| | |
|---|---|
| **Name:** Suraj Chandramauli | **Roll No: 40** |
| **Exp. No:** 9 | **Date:** 26/06/2021 |

# Procedures

**Aim:**

Demonstration of Procedures in SQL:

**I.**

a) Create a table called ‘employee’ with the following attributes

| Employee | |
|---|---|
| Field names | Type |
| ID | Number (3) |
| name | Varchar (20) |

b) Show the structure of the table.

c) Insert the following values into employee table.

| **id** | **name** |
|---|---|
| 101 | Nithya |
| 102 | Maya |

d) Display all the values in ‘employee’ table.

e) Create a procedure to insert a number as ‘id’ to the ‘employee’ table. ‘name’ field of the table should accept default value as the ‘user’ of the given computer.

f) Display all the values in ‘employee’ table.

g) Create a procedure to include a new field called ‘age’ to the ‘employee’ table and also insert values for this field in all the existing rows as 20, 30, 18 respectively.

h) Create a procedure for employee to get the employee details where age>20.

i) Create a procedure to get employee details using ‘id’.

**II.** Create a procedure to add 2 numbers.

**III.** Create a procedure to find largest among the given numbers.

**Theory:**

- A Procedure in PL/SQL is a subprogram unit that consists of a group of PL/SQL statements that can be called by name. Each procedure in PL/SQL has its own unique name by which it can be referred to and called. This subprogram unit in the Oracle database is stored as a database object.
- Procedures are standalone blocks of a program that can be stored in the database
- Call to these PLSQL procedures can be made by referring to their name, to execute the PL/SQL statements.
- It is mainly used to execute a process in PL/SQL.
- It can have nested blocks, or it can be defined and nested inside the other blocks or packages.
- It contains declaration part (optional), execution part, exception handling part (optional).
- The values can be passed into Oracle procedure or fetched from the procedure through parameters.
- These parameters should be included in the calling statement.
- A Procedure in SQL can have a RETURN statement to return the control to the calling block, but it cannot return any values through the RETURN statement.
- Procedures cannot be called directly from SELECT statements. They can be called from another block or through EXEC keyword.
- Syntax for Procedures in SQL:
- CREATE OR REPLACE PROCEDURE <procedure_name> ( <parameterl IN/OUT
- <datatype> .. . ) [ IS | AS ] <declaration_part> BEGIN <execution part> EXCEPTION
- <exception handling part> END;
- Example for Procedures in SQL is:
- CREATE OR REPLACE PROCEDURE welcome_msg (p_name IN VARCHAR2) IS BEGIN dbms_output.put_line (‘Welcome '|| p_name); END; / EXEC welcome_msg (‘Friends’);

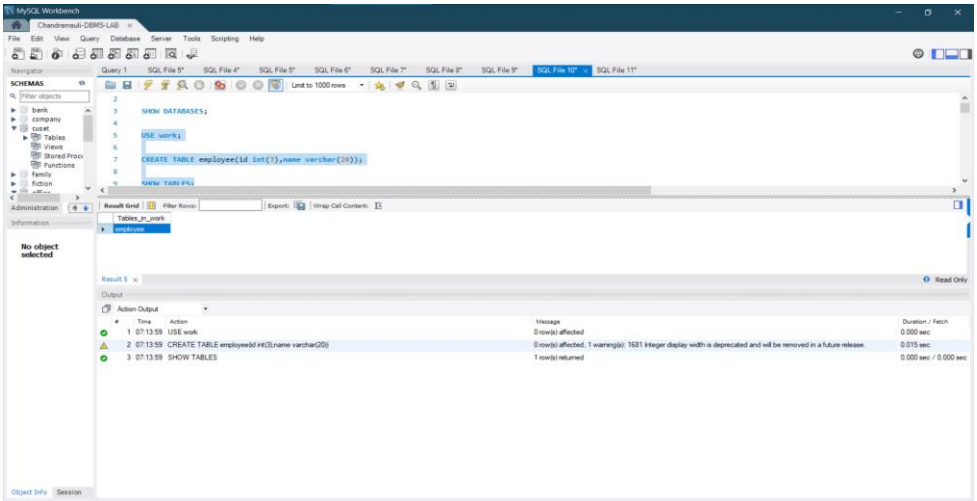
**Result:** As a result procedures in SQL is familiarized and output is verified.

**Remarks:**(To be filled by faculty)

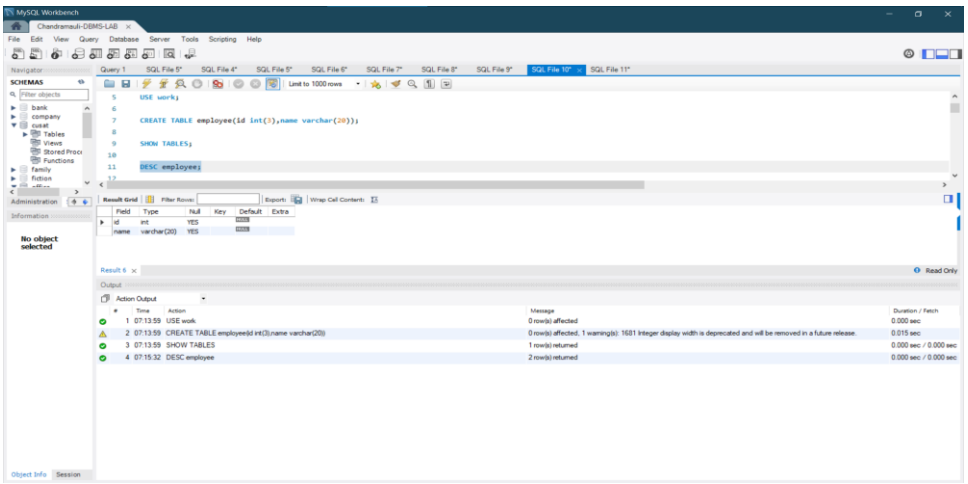
**OUTPUT:**

I)

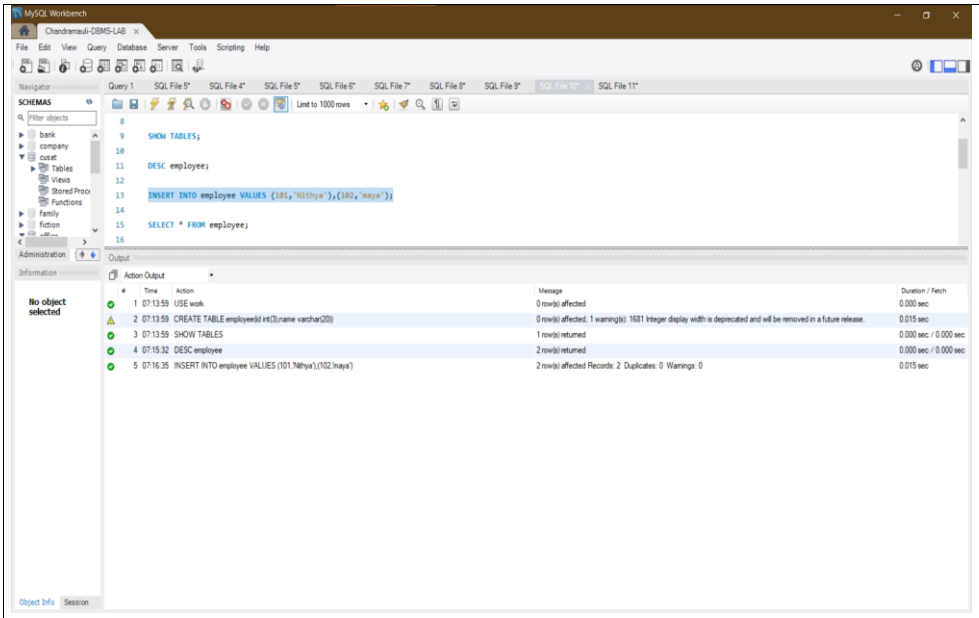
a) Create a table called ‘employee’ with attributes



