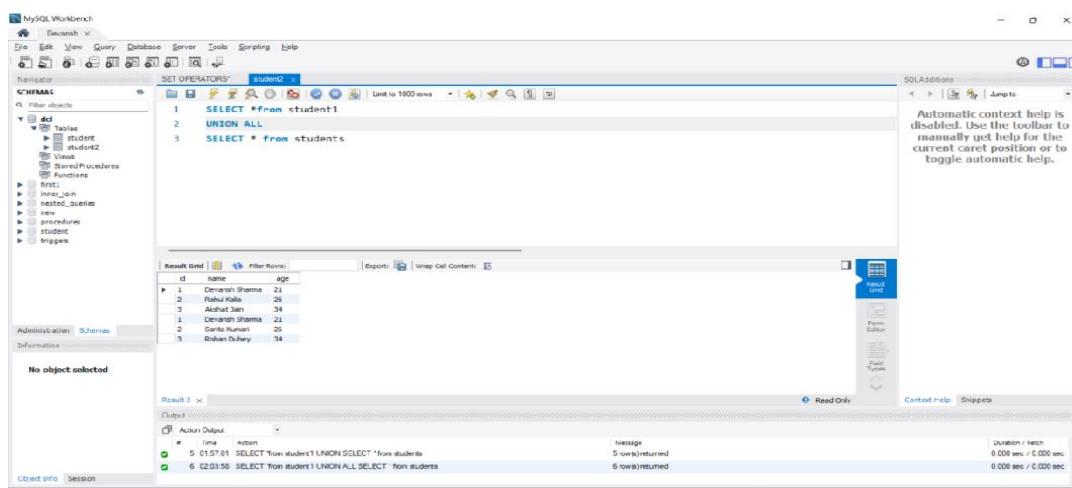
UNION



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

- Schemas:** The schema '4cl' is selected.
- Tables:** The table 'student' is selected.
- SQL Editor:** The query is:1 SELECT * from student1
2 UNION
3 SELECT * from students
- Result Grid:** The results show three rows of data from the 'student' table.
- Output Window:** The log shows two actions:
 - Action 4: Insert into student(id, name, age) VALUES (1, 'Devarsh Sharma', 21)
 - Action 5: SELECT * from student UNION SELECT * from studentsMessage: 3 rows affected, Records: 3, Duplicates: 0, Warnings: 0. Duration / Fetch: 0.000 sec / 0.000 sec.

UNION ALL



The screenshot shows the MySQL Workbench interface with a query editor window titled 'student2'. The query is:

```

1 SELECT * from student1
2 UNION ALL
3 SELECT * from students

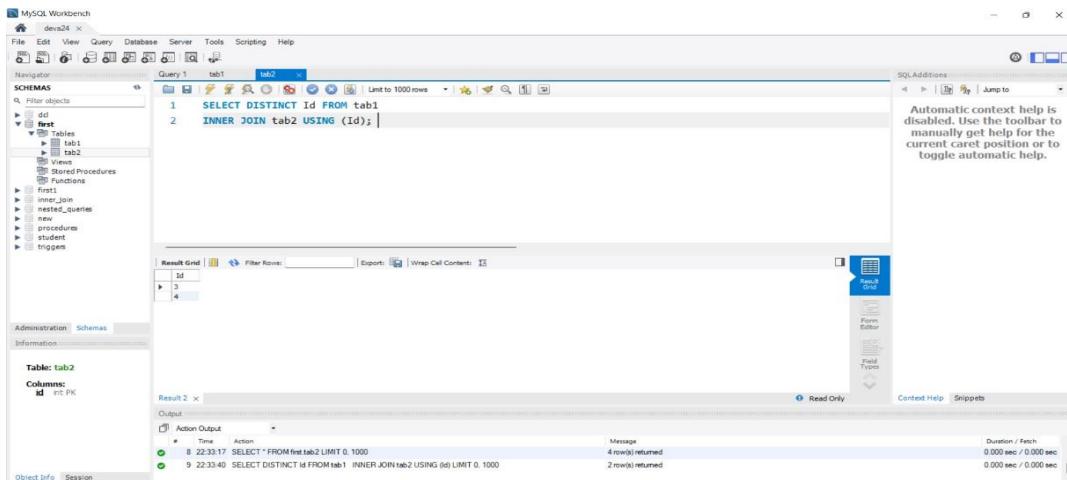
```

