**Lab Cycle 3**

**Date : 26/06/2023**

**Experiment No: 5**

**Familiarisation of Stored Procedure, Function, Cursor and Triggers**

* A procedure (often called a stored procedure) is a **collection of pre-compiled SQL statements** stored inside the database.

Syntax: CREATE PROCEDURE procedure\_name(parameter\_list)

BEGIN

statements;

END //

* A stored function is a special kind stored program that returns a single

value.

Syntax: DELIMITER $$

CREATE FUNCTION function\_name(param1,param2,...)

RETURNS datatype

[NOT] DETERMINISTIC

BEGIN

-- statements

END $$

DELIMITER ;

* A cursor in database is a construct which allows you to iterate/traversal the records of a table. In MySQL you can use cursors with in a stored program such as procedures, functions etc.

Syntax: DECLARE cursor\_name CURSOR FOR select\_statement;

* A trigger in MySQL is a set of SQL statements that reside in a system catalog. **It is a special type of stored procedure that is invoked automatically in response to an event.**

Syntax: CREATE TRIGGER trigger\_name

{BEFORE | AFTER} {INSERT | UPDATE| DELETE }

ON table\_name FOR EACH ROW

trigger\_body;

1. Create a procedure which will receive account\_id and amount to withdraw. If the account does not exist, it will display a message. Otherwise, if the account exists, it will allow the withdrawal only if the new balance after the withdrawal is at least 1000.

**SQL:**

create table account(account\_id int,balance int);

insert into account values(1000,10000);

insert into account values(1001,5000);

insert into account values(1002,6000);

DELIMITER //

CREATE PROCEDURE withdraw\_amount(IN id INT, IN amount DECIMAL(10,2))

BEGIN

    DECLARE current\_balance DECIMAL(10,2);

    SELECT balance INTO current\_balance FROM account WHERE account\_id = id;

    IF current\_balance IS NULL THEN

        SELECT 'Account does not exist.';

    ELSE

        IF (current\_balance - amount) >= 1000 THEN

            UPDATE account SET balance = balance - amount WHERE id = account\_id;

            SELECT CONCAT('Withdrawal successful. New balance: ', balance) AS message FROM account WHERE account\_id = id;

        ELSE

            SELECT 'Insufficient balance. Minimum balance should be at least 1000.' AS message;

        END IF;

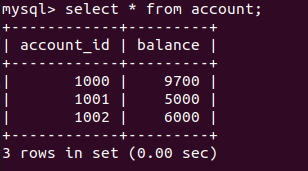
    END IF;

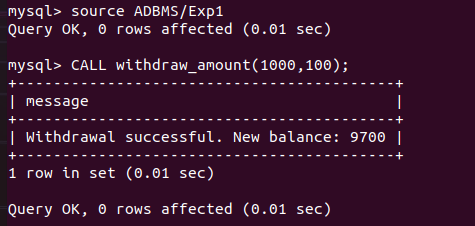
END //

DELIMITER ;

Source ADBMS/Exp1

**Output:**

****

****

2. Create a ‘Customer’ table with attributes customer id, name, city and credits. Write a stored procedure to display the details of a particular customer from the customer table, where name is passed as a parameter.

**SQL:**

create table customer (customer\_id int,name varchar(25),credits int,city varchar(20));

insert into customer values(100,"John",5500,’Kottayam’);

insert into customer values(101,"Abraham",4000,’Kottayam’);

insert into customer values(102,"George",400,’Kottayam’);

DELIMITER //

CREATE PROCEDURE Display\_Customers(IN Cust\_name VARCHAR(25))

BEGIN

    SELECT \* FROM customer WHERE name = Cust\_name;

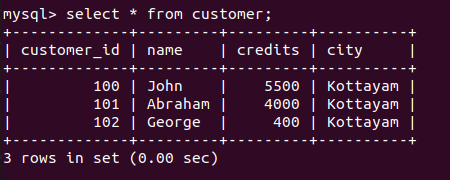
END //

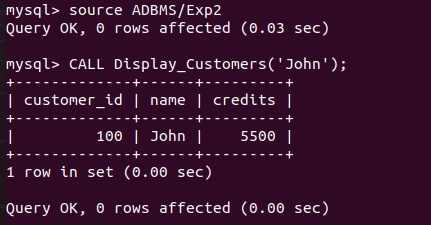
DELIMITER ;

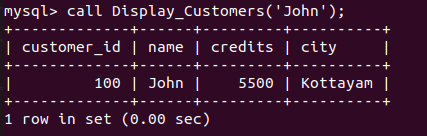
Source ADBMS/Exp2

Call Display\_Customers(‘John’)

