****

**DATABASE MANAGEMENT SYSTEMS**

**(Code 18CSC303J)**

**B.Tech (CSE) – 3rd year/6th Semester**

**Name: Aditya Saxena**

**Registration No: RA2011003030223**

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

FACULTY OF ENGINEERING & TECHNOLOGY

SRM INSTITUTE OF SCIENCE & TECHNOLOGY,

DELHI NCR CAMPUS, MODINAGAR

SIKRI KALAN, DELHI MEERUT ROAD, DIST. – GHAZIABAD - 201204

https://www.srmup.in/

Even Semester (2022-2023)

****

**BONAFIDE CERTIFICATE**

**Registration no.RA201103030223**

*Certified to be the bonafide record of work done by* ***Aditya Saxena*** *of 6th semester 3nd year* ***B. TECH*** *degree course in* ***SRM INSTITUTE OF SCIENCE AND TECHNOLOGY,*** *NCR Campus of Department of Computer Science & Engineering, in DATABASE MANAGEMENT SYSTEMS, during the academic year 2022-23.*

**Dr. Rolly Gupta Head of the Department**

**(Assistant Professor) (CSE)**

*Submitted for university examination held on \_\_/\_\_\_\_/\_\_\_ at SRM IST, NCR Campus.*

**Internal Examiner-I Internal Examiner-II**

***Index***

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **S.No.** | **Table of Contents** | **Date of Experiment** | **Date of Submission** | **Faculty Sign** |
| 1 | Lab 1: SQL Data Definition Language Commands | 09/01/23 | 09/01/23 |  |
| 2 | Lab 2: SQL Data Manipulation Language Commands | 16/01/23 | 16/01/23 |  |
| 3 | Lab 3: SQL Data Control Language Commands and Transaction Control commands | 23/01/23 | 23/01/23 |  |
| 4 | Lab 4: Inbuilt functions in SQL | 30/01/23 | 30/01/23 |  |
| 5 | Lab 5: Construct an ER Model for the application to be constructed to a Database | 13/02/23 | 13/02/23 |  |
| 6 | Lab 6: Generates the SQL commands from an ER Model | 20/02/23 | 20/02/23 |  |
| 7 | Lab 7: Join Queries | 27/02/23 | 27/02/23 |  |
| 8 | Lab 8: Set Operators & Views. | 20/03/23 | 20/03/23 |  |
| 9 | Lab9: PL/SQL Conditional and Iterative Statements | 27/03/23 | 27/03/23 |  |
| 10 | Lab10: PL/SQL Procedures on sample exercise. | 27/03/23 | 27/03/23 |  |
| 11 | Lab 11: PL/SQL Functions | 03/04/23 | 03/04/23 |  |
| 12 | Lab 12: PL/SQL Cursors | 03/04/23 | 03/04/23 |  |
| 13 | Lab 13: PL/SQL Exception Handling | 10/04/23 | 10/04/23 |  |
| 14 | Lab 14: PL/SQL Trigger | 10/04/23 | 10/04/23 |  |
| 15/16 | \*Lab 15/16: Frame and execute the appropriate PL/SQL Cursors and Exceptional Handling for the project. | 17/04/23 | 17/04/23 |  |

**EXPERIMENT 1**

**AIM: –** SQL Data Definition: write queries for each

* Create
* Alter
  + Drop
  + Rename

**PROCEDURE:-**

1. Goto Start and click on programs.
2. Click on SQL Workbench.
3. Then a pop-up menu will appear to authenticate you as user.
4. Now SQL Workbench window will get opened for working with various SQL commands in it.

**THEORY:-**

**CREATE**

It defines each column of the table uniquely. Each column has minimum of three attributes, a name , data type and size.

#### Syntax:

Create table <table name> (<col1> <datatype>(<size>),<col2> <datatype><size>));

1. create table halls(hall\_id int primary key,hall\_title varchar(25),seats\_no int);
2. create table movie(movie\_id int primary key,movie\_title varchar(25),movie\_genre varchar(25),director varchar(25),duration int);
3. create table screening(scr\_id int primary key,movie\_id int,hall\_id int,scr\_date date,scr\_start time,scr\_end time,foreign key fk1(movie\_id) references movie(movie\_id), foreign key fk2(hall\_id) references halls(hall\_id));

**ALTER**

It is used to add, delete, or modify columns in an existing table. This statement is also used to add and drop various constraints on an existing table.

#### Syntax:

Alter table <tablename> modify(<col><newdatatype>(<newsize>)); Ex:alter table emp modify(ename varchar2(15));

1. alter table halls drop column hall\_id;
2. alter table movie rename column movie\_title to movie\_title;
3. alter table halls add column location vatchar(25);
4. alter table movie modify column movie\_title varchar(20);

**RENAME**

This command simply used to change the table name.

#### Syntax:

rename 'TableName.oldcolumn\_name', 'newcolumn\_name', 'COLUMN';

1. rename table halls to cinema\_halls;
2. rename table movie to movie\_theatres;
3. rename table screening to premiere;

**DROP**

The DROP TABLE statement is used to drop an existing table in a database or table.

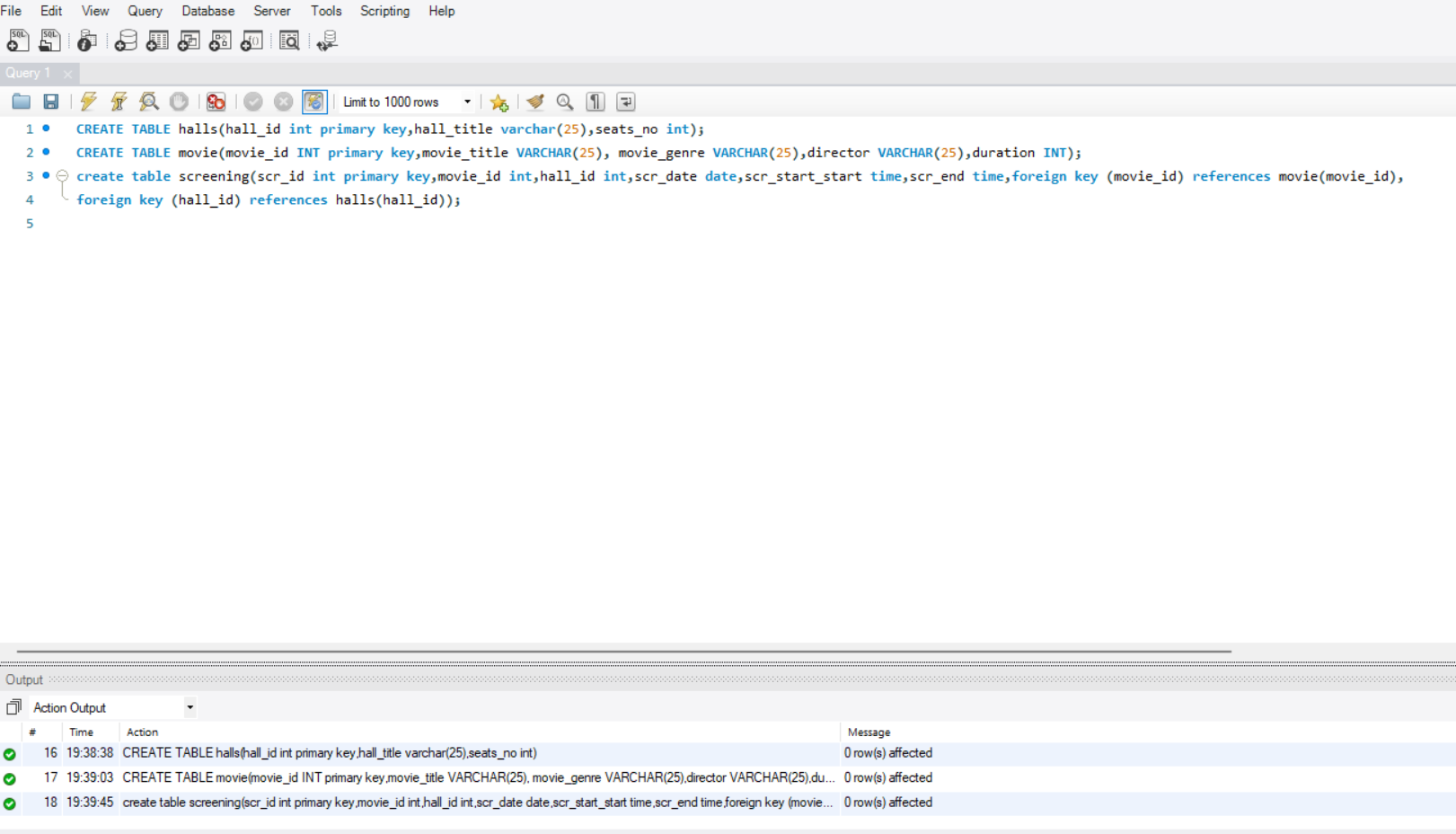
#### Syntax:

DROP TABLE table\_name;

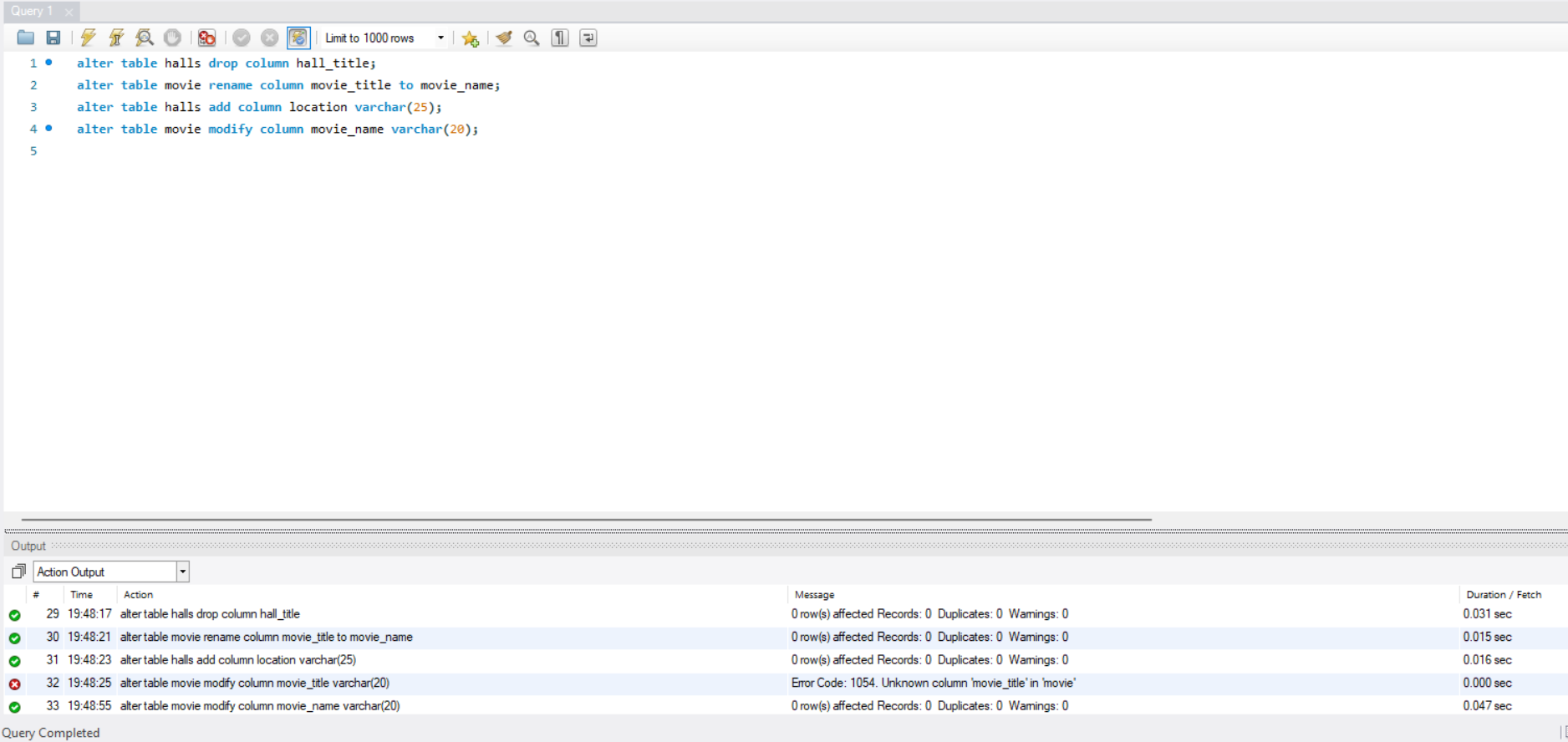
1. drop table screening;
2. drop table halls;
3. drop table movie;

**OUTPUT:**

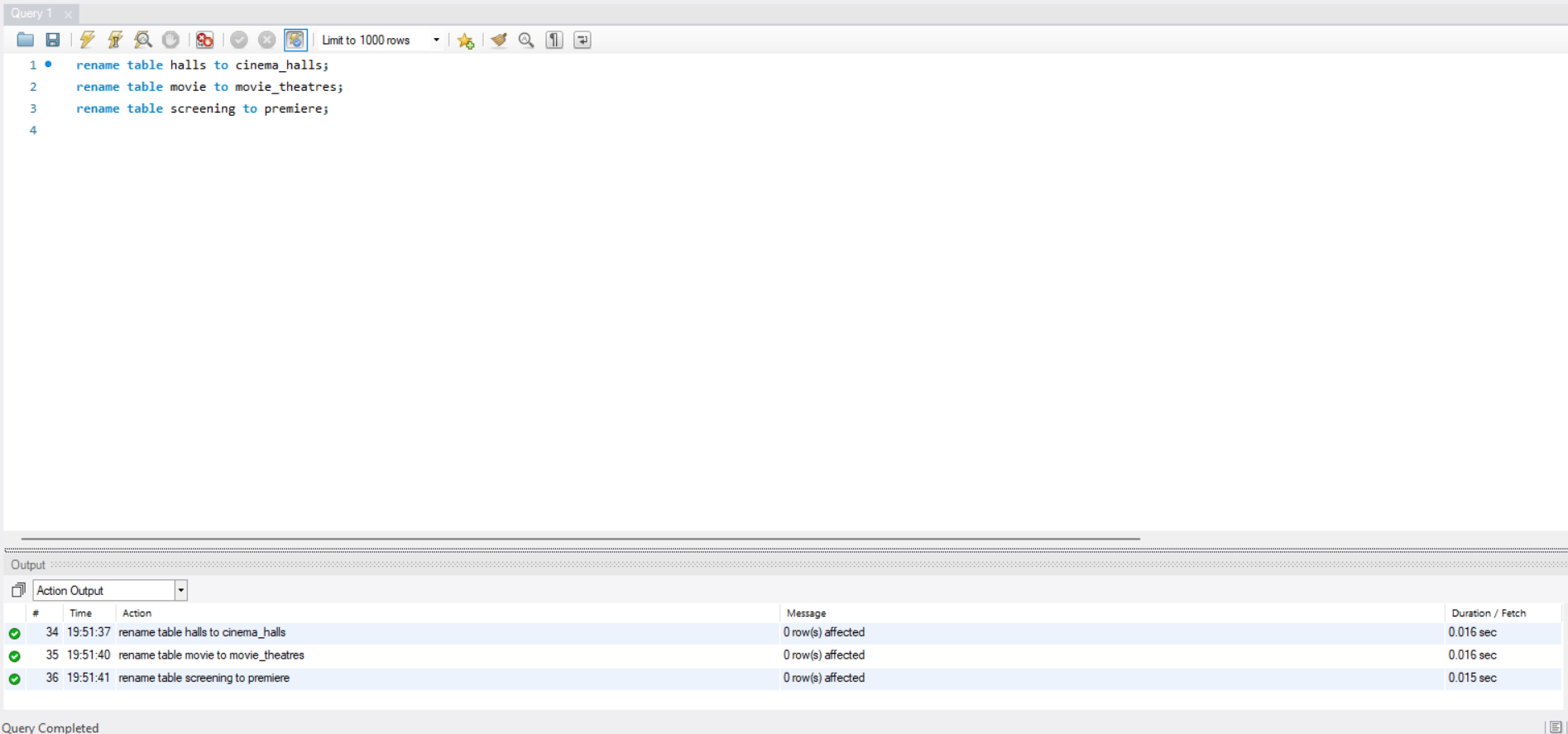
**CREATE**

****

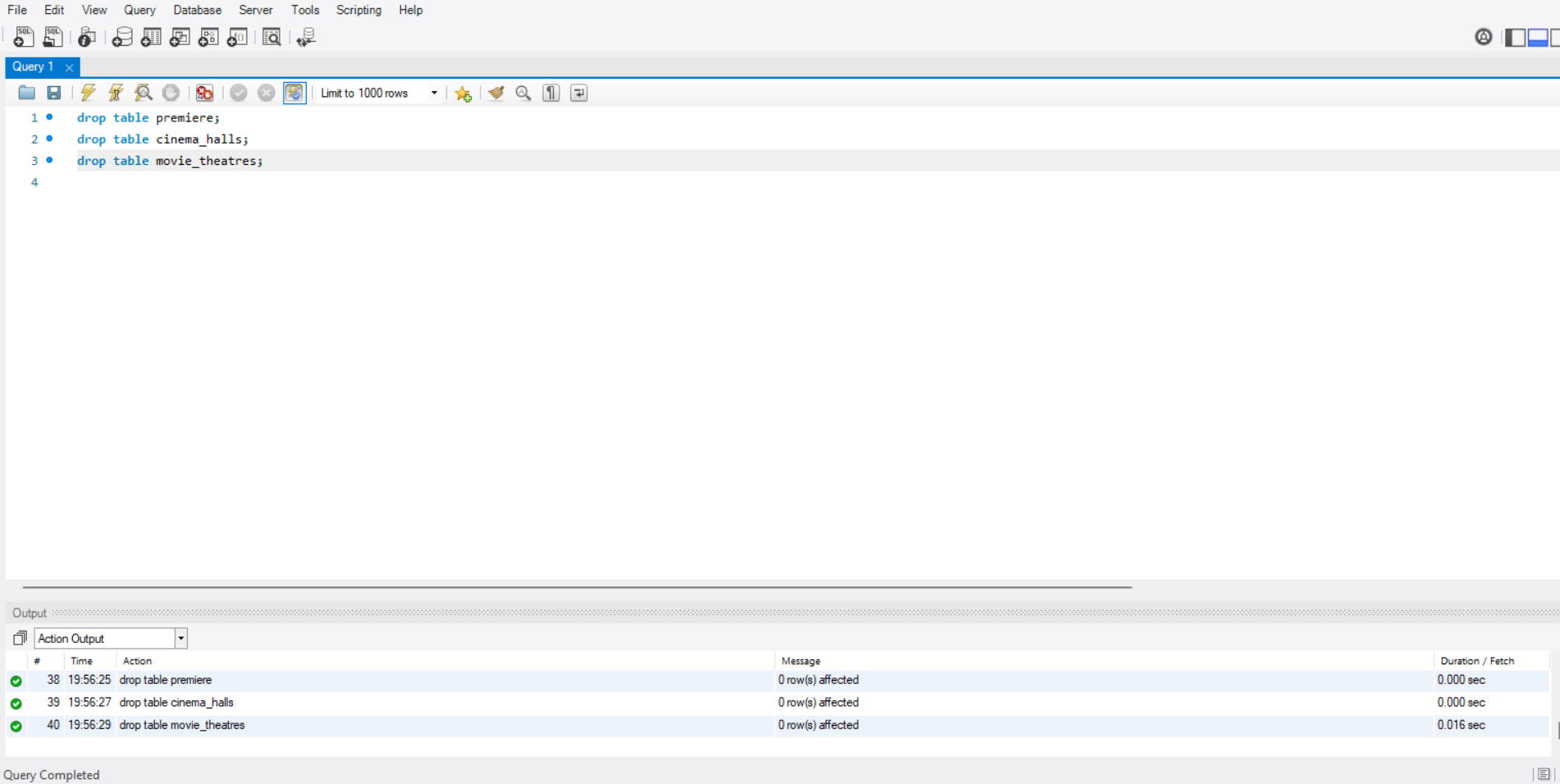
**ALTER**

****

**RENAME**

****

**DROP**



**RESULT:** The DDL commands have been executed successfully.

**EXPERIMENT 2**

**AIM:-** SQL data manipulation: write queries for each

* + Insert Command
  + Update Command
  + Delete Command
  + Select Command

**PROCEDURE:-**

1. Goto Start and click on programs.
2. Click on SQL Workbench.
3. Then a pop-up menu will appear to authenticate you as user.
4. Now SQL Workbench window will get opened for working with various SQL commands in it.

**THEORY:-**

**INSERT**

The INSERT INTO statement is used to insert new records in a table.

#### Syntax:

INSERT INTO table\_name (column1, column2, column3, ...)  
VALUES (value1, value2, value3, ...);

1. insert into halls values(1,'Audi 1',50);
2. insert into movie values(1,"QWERTY","Action","XYZ",2);
3. insert into screening values(1,1,1,"2023-03-26","19:30:00","20:45:30");

**UPDATE**

The UPDATE statement is used to modify the existing records in a table.

#### Syntax:

UPDATE table\_name SET column1 = value1, column2 = value2, ...  
WHERE condition;

1. update halls set hall\_title='Audi-1' where hall\_id=1;
2. update movie set movie\_title=”QWERTY-1” where movie\_id=1;
3. update screening set scr\_date=”2023-03-27” where scr\_id=1;

**SELECT**

The SELECT statement is used to select data from a database.The data returned is stored in a result table, called the result-set.

#### Syntax:

SELECT column1, column2, ...FROM table\_name;

1. select \* from halls where seats\_no=50;
2. select \* from movie where movie\_genre="Action";
3. select \* from screening where scr\_id=1;

**DELETE**

The DELETE statement is used to delete existing records in a table.

