Department of Computer Science

Gujarat University

Certificate

Roll No: **30** Seat No: _____

This is to certify that Mr./Ms. ***Rathod Ajinkya*** student of MCA Semester – II has duly completed his/her term work for the semester ending in June 2020, in the subject of ***Relational Database Management System-II*** towards partial fulfilment of his/her Degree of Masters in Computer Applications.

Date of Submission: **01/07/2020** Internal Faculty

Head of Department

Department of Computer Science
Rollwala Computer Centre
Gujarat University

MCA –II

Subject: - RELATIONAL DATABASE MANAGEMENT SYSTEMS - II

Name :- Ajinkya Rathod

Roll No.: - 30 **Exam Seat No.: -**_____

Sr. No.	Contents	Pg. No	Date	Signature
	Assignment-1		01/07/2020	
	File Storage, Query Processing			
	And Transaction			
	Assignment-2	08	01/07/2020	
	Transaction Management and Recovery			

D E P A R T M E N T O F C O M P U T E R S C I E N C E

R O L L W A L A C O M P U T E R C E N T R E

G U J A R A T U N I V E R S I T Y

M.C.A. – II

R O L L NO : 30

N A M E : Ajinkya Rathod

S U B J E C T : RELATIONAL DATABASE MANAGEMENT SYSTEM -II

NO.	TITLE	PAGE NO.	DATE	SIGN
	Google Classroom Assignments(Cursors Practice)	23		
	1. Write a cursor to fetch employee Details			
	2. Write a cursor to find all employees in Dept passed as a argument			
	3. . Write a cursor to increase salary of employees acc. to Dept.			
	ASSIGNMENT 1	47		
01	Write PL/SQL procedures to delete rows from emp table		01/07/20	
	based on a parameter and display no. of employees			
	deleted.			
02	Write PL/SQL procedures to perform withdraw and		01/07/20	
	deposit operations.			
03	Write Pl/SQL procedures to perform withdraw and		01/07/20	
	transfer operations.			
04	Write Pl/SQL procedures to retrieve all the details of the		01/07/20	
	employees with grade 5 and to apply pattern matching			
	through the procedure.			
05	Write functions named SUMMATION() to add 2 numbers		01/07/20	
	and other to concate 2 strings.			
06	Write 3 procedures respectively to check if the number is		01/07/20	
	>0, to check if the entered date is > sysdate and to check if			
	the name entered is in uppercase. And write one final to			
	call procedure them as a, b and c respectively.			
07	Write a procedure to display first n fibonacci numbers.		01/07/20	
08	Write a procedure to find simple interest.		01/07/20	
09	Write a procedure to reverse entered number.		01/07/20	

D E P A R T M E N T O F C O M P U T E R S C I E N C E

R O L L W A L A C O M P U T E R C E N T R E

G U J A R A T U N I V E R S I T Y

M.C.A. – II

R O L L NO : 30

N A M E : Ajinkya Rathod

S U B J E C T : RELATIONAL DATABASE MANAGEMENT SYSTEM -II

10	Write a procedure and find its equivalent roman value.		01/07/20	
11	Write a program to enter a number and find addition of		01/07/20	
	each digit of that number using function.			
12	Write a program to input your birthdate and should return		01/07/20	
	your age in year, month and days.			
13	Write a program to reverse the string.		01/07/20	
14	Write a program to convert the string into toggle case.		01/07/20	
15	Write a PL/SQL block to convert given numbers into text		01/07/20	
	Words.			
	ASSIGNMENT 2	77		
	GENERAL PL/SQL BLOCKS	77		
01	WAP to input two numbers and find out what is the		01/07/20	
	Output of all arithmetic operations.			
02	WAP to input rollno and three subject marks. Find out		01/07/20	
	total, percentage, result and grade for the student from the			
	entered data.			
03	WAP to print first 10 odd numbers using for loop.		01/07/20	
04	WAP to print prime numbers upto 10 using while loop.		01/07/20	
05	WAP to input three nos and find out maximum and		01/07/20	
	minimum from it.			
06	WAP to input empno from keybord. Check whether inputed		01/07/20	
	empno exist in emp table or not. If not give error message			
	otherwise display name and salary of that employee			
07	WAP to insert record in Customer table		01/07/20	
	CURSORS	93		

D E P A R T M E N T O F C O M P U T E R S C I E N C E

R O L L W A L A C O M P U T E R C E N T R E

G U J A R A T U N I V E R S I T Y

M.C.A. – II

R O L L NO : 30

N A M E : Ajinkya Rathod

S U B J E C T : RELATIONAL DATABASE MANAGEMENT SYSTEM -II

01	Create a cursor for the emp table. Produce the output in	93	01/07/20	
	following format: {empname} employee working in			
	department {deptno} earns Rs. {salary}.			
02	Create a cursor for updating the salary of emp working in		01/07/20	
	deptno 10 by 20%. And display no. of rows affected.			
03	Create a cursor for updating the salary of emp working in deptno 10 by 20%. Use explicit cursor		01/07/20	
04	WAP that will display the name, department and salary of		01/07/20	
	the first 10 employees getting the highest salary			
05	WAP using parameterized cursor to display all the		01/07/20	
	information of employee living in specified city.			
06	WAP which display the sum of salary department wise.		01/07/20	
07	Create a cursor to generate different two tables from one		01/07/20	
	master table.			
	FUNCTIONS	107		
01	WAF which accepts the name from user and returns the	108	01/07/20	
	length of that name.			
02	WAF which accepts one number and return TRUE if no		01/07/20	
	is prime and return FALSE if No. is not prime.			
03	Write a function which accepts the department no and		01/07/20	
	returns maximum salary of that Department.			
04	Write a function to display whether the entered (User Input)		01/07/20	
	employee no exists or not.			
05	WAF which accepts one no and returns that no+100.		01/07/20	
06	WAF which accepts the empno. and raises the salary .		01/07/20	
07	WAF which accepts the empno and returns the experience		01/07/20	

R O L L NO : 30
N A M E : Ajinkya Rathod
S U B J E C T : RELATIONAL DATABASE MANAGEMENT SYSTEM -II

	in years.			
	PROCEDURES	121		
01	Write a procedure which accepts the empno and returns	122	01/07/20	
	the associated empname.			
02	WAP which accepts the student rollno and returns the		01/07/20	
	name,city and marks of all the subjects of that student.			
03	WAP which accepts the name from the user. Return case		01/07/20	
	Of the name i.e. UPPER, LOWER or MIXCASE.			
04	WAP which accepts the student roll and returns highest		01/07/20	
	percent and name of that student to the calling block.			
05	WAP which accepts the date of joining for specific employee		01/07/20	
	and returns the years of experience along with its name.			
06	WAP which accepts the student roll no and returns the		01/07/20	
	result (i.e. first class, second class, third class or fail).			
	ASSIGNMENT 3			
	TRIGGERS	132		
01	WAT For Insert, Update and Delete on Customer	133	01/07/20	
02	WAP to create trigger signal to restrict entering negative		01/07/20	
	value in balance.			
03	WAT to perform data validation using select statement.		01/07/20	
04	Write an example to create sales table which provides free		01/07/20	
	Shipping on orders above 500			
	TRANSACTION	165		
05	Create procedure to commence a transaction using auto		01/07/20	
	commit.			

D E P A R T M E N T O F C O M P U T E R S C I E N C E

R O L L W A L A C O M P U T E R C E N T R E

G U J A R A T U N I V E R S I T Y

M.C.A. – II

R O L L NO : 30

N A M E : Ajinkya Rathod

S U B J E C T : RELATIONAL DATABASE MANAGEMENT SYSTEM -II

06	Create a procedure to commence a transaction using start		01/07/20	
	transaction.			
07	Create a procedure which display use of savepoint with a		01/07/20	
	Transaction.			
	DIY	166		
01	Write a Trigger that stores the old data table of student		01/07/20	
	table in student_backup while updating the student table.			
02	Write a trigger, that ensures the empno of emp table is in a		01/07/20	
	format ‘E00001’ .If not, than complete empno with this			
	format before inserting into the employee table.			
03	Write a trigger which checks the age of employee while		01/07/20	
	inserting the record in emp table.			
04	Write a trigger which converts the employee name in upper		01/07/20	
	case if it is inserted in any other case.			
05	WAT that stores the data of emp table in emp_backup		01/07/20	
	table for every delete operation and store the old data for			
	every update operation.			
06	WAT which display the message ‘Updating’,’Deleting’ or		01/07/20	
	’Inserting’ when Update, Delete or Insert operation is			
	performed on the emp table respectively.			
07	WAT which generate an error if any user try to delete from		01/07/20	
	product_master table on weekends.			
08	WAT which inserts the value of client_no in the		01/07/20	
	client_master table whenever user tries to insert data in			
	the emp table			
09	WAT to calculate the Income Tax and insert it in emp table.		01/07/20	

Thomas Write Rule

There are mainly 3 rules in Thomas Write Rule:

1. If read-transaction (A) > transaction (T), then abort it and rollback the "T" and reject.

2. If Write-transaction(A) > transaction(T), we should not execute write operation and continue.

3. If 1st and 2nd rule don't occurs, execute the write of T and set write-transaction (A) to transaction(T)

Trans 1	Trans 2
R(A)	
	W(A)
	Commit.
W(A)	
Commit.	

In Thomas rule, we can do all operations while 2nd transaction occurs before 1st transaction.

They are considered as outdated writes.

Aries Algorithm

An aries algorithm is also called recovery algorithm to work with 'no-force'.

Full form of Aries is:
"Algorithm for Recovery and Isolation Exploiting Semantics".

The main principles behind Aries are:

1. Write-Ahead logging
2. Repeating history during Redo.
3. Logging changes during Undo

The recovery process consists of 3 phases:

1. Analysis
2. Redo
3. Undo.

1. Analysis

→ The recovery determines the very first log which next should start. It uses a log from check point record.

2. Redo

From starting, log is read from forward.

Every update is done again

3. Undo

log is viewed again in previous and update undone transaction

The redo phase actually applies updates from log to database. Typically REDO operation is applied to only committed transactions.

However in Aries, certain information log will provide

start point for REDO, from which REDO operations are applied until end of log is reached.

Information stored by Aries and in data pages will allow ARIES to determine whether operation to be redone has actually been applied to database.

Thus only necessary Redo are appl-ed

During Undo, log is scanned backwards and operation of transaction were active at the time of crash are undone in reverse order.

NEW INDIA
PAGE NO.: 1 DATE

3. Short note on CLR.

Ans CLR is abbreviated form of compensation log record.
It is written just before change recorded in update log record is undone

→ A compensation log record also contains field called UndoNextLSN which is LSN of next log record.
The field in CLR is set to value of previous LSN in U.

→ The transaction action must be rollback and undo action described by CLR is definately required

→ Number of CLS that can be written during undo is no more than number of update log records for active transaction at time of crash.

→ It may will happen that CLR is written to stable storage.

→ A compensation log record describes action taken to undo action recorded in corresponding update log record and it appended in long tail just like any other log record.

Q4 Define transaction with reference to any SQL.

Ans A transaction is set of one or more SQL statements that are logically grouped together and must be applied to database entirely or not at all.

We expect database to follow the ACID principle

-> Atomic

-> Consistent

-> Isolated

-> Durable

-> With TCL stored program support, we encapsulate under defined statements of transaction into

PAGE NO.: DATE

single stored program.

→ There are 2 most popular storage engines

→ My ISAM
→ InnoDB.

Q5 Q. is ans four properties of transaction to ensure data integrity

(1.) ATOMICITY

A User should be able to regard the execution of each transaction as atomic, i.e. all actions are carried out or none.

(2.) Consistency

Each transaction run by itself with no concurrent execution of other transactions must preserve the consistency of the database.

Ensuring this property of transaction is responsibility of user

(3.) Isolation:

User should be able to understand a transaction without considering effect of other concurrently executing transactions.

Transactions are isolated or protected from effects of concurrently scheduling information.

(4.) Durability

Once DBMS says that trans. is completed it effect should persist even if system crashes before all changes are reflected on disk.

Q6 role of DBA in security

1) Creating new accounts

Each new user or group of users must be assigned an authorization id & and password.

To access the database, same authorization id the user executing the program

DBA plays an imp. role in In conjuction with owners of data, it also contributes to developing a security policy.

This is responsible for overall security of system

(2) Mandatory Control Issues

It DBMS supports this control - some customised systems for applications with very high security requirement provide which the DBA must assign security class to each db object.

DBA is also responsible for maintaining audit - trail which is log of updates.

This log is just mirror extension of log to recover from crashes

Q: Discuss two important assumptions with respect to transactions.

Ans
- Transaction interact with each other only via database read and write operations.

(2) It is tilled collection of independent objects.

When objects are added, from database relationship between database objects what we want to exploit for performance, some additional issue arise.

