Disciplinary

Database Management System

CS3003

Assignment 4

Sheikh Muhammed Tadeeb (AU19B1014)

❖ Problem Statement:

Write the PL/SQL constructs on the company database schema to demonstrate ACID properties and accomplish the following tasks: -

1) Creating five meaningful stored procedures on the database company.

2) Creating five meaningful cursors on the database company.

3) Creating five meaningful triggers on the database company.

❖ Part 1 (Creating Stored procedures):

1) Creating a stored procedure to get the department number of the employee by entering his/her SSN and first name.

Functionalities Used	Purpose
1) IN parameter	To take user entry and put them inside stored procedure
2) Joins	For checking the name and Ssn from employee table and displaying Department number from department table

```
MariaDB [company]>  CREATE PROCEDURE Get_emp_dept ( IN i_ssn int(5), IN i_Fname varchar(20))
    -> BEGIN
    -> Select Dno from department d RIGHT JOIN employee e ON d.Dnumber = e.Dno where e.Ssn = i_ssn AND e.F_name = i_Fname;
    -> END //
Query OK, 0 rows affected (0.012 sec)

MariaDB [company]> call Get_emp_dept(4534,'Preeti') //
+-----+
| Dno |
+-----+
|   5 |
+-----+
1 row in set (0.006 sec)
```

2) Creating a stored procedure for changing the project location and confirming upon the change.

Functionalities Used	Purpose
1) IN parameter	To take user entry and put them inside stored procedure
2) OUT parameter	To display out the value to the end user.
3) IF statement	The IF-THEN statement allows us to execute a set of SQL statements based on a specified condition.

```
MariaDB [company]> delimiter //
MariaDB [company]> CREATE PROCEDURE update_location (IN o_location varchar(30),IN n_location varchar(30),OUT status char(40))
    ->      BEGIN
    ->      IF o_location IN ('Pune','Indore','Ujjain') then
    ->      Update project SET Plocation = n_location where Plocation = o_location;
    ->      SET status = 'done';
    ->      Select status;
    ->      else
    ->      SET status = 'No department in the city';
    ->      Select status;
    ->      END IF;
    ->      END //
Query OK, 0 rows affected (0.010 sec)

MariaDB [company]> call update_location('GOA','Dewas',@status);
    -> //
+--------------------------+
| status                   |
+--------------------------+
| No department in the city |
+--------------------------+
1 row in set (0.000 sec)
```

```
MariaDB [company]> delimiter ;
MariaDB [company]> call update_location('Pune','Dewas',@status);
+---------+
| status  |
+---------+
| done    |
+---------+
1 row in set (0.000 sec)
```

3) Incrementing the salary of employee by entered percent and criteria.

```
MariaDB [company]> delimiter //
MariaDB [company]> CREATE PROCEDURE increment (IN percent int(3),IN sal int(11))
    ->    BEGIN
    ->    Declare increase int(3);
    ->    Declare s_value int(11);
    ->    Declare r_num int(8);
    ->    Declare msg varchar(30);
    ->    SET msg = 'All employees have salary greater than 50000';
    ->    SET increase = percent;
    ->    Select COUNT(*) INTO s_value from employee where Salary < sal;
    ->    Select COUNT(*) INTO r_num from employee;
    ->    While  s_value > 0 DO
    ->    IF s_value > 0  then
    ->    Select Salary+Salary*(increase/100) As 'Incremented Salary' from employee;
    ->    SET s_value = -1;
    ->    ELSE
    ->    Select msg;
    ->    END IF;
    ->    END WHILE;
    ->    END //
Query OK, 0 rows affected (0.010 sec)

MariaDB [company]> delimiter ;
MariaDB [company]> call increment(5,60000);
+--------------------+
| Incremented Salary |
+--------------------+
|         31500.0000 |
|         42000.0000 |
|         26250.0000 |
|        210000.0000 |
|         57750.0000 |
|        105000.0000 |
|        215250.0000 |
|         26250.0000 |
+--------------------+
8 rows in set (0.001 sec)

Query OK, 2 rows affected (0.009 sec)
```

Functionalities Used	Purpose
1) IN parameter	To take user entry and put them inside stored procedure
2) Variable (Declare, SET)	Guide us on how to use variables to hold immediate result inside stored procedures.
3) WHILE LOOP	Show us how to execute a loop as long as a condition is true.
4) IF statement	The IF-THEN statement allows us to execute a set of SQL statements based on a specified condition.

4) Retrieve the name of each employee who works on all the projects controlled by some specific department number.