The result grid displays the combined results of both queries:

ID	Name	Age
1	Darshan Sharma	21
2	Rakul Patel	26
3	Ankit Jain	24
1	Darshan Sharma	21
2	Gupta Humani	20
3	Sneha Uday	24

The SQLAdditions panel on the right indicates that automatic context help is disabled.

INTERSECT



The screenshot shows the MySQL Workbench interface with two query windows. Query 1 (tab1) contains:

```

1 SELECT DISTINCT Id FROM tab1
2 INNER JOIN tab2 USING (id);

```

The result grid shows the intersection of the two tables:

ID
3
4

Query 2 (tab2) contains:

```

1 SELECT * FROM tab1 LIMIT 0, 1000
2 SELECT DISTINCT Id FROM tab1 INNER JOIN tab2 USING (id) LIMIT 0, 1000

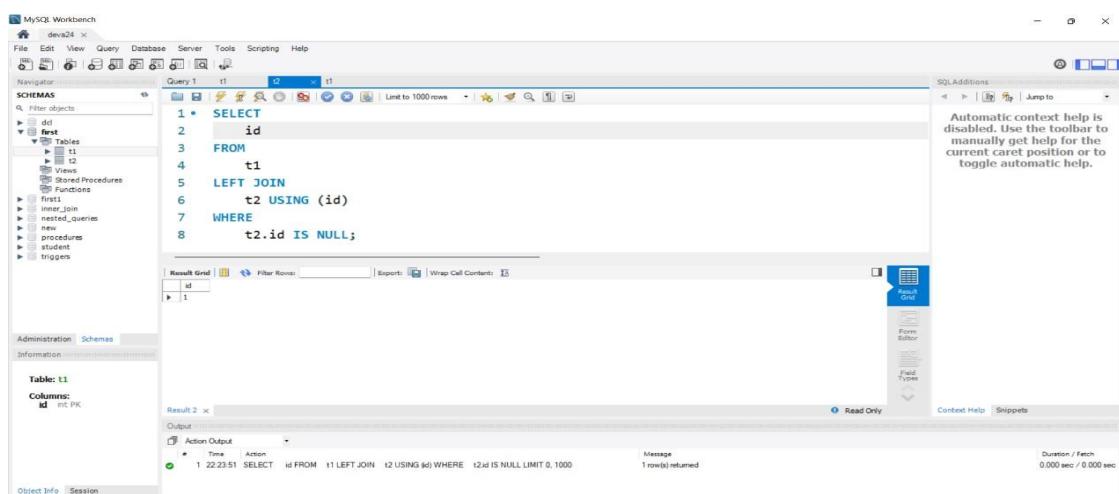
```

The result grid shows the distinct IDs from tab1:

ID
3
4

The SQLAdditions panel on the right indicates that automatic context help is disabled.

MINUS



The screenshot shows the MySQL Workbench interface with two query windows. Query 1 (t1) contains:

```

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

```

The result grid shows the IDs from t1 that do not have a corresponding row in t2:

ID
1

Query 2 (t2) contains:

```

1 SELECT * FROM t1 LEFT JOIN t2 USING (id) WHERE t2.id IS NULL LIMIT 0, 1000

```

The result grid shows the same ID from t1:

ID
1

The SQLAdditions panel on the right indicates that automatic context help is disabled.

