| Experiment No.9 |
| --- |
| Views & Triggers |
| Date of Performance: 2/4/24 |
| Date of Submission: 9/4/24 |

**Experiment No. 9:** : Views and Triggers

**Course Outcome [CSL503.3]:** Apply SQL queries ,triggers and procedures for specific module/task

**Aim**: Views and Triggers

**Theory:**

**VIEWS**

**A view is nothing more than a SQL statement that is stored in the database with an associated name. A view is actually a composition of a table in the form of a predefined SQL query.**

A view can contain all rows of a table or select rows from a table. A view can be created from one or many tables which depends on the written SQL query to create a view.

Views, which are a type of virtual tables allow users to do the following −

* Structure data in a way that users or classes of users find natural or intuitive.
* Restrict access to the data in such a way that a user can see and (sometimes) modify exactly what they need and no more.
* Summarize data from various tables which can be used to generate reports.

## Creating Views

Database views are created using the **CREATE VIEW** statement. Views can be created from a single table, multiple tables or another view.

To create a view, a user must have the appropriate system privilege according to the specific implementation.

The basic **CREATE VIEW** syntax is as follows −

CREATE VIEW view\_name AS

SELECT column1, column2.....

FROM table\_name

WHERE [condition];

You can include multiple tables in your SELECT statement in a similar way as you use them in a normal SQL SELECT query.

**TIGGERS:**

**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.

### Benefits of Triggers

Triggers can be written for the following purposes −

* Generating some derived column values automatically
* Enforcing referential integrity
* Event logging and storing information on table access
* Auditing
* Synchronous replication of tables
* Imposing security authorizations
* Preventing invalid transactions

## Creating Triggers

The syntax for creating a trigger is −

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;

Where,

* CREATE [OR REPLACE] TRIGGER trigger\_name − Creates or replaces an existing trigger with the *trigger\_name*.
* {BEFORE | AFTER | INSTEAD OF} − This specifies when the trigger will be executed. The INSTEAD OF clause is used for creating trigger on a view.
* {INSERT [OR] | UPDATE [OR] | DELETE} − This specifies the DML operation.
* [OF col\_name] − This specifies the column name that will be updated.
* [ON table\_name] − This specifies the name of the table associated with the trigger.
* [REFERENCING OLD AS o NEW AS n] − This allows you to refer new and old values for various DML statements, such as INSERT, UPDATE, and DELETE.
* [FOR EACH ROW] − This specifies a row-level trigger, i.e., the trigger will be executed for each row being affected. Otherwise the trigger will execute just once when the SQL statement is executed, which is called a table level trigger.
* WHEN (condition) − This provides a condition for rows for which the trigger would fire. This clause is valid only for row-level triggers.

**Code:**

create table Empl (

id int (10),

Fname char (50),

Lname char (50),

Salary char (20)

);

create table Staff (

id int(10),

city char(20)

);

select \* from Empl;

select \* from Staff;

insert into Empl(id,Fname,Lname,Salary)

values (1,"ABC","M",10000),

(2,"XYZ","N",30000),

(3,"PQR","O",40000);

insert into Staff (id,city)

values (1,"Pune"),

(5,"Mumbai"),

(6,"Banglore");

create view View1 as select id ,Fname,Lname from Empl;

select \* from View1;

create view View2 as select id ,Fname,Lname from Empl where Salary >10000;

select \* from View2;

drop view View30;

create view View30 as select Empl.Fname, Empl.Lname, Staff.city from Empl, Staff where Empl.id = Staff.id;

select \* from View30;

-- Triggers

delimiter //

create trigger tigg20

before

insert on Empl

for each row

begin

if new.Salary > 40000 then set new.Salary = 0;

end if ;

end;

-- Insert in Trigger --

insert into Empl(id,Fname,Lname,Salary)

values (4,"XYZ","L",80000);

select \*from Empl;

delimiter //

create trigger tigg38

after update on Empl

for each row

begin

if new.Salary = 10000 then

update Empl set Salary = 85000;

elseif new.Salary < 10000 then

update Empl set Salary = 72000;

end if;