Functionalities Used	Purpose
5) IN parameter	To take user entry and put them inside stored procedure
6) NOT EXISTS	The NOT EXISTS in SQL Server will check the Subquery for rows existence, and if there are no rows then it will return TRUE, otherwise FALSE.
7) EXCEPT	The SQL EXCEPT clause/operator is used to combine two SELECT statements and returns rows from the first SELECT statement that are not returned by the second SELECT statement. This means EXCEPT returns only rows, which are not available in the second SELECT statement.

```
MariaDB [company]> delimiter //
MariaDB [company]> Create procedure project_associated (IN D_num int(4) )
    -> BEGIN
    -> SELECT F_name, L_name
    -> FROM EMPLOYEE
    -> WHERE NOT EXISTS (
    -> (SELECT Pnumber FROM project WHERE Dnum = D_num) EXCEPT (SELECT Pno FROM works_on WHERE Ssn = Essn)
    -> );
    -> END //
Query OK, 0 rows affected (0.010 sec)

MariaDB [company]> delimiter ;
MariaDB [company]> call project_associated(4);
+--------+--------+
| F_name | L_name |
+--------+--------+
| John   | Smith  |
| Ramesh | Jain   |
+--------+--------+
2 rows in set (0.011 sec)

Query OK, 0 rows affected (0.014 sec)
```

5) Creating a procedure to assign an employee a new project, if the project and employee is same then the duplicity error will be executed.

Functionalities Used	Purpose
1) IN parameter	To take user entry and put them inside stored procedure
2) Error handling	Show us how to handle delicacy exception and errors in stored procedures.

```
MariaDB [company]> delimiter //
MariaDB [company]> CREATE PROCEDURE appoint_project ( IN e_Id INT(5),  IN p_num INT(5) )
    -> BEGIN
    -> DECLARE EXIT HANDLER FOR 1062 SELECT 'Duplicate keys error encountered' Message;
    -> DECLARE EXIT HANDLER FOR SQLEXCEPTION SELECT 'SQLException encountered' Message;
    -> DECLARE EXIT HANDLER FOR SQLSTATE '23000' SELECT 'SQLSTATE 23000' ErrorCode;
    -> INSERT INTO works_on (Essn,Pno) VALUES(e_Id,p_num);
    -> SELECT COUNT(*) FROM works_on WHERE Essn = e_Id;
    -> END //
Query OK, 0 rows affected (0.010 sec)

MariaDB [company]> delimiter ;
MariaDB [company]> call appoint_project(9876,5);
+---------------------------------+
| Message                         |
+---------------------------------+
| Duplicate keys error encountered |
+---------------------------------+
1 row in set (0.006 sec)

Query OK, 0 rows affected (0.009 sec)
```

❖ Checking existence of our stored procedures:

1) SELECT routine_name FROM information_schema.routines WHERE routine_type = 'PROCEDURE' AND routine_schema = 'company';

```
MariaDB [company]> SELECT routine_name FROM information_schema.routines WHERE
    ->   routine_type = 'PROCEDURE' AND routine_schema = 'company';
+--------------------+
| routine_name       |
+--------------------+
| appoint_project    |
| Get_emp_dept       |
| increment          |
| project_associated |
| update_location    |
+--------------------+
5 rows in set (0.001 sec)
```

❖ Part -2 (Creating Cursors):

1) Displaying First name and last name of each employee separately and handling NOT FOUND error.

Functionalities Used	Purpose
1) IF statement	The IF-THEN statement allows us to execute a set of SQL statements based on a specified condition.
2) NOT FOUND Error handling	Because each time you call the FETCH statement, the cursor attempts to read the next row in the result set. When the cursor reaches the end of the result set, it will not be able to get the data, and a condition is raised. The handler is used to handle this condition.
3) Variable (Declare, SET)	Guide us on how to use variables to hold immediate result inside stored procedures.
4) WHILE LOOP	Show us how to execute a loop as long as a condition is true.

```
MariaDB [company]> delimiter //
MariaDB [company]> Create procedure emp_name ()
    ->     Begin
    ->     Declare e_Fname varchar(30);
    ->     Declare e_Lname varchar(30);
    ->     Declare finished int default 0;
    ->     declare c1 cursor for select F_name,L_name from employee;
    ->     declare continue handler for NOT FOUND set finished = 1;
    ->     open c1;
    ->     get_emp : LOOP
    ->     fetch c1 into e_Fname,e_Lname;
    ->     if finished = 1 then
    ->     leave get_emp;
    ->     end if;
    ->     Select concat(e_Fname,concat('-',e_Lname));
    ->     END LOOP get_emp;
    ->     close c1;
    ->     END //
Query OK, 0 rows affected (0.010 sec)

MariaDB [company]> delimiter ;
MariaDB [company]> call emp_name();
+-----------------------------------+
| concat(e_Fname,concat('-',e_Lname)) |
+-----------------------------------+
| John-Smith                        |
+-----------------------------------+
1 row in set (0.006 sec)

+-----------------------------------+
| concat(e_Fname,concat('-',e_Lname)) |
+-----------------------------------+
| Franklin-Wong                     |
+-----------------------------------+
1 row in set (0.009 sec)

+-----------------------------------+
| concat(e_Fname,concat('-',e_Lname)) |
+-----------------------------------+
| Preeti-Pal                        |
+-----------------------------------+
```