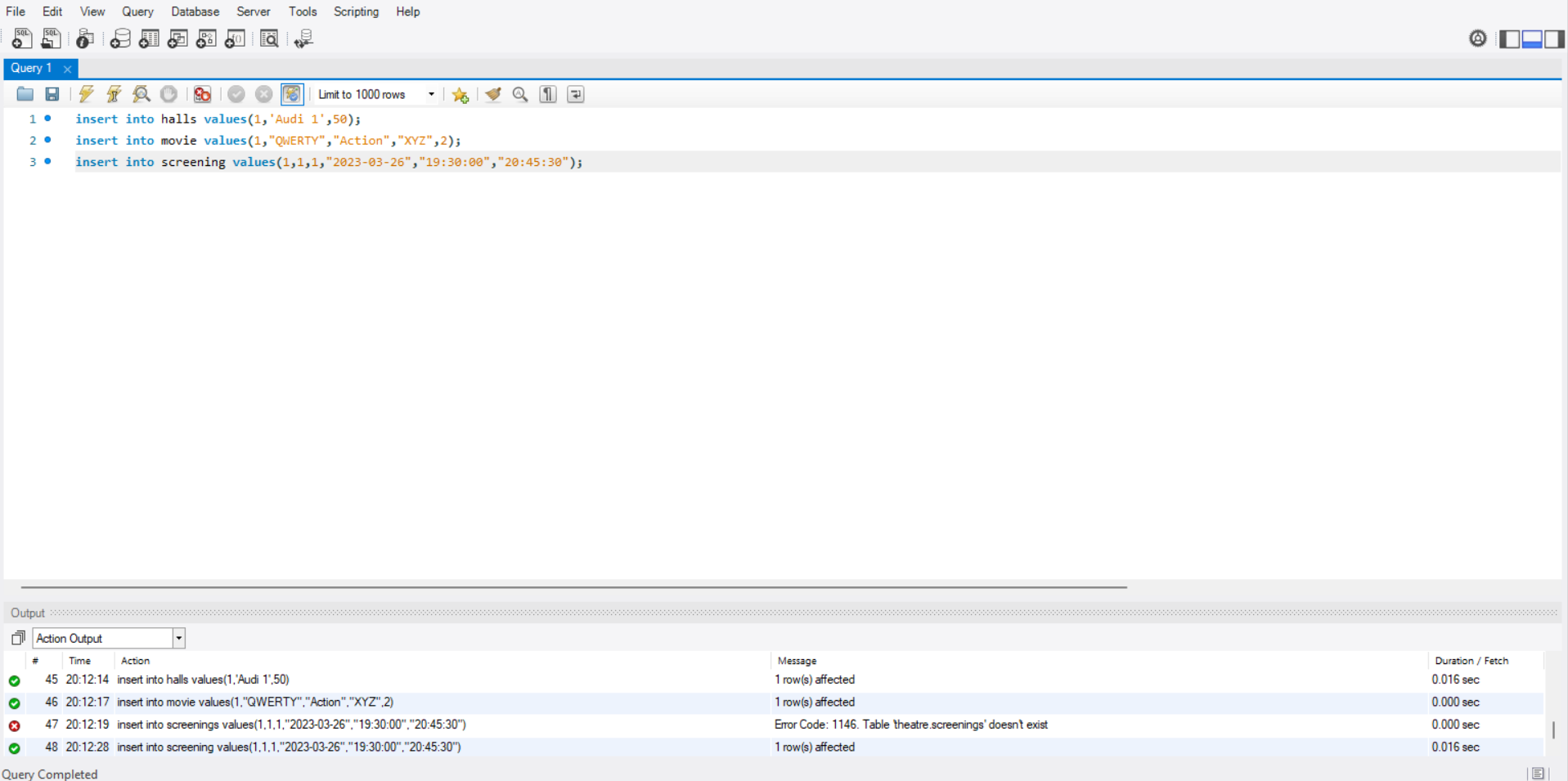
#### Syntax:

DELETE FROM table\_name WHERE condition;

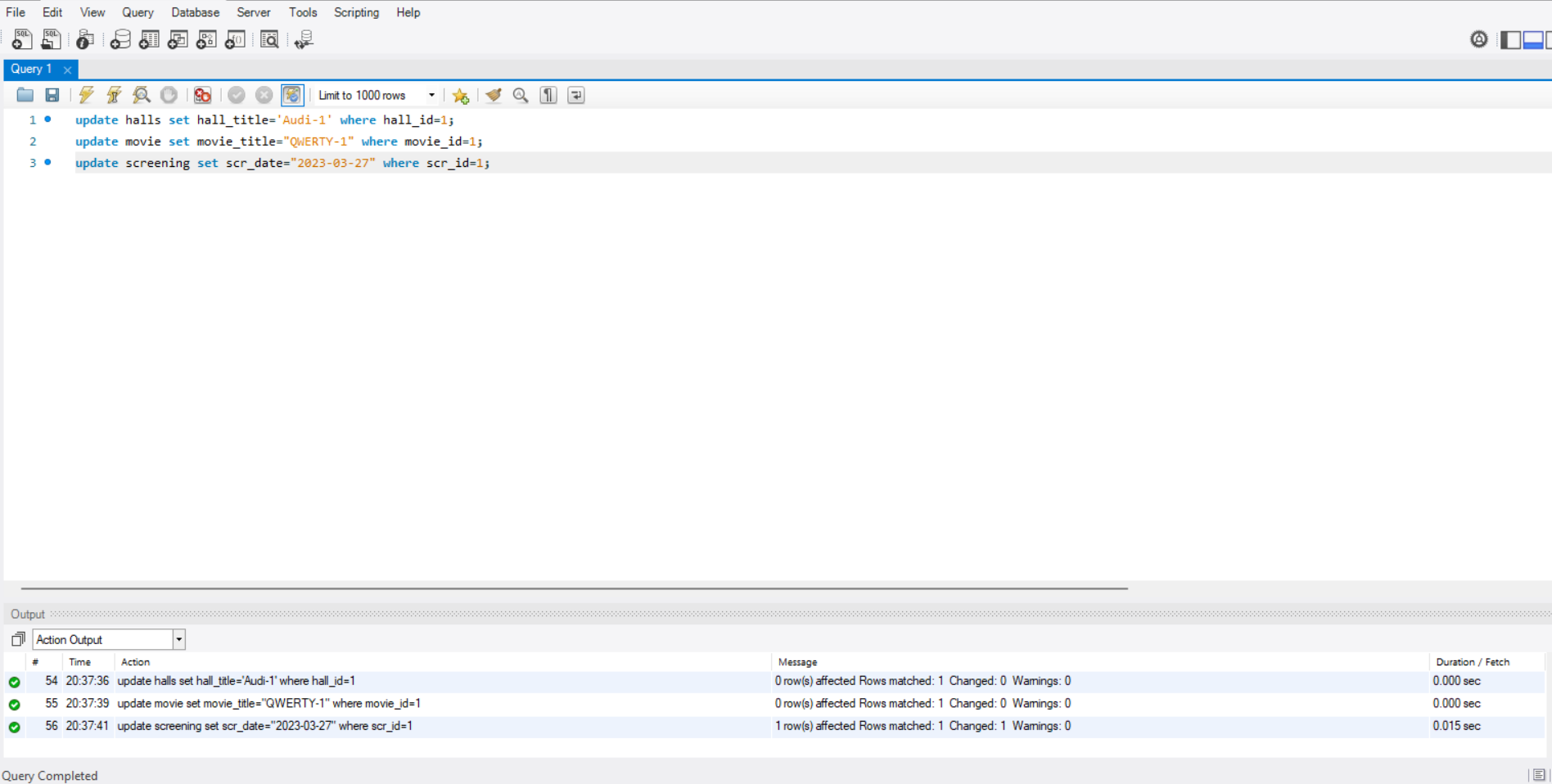
1. delete from halls where hall\_id=3;
2. delete from movie where movie\_genre="Action";
3. delete from screening where scr\_id=3;

**OUTPUT:**

**INSERT**

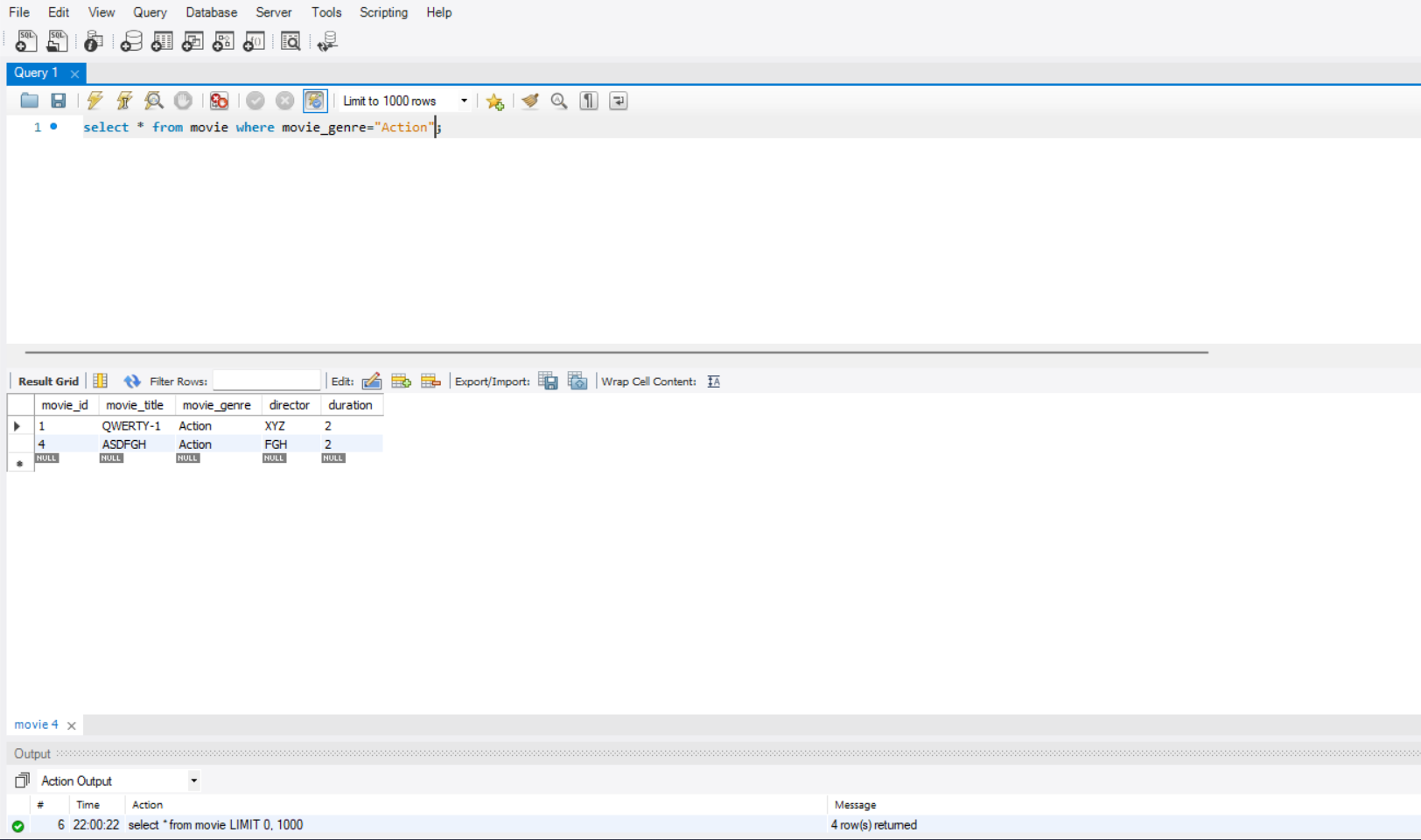


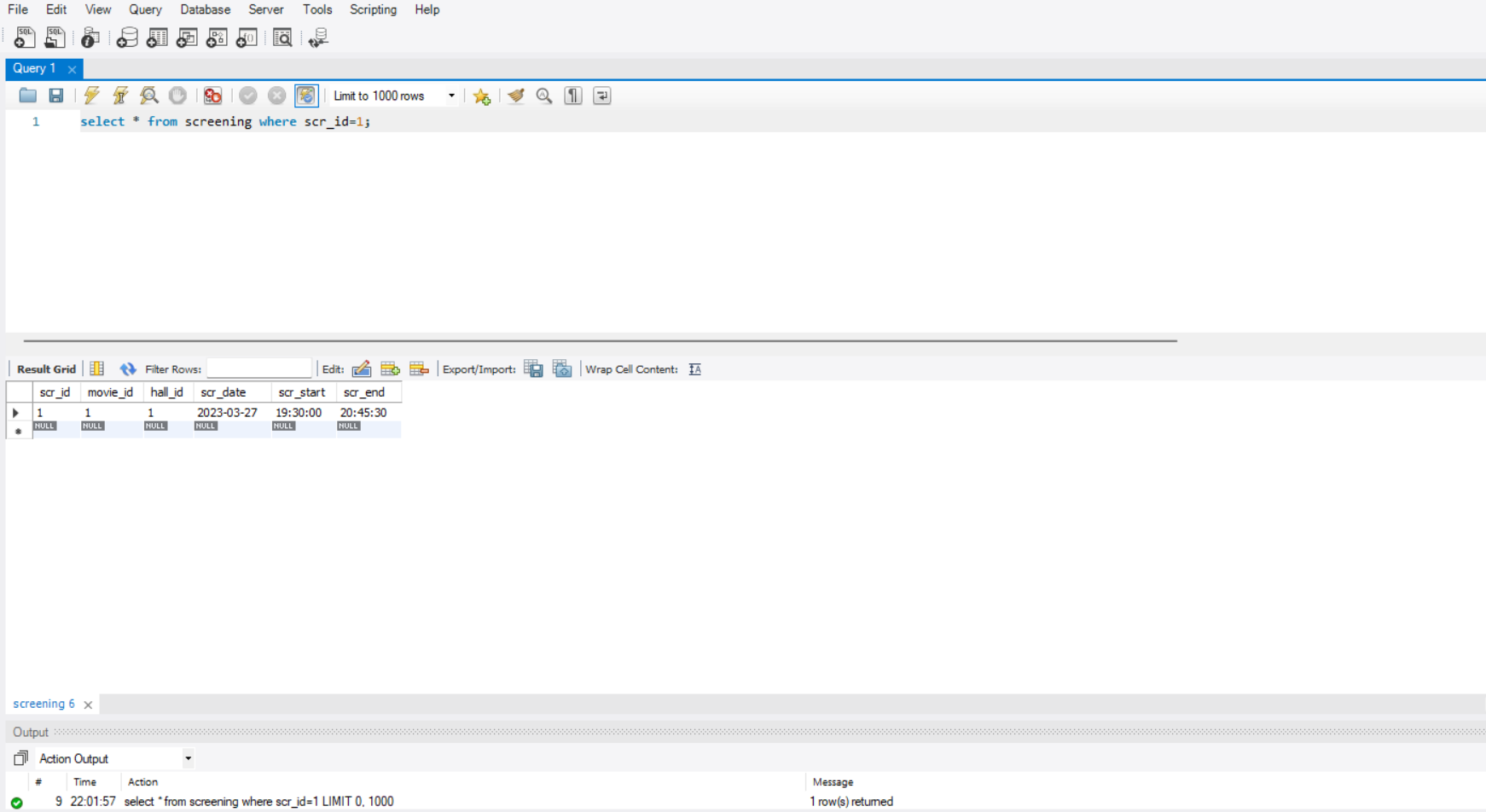
**UPDATE**

****

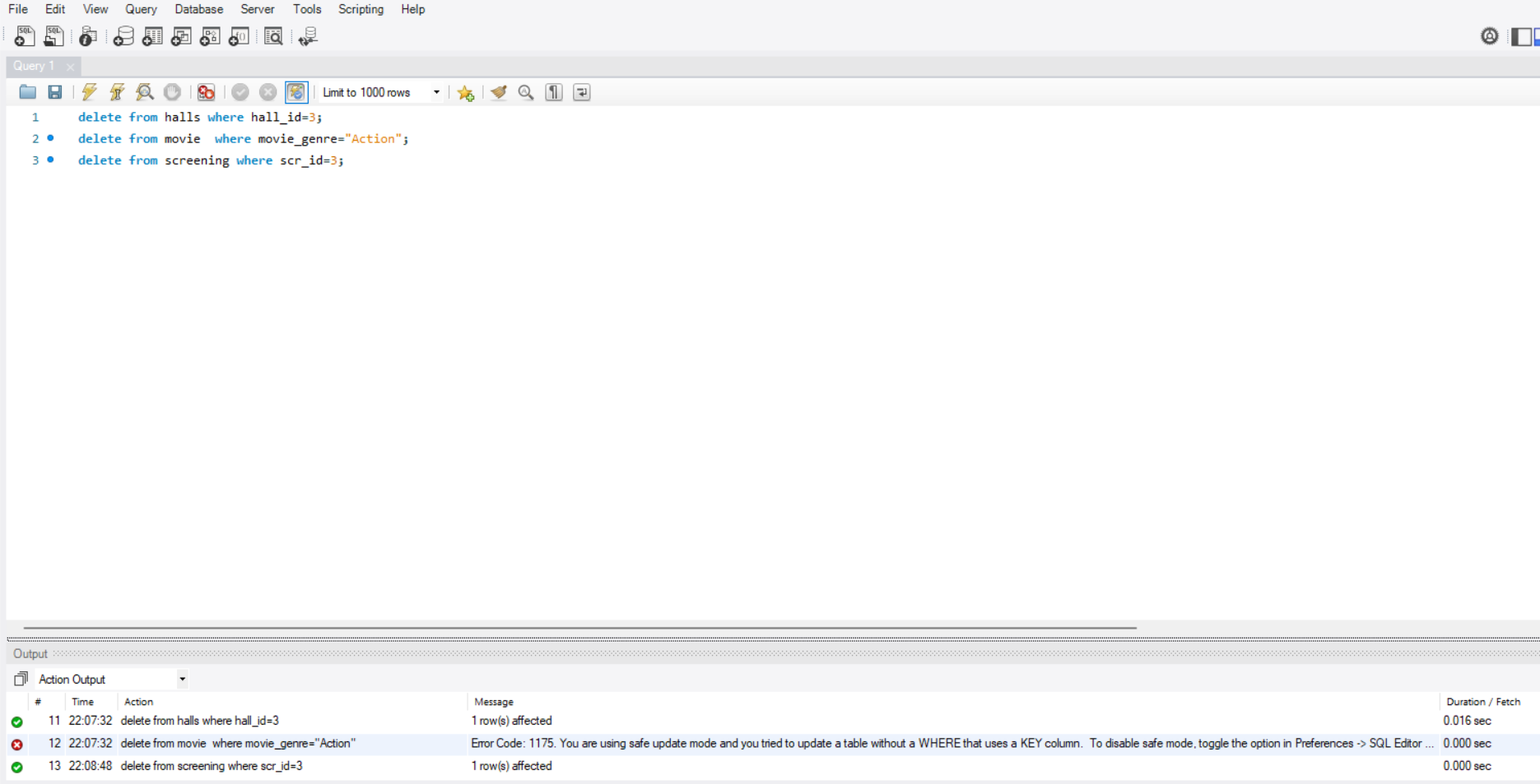
**SELECT**

****

****

****

**DELETE**

****

**RESULT:** The DML commands have been executed successfully.

**EXPERIMENT 3**

**AIM:** – SQL data control language and transaction control Language.

**PROCEDURE:-**

1. Goto Start and click on programs.
2. Click on SQL Workbench.
3. Then a pop-up menu will appear to authenticate you as user.
4. Now SQL Workbench window will get opened for working with various SQL commands in it.

**THEORY:-**

**DATA CONTROL LANGUAGES**

**Grant**

SQL Grant is used to provide permissions like Select, All, Execute to user on the database objects like Tables, Views, Databases and other objects in a SQL Server.

#### Syntax:

Grant privilageName on objectName To{userName/Public/roleName} [with Grant Option]

create user admin1 identified by password];

grant create session to admin]; grant connect to admin1;

**Revoke**

SQL Revoke is used to remove the permissions or privileges of a user on database objects set by the Grant command.

*Syntax :*

Revoke privilageName on objectName from{userName/public/roleName}

revoke create session from admin1;

revoke connect from admin!; revoke create table from admin 1;

**TRANSACTION CONTROL LANGUAGES**

**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.

*Syntax :* COMMIT;

grant connect to admin1,

revoke connect from admin1;

commit;

**Rollback**

This command is used to get the data or restore the data to the last savepoint or last committed state. If due to some reasons the data inserted, deleted or updated is not correct, you can rollback the data to a particular savepoint or if savepoint is not done, then to the last committed state.

*Syntax:*rollback;

update movie set movie\_genre="Comedy" where movie\_title="QWERTY-1";

select \* from movie;

rollback;

**Savepoint**

This command is used to save the data at a particular point temporarily, so that whenever needed can be rollback to that particular point.