**Output:**

****

****

****

3. Create a stored procedure to determine membership of a particular customer based on the following credits:

Above 5000 = Membership Platinum

1000 to 5000 = Gold

< 1000 = silver

[Use IN and OUT Parameters]

**SQL:** DELIMITER //

CREATE procedure Membership(IN id INT,OUT membershiplevel varchar(50))

BEGIN

DECLARE customercredits INT;

select credits INTO customercredits from customer where customer\_id = id;

IF customercredits > 5000 THEN

        SET membershiplevel='Membership Platinum';

ELSEIF customercredits BETWEEN 1000 AND 5000 THEN

        SET membershiplevel='Gold';

ELSE

        SET membershiplevel='Silver';

END IF;

END//

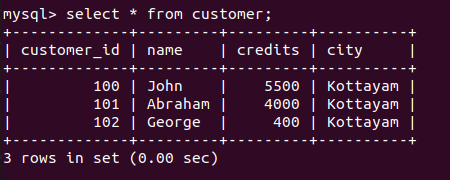
DELIMITER ;

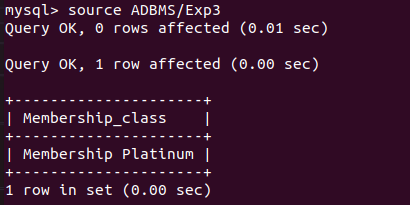
Source ADBMS/Exp3

CALL Membership(100,@membershiplevel);

select @membershiplevel as Membership\_class;

**Output:**

****

****

4. Write a function that takes employee name as parameter and returns the number of employees with this name. Use the function to update details of employees with unique names. For other cases, the program (not the function) should display error messages - “No Employee” or “Multiple employees”.

**SQL:**

CREATE TABLE Employees (

EmployeeID INT PRIMARY KEY,

Name VARCHAR(100)

);

INSERT INTO Employees (EmployeeID, Name)

VALUES

(1, 'John Smith'),

(2, 'Jane Doe'),

(3, 'Michael Johnson');

DELIMITER //

CREATE FUNCTION GetEmployeeCountByName(employeeName VARCHAR(100))

RETURNS INT

DETERMINISTIC

BEGIN

DECLARE empCount INT;

SELECT COUNT(\*) INTO empCount FROM Employees WHERE ename = employeeName;

RETURN empCount;

END //

DELIMITER ;

DELIMITER //

CREATE PROCEDURE UpdateEmployeeDetails(IN employeeName VARCHAR(100))

BEGIN

DECLARE empCount INT;

SET empCount = (SELECT GetEmployeeCountByName(employeeName));

IF empCount = 1 THEN

UPDATE Employees

SET age = 24

WHERE ename = employeeName;

SELECT 'Employee details updated successfully.' AS message;

ELSEIF empCount = 0 THEN

SELECT 'No Employee with the given name.' AS message;

ELSE

SELECT 'Multiple employees with the given name.' AS message;

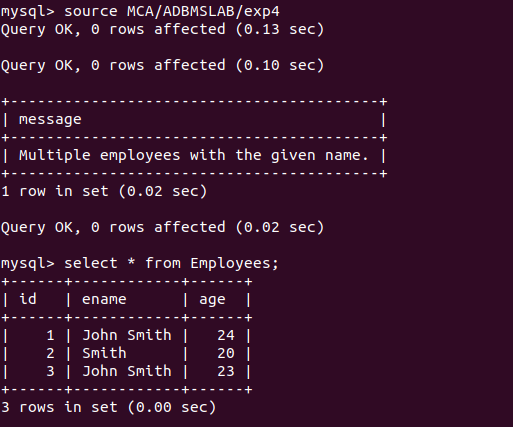
END IF;

END //

DELIMITER ;

Source ADBMS/Exp4

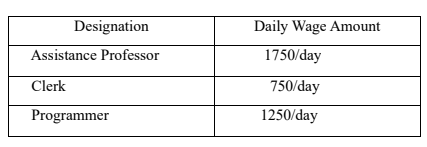
CALL UpdateEmployeeDetails('John Smith');

**Output: **

5. Write a stored procedure using cursor to calculate salary of each employee. Consider an

Emp\_salary table have the following attributes emp\_id, emp\_name, no\_of\_working\_days,

designation and salary.