CREATION OF VIEW

The screenshot shows the MySQL Workbench interface. In the left sidebar, under the 'Schemas' section, the 'first1' schema is selected. In the main SQL editor window, the following SQL code is written:

```
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 SQL editor, 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 status bar at the bottom right indicates a duration of 0.000 sec / 0.000 sec.

MODIFICATION OF VIEW

The screenshot shows the MySQL Workbench interface. In the left sidebar, under the 'Schemas' section, the 'first1' schema is selected. In the main SQL editor window, the following SQL code is written:

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

Below the SQL editor, the 'Result Grid' shows the data from the 'new_data' table:

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

The status bar at the bottom right indicates a duration of 0.015 sec / 0.000 sec.

DELETION OF VIEW

The screenshot shows the MySQL Workbench interface. In the top-left pane, the Navigator displays the schema 'first1' with its tables ('city', 'student') and views ('new_data'). The main query editor window contains the SQL command:

```
1 • DROP VIEW new_data;
```

The Output pane shows the execution results:

Action	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

ROWNUM QUERY

The screenshot shows the MySQL Workbench interface. In the top-left pane, the Navigator displays the schema 'data24' with its tables ('city', 'student_table', 'courses') and views ('student_table'). The main query editor window contains the SQL command:

```
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 Result Grid pane displays the data from the 'data' view:

id	name	age	cty	cid	ctyname
2	Salman Khan	24	1	1	Agra
3	Mehra Khan	35	3	3	Bhopal
4	Sertia Kumar	38	2	2	Delhi

The Output pane shows the execution results:

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

EXPERIMENT -09

Conditional & Iterative Statements

AIM – PL/SQL conditional and iterative statements.

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

QUERY:

a) **Conditional Statement**

FOR CREATING PROCEDURE

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

FOR EXECUTION

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

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

- Schemas:** dev24
- Query Editor (Query 1):**

```

1  DELIMITER $$

2  3  CREATE PROCEDURE GetCustomerLevel(
4      IN pCustomerNumber INT,
5      OUT pCustomerLevel VARCHAR(20))
6  BEGIN
7      DECLARE credit DECIMAL(10,2) DEFAULT 0;
8
9      SELECT creditLimit
10     INTO credit
11    FROM customers
12   WHERE customerNumber = pCustomerNumber;
13
14   IF credit > 50000 THEN
15       SET pCustomerLevel = 'PLATINUM';
16   END IF;
17 END$$
18
19  DELIMITER ;

```
- Output:**
 - Action Output: 1 rows affected
 - Time: 0.000 sec

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

- Schemas:** dev24
- Query Editor (Query 1):**

```

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

```

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

Iterative Statement

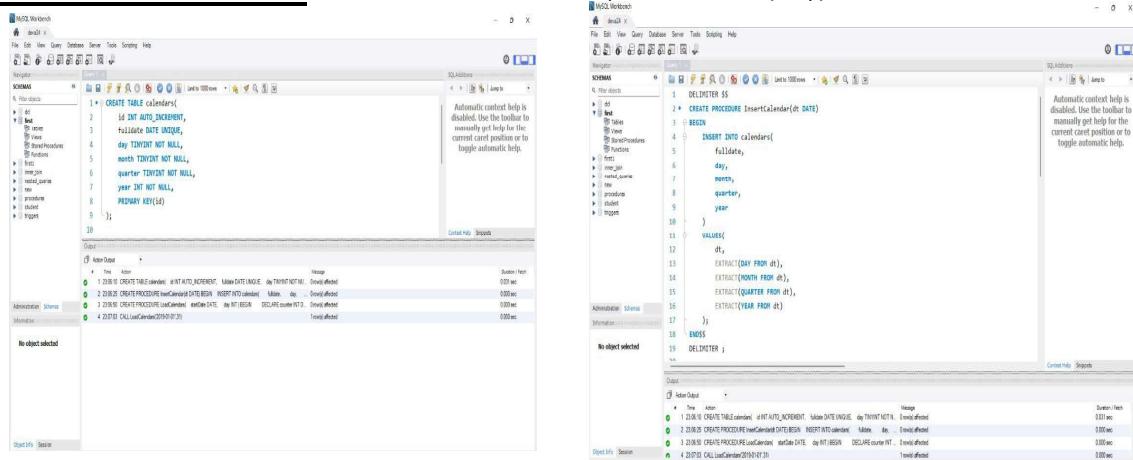
```
CREATE TABLE calendars (
id INT AUTO_INCREMENT,
fulldate DATE UNIQUE, day TINYINT NOT NULL,
month TINYINT NOT NULL,
quarter TINYINT NOT NULL, year INT NOT NULL, PRIMARY KEY(id));
```