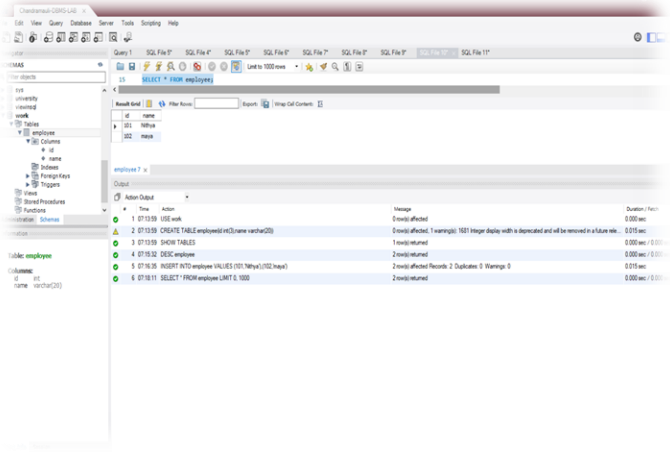

b) ) Show the structure of the table



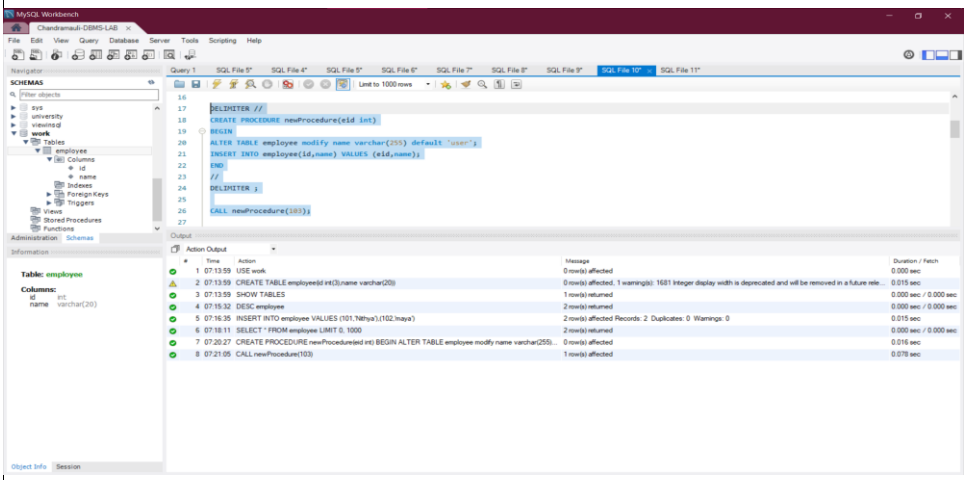

c) Insert the values into employee table.


MySQL Workbench
Chandramauli-DBMS-LAB
File Edit View Query Database Server Tools Scripting Help
Navigator
Query 1 SQL File 5* SQL File 4* SQL File 5* SQL File 6* SQL File 7* SQL File 8* SQL File 9* SQL File 10* SQL File 11*
SCHEMAS
Filter objects
bank
company
cusat
Tables
Views
Stored Procs
Functions
family
fiction
Administration
Information
No object selected
Object Info Session
Limit to 1000 rows
SHOW TABLES;
DESC employee;
INSERT INTO employee VALUES (101,'Nithya'),(102,'maya');
SELECT * FROM employee;
Output
Action Output
# Time Action Message Duration / Fetch
1 07:13:59 USE work 0 row(s) affected 0.000 sec
2 07:13:59 CREATE TABLE employee(id int(3),name varchar(20)) 0 row(s) affected, 1 warning(s): 1681 Integer display width is deprecated and will be removed in a future release. 0.015 sec
3 07:13:59 SHOW TABLES 1 row(s) returned 0.000 sec / 0.000 sec
4 07:15:32 DESC employee 2 row(s) returned 0.000 sec / 0.000 sec
5 07:16:35 INSERT INTO employee VALUES (101,'Nithya'),(102,'maya') 2 row(s) affected Records: 2 Duplicates: 0 Warnings: 0 0.015 sec

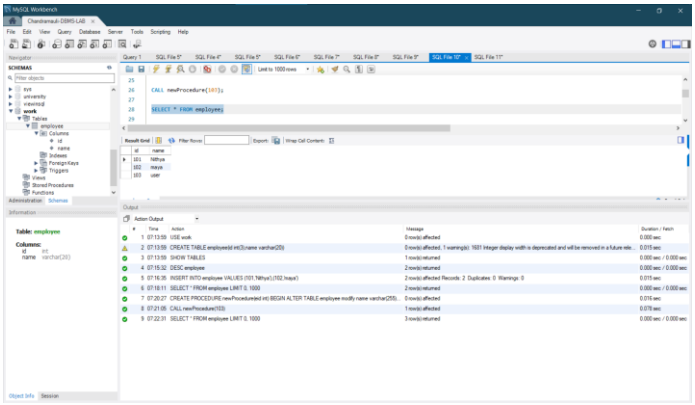
d) Display all the values in ‘employee’ table



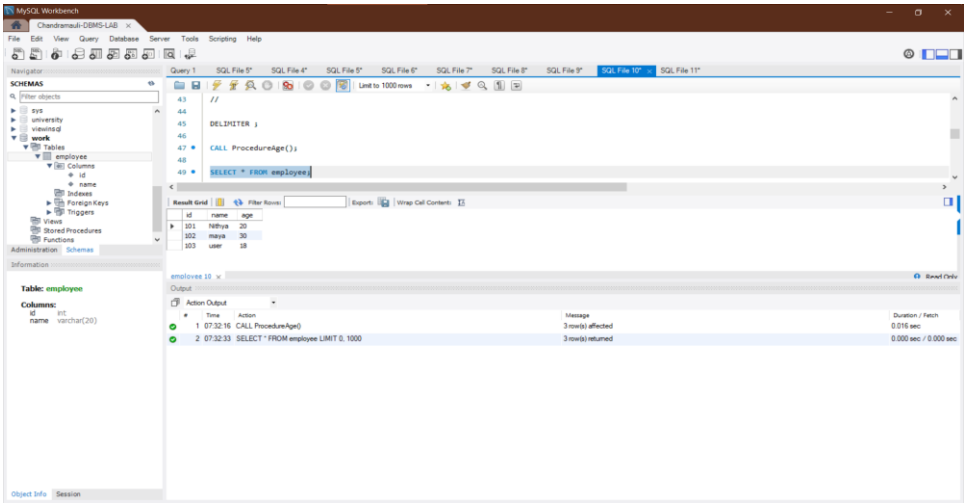
e) ) Create a procedure to insert a number as ‘id’ to the ‘employee’ table. ‘name’ field of the table should accept default value as the ‘user’ of the given computer.