**SQL:**

create table Emp\_salary(emp\_id int, emp\_name varchar(100), no\_of\_working\_days int, designation varchar(100), salary Decimal(10,2));

insert into Emp\_salary values(1,'Samuel',30,'Assistance Professor',0.0);

insert into Emp\_salary values(2,'John',30,'Clerk',0.0);

insert into Emp\_salary values(3,'Ram',30,'Programmer',0.0);

DELIMITER //

CREATE PROCEDURE CalculateEmployeeSalary()

BEGIN

    DECLARE done INT DEFAULT FALSE;

    DECLARE empId INT;

    DECLARE workingDays INT;

    DECLARE position VARCHAR(100);

    DECLARE sal DECIMAL(10, 2);

    DECLARE employeeCursor CURSOR FOR

        SELECT emp\_id,no\_of\_working\_days,designation,salary FROM Emp\_salary;

    DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;

    OPEN employeeCursor;

    read\_loop: LOOP

        FETCH employeeCursor INTO empId,workingDays,position,sal;

        IF done THEN

            LEAVE read\_loop;

        END IF;

         IF position = 'Assistance Professor' THEN

            SET sal = workingDays \* 1750;

        ELSEIF position = 'Clerk' THEN

            SET sal = workingDays \* 750;

        ELSE

            SET sal = workingDays \* 1250;

        END IF;

        UPDATE Emp\_salary SET salary = sal WHERE emp\_id = empId;

    END LOOP;

    CLOSE employeeCursor;

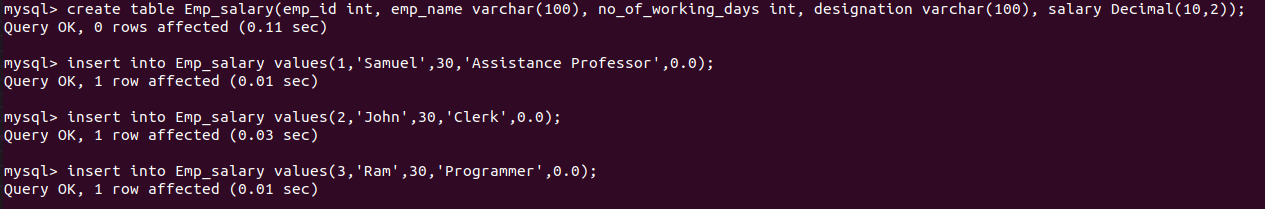
END //

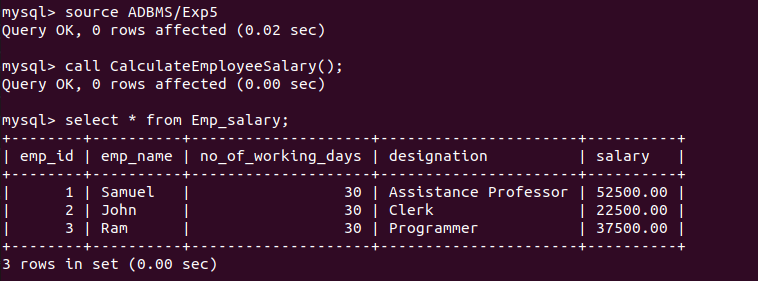
DELIMITER ;

Source ADBMS/Exp5

call CalculateEmployeeSalary();

**Output:**

****

****

6. Write a procedure to calculate the electricity bill of all customers. Electricity board charges the following rates to domestic uses to find the consumption of energy.

a) For first 100 units Rs:2 per unit.

b) 101 to 200 units Rs:2.5 per unit.

c) 201 to 300 units Rs: 3 per unit.

d) Above 300 units Rs: 4 per unit

Consider the table ‘Bill’ with fields customer\_id, name, pre\_reading, cur\_reading , unit,

and amount.

**SQL:**

create table Bill( customer\_id INT,name VARCHAR(255),pre\_reading INT,cur\_reading INT,unit int,amount int);

DELIMITER //

CREATE PROCEDURE CalculateBill()

BEGIN

    DECLARE done INT DEFAULT FALSE;

    DECLARE cust\_id INT;

    DECLARE cust\_name VARCHAR(255);

    DECLARE pre\_reading\_val INT;

    DECLARE cur\_reading\_val INT;

    DECLARE units\_consumed INT;

    DECLARE bill\_amount DECIMAL(10, 2);

    DECLARE cur CURSOR FOR SELECT customer\_id, name, pre\_reading, cur\_reading FROM Bill;

    DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;

    OPEN cur;

    read\_loop: LOOP

        FETCH cur INTO cust\_id, cust\_name, pre\_reading\_val, cur\_reading\_val;

        IF done THEN

            LEAVE read\_loop;

        END IF;

        SET units\_consumed = cur\_reading\_val - pre\_reading\_val;

        IF units\_consumed <= 100 THEN

            SET bill\_amount = units\_consumed \* 2;