*Syntax:*Savepoint A;

create table al(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);

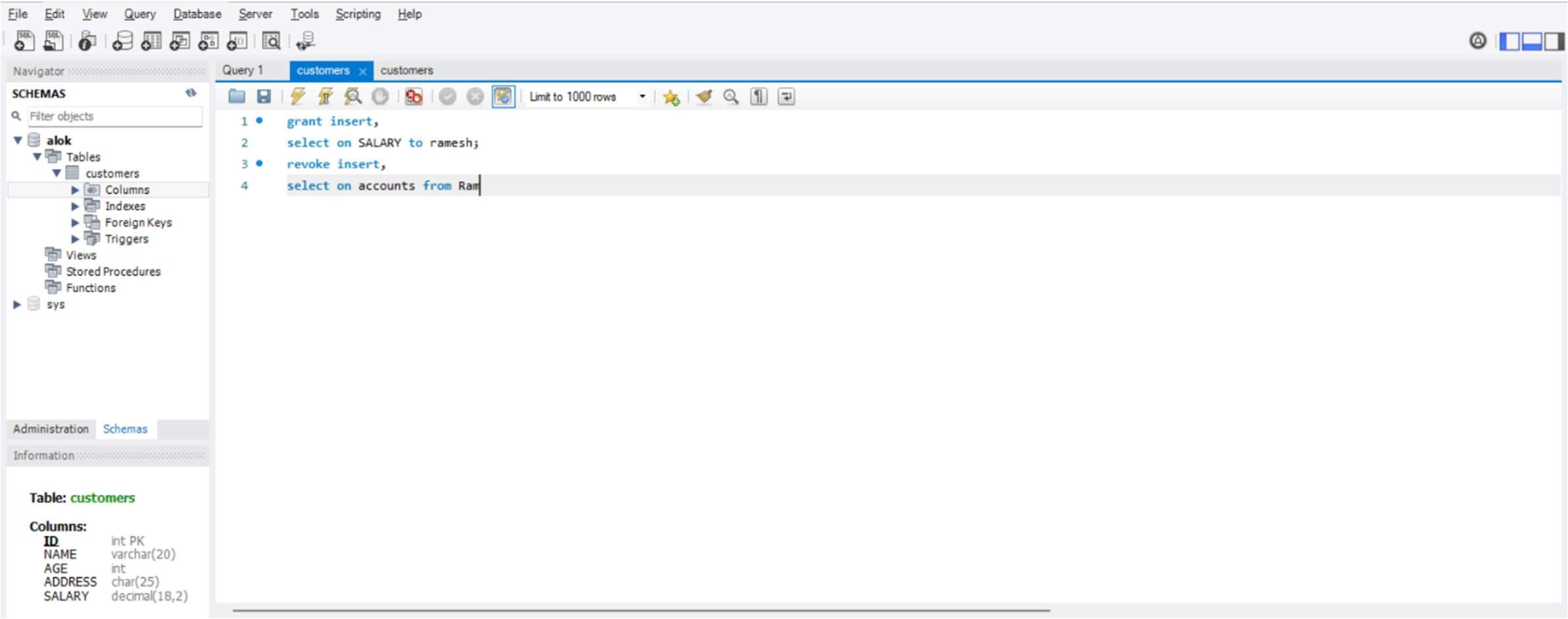
savepoint a1 table;

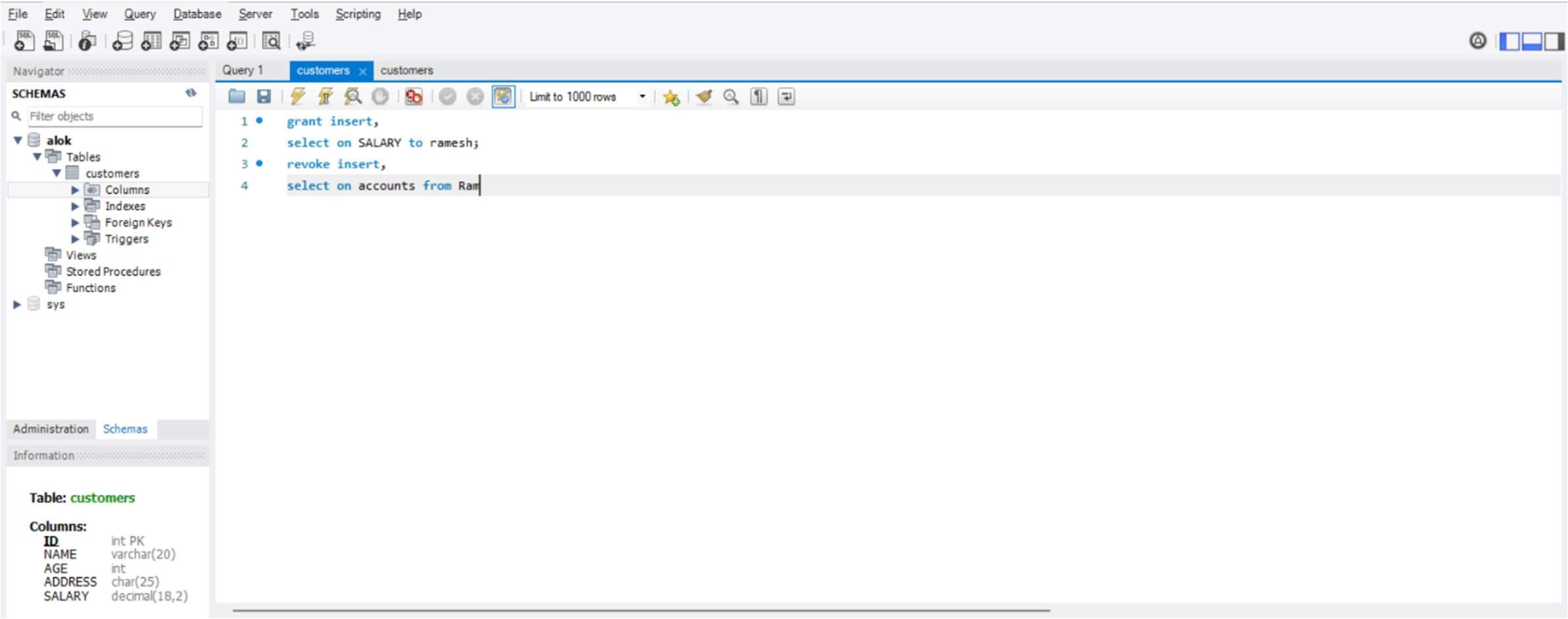
update a1 set a\_serialno=242 where a\_id=2;

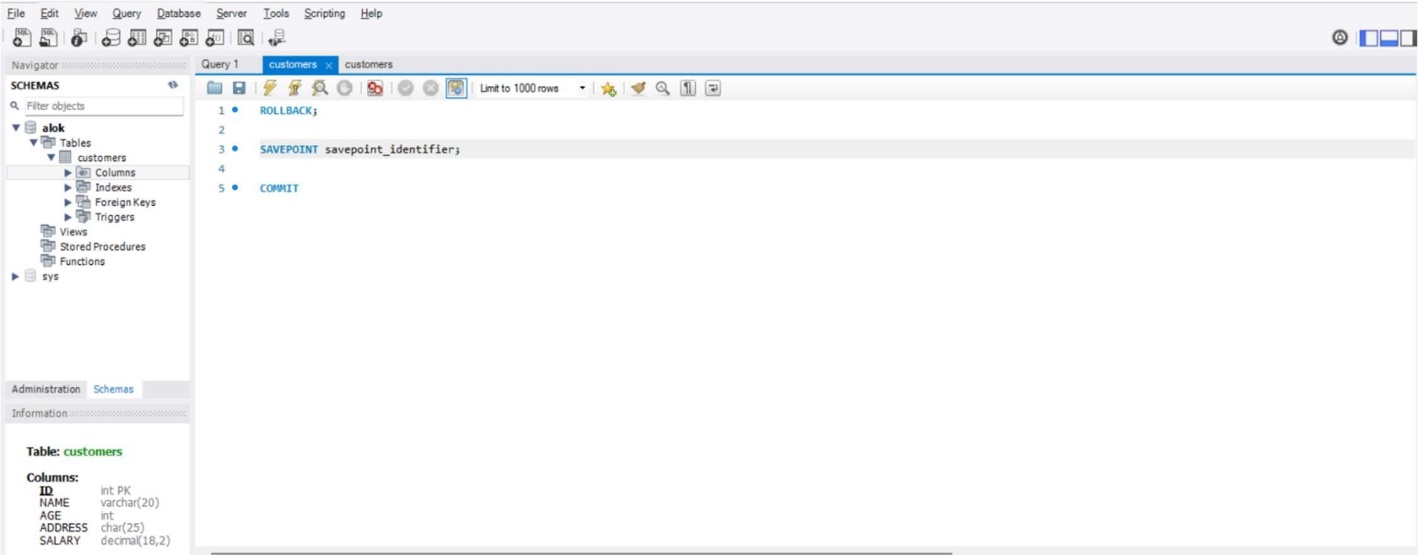
savepoint al\_update;

drop table al;

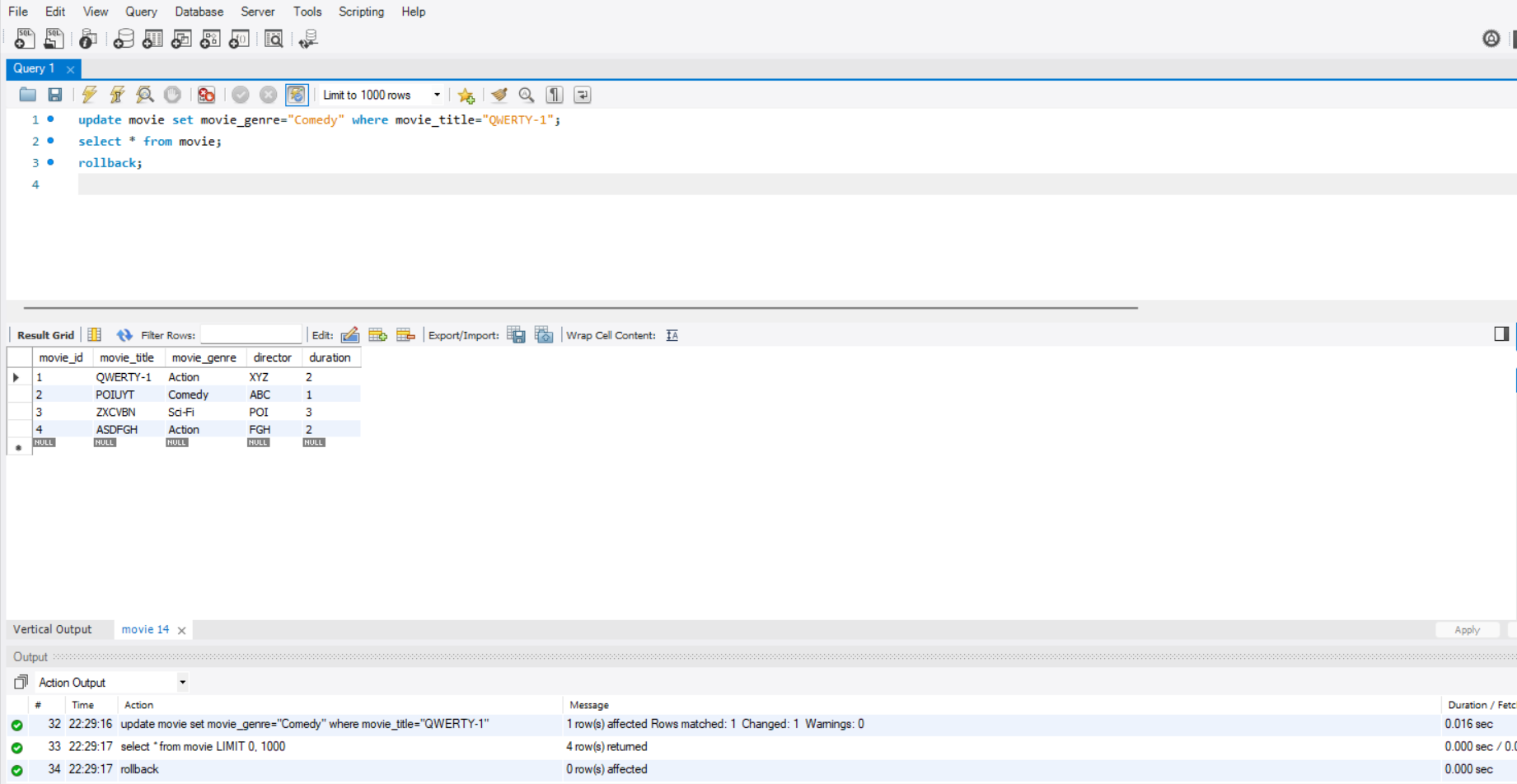
**OUTPUT**

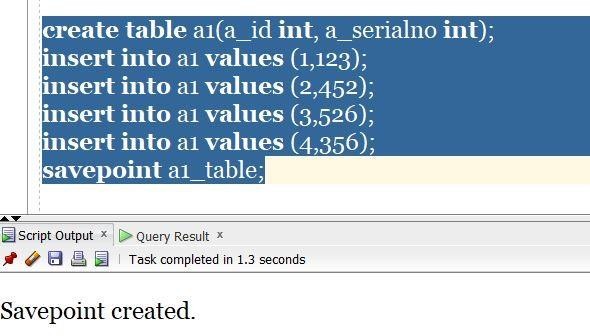
**GRANT**

**REVOKE**

**COMMIT**

**ROLLBACK**

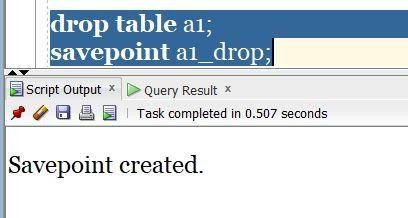
****

**SAVEPOINT**

**EXPERIMENT 4**

**AIM:**– Inbuilt functions in SQL:

1. NVL functions



**RESULT:** The DCL and TCL commands have been executed successfully.

**EXPERIMENT 4**

**AIM:** – Inbuilt functions in SQL.

1. NVL functions
2. Case function
3. String function
4. Date function
5. Aggregation function
6. Numeric function
7. Case manipulation function
8. Character manipulation function

**PROCEDURE:-**

1. Goto Start and click on programs.
2. Click on SQL Workbench.
3. Then a pop-up menu will appear to authenticate you as user.
4. Now SQL Workbench window will get opened for working with various SQL commands in it.

**THEORY:-**

**Case Function:-** The CASE expression goes through conditions and returns a value when the first condition is met (like an if-then-else statement). So, once a condition is true, it will stop reading and return the result. If no conditions are true, it returns the value in the ELSE clause. If there is no ELSE part and no conditions are true, it returns NULL.

*Syntax:-*

CASE  
    WHEN condition1 THEN result1  
    WHEN condition2 THEN resultN   
    ELSE result  
END;

• select Stu\_name, case S\_id when 1 then 'good' when 4 then 'bad'

else 'ok' end as student\_type from student;

• select Sub\_name, case Sub\_id when 255 then 'core' when 283 then

'CS' else 'E' end as subject\_type from subject;

• select Sub\_name, case Sub\_id when 255 then 'core' when 283 then

'CS' else 'E' end as subject\_type from subject;

**String Function:-**They are used to perform an operation on input string and return an output string.

*Syntax:-* SELECT char\_length('Hello!');

• select UCASE(name) as NAME from student;

• select id,name,CHARACTER\_LENGTH(name) as

character\_length,CONCAT(age," - ",gender) as Info from student;

• SELECT LOCATE("are","Hello sir! How are you?") as position;

• SELECT SUBSTRING("Welcome to my code",4,9) AS output;

**Date Function:-** The following table has a list of all the important Date and Time related functions available through SQL. There are various other functions supported by your RDBMS.

*Syntax:-*

• select current\_date;

• select current\_timestamp;

• select month("2022-10-20 09:45:21") as month;

• select monthname("2022-10-20 09:45:21") as month;

• select dayname("2022-10-20 09:45:21");

**Aggregation Function:-** 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.

*Syntax:-* COUNT( [ALL|DISTINCT] expression )

• select id,name,max(percentage) from student;

• select id,name,avg(percentage) as average,count(id) as total\_count from student;

• select sum(percentage) as total\_sum,min(percentage) as min\_percentage from

student;

**Numeric Function:-** Numeric Functions are used to perform operations on numbers and return numbers. Following are the numeric functions defined in SQL.

Syntax: SELECT ABS(-243.5);

• select id,name,percentage,(percentage+5) as new\_percentage from student;

• select floor(4.1234),ceil(4.3),round(4.8),abs(-6.5);

• select pow(2,2),sqrt(16),round(rand()\*100);

• select sin(3.141592653589793238);

**Case manipulation Function-** You use character-manipulation functions to manipulate character strings. The types of character-manipulation functions are CONCAT , SUBSTR , INSTR , LPAD , RPAD , TRIM , and REPLACE .

*Syntax:-*

• SELECT LOWER('SATYAM ARYA') from dual;

• SELECT UPPER('Tanishq') from dual;

• SELECT UPPER('I love cooking') from dual;

**Character manipulation Function:-** You use character-manipulation functions to manipulate character strings. The types of character-manipulation functions are CONCAT , SUBSTR , INSTR , LPAD , RPAD , TRIM , and REPLACE .

*Syntax:-*

• select instr('this is Tanishq','tan') as location;

• select concat(concat('tan','ishq'),' Raj') as full\_name;

• select trim(leading'0' from '00120341523413000') as trimmed;

**NVL Function:-** You can use the NVL function to convert an expression that evaluates to NULL to a value that you specify. The NVL function accepts two arguments: the first argument takes the name of the expression to be evaluated; the second argument specifies the value that the function returns when the first argument evaluates to NULL.

*Syntax:-*

 • SELECT IFNULL('John', 'N/A');

• SELECT IFNULL(NULL, 'N/A');

**OUTPUT**

**Case Function**

A screenshot of a computer screen

Description automatically generated with medium confidence

**String Function**

Text

Description automatically generated

**Date Function**

Text

Description automatically generated

**Aggregation Function**

Text

Description automatically generated

**Numeric Function**

Text

Description automatically generated

**Case Manipulation Function**

Text

Description automatically generated

**Character manipulation Function**

Text

Description automatically generated

**NVL Function**

Text

Description automatically generated

**RESULT:** The inbuilt functions has been successfully executed.

**EXPERIMENT 5**

**AIM:** Construct a E-R model for the application to be constructed to a Database.

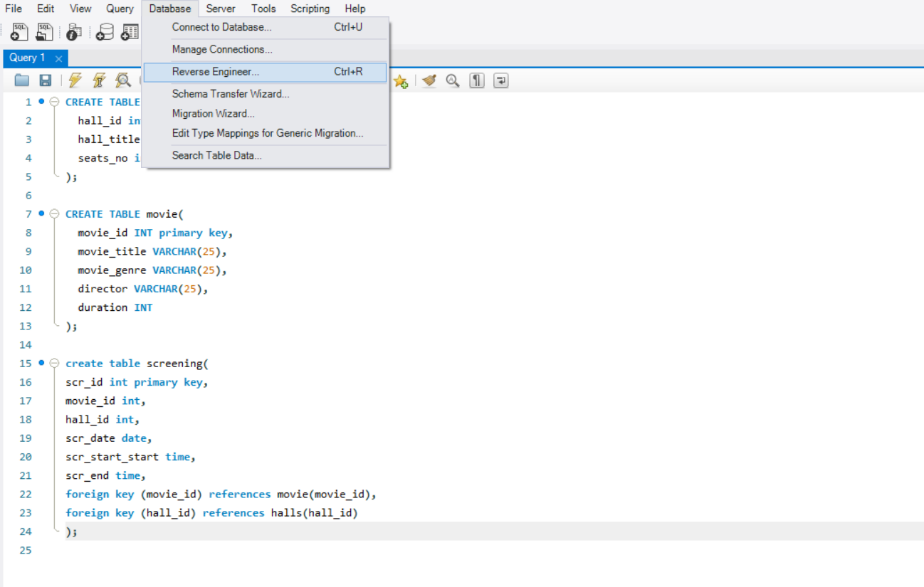
**PROCEDURE:-**

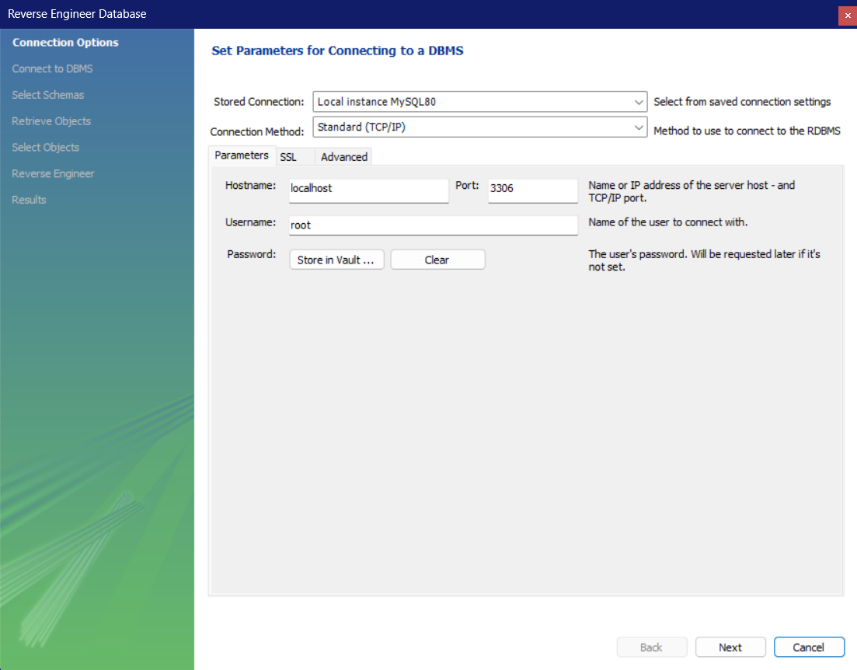
1. To construct ER-Model go to menu **Database**, choose **Reverse Engineer...** option.
2. Provide connection details to your database and click **Next**. Wait for the connection and click **Next** again.
3. When successfully connected wizard will show you list of available schemas on the server. Select the ones you want to reverse engineer.
4. Wait for the schemas being read and continue with **Next**. On next screen you have an option to **select object types** and **filter specific objects**. Let's ignore it and import all objects. Click **Execute >**.
5. Wait for reverse engineering to take place and when done continue with **Next**. Final screen shows you a summary of the import. Close with **Finish**.
6. When the process ends with success you get a new model (see above for more details) with default ER diagram with all the tables and views from all imported schemas.