==========

==========

==========

==========

==========

==========

Google Classroom Work

==========

==========

==========

==========

==========

==========

```
 /*
================================================================
 *
 * Roll No: 30
 *
 * File:      CURSOR-1.txt
 * Copyright: 22-Apr-2020 by Ajinkya Rathod(ajinzrathod)
 *
 * Content: Write a cursor to fetch employee Details
 *
 *
================================================================ */


DELIMITER //


================================================================


CREATE OR REPLACE TABLE employee_details(
```

```
    emp_id INT(4) PRIMARY KEY
AUTO_INCREMENT,
    emp_name varchar (255),
    emp_post varchar (255),
    emp_salary DECIMAL (9, 2)) //

==========
==========

INSERT INTO employee_details VALUES
    ('', 'Twisha' , 'Web Developer', 45000.12),
    ('', 'Lisa' , 'Programmer', 45948.45),
    ('', 'Test', 'Senior Manager', 87895.25),
    ('', 'Jack', 'Assistant Manager', 87945.26),
    ('', 'John', 'System Analyst', 88552.00) //


==========
==========

SELECT * FROM employee_details //
```

```
+--------+----------+-----------------+-----------
-+
| emp_id | emp_name | emp_post          |
emp_salary |
+--------+----------+-----------------+-----------
-+
|      1 | Twisha    | Web Developer      |
45000.12 |
|      2 | Lisa      | Programmer         |   45948.45 |
|      3 | Test      | Senior Manager     |   87895.25
|
|      4 | Jack      | Assistant Manager |   87945.26
|
|      5 | John      | System Analyst     |   88552.00
|
+--------+----------+-----------------+-----------
-+
```

```
==========================================
=============
```

```
CREATE OR REPLACE procedure
fetch_emp_records()
BEGIN
    DECLARE id INT (4);
    DECLARE name varchar (255);
    DECLARE post varchar (255);
    DECLARE salary decimal (9, 2);

    DECLARE is_fetch_completed INT DEFAULT
0;
    DECLARE emp_cursor CURSOR FOR SELECT
* FROM employee_details;
    DECLARE CONTINUE HANDLER FOR NOT
FOUND SET is_fetch_completed = 1;

    OPEN emp_cursor;

    fetch_emp_details : LOOP
        FETCH emp_cursor INTO id, name, post,
salary;
        IF is_fetch_completed  = 1 THEN
            LEAVE fetch_emp_details;
        END IF;
```

```
    SELECT CONCAT ('ID: ', id, ' || NAME: ', name, ' || POST: ' , post, ' || SALARY: ', salary) as 'Employee Details';
    END LOOP fetch_emp_details;

    CLOSE emp_cursor;
END //

==========
==========

call fetch_emp_records() //

+-----------------------------------------------------------+
| Employee Details                                          |
+-----------------------------------------------------------+
| ID: 1 || NAME: Twisha || POST: Web Developer || SALARY: 45000.12 |
+-----------------------------------------------------------+
```

1 row in set (0.001 sec)

```
+-------------------------------------------------
--------+
| Employee Details
|
+-------------------------------------------------
--------+
| ID: 2 || NAME: Lisa || POST: Programmer ||
SALARY: 45948.45 |
+-------------------------------------------------
--------+
```

1 row in set (0.016 sec)

```
+-------------------------------------------------
------------+
| Employee Details
|
+-------------------------------------------------
------------+
| ID: 3 || NAME: Test || POST: Senior Manager
|| SALARY: 87895.25 |
+-------------------------------------------------
------------+
```

1 row in set (0.028 sec)

```
+-----------------------------------------------
--------------+
| Employee Details
|
+-----------------------------------------------
--------------+
| ID: 4 || NAME: Jack || POST: Assistant
Manager || SALARY: 87945.26 |
+-----------------------------------------------
--------------+
```

1 row in set (0.036 sec)

```
+-----------------------------------------------
------------+
| Employee Details
|
+-----------------------------------------------
------------+
| ID: 5 || NAME: John || POST: System Analyst
|| SALARY: 88552.00 |
+-----------------------------------------------
------------+
```

1 row in set (0.040 sec)

Query OK, 0 rows affected (0.046 sec)

==========
==========

/*
==========
==========

* Roll No: 30
*
* File: CURSOR-2.txt
* Copyright: 22-Apr-2020 by Ajinkya Rathod(ajinzrathod)
*
* Content: Write a cursor to find all employess in Dept passed as a argument
*
*
==========
========== */

```
==========
==========

DELIMITER //

==========
==========

CREATE OR REPLACE TABLE emp_details(
    emp_id INT(4) PRIMARY KEY
AUTO_INCREMENT,
    dept_id char (1)) //

CREATE OR REPLACE TABLE dept_details(
    dept_id CHAR (1),
    dept_name VARCHAR (255)) //

==========
==========

INSERT INTO emp_details VALUES
    ('', '2'),
```

```
    ('', '4'),
    ('', '8'),
    ('', '3'),
    ('', '8') //

INSERT INTO dept_details VALUES
    (1, 'Analyst'),
    (2, 'Coding'),
    (3, 'Programming'),
    (4, 'Designing'),
    (5, 'Senior Manager'),
    (6 , 'Junior Manager'),
    (7 , 'Assistant Manager'),
    (8 , 'Tester') //

==============================
==============

SELECT * FROM emp_details //

+--------+---------+
| emp_id | dept_id |
```

```
+--------+---------+
|      1 | 2       |
|      2 | 4       |
|      3 | 8       |
|      4 | 3       |
|      5 | 8       |
+--------+---------+
5 rows in set (0.001 sec)

================================================
==============

SELECT * FROM dept_details //

+---------+------------------+
| dept_id | dept_name        |
+---------+------------------+
| 1       | Analyst          |
| 2       | Coding           |
| 3       | Programming      |
| 4       | Designing        |
| 5       | Senior Manager   |
```

```
| 6       | Junior Manager      |
| 7       | Assistant Manager  |
| 8       | Tester                  |
+---------+-------------------+
8 rows in set (0.000 sec)
```

==========

```
CREATE OR REPLACE PROCEDURE
fetch_emp_from_dept (IN dname varchar (255))
BEGIN

    DECLARE id INT (4);

    DECLARE fetching INT DEFAULT  1;

    DECLARE dept_cursor CURSOR FOR SELECT
e.emp_id FROM emp_details e JOIN
    dept_details d  ON e.dept_id = d.dept_id
WHERE d.dept_name = dname;
```

```
    DECLARE CONTINUE HANDLER  FOR NOT
FOUND SET fetching = 0;

    OPEN dept_cursor;

        records : LOOP

            fetch dept_cursor INTO id;
            IF fetching = 0 THEN
                LEAVE records;
            END IF;
            SELECT id as 'Emp Details';

        END LOOP records;

    CLOSE dept_cursor;

END //

==============================
=============
```

```
call fetch_emp_from_dept('Tester') //


+-------------+
| Emp Details |
+-------------+
|           3 |
+-------------+
1 row in set (0.001 sec)

+-------------+
| Emp Details |
+-------------+
|           5 |
+-------------+
1 row in set (0.009 sec)

Query OK, 0 rows affected (0.020 sec)

================================
=============
```

```
 /*
================================
================================

 * Roll No: 30
 *
 * File:      CURSOR-3.txt
 * Copyright: 24-Apr-2020 by Ajinkya Rathod(ajinzrathod)
 *
 * Content: Write a cursor to increase salary of employess acc. to Dept.
 *
 *
================================
================================ */



mysql> create database aj;
Query OK, 1 row affected (0.00 sec)

================================
================================
```

```
mysql> use aj;
Database changed

==============================
==============================

mysql> show tables;
Empty set (0.00 sec)

==============================
==============================

mysql> CREATE table emp (
    -> e_id int(3) primary key auto_increment,
    -> e_name varchar(255),
    -> e_salary decimal(9, 2),
    -> dept_id int(3));
Query OK, 0 rows affected (0.43 sec)

==============================
==============================

mysql> CREATE table dept (
```

```
    -> dept_id int(3) PRIMARY KEY
auto_increment,
    -> dept_name varchar(255));
Query OK, 0 rows affected (0.41 sec)


================================
================================


mysql> desc emp;
+---------+-------------+------+-----+---------
+---------------+
| Field    | Type         | Null | Key | Default |
Extra          |
+---------+-------------+------+-----+---------
+---------------+
| e_id     | int(3)       | NO   | PRI | NULL    |
auto_increment |
| e_name   | varchar(255) | YES  |     | NULL    |
|
| e_salary | decimal(9,2) | YES  |     | NULL    |
|
| dept_id  | int(3)       | YES  |     | NULL    |
|
```

```
+---------+-------------+------+-----+---------
+---------------+
4 rows in set (0.00 sec)

================================
================================

mysql> desc dept;
+----------+-------------+------+-----+---------
+---------------+
| Field     | Type          | Null | Key | Default |
Extra          |
+----------+-------------+------+-----+---------
+---------------+
| dept_id   | int(3)        | NO   | PRI | NULL    |
auto_increment |
| dept_name | varchar(255) | YES  |     | NULL
|                |
+----------+-------------+------+-----+---------
+---------------+
2 rows in set (0.00 sec)

================================
================================
```

```
INSERT into emp (e_name, e_salary, dept_id)
VALUES
('Rachel McAdams', 82000, 1),
('Ajinkya Rathod', 89000, 1),
('Ken Adams', 105000, 1),
('Regina Phalange', 78000, 1) //

================================
================================

INSERT into dept (dept_name)
VALUES
('Statistical Analysis'),
('Quality'),
('Finance'),
('Accounts') //

================================
================================

mysql> select * from emp //
```

```
+------+----------------+----------+---------+
| e_id | e_name         | e_salary  | dept_id |
+------+----------------+----------+---------+
|    2 | Rachel McAdams  |  82000.00 |       1 |
|    3 | Ajinkya Rathod  |  89000.00 |       1 |
|    4 | Ken Adams       | 105000.00 |       1 |
|    5 | Regina Phalange |  78000.00 |       1 |
|    6 | Monica Geller   |  99000.00 |       2 |
|    7 | Ross Geller     |  78000.00 |       3 |
|    8 | Phoebe Buffay   |  56800.00 |       2 |
+------+----------------+----------+---------+
7 rows in set (0.00 sec)

================================
================================


mysql> select * from dept //
+---------+--------------------+
| dept_id | dept_name          |
+---------+--------------------+
|       1 | Statistical Analysis |
```

```
|       2 | Quality              |
|       3 | Finance              |
|       4 | Accounts             |
+---------+----------------------+
4 rows in set (0.00 sec)
```

==========

```
CREATE PROCEDURE increase_salary ()
BEGIN
    DECLARE eid INT(3);
    DECLARE did INT(3);

    DECLARE processing INT DEFAULT 1;
    DECLARE cursor_sal CURSOR FOR SELECT e_id, dept_id FROM emp;
    DECLARE CONTINUE HANDLER FOR NOT FOUND SET processing = 0;

    OPEN cursor_sal;
        inc_sal : LOOP
            FETCH cursor_sal into eid, did;
```

```
            IF processing = 0 THEN
                LEAVE inc_sal;
            END If;

            IF did = 1 THEN
                UPDATE emp SET e_salary =
e_salary + 1000 WHERE e_id = eid;
            ELSEIF did = 2 THEN
                UPDATE emp SET e_salary =
e_salary + 2000 WHERE e_id = eid;
            ELSEIF did = 3 THEN
                UPDATE emp SET e_salary =
e_salary + 3000 WHERE e_id = eid;
            END IF;
        END LOOP  inc_sal;
    CLOSE cursor_sal;
END //

==============================
==============================

mysql> call increase_salary() //
Query OK, 0 rows affected (0.44 sec)
```

==========
==========

```
mysql> select * from emp //
+------+----------------+----------+---------+
| e_id | e_name         | e_salary  | dept_id |
+------+----------------+----------+---------+
|    2 | Rachel McAdams  |  83000.00 |       1 |
|    3 | Ajinkya Rathod    |  90000.00 |       1 |
|    4 | Ken Adams         | 106000.00 |       1 |
|    5 | Regina Phalange  |  79000.00 |       1 |
|    6 | Monica Geller     | 101000.00  |       2 |
|    7 | Ross Geller        |  81000.00   |       3 |
|    8 | Phoebe Buffay    |  58800.00   |       2 |
+------+----------------+----------+---------+
7 rows in set (0.00 sec)
```

==========
==========

==========
==========

Assignment 1: PL/SQL

==========
==========

Q.1 Develop the following:

a. Create a procedure that deletes rows from the emp table.

It should accept 1 parameter, job; only delete the employee's with that job.

Display how many employees were deleted. Write a PL/SQL block to invoke the procedure.

b. Change the above procedure so that it returns the number of Employees removed via an OUT parameter.

Write a PL/SQL block to invoke the procedure and display how many employees were deleted.

```
CREATE TABLE IF NOT EXISTS emp1 (empid int(2) PRIMARY KEY, name VARCHAR(30), salary FLOAT, job VARCHAR(15));


INSERT INTO emp1 VALUES(1, 'Ajinkya', 92015, 'Operator');

INSERT INTO emp1 VALUES(2, 'Nirav', 10988, 'Salesman');

INSERT INTO emp1 VALUES(3, 'Pradip', 12522, 'Operator');

INSERT INTO emp1 VALUES(5, 'Nirav Chavda', 44979, 'Manager');

INSERT INTO emp1 VALUES(7, 'Lakshya', 20973, 'Supervisor');

INSERT INTO emp1 VALUES(9, 'Ghanshyam', 20025, 'Supervisor');

INSERT INTO emp1 VALUES(10, 'Maxmuller', 24029, 'Manager');

INSERT INTO emp1 VALUES(11, 'Fredrick', 21966, 'Supervisor');



DELIMITER //
```

```
CREATE OR REPLACE PROCEDURE
deleteByJob1(IN p_job VARCHAR(15))

BEGIN

DECLARE myCount INT DEFAULT 0;

SELECT COUNT(*) INTO myCount FROM emp1
WHERE job=p_job;

DELETE FROM emp1 WHERE job=p_job;

SELECT myCount AS "Number of employees
deleted";

END//

DELIMITER ;

CALL deleteByJob1('Operator');



DELIMITER //

CREATE OR REPLACE PROCEDURE
deleteByJob2(IN p_job VARCHAR(15), OUT
deletedEmp INT)

BEGIN

SELECT COUNT(*) INTO deletedEmp FROM emp1
WHERE job=p_job;

DELETE FROM emp1 WHERE job=p_job;

END//
```

DELIMITER ;

Q.2 Create a table having the following structure -> Accounts(Account_id, branch_name, amount_balance)

a. Write a PL/SQL procedure to perform withdraw operation that only permits a withdrawal if there sufficient funds in the account.

