| Experiment No.8 |
| --- |
| Functions, Cursor & Procedure |
| Date of Performance: 26/3/24 |
| Date of Submission: 2/4/24 |

**Experiment No. 8. Functions , cursor and procedure**

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

**Aim**: Functions , cursor and procedure.

**Theory:**

## Creating a Function

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

CREATE [OR REPLACE] FUNCTION function\_name

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

RETURN return\_datatype

{IS | AS}

BEGIN

< function\_body >

END [function\_name];

Where,

* *function-name* specifies the name of the function.
* [OR REPLACE] option allows the modification of an existing function.
* The optional parameter list contains name, mode and types of the parameters. IN represents the value that will be passed from outside and OUT represents the parameter that will be used to return a value outside of the procedure.
* The function must contain a **return** statement.
* The *RETURN* clause specifies the data type you are going to return from the function.
* *function-body* contains the executable part.
* The AS keyword is used instead of the IS keyword for creating a standalone function.

CURSOR

**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.**

You can name a cursor so that it could be referred to in a program to fetch and process the rows returned by the SQL statement, one at a time. There are two types of cursors −

* Implicit cursors
* Explicit 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.

Whenever a DML statement (INSERT, UPDATE and DELETE) is issued, an implicit cursor is associated with this statement. For INSERT operations, the cursor holds the data that needs to be inserted. For UPDATE and DELETE operations, the cursor identifies the rows that would be affected.

In PL/SQL, you can refer to the most recent implicit cursor as the **SQL cursor**, which always has attributes such as **%FOUND, %ISOPEN, %NOTFOUND**, and **%ROWCOUNT**. The SQL cursor has additional attributes, **%BULK\_ROWCOUNT** and **%BULK\_EXCEPTIONS**, designed for use with the **FORALL** statement. The following table provides the description of the most used attributes −

| S.No | Attribute & Description |
| --- | --- |
| 1 | **%FOUND**  Returns TRUE if an INSERT, UPDATE, or DELETE statement affected one or more rows or a SELECT INTO statement returned one or more rows. Otherwise, it returns FALSE. |
| 2 | **%NOTFOUND**  The logical opposite of %FOUND. It returns TRUE if an INSERT, UPDATE, or DELETE statement affected no rows, or a SELECT INTO statement returned no rows. Otherwise, it returns FALSE. |
| 3 | **%ISOPEN**  Always returns FALSE for implicit cursors, because Oracle closes the SQL cursor automatically after executing its associated SQL statement. |
| 4 | **%ROWCOUNT**  Returns the number of rows affected by an INSERT, UPDATE, or DELETE statement, or returned by a SELECT INTO statement. |

**Procedure:**

**A procedure is created with the CREATE OR REPLACE PROCEDURE statement. The simplified syntax for the CREATE OR REPLACE PROCEDURE statement is as follows −**

CREATE [OR REPLACE] PROCEDURE procedure\_name

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

{IS | AS}

BEGIN

< procedure\_body >

END procedure\_name;

Where,

* *procedure-name* specifies the name of the procedure.
* [OR REPLACE] option allows the modification of an existing procedure.
* The optional parameter list contains name, mode and types of the parameters. **IN** represents the value that will be passed from outside and OUT represents the parameter that will be used to return a value outside of the procedure.
* *procedure-body* contains the executable part.
* The AS keyword is used instead of the IS keyword for creating a standalone procedure.