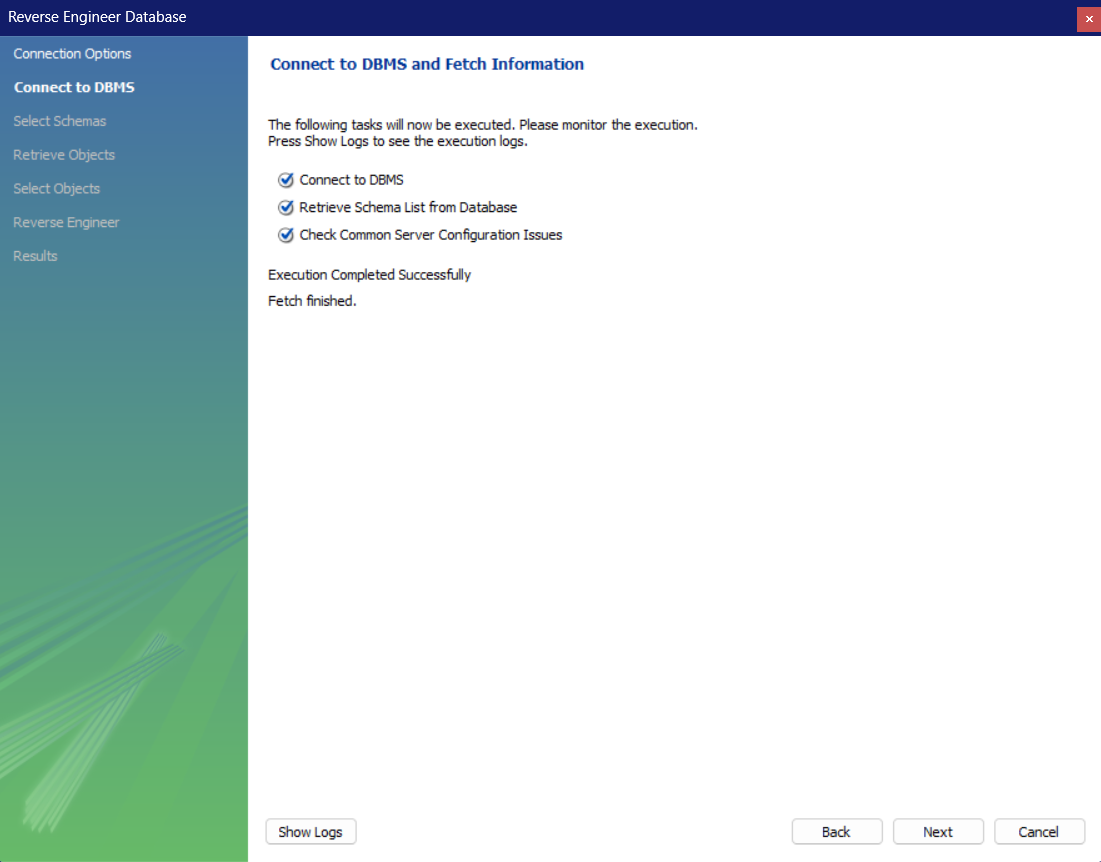
**THEORY:-**

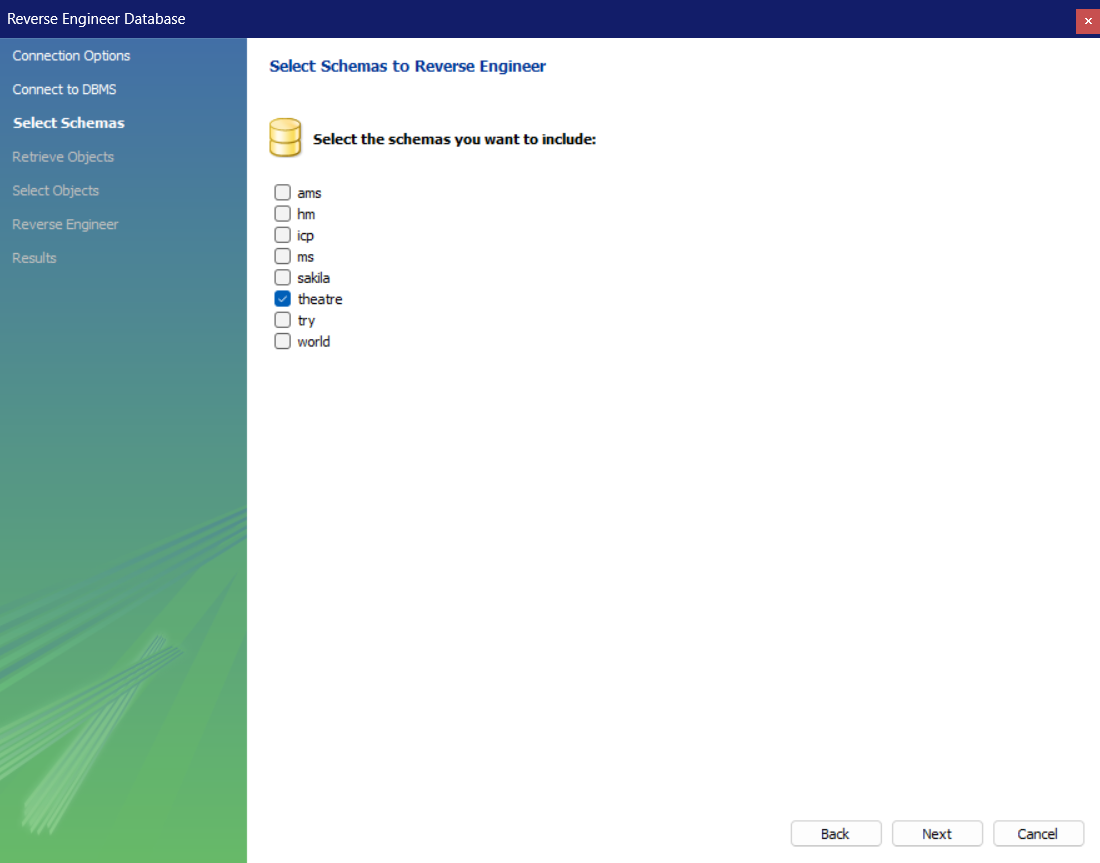
ER diagrams are a useful tool for understanding and modeling and design of databases.We normally build a physical database from the ER diagram – this is called **forward engineering**. But there are times when you want to do the opposite – create diagram from existing database schema. This task is called **reverse engineering**, as you first have working tool and you create model from it.

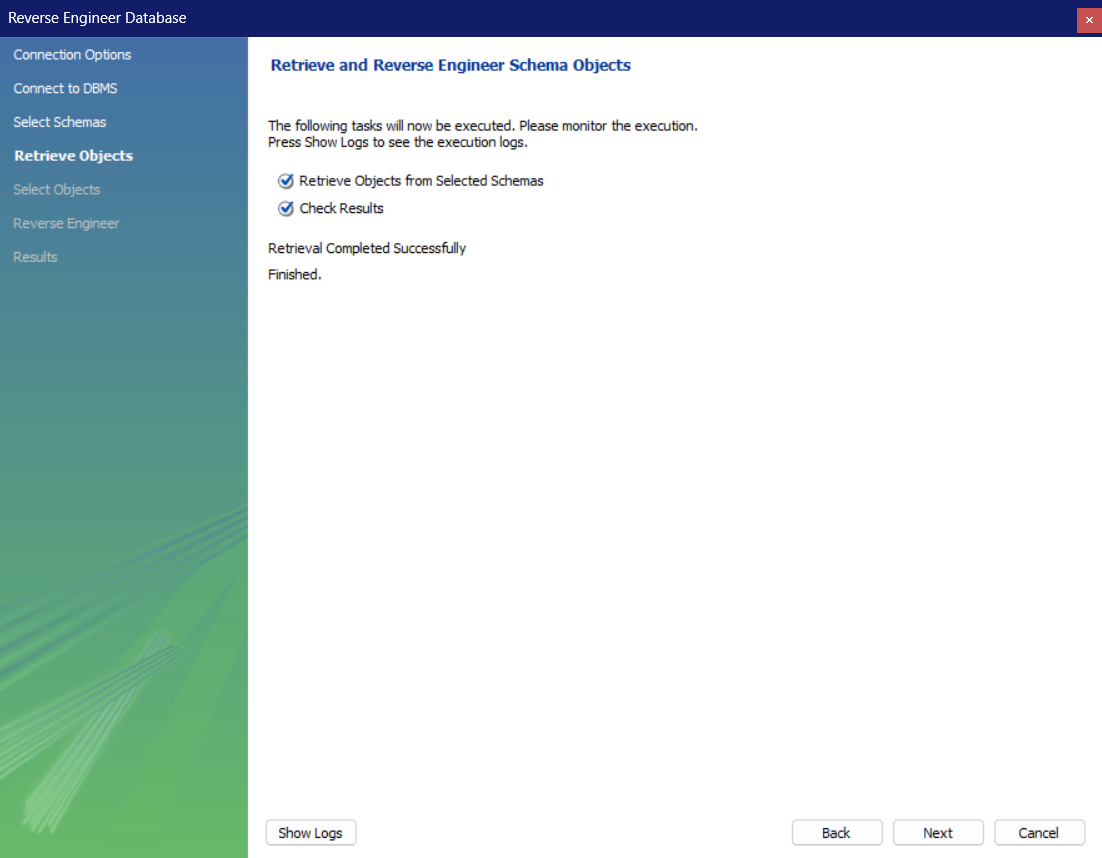
**OUTPUT:-**

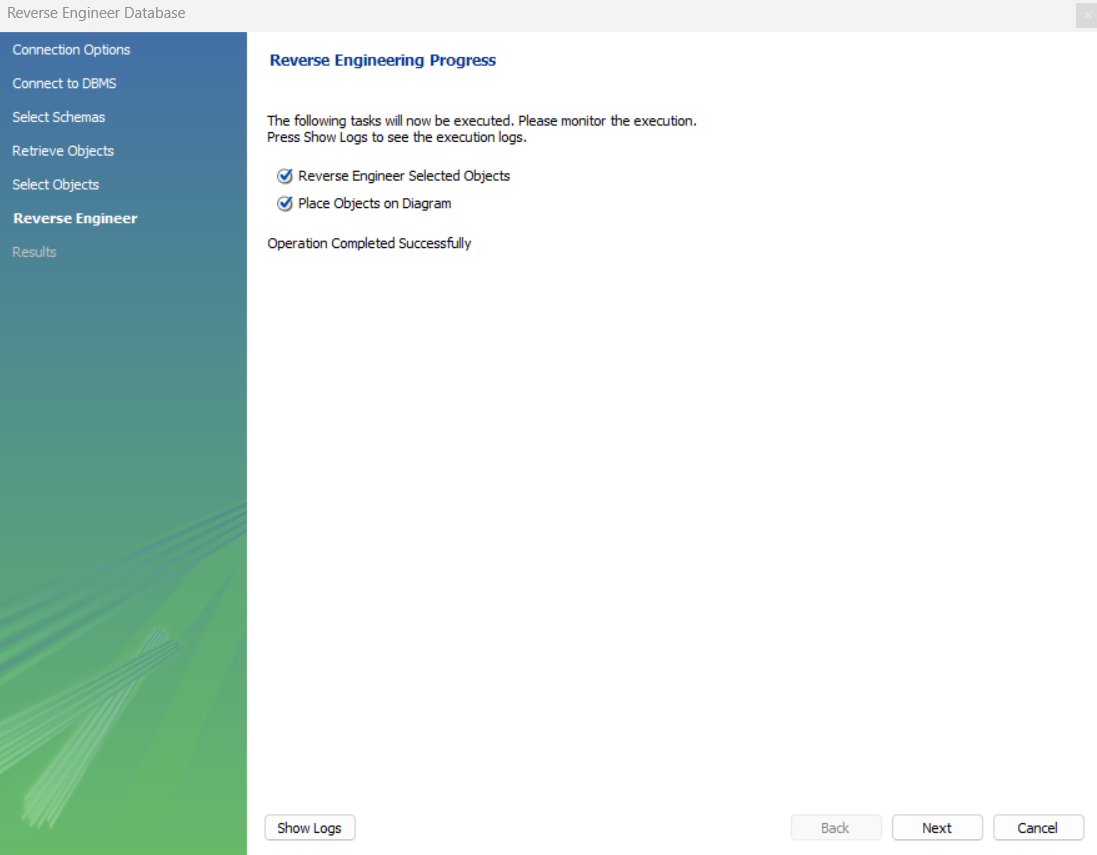
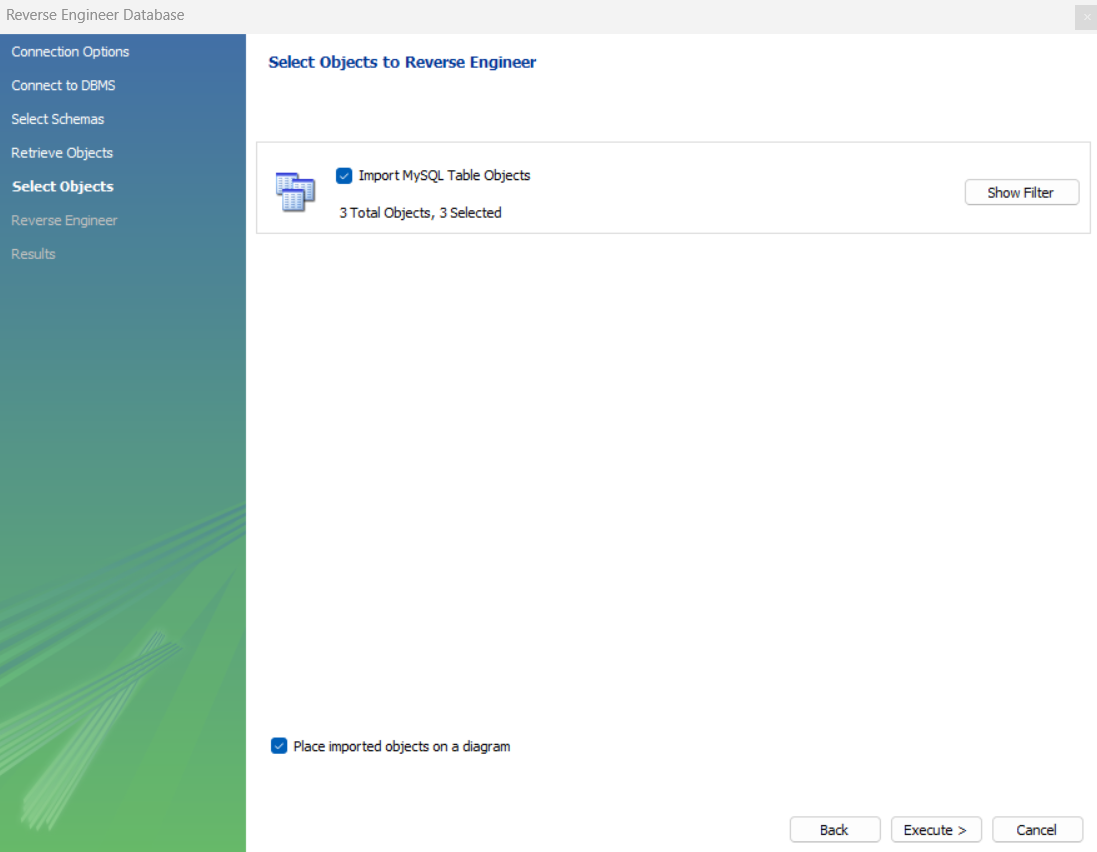


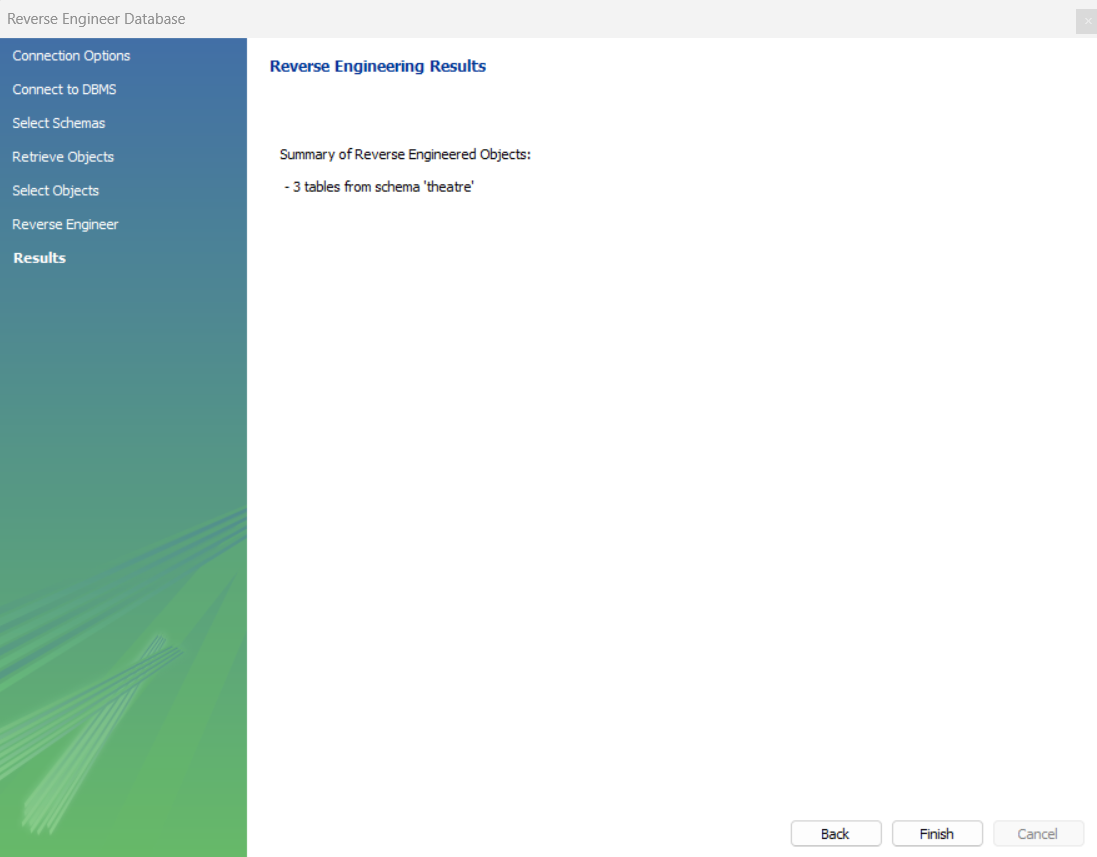


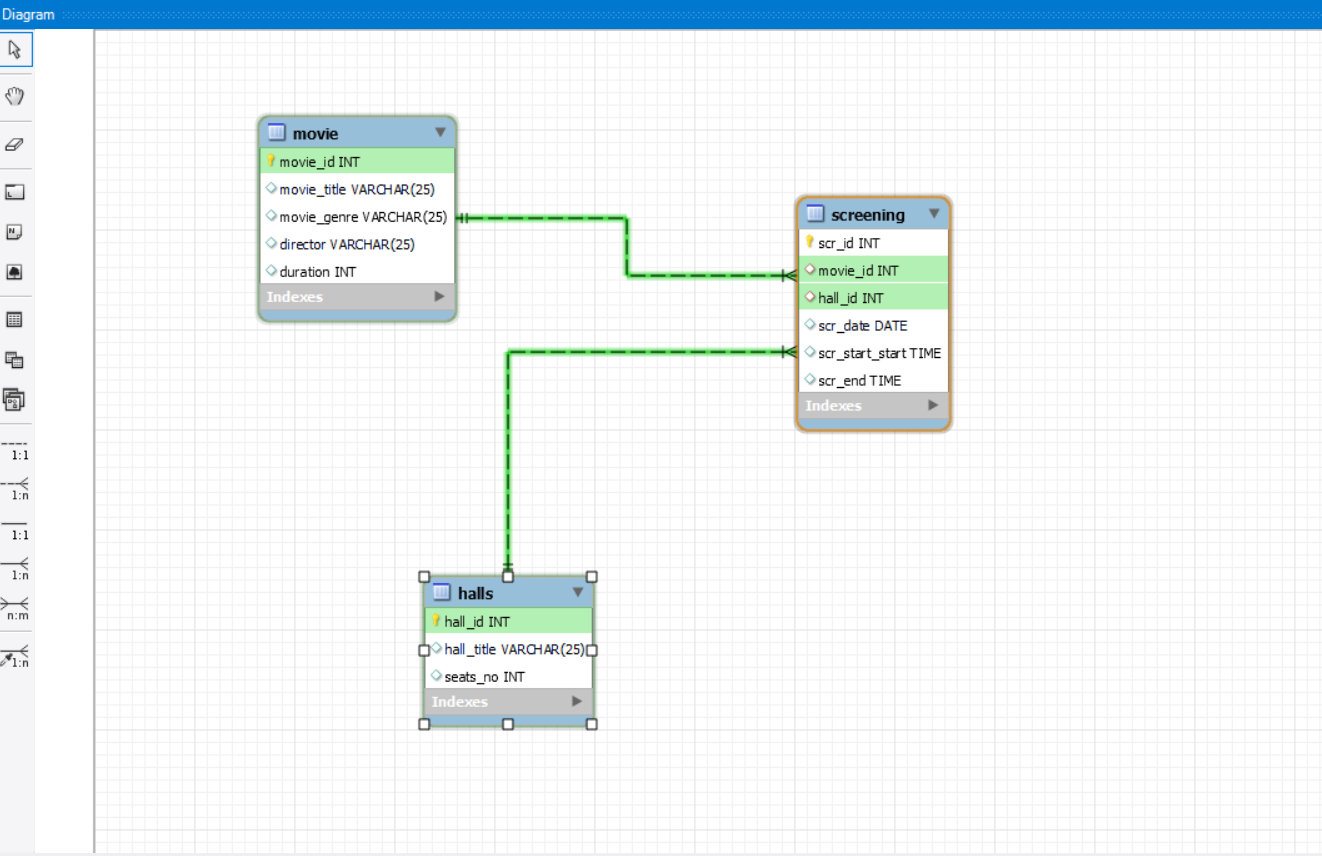












**RESULT:** ER-Diagram using the reverse engineering has been constructed successfully.

**EXPERIMENT-6**

**AIM:** Construction of Relational Table from the ER Diagram.

**PROCEDURE:-**

1. Select the model that you wish to forward engineer and then choose the Database, Forward Engineer menu item from the main menu.

2. The first step of the process is to connect to a MySQL server to create the new database schema. As the following figure shows, this page enables you to use a previously stored connection or to enter the connection parameters.

3. Click Next after setting the connection parameters. The next page of the wizard displays Catalog Validation, as the following figure shows. Validation is available only in the Commercial Edition.

4. Click Run Validations to validate the catalog and Click Next to continue.

5. The next page enables you to set options for the database to be created (see the following figure). These options are as described in Section 9.4.1.1.1, “Creating a Schema”.