The procedure should take Account_id and withdrawal amount as input.

b. Write a procedure to deposit money into someone's account. The procedure should accept account_id and deposit amount.

```
CREATE TABLE IF NOT EXISTS
Accounts(Account_id INT, branch_name
VARCHAR(20), amount_balance FLOAT);

INSERT INTO Accounts VALUES(603,
'Gandhidham', 370205);
```

```
INSERT INTO Accounts VALUES(604, 'Adipur', 654654);

INSERT INTO Accounts VALUES(605, 'Poona', 986551);



DELIMITER //

CREATE OR REPLACE PROCEDURE withdrawFunds(IN u_Account_id INT, IN withdrawAmount FLOAT)

BEGIN

DECLARE account INT;

DECLARE amount FLOAT;

DECLARE message VARCHAR(50) DEFAULT "";

SELECT Account_id, amount_balance INTO account, amount FROM Accounts WHERE Account_id = u_Account_id;

IF account IS NULL THEN

    SET message = CONCAT(message, "Account No. ", u_Account_id ," not found!");

ELSE

    IF withdrawAmount > amount THEN

        SET message = "Insufficient Funds!";

    ELSEIF withdrawAmount < 0 THEN
```

```
        SET message = "Withdraw amount
should be positive value only!";
    ELSEIF withdrawAmount IS NULL THEN
        SET message = "Withdraw amount
cannot be empty!";
    ELSE
        UPDATE Accounts SET amount_balance =
amount_balance-withdrawAmount WHERE
Account_id=account;
        SET message = CONCAT(message, "Rs.
", withdrawAmount, " withdrawn from Account
no. ", account);
        SELECT *, amount AS "Previous Balance"
FROM Accounts WHERE Account_id=account;
    END IF;
END IF;
SELECT message;
END//
DELIMITER ;



DELIMITER //
```

```
CREATE OR REPLACE PROCEDURE
depositMoney(IN u_Account_id INT, IN
depositAmount FLOAT)
BEGIN
DECLARE account INT;
DECLARE amount FLOAT;
DECLARE message VARCHAR(50) DEFAULT "";
SELECT Account_id, amount_balance INTO
account, amount FROM Accounts WHERE
Account_id = u_Account_id;
IF account IS NULL THEN
    SET message = CONCAT(message, "Account
No. ", u_Account_id ," not found!");
ELSE
    IF depositAmount IS NULL THEN
        SET message = "Deposit amount cannot
be empty!";
    ELSEIF depositAmount < 0 THEN
        SET message = "Deposit amount should
be positive value only!";
    ELSE
        UPDATE Accounts SET amount_balance =
amount_balance+depositAmount WHERE
Account_id=account;
```

```
        SET message = CONCAT(message, "Rs. ", depositAmount, " deposited into Account no. ", account);
        SELECT *, amount AS "Previous Balance" FROM Accounts WHERE Account_id=account;
    END IF;
END IF;
SELECT message;
END//
DELIMITER ;
```

Q.3 Create a table having the following structure -> Accounts(Account_id, branch_name, amount_balance)

a. Write a PL/SQL procedure to perform withdraw operation that only permits a withdrawal if there sufficient funds in the account.

The procedure should take Account_id and withdrawal amount as input.

b. Write a procedure to transfer money from one person's account to another.

The procedure should table two account_id’s one for giver and one for receiver and the amount to be transferred.

```
CREATE TABLE IF NOT EXISTS
Accounts(Account_id INT, branch_name
VARCHAR(20), amount_balance FLOAT);


INSERT INTO Accounts VALUES(603,
'Gandhidham', 370205);

INSERT INTO Accounts VALUES(604, 'Adipur',
654654);

INSERT INTO Accounts VALUES(605, 'Poona',
986551);


DELIMITER //

CREATE OR REPLACE PROCEDURE
withdrawFunds(IN u_Account_id INT, IN
withdrawAmount FLOAT)

BEGIN

DECLARE account INT;

DECLARE amount FLOAT;

DECLARE message VARCHAR(50) DEFAULT "";
```

```
SELECT Account_id, amount_balance INTO
account, amount FROM Accounts WHERE
Account_id = u_Account_id;
IF account IS NULL THEN
    SET message = CONCAT(message, "Account
No. ", u_Account_id ," not found!");
ELSE
    IF withdrawAmount > amount THEN
        SET message = "Insufficient Funds!";
    ELSEIF withdrawAmount < 0 THEN
        SET message = "Withdraw amount
should be positive value only!";
    ELSEIF withdrawAmount IS NULL THEN
        SET message = "Withdraw amount
cannot be empty!";
    ELSE
        UPDATE Accounts SET amount_balance =
amount_balance-withdrawAmount WHERE
Account_id=account;
        SET message = CONCAT(message, "Rs.
", withdrawAmount, " withdrawn from Account
no. ", account);
        SELECT *, amount AS "Previous Balance"
FROM Accounts WHERE Account_id=account;
```

```
    END IF;
END IF;
SELECT message;
END//
DELIMITER ;



DELIMITER //
CREATE OR REPLACE PROCEDURE
transferFunds(IN senderAccount INT, IN
receiverAccount INT, transferAmount FLOAT)
BEGIN
DECLARE sAccount, rAccount INT;
DECLARE sAmount, rAmount FLOAT;
DECLARE sMessage, rMessage VARCHAR(50)
DEFAULT "";


SELECT Account_id, amount_balance INTO
sAccount, sAmount FROM Accounts WHERE
Account_id=senderAccount;
SELECT Account_id, amount_balance INTO
rAccount, rAmount FROM Accounts WHERE
Account_id=receiverAccount;
```

```
IF sAccount IS NULL THEN
    SET sMessage = CONCAT(sMessage, "Sender account no. ", senderAccount ," not found!");
ELSEIF rAccount IS NULL THEN
    SET rMessage = CONCAT(rMessage, "Receiver account no. ", receiverAccount ," not found!");
ELSE
    IF transferAmount IS NULL THEN
        SET sMessage = "Transfer Amount cannot be empty!";
    ELSEIF transferAmount < 0 THEN
        SET sMessage = "Transfer amount cannot be a negative value!";
    ELSEIF transferAmount > sAmount THEN
        SET sMessage = "Insufficient Funds!";
    ELSE
        UPDATE Accounts SET amount_balance = amount_balance - transferAmount WHERE Account_id=senderAccount;
        UPDATE Accounts SET amount_balance = amount_balance + transferAmount WHERE Account_id=receiverAccount;
```

```
        SET sMessage = CONCAT(sMessage, "Rs.
", transferAmount, " withdrawn from Account no.
", senderAccount);

        SET rMessage = CONCAT(rMessage, "Rs.
", transferAmount, " deposited into Account no.
", receiverAccount);

        SELECT *, sAmount AS "Previous
Balance" FROM Accounts WHERE
Account_id=sAccount UNION SELECT *, rAmount
AS "Previous Balance" FROM Accounts WHERE
Account_id=rAccount;

    END IF;

END IF;


SELECT sMessage UNION SELECT rMessage;

END//

DELIMITER ;
```

Q.6 Create three different procedures and one final procedure to call them as follows

a. First procedure check for the number is > 0 or not.

b. Second procedure accepts one date argument and check that is < sysdate or not.

c. Third procedure accepts a name and check whether it is in uppercase or not.

```
DELIMITER //
CREATE OR REPLACE PROCEDURE isNaturalNumber(IN num INT)
BEGIN
DECLARE message VARCHAR(30) DEFAULT "";
IF num > 0 THEN
    SET message = CONCAT(message, num, " is greater than 0.");
ELSEIF num < 0 THEN
    SET message = CONCAT(message, num, " is less than 0.");
ELSE
    SET message = CONCAT(message, "Number is 0.");
END IF;
SELECT message AS "Result";
END//
DELIMITER ;
```

```
DELIMITER //
CREATE OR REPLACE PROCEDURE
dateCompare(IN userDate DATE)
BEGIN
DECLARE result INT;
DECLARE message VARCHAR(50) DEFAULT "";
SET result = DATEDIFF(userDate,
CURRENT_DATE);
IF result > 0 THEN
    SET message = CONCAT(message, userDate,
" is greater than ", CURRENT_DATE);
ELSEIF result < 0 THEN
    SET message = CONCAT(message, userDate,
" is less than ", CURRENT_DATE);
ELSE
    SET message = "Both dates are same.";
END IF;
SELECT message;
END//
DELIMITER ;
CALL dateCompare('2020-04-05');
```

```
DELIMITER //
CREATE OR REPLACE PROCEDURE checkUpperCase(IN name VARCHAR(15))
BEGIN
DECLARE i, len INT DEFAULT 1;
DECLARE temp VARCHAR(1);
DECLARE isUpper BOOLEAN DEFAULT true;
SET len = LENGTH(name);
WHILE isUpper AND i <= len DO
SET temp = substr(name, i, 1);
IF ASCII(temp) >= 97 AND ASCII(temp) <= 122 THEN
    SET isUpper = false;
END IF;
SET i = i + 1;
END WHILE;
IF isUpper THEN
    SELECT CONCAT(name, " is in uppercase...") AS "Message";
ELSE
    SELECT CONCAT(name, " is not in uppercase...") AS "Message";
END IF;
```

```
END//
DELIMITER ;
CALL checkUpperCase('NIRAv');

DELIMITER //
CREATE OR REPLACE PROCEDURE myMenu()
BEGIN
CALL isNaturalNumber(@arg1);
CALL dateCompare(@arg2);
CALL checkUpperCase(@arg3);
END//
DELIMITER ;
```

Q.7 Write a procedure to display first n fibonacci numbers.

```
DELIMITER //
CREATE OR REPLACE PROCEDURE fibonacci(IN n
INT)
```

```
BEGIN
DECLARE count INTEGER DEFAULT 0;
DECLARE firstTerm, nextTerm INT DEFAULT 0;
DECLARE secondTerm INT DEFAULT 1;
DECLARE myStr VARCHAR(100) DEFAULT '';
REPEAT
SET myStr = CONCAT(myStr, firstTerm, ", ");
SET nextTerm = firstTerm + secondTerm;
SET firstTerm = secondTerm;
SET secondTerm = nextTerm;
SET count = count + 1;
UNTIL count=n
END REPEAT;
SELECT myStr AS "Fibonacci Series";
END//
DELIMITER ;
```

Q.8 Write a procedure to find simple interest.

```
DELIMITER //
```

```
CREATE OR REPLACE PROCEDURE
simpleInterest(IN p FLOAT, IN r FLOAT, IN n
FLOAT, OUT simple FLOAT)
BEGIN
SET simple = (p * r * n) / 100;
END//
DELIMITER ;
SET @principle = 10000;
SET @rate = 2.5;
SET @time = 4.5;
CALL simpleInterest(@principle, @rate, @time,
@simple);
SELECT CONCAT("Rs. ", @simple) AS "Simple
Interest";
```

Q.9 Write a procedure to reverse entered number.

```
DELIMITER //
CREATE OR REPLACE PROCEDURE
reverseNum(IN num INT)
BEGIN
```

```
DECLARE reversedNum, remainder INT DEFAULT 0;
WHILE num<>0 DO
SET remainder = MOD(num, 10);
SET reversedNum = (reversedNum * 10) + remainder;
SET num = num DIV 10;
END WHILE;
SELECT reversedNum;
END//
DELIMITER ;
```

Q.10 Write a procedure and find its equivalent roman value.

```
DELIMITER //
CREATE OR REPLACE PROCEDURE romanNum(IN num INT)
BEGIN
DECLARE num2 INT;
```

```
DECLARE str VARCHAR(20) DEFAULT '';
SET num2 = num;
WHILE num2<>0
DO
CASE
WHEN num2 >= 1000
THEN
SET str = CONCAT(str, "M");
SET num2 = num2 - 1000;
WHEN num2 >= 900
THEN
SET str = CONCAT(str, "CM");
SET num2 = num2 - 900;
WHEN num2 >= 500
THEN
SET str = CONCAT(str, "D");
SET num2 = num2 - 500;
WHEN num2 >= 400
THEN
SET str = CONCAT(str, "CD");
SET num2 = num2 - 400;
WHEN num2 >= 100
```

```
THEN
SET str = CONCAT(str, "C");
SET num2 = num2 - 100;
WHEN num2 >= 90
THEN
SET str = CONCAT(str, "XC");
SET num2 = num2 - 90;
WHEN num2 >= 50
THEN
SET str = CONCAT(str, "L");
SET num2 = num2 - 50;
WHEN num2 >= 40
THEN
SET str = CONCAT(str, "XL");
SET num2 = num2 - 40;
WHEN num2 >= 10
THEN
SET str = CONCAT(str, "X");
SET num2 = num2 - 10;
WHEN num2 >= 9
THEN
SET str = CONCAT(str, "IX");
```

```
SET num2 = num2 - 9;
WHEN num2 >= 5
THEN
SET str = CONCAT(str, "V");
SET num2 = num2 - 5;
WHEN num2 >= 4
THEN
SET str = CONCAT(str, "IV");
SET num2 = num2 - 4;
ELSE
SET str = CONCAT(str, "I");
SET num2 = num2 - 1;
END CASE;
END WHILE;
SELECT str;
END//
DELIMITER ;
```