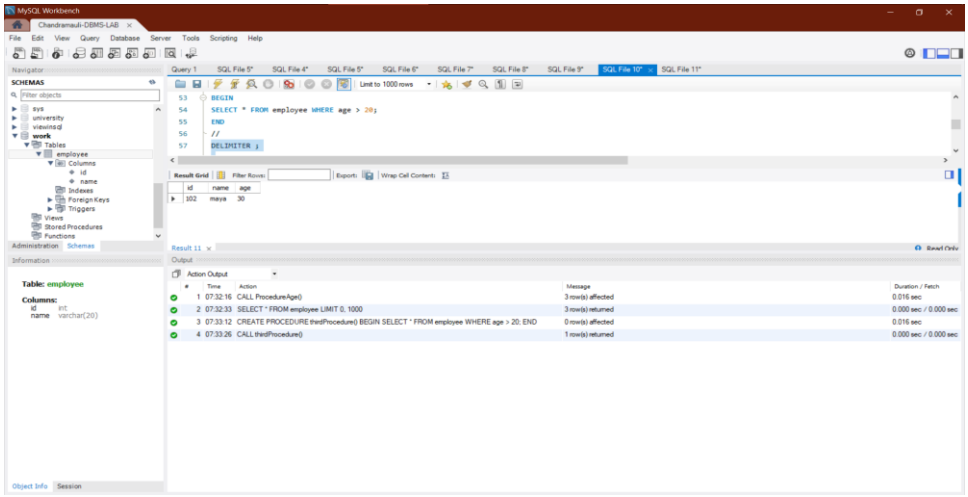

f) Display all the values in ‘employee’ table



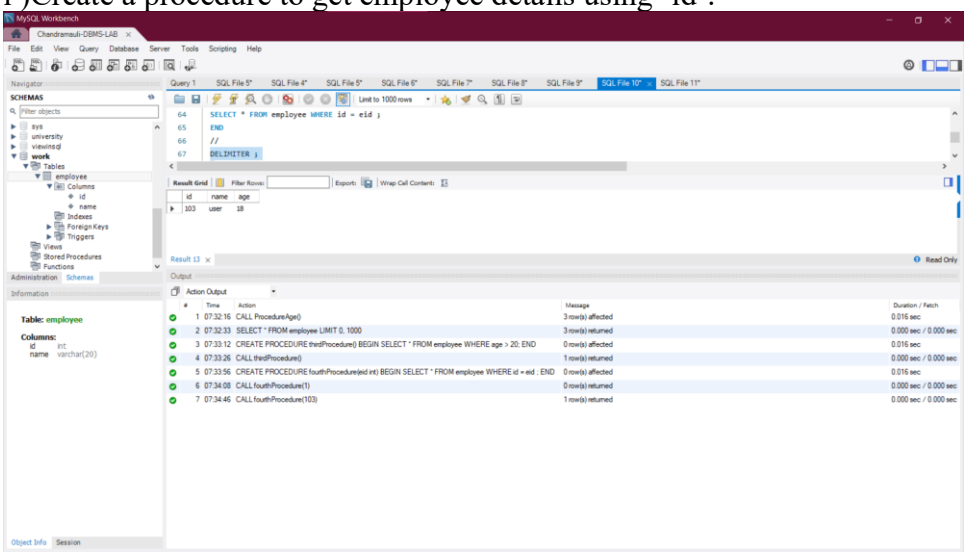
g) Create a procedure to include a new field called ‘age’ to the ‘employee’ table and also insert values for this field in all the existing rows as 20, 30, 18 respectively.



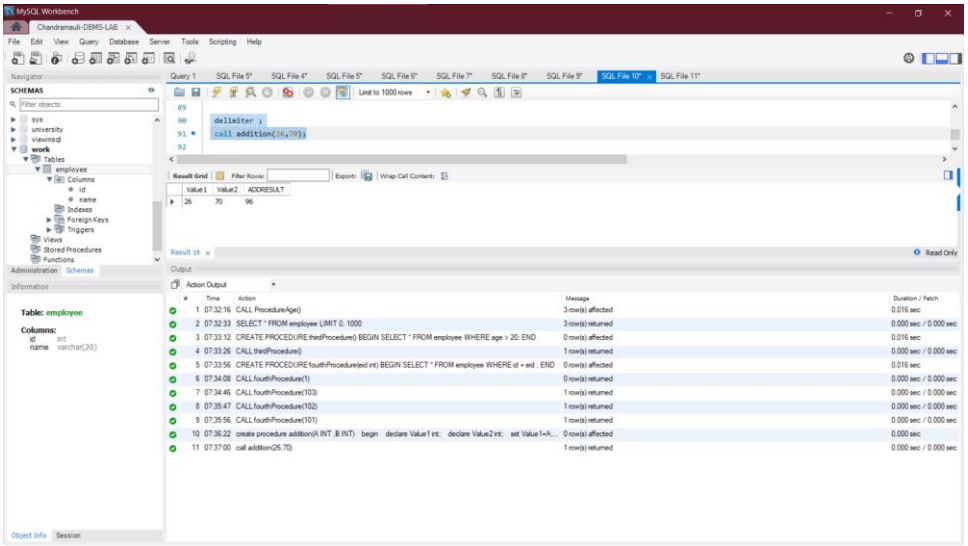

h) Create a procedure for employee to get the employee details where age>20.



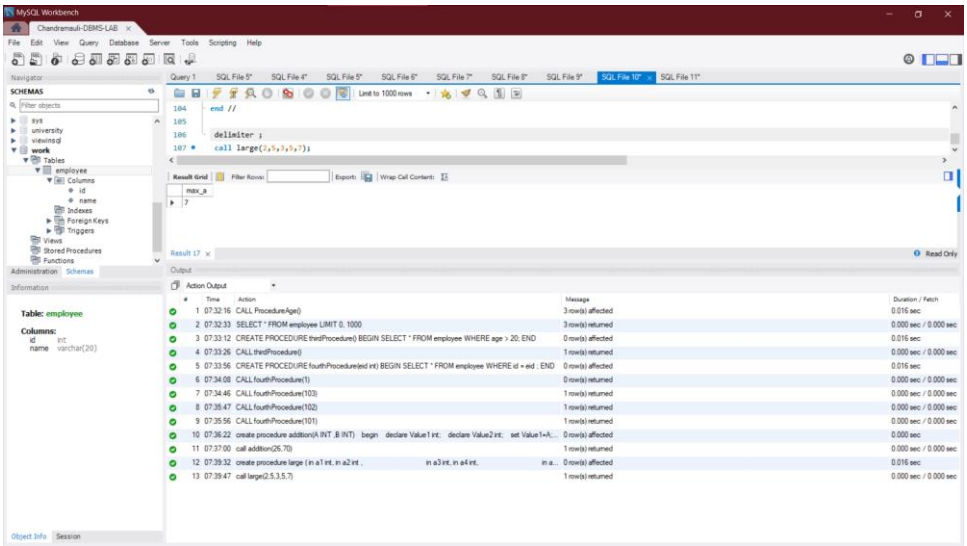

i )Create a procedure to get employee details using ‘id’.



(II) Create a procedure to add 2 numbers.




(III) Create a procedure to find largest among the given numbers.




(LEFT SIDE OF A PAGE)

| | |
|---|---|
| **Name:** Suraj Chandramauli | **Roll No:40** |
| **Exp. No:**10 | **Date: 26**/06/2021 |

# Functions

**Aim:**

Demonstration of Functions in SQL:

Consider the ‘bank’ table and do the following operations.

| BANK | | |
|---|---|---|
| Acc_no | B_name | balance |
| 101 | SBI | 25000 |
| 102 | SBI | 5000 |
| 103 | FEDRAL | 10000 |
| 104 | AXIS | 15000 |
| 105 | CANARA | 50000 |

a) Create a function for withdrawing money from an account in a bank management system which uses bank table. The minimum balance the account should hold is 500.

b) Display all the values in ‘bank’ table.

c) Create a function for depositing money to an account in a bank management system which uses bank table.

d) Display all the values in ‘bank’ table.