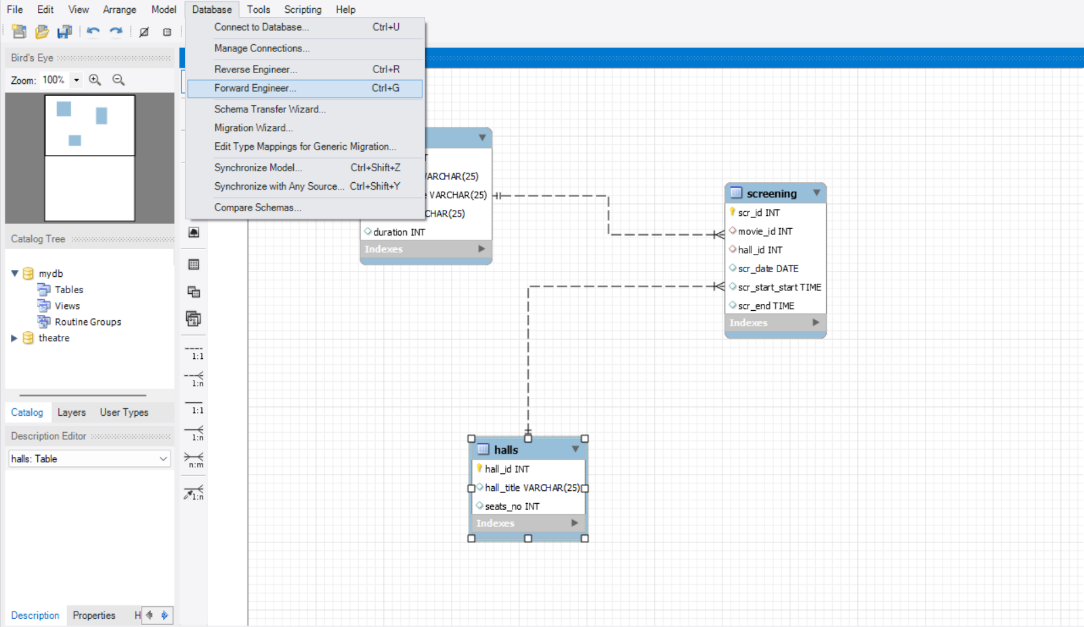
6. As the following figure shows, the Select Objects page enables you to select the objects to forward engineer: Table objects, view objects, routine objects, trigger objects, and user objects.

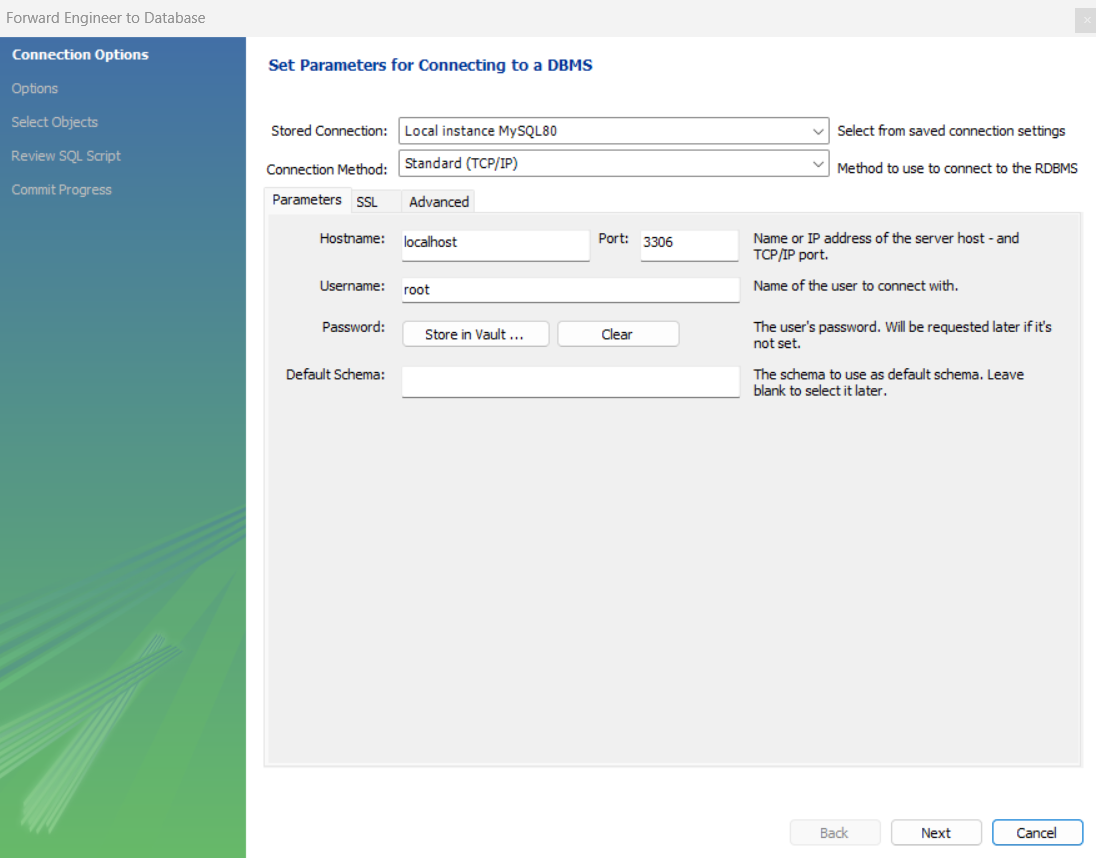
7. To select a subset of objects to forward engineer, use the Show Filter/Hide Filter button, then select specific objects. After you have selected your objects, click Next to continue.

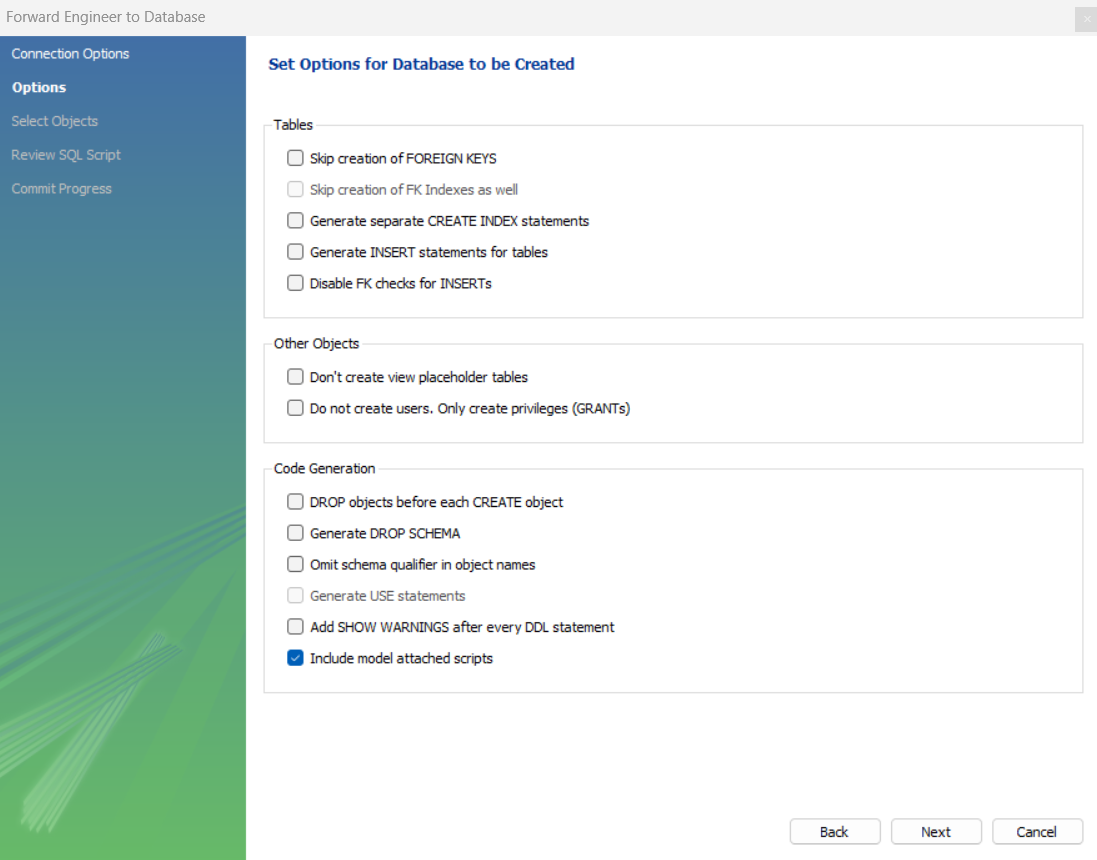
8. Click Next to continue if you are satisfied with the generated script and the next page of the wizard displays the results of the forward engineering process.

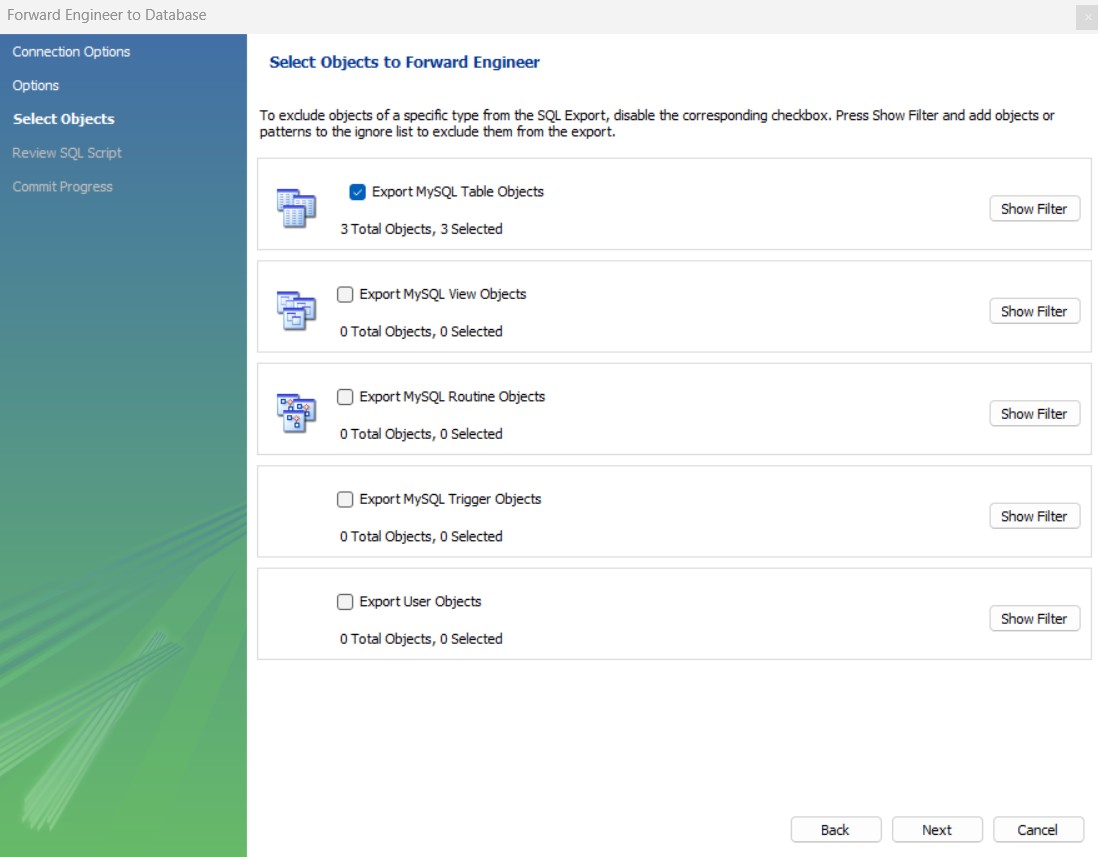
**THEORY:-** Use forward engineering to export your schema design to a MySQL server.Select the model that you wish to forward engineer and then choose the Database, Forward Engineer menu item from the main menu.

**OUTPUT:-**









-- MySQL Workbench Forward Engineering

SET @OLD\_UNIQUE\_CHECKS=@@UNIQUE\_CHECKS, UNIQUE\_CHECKS=0;

SET @OLD\_FOREIGN\_KEY\_CHECKS=@@FOREIGN\_KEY\_CHECKS, FOREIGN\_KEY\_CHECKS=0;

SET @OLD\_SQL\_MODE=@@SQL\_MODE, SQL\_MODE='ONLY\_FULL\_GROUP\_BY,STRICT\_TRANS\_TABLES,NO\_ZERO\_IN\_DATE,NO\_ZERO\_DATE,ERROR\_FOR\_DIVISION\_BY\_ZERO,NO\_ENGINE\_SUBSTITUTION';

-- -----------------------------------------------------

-- Schema mydb

-- -----------------------------------------------------

-- -----------------------------------------------------

-- Schema theatre

-- -----------------------------------------------------

-- -----------------------------------------------------

-- Schema theatre

-- -----------------------------------------------------

CREATE SCHEMA IF NOT EXISTS `theatre` DEFAULT CHARACTER SET utf8mb4 COLLATE utf8mb4\_0900\_ai\_ci ;

USE `theatre` ;

-- -----------------------------------------------------

-- Table `theatre`.`halls`

-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS `theatre`.`halls` (

`hall\_id` INT NOT NULL,

`hall\_title` VARCHAR(25) NULL DEFAULT NULL,

`seats\_no` INT NULL DEFAULT NULL,

PRIMARY KEY (`hall\_id`))

ENGINE = InnoDB

DEFAULT CHARACTER SET = utf8mb4

COLLATE = utf8mb4\_0900\_ai\_ci;

-- -----------------------------------------------------

-- Table `theatre`.`movie`

-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS `theatre`.`movie` (

`movie\_id` INT NOT NULL,

`movie\_title` VARCHAR(25) NULL DEFAULT NULL,

`movie\_genre` VARCHAR(25) NULL DEFAULT NULL,

`director` VARCHAR(25) NULL DEFAULT NULL,

`duration` INT NULL DEFAULT NULL,

PRIMARY KEY (`movie\_id`))

ENGINE = InnoDB

DEFAULT CHARACTER SET = utf8mb4

COLLATE = utf8mb4\_0900\_ai\_ci;

-- -----------------------------------------------------

-- Table `theatre`.`screening`

-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS `theatre`.`screening` (

`scr\_id` INT NOT NULL,

`movie\_id` INT NULL DEFAULT NULL,

`hall\_id` INT NULL DEFAULT NULL,

`scr\_date` DATE NULL DEFAULT NULL,

`scr\_start` TIME NULL DEFAULT NULL,

`scr\_end` TIME NULL DEFAULT NULL,

PRIMARY KEY (`scr\_id`),

INDEX `fk1` (`movie\_id` ASC) VISIBLE,

INDEX `fk2` (`hall\_id` ASC) VISIBLE,

CONSTRAINT `screening\_ibfk\_1`

FOREIGN KEY (`movie\_id`)

REFERENCES `theatre`.`movie` (`movie\_id`),

CONSTRAINT `screening\_ibfk\_2`

FOREIGN KEY (`hall\_id`)

REFERENCES `theatre`.`halls` (`hall\_id`))

ENGINE = InnoDB

DEFAULT CHARACTER SET = utf8mb4

COLLATE = utf8mb4\_0900\_ai\_ci;

SET SQL\_MODE=@OLD\_SQL\_MODE;

SET FOREIGN\_KEY\_CHECKS=@OLD\_FOREIGN\_KEY\_CHECKS;

SET UNIQUE\_CHECKS=@OLD\_UNIQUE\_CHECKS;

**RESULT:** Construction of Relational Table from the ER Diagram using the forward engineering has been executed successfully.

**EXPERIMENT-7**

**AIM:-**To Execute Joins in MySQL

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

**PROCEDURE:-**

1. Goto Start and click on programs.
2. Click on SQL Workbench.
3. Then a pop-up menu will appear to authenticate you as user.
4. Now SQL Workbench window will get opened for working with various SQL commands in it.

**THEORY:-**

MySQL INNER JOIN (SIMPLE JOIN): - The INNER JOIN keyword selects records that have matching values in both tables.

### *Syntax:-*

SELECT column\_name(s)  
FROM table1  
INNER JOIN table2ON table1.column\_name = table2.column\_name;

* 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;

## MySQL LEFT OUTER JOIN:- The LEFT JOIN keyword returns all records from the left table (table1), and the matching records (if any) from the right table (table2).

## *Syntax:-*

## SELECT column\_name(s)

## FROM table1

## LEFT JOIN table2

## ON table1.column\_name = table2.column\_name;

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

## MySQL RIGHT JOIN :- The RIGHT JOIN keyword returns all records from the right table (table2), and the matching records (if any) from the left table (table1).

## *Syntax:-*

## SELECT column\_name(s)

## FROM table1

## RIGHT JOIN table2

## ON table1.column\_name = table2.column\_name;

* SELECT \* FROM personal LEFT JOIN city ON personal . city—city. cid UNION SELECT \* FROM persona RIGHT JOIN city ON personal . city=city. Cid;

## MySQL Full Outer JOIN :- The FULL JOIN or FULL OUTER JOIN keyword is used to select all records from the left table and right table. It combines both tables into a result-set and returns it to the user. Note that MySQL FULL JOIN is known to create large datasets.

*Syntax :-*

SELECT expression FROM

table1 [t1] FULL JOIN table2 [t2]

ON table1.column\_name = table2.column\_name;

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

## MySQL Self Join:- A self join is a regular join, but the table is joined with itself.

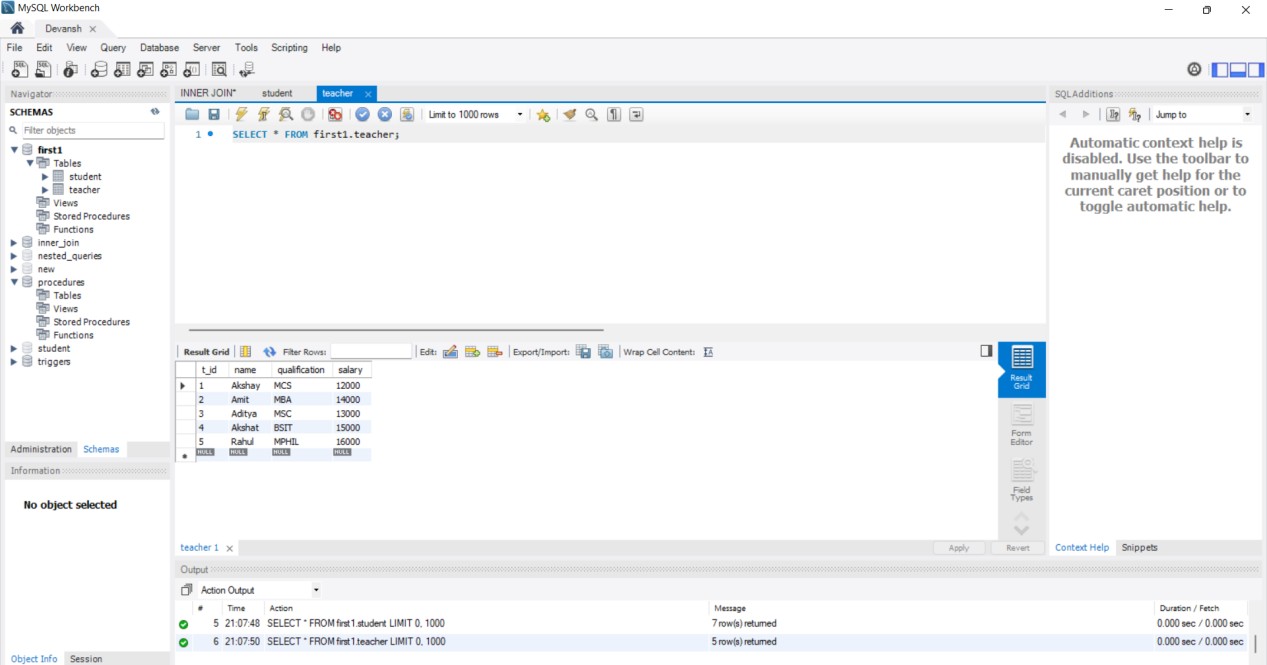
### *Syntax:-*

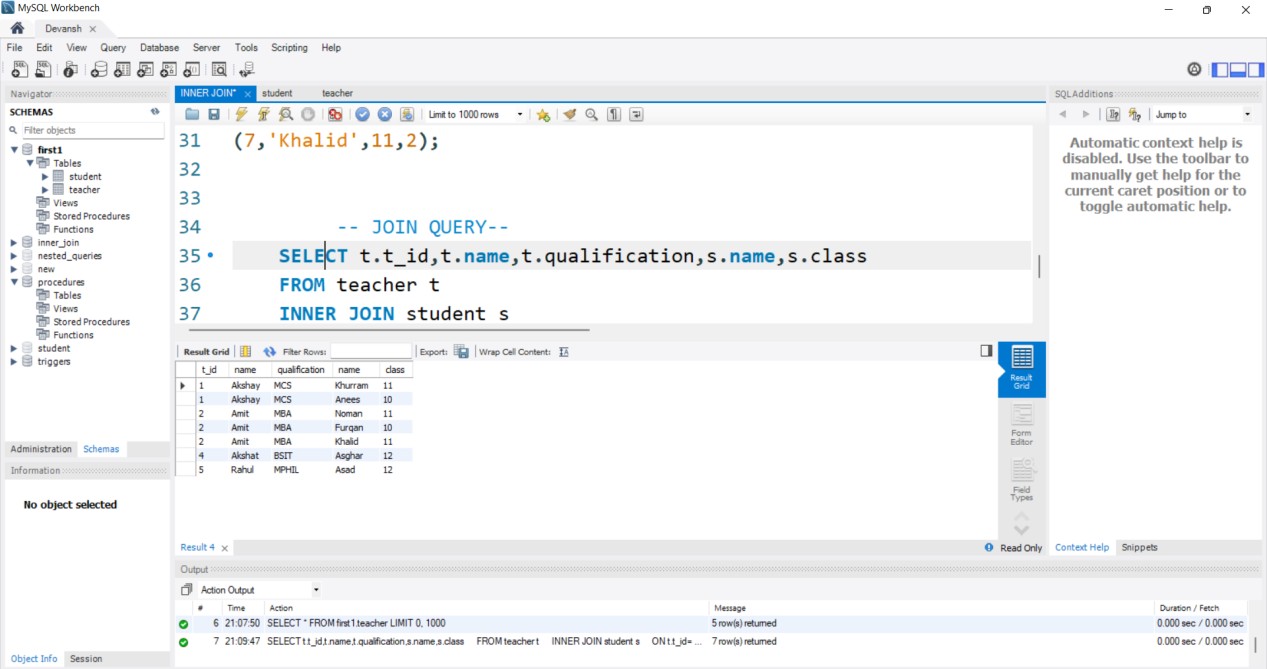
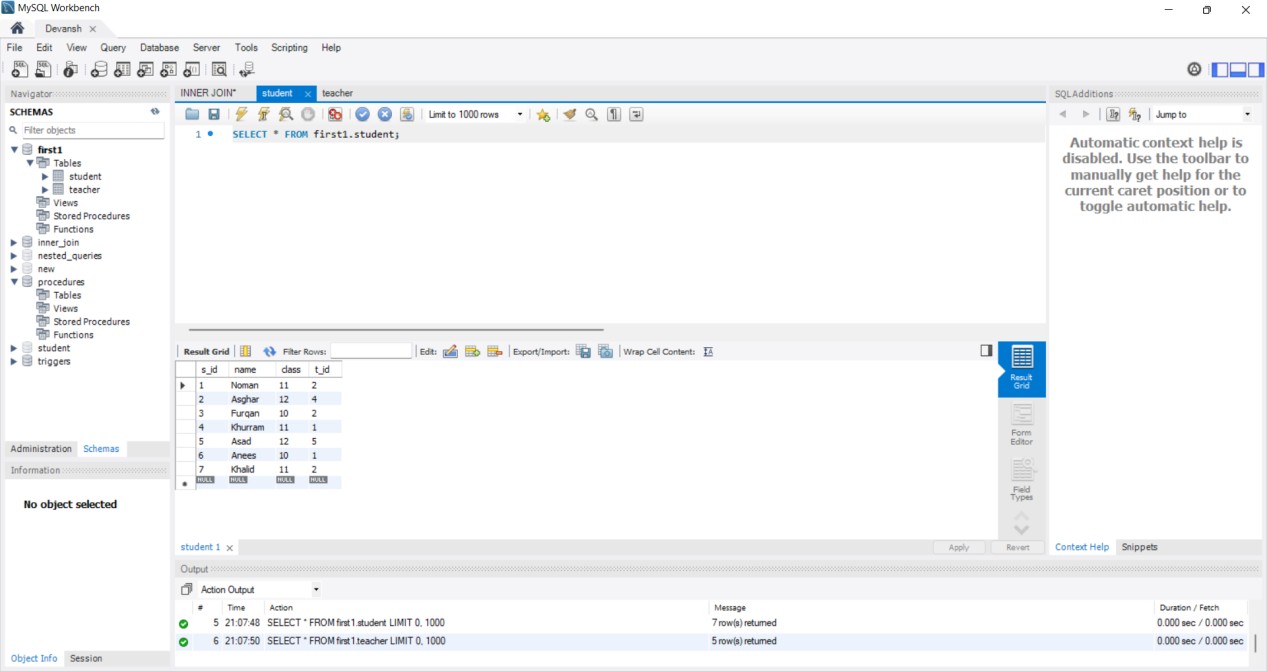
SELECT column\_name(s)  
FROM table1 T1, table1 T2  
WHERE condition;