Q.11 Write a program to enter a number and find addition of each digit of that number using function.

```
DELIMITER //
CREATE OR REPLACE FUNCTION addDigits(num INT) RETURNS INT
BEGIN
DECLARE result, remainder INT DEFAULT 0;
WHILE num <> 0 DO
    SET remainder = MOD(num, 10);
    SET result = result + remainder;
    SET num = num DIV 10;
END WHILE;
RETURN result;
END//
DELIMITER ;
```

Q. 12 Write a program to input your birthdate and should return your age in year, month and days.

```
DELIMITER //
CREATE OR REPLACE PROCEDURE calculateAge(IN birthdate DATE)
BEGIN
```

```
DECLARE dd, u_dd, r_dd, mm, u_mm, r_mm,
yy, u_yy, r_yy INT;
IF birthdate IS NULL THEN
    SELECT "Invalid Date!" AS "Error";
ELSEIF DATEDIFF(birthdate, CURRENT_DATE) >
0 THEN
    SELECT "Birthdate cannot be greater than
Current Date" AS "Error";
ELSE
    SET dd = EXTRACT(DAY from
CURRENT_DATE);
    SET mm = EXTRACT(MONTH from
CURRENT_DATE);
    SET yy = EXTRACT(YEAR from
CURRENT_DATE);
    SET u_dd = EXTRACT(DAY from birthdate);
    SET u_mm = EXTRACT(MONTH from
birthdate);
    SET u_yy = EXTRACT(YEAR from birthdate);

    SET r_yy = yy - u_yy;
    IF u_mm > mm THEN
        SET r_mm = 12 - (u_mm - mm);
        SET r_yy = r_yy - 1;
```

```
    ELSE
    SET r_mm = mm - u_mm;
    END IF;


    IF u_dd > dd THEN
        SET r_dd = EXTRACT(DAY FROM
LAST_DAY(birthdate)) - (u_dd - dd);
        SET r_mm = r_mm - 1;
    ELSE
    SET r_dd = dd - u_dd;
    END IF;
    SELECT CONCAT(r_dd, " days, ", r_mm, "
months, ", r_yy, " years.") AS "AGE";
END IF;
END//
DELIMITER ;
```

Q. 13 Write a program to reverse the string.

```
DELIMITER //
CREATE OR REPLACE PROCEDURE reverseStr(IN
userString VARCHAR(15))
```

```
BEGIN
SELECT REVERSE(userString);
END//
DELIMITER ;



DELIMITER //
CREATE OR REPLACE PROCEDURE
reverseString(IN userString VARCHAR(15))
BEGIN
DECLARE start, stop INT;
DECLARE swap VARCHAR(1);
SET start = 1;
SET stop = LENGTH(userString);
WHILE start < stop DO
    SET swap = substr(userString, start, 1);
    SET userString = INSERT(userString, start,
1, substr(userString, stop, 1));
    SET userString = INSERT(userString, stop, 1,
swap);
    SET start = start + 1;
    SET stop = stop - 1;
END WHILE;
```

```
SELECT userString;
END//
DELIMITER ;
```

Q.14 Write a program to convert the string into toggle case.

```
DELIMITER //
CREATE OR REPLACE PROCEDURE
toggleString(IN userString VARCHAR(20))
BEGIN
DECLARE start, stop INT DEFAULT 1;
DECLARE temp INT;
SET stop = LENGTH(userString);
WHILE start <= stop DO
SET temp = ASCII(substr(userString, start, 1));
    IF temp >= 65 AND temp <= 90 THEN
        SET temp = temp + 32;
        SET userString = INSERT(userString,
start, 1, CHAR(temp));
    ELSEIF temp >= 97 AND temp <= 122 THEN
        SET temp = temp - 32;
```

```
            SET userString = INSERT(userString,
start, 1, CHAR(temp));
        END IF;
        SET start = start + 1;
END WHILE;
SELECT userString AS "Toggled String";
END//
DELIMITER ;
```

Q. 15 Write a PL/SQL block to convert given numbers into text words

```
DELIMITER //
CREATE OR REPLACE PROCEDURE numToText(IN
userNumber INT)
BEGIN
DECLARE numInText VARCHAR(100) DEFAULT "
";
DECLARE power, remainder INT;
REPEAT
        SET remainder = MOD(userNumber, 10);
```

```
    CASE
    WHEN remainder = 0 THEN
        SET numInText = INSERT(numInText, 1,
0, "Zero ");
    WHEN remainder = 1 THEN
        SET numInText = INSERT(numInText, 1,
0, "One ");
    WHEN remainder = 2 THEN
        SET numInText = INSERT(numInText, 1,
0, "Two ");
    WHEN remainder = 3 THEN
        SET numInText = INSERT(numInText, 1,
0, "Three ");
    WHEN remainder = 4 THEN


        SET numInText = INSERT(numInText, 1,
0, "Four ");
    WHEN remainder = 5 THEN
        SET numInText = INSERT(numInText, 1,
0, "Five ");
    WHEN remainder = 6 THEN
        SET numInText = INSERT(numInText, 1,
0, "Six ");
    WHEN remainder = 7 THEN
```

```
        SET numInText = INSERT(numInText, 1,
0, "Seven ");
    WHEN remainder = 8 THEN
        SET numInText = INSERT(numInText, 1,
0, "Eight ");
    WHEN remainder = 9 THEN
        SET numInText = INSERT(numInText, 1,
0, "Nine ");
    END CASE;


    SET userNumber = userNumber DIV 10;
UNTIL userNumber = 0
END REPEAT;
SELECT numInText AS "Number in Text";
END//
DELIMITER ;
```

==========

Assignment 2: GENERAL PL/SQL BLOCKS

==========

NAME : Ajinkya Rathod

ROLL NO : 30

==========
==========
==========
==========
===

QUESTION 1 : WAP to input two numbers and find out what is the output of all arithmetic operations. (Addition, Subtraction, Multiplication, Division etc.)

```
DELIMITER //
CREATE PROCEDURE arithmetic(no1 INT,no2 INT)
    BEGIN
    DECLARE ans float;
    SET ans=no1 + no2;
    SELECT ans AS "addition";
    SET ans=no1 - no2;
```

```
    SELECT ans AS "subtraction";
    SET ans=no1*no2;
    SELECT ans AS "multiplication";
    SET ans=no1 / no2;
    SELECT ans AS "division";
    END//
```

****

OUTPUT :

```
 CALL arithmetic(20,10) //

+----------+
| addition |
+----------+
|       30 |
+----------+
1 row in set (0.109 sec)

+-------------+
| subtraction |
+-------------+
|          10 |
```

```
+------------+
1 row in set (0.116 sec)

+----------------+
| multiplication |
+----------------+
|            200 |
+----------------+
1 row in set (0.122 sec)

+----------+
| division |
+----------+
|        2 |
+----------+
1 row in set (0.128 sec)

**********************************

==========
==========
==========
```

==========
===

QUESTION 2 : WAP to input rollno and three subject marks. Find out total, percentage, result and grade for the student FROM the entered data.

```
 DELIMITER //
 CREATE PROCEDURE marksheet(rollno INT,sub1 INT,sub2 INT,sub3 INT)
     BEGIN
     DECLARE total INT DEFAULT 0;
     DECLARE percentage float;
     DECLARE result varchar(10);
     DECLARE grade varchar(10);

     SET total=sub1 + sub2 + sub3;
     SET percentage=total / 3;

     IF (sub1 >= 35 AND sub2 >= 35 AND sub3 >= 35) then
     SET result="PASS";
```

```
    ELSE
    SET result="FAIL";
    END IF;


    IF(percentage < 35 OR result="FAIL") then
    SET grade="F";
    ELSEIF(percentage >= 35 AND percentage <
60) then
    SET grade="C";
    ELSEIF(percentage >= 60 AND percentage <
80) then
    SET grade="B";
    ELSE
    SET grade="A";
    END IF;


    SELECT rollno AS "roll no",sub1 AS
"subject1",sub2 AS "subject2",sub3 AS
"subject3",total AS "total marks",percentage AS
"percentage",grade AS "grade",result AS
"result";
    END//
```

****

OUTPUT :

CALL marksheet(1,100, 98, 78) //

```
+---------+----------+----------+----------+-------
------+------------+-------+--------+
| roll no | subject1 | subject2 | subject3 | total
marks | percentage | grade | result |
+---------+----------+----------+----------+-------
------+------------+-------+--------+
|       1 |      100 |       98 |       78 |         276 |
92 | A     | PASS   |
+---------+----------+----------+----------+-------
------+------------+-------+--------+
```

****

==========
==========
==========
==========
===

QUESTION 3 : WAP to PRINT first 10 odd numbers using for loop.

```
DELIMITER //
CREATE PROCEDURE oddno()
    BEGIN
    DECLARE no varchar(30)  DEFAULT "1";
    DECLARE count INT DEFAULT 1;
    DECLARE i INT DEFAULT 2;

    WHILE count < 10 do
    IF i%2 !=0 then
    SET no=concat(no,",",i);
    SET count=count + 1;
    END IF;

    SET i = i + 1;
    END WHILE;

    SELECT no AS "First 10 Odd numbers";

    END //
```

****

OUTPUT :

```
 CALL oddno()
+------------------------+
| First 10 Odd numbers     |
+------------------------+
| 1,3,5,7,9,11,13,15,17,19 |
+------------------------+
**********************************
```

==========
==========
==========
==========
===

QUESTION 4: WAP to PRINT prime numbers upto 10 using WHILE loop

```
 DELIMITER //

 CREATE PROCEDURE primeno()
     BEGIN
     DECLARE i INT DEFAULT 2;
```

```
DECLARE no varchar(30) DEFAULT "";
DECLARE j INT DEFAULT 2;
DECLARE flag INT;

WHILE i < 10 do
SET flag=1;
SET j=2;
WHILE j < i and flag do
IF i%j=0 then
SET flag=0;
END IF;
SET j=j + 1;
END WHILE;

IF flag then
SET no=concat(no,i,",");
END IF;
SET i=i + 1;
END WHILE;

SELECT no AS "Prime Numbers";
```

```
    END //
```

****

OUTPUT :

```
call prime() //
+----------------------+
| Prime Numbers |
+----------------------+
| 2,3,5,7,              |
+----------------------+
```

****

==========

QUESTION 5 : WAP to input three nos and find out maximum and minimum FROM it.

```
 DELIMITER //
```

```
 CREATE PROCEDURE minmax(no1 INT,no2
INT,no3 INT)
    BEGIN
    DECLARE min INT;
    DECLARE max INT;

    IF no1 < no2 then
    IF no1 < no3 then
    SET min=no1;
    ELSE
    SET min=no3;
    END IF;

    ELSE
    IF no2 < no3 then
    SET min=no2;
    ELSE
    SET min=no3;
    END IF;

    END IF;
```

```
    IF no1 > no2 then
    IF no1 > no3 then
    SET max=no1;
    ELSE
    SET max=no3;
    END IF;

    ELSE

    IF no2 > no3 then
    SET max=no2;
    ELSE
    SET max=no3;
    END IF;

    END IF;

    SELECT min AS "min",max AS "max";
    END //
```

****

OUTPUT :

```
 CALL minmax(45,86,78) //
 +-----------+-----------+
| min no | max no |
+-----------+-----------+
|        45 |        86 |
+-----------+-----------+
**********************************
```

```
==========
==========
==========
==========
===
```

QUESTION 6 : WAP to input empno FROM keybord. Check whether inputed empno exist in emp table or not. IF not give error message otherwise display name and salary of that employe

```
 DELIMITER //

 CREATE OR REPLACE PROCEDURE checkemp(no
INT)
```

```
    BEGIN
    DECLARE name varchar(30) DEFAULT
"NULL";
    DECLARE sala INT;
    SELECT e_name INTO name FROM emp
WHERE e_id=no;
    SELECT e_salary INTO sala FROM emp
WHERE e_id=no;
    SELECT no,name ,sala;


    END //
```

****

OUTPUT :

```
 CALL checkemp(1) //
+------+----------------+-------+
| no   | name           | sala  |
+------+----------------+-------+
|    3 | Ajinkya Rathod | 90000 |
+------+----------------+-------+
```

****

==========
==========
==========
==========
======

QUESTION 7 : WAP to INSERT record in Customer table.
Customer(cid,cname,address,city);

```
 CREATE table customer(cid INT,cname varchar(255),address varchar(255),city varchar(20)) //

 DELIMITER //

 CREATE  PROCEDURE icstmr(id INT,name varchar(15),address varchar(50),city varchar(20))
     BEGIN
     INSERT INTO customer values(id,name,address,city);
     END //
```

****

OUTPUT :

```
 CALL icstmr(1,'Lax','Megchar Porichi','Kutch') //

 SELECT * FROM customer //
+------+-------+----------------+-------+
| cid  | cname | address        | city  |
+------+-------+----------------+-------+
|    1 | Lax   | Megchar Porichi | Kutch |
+------+-------+----------------+-------+

**********************************

==========
==========
==========
==========
======

==========
==========
```

CURSORS

```
==========
==========
```

1) Create a cursor for the emp table. Produce the output in following format:

{empname} employee working in department {deptno} earns Rs. {salary}.