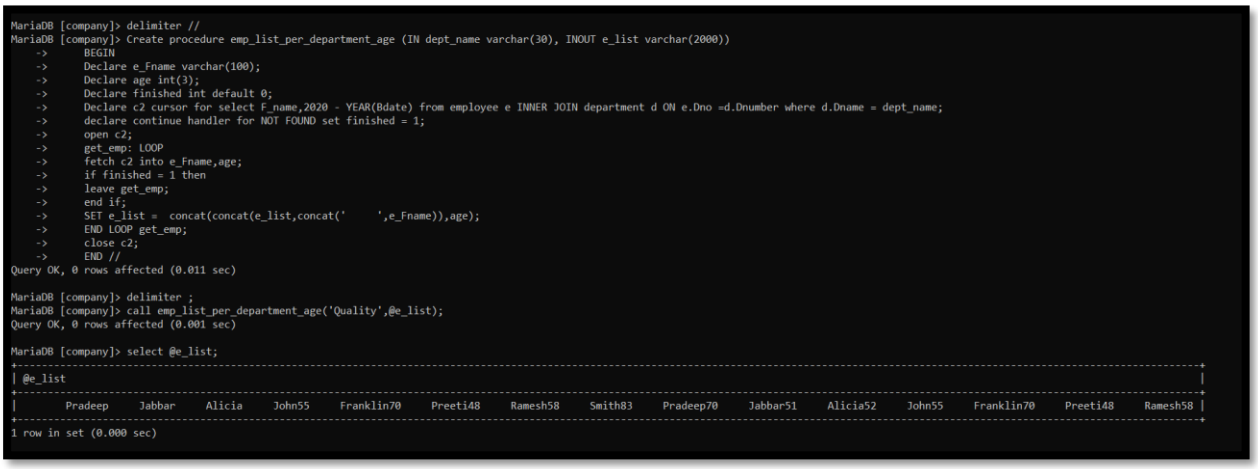
2) Creating a cursor to find the names of all employees working in a particular department finding them with their department number.

Functionalities Used	Purpose
1) IF statement	The IF-THEN statement allows us to execute a set of SQL statements based on a specified condition.
2) NOT FOUND Error handling	Because each time you call the FETCH statement, the cursor attempts to read the next row in the result set. When the cursor reaches the end of the result set, it will not be able to get the data, and a condition is raised. The handler is used to handle this condition.
3) Variable (Declare, SET)	Guide us on how to use variables to hold immediate result inside stored procedures.
4) WHILE LOOP	Show us how to execute a loop as long as a condition is true.
5) INOUT parameter	An INOUT parameter is a combination of IN and OUT parameters. It means that the calling program may pass the argument, and the stored procedure can modify the INOUT parameter, and pass the new value back to the calling program.
6) CONCAT()	The CONCAT() function adds two or more strings together.

```
MariaDB [company]> delimiter //
MariaDB [company]> Create procedure emp_list_per_department (IN e_dept int(11), INOUT e_list varchar(2000))
    ->      BEGIN
    ->      Declare e_Fname varchar(100);
    ->      Declare finished int default 0;
    ->      Declare c2 cursor for select F_name from employee where Dno = e_dept;
    ->      declare continue handler for NOT FOUND set finished = 1;
    ->      open c2;
    ->      get_emp: LOOP
    ->      fetch c2 into e_Fname;
    ->      if finished = 1 then
    ->      leave get_emp;
    ->      end if;
    ->      SET e_list =  concat(e_list,concat('     ',e_Fname));
    ->      END LOOP get_emp;
    ->      close c2;
    ->      END //
Query OK, 0 rows affected (0.010 sec)

MariaDB [company]> delimiter ;
MariaDB [company]> call emp_list_per_department(5,@e_list);
Query OK, 0 rows affected (0.001 sec)

MariaDB [company]> select @e_list ;
+-----------------------------------------------------------------------------------------------------------------------------------------------------------+
| @e_list                                                                                                                                                   |
+-----------------------------------------------------------------------------------------------------------------------------------------------------------+
|      Ramesh     John     Franklin     Preeti     Ramesh     Pradeep     Jabbar     Alicia     John     Franklin     Preeti     Ramesh |
+-----------------------------------------------------------------------------------------------------------------------------------------------------------+
1 row in set (0.000 sec)

MariaDB [company]> set @e_list = '  ';
Query OK, 0 rows affected (0.005 sec)

MariaDB [company]> call emp_list_per_department(4,@e_list);
Query OK, 0 rows affected (0.000 sec)

MariaDB [company]> select @e_list ;
+-------------------------------------+
| @e_list                             |
+-------------------------------------+
|        Pradeep     Jabbar     Alicia |
+-------------------------------------+
1 row in set (0.000 sec)
```

3) Creating a cursor to find the names of all employees working in a particular department finding them with their department name and age.