FOR CREATING PROCEDURE

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



```

DELIMITER $$
CREATE PROCEDURE LoadCalendars(
    startDate DATE,
    day INT
)
BEGIN
    DECLARE counter INT DEFAULT 1;
    DECLARE dt DATE DEFAULT startDate;

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

```

The screenshot shows the MySQL Workbench interface with the SQL editor tab open. The code for the stored procedure is pasted into the editor. The output pane shows the execution of the procedure with the command `CALL LoadCalendars('2019-01-01',31);` and the resulting message `1 row(s) affected`.

The screenshot shows the MySQL Workbench interface with the SQL editor tab open. The output pane displays the command `CALL LoadCalendars('2019-01-01',31);` and the message `1 row(s) affected`.

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

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

EXPERIMENT 10

PL/SQL Procedures on sample exercise.

Aim: To Execute Procedure in MySql

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

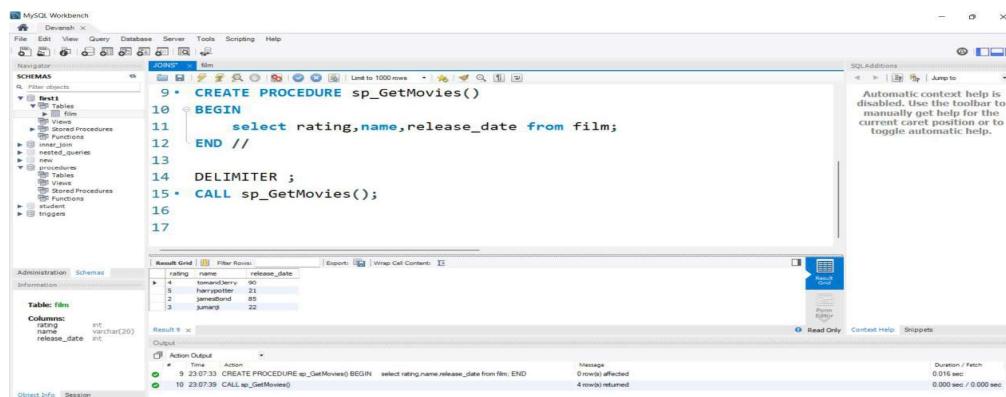
If we consider the enterprise application, we always need to perform specific tasks such as database cleanup, processing payroll, and many more on the database regularly. Such tasks involve multiple SQL statements for executing each task. This process might be easy if we group these tasks into a single 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 //  
PROCEDURE sp_GetMovies()BEGIN  
select rating,name,release_date from film;END //  
  
DELIMITER ;
```

```
CALL sp_GetMovies();
```



The screenshot shows the MySQL Workbench interface. In the top-left pane, the 'Schemas' tree shows a database named 'first1'. Under 'first1', there is a 'Tables' node which contains a table named 'film'. The 'film' table has three columns: 'rating' (int), 'name' (varchar(20)), and 'release_date' (int). Below the table definition, the table structure is listed: 'Table: film', 'Columns: rating int, name varchar(20), release_date int'. In the main query editor window, the following SQL code is written:

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

Below the code, the 'Result Grid' shows the data from the 'film' table:

	rating	name	release_date
1	4	tomandJerry	90
2	5	harrypotter	21
3	2	jamesBond	85
4	3	jumanji	22

The 'Object Info' tab at the bottom left shows the procedure 'sp_GetMovies' with the following details:

- Action: Create
- Time: 23:07:33
- Message: 0 rows affected
- Duration / Fetch: 0.015 sec / 0.000 sec / 0.000 sec

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

EXPERIMENT 11

PL/SQL Functions

AIM - PL/SQL functions

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

QUERY

CREATING TABLE FOR FUNCTION