        ELSEIF units\_consumed <= 200 THEN

            SET bill\_amount = units\_consumed \* 2.5;

        ELSEIF units\_consumed <= 300 THEN

            SET bill\_amount = units\_consumed \* 3;

        ELSE

            SET bill\_amount = units\_consumed \* 4;

        END IF;

        UPDATE Bill

        SET unit = units\_consumed, amount = bill\_amount

        WHERE customer\_id = cust\_id;

    END LOOP;

    CLOSE cur;

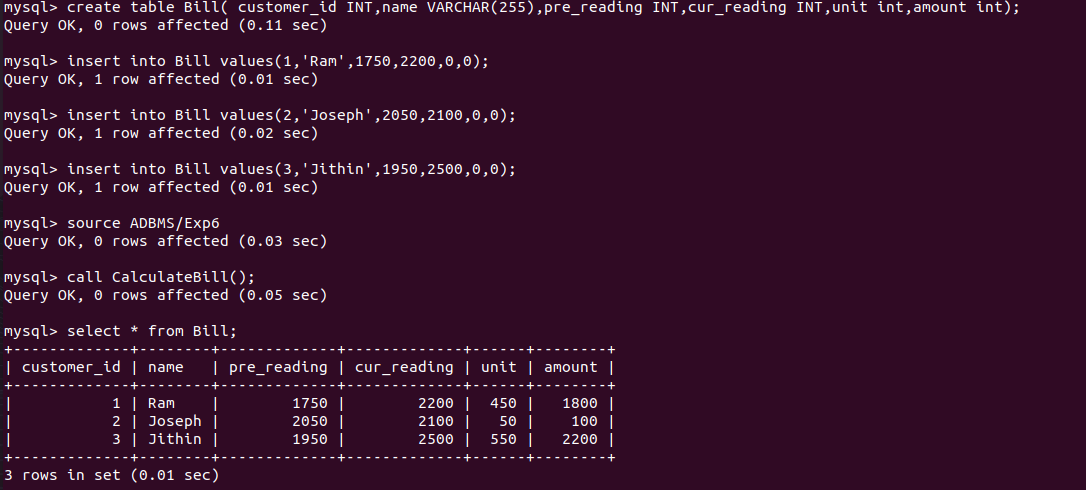
END //

DELIMITER ;

Source ADBMS/Exp6

call CalculateBill();

**Output:**

****

7. Create a trigger on employee table such that whenever a row is deleted, it is moved to history table named ‘Emp\_history’ with the same structure as employee table. ‘Emp\_history’ will contain an additional column “Date\_of\_deletion” to store the date on which the row is removed.

[ After Delete Trigger]

**SQL:** Create table emp\_history (employee\_id int,employee\_name varchar(50),employee\_department varchar(50),date\_of\_deletion date);

Create table employee (   employee\_id int, employee\_name varchar(50), employee\_department varchar(50));

Insert into employee (employee\_id, employee\_name, employee\_department) values(1, 'john doe', 'sales'),(2, 'jane smith', 'marketing'),(3, 'robert johnson', 'finance');

DELIMITER //

CREATE TRIGGER trg\_employee\_delete

AFTER DELETE ON employee

FOR EACH ROW

BEGIN

  INSERT INTO Emp\_history (employee\_id, employee\_name, employee\_department, Date\_of\_deletion)

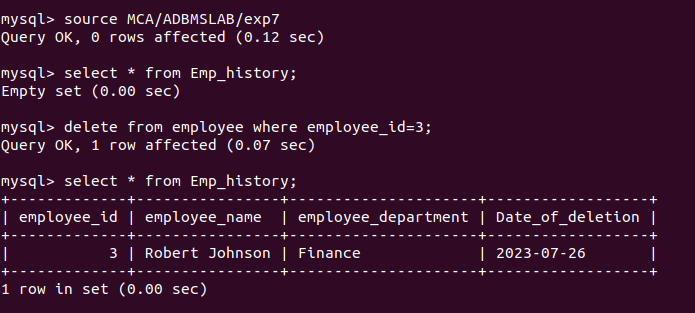
  VALUES (OLD.employee\_id, OLD.employee\_name, OLD.employee\_department, CURDATE());

END //

DELIMITER ;

Source ADBMS/Exp7

Delete from employee where employee\_id=1;

**Output: **

8. Before insert a new record in emp\_details table, create a trigger that check the column value of FIRST\_NAME, LAST\_NAME, JOB\_ID and if there are any space(s) before or after the FIRST\_NAME, LAST\_NAME, TRIM () function will remove those. The value of the JOB\_ID will be converted to upper cases by UPPER () function. [Before Insert Trigger]