Functionalities Used	Purpose
1) IF statement	The IF-THEN statement allows us to execute a set of SQL statements based on a specified condition.
2) NOT FOUND Error handling	Because each time you call the FETCH statement, the cursor attempts to read the next row in the result set. When the cursor reaches the end of the result set, it will not be able to get the data, and a condition is raised. The handler is used to handle this condition.
3) Variable (Declare, SET)	Guide us on how to use variables to hold immediate result inside stored procedures.
4) WHILE LOOP	Show us how to execute a loop as long as a condition is true.
5) INOUT parameter	An INOUT parameter is a combination of IN and OUT parameters. It means that the calling program may pass the argument, and the stored procedure can modify the INOUT parameter, and pass the new value back to the calling program.
6) CONCAT()	The CONCAT() function adds two or more strings together.



```
MariaDB [company]> delimiter //
MariaDB [company]> Create procedure emp_list_per_department_age (IN dept_name varchar(30), INOUT e_list varchar(2000))
    ->      BEGIN
    ->      Declare e_Fname varchar(100);
    ->      Declare age int(3);
    ->      Declare finished int default 0;
    ->      Declare c2 cursor for select F_name,2020 - YEAR(Bdate) from employee e INNER JOIN department d ON e.Dno =d.Dnumber where d.Dname = dept_name;
    ->      declare continue handler for NOT FOUND set finished = 1;
    ->      open c2;
    ->      get_emp: LOOP
    ->      fetch c2 into e_Fname,age;
    ->      if finished = 1 then
    ->      leave get_emp;
    ->      end if;
    ->      SET e_list =  concat(concat(e_list,concat('     ',e_Fname)),age);
    ->      END LOOP get_emp;
    ->      close c2;
    ->      END //
Query OK, 0 rows affected (0.011 sec)

MariaDB [company]> delimiter ;
MariaDB [company]> call emp_list_per_department_age('Quality',@e_list);
Query OK, 0 rows affected (0.001 sec)

MariaDB [company]> select @e_list;
+-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------+
| @e_list                                                                                                                                                                                       |
+-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------+
|       Pradeep     Jabbar     Alicia     John55     Franklin70     Preeti48     Ramesh58     Smith83     Pradeep70     Jabbar51     Alicia52     John55     Franklin70     Preeti48     Ramesh58 |
+-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------+
1 row in set (0.000 sec)
```

4) Creating a cursor to non-uniformly incrementing the salary of only managers of a department.

Functionalities Used	Purpose
1) Multiple IF statements	The IF-THEN statement allows us to execute a set of SQL statements based on a specified condition.
2) NOT FOUND Error handling	Because each time you call the FETCH statement, the cursor attempts to read the next row in the result set. When the cursor reaches the end of the result set, it will not be able to get the data, and a condition is raised. The handler is used to handle this condition.
3) Variable (Declare, SET)	Guide us on how to use variables to hold immediate result inside stored procedures.
4) WHILE LOOP	Show us how to execute a loop as long as a condition is true.

```
MariaDB [company]> delimiter //
MariaDB [company]>  Create procedure update_salary_based_designation()
    ->          BEGIN
    ->          Declare e_ssn int(11);
    ->          Declare e_salary int(11);
    ->          Declare finished int default 0;
    ->          Declare c3 cursor for select Ssn,Salary from employee;
    ->          declare continue handler for NOT FOUND set finished = 1;
    ->          open c3;
    ->          get_emp: LOOP
    ->          fetch c3 into e_Ssn,e_salary;
    ->          if finished = 1 then
    ->          leave get_emp;
    ->          end if;
    ->
    ->          if e_ssn = 8886 then
    ->          Update employee SET Salary = Salary + 2000 where Ssn = e_ssn;
    ->          end if;
    ->
    ->          if e_ssn = 9876 then
    ->          Update employee SET Salary = Salary + 1000 where Ssn = e_ssn;
    ->          end if;
    ->
    ->          if e_ssn = 3334 then
    ->          Update employee SET Salary = Salary + 1500 where Ssn = e_ssn;
    ->          end if;
    ->
    ->          END LOOP get_emp;
    ->          close c3;
    ->          END //
Query OK, 0 rows affected (0.016 sec)

MariaDB [company]> delimiter ;
MariaDB [company]> call update_salary_based_designation();
Query OK, 3 rows affected (0.015 sec)

MariaDB [company]> select Ssn , Salary from employee;
+------+--------+
| Ssn  | Salary |
+------+--------+
| 1234 |  30000 |
| 3334 |  41500 |
| 4534 |  25000 |
| 6668 | 200000 |
| 8886 |  57000 |
| 9876 | 101000 |
| 9879 | 205000 |
| 9998 |  25000 |
+------+--------+
8 rows in set (0.000 sec)
```

```
MariaDB [company]> call update_salary_based_designation();
Query OK, 3 rows affected (0.010 sec)

MariaDB [company]> select Ssn , Salary from employee;
+------+--------+
| Ssn  | Salary |
+------+--------+
| 1234 |  30000 |
| 3334 |  43000 |
| 4534 |  25000 |
| 6668 | 200000 |
| 8886 |  59000 |
| 9876 | 102000 |
| 9879 | 205000 |
| 9998 |  25000 |
+------+--------+
8 rows in set (0.000 sec)
```