EMP(empno, empname, salary, deptno);

```
DELIMITER //
CREATE OR REPLACE PROCEDURE showEmployees()
BEGIN

DECLARE found BOOLEAN DEFAULT true;
DECLARE v_id INT;
DECLARE v_name, v_dept VARCHAR(15);
DECLARE v_salary FLOAT;
DECLARE list CURSOR FOR SELECT * FROM emp3;
DECLARE CONTINUE HANDLER FOR NOT FOUND SET found = false;

OPEN list;
```

```
WHILE found DO
    FETCH list INTO v_id, v_name, v_salary, v_dept;
    SELECT CONCAT("EmpID: ", v_id, ", ", v_name, " employee working in ",v_dept, " department earns Rs. ", v_salary) AS "Message";
END WHILE;


END//
DELIMITER ;
```

2) Create a cursor for updating the salary of emp working in deptno 10 by 20%.

If any rows are affected than display the no of rows affected. Use implicit cursor.

```
DELIMITER //
CREATE OR REPLACE PROCEDURE updateSalary(u_dept VARCHAR(15))
BEGIN
DECLARE v_empid INT;
DECLARE v_name VARCHAR(15);
```

```
DECLARE v_salary FLOAT;
DECLARE rowsAffected INT DEFAULT -1;
DECLARE found BOOLEAN DEFAULT true;
DECLARE salaryCur CURSOR FOR SELECT empid, name, salary FROM emp3 WHERE LOWER(dept) = LOWER(u_dept);
DECLARE CONTINUE HANDLER FOR NOT FOUND SET found = false;

OPEN salaryCur;

myLoop:LOOP
    IF NOT found THEN
        LEAVE myLoop;
    END IF;

    FETCH salaryCur INTO v_empid, v_name, v_salary;
    UPDATE emp3 SET salary = v_salary + (v_salary * 20 / 100) WHERE empid = v_empid;
    SET rowsAffected = rowsAffected + 1;
END LOOP myLoop;
CLOSE salaryCur;
```

```
IF rowsAffected = 0 THEN
    SELECT CONCAT("No employees found belonging to ", u_dept, " department!") AS "Message";
ELSE
    SELECT CONCAT("Updated salary of ", rowsAffected, " employees from ", u_dept, " by 20%.") AS "Message";
END IF;
END//
DELIMITER ;
```

4) WAP that will display the name, department and salary of the first 10 employees

getting the highest salary.

```
DELIMITER //
CREATE OR REPLACE PROCEDURE first_ten_employees()
BEGIN

DECLARE found BOOLEAN DEFAULT true;
```

```
DECLARE count INT DEFAULT 0;
DECLARE v_id INT;
DECLARE v_name, v_dept VARCHAR(15);
DECLARE v_salary FLOAT;
DECLARE list CURSOR FOR SELECT * FROM
emp3 ORDER BY salary DESC;
DECLARE CONTINUE HANDLER FOR NOT FOUND
SET found = false;

OPEN list;
WHILE found AND count < 10 DO
    FETCH list INTO v_id, v_name, v_salary,
v_dept;
    SELECT v_name, v_salary, v_dept;
    SET count = count + 1;
END WHILE;
CLOSE list;

END//
DELIMITER ;
```

5) WAP using parameterized cursor to display all the information of employee living in

specified city. Ask the city from user.

```
CREATE TABLE IF NOT EXISTS emp4(
    empid INT(2) PRIMARY KEY,
    emp_name VARCHAR(20) NOT NULL,
    salary FLOAT NOT NULL,
    city VARCHAR(20)
);
Query OK, 0 rows affected (0.291 sec)

SELECT * FROM emp4;
Empty set (0.002 sec)

INSERT INTO emp4 VALUES(401, "Juned Ajmeri", 21500, "Ahmedabad"),
(402, "Awasthi Pratik", 21000, "Ahmedabad"),
(403, "Baraiya Dhaval", 22000, "Bhavnagar"),
(404, "Rathod Ajinkya", 41000, "Gandhidham"),
(405, "Chavda Nirav", 51000, "Mundra"),
(406, "Milind Modi", 51000, "Ahmedabad"),
(407, "Lakshya Choudhary", 20000, "Gandhidham"),
```

```
(408, "Kshitij Modi", 21500, "Patan"),
(409, "Neel Rana", 24000, "Ahmedabad"),
(410, "Pradip Karmakar", 37500, "Navsari");

Query OK, 10 rows affected (0.103 sec)
Records: 10  Duplicates: 0  Warnings: 0



DELIMITER //
CREATE OR REPLACE PROCEDURE
getEmpByCity(v_city VARCHAR(20))
BEGIN


DECLARE e_id INT;
DECLARE e_name VARCHAR(20);
DECLARE e_salary FLOAT;
DECLARE found INT DEFAULT 1;
DECLARE empCity CURSOR FOR SELECT empid,
emp_name, salary FROM emp4 WHERE
UPPER(city) = UPPER(v_city);
DECLARE CONTINUE HANDLER FOR NOT FOUND
SET found = 0;
```

```
OPEN empCity;

myLoop: LOOP
    FETCH empCity INTO e_id, e_name, e_salary;
    IF found = 0 THEN
        LEAVE myLoop;
    END IF;
    SELECT e_id, e_name, e_salary, v_city;

END LOOP myLoop;

CLOSE empCity;

END//
DELIMITER ;
```

6) WAP which display the sum of salary department wise.

```
CREATE TABLE IF NOT EXISTS emp3 (
  empid INT(2) DEFAULT NULL,
```

```
  name VARCHAR(15) DEFAULT NULL,
  salary FLOAT DEFAULT NULL,
  dept VARCHAR(15) DEFAULT NULL
);

INSERT INTO emp3 VALUES(1, 'Juned Ajmeri', 12000, 'HR'),
      (2, 'Pratik Awasthi', 11000, 'HR'),
      (3, 'Dhaval Baraiya', 12500, 'HR'),
      (5, 'Nirav Chavda', 45000, 'Finance'),
      (7, 'Lakshya Choudhary', 21000, 'Sales'),
      (9, 'Neel Golarana', 20000, 'Sales'),
      (10, 'Pradip Karmakar', 24000, 'Finance'),
      (11, 'Keshri Rakesh', 22000, 'Sales'),
      (13, 'Rakesh Makhija', 10000, 'Production'),
      (14, 'Harshad Makwana', 10000, 'Marketing'),
      (21, 'Shubham Namjoshi', 21000, 'Production'),
      (30, 'Ajinkya Rathod', 30000, 'Marketing');

DELIMITER //
CREATE OR REPLACE PROCEDURE deptSalary()
```

```
BEGIN
DECLARE v_dept VARCHAR(20);
DECLARE v_sum FLOAT;
DECLARE found INT DEFAULT 1;
DECLARE myCur CURSOR FOR SELECT dept, SUM(salary) FROM emp3 GROUP BY dept;
DECLARE CONTINUE HANDLER FOR NOT FOUND SET found = 0;

OPEN myCur;
myLoop: LOOP

    FETCH myCur INTO v_dept, v_sum;
    IF NOT found THEN
        LEAVE myLoop;
    END IF;
    SELECT v_dept, v_sum;
END LOOP myLoop;
CLOSE myCur;
END//
DELIMITER ;
```

7) Create a cursor to generate defferent two tables from one master table.

Student(Rno, Name, Std, B_date, Sex);

Girl_Table(Rno, Name, Std, B_date);

Boy_Table(Rno, Name, Std, B_date);

First fetch the row from Student table. If sex is ‘M’ then insert that row in Boy_Table

and if ‘F’ then insert that row in Girl_Table.

In both table Rollno entry must be in Sequence(Using create sequence command).

```
CREATE TABLE IF NOT EXISTS stud(
    rollno INT AUTO_INCREMENT,
    name VARCHAR(20) NOT NULL,
    std INT,
    bdate DATE,
    sex VARCHAR(1),
    PRIMARY KEY(rollno)
);
```

Query OK, 0 rows affected (0.255 sec)

```
CREATE TABLE IF NOT EXISTS boy_table(
```

```
    rollno INT,
    name VARCHAR(20) NOT NULL,
    std INT,
    bdate DATE,
    PRIMARY KEY(rollno)
);
Query OK, 0 rows affected (0.210 sec)

CREATE TABLE IF NOT EXISTS girl_table(
    rollno INT,
    name VARCHAR(20) NOT NULL,
    std INT,
    bdate DATE,
    PRIMARY KEY(rollno)
);
Query OK, 0 rows affected (0.271 sec)

INSERT INTO stud (name, std, bdate, sex)
VALUES('Awasthi Pratik', 10, '1998-12-21', 'M'),
('Awasthi Pratik', 10, '1998-12-21', 'M'),
('Ankita Gaonkar', 9, '1998-12-21', 'F'),
('Mohini Parmar', 10, '1998-12-21', 'F'),
```

```
('Shivangi Chotaliya', 8, '1998-12-21', 'F'),
('Pradip Karmakar', 12, '1998-12-21', 'M'),
('Ajinkya Rathod', 11, '1998-12-21', 'M');
Query OK, 7 rows affected (0.092 sec)
Records: 7  Duplicates: 0  Warnings: 0



DELIMITER //
CREATE OR REPLACE PROCEDURE splitData()
BEGIN
    DECLARE v_rollno, v_std, found INT;
    DECLARE v_name VARCHAR(20);
    DECLARE v_bdate DATE;
    DECLARE v_sex VARCHAR(1);
    DECLARE myCur CURSOR FOR SELECT *
FROM stud;
    DECLARE CONTINUE HANDLER FOR NOT
FOUND SET found = 0;
    OPEN myCur;
    myLoop: LOOP
        FETCH myCur INTO v_rollno, v_name,
v_std, v_bdate, v_sex;
        IF NOT found THEN
```

```
            LEAVE myLoop;
        END IF;
        IF v_sex = "M" OR v_sex = "m" THEN
            INSERT INTO boy_table
VALUES(v_rollno, v_name, v_std, v_bdate);
        ELSEIF v_sex = "F" OR v_sex = "f" THEN
            INSERT INTO girl_table
VALUES(v_rollno, v_name, v_std, v_bdate);
        END IF;
    END LOOP myLoop;
    CLOSE myCur;
END//
```

==========

Assignment 2: Functions

==========

NAME : Ajinkya Rathod

ROLL NO : 30

==========
==========
========

QUESTION 1 : WAF which accepts the name from user and RETURNS the length of that name.

==========
==========
========

```
Begin
    DECLARE aj INT DEFAULT 1;
    DECLARE ch varchar(1);

    SET ch = substr(name,aj,1);

      WHILE(ch!="") DO
      SET aj = aj+1;
      end while;
      SET aj = aj-1;
      RETURN aj;

  End //
```

****

select length("Ajinkya")as Length;

```
+--------+
| Length |
+--------+
|      7 |
+--------+
```

****

==========

QUESTION 2 : WAF which accepts one number and RETURN TRUE IF no is prime and RETURN FALSE IF No. is not prime.

==========

```
DELIMITER //
CREATE OR REPLACE  function prime(no int)
RETURNS varchar(10)
```

```
BEGIN
DECLARE i int default 2;
while i < no do
IF no % i  =  0 THEN
RETURN "false";
END IF;
set i  =  i + 1;
END while;
RETURN "true";
END //
```

****

OUTPUT :

```
select prime(8191) //
 + ------------ +
| prime(8191) |
 + ------------ +
| true        |
 + ------------ +
1 row in set (0.030 sec)
```

****

==========

LETS LOOK AT EMPLOYEE TABLE BEFORE WORKING ON IT.

```
+------+----------------+----------+---------+
| e_id | e_name         | e_salary  | dept_id |
+------+----------------+----------+---------+
|    2 | Rachel McAdams  |  83000.00 |       1 |
|    3 | Ajinkya Rathod  |  90000.00 |       1 |
|    4 | Ken Adams       | 106000.00 |       1 |
|    5 | Regina Phalange |  79000.00 |       1 |
|    6 | Monica Geller   | 101000.00 |       2 |
|    7 | Ross Geller     |  81000.00 |       3 |
|    8 | Phoebe Buffay   |  58800.00 |       2 |
|    9 | Steffi Rathod   |      6.00 |       2 |
|   10 | John Doe        |    206.00 |       3 |
+------+----------------+----------+---------+
```

==========
==========
========

==========
==========
========

QUESTION 3 : Write a function which accepts the department no and RETURNS maximum salary of that Department. Handle the error IF dept no. does not exist or SELECT statement RETURN more than one row. EMP(Empno, deptno, salary).