```
CREATE TABLE
employee(emp_id INT,
fname
varchar(50),
lname
varchar(50),
start_date
date
);
INSERT INTO
employee(emp_id, fname, lname, start_da
te)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 a query editor window titled 'Query 1'. The code entered is:

```
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','Tyler','2000-02-09'),
13    (4,'Susan','Hawthorne','2000-01-15')
```

The results grid shows the following data:

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

The status bar at the bottom indicates:

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

The screenshot shows the MySQL Workbench interface with a query editor window titled 'Query 1'. The code entered is:

```
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 results grid shows the following data:

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

The status bar at the bottom indicates:

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

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

EXPERIMENT 12

PL/SQL Cursors

Aim:- To Execute Cursors in MySql

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

QUERY:-

```
CREATE TABLE GetVatsaCursor(
C_ID INT PRIMARY KEY AUTO_INCREMENT,
c_name VARCHAR(50), c_address VARCHAR(200)); CREATE TABLE Vbackupdata(C_ID INT,
c_name VARCHAR(50), c_address VARCHAR(200));
INSERT INTO GetVatsaCursor(c_name, c_address) VALUES('Test', '132, VatsaColony'),
('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. In the central query editor, the following PL/SQL code is running:

```

51 • CALL firstCurs();
52 SELECT * FROM Vbackupdata;
53

```

The results grid displays a list of records from the Vbackupdata view:

	c_id	c_name	c_address
1	Test	132, Vatva Colony	
2	Pranali	133, Vatva Colony	
3	Vatva	134, Vatva Colony	
4	Onkar	135, Vatva Colony	
5	Pranali	136, Vatva Colony	
6	Simran	137, Vatva Colony	
7	Anamika	138, Vatva Colony	
8	Anamika	139, Vatva Colony	
9	Radhika	140, Vatva Colony	
10	Pranali	141, Vatva Colony	

Below the results, the output pane shows the execution details:

- Action Output: Time Action
- 9 22:59:29 CALL firstCurs(); SELECT * FROM Vbackupdata;
- 10 22:59:29 CALL firstCurs(); SELECT * FROM Vbackupdata;

Message: 1 row(s) affected. 20 row(s) returned.

Duration / Fetch: 0.000 sec. / 0.000 sec.

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

EXPERIMENT 13

PL/SQL Exception Handling

Aim- PL/SQL exception handling.

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

QUERY:-

```
DROP PROCEDURE IF EXISTS
InsertSupplierProduct;DELIMITER $$

CREATE PROCEDURE
InsertSupplierProduct( IN inSupplierId
INT,
IN inProductId INT
)
BEGIN

-- exit if the duplicate key occurs

DECLARE EXIT HANDLER FOR 1062 SELECT

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

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

-- insert a new row into the SupplierProducts

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

FROM SupplierProducts

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

OUTPUT

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

- Query Editor:** Displays the SQL code for creating a stored procedure named `InsertSupplierProduct`. The code includes a `DROP PROCEDURE IF EXISTS` statement, a `CREATE PROCEDURE` block with parameters `IN inSupplierId INT` and `IN inProductId INT`, and exception handling blocks for `1062` (Duplicate keys error) and `SQLSTATE 23000` (SELECT 'SQLException encountered' Message).
- Result Grid:** Shows the output of the procedure creation. It lists two rows:
 - Action: `CREATE PROCEDURE InsertSupplierProduct(IN inSupplierId INT, IN inProductId INT) BEGIN -- exit if the duplicate key occurs`
 - Action: `CALL InsertSupplierProduct(1,3)`
- Message:** Shows the message "SQLException encountered".
- Information Bar:** Includes a tooltip: "Automatic context help is disabled. Press F1 to manually get help for the current caret position or to toggle automatic help."

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

EXPERIMENT 14 **PL/SOL 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_AfterInsetEmpAFTER
INSERT
ON emp
FOR EACH
ROW
BEGIN
INSERT INTO emp_audit
```

```
VALUES(null,concat('newrow',date_format(now(),'%d-%m-%y %h:%i:%s
%op')));END//
```

```
DELIMITER
; INSERT
INTO emp
VALUES
(null,'Akash',22),
(null,'Devansh',18),
(null,'Akshat',21),
(null,'Rahul',24);
```