**Code :  
  
create table Customer (**

**id int(10),**

**name char(50),**

**age int(10)**

**);**

**desc table customer;**

**-- insert**

**insert into Customer (id, name, age) values**

**(1, "Anjali Gupta", 19),**

**(2, "Isha Soni", 20),**

**(3, "Aisha Singh", 21),**

**(4, "Sarth ", 20),**

**(5, "Teena Gupta", 22);**

**SELECT \* FROM customer;**

**CALL get\_info;**

**CALL get\_info(19);**

**CALL get\_info(@records);**

**SELECT @records AS totalrecord;**

**CALL get\_info(@records, 25);**

**SELECT @records as totalrecord;**

**SELECT id,name, age,**

**emp\_occupation(age) FROM customer**

**ORDER BY age;**

**-- procedure :**

**-- CREATE DEFINER=`root`@`localhost` PROCEDURE `get\_info`(in age int)**

**-- BEGIN**

**-- select \* from customer where customer.age=age;**

**-- END**

**-- Function :**

**-- CREATE DEFINER=`root`@`localhost` FUNCTION `emp\_occupation`(age int) RETURNS varchar(50) CHARSET utf8mb4**

**-- DETERMINISTIC**

**-- BEGIN**

**-- declare emp\_occupation varchar(50);**

**-- if age=20**

**-- then set emp\_occupation="Scientist"; elseif age<=25 and age>20**

**-- then set emp\_occupation="Engineer";**

**-- elseif age<20**

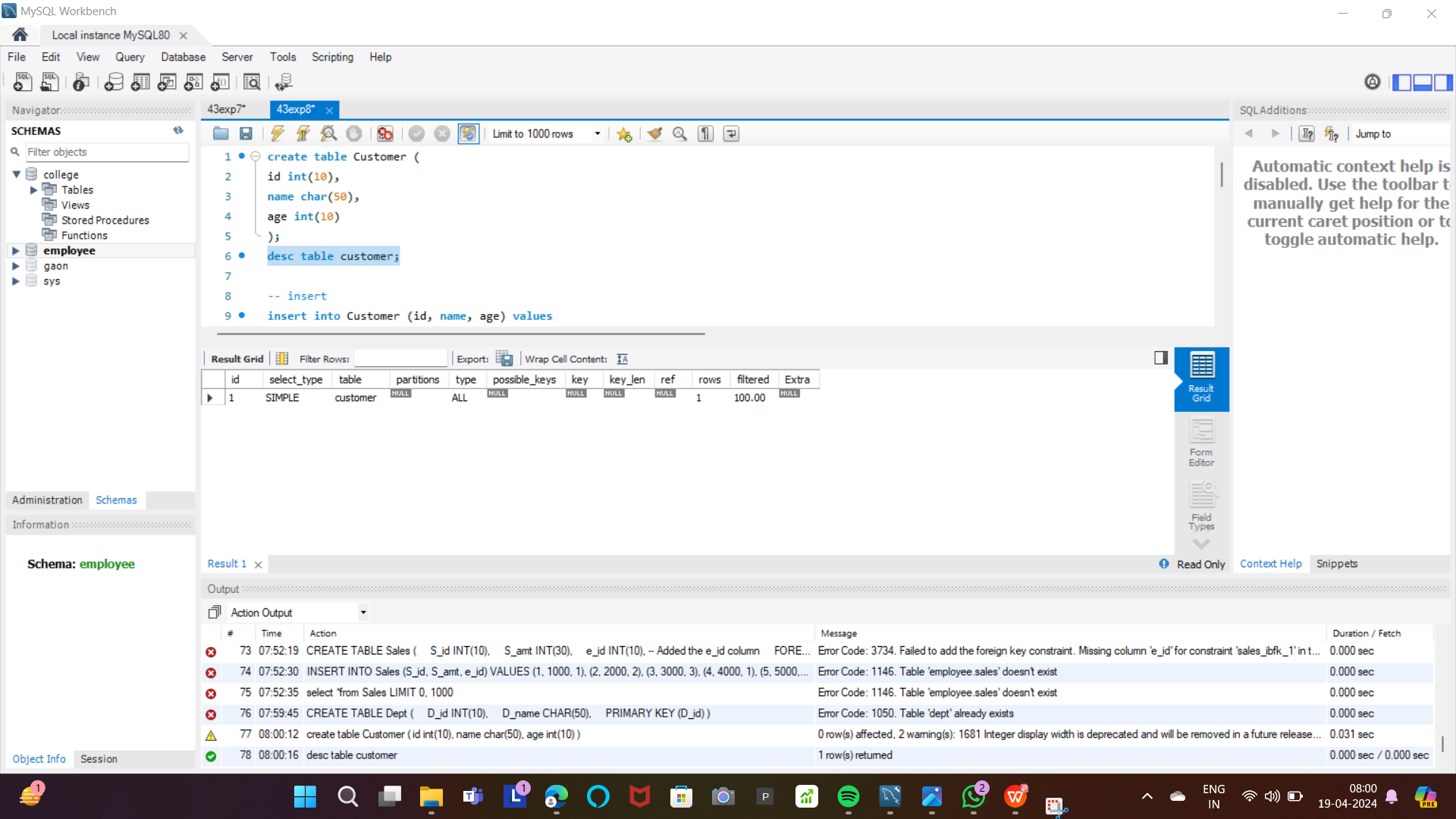
**-- then set emp\_occupation="Actor";**