==========
==========
========

```
DELIMITER //
CREATE OR REPLACE  function max_sal(depno varchar(10)) RETURNS varchar(30)
    BEGIN
    DECLARE income int;
    DECLARE c int default 0;
    SELECT max(e_salary) into income from emp WHERE dept_id = depno;
```

```
    SELECT count(*) into c from emp WHERE
e_salary = income and dept_id = depno;
    IF c > 1 THEN
    RETURN "Multiple Records";
    ELSEIF c  =  0 THEN
    RETURN "Department not exists";
    ELSE
    RETURN income;
    END IF;
    END //
```

****

OUTPUT :

```
 SELECT max_sal(1) as "MAX" //
 + -------- +
| MAX     |
 + -------- +
| 106000 |
 + -------- +
1 row in set (0.135 sec)
```

****

==========
==========
========

QUESTION 4 : Write a function to display whether the entered (User Input) employee no exists or not.

==========
==========
========

```
DELIMITER //

CREATE OR REPLACE   function emp_exist(no varchar(10)) RETURNS varchar(30)
    BEGIN
    DECLARE i int default 0;
    DECLARE result varchar(30);
    SELECT count(*) into i from emp WHERE e_id = no;

    IF i = 0 THEN
```

```
    set result = "FALSE";
    ELSE
    set result = "TRUE";
    END IF;

    RETURN result;
    END //
```

****

Output

```
 SELECT emp_exist(6) as "employee" //
 + ---------- +
| employee |
 + ---------- +
| TRUE     |
 + ---------- +
```

****

==========
==========
========

QUESTION 5 : WAF which accepts one no and RETURNS that no + 100. Use IN OUT mode.

==========
==========
========

```
DELIMITER //

CREATE OR REPLACE   function addition(no int)
RETURNS int
    BEGIN
    set no = no  +  100;
    RETURN no;
    END //
```

****

OUTPUT :

```
SELECT addition(30) as " add 100" //
 + --------- +
| add 100 |
 + --------- +
|     130 |
```

+ -------- +

****

==========

==========

========

QUESTION 6 : WAF which accepts the empno. IF salary < 10000 than give raise by 30%. IF salary < 20000 and salary >= 10000 than give raise by 20%.

IF salary>20000 than give raise by 10%. Handle the error IF any.

==========

==========

========

```
DELIMITER //

CREATE OR REPLACE  function empsal(no varchar(30)) RETURNS varchar(30)
    BEGIN
    DECLARE income int default 1;
    SELECT e_salary into income from emp WHERE e_id = no;
```

```
    IF income < 10000 THEN
    UPDATE emp set e_salary = e_salary +
(e_salary*0.3) WHERE e_id = no;
    ELSEIF income >= 10000 and income <
20000 THEN
    UPDATE emp set e_salary = e_salary +
(e_salary*0.2) WHERE e_id = no;
    ELSE
    UPDATE emp set e_salary = e_salary +
(e_salary*0.1) WHERE e_id = no;
    END IF;

    IF income = 1 THEN
    RETURN "Employee Not Found. ";
    END IF;

    RETURN "Salary++ ";
    END //
```

****

OUTPUT :

SELECT empsal(2) as "salary" //

```
 + ------------------------------- +
| salary                          |
 + ------------------------------- +
| Salary Raised Succesfully |
 + ------------------------------- +
```

****

==========

QUESTION 7 : WAF which accepts the empno and RETURNS the experience in years.

Handle the error IF empno does not exist. EMP(Empno, Empname, DOJ);

==========

ALTER table emp ADD date_of_joining date //

```
INSERT into emp values('','Dev', 78540, 4,'2014-04-04') //


DELIMITER //


CREATE OR REPLACE function empage(no int)
RETURNS int
    BEGIN
    DECLARE Joining_date date default "0001-01-01";
    DECLARE exp int;
    SELECT date_of_joining into Joining_date from emp WHERE e_id = no;
    IF Joining_date = "0001-01-01" THEN
    signal sqlstate "45000"
    set message_text = "Employee Not Found";
    ELSE
    set exp = year(curdate())-year(Joining_date);
    END IF;
    RETURN exp;


    END //
```

****

OUTPUT :

SELECT empage(11) as "EXP" //

```
+------+
| EXP  |
+------+
|    6 |
+------+
```

****

==========

==========

==========

==========

Assignment 2: Procedures

==========

==========

1) Write a procedure which accepts the empno and returns the associated empname. If

empno does not exist than give proper error message.

EMP(Empno, Empname).

```
CREATE TABLE IF NOT EXISTS emp3(
    empid INT(2),
    name VARCHAR(15),
    salary FLOAT,
    dept VARCHAR(15)
);

INSERT INTO emp3 VALUES
    (1, 'Ajinkya Rathod', 70043, 'Programming');

DELIMITER //
CREATE OR REPLACE PROCEDURE
getEmpName(id INT, OUT empname
VARCHAR(15))
BEGIN
DECLARE ename VARCHAR(15);
```

```
SELECT name INTO ename FROM emp3 WHERE empid = id;
IF ename IS NULL THEN
    SELECT "No such emp";
    SET empname = '';
ELSE
    SET empname = ename;
END IF;
END//
DELIMITER ;
```

2) WAP which accepts the student rollno and returns the name,city and marks of all the subjects of that student.

STUDENT (Stud_ID, Stud_name, m1, m2, m3).

```
CREATE TABLE IF NOT EXISTS students (
  rollno INT(2) NOT NULL PRIMARY KEY,
  name VARCHAR(30) NOT NULL,
  city VARCHAR(20) NOT NULL,
  marks1 FLOAT,
  marks2 FLOAT,
```

```
  marks3 FLOAT
);



DELIMITER //
CREATE OR REPLACE PROCEDURE
getDetails(std_id INT, OUT std_name
VARCHAR(30), OUT std_city VARCHAR(20), OUT
std_m1 FLOAT, OUT std_m2 FLOAT, OUT std_m3
FLOAT)
BEGIN
DECLARE sid INT;
DECLARE sname VARCHAR(30);
DECLARE scity VARCHAR(20);
DECLARE sm1, sm2, sm3 FLOAT;
SELECT * INTO sid, sname, scity, sm1, sm2,
sm3 FROM students WHERE rollno = std_id;
IF sid IS NULL THEN
    SELECT "No student";
    SET std_name = '';
    SET std_city = '';
    SET std_m1 = 0;
    SET std_m2 = 0;
```

```
    SET std_m3 = 0;
ELSE
    SET std_name = sname;
    SET std_city = scity;
    SET std_m1 = sm1;
    SET std_m2 = sm2;
    SET std_m3 = sm3;
END IF;
END//
DELIMITER ;
```

3) WAP which accepts the name from the user. Return UPPER if name is in uppercase,

LOWER if name is in lowercase, MIXCASE if name is entered using both the case.

```
DELIMITER //
CREATE OR REPLACE PROCEDURE strType(str VARCHAR(20), OUT  type VARCHAR(10))
BEGIN
DECLARE ch VARCHAR(1);
DECLARE isLower, isUpper BOOLEAN;
```

```
DECLARE lent, i INT;

    SET isUpper = false;
    SET isLower = false;
    SET i = 1;
    SET lent = LENGTH(str);

    WHILE i <= lent DO
        SET ch = SUBSTR(str, i, 1);
        IF ASCII(ch) <= 90 AND ASCII(ch) <=
65 THEN
            SET isUpper = true;
        ELSEIF ASCII(ch) <= 122 AND ASCII(ch)
>= 97 THEN
            SET isLower = true;
        END IF;
        SET i = i + 1;
    END WHILE;

    IF isUpper AND isLower THEN
        SET type = "MIXCASE";
    ELSEIF isUpper THEN
```

```
        SET type = "UPPER";
    ELSE
        SET type = "LOWER";
    END IF;
END//
DELIMITER ;
```

4) WAP which accepts the student rollno and returns the highest percent and name of that student to the calling block.

STUDENT(Stud_ID,Stud_name,percent);

```
CREATE TABLE IF NOT EXISTS students1 (
  rollno INT(2) PRIMARY KEY,
  name VARCHAR(30) NOT NULL,
  percent FLOAT
);

INSERT INTO students1 VALUES(1, 'Ajinkya', 45.65);
```

```
DELIMITER //
CREATE OR REPLACE PROCEDURE getHighestStudent(OUT std_name VARCHAR(30), OUT std_per FLOAT)
BEGIN
DECLARE per FLOAT;
DECLARE nm VARCHAR(30);
SELECT MAX(percent) INTO per FROM students1;
SELECT name INTO nm FROM students1 WHERE percent = per;
SET std_name = nm;
SET std_per = per;
END//
DELIMITER ;
```

5) WAP which accepts the date of joining for specific employee and returns the years of

experience along with its name. Accept the Employee no from user.

EMP (empno, empname, DOJ);

```
CREATE TABLE IF NOT EXISTS emp5(
```

```
    empid INT PRIMARY KEY,
    empname VARCHAR(20) NOT NULL,
    doj DATE
);

INSERT INTO emp5 VALUES(1, 'Ajinkya', '2019-04-05');



DELIMITER //
CREATE OR REPLACE PROCEDURE getEmpExperience(id INT, OUT e_exp INT)
BEGIN
DECLARE count, exp INT DEFAULT -1;
DECLARE v_doj DATE;
    SELECT COUNT(empid) INTO count FROM emp5 WHERE empid = id;
    IF count = 0 THEN
        SELECT "No emp";
    ELSEIF count > 1 THEN
        SELECT "Multiple employee with same id exists!" AS "Error";
    ELSE
```

```
        SELECT doj INTO v_doj FROM emp5
WHERE empid = id;
        SET exp = DATEDIFF(CURRENT_DATE,
v_doj) / 365;
    END IF;
    SET e_exp = exp;
END//
DELIMITER ;
```

6) WAP which accepts the student roll no and returns the result (in the form of class: first class, second class, third class or fail).

STUDENT (Stud_ID, Stud_name,m1, m2, m3).

```
CREATE TABLE IF NOT EXISTS students (
  rollno INT(2) NOT NULL PRIMARY KEY,
  name VARCHAR(30) NOT NULL,
  city VARCHAR(20) NOT NULL,
  marks1 FLOAT,
  marks2 FLOAT,
  marks3 FLOAT
```

```
);

INSERT INTO students VALUES(1, 'Ajinkya',
'Pune', 35, 20, 100);

DELIMITER //
CREATE OR REPLACE PROCEDURE getResult(roll
INT, OUT class VARCHAR(20))
BEGIN
DECLARE per FLOAT;
    SELECT ((marks1 + marks2 + marks3) / 3)
INTO per FROM students WHERE rollno = roll;
    IF per IS NULL THEN
        SELECT "No such student" AS "Error";
        SET class = "NULL";
    ELSE
        IF per >= 80 THEN
            SET class = "1st Class";
        ELSEIF per >= 60 THEN
            SET class = "2nd Class";
        ELSEIF per >= 40 THEN
            SET class = "3rd Class";
        ELSE
```

```
            SET class = "Failed ## ";
        END IF;
    END IF;
END//
```

==========

Assignment 3: Triggers

==========

==========
==========
========

TRIGGERS 2

==========
==========
========

name : Ajinkya

roll no : 30

==========
==========
==*

QUESTION 1 : (example 11.2) : This example is divided in three categories :

Insert, Update and Delete Insert

a. Write a trigger which updates the sale value if customer already exists

else create new entry of customer

==========
==========
==*

```
 -- (ex.11-1)


 delimiter //
 create or replace trigger sales_bi_trg BEFORE
INSERT ON sales
      for each row
```

```
begin
declare row_cnt int;

select customer_id into row_cnt from sales
where customer_id = NEW.customer_id;
if(row_cnt>0) then
update sales_s1 set sale_value =
NEW.sale_value where customer_id =
NEW.customer_id;
else
insert into
sales_s1(customer_id,sale_value)
values(NEW.customer_id,NEW.sale_value);
end if;
end //

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

create table sale(
salesid int(10) primary key,
customerid int(10),
```

```
    salesname varchar(20),
    sale_value decimal(6,2)
);
create table customer_sales_table(
    customerid int(10) primary key,
    sale_value decimal(6,2)
);


Delimiter //
create trigger sales_bi_trg
    BEFORE INSERT ON sale
    FOR EACH ROW
    BEGIN
    Declare row_count Integer;
    Select Count(*) INTO row_count from
customer_sales_table
    where CustomerID  =  NEW.CustomerID;

    IF row_count > 0 THEN
    Update customer_sales_table
```

```
    Set sale_value  =  sale_value +
NEW.Sale_value
    Where CustomerID  =  NEW.CustomerID;
    ELSE
    Insert into customer_sales_table
    (CustomerID,sale_value)
    values
    (NEW.CustomerID,NEW.Sale_value);
    END IF;
    END //
```

Output :

```
select * from customer_sales_table;

insert into sale(
      customerid,
      salesname,
      sale_value)
values
(1,'pendrive',50.00);
```

select *from customer_sales_table;

==========
==========
=====

b. Update : If the customer is updating , WAT to update the sales value by

incrementing the Sale_vale field

==========
==========
=====

-- (ex.11-1) s

delimiter //

create or replace trigger sales_bi_trg AFTER UPDATE ON sales

FOR EACH ROW

begin

```
        update sales_s1 set sale_value =
sale_value+(NEW.sale_value-OLD.sale_value)
            where customer_id =
NEW.customer_id;
            end //
            delimiter ;


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

delimiter //
create trigger sales_bu_trg
    before update on sale
    for each row
    begin
    update customer_sales_table
        set sale_value  =  sale_value +
(NEW.Sale_value-OLD.Sale_value)
        where CustomerID  =  NEW.CustomerID;
        END//
```