**SQL:** Create table emp\_details (id int primary key auto\_increment,first\_name varchar(50),last\_name varchar(50),job\_id varchar(50));

DELIMITER $$

CREATE TRIGGER before\_insert\_emp\_details

BEFORE INSERT ON emp\_details

FOR EACH ROW

BEGIN

    SET NEW.FIRST\_NAME = TRIM(NEW.FIRST\_NAME);

    SET NEW.LAST\_NAME = TRIM(NEW.LAST\_NAME);

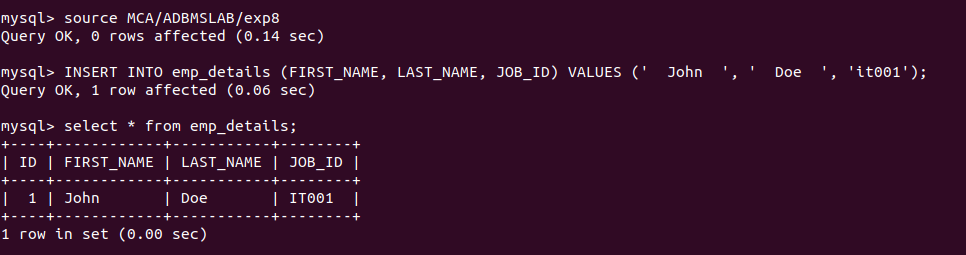
    SET NEW.JOB\_ID = UPPER(NEW.JOB\_ID);

END$$

DELIMITER ;

Source ADBMS/Exp8

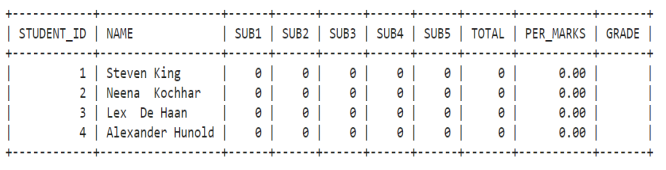
Insert into emp\_details (first\_name, last\_name, job\_id) values ('  john  ', '  doe  ', 'it001');

**Output: **

9. Consider the following table with sample data. Create a trigger to calculate total marks,

percentage and grade of the students, when marks of the subjects are updated. [After Update

Trigger]



For this sample calculation, the following conditions are assumed:

Total Marks (will be stored in TOTAL column) : TOTAL = SUB1 + SUB2 + SUB3 + SUB4 +

SUB5.

Percentage of Marks (will be stored in PER\_MARKS column): PER\_MARKS = (TOTAL)/5

Grade (will be stored in GRADE column):

- If PER\_MARKS>=90 -> 'EXCELLENT'

- If PER\_MARKS>=75 AND PER\_MARKS<90 -> 'VERY GOOD'

- If PER\_MARKS>=60 AND PER\_MARKS<75 -> 'GOOD'

- If PER\_MARKS>=40 AND PER\_MARKS<60 -> 'AVERAGE'

- If PER\_MARKS<40-> 'NOT PROMOTED'

**SQL:** CREATE TABLE students (student\_id INT,name VARCHAR(50),sub1 INT,sub2 INT,sub3 INT,sub4 INT,sub5 INT,total INT,per\_marks DECIMAL(5,2),grade VARCHAR(20));

DELIMITER //

CREATE TRIGGER calculate\_marks

BEFORE UPDATE ON students

FOR EACH ROW

BEGIN

  SET NEW.total = NEW.sub1 + NEW.sub2 + NEW.sub3 + NEW.sub4 + NEW.sub5;

  SET NEW.per\_marks = NEW.total / 5;

  IF NEW.per\_marks >= 90 THEN

    SET NEW.grade = 'EXCELLENT';

  ELSEIF NEW.per\_marks >= 75 AND NEW.per\_marks < 90 THEN

    SET NEW.grade = 'VERY GOOD';

  ELSEIF NEW.per\_marks >= 60 AND NEW.per\_marks < 75 THEN

    SET NEW.grade = 'GOOD';

  ELSEIF NEW.per\_marks >= 40 AND NEW.per\_marks < 60 THEN

    SET NEW.grade = 'AVERAGE';

  ELSE

    SET NEW.grade = 'NOT PROMOTED';

  END IF;

END //

DELIMITER ;

Source ADBMS/Exp9

INSERT INTO students (student\_id, name, sub1, sub2, sub3, sub4, sub5) VALUES (1, 'John Doe', 85, 90, 77, 92, 88);

update students set sub1=86;

**Output:** 