The screenshot shows the MySQL Workbench interface with the 'emp_audit' table selected. The table has columns 'id' (int AI PK), 'name' (varchar(50)), and 'age' (int). The data grid displays four rows:

	id	name	age
1	1	Aakash	22
2	2	Dinesh	28
3	3	Alokhant	21
4	4	Rahul	24

The SQL Editor at the bottom shows the query: `SELECT * FROM first1.emp_audit;`

This screenshot shows the same MySQL Workbench interface after inserting four new rows into the 'emp_audit' table. The data grid now displays eight rows, including the original four and the four new audit log entries:

	id	audit_description
1	1	A new row is inserted at 24-04-22 10:50:54 PM
2	2	A new row is inserted at 24-04-22 10:50:54 PM
3	3	A new row is inserted at 24-04-22 10:50:54 PM
4	4	A new row is inserted at 24-04-22 10:50:54 PM
5	5	5 22:51:00 SELECT * FROM first1.emp_audit
6	6	6 22:51:03 SELECT * FROM first1.emp_audit

The SQL Editor at the bottom shows the query: `SELECT * FROM first1.emp_audit;`

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

EXPERIMENT 15

Frame and Execute PL/SQL Cursor & ExceptionalHandling

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 Basedon 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 managementsystem 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 administrativemanagement 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 topapers.

MODULES

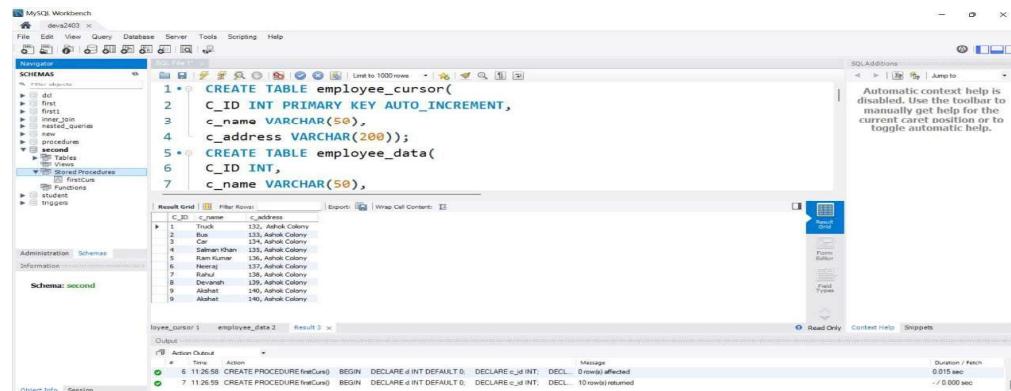
- Tenant
- Owner
- Contract
- Payment

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



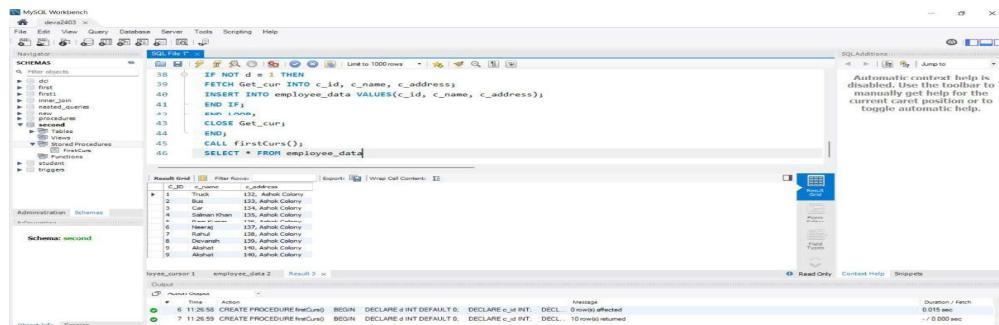
```

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

C_ID	c_name	c_address
1	Truck	132, Ashok Colony
2	Bala	133, Ashok Colony
3	Car	134, Ashok Colony
4	Suv	135, Ashok Colony
5	Ram Kumar	136, Ashok Colony
6	Neeraj	137, Ashok Colony
7	Shivam	138, Ashok Colony
8	Devansh	139, Ashok Colony
9	Akhila	140, Ashok Colony
9	Aishwari	140, Ashok Colony