Output :

select *from customer_sales_table;

update sale set sale_value = 100.00 where salesid = 2 and customerid = 2;

select *from customer_sales_table;

```
+----------------+---------------+
| customerid | sale_value |
+----------------+---------------+
|        1        |    450.00   |
|        2        |    100.00   |
|        3        |   1000.00   |
+----------------+---------------+
```

==========
==========
==*

c. Delete : If the customer is deleting , WAT to update the sales value by

decrementing the Sale_vale field

==========
==========
==*

--(ex.11-1) delete //completed

```
delimiter //
create or replace trigger sales_bi_trg BEFORE
DELETE ON sales
    FOR EACH ROW
    begin
    update sales_s1 set sale_value = sale_value-
OLD.sale_value
        where customer_id = OLD.customer_id;
        end //
        delimiter ;
```



```
delimiter //
```

```
create trigger delete_totalby_trg
      Before delete on sale
      for each row
      begin
      update customer_sales_table
           set sale_value  =  sale_value -
(OLD.Sale_value)
           where CustomerID  =  OLD.CustomerID;
           end//

           Output :

           select *from customer_sales_table;
+-----------------+----------------+
| customerid | sale_value |
+-----------------+----------------+
|         1         |     450.00    |
|         2         |     100.00    |
|         3         |    1000.00   |
+-----------------+----------------+
3 rows in set (0.019 sec)
```

```
delete from sale where salesid = 3 and
customerid = 3;
```

Query OK, 1 row affected (0.080 sec)

select *from customer_sales_table;

```
+----------------+---------------+
| customerid | sale_value |
+----------------+---------------+
|        1        |    450.00    |
|        2        |    100.00    |
|        3        |      0.00     |
+----------------+---------------+
```

==========
==========
==

-- QUESTION 2 :

-- Q(example 11.4) Wirte a program to create trigger signal to restrict entering negative value in balance.

==========
==========
=====*

```
delimiter //
create trigger acc_res BEFORE UPDATE ON
account FOR EACH ROW
    begin
    if(NEW.amount<0) then
    SIGNAL SQLSTATE '80000'
    set message_text = 'Account amount cannot
be less than 0';
    end if;
    end //
```



```
delimiter //
create trigger validate_nagative
    Before Insert on sale
    for each row
```

```
begin
IF(NEW.Sale_value < '0') Then
SIGNAL SQLSTATE '80000'
SET MESSAGE_TEXT  =  'negative value invalid';
END IF;
end//
Query OK, 0 rows affected (0.163 sec)
```

Output :

```
insert into sale values(4,4,'keyboard',-500);
    ERROR 1644 (80000): negative value invalid
```

==========

-- QUESTION 3 :

-- Q(example 11.5) Write a trigger to perform data validation using select statement.

==========*

-- example 11-5 // executes even if the amount is in negative //trigger on account

```
delimiter //
create or replace trigger acc_balance BEFORE update ON account FOR EACH ROW
    begin
    declare dummy int;
    if (NEW.amount<0) then
    select 'Amount balanace has to be greater than 0' into dummy
    from account
    where account_id = NEW.account_id;
    end if;
    end //
    delimiter ;
```

```
--------------------------------------------------
--------------------------------------------------
--------------------------------------------------
----------------------------------------


delimiter //
create trigger data_validation
	Before insert on sale
	for each row
	begin
	Declare dummy_message varchar(200);
	IF(NEW.SalesName  =  '' or NEW.SalesName
=  ' ') THEN
	SIGNAL SQLSTATE '80000'
	SET MESSAGE_TEXT  =  'Enter something in
sales name';
	END IF;
	IF(NEW.Sale_value < '0') Then
	SIGNAL SQLSTATE '80000'
	SET MESSAGE_TEXT  =  'dont put negative
value in sales value';
	END IF;
	end//
```

Query OK, 0 rows affected (0.171 sec)

Output :

insert into sale values (5,5,'',5000.00);

ERROR 1644 (80000): Enter something in sales name

insert into sale values (5,5,'',-500);

ERROR 1644 (80000): dont put negative value in sales value

==========

QUESTION 4 :

Q(figure 2.17) :write a example to create a sales table whichprovides free shipping on orders above 500

==========

==========
=====

```
create table sales(salesid
int(10),saleproduct varchar(20),value
decimal(6,2),free_shipping varchar(2),discount
decimal(4,2));

Delimiter //
Create TRIGGER sale_bi_trg
BEFORE INSERT ON sales
For Each Row
BEGIN
IF NEW.value>500 THEN
SET NEW.free_shipping  =  'Y';
ELSE
SET NEW.free_shipping = 'N';
END IF;

IF NEW.value>1000 THEN
SET NEW.discount  =
NEW.value*.15;
ELSE
```

```
SET NEW.discount = 0;
END IF;
END//


insert into sales(
    salesid,saleproduct,value
    )
values(1,'purse',2000
    );

insert into sales(
    salesid,saleproduct,value)
values(2,'shoes',1000
    );

insert into sales(
    salesid,saleproduct,value)
values(3,'watch',700
    );

    insert into sales(
```

```
salesid,saleproduct,value) values(4,'belt',500
);

select *from sales;

+----------+----------------+-----------+-------
-----------+------------+
| salesid | saleproduct | value     | free_shipping
| discount |
+----------+----------------+-----------+-------
-----------+------------+
|       2     | shoes           | 1000.00 |           Y
|     0.00    |
|       3     | watch           |  700.00 |           Y
|     0.00    |
|       4     | belt            |  500.00 |           N
|     0.00    |
|       1     | purse           | 2000.00 |           Y
|    99.99   |
+----------+----------------+-----------+-------
-----------+------------+
```

==========
==========
==*

QUESTION 7 :

Q(example 8.3) : create a procedure which displays use of Savepoint with a transaction

==========
==========
===

```
delimiter //
create procedure
savepoint_example(in_depart_name
varchar(100),in_location
varchar(100),in_address1 varchar(100),
    in_address2 varchar(100),in_zipcode
varchar(100),in_manager_id int)
begin
declare sp_loc int default 0;
declare duplicate_dept int default 0;
```

```
start transaction;

select count(*) into sp_loc from location where
location = in_location;

if sp_loc = 0 then

insert into audit_log(audit_message)
    values(concat('creating New
location',in_location));

    insert into
location(location,address1,address2,zipcode)

    values(in_location,in_address1,in_address2,i
n_zipcode);

        else

        update location set address1 =
in_address1,address2 = in_address2,
            zipcode = in_zipcode where location-
in_location;
```

```
        end if;

        SAVEPOINT location_exist;

        begin
        declare duplicate_key condition for
1062;
        declare continue handler for
duplicate_key
        begin
        set duplicate_dept = 1;
        ROLLBACK to SAVEPOINT
location_exist;
        end;

        insert into audit_log(audit_message)
            values(concat('creating New
department',in_depart_name));

            insert into
department(dept_name,location,manager_id)
```

```
    values(in_depart_name,in_location,in_manag
er_id);

                    if duplicate_dept = 1 then

                    update department set
location = in_location,manager_id =
in_manager_id
                        where dept_name =
in_depart_name;

                        end if;

                        end;

                        commit;

                        end //
                        delimiter;

-----------------------------------------------
-----------------------------------------------
```

```
-------------------------------------------------
-----------------------------------------

create table audit_log ( audit_message
varchar(100) );

     create table location(location
varchar(50),address1 varchar(200),address2
varchar(200),zipcode int(6));

          create table department(dept_name
varchar(20),manager_id int(10),location
varchar(50));

delimiter //
create procedure
savepoint_example(in_depart_name
varchar(100),in_location
varchar(100),in_address1 varchar(100),
     in_address2 varchar(100),in_zipcode
varchar(100),in_manager_id int)
begin
declare sp_loc int default 0;
declare duplicate_dept int default 0;
```

```
start transaction;

select count(*) into sp_loc from location where
location = in_location;

if sp_loc = 0 then

insert into audit_log(audit_message)
    values(concat('creating New
location',in_location));

    insert into
location(location,address1,address2,zipcode)

    values(in_location,in_address1,in_address2,i
n_zipcode);

        else

        update location set address1 =
in_address1,address2 = in_address2,
            zipcode = in_zipcode where location-
in_location;
```

```
        end if;

        SAVEPOINT location_exist;

        begin
        declare duplicate_key condition for
1062;
        declare continue handler for
duplicate_key
        begin
        set duplicate_dept = 1;
        ROLLBACK to SAVEPOINT
location_exist;
        end;

        insert into audit_log(audit_message)
            values(concat('creating New
department',in_depart_name));

            insert into
department(dept_name,location,manager_id)
```

```
    values(in_depart_name,in_location,in_manag
er_id);

                    if duplicate_dept = 1 then

                    update department set
location = in_location,manager_id =
in_manager_id
                        where dept_name =
in_depart_name;

                        end if;

                        end;

                        commit;

                        end //
```

==========
==========
==*

-- QUESTION 5 :

-- Q(example 8.1) : Create a procedure to commence a transaction using auto commit.

==========
==========
===

```
delimiter //
create procedure tfer_fund(from_acc int,to_acc int,tfer_amt decimal(8,2))
    begin
    set autocommit = 0;
    update account set amount = amount-tfer_amt where account_id = from_acc;
        update account set amount = amount+tfer_amt where account_id = to_acc;
            commit;
            end //
            delimiter ;
```

```
--------------------------------------------------
--------------------------------------------------
--------------------------------------------------
----------------------------------------

create table account(
	account_id int(10),
	account_holder varchar(10),
	contact int(10),
	balance decimal(8,2));

insert into account values
	(1,'ajinkya',96387,2000),
	(2,'vraj',99499,2000),

	CREATE PROCEDURE tfer_funds
	(from_account int, to_account
int,tfer_amount numeric(10,2))
	BEGIN
	SET autocommit = 0;
	UPDATE account
	SET balance = balance-tfer_amount
	WHERE account_id = from_account;
```

```
    UPDATE account
    SET balance = balance+tfer_amount
    WHERE account_id = to_account;
    COMMIT;
    END //

    ================================
================================
=====
-- QUESTION 6 :
-- create a stored procedure to commence
transaction with start transaction
================================
================================
===



delimiter //
create procedure tfer_fund1(from_acc int,to_acc
int,tfer_amt decimal(8,2))
    begin
    start transaction;
```

```
update account
    set amount = amount-tfer_amt
    where account_id = from_acc;

    update account
        set amount = amount+tfer_amt
        where account_id = to_acc;

        commit;
        end //
        delimiter ;
```

```
delimiter //
CREATE PROCEDURE tfer_fund
(from_account int, to_account int,tfer_amount
numeric(10,2))
BEGIN
START TRANSACTION;
```

```
UPDATE account
SET balance = balance-tfer_amount
WHERE account_id = from_account;
UPDATE account
SET balance = balance+tfer_amount
WHERE account_id = to_account;
COMMIT;
END//
```

```
select * from account //
+----------------+--------------------+-----------
---------+-------------+
| account_id | account_holder | contact          |
balance  |
+----------------+--------------------+-----------
---------+-------------+
|          1          |     ajinkya           |  96387 |
2000.00 |
|          2          |     vraj           | 99499 |
2000.00 |
+----------------+--------------------+---------
---------+-------------+
```

call tfer_fund(2,'1000') //

select * from account;

```
+---------------+---------------------+-----------+-------------+
| account_id | account_holder | contact          | balance    |
+---------------+---------------------+-----------+-------------+
|          1          | ajinkya                |  96387 | 2000.00 |
|          2          | vraj                 |  99499 | 3000.00 |
+---------------+---------------------+-----------+-------------+
```

==========

Assignment 3: Transactions

==========

5. Q(example 8.1) : Create a procedure to commence a transaction using auto commit.

6. Q(example 8.2) : Create a procedure to commence a transaction using start transaction.

7. Q(example 8.3) : create a procedure which displays use of Savepoint with a transaction

```
DELIMITER //
CREATE OR REPLACE PROCEDURE sales_commit(sv INT, uid INT)
BEGIN
    DECLARE any_value INT;
    START TRANSACTION;
        SET autocommit = OFF;
        SAVEPOINT aj_savepoint;
        INSERT INTO aj_table(sv1, customer_id) VALUES(sv, uid);
        SELECT any_value(sv1) INTO any_value FROM aj_table WHERE customer_id = uid;
```

```
        IF (any_value) THEN
            ROLLBACK TO SAVEPOINT
aj_savepoint;
            SELECT "Any message";
        END IF;
    COMMIT;
END//
```

==========

TRIGGERS: DIY

==========

NAME : Ajinkya Rathod

ROLL : 30

==========

QUESTION 1 : Write a TRIGGER that stores the old data TABLE of

student TABLE in student_backup WHILE updating the student TABLE.