5) Creating a cursor for incrementing or decrementing the working time for a specified project.

```
MariaDB [company]> delimiter //
MariaDB [company]> Create procedure increment_decrement_project_hrs (IN time TIME(4),validation char(2),IN Pnum int(6))
    ->          BEGIN
    ->          Declare inc_time TIME;
    ->          Declare finished int default 0;
    ->          Declare c5 cursor for select Hours from works_on ;
    ->          declare continue handler for NOT FOUND set finished = 1;
    ->          open c5;
    ->          get_emp: LOOP
    ->          fetch c5 into inc_time;
    ->          if finished = 1 then
    ->          leave get_emp;
    ->          end if;
    ->
    ->          if validation = 'i' then
    ->          Update works_on SET Hours = inc_time+time where Pnum = Pno;
    ->          end if;
    ->
    ->           if validation = 'd' then
    ->          Update works_on SET Hours = inc_time-time where Pnum = Pno;
    ->          end if;
    ->
    ->          END LOOP get_emp;
    ->          close c5;
    ->          END //
Query OK, 0 rows affected (0.011 sec)

MariaDB [company]> select * from works_on //
+------+-----+----------+
| Essn | Pno | Hours    |
+------+-----+----------+
| 1234 |   1 | 33:05:00 |
| 1234 |   2 | 07:05:00 |
| 1234 |   3 | 30:10:00 |
| 3334 |   4 | 20:22:00 |
| 4534 |   1 | 20:00:00 |
| 4534 |   2 | 20:00:00 |
| 6668 |   2 | 12:00:00 |
| 6668 |   3 | 40:00:00 |
| 8886 |   5 | NULL     |
| 9876 |   1 | NULL     |
| 9876 |   5 | 15:00:00 |
+------+-----+----------+
11 rows in set (0.000 sec)
```

```
MariaDB [company]> call increment_decrement_project_hrs('01:00:00','d',2) //
Query OK, 27 rows affected (0.026 sec)

MariaDB [company]> select * from works_on //
+------+-----+----------+
| Essn | Pno | Hours    |
+------+-----+----------+
| 1234 |   1 | 33:05:00 |
| 1234 |   2 | 14:00:00 |
| 1234 |   3 | 30:10:00 |
| 3334 |   4 | 20:22:00 |
| 4534 |   1 | 20:00:00 |
| 4534 |   2 | 14:00:00 |
| 6668 |   2 | 14:00:00 |
| 6668 |   3 | 40:00:00 |
| 8886 |   5 | NULL     |
| 9876 |   1 | NULL     |
| 9876 |   5 | 15:00:00 |
+------+-----+----------+
11 rows in set (0.000 sec)
```

❖ Part – 3 (Creating triggers):

1) Maintaining audit of all the changes happening in the department table.

- **Insert:**

```
MariaDB [company]> CREATE TABLE audit_dept (
    -> Operation varchar(20),
    -> Timeops datetime ,
    -> Old_value varchar(200),
    -> New_value varchar(200),
    -> Manager_info int(8) );
Query OK, 0 rows affected (0.049 sec)

MariaDB [company]> delimiter //
MariaDB [company]> Create trigger insert_department after insert
    -> On department
    -> for each row
    -> BEGIN
    -> insert into audit_dept values ('insert' ,curdate() , NULL , concat(NEW.Dname,'-',NEW.Dnumber), NEW.Mgr_ssn);
    -> END //
Query OK, 0 rows affected (0.026 sec)

MariaDB [company]> delimiter ;
MariaDB [company]> insert into department values ('Design',8,'1988-08-11',9879);
Query OK, 1 row affected (0.011 sec)

MariaDB [company]> select * from audit_dept;
+-----------+---------------------+-----------+-----------+--------------+
| Operation | Timeops             | Old_value | New_value | Manager_info |
+-----------+---------------------+-----------+-----------+--------------+
| insert    | 2020-10-30 00:00:00 | NULL      | Design-8  |         9879 |
+-----------+---------------------+-----------+-----------+--------------+
1 row in set (0.000 sec)
```

- **Update:**

```
MariaDB [company]> delimiter //
MariaDB [company]> Create trigger update_department after update
    -> On department
    -> for each row
    -> BEGIN
    -> insert into audit_dept values ('update' ,curdate() , concat(OLD.Dname,'-',OLD.Dnumber) , concat(NEW.Dname,'-',NEW.Dnumber), NEW.Mgr_ssn);
    -> END //
Query OK, 0 rows affected (0.020 sec)

MariaDB [company]> delimiter ;
MariaDB [company]> Update department SET Dname = 'Technology',Dnumber = 9 where Dnumber = 8;
Query OK, 1 row affected (0.017 sec)
Rows matched: 1  Changed: 1  Warnings: 0

MariaDB [company]> select * from audit_dept;
+-----------+---------------------+-----------+-------------+--------------+
| Operation | Timeops             | Old_value | New_value   | Manager_info |
+-----------+---------------------+-----------+-------------+--------------+
| insert    | 2020-10-30 00:00:00 | NULL      | Design-8    |         9879 |
| update    | 2020-10-30 00:00:00 | Design-8  | Technology-9 |         9879 |
+-----------+---------------------+-----------+-------------+--------------+
2 rows in set (0.000 sec)
```