* select a.movie\_id,a.movie\_title,b.movie\_id,b.movie\_title from movie a,movie b where a.movie\_id=b.duration;

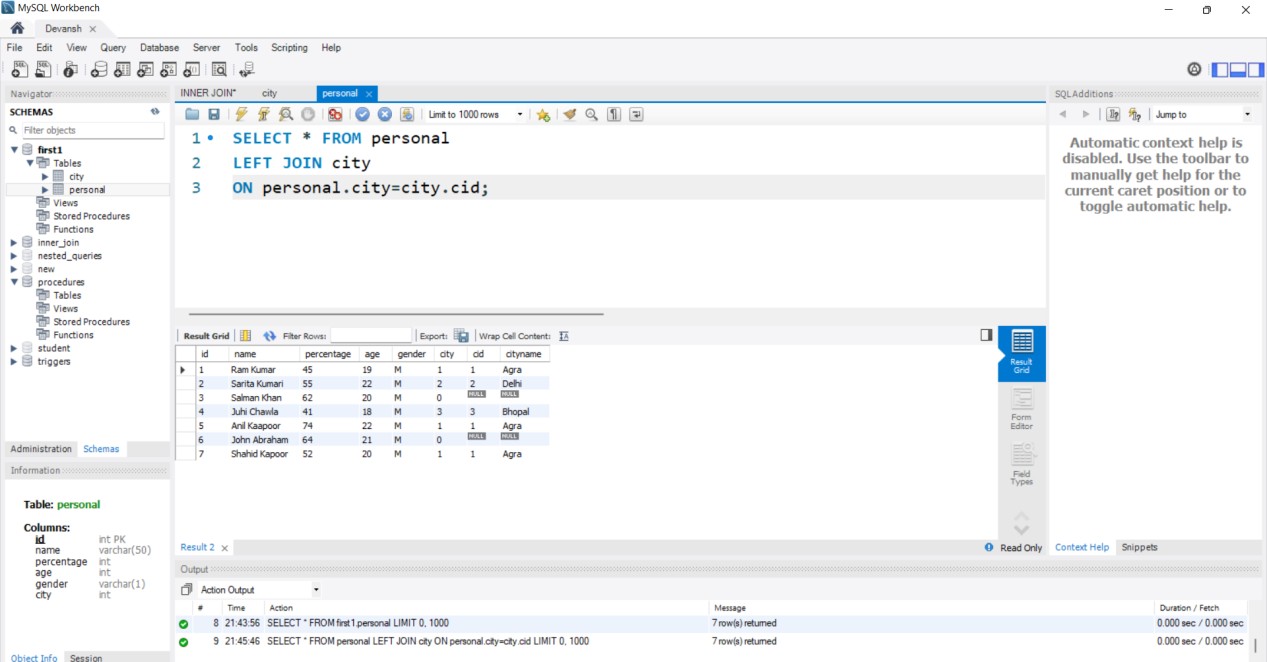
**OUTPUT:-**

MySQL INNER JOIN

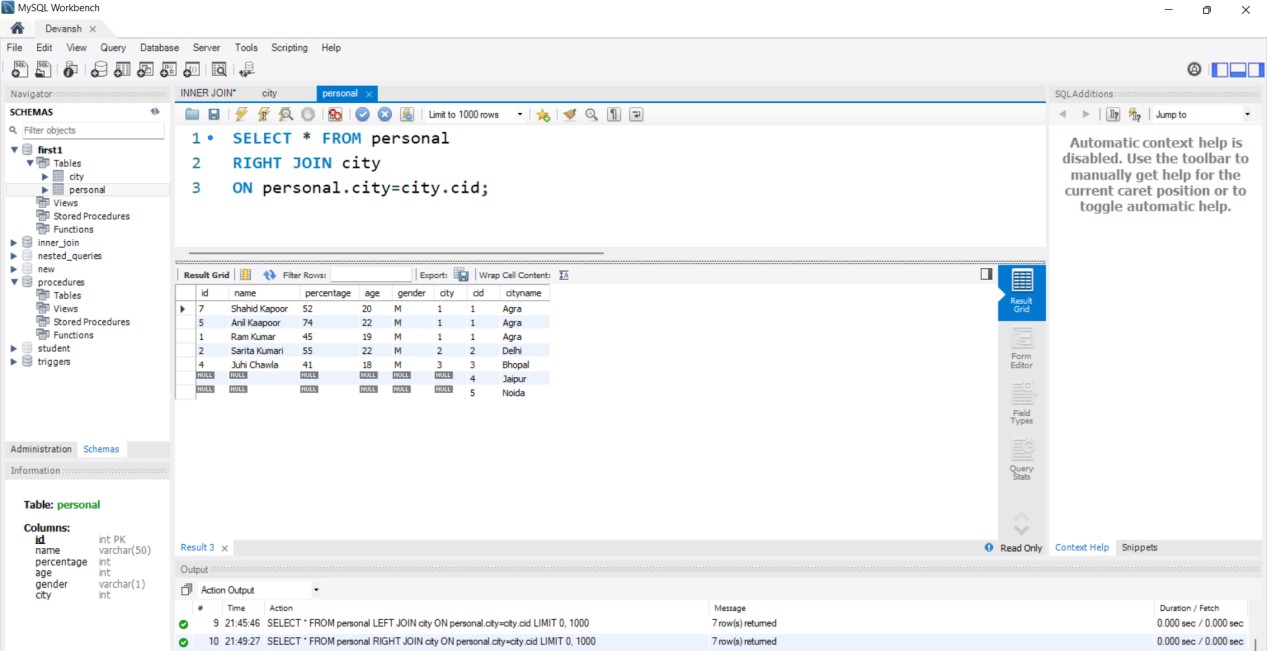




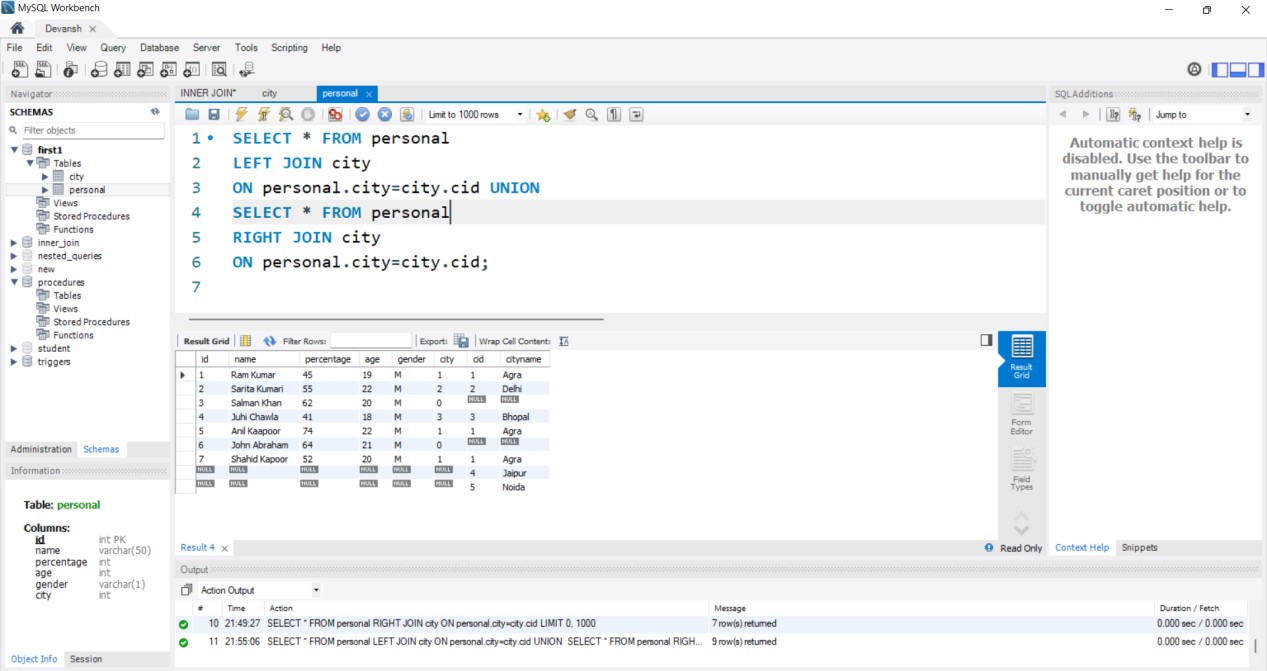
MySQL LEFT OUTER JOIN



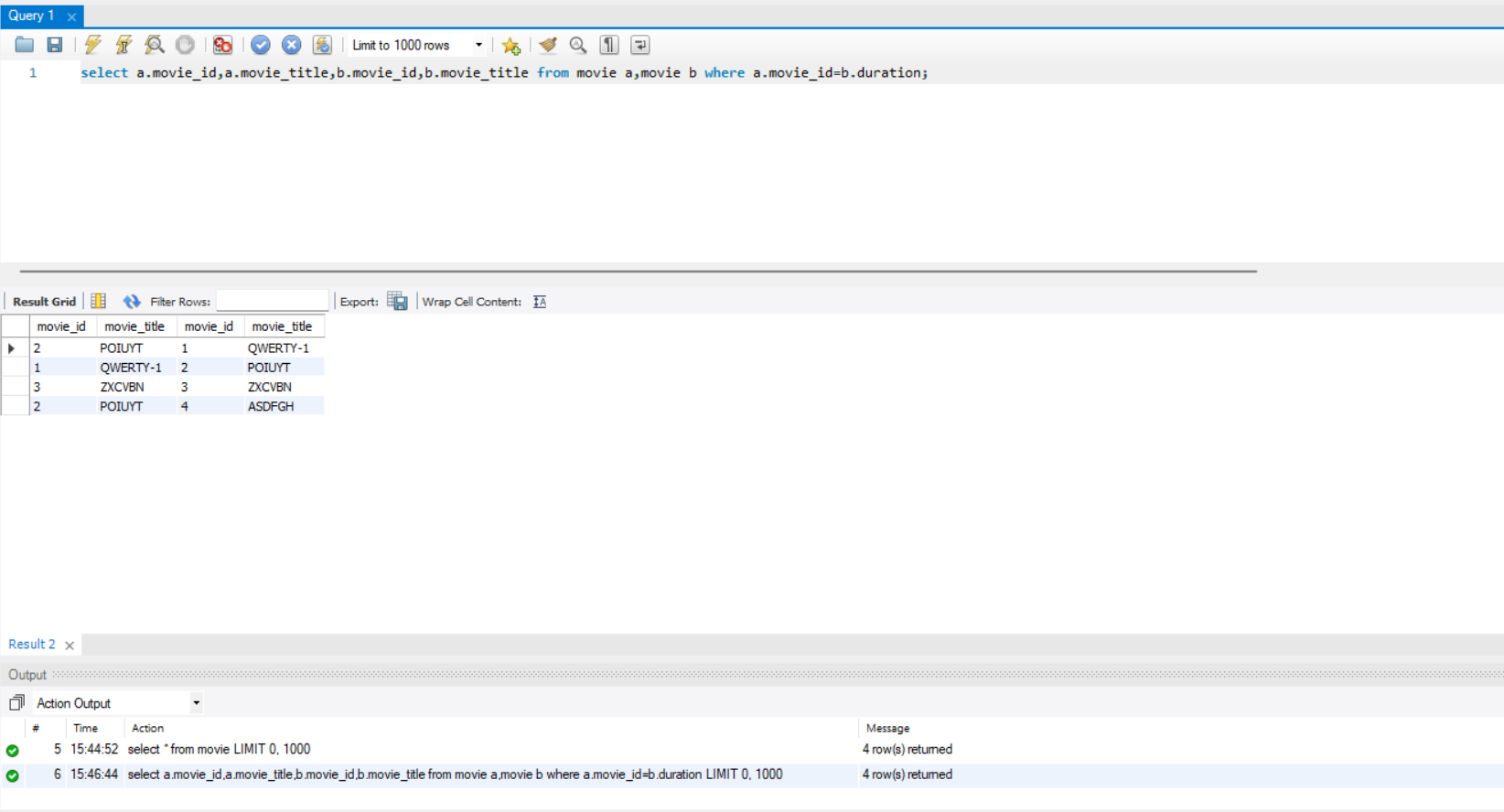
MySQL RIGHT JOIN



MySQL Full Outer JOIN



MySQL Self Join

****

**RESULT:** Joins in MySQL has been executed successfully.

**EXPERIMENT-8**

**AIM:-**To Execute Set Operators and Views in MySQL.

**PROCEDURE:-**

1. Goto Start and click on programs.
2. Click on SQL Workbench.
3. Then a pop-up menu will appear to authenticate you as user.
4. Now SQL Workbench window will get opened for working with various SQL commands in it.

**THEORY:-**

Set operators in SQL:-

Union :- It is used to combine the result of two select statements. Duplicate rows will be eliminated from the results obtained after performing the UNION operation.

*Syntax:-*

SELECT \* FROM *table1*

UNION

SELECT \*FROM t*able2*;

Union All:- This operator combines all the records from both the queries.Duplicate rows will be not be eliminated from the results obtained after performing the UNION ALL operation.

*Syntax:-*

SELECT \* from *A\_table*

UNION ALL

SELECT \* from *B\_table*;

Intersect:- It is used to combine two SELECT statements, but it only returns the records which are common from both SELECT statements.

*Syntax:-*

SELECT \* from *A\_table*

INTERSECT

SELECT \* from *B\_table*;

Minus:- It displays the rows which are present in the first query but absent in the second query with no duplicates.

*Syntax:-*

SELECT \* from *A\_table*

MINUS

SELECT \* from *B\_table*;

Views:- Views in SQL are kind of virtual tables. A view also has rows and columns as they are in a real table in the database. We can create a view by selecting fields from one or more tables present in the database. A View can either have all the rows of a table or specific rows based on certain condition.

CREATING VIEWS :- We can create View using **CREATE VIEW** statement.

*Syntax:-*

CREATE VIEW view\_name AS

SELECT column1, column2.....

FROM table\_name

WHERE condition;

* create view academics as select stu\_name from student;

### MODIFICATION OF VIEWS: -  A view can be modified by simply using the ALTER VIEW keyword instead, and then changing the structure of the SELECT statement.

*Syntax:-*

### ALTER VIEW view\_name AS

### SELECT column1, column2, ...

### FROM table\_name

### WHERE condition;

* alter view academics compile;

### DELETION OF VIEW :- To delete a view in SQL, you can use the DROP VIEW statement.

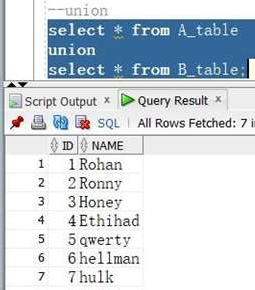
*Syntax:-*

### DROP VIEW view\_name;

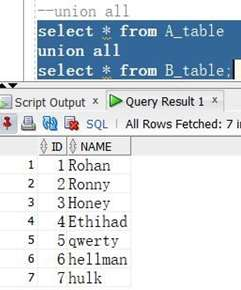
* drop view academics;

**OUTPUT:-**

UNION



UNION ALL

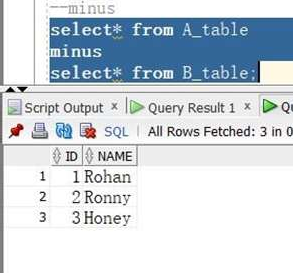


INTERSECT

Graphical user interface, text, application

Description automatically generated

MINUS

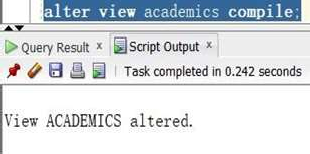


CREATION OF VIEW

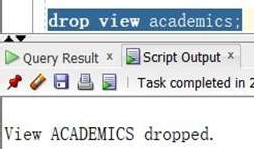
Graphical user interface, text, application, email

Description automatically generated

MODIFICATION OF VIEW



DELETION OF VIEWS



**RESULT:-** Set operators and Views in MySQL has been executed successfully.

**EXPERIMENT-9**

**AIM-** Write a PL/SQL query for Conditional and Iterative Statements.

**PROCEDURE:-**

1. Open your web browser and navigate to the Oracle Live SQL and log in using your Oracle account.
2. Click on the "SQL Worksheet" tab at the top of the page to open a new worksheet.
3. In the worksheet, you can write your PL/SQL code.
4. Once you have written your code, click on the "Run" button at the top of the page to execute the code.

**THEORY:-**

Conditional Statements:-

Conditional statements are used to execute different blocks of code based on specific conditions. The following are the conditional statements available in PL/SQL:

IF-THEN:

This statement executes a block of code if a specified condition is true.

*Syntax:-*

IF condition THEN

-- code to execute

END IF;

IF-THEN-ELSE:

This statement executes a block of code if a specified condition is true and another block of code if the condition is false.

*Syntax:-*

IF condition THEN

-- code to execute if condition is true

ELSE

-- code to execute if condition is false

END IF;

IF-THEN-ELSIF:

This statement executes a block of code if a specified condition is true and another block of code if the condition is false. It can also test additional conditions using the ELSIF clause.

*Syntax:-*

IF condition1 THEN

-- code to execute if condition1 is true

ELSIF condition2 THEN

-- code to execute if condition2 is true

ELSE

-- code to execute if all conditions are false

END IF;

Iterative Statements:-

Iterative statements are used to execute a block of code repeatedly. The following are the iterative statements available in PL/SQL:

FOR-LOOP:

This statement executes a block of code for a specified number of times.

*Syntax:-*

FOR i IN range LOOP

-- code to execute

END LOOP;

WHILE-LOOP:

This statement executes a block of code repeatedly while a specified condition is true.