Student_backup (Stud_ID, Stud_name, Address, Contact_no, Branch, Operation_date)

Student (Stud_ID, Stud_name, Address, Contact_no, Branch)

==========

```
CREATE TABLE student(
    stud_id int(10),
    stud_name varchar(20),
    address varchar(100),
    contact_no varchar(13),
    branch varchar(10));
```

```
INSERT INTO student values(1, 'Ajinkya', 'PLot 742', '9685479855', 'Analysis');
```

```
CREATE OR REPLACE TABLE student_backup(
```

```
    stud_id int(10),
    stud_name varchar(20),
    address varchar(20),
    contact_no varchar(13),
    branch varchar(10),
    operation_date date);

DELIMITER //
CREATE OR REPLACE TRIGGER stud_UPDATE
BEFORE UPDATE on student
FOR EACH row
BEGIN
INSERT INTO student_backup
values(old.stud_id,old.stud_name,old.address,ol
d.contact_no,old.branch,curdate());
END//
```

****

Output:

```
SELECT * from student_backup //
```

Empty

UPDATE student SET stud_name = 'Dev' WHERE stud_id = 1 //

SELECT * from student_backup //

+---------+-----------+----------+------------+-----------+---------------+

| stud_id | stud_name | address | contact_no | branch | operation_date |

+---------+-----------+----------+------------+-----------+---------------+

| 1 | Ajinkya | PLot 742 | 9685479855 | Analysis | 2020-05-15 |

+---------+-----------+----------+------------+-----------+---------------+

****

==========

QUESTION 2 : Write a TRIGGER, that ensures the e_id of emp TABLE is in a format ‘E00001’ (e_id must start with ‘E’ and must be 6 characters long).

IF not, than complete e_id with this format BEFORE inserting INTO the employee TABLE.

```
==========
==========
==========

DELIMITER //

CREATE or REPLACE TRIGGER employee BEFORE
INSERT on emp FOR EACH row
BEGIN
DECLARE len int;
DECLARE temp varchar(6);
DECLARE chr varchar(1);

SET len=length(new.e_id);

IF len>6 then
signal sqlstate "45000"
SET message_text="Max 6 ";
ELSE
SET chr=substr(new.e_id,1,1);
IF chr="E" then
SET temp=substr(new.e_id,2);
SET new.e_id="E";
```

```
IF len=1 then
SET new.e_id=concat(new.e_id,"00000");
ELSEIF len=2 then
SET new.e_id=concat(new.e_id,"0000",temp);
ELSEIF len=3 then
SET new.e_id=concat(new.e_id,"000",temp);
ELSEIF len=4 then
SET new.e_id=concat(new.e_id,"00",temp);
ELSEIF len=5 then
SET new.e_id=concat(new.e_id,"0",temp);
ELSEIF len=6 then
SET new.e_id=concat(new.e_id,temp);
END IF;

ELSE
SET temp=new.e_id;
SET new.e_id="E";

IF len=1 then
SET new.e_id=concat(new.e_id,"0000",temp);
ELSEIF len=2 then
SET new.e_id=concat(new.e_id,"000",temp);
```

```
ELSEIF len=3 then
SET new.e_id=concat(new.e_id,"00",temp);
ELSEIF len=4 then
SET new.e_id=concat(new.e_id,"0",temp);
ELSEIF len=5 then
SET new.e_id=concat(new.e_id,temp);
ELSEIF len=6 then
signal sqlstate "45000"
SET message_text="InValid";

END IF;
END IF;
END IF;
END //
```

****

OUTPUT :

```
INSERT INTO emp
values('','Prem',86000,1,2020/05/08) //
```

****

==========
==========
==========

QUESTION 3 : Write a TRIGGER which checks the age of employee WHILE inserting the record in emp TABLE. IF it is negative than generate the error and

display proper message

==========
==========
==========

```
DELIMITER //
CREATE or REPLACE TRIGGER emp_age BEFORE INSERT on emp FOR EACH row
BEGIN
IF new.age<0 then
signal sqlstate "45000"
SET message_text="Age InValid";
END IF;
END //
```

****

OUTPUT :

INSERT INTO emp values('','Lol',10000, 3, NULL, -15) //

ERROR 1644 (45000): Age InValid

****

==========

==========

==========

QUESTION 4 : Write a TRIGGER which converts the employee name in upper case

IF it is inserted in any other case.

Change should be done BEFORE insertion only.

==========

==========

==========

DELIMITER //

```
CREATE or REPLACE TRIGGER emp_name
BEFORE INSERT on emp FOR EACH row
BEGIN
DECLARE i int default 1;
DECLARE len int;
DECLARE ch varchar(1);
DECLARE temp1 varchar(30);
DECLARE temp2 varchar(30);

SET len=length(new.e_name);

WHILE i<=len do
SET ch=substr(new.e_name,i,1);

IF ascii(ch)>96 and ascii(ch)<123 then
SET ch=chr(ascii(ch)-32);
SET temp1=substr(new.e_name,1,i-1);
SET temp2=substr(new.e_name,i+1);
SET new.e_name=concat(temp1,ch,temp2);
END IF;
SET i=i+1;
END WHILE;
```

END //

****

OUTPUT :

INSERT INTO emp values('','Lol',10000, 3, NULL, 25) //

SELECT * FROM emp //

```
+------+----------------+----------+---------+---------------+------+
| e_id | e_name         | e_salary  | dept_id | date_of_joining | age  |
+------+----------------+----------+---------+---------------+------+
16 | LOL            |  10006.00 |       3 | NULL          |   25 |
+------+----------------+----------+---------+---------------+------+
```

****

==========
==========
==========

QUESTION 5 : WAT that stores the data of emp TABLE in emp_backup TABLE FOR every DELETE operation and store the old data FOR every UPDATE operation.

EMP(e_id, Empname, salary);
Emp_Backup(e_id,Empname,Date_of_operation, Type_of_operation (i.e.UPDATE or DELETE));

==========
==========
==========

```
CREATE TABLE emp2(
    e_id int(10),
    empname varchar(10),
    salary int(10));

CREATE TABLE emp2_backup(
    emppno int(10),
    empname varchar(10),
    date_of_operation date,
    type_of_operation varchar(10));
```

```
INSERT INTO emp values
(1,'Ajinkya',10000),
(2,'Dev',20000),
(3,'Lax',30000),
(4,'Prad',40000),
(5,'Neer',50000);

SELECT * FROM emp //

DELIMITER //
CREATE or REPLACE TRIGGER emp_del BEFORE
DELETE on emp2 FOR EACH row
BEGIN
INSERT INTO emp2_backup
values(old.e_id,old.empname,sysdate(),"DELETE
");
END //

DELIMITER //
CREATE or REPLACE TRIGGER emp_UPDATE
BEFORE UPDATE on emp2 FOR EACH row
BEGIN
```

```
INSERT INTO emp2_backup
values(old.e_id,old.empname,sysdate(),"UPDATE
");
END //
```

****

OUTPUT :

```
DELETE from emp2 where e_id=4 //

UPDATE emp2 SET salary=45000 where e_id=1
//
```

****

==========

QUESTION 6 : WAT which display the message ‘Updating’,’Deleting’ or ’inserting’ when UPDATE, DELETE or INSERT operation is perFORmed on the

emp TABLE respectively.

==========
==========
==========

DELIMITER //

CREATE TRIGGER emp_INSERT_res BEFORE INSERT ON emp FOR EACH ROW

BEGIN

SIGNAL SQLSTATE '80000'

SET MESSAGE_TEXT='inserting An EMP.';

END //

****

INSERT INTO emp values('', 'Jinal', 10000, 5, NULL, DEFAULT);

ERROR 1644 (80000): inserting An EMP.

****

DELIMITER //

CREATE TRIGGER emp_UPDATE_res BEFORE UPDATE ON emp FOR EACH ROW

```
BEGIN
SIGNAL SQLSTATE '80000'
SET MESSAGE_TEXT='Updating An EMP.';
END //

**********************************************

UPDATE emp SET e_salary=10000 where e_id=3
//
ERROR 1644 (80000): updating An EMP.
**********************************************



DELIMITER //
CREATE TRIGGER emp_DELETE_res BEFORE
DELETE ON emp  FOR EACH ROW
BEGIN
SIGNAL SQLSTATE '80000'
SET MESSAGE_TEXT='Deleting An EMP.';
END //

**********************************************

DELETE from emp where e_id=1;
```

ERROR 1644 (80000): deleting An EMP.

****

==========

QUESTION 7 : WAT which generate an error IF any user try to DELETE from product_master TABLE on weekENDs

(i.e. Saturday and Sunday).

==========

```
CREATE TABLE product_master(
    pno varchar(10),
    pname varchar(10),
    stock int(10)) //

INSERT INTO product_master values
('p01','usb',10),
('p02','lappi',15),
('p03','pen drive',20) //
```

```
DELIMITER //
CREATE or REPLACE TRIGGER pro_del BEFORE
DELETE on product_master FOR EACH row
BEGIN
DECLARE day int;
SET day = dayofweek(curdate());
IF day = 1 or day = 7 then
signal sqlstate "45000"
SET message_text="DELETE NOT PSSIBLE";
END IF;
END //
```

****

```
DELETE from product_master where p_id="p01";
ERROR 1644 (45000): DELETE NOT PSSIBLE
```

****

==========
==========
==========

QUESTION 8 : We have two TABLEs student_mast and stu_log. student_mast have three column STUDENT_ID, NAME, ST_CLASS. stu_log TABLE has

two columns user_id and description. WAT which INSERTs the student details in stu_log TABLE as soon as we promote the students in

student master TABLE(e.g. when a student is promoted from sem 2 to 3, auto entry in log TABLE)

==========
==========
==========

```
CREATE TABLE stud_master(
    student_id int PRIMARY KEY,
    name varchar(255),
    class int) //

CREATE TABLE stud_log(
    user_id int,
    description varchar(255)) //
```

```
INSERT INTO stud_master values(
    1,'Ajinkya',2),
    (2,'Nirav',6) //


DELIMITER //
CREATE TRIGGER stud_log BEFORE UPDATE ON stud_master  FOR EACH ROW
BEGIN
DECLARE DES VARCHAR(255) DEFAULT ' ';
DECLARE SID INT;
DECLARE SEM_NEW INT;
DECLARE SEM_OLD INT;
SET SEM_OLD=OLD.CLASS;
SET SEM_NEW =SEM_OLD +1;
SET DES= CONCAT('student is promoted from sem ',SEM_OLD,'to',SEM_NEW,DES);
SET SID=OLD.STUDENT_ID;
INSERT INTO STUD_LOG VALUES(SID,DES);
END//
```

****

OUTPUT:

UPDATE stud_master SET class=class + 1 where student_id = 1 //

```
+---------+-----------------------------------+
| user_id | description                       |
+---------+-----------------------------------+
|       1 | student is promoted from sem 2to3  |
+---------+-----------------------------------+
```

****

==========

QUESTION 9 : WAT to calculate the Income Tax amount and INSERT it in emp TABLE.
EMP(emp_no,emp_name, emp_income, income_tax);

IF emp_income <100000 and >=50000 then incometax = 10%

IF emp_income <200000 and >=100000 then incometax = 15%

IF emp_income <300000 and >=200000 then incometax = 20%

==========

```
CREATE TABLE emp3(
    e_id int(10),
    empname varchar(20),
    salary int(10),
    income_tax decimal(6,2));

DELIMITER //
CREATE or REPLACE TRIGGER emp3_tax BEFORE INSERT on emp3 FOR EACH row
BEGIN
IF new.salary<100000 and new.salary >=50000 then
SET new.income_tax=(new.salary*10)/100;
ELSEIF new.salary<200000 and new.salary >=100000 then
SET new.income_tax=(new.salary*15)/100;
ELSEIF new.salary<300000 and new.salary >=200000 then
```

```
SET new.income_tax=(new.salary*20)/100;
ELSE
SET new.income_tax="0";
END IF;

END //
```

****

OUTPUT :

```
INSERT INTO emp3(e_id,empname,salary)
values(1,'Thindi',60000),(2,'Thindo',78110000)
//
SELECT * FROM emp3 //
```

```
+------+---------+----------+------------+
| e_id | empname | salary   | income_tax |
+------+---------+----------+------------+
|    1 | Thindi  |    60000 |    6000.00 |
|    2 | Thindo  | 78110000 |       0.00 |
+------+---------+----------+------------+
```

****

==========
==========**

9) WAT to calculate the Income Tax amount and insert it in emp table. EMP(emp_no,emp_name,

emp_income, income_tax);

If emp_income <100000 and >=50000 then incometax = 10%

If emp_income <200000 and >=100000 then incometax = 15%

If emp_income <300000 and >=200000 then incometax = 20%

CREATE TABLE IF NOT EXISTS Emp_Master(

```
    emp_no INT PRIMARY KEY
AUTO_INCREMENT,
    emp_name VARCHAR(30),
    emp_income FLOAT,
    income_tax FLOAT
    );



DELIMITER //
CREATE OR REPLACE TRIGGER setIncomeTax
BEFORE INSERT ON Emp_Master FOR EACH ROW
BEGIN
    IF NEW.emp_income >= 200000 AND
NEW.emp_income < 300000 THEN
        SET NEW.income_tax =
(NEW.emp_income * 20) / 100;
    ELSEIF NEW.emp_income >= 50000 AND
NEW.emp_income < 100000 THEN
        SET NEW.income_tax =
(NEW.emp_income * 10) / 100;
    ELSEIF NEW.emp_income >= 100000 AND
NEW.emp_income < 200000 THEN
        SET NEW.income_tax =
(NEW.emp_income * 15) / 100;
```

```
    ELSE
        SET NEW.income_tax = 0;
    END IF;
END//
```