**Theory:**

- Functions is a standalone PL/SQL subprogram. Like PL/SQL procedure, functions have a unique name by which it can be referred. These are stored as PL/SQL database objects. Below are some of the characteristics of functions.
- Functions are a standalone block that is mainly used for calculation purpose. Function use RETURN keyword to return the value, and the datatype of this is defined at the time of creation. A Function should either return a value or raise the exception, i.e. return is mandatory in functions.
- Function with no DML statements can be directly called in SELECT query whereas the function with DML operation can only be called from other PL/SQL blocks. It can have nested blocks, or it can be defined and nested inside the other blocks or packages. It contains declaration part (optional), execution part, exception handling part (optional).
- The values can be passed into the function or fetched from the procedure through the parameters. These parameters should be included in the calling statement.
- A PLSQL function can also return the value through OUT parameters other than using RETURN. Since it will always return the value, in calling statement it always accompanies with assignment operator to populate the variables.
- Syntax for Functions in SQL is:

CREATE OR REPLACE FUNCTION <procedure_name> ( <parameterl IN/OUT <datatype> ) RETURN <datatype> [ IS | AS ] <declaration_part> BEGIN <execution part> EXCEPTION <exception handling part> END;

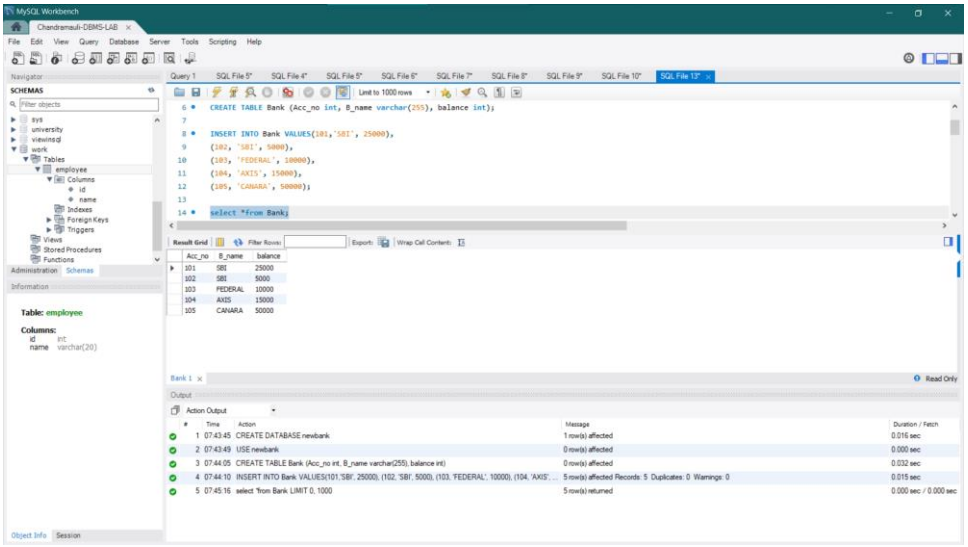
- Example for Functions in SQL is:

  CREATE OR REPLACE FUNCTION welcome_msgJune ( p_name IN VARCHAR2) RETURN VAR.CHAR2 IS BEGIN RETURN (‘Welcome ‘|| p_name); END; / DECLARE lv_msg VARCHAR2(250); BEGIN lv_msg := welcome_msg_func (‘TRIKKZ’); dbms_output.put_line(lv_msg); END; SELECT welcome_msg_func(‘TRIKKZ:’) FROM DUAL;

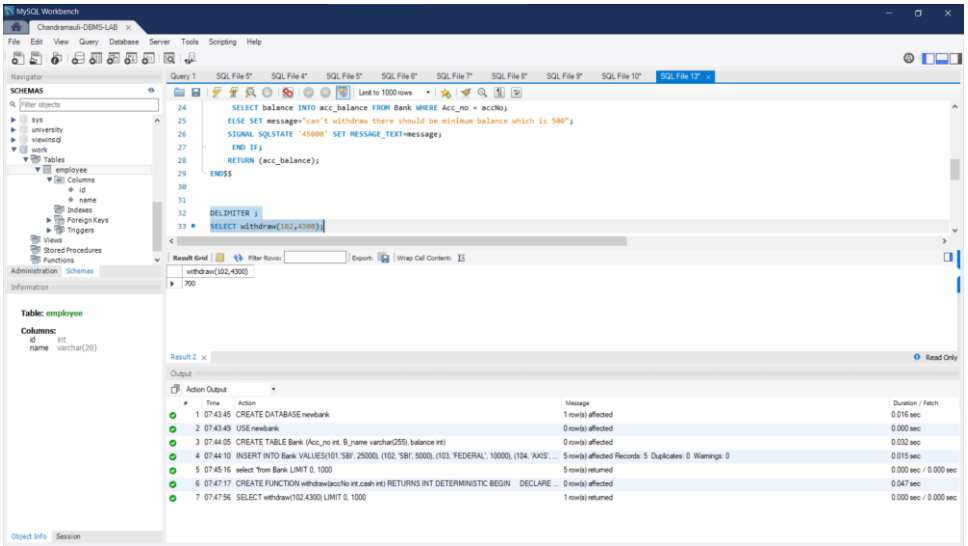
**Result:** As a result functions in SQL are familiarized and output is verified.

**Remarks:**(To be filled by faculty)

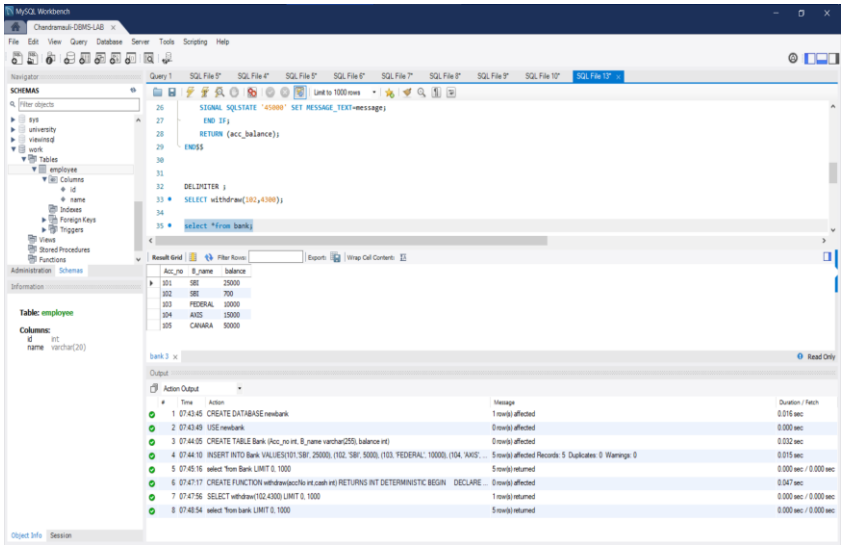
**OUTPUT:**



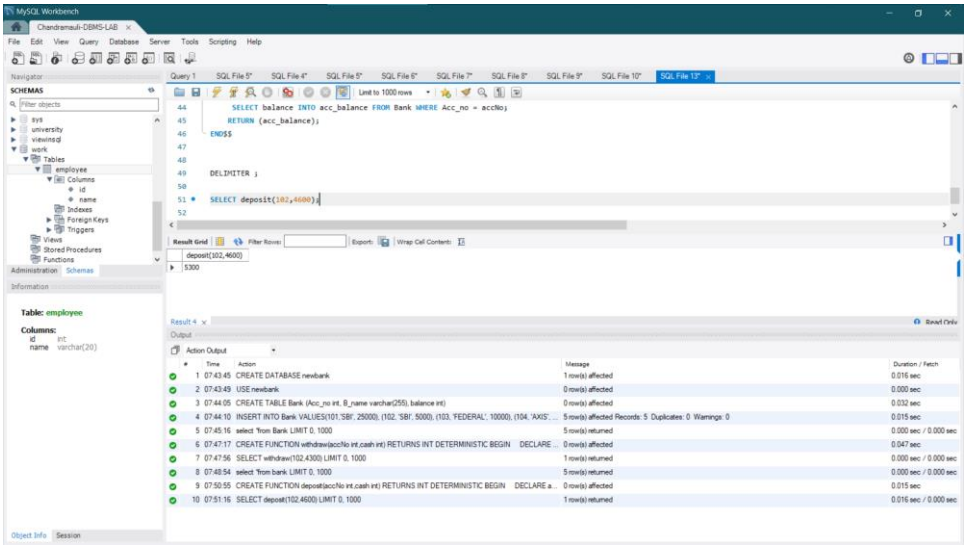

a) Create a function for withdrawing money from an account in a bank management system which uses bank table. The minimum balance the account should hold is 500.