*Syntax:-*

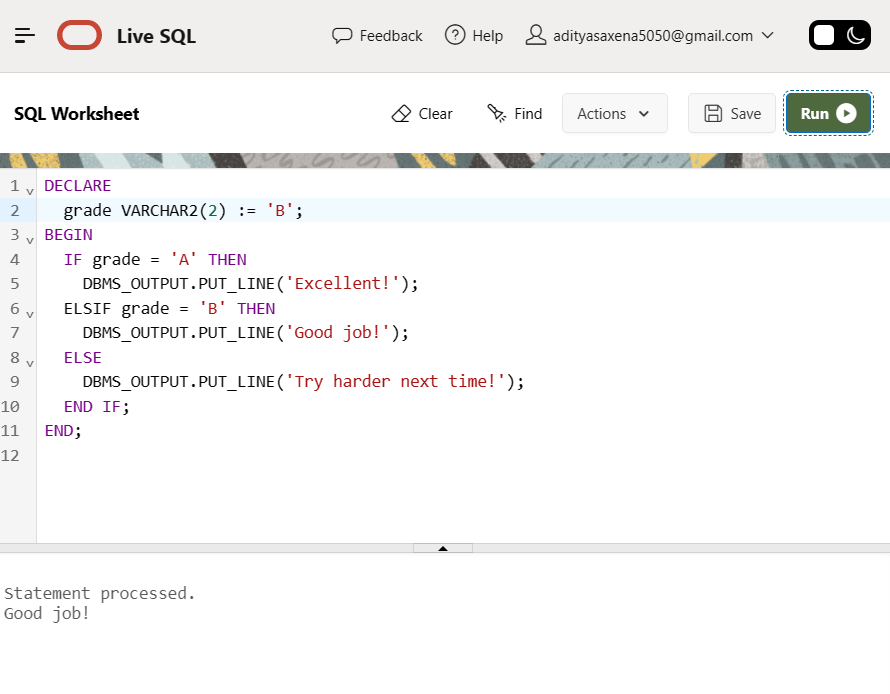
WHILE condition LOOP

-- code to execute

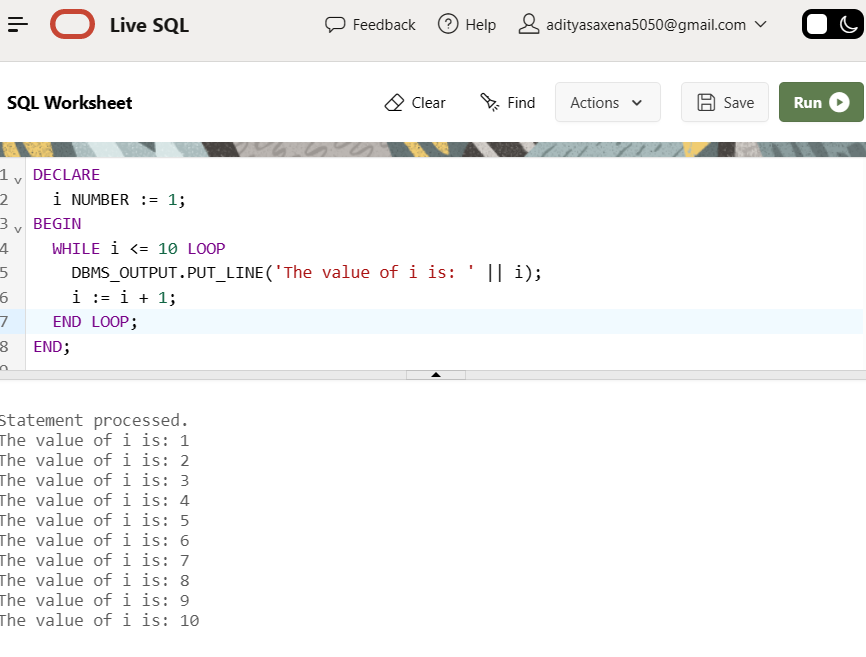
END LOOP;

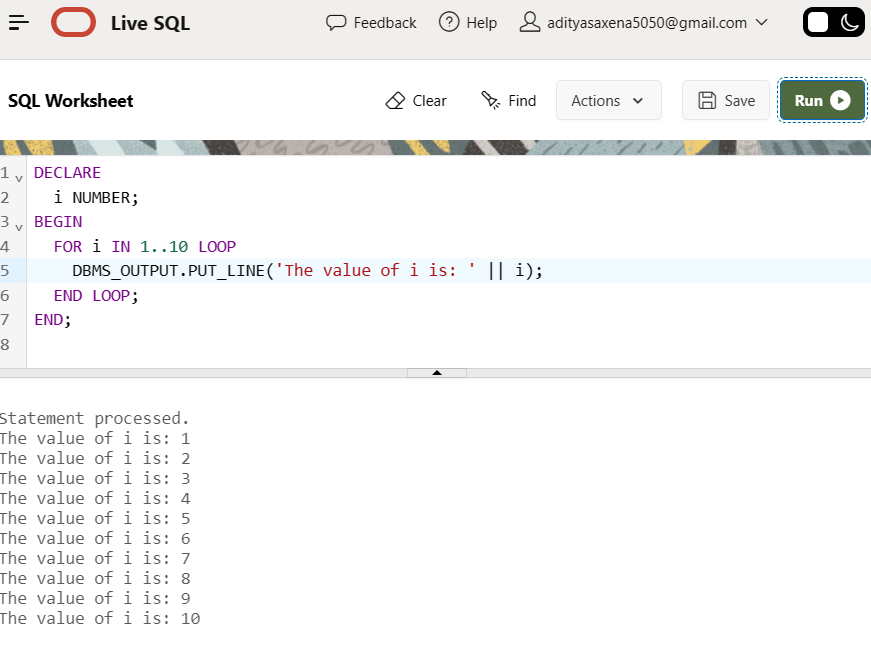
**OUTPUT:-**

Conditional Statements



Iterative Statements





**RESULT-** PL/SQL query for Conditional and Iterative Statements has been executed successfully.

**EXPERIMENT-10**

**AIM-** Write a PL/SQL query for Procedures on sample exercise.

**PROCEDURE:-**

1. Open your web browser and navigate to the Oracle Live SQL and log in using your Oracle account.
2. Click on the "SQL Worksheet" tab at the top of the page to open a new worksheet.
3. In the worksheet, you can write your PL/SQL code.
4. Once you have written your code, click on the "Run" button at the top of the page to execute the code.

**THEORY:-**

PL/SQL subprograms are named PL/SQL blocks that can be invoked with a set of parameters. PL/SQL provides two kinds of subprograms -

* Functions − These subprograms return a single value; mainly used to compute and return a value.
* Procedures − These subprograms do not return a value directly; mainly used to perform an action.

*Syntax*:-

CREATE [OR REPLACE] PROCEDURE procedure\_name

[(parameter\_name [IN | OUT | IN OUT] type [, ...])]

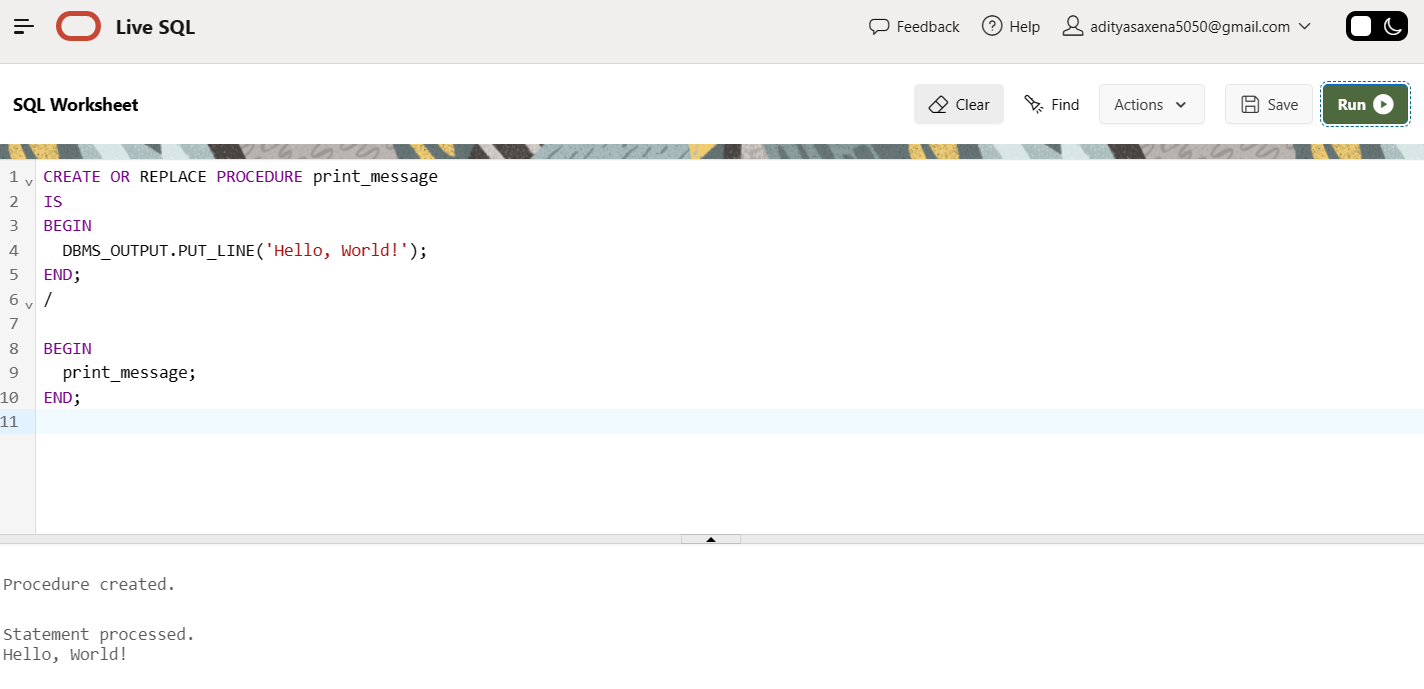
{IS | AS}

BEGIN

< procedure\_body >

END procedure\_name;

**OUTPUT:-**



**RESULT:-** PL/SQL query for Procedures on sample exercise has been executed successfully.

**EXPERIMENT-11**

**AIM-** Write a PL/SQL query for functions on sample exercise.

**PROCEDURE:-**

1. Open your web browser and navigate to the Oracle Live SQL and log in using your Oracle account.
2. Click on the "SQL Worksheet" tab at the top of the page to open a new worksheet.
3. In the worksheet, you can write your PL/SQL code.
4. Once you have written your code, click on the "Run" button at the top of the page to execute the code.

**THEORY:-**

A standalone function is created using the CREATE FUNCTION statement. The simplified syntax for the CREATE OR REPLACE PROCEDURE statement is as follows − on sample exercise.

*Syntax:-*

CREATE [OR REPLACE] FUNCTION function\_name

[(parameter\_name [IN | OUT | IN OUT] type [, ...])]

RETURN return\_datatype

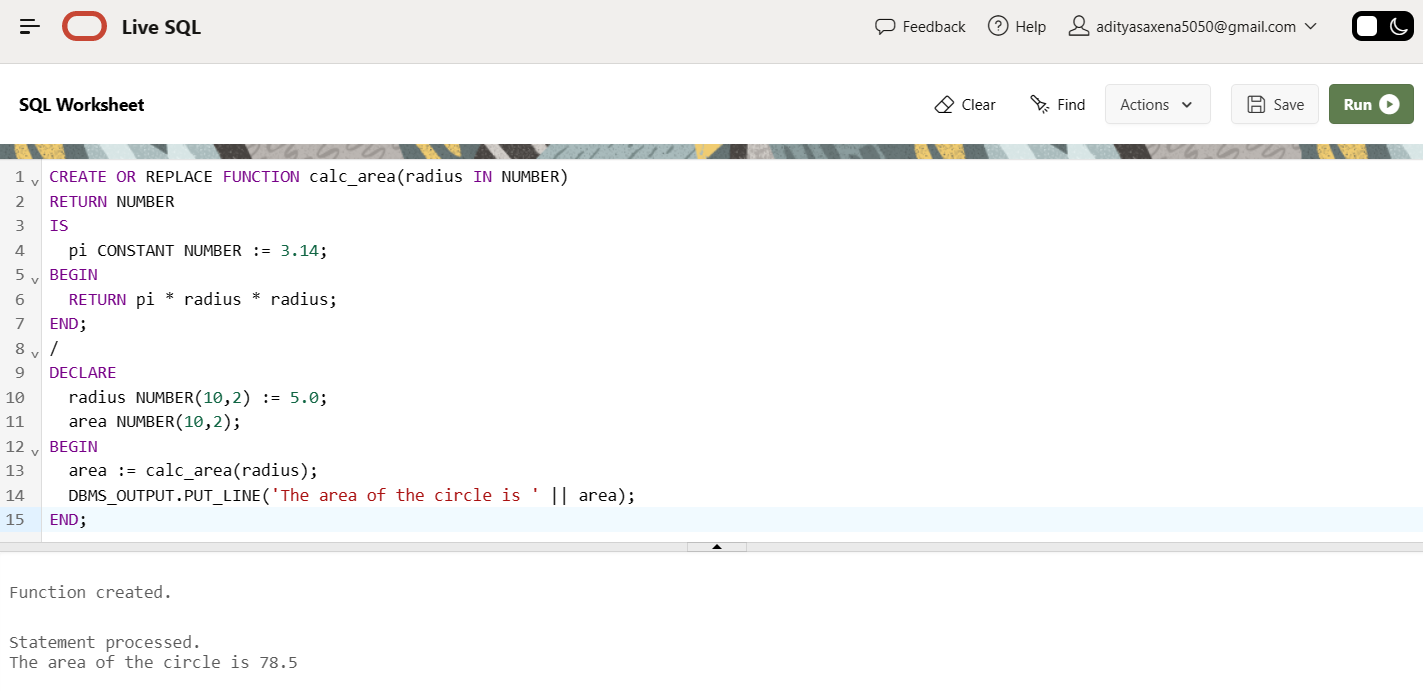
{IS | AS}

BEGIN

< function\_body >

END [function\_name];

**OUTPUT:-**



**RESULT:-** PL/SQL query for Functions on sample exercise has been executed successfully.

**EXPERIMENT-12**

**AIM-** Write a PL/SQL query for Cursors.

**PROCEDURE:-**

1. Open your web browser and navigate to the Oracle Live SQL and log in using your Oracle account.
2. Click on the "SQL Worksheet" tab at the top of the page to open a new worksheet.
3. In the worksheet, you can write your PL/SQL code.
4. Once you have written your code, click on the "Run" button at the top of the page to execute the code.

**THEORY:-**

A cursor is a pointer to this context area. PL/SQL controls the context area through a cursor. A cursor holds the rows (one or more) returned by a SQL statement. The set of rows the cursor holds is referred to as the active set.

There are two types of cursors −

Implicit cursors- Implicit cursors are automatically created by Oracle whenever an SQL statement is executed, when there is no explicit cursor for the statement. Programmers cannot control the implicit cursors and the information in it.

*Syntax:-*

BEGIN

-- SQL statement that retrieves data

SELECT column1, column2, ...

FROM table\_name

WHERE condition;

-- Processing of retrieved data

-- ...

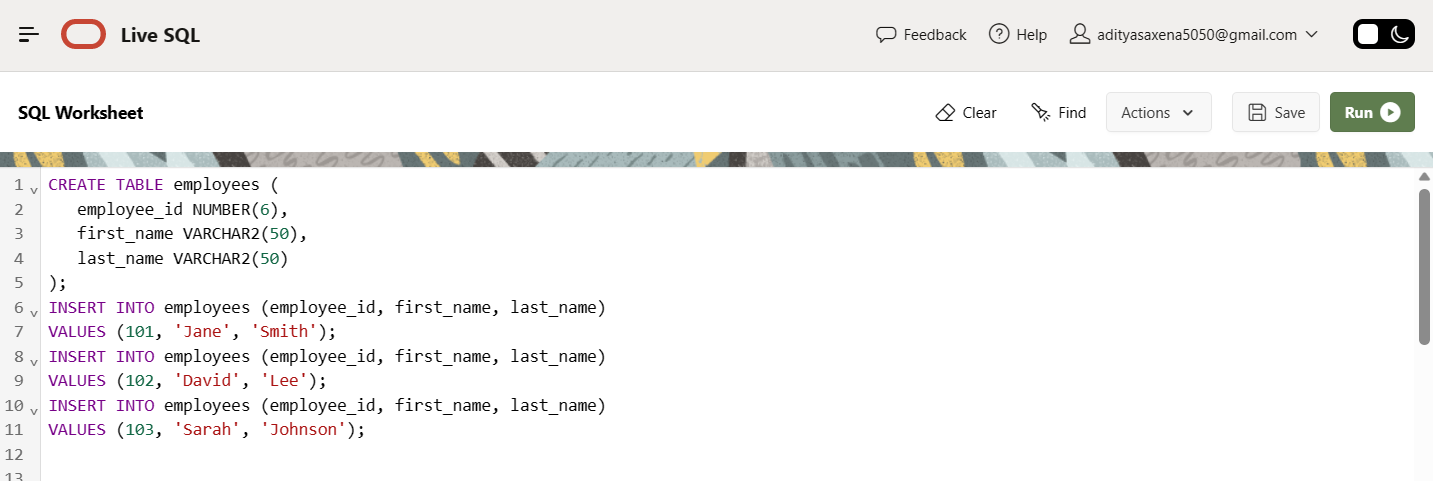
END;

Explicit cursors- Explicit cursors are programmer-defined cursors for gaining more control over the context area. An explicit cursor should be defined in the declaration section of the PL/SQL Block. It is created on a SELECT Statement which returns more than one row.

*Syntax:-*

CURSOR cursor\_name IS select\_statement;

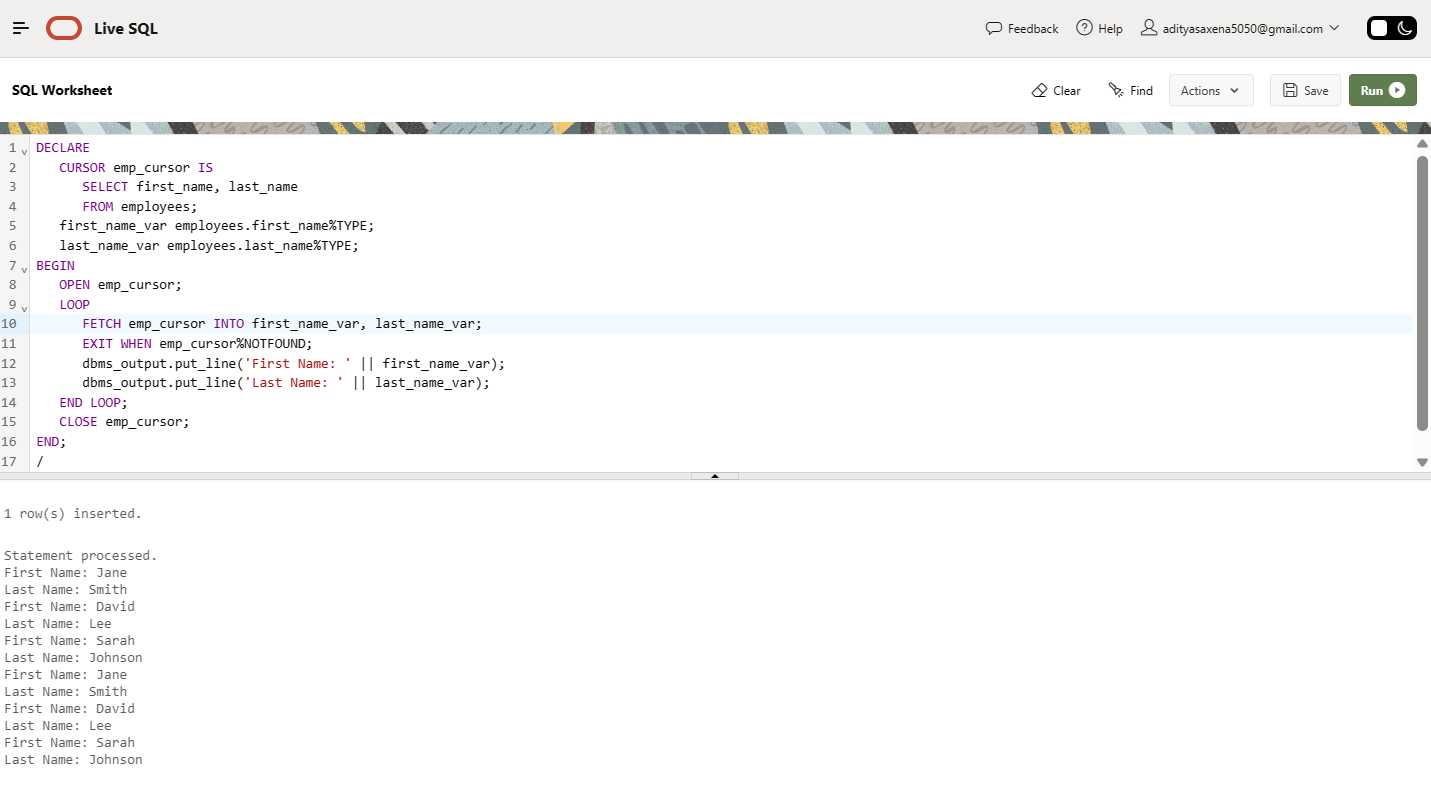
**OUTPUT:-**



Implicit cursors-



Explicit cursors-



**RESULT:-** PL/SQL query for Cursors has been executed successfully.

**EXPERIMENT-13**

**AIM-** Write a PL/SQL query for Exception Handling.

**PROCEDURE:-**

1. Open your web browser and navigate to the Oracle Live SQL and log in using your Oracle account.
2. Click on the "SQL Worksheet" tab at the top of the page to open a new worksheet.
3. In the worksheet, you can write your PL/SQL code.
4. Once you have written your code, click on the "Run" button at the top of the page to execute the code.

**THEORY:-**

An error occurs during the program execution is called Exception in PL/SQL.

PL/SQL facilitates programmers to catch such conditions using exception block in the program and an appropriate action is taken against the error condition.