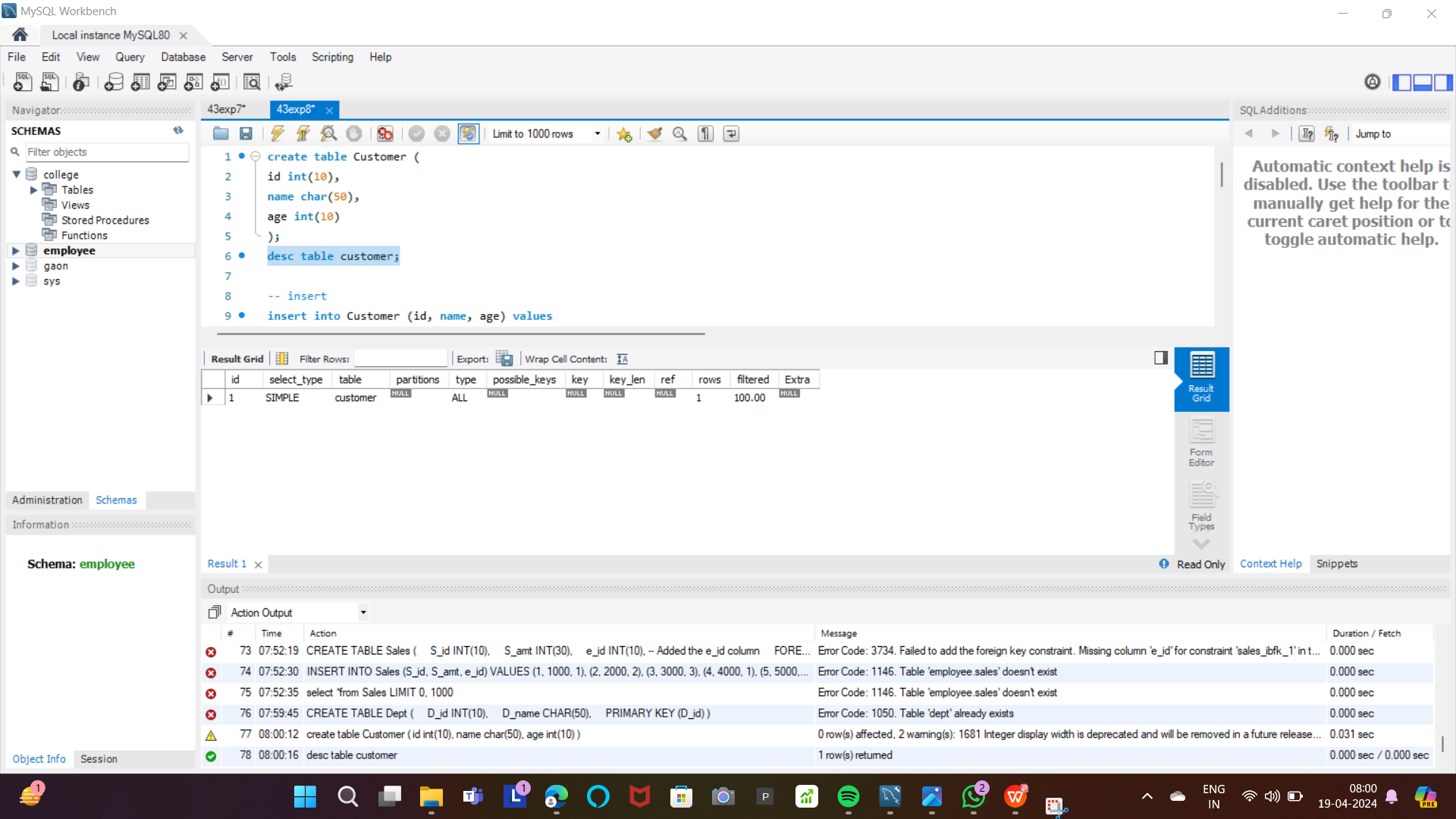
**-- end if;**

**-- RETURN (emp\_occupation);**

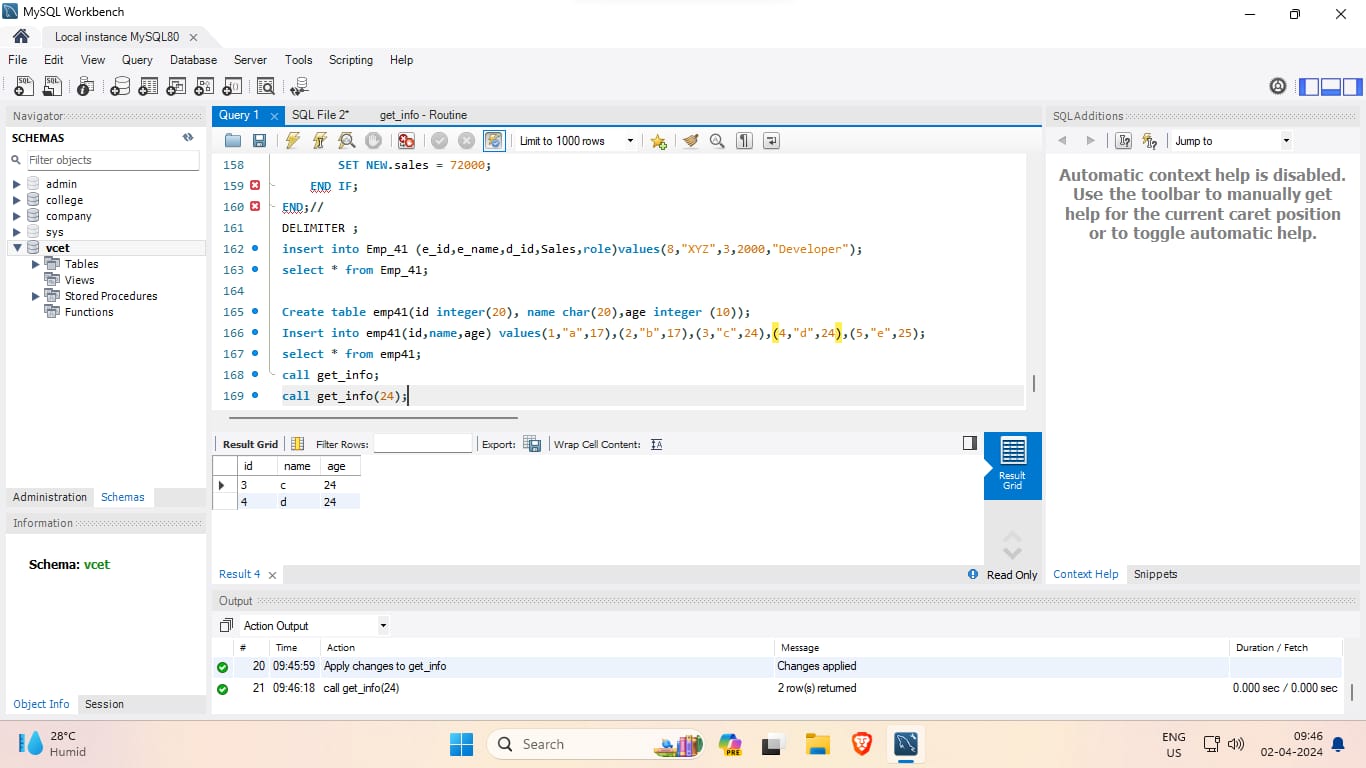
**-- END**

**Table Creation:**

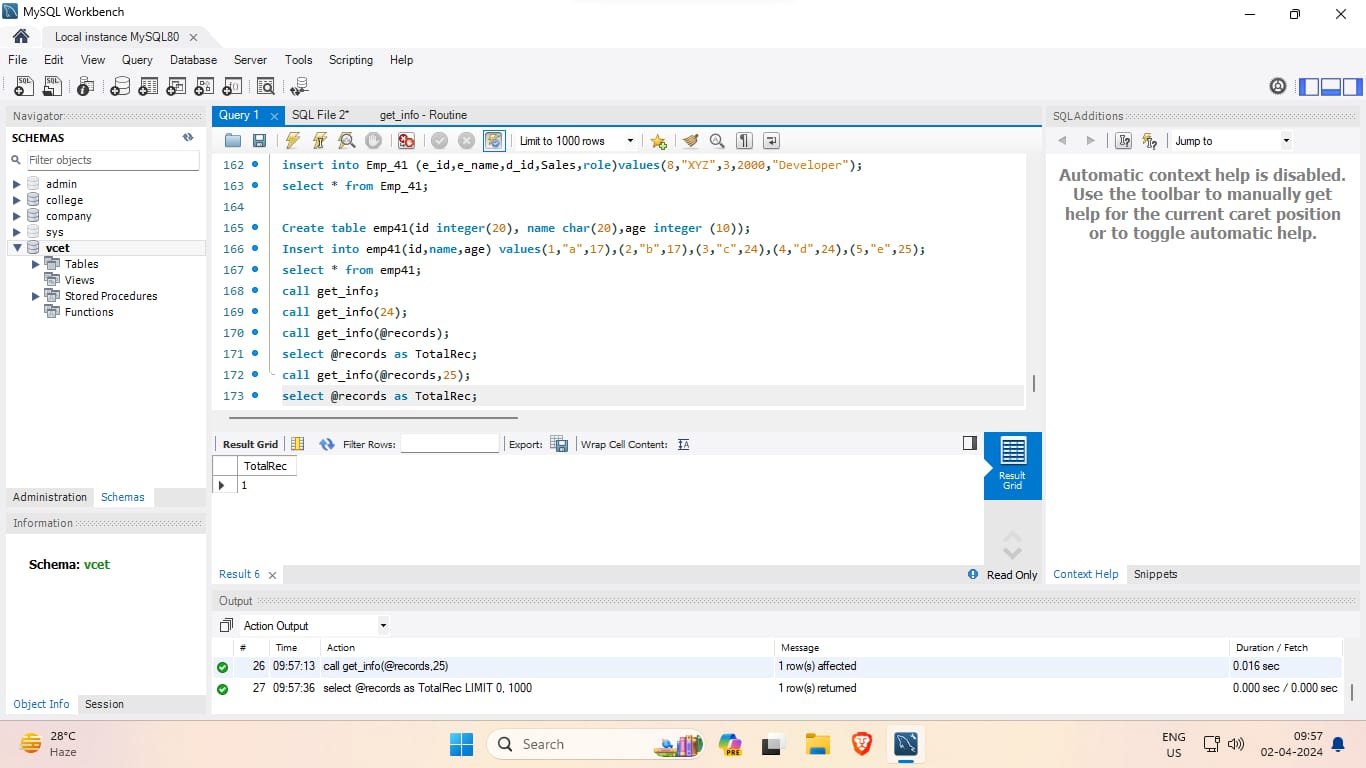