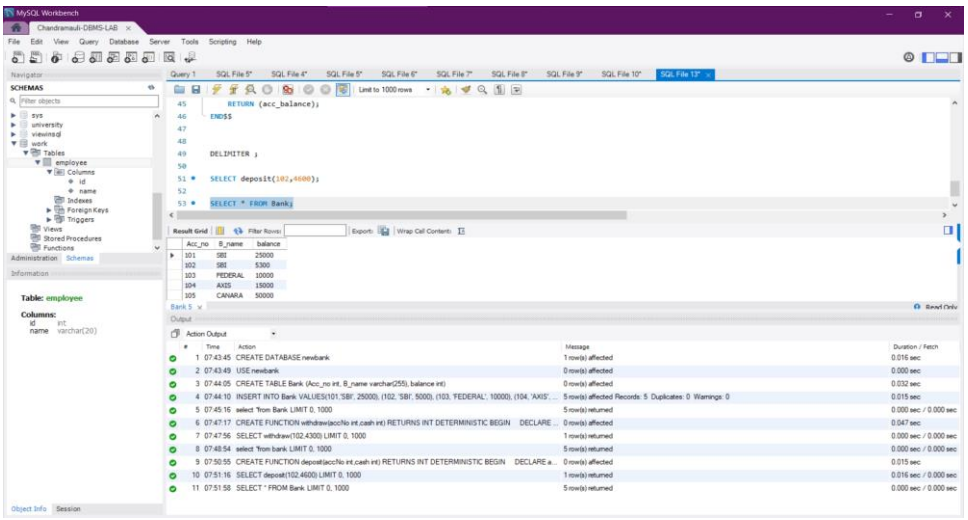
b) Display all the values in ‘bank’ table




c) ) Create a function for depositing money to an account in a bank management system which uses bank table




d) Display all the values in ‘bank’ table.



(LEFT SIDE OF A PAGE)

| | |
|---|---|
| **Name:** Suraj Chandramauli | **Roll No:40** |
| **Exp. No:**11 | **Date: 26**/06/2021 |

# Cursor

**Aim:**

Demonstration of Cursors in SQL:

Consider the ‘student table’ of a particular class and do the following operations.

| Roll_No | Name | M1 | M2 | M3 | Total | Percentage | grade |
|---|---|---|---|---|---|---|---|
| 15 | Jenny | 20 | 30 | 20 | 0 | 0 | 0 |
| 41 | Reena | 98 | 90 | 85 | 0 | 0 | 0 |
| 22 | Leena | 40 | 45 | 60 | 0 | 0 | 0 |
| 23 | Boban | 50 | 30 | 20 | 0 | 0 | 0 |

**I. Explicit cursor**

a) Create a cursor for calculating the total marks and percentage of the student, grade of the student in a student management system which uses a student table. Grade the student according to the following rules.

| **Total Marks** | **Grade** |
|---|---|
| >=250 | Distinction |
| 180-250 | First class |
| 120-179 | Second class |
| 180-80 | Third  class |
| <80 | fail |

b) Display all the details of student table.

c) Create a cursor to find student who got the highest mark from the ‘student’ table and display the particular student details in the following format. Remark for a highest scored student is ‘topper of the class’.

**Roll no:**

**Student name:**

**Total marks:**

**Grade:**

**Remarks:**

d) Display all the details of student table.

**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 2 types of cursors: Implicit cursor and Explicit cursor.
- Implicit Cursors are also known as Default Cursors of SQL SERVER. These Cursors are allocated by SQL SERVER when the user performs DML operations.
- 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.
- Explicit Cursors are created by users whenever the user requires them. Explicit Cursors are used for Fetching data from Table in Row-By-Row manner.
- 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
- Declaring the cursor simply means to create one named context area for the 'SELECT' statement that is defined in the declaration part. The name of this context area is same as the cursor name.
- Opening the cursor will instruct the PL/SQL to allocate the memory for this cursor. It will make the cursor ready to fetch the records.
- In this process, the 'SELECT' statement is executed and the rows fetched is stored in the allocated memory. These are now called as active sets. Fetching data from the cursor is a record-level activity that means we can access the data in a record-by-record way. Each fetch statement will fetch one active set and holds the information of that particular record. This statement is same as 'SELECT' statement that fetches the record and assigns to the variable in the 'INTO' clause, but it will not throw any exceptions
- Once all the record is fetched now, we need to close the cursor so that the memory allocated to this context area will be released.
- Syntax for cursor is:

  DECLARE CURSOR <cursor_name> IS <SELECT statement^> <cursor_variable declaration> BEGIN OPEN <cursor_name>; FETCH <cursor_name> INTO

<cursor_variable>; . . CLOSE <cursor_name>; END;

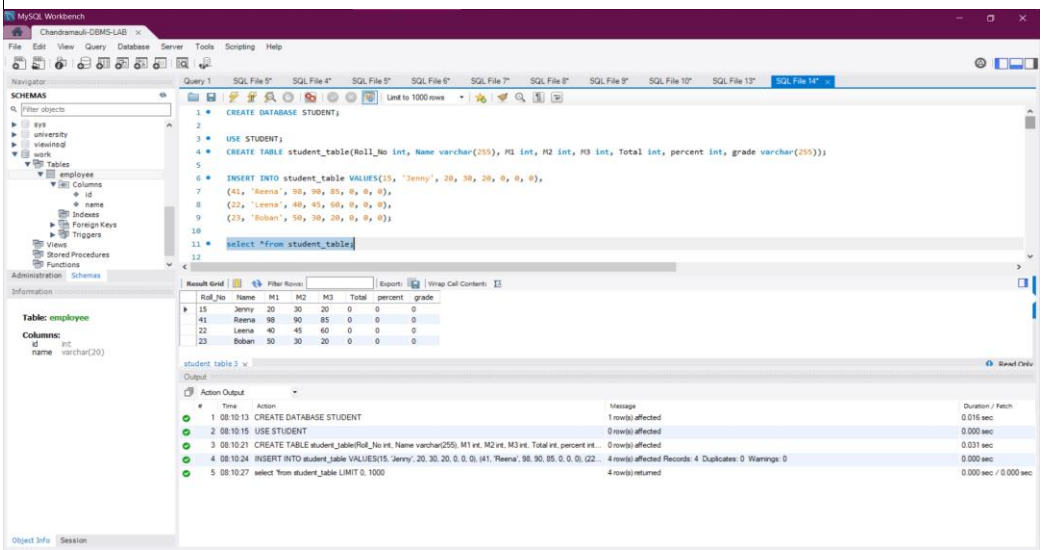
- Example for cursor is:

  DECLARE CURSOR TRIKKZ_det IS SELECT emp_name FROM emp; lv_emp_name emp.emp_name%type; BEGIN OPEN TRIKKZ_det; LOOP FETCH TRIKKZ_det INTO lv_emp_name; IF TRIKKZ_det%NOTFOUND THEN EXIT; END IF; Dbms_output.put_line('Employee Fetched:'||lv_emp_name); END LOOP; Dbms_output.put_line('Total rows fetched is'||guru99_det%R0WCOUNT); CLOSE TRIKKZ_det; END: /
- "FOR LOOP" statement can be used for working with cursors. We can give the cursor name instead of range limit in the FOR loop statement so that the loop will work from the first record of the cursor to the last record of the cursor. The cursor variable, opening of cursor, fetching and closing of the cursor will be done implicitly by the FOR loop.
- Syntax for cursor using FOR LOOP is:
- DECLARE CURSOR <cursor_name> IS <SELECT statement>; BEGIN FOR I IN
- <cursor_name> LOOP . . END LOOP; END;
- Example for cursor using FOR LOOP is:
- DECLARE CURSOR TRIKKZ_det IS SELECT emp_name FROM emp; BEGIN FOR
- lv_emp_name IN TRIKKZ_det LOOP Dbms_output.put_line('Employee Fetched:'||lv_emp_name.emp_name); END LOOP; END; /

**Result:** As a result cursors in SQL are familiarized and output is verified.

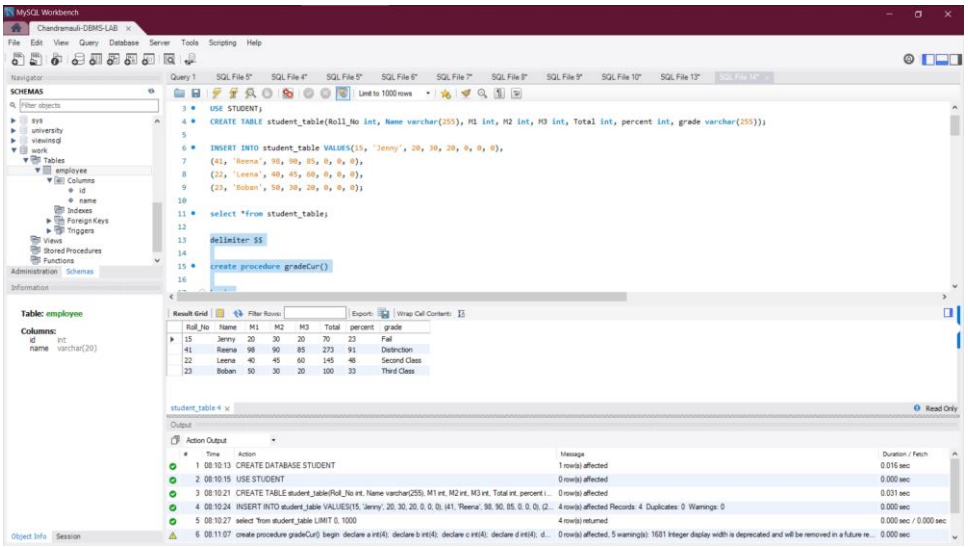
**Remarks:**(To be filled by faculty)

**OUTPUT:**

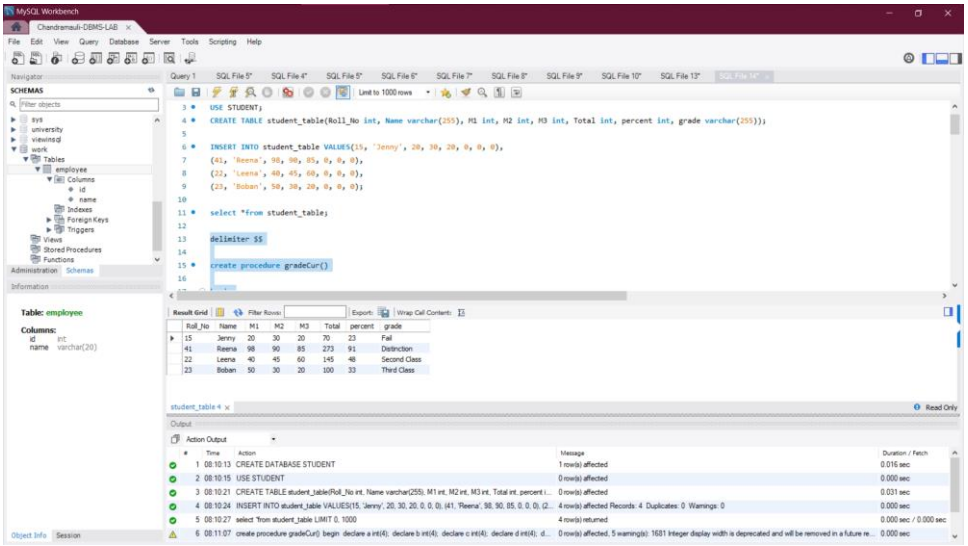



I) Explicit cursor

a) Create a cursor for calculating the total marks and percentage of the student, grade of the student in a student management system which uses a student table.



b) Display all the details of student table.




c) Create a cursor to find student who got the highest mark from the ‘student’ table and display the particular student details in the following format. Remark for a highest scored student is ‘topper of the class’.

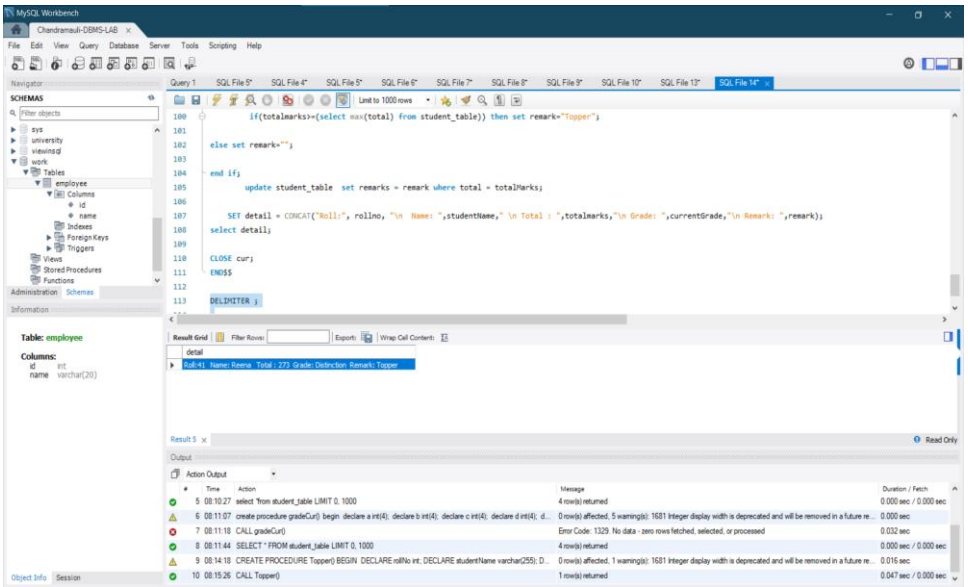
Roll no:

Student name:

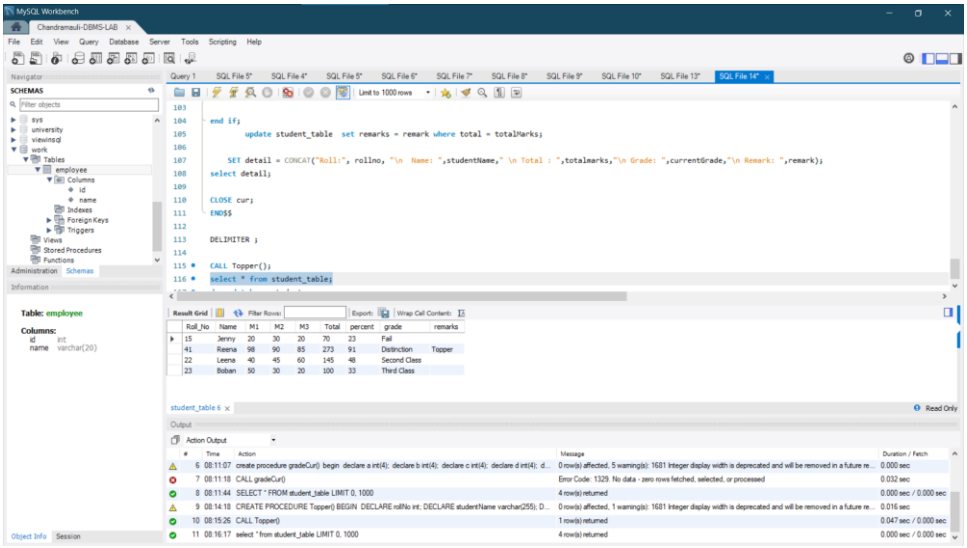
Total marks:

Grade:

Remarks:




d) Display all the details of student table




(LEFT SIDE OF A PAGE)

| | |
|---|---|
| **Name:** Suraj Chandramauli | **Roll No: 40** |
| **Exp. No:** 12 | **Date: 26**/06/2021 |

# Triggers

**Aim:**

Demonstration of trigger sin SQL is:

a) Create a table called 'reservations' with following fields.

| reservations | | |
|---|---|---|
| name | Null | Type |
| Fight_id | Not null | Char(6) |
| Customer_Phone | Not null | number |

b) Create another table called 'flights' with the following fileds. Make 'flight_id' the primary key in 'flights' table.

| flights | | |
|---|---|---|
| Flight_id | Not null | Char(6) |
| Seats | Not null | number |

c) Show the structure of 'reservations' table.

d) Show the structure of 'flights' table.

e) Insert the following rows into 'flights' table.

| Flight_id | Seats |
|---|---|
| ACO529 | 120 |
| Aco530 | 0 |

f) Display all the details of 'flights' table.

g) Create a trigger **RES_TRG** that will ensure that when a new row is inserted into the **RESERVATIONS** table, the flight id is in the **FLIGHTS** table and that the number of seats on this flight, SEATS is greater than 0.

Here are the details of how the trigger should behave:

- If flight id is not in the flights table it should raise application error 'Invalid flight id'.
- If flight id is in the flights table, (for example AC0529) but SEATS = 0, then it should raise application error 'Flight AC0529 has no seats left'.
- If flight id is in the flights table and SEATS > 0, then it should update the appropriate row in flights table by setting SEATS = SEATS – 1 for this flight.

**Theory:**

- Triggers are stored programs that are fired by Oracle engine automatically when DML Statements like insert, update, delete are executed on the table or some events occur. The code to be executed in case of a trigger can be defined as per the requirement. You can choose the event upon which the trigger needs to be fired and the timing of the execution. The purpose of trigger is to maintain the integrity of information on the database.
- Uses of Triggers are 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.
- Syntax for creating a trigger in SQL 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;
- 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.
- Example for triggers in SQL is:

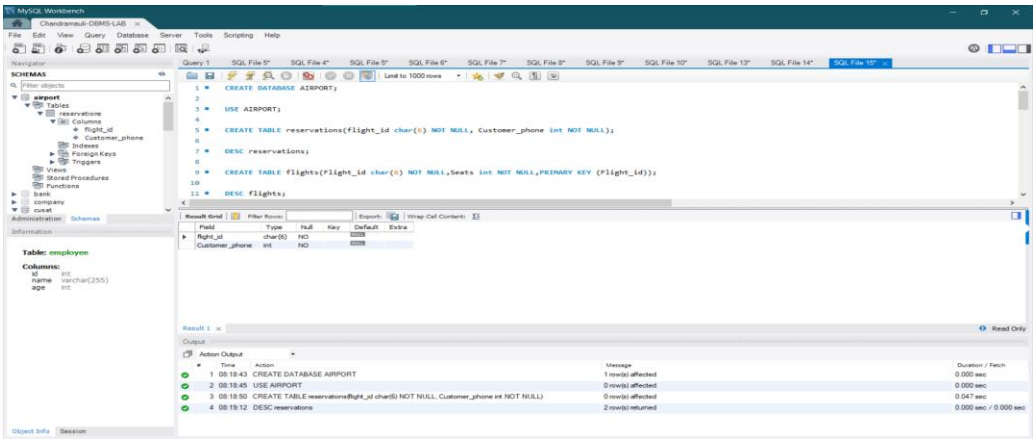
  CREATE OR REPLACE TRIGGER display_salary_changes BEFORE DELETE OR INSERT OR UPDATE ON customers FOR EACH ROW WHEN (NEW.ID > 0) DECLARE sal_diff number; BEGIN sal_diff := :NEW.salary - :OLD.salary; dbms_output.put_line('Old salary: ' || :OLD.salary); dbms_output.put_line('New salary: ' || :NEW.salary); dbms_output.put_line('Salary difference: ' || sal_diff); END; /

**Result:** As a result triggers in SQL is familiarized and output is verified.

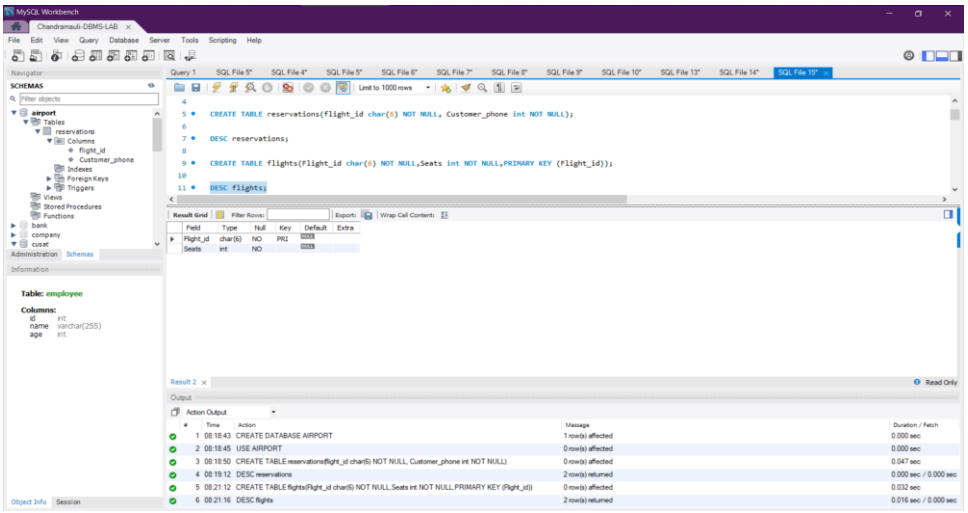
**Remarks:**(To be filled by faculty)

**OUTPUT:**

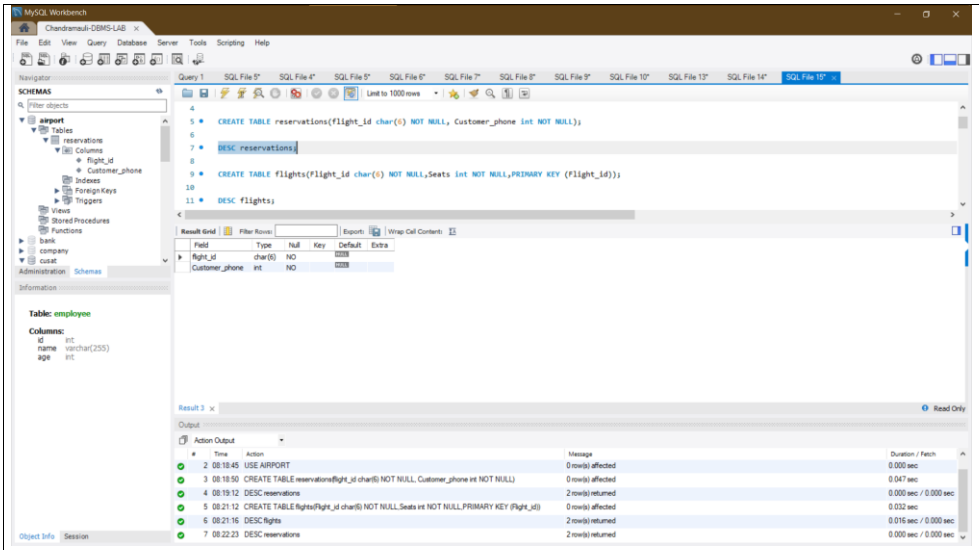
a)Create a table called ‘reservations’ with fields.