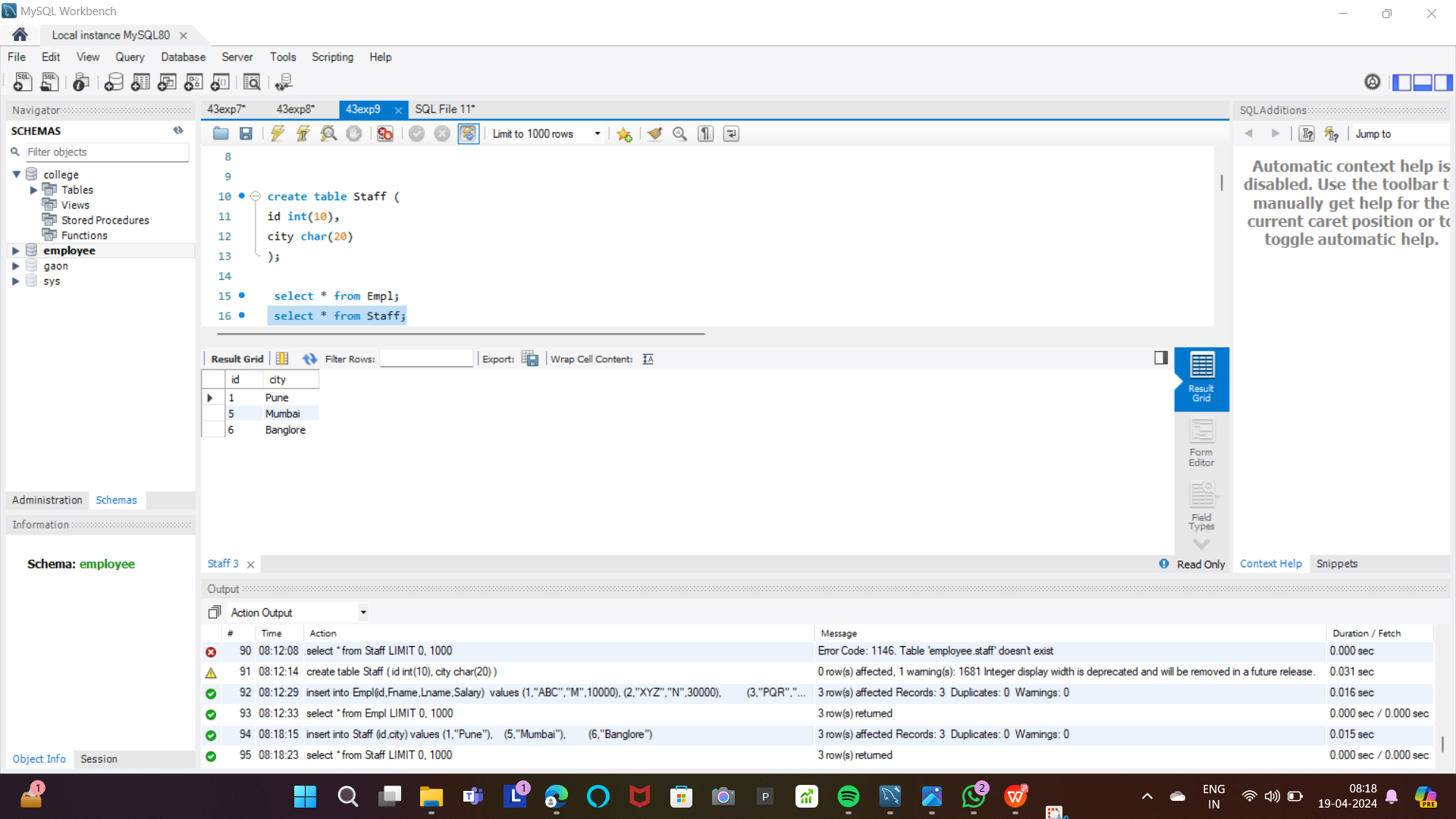
end;