- **Delete:**

```
MariaDB [company]> delimiter //
MariaDB [company]> Create trigger delete_department after delete
    -> On department
    -> for each row
    -> BEGIN
    -> insert into audit_dept values ('delete' ,curdate() , concat(OLD.Dname,'-',OLD.Dnumber) ,NULL, OLD.Mgr_ssn);
    -> END //
Query OK, 0 rows affected (0.016 sec)

MariaDB [company]> delete from department where Dnumber = 9 //
Query OK, 1 row affected (0.009 sec)

MariaDB [company]> select * from audit_dept //
+-----------+---------------------+-------------+-------------+--------------+
| Operation | Timeops             | Old_value   | New_value   | Manager_info |
+-----------+---------------------+-------------+-------------+--------------+
| insert    | 2020-10-30 00:00:00 | NULL        | Design-8    |         9879 |
| update    | 2020-10-30 00:00:00 | Design-8    | Technology-9 |         9879 |
| delete    | 2020-10-30 00:00:00 | Technology-9 | NULL        |         9879 |
+-----------+---------------------+-------------+-------------+--------------+
3 rows in set (0.000 sec)
```

Functionalities Used	Purpose
1) NEW & OLD pseudocode	When a row-level trigger fires, the PL/SQL runtime system creates and populates the two pseudo records OLD and NEW. They are called pseudo records because they have some, but not all, of the properties of records.
2) Concat()	It CONCAT() function adds two or more strings together.
3) Curdate()	The CURDATE() function returns the current date.

2) Creating trigger for auto-incrimination of project number as project is following a uniform order.

```
MariaDB [company]> delimiter //
MariaDB [company]> create trigger trig_project before insert on project
    -> for each row
    -> BEGIN
    -> declare maxno int;
    -> select max(Pnumber) into maxno from project;
    -> SET NEW.Pnumber = maxno + 1;
    -> END //
Query OK, 0 rows affected (0.024 sec)

MariaDB [company]> delimiter ;
MariaDB [company]> insert into project values ('F',null,'Bhopal',1);
Query OK, 1 row affected (0.009 sec)

MariaDB [company]> select * from project;
+-------+---------+-----------+------+
| Pname | Pnumber | Plocation | Dnum |
+-------+---------+-----------+------+
| X     |       1 | Dewas     |    5 |
| Y     |       2 | Dewas     |    5 |
| Z     |       3 | Dewas     |    4 |
| P     |       4 | Indore    |    1 |
| C     |       5 | Ujjain    |    5 |
| F     |       6 | Bhopal    |    1 |
+-------+---------+-----------+------+
6 rows in set (0.000 sec)
```

NOTE: The above trigger has one drawback i.e. if the table is empty it won't work so we will use this trigger then :-

```
MariaDB [company]> create trigger trig_project before insert on project
    -> for each row
    -> BEGIN
    -> declare maxno int;
    -> select max(Pnumber) into maxno from project;
    -> if maxno is null then
    -> SET maxno = 1;
    -> end if;
    -> SET NEW.Pnumber = maxno + 1;
    -> END //
Query OK, 0 rows affected (0.020 sec)
```

Functionalities Used	Purpose
1) Variable (Declare, SET)	Guide us on how to use variables to hold immediate result inside stored procedures.
2) max()	Finds the maximum of given parameter.
3) IF statement	The IF-THEN statement allows us to execute a set of SQL statements based on a specified condition.

3) Creating trigger to count the number of employees working on different project of the company.

```
MariaDB [company]> delimiter //
MariaDB [company]> create trigger trig_Total after insert on works_on
    -> for each row
    -> BEGIN
    -> Declare Pnum1 int(5);
    -> Declare Pnum2 int(5);
    -> Declare Pnum3 int(5);
    -> Declare Pnum4 int(5);
    -> Declare Pnum5 int(5);
    -> Select COUNT(Essn) into Pnum1 from works_on where Pno =1 group by Pno;
    -> Select COUNT(Essn) into Pnum2 from works_on where Pno =2 group by Pno;
    -> Select COUNT(Essn) into Pnum3 from works_on where Pno =3 group by Pno;
    -> Select COUNT(Essn) into Pnum4 from works_on where Pno =4 group by Pno;
    -> Select COUNT(Essn) into Pnum5 from works_on where Pno =5 group by Pno;
    -> insert into count values (1,Pnum1);
    -> insert into count values (2,Pnum2);
    -> insert into count values (3,Pnum3);
    -> insert into count values (4,Pnum4);
    -> insert into count values (5,Pnum5);
    -> END //
Query OK, 0 rows affected (0.023 sec)

MariaDB [company]> delimiter ;
MariaDB [company]> insert into works_on values (6668,2,'12:00:00');
Query OK, 1 row affected (0.018 sec)

MariaDB [company]> select * from count ;
+---------+-----------------+
| Pnumber | employee_working |
+---------+-----------------+
|       1 |               3 |
|       2 |               3 |
|       3 |               2 |
|       4 |               1 |
|       5 |               2 |
+---------+-----------------+
5 rows in set (0.000 sec)
```

Functionalities Used	Purpose
1) Variable (Declare, SET)	Guide us on how to use variables to hold immediate result inside stored procedures.