```

6 11:26:58 CREATE PROCEDURE firstCurs() BEGIN DECLARE d INT DEFAULT 0; DECLARE c_id INT; DECL... 0 rows affected
7 11:26:59 CREATE PROCEDURE(firstCurs) BEGIN DECLARE d INT DEFAULT 0; DECLARE c_id INT; DECL... 10 rows returned
  
```



```

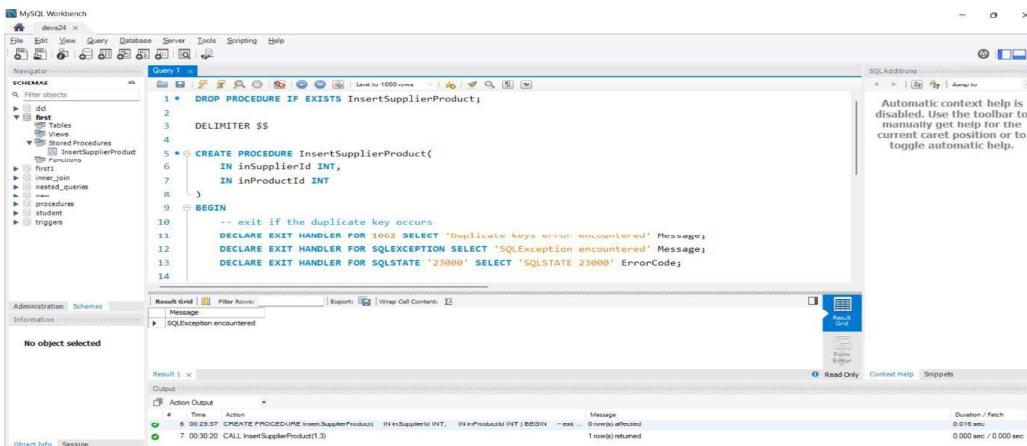
IF NOT d = 1 THEN
  FETCH Get_Cur INTO c_id, c_name, c_address;
  INSERT INTO employee_data VALUES(c_id, c_name, c_address);
END IF;
CLOSE Get_Cur;
CALL firstCurs();
SELECT * FROM employee_data;
  
```

C_ID	c_name	c_address
1	Truck	132, Ashok Colony
2	Bala	133, Ashok Colony
3	Car	134, Ashok Colony
4	Suv	135, Ashok Colony
5	Ram Kumar	136, Ashok Colony
6	Neeraj	137, Ashok Colony
7	Shivam	138, Ashok Colony
8	Devansh	139, Ashok Colony
9	Akhila	140, Ashok Colony
9	Aishwari	140, Ashok Colony

```

6 11:26:58 CREATE PROCEDURE(firstCurs) BEGIN DECLARE d INT DEFAULT 0; DECLARE c_id INT; DECL... 0 rows affected
7 11:26:59 CREATE PROCEDURE(firstCurs) BEGIN DECLARE d INT DEFAULT 0; DECLARE c_id INT; DECL... 10 rows returned
  
```

b) Exception Handling



```

DROP PROCEDURE IF EXISTS InsertSupplierProduct;
DELIMITER $$

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

```

6 00:23:37 CREATE PROCEDURE InsertSupplierProduct( IN inSupplierId INT, IN inProductId INT ) BEGIN ... 0 rows affected
7 00:30:20 CALL InsertSupplierProduct(1)
  
```

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