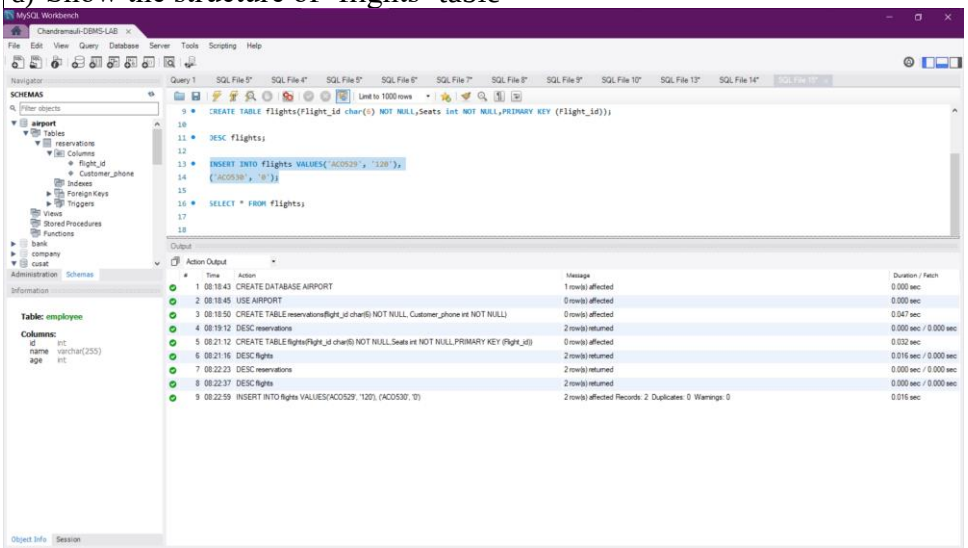

b) Create another table called ‘flights’ with fields. Make ‘flight_id’ the primary key in ‘flights’ table.



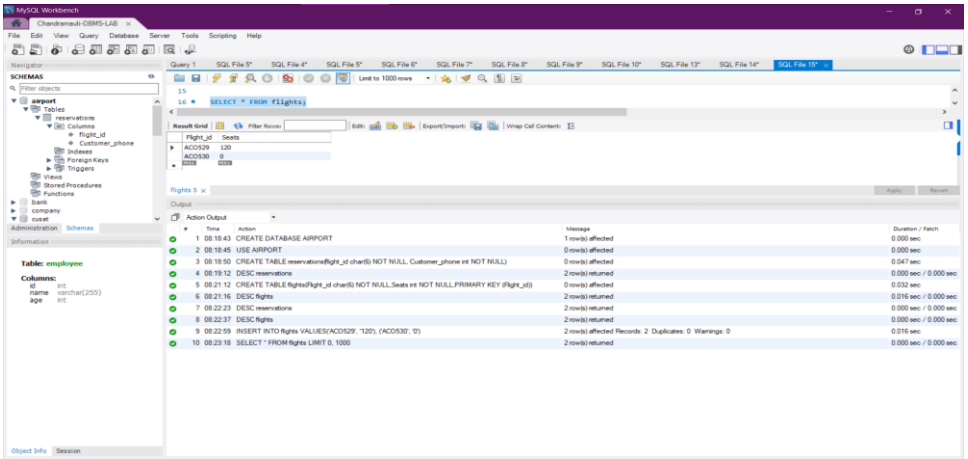

c) Show the structure of ‘reservations’ table



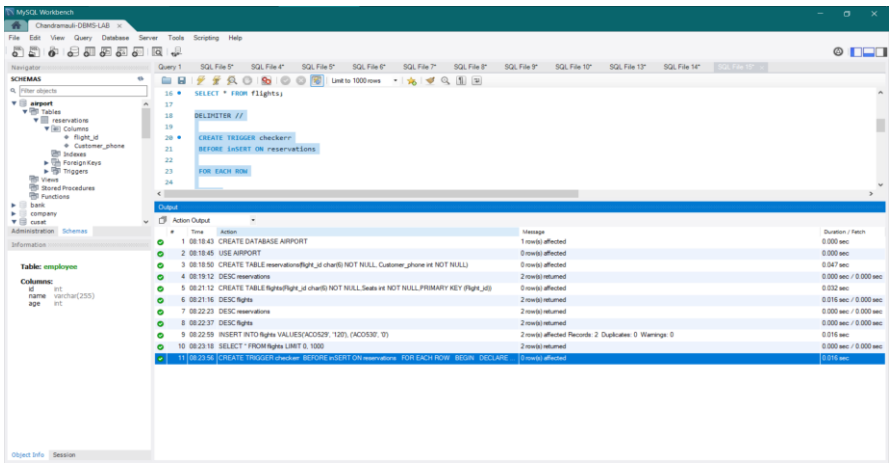

d) Show the structure of ‘flights’ table



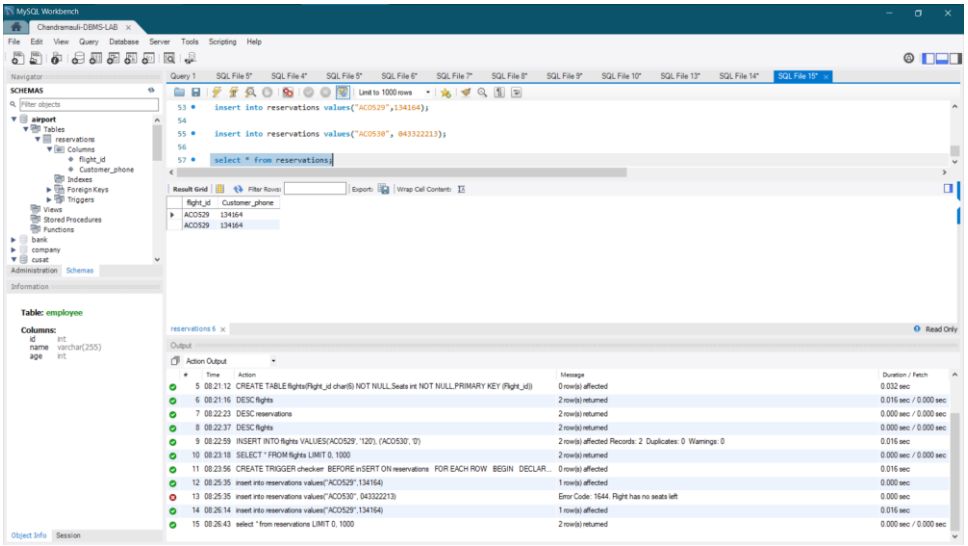
e) Insert the rows into ‘flights’ table.




f) Display all the details of ‘flights’ table




g) Create a trigger **RES_TRG** that will ensure that when a new row is inserted into the **RESERVATIONS** table, the flight id is in the **FLIGHTS** table and that the number of seats on this flight, SEATS is greater than 0.




(LEFT SIDE OF A PAGE)