4) Creating trigger for automatically updating department number in employee if the department number is updated for a particular Department name.

```
MariaDB [company]> delimiter //
MariaDB [company]>
MariaDB [company]> Create trigger updatation after update ON department
    -> for each row
    -> BEGIN
    -> DECLARE name varchar(20) ;
    -> DECLARE Dnum int(4);
    -> select DName into name from department where Mgr_ssn = 8886;
    ->
    -> if name = 'Research' then
    -> Update employee SET Dno = NEW.Dnumber where Dno = OLD.Dnumber;
    -> END IF;
    -> END //
Query OK, 0 rows affected (0.022 sec)

MariaDB [company]> delimiter ;
MariaDB [company]> Update department SET Dnumber = 2 where DName = 'Research' ;
ERROR 1451 (23000): Cannot delete or update a parent row: a foreign key constraint fails (`company`.`dept_location`, CONSTRAINT `dept_location_ibfk_1` FOREIGN KEY (`Dnumber
`) REFERENCES `department` (`Dnumber`))
MariaDB [company]> SET FOREIGN_key_checks = 0;
Query OK, 0 rows affected (0.006 sec)

MariaDB [company]> Update department SET Dnumber = 2 where DName = 'Research' ;
Query OK, 1 row affected (0.017 sec)
Rows matched: 1  Changed: 1  Warnings: 0

MariaDB [company]> SET FOREIGN_key_checks = 1;
Query OK, 0 rows affected (0.000 sec)

MariaDB [company]> select * from employee;
+------+---------+-------+-------+------------+--------+--------+--------------------------+-----+-----------+
| Ssn  | F_name  | M_name | L_name | Bdate      | gender | Salary | Address                  | Dno | Super_ssn |
+------+---------+-------+-------+------------+--------+--------+--------------------------+-----+-----------+
| 1234 | John    | B      | Smith  | 1965-01-09 | M      |  30000 | 12 Amarpura, Ujjain      |   5 |      3334 |
| 3334 | Franklin | T      | Wong   | 1950-12-08 | M      |  43000 | 21 Shivhare colony, Alandi |   5 |      8886 |
| 4534 | Preeti  | L      | Pal    | 1972-07-31 | F      |  25000 | 2 Shastri Nagar, Pune    |   5 |      3334 |
| 6668 | Ramesh  | NULL   | Jain   | 1962-09-15 | M      | 200000 | 23 Nanakheda, Ujjain     |   5 |      3334 |
| 8886 | Smith   | H      | Borg   | 1937-11-10 | M      |  59000 | 12 Hira Mil, Indore      |   2 |      NULL |
| 9876 | Pradeep | NULL   | Kumar  | 1950-05-08 | M      | 102000 | 2 Azad Nagar, Pune       |   4 |      8886 |
| 9879 | Jabbar  | NULL   | Ahmad  | 1969-03-29 | M      | 205000 | 22 Poonam colony, Indore |   4 |      9876 |
| 9998 | Alicia  | NULL   | Sharma | 1968-01-19 | F      |  25000 | 32 Mangal colony, Pune   |   4 |      9876 |
+------+---------+-------+-------+------------+--------+--------+--------------------------+-----+-----------+
```

Functionalities Used	Purpose
1) Variable (Declare, SET)	Guide us on how to use variables to hold immediate result inside stored procedures.
2) IF statement	The IF-THEN statement allows us to execute a set of SQL statements based on a specified condition.

5) Creating trigger for counting the number of dependents of a particular employee.

```
MariaDB [company]> create table count_dependents (
    -> Employee_id int(6),
    -> Nos_of_dependents int(5) );
Query OK, 0 rows affected (0.040 sec)
```

```
MariaDB [company]> delimiter //
MariaDB [company]> create trigger count_dependent after insert ON dependent
    -> for each row
    -> BEGIN
    -> Declare Nos_of_dependents int(6);
    -> select count(*) into Nos_of_dependents from dependent  where Essn = 3334 group by Essn;
    -> insert into count_dependents values (3334,Nos_of_dependents);
    -> END //
Query OK, 0 rows affected (0.016 sec)

MariaDB [company]> delimiter ;
MariaDB [company]> insert into dependent values (3334,'Rameshwar','M','1999-19-09','Son');
Query OK, 1 row affected, 1 warning (0.011 sec)

MariaDB [company]> select * from count_dependents;
+-------------+-------------------+
| Employee_id | Nos_of_dependents |
+-------------+-------------------+
|        3334 |                 5 |
+-------------+-------------------+
1 row in set (0.000 sec)
```

Functionalities Used	Purpose
1) Variable (Declare, SET)	Guide us on how to use variables to hold immediate result inside stored procedures.