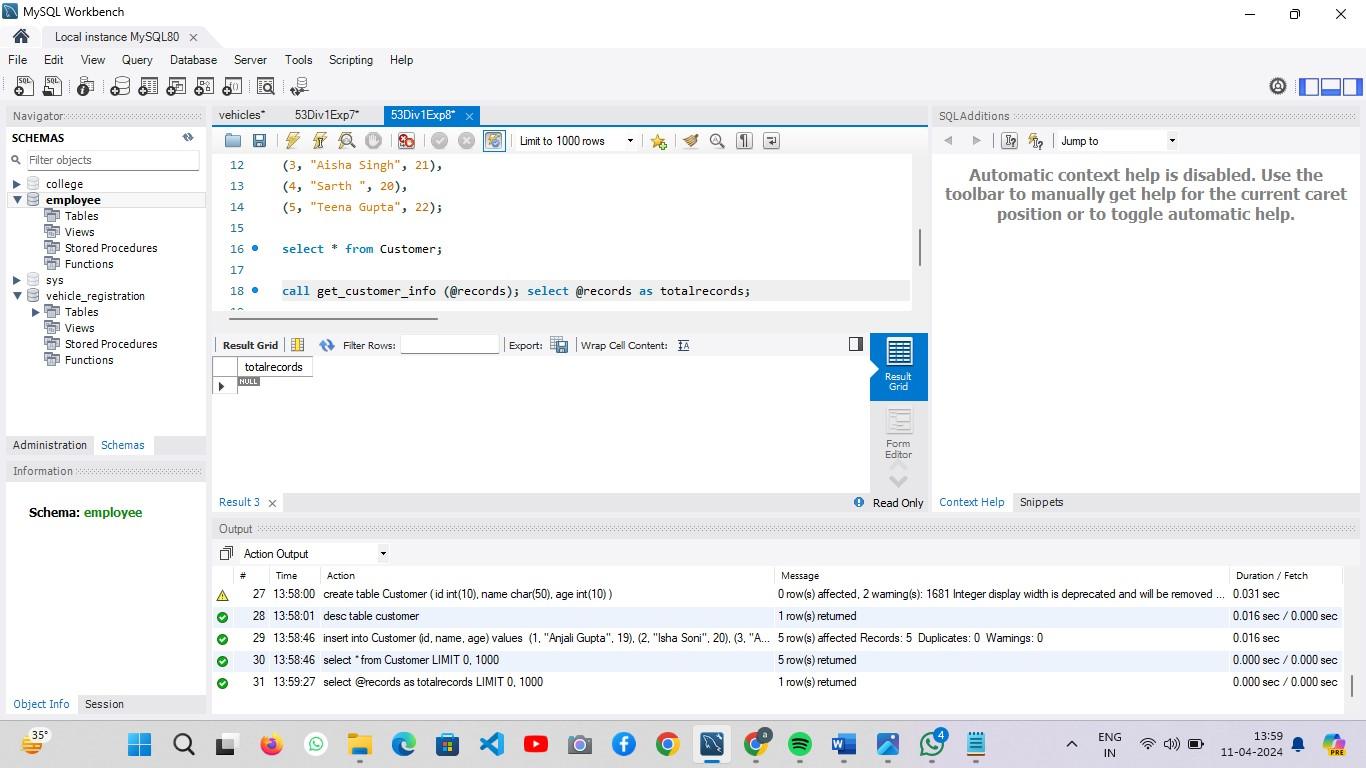
**Procedure (Type in):**



**Procedure (Type out):**



**Procedure (In-out):**



**Function:**

hy

**Conclusion:**

In this SQL experiment, we explored functions, cursors, and procedures. Functions are used for computations and return values, while cursors manage result sets for iterative processing. Procedures encapsulate sequences of SQL statements for reusability.

Mastering these constructs enhances database efficiency and application development, empowering developers with modular and scalable solutions.