There are two type of exceptions:

* System-defined Exceptions
* User-defined Exceptions

*Syntax:-*

DECLARE

<declarations section>

BEGIN

<executable command(s)>

EXCEPTION

<exception handling goes here >

WHEN exception1 THEN

exception1-handling-statements

WHEN exception2  THEN

exception2-handling-statements

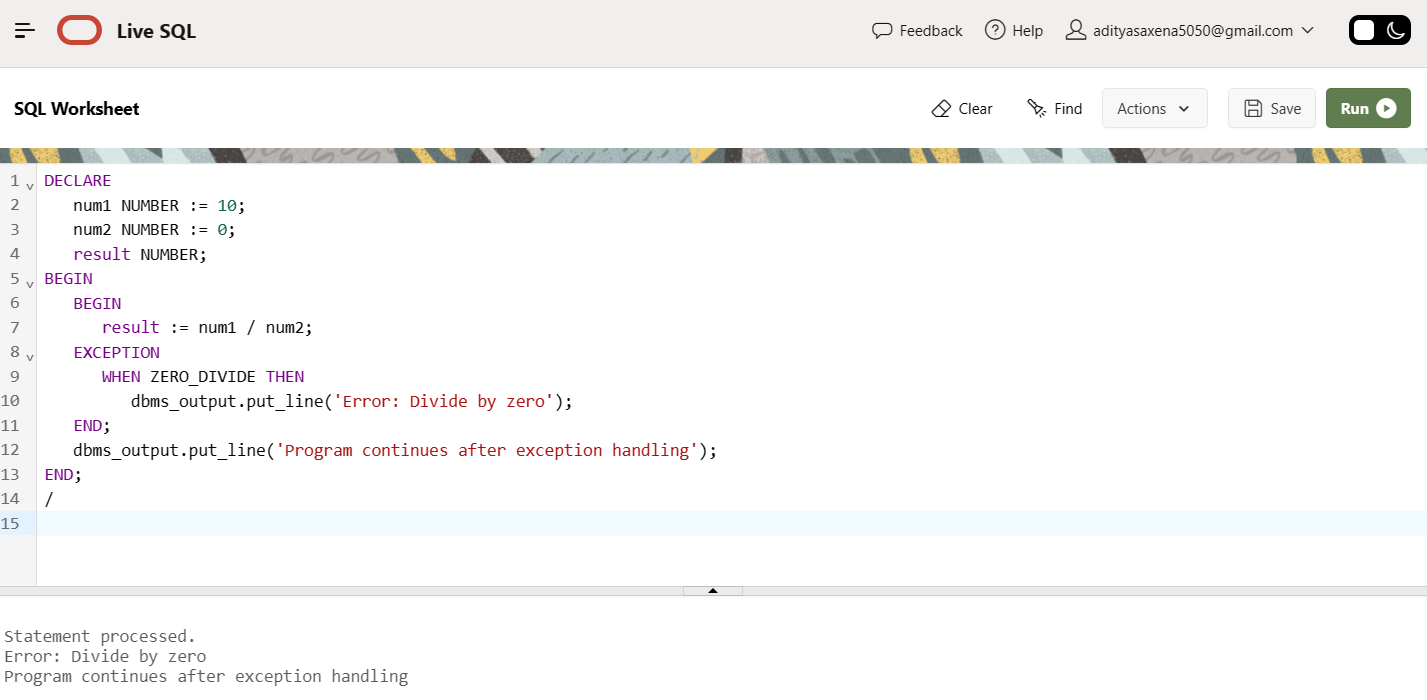
 ........

WHEN others THEN

exception3-handling-statements

END;

**OUTPUT:-**



**RESULT:-** PL/SQL query for Exception Handling has been executed successfully.

**EXPERIMENT-14**

**AIM-** Write a PL/SQL query for Trigger.

**PROCEDURE:-**

1. Open your web browser and navigate to the Oracle Live SQL and log in using your Oracle account.
2. Click on the "SQL Worksheet" tab at the top of the page to open a new worksheet.
3. In the worksheet, you can write your PL/SQL code.
4. Once you have written your code, click on the "Run" button at the top of the page to execute the code.

**THEORY:-**

 Triggers are stored programs, which are automatically executed or fired when some events occur. Triggers are, in fact, written to be executed in response to any of the following events −

* A database manipulation (DML) statement (DELETE, INSERT, or UPDATE)
* A database definition (DDL) statement (CREATE, ALTER, or DROP).
* A database operation (SERVERERROR, LOGON, LOGOFF, STARTUP, or SHUTDOWN).

Triggers can be defined on the table, view, schema, or database with which the event is associated.

*Syntax:-*

CREATE [OR REPLACE ] TRIGGER trigger\_name

{BEFORE | AFTER | INSTEAD OF } {INSERT [OR] | UPDATE [OR] | DELETE}

[OF col\_name] ON table\_name

[REFERENCING OLD AS o NEW AS n]

[FOR EACH ROW]

WHEN (condition)

DECLARE

Declaration-statements

BEGIN

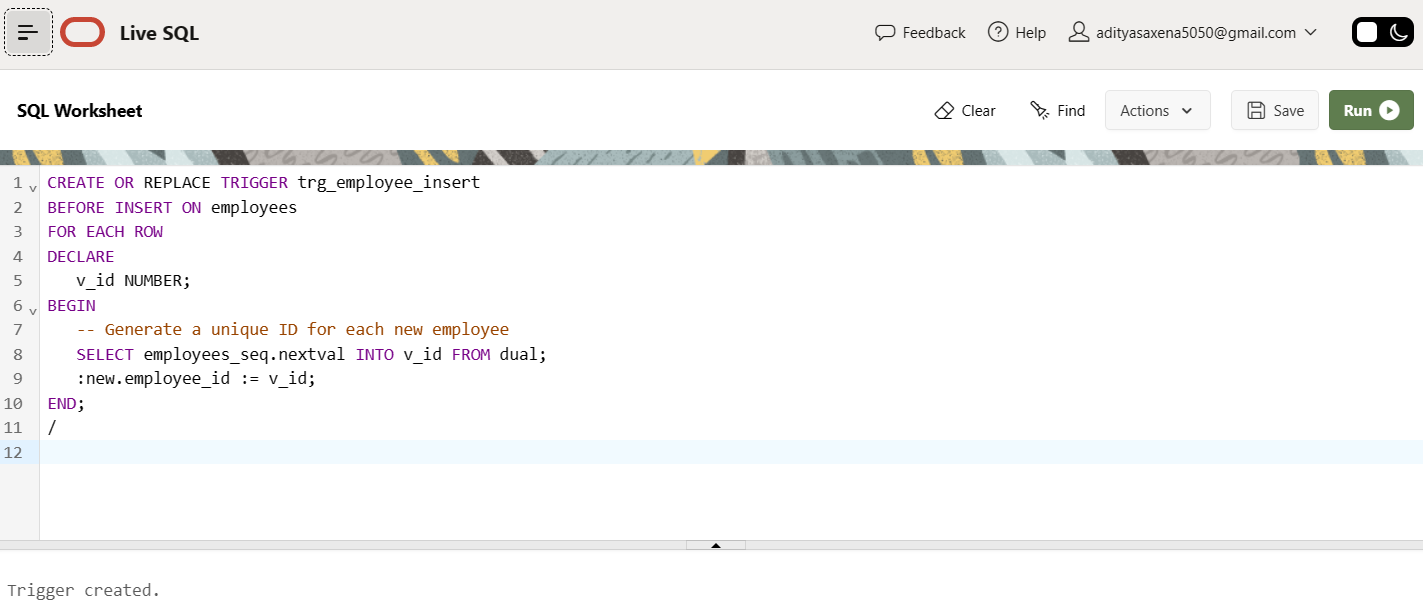
Executable-statements

EXCEPTION

Exception-handling-statements

END;

**OUTPUT:-**



**RESULT:-** PL/SQL query for Exception Handling triggers has been executed successfully.

**\*EXPERIMENT-15/16**

**AIM** -Frame and execute the appropriate PL/SQL Cursors and Exceptional Handling for the project.

**PROCEDURE:-**

1. Open your web browser and navigate to the Oracle Live SQL and log in using your Oracle account.
2. Click on the "SQL Worksheet" tab at the top of the page to open a new worksheet.
3. In the worksheet, you can write your PL/SQL code.
4. Once you have written your code, click on the "Run" button at the top of the page to execute the code.

**THEORY:-**

The **Online Store Management System** is a web-based application that facilitates the management of an online store. This system enables the store owner to manage products, categories, orders, customers, and payments.

In order to implement this system, a database schema needs to be designed that can store all the necessary information related to the online store. The schema should have tables for storing information about customers, products, categories, orders, and payments.

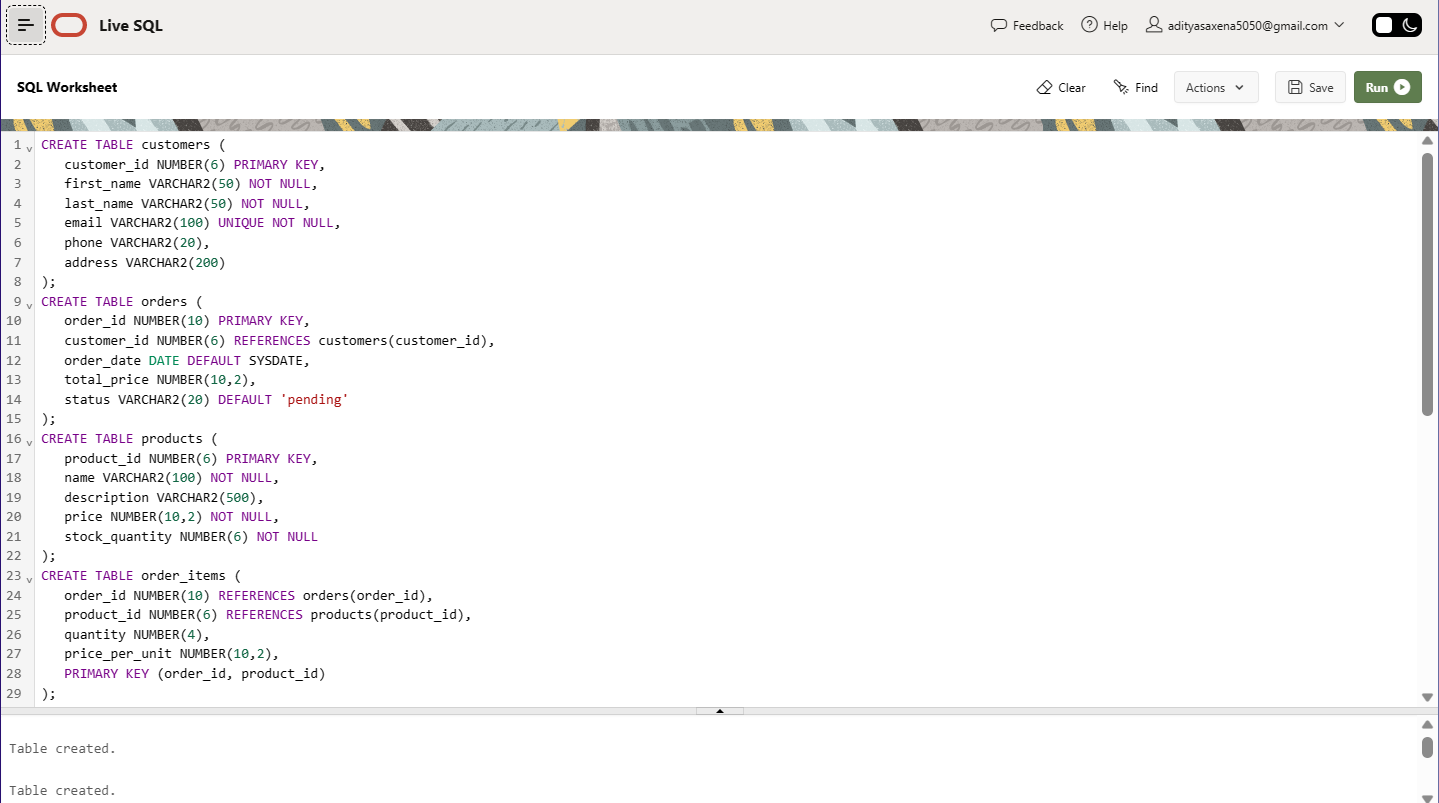
Once the database schema is designed, the next step is to create the necessary PL/SQL cursors and exceptional handling procedures to manage the data stored in the database. These cursors can be used to retrieve and manipulate data from the database, while the exceptional handling procedures can be used to handle any errors that may occur during data manipulation.

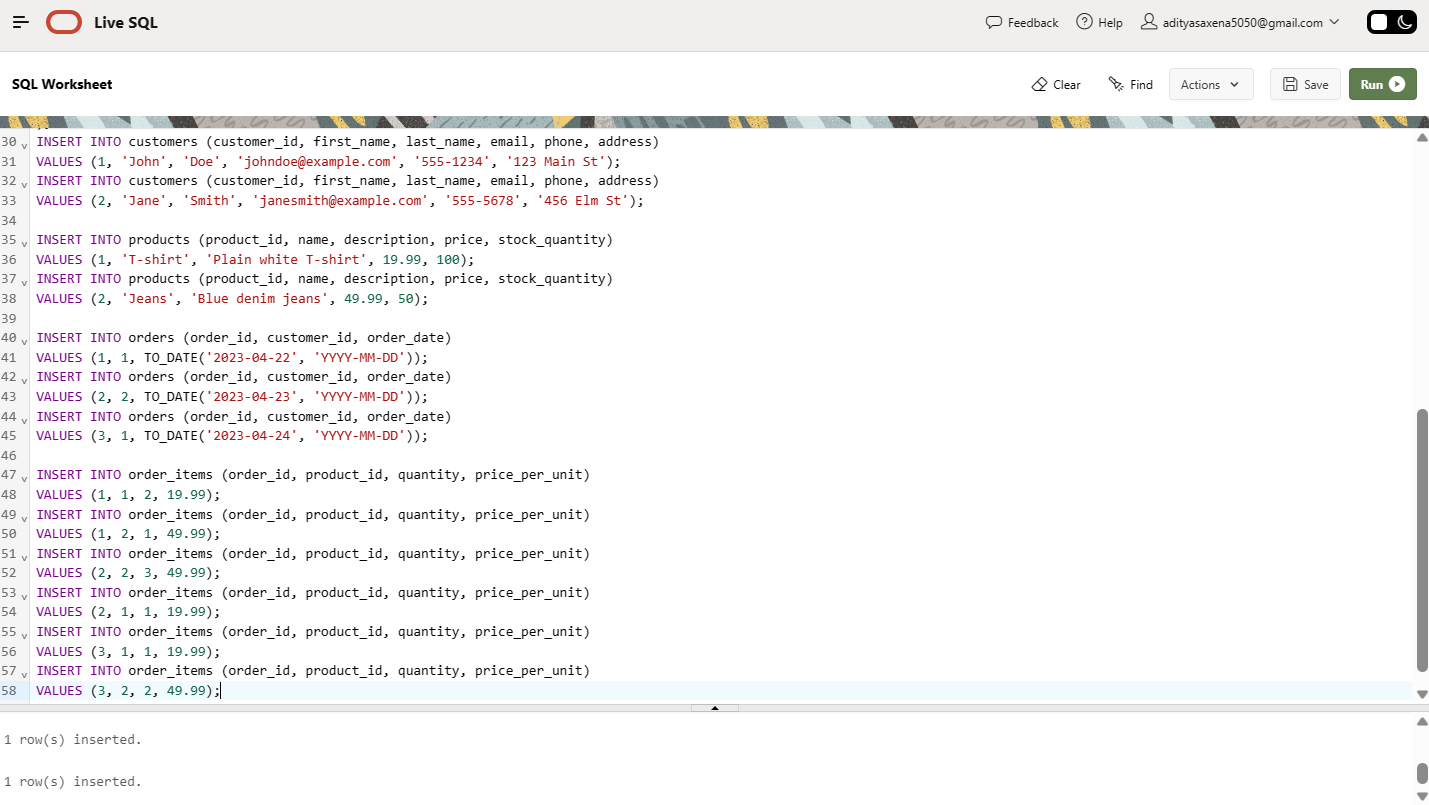
For example, an exceptional handling procedure can be created to handle any errors that occur during the insertion of data into the database. This procedure can be designed to roll back the transaction and display an appropriate error message to the user.

In addition to this, PL/SQL cursors can be used to retrieve data from the database based on user input. For instance, a cursor can be created to retrieve all the products that fall under a particular category.

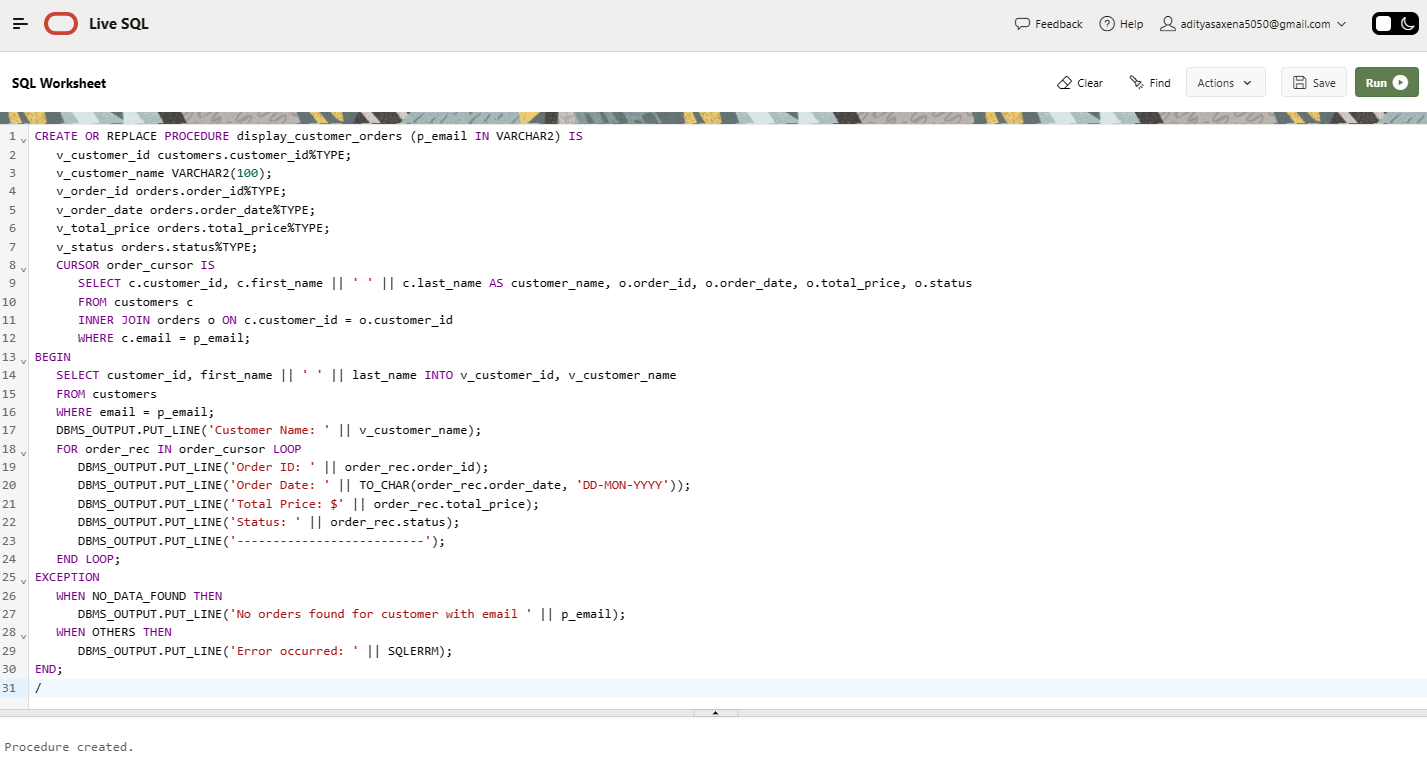
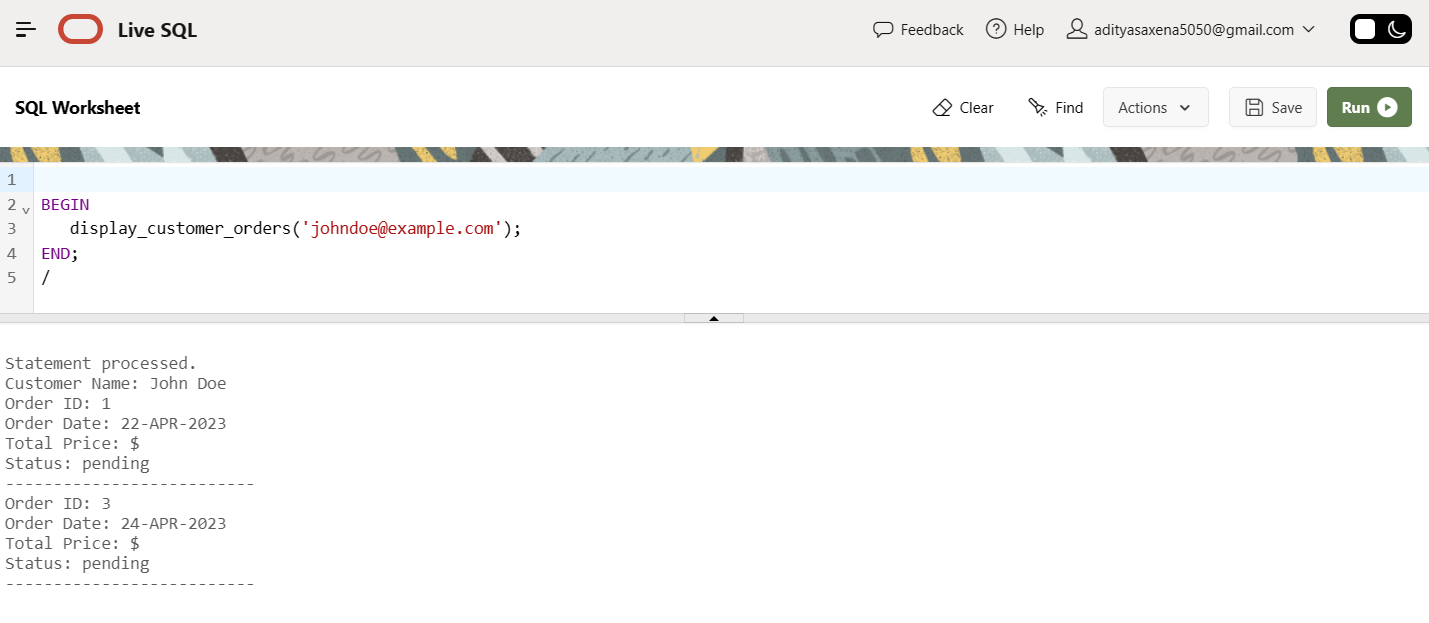
Overall, the Online Store Management System can be effectively implemented using a combination of database design, PL/SQL cursors, and exceptional handling procedures. This approach can ensure that the system is robust, reliable, and efficient in managing an online store.

**OUTPUT:-**

****

****

PL/SQL Cursors and Exceptional Handling Queries: -

****

**RESULT:-** Frame and execution by PL/SQL Cursors and Exceptional Handling for the project is done.