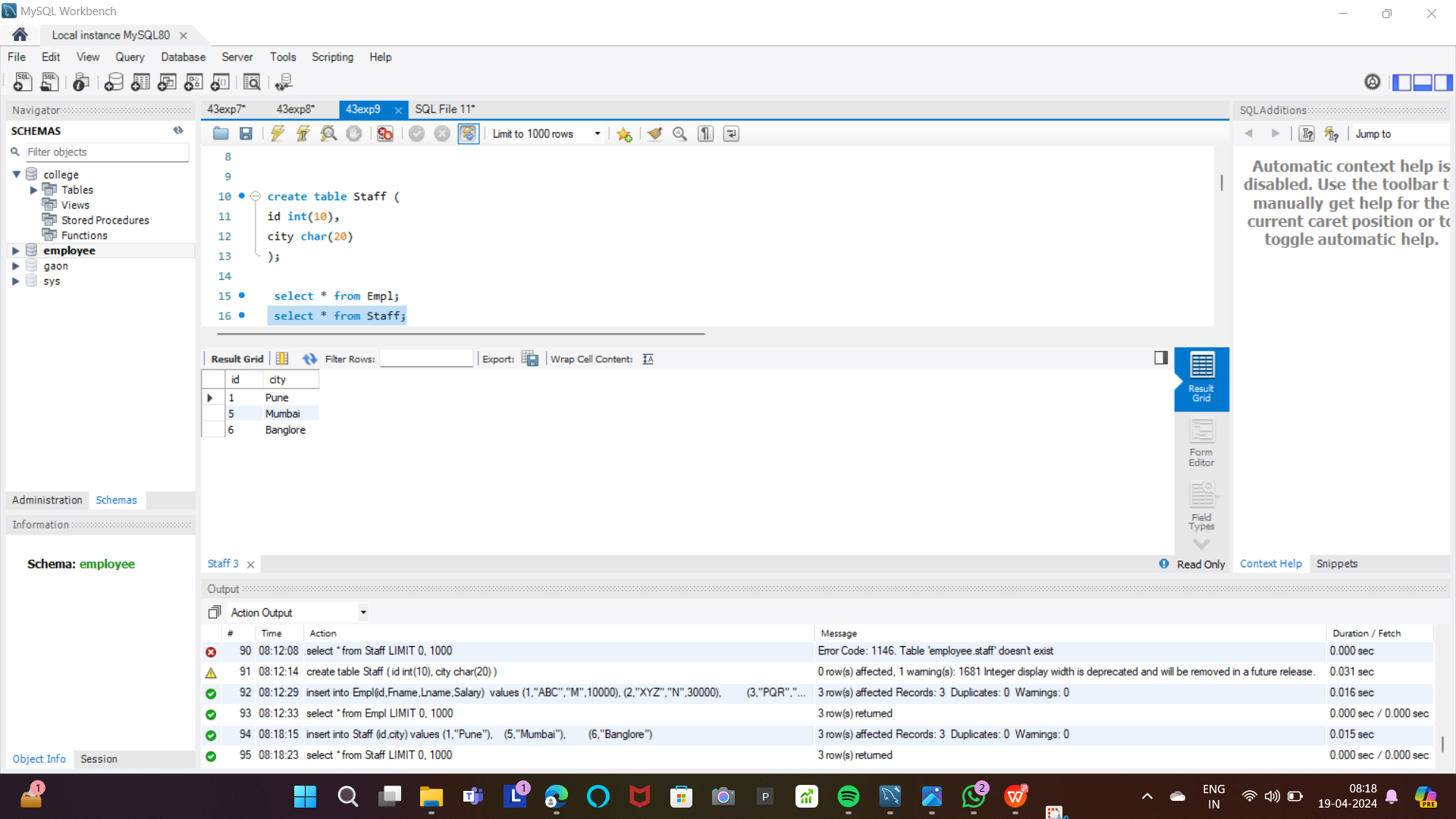
update Empl set Salary = 8000;

select \* from Empl;