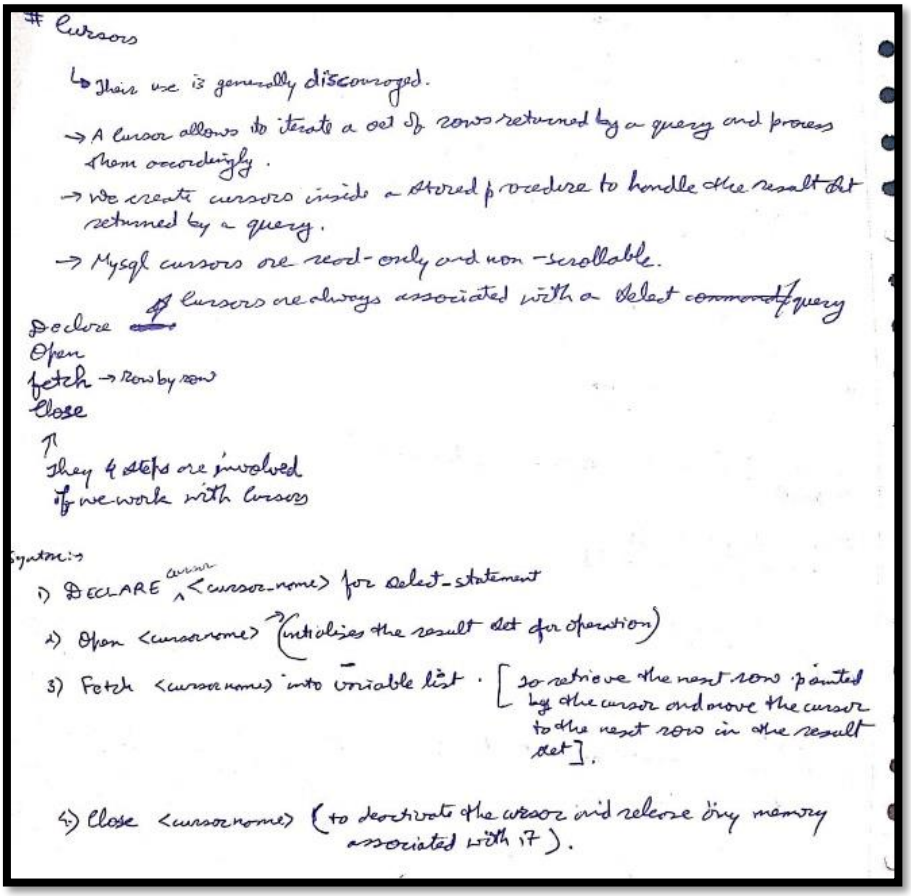
❖ Conclusion:

1) Cursors are particularly useful in stored procedures. They allow you to use only one query to accomplish a task that would otherwise require several queries. However, all cursor operations must execute within a single procedure. A stored procedure cannot open, fetch, or close a cursor that was not declared in the procedure. Cursors are undefined outside the scope of the stored procedure.

2) Stored procedures provide improved performance because fewer calls need to be sent to the database. For example, if a stored procedure has four SQL statements in the code, then there only needs to be a single call to the database instead of four calls for each individual SQL statement. Of course there is always a trade-off. There is an increased workload on the server side that needs to be taken into account.

3) Triggers help the database designer ensure certain actions, such as maintaining an audit file, are completed regardless of which program or user makes changes to the data.

❖ Paper-Work:



TRIGGERS

They are associated with an event and when the event occurs (associated with the table) they are automatically called

They are special types of stored procedures.

They are used for → Data Integrity
to audit the changes in table data

We can't have select statement inside trigger.

Syntax

CREATE trigger <triggername> <triggertime> <triggerevent>
ON <tablename>
BEGIN
END;

→ drop trigger <triggername>

<triggerevent> → insert, update, delete
<triggertime> → before and after

INSERT
(NEW)

UPDATE
(OLD, NEW)

delete
(OLD)

PL/SQL :→ Procedural language extension of SQL

PL/SQL units are stored permanently in the database for reuse

It includes
- functions
- stored procedures
- triggers
- cursors
- loops
- logical statements etc.

We can use SQL statements with PL/SQL

PL/SQL is not case sensitive.

PL/SQL variables :→

→ Variables are containers to store our data.
We need to define their datatype which defines the size and layout of the variable's memory.
→ Variables first needs to be declared before use.
→ They aren't case sensitive and we can't use a keyword as variable.

→ Global V	→ Local V.
which can be used anywhere in the PL/SQL code	which can be used only in a specific unit of PL/SQL code where its declared.

Syntax :→

DECLARE <variablename> <datatype> (size) [DEFAULT <default_value>];

→ Variable should not exceed 30 characters?
→ Default value of a variable is Null.