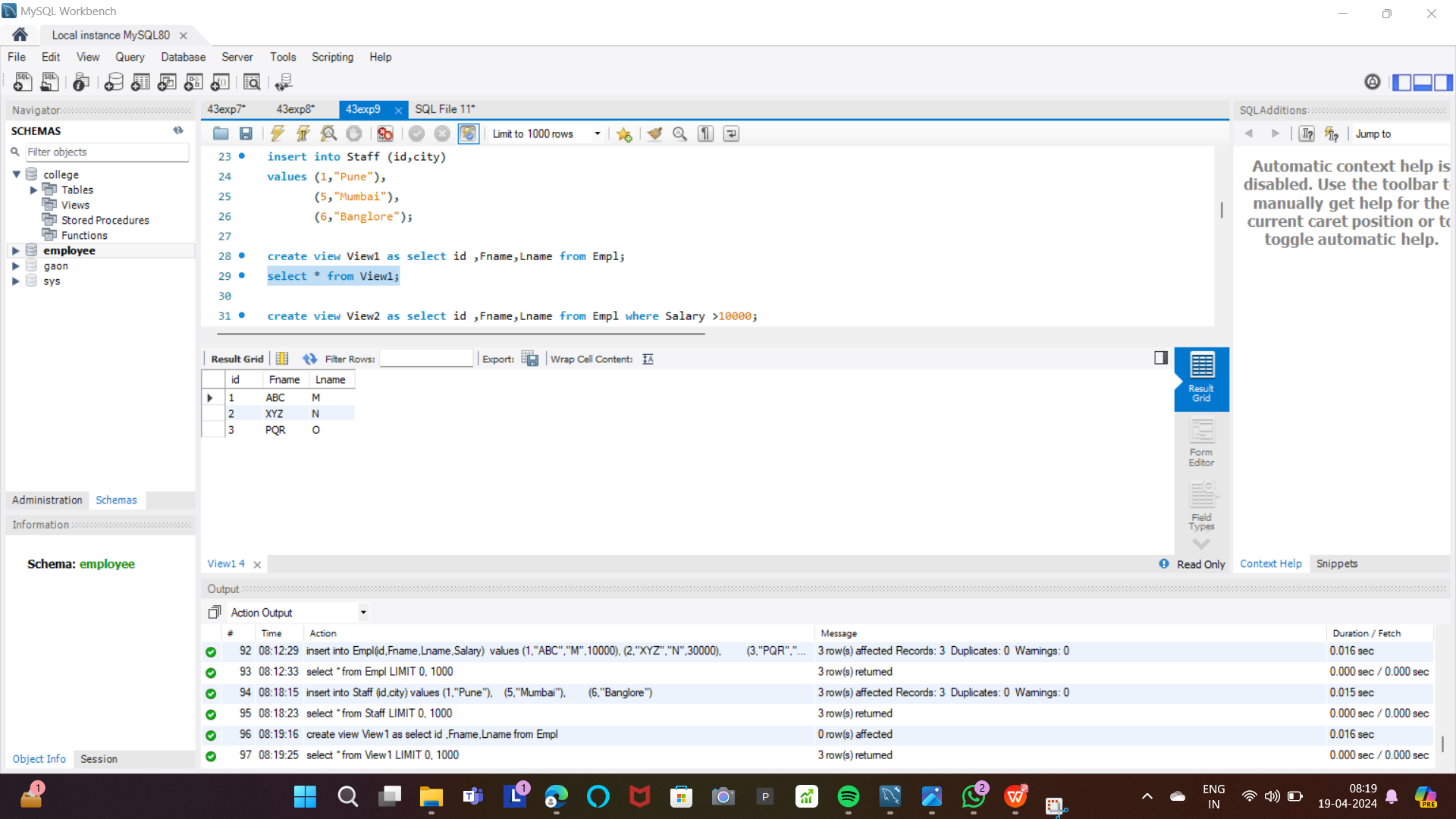
**Output:**

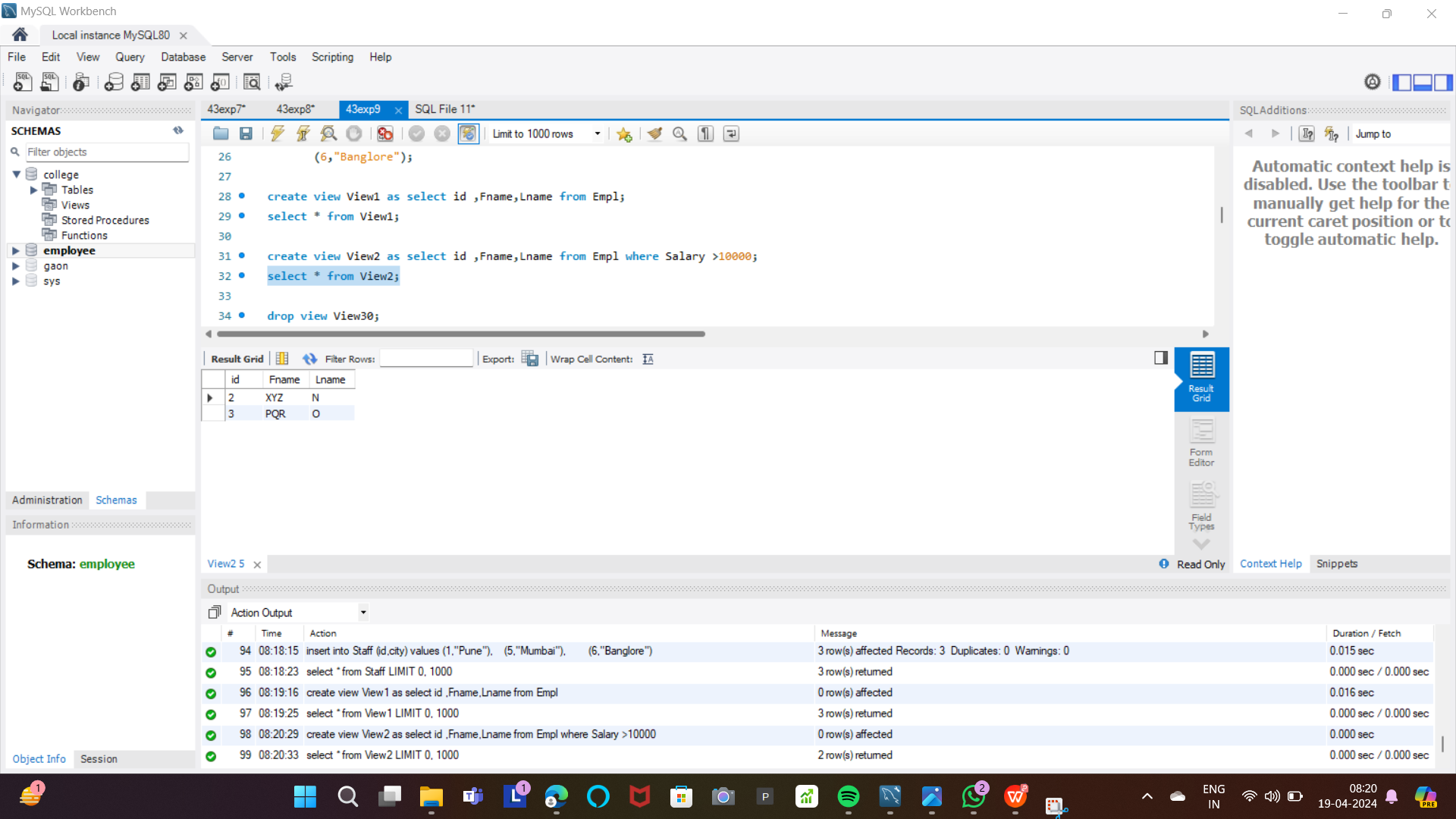
**Tables(Create and insert ):**



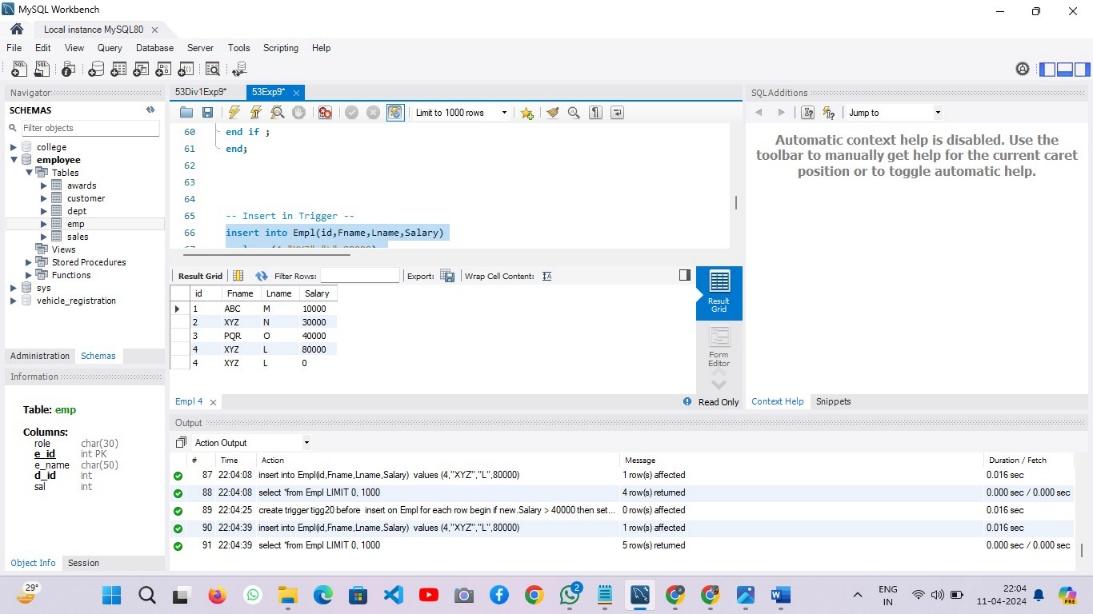


**Views:**





**Triggers:**



**Conclusion:**

This experiment delved into the practical applications of views and triggers in SQL. Views provide a flexible way to structure and present data, allowing for intuitive access and data summarization. On the other hand, triggers offer automated responses to database events, facilitating tasks such as enforcing data integrity, logging events, and generating derived column values. Understanding and leveraging views and triggers enhances database management capabilities, offering valuable tools for data organization, access control, and automated actions.