**STORED PROCEDURES and USER-DEFINED FUNCTIONS:**

***Objective :***

To encapsulate business logic using stored procedures and functions.

***Stored Procedure :***

A Stored Procedure is a precompiled collection of one or more SQL statements that can be executed as a unit.

***Queries :***

**delimiter $$**

**create procedure salary\_based\_display (input\_salary int, input\_department varchar(30))**

**begin**

**select \* from employees where salary >= input\_salary and department=input\_department;**

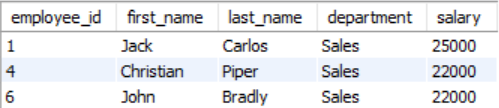
**end $$**

**delimiter ;**

// This code creates a stored procedure salary\_based\_display that takes two input parameters: input\_salary (an integer) and input\_department (a string). The procedure retrieves all records from the employees table where the salary is greater than or equal to input\_salary and the department matches the specified input\_department.

**call salary\_based\_display(20000,"Sales");**

// The above statement invokes the stored procedure salary\_based\_display, passing 20000 and "Sales" as the value for the input\_salary parameter.



**delimiter $$**

**create procedure Get\_Employees\_Aggregates (out employee\_count int,**

**out average\_salary decimal(12,2),**

**out salary\_sum int)**

**begin**

-- Calculate total number of employees

**select count(\*) into employee\_count from employees;**

-- Calculate average salary of employees

**select avg(salary) into average\_salary from employees;**

-- Calculate total salary of all employees

**select sum(salary) into salary\_sum from employees;**

**end $$**

**delimiter ;**

// The provided code calculates and returns three aggregate values related to employees: the total number of employees (*employee\_count*), the average salary of employees (*average\_salary*), and the total salary of all employees (*salary\_sum*). The procedure performs these calculations using SQL aggregate functions on the employees table and stores the results in the output parameters. The procedure does not return a result set but instead outputs the values through the defined parameters.

**call Get\_Employees\_Aggregates(@employee\_count, @average\_salary, @salary\_sum);**

**select @employee\_count, @average\_salary, @salary\_sum;**

// Calling the stored procedure, storing outputs and retrieving the outputs



***User Defined Functions :***

A User Defined Function (UDF) is a custom function created by the user to extend the functionality of MySQL by recalling the custom functions where needed.

**delimiter $$**

**create function calculate\_discount(total\_amount decimal(10,2))**

**returns decimal(10,2)**

**deterministic**

**begin**

**declare discount decimal(10,2);**

**if total\_amount >= 1000 then**

**set discount = total\_amount \* 0.10;** -- 10% discount for amounts over 1000

**elseif total\_amount >= 500 then**

**set discount = total\_amount \* 0.05;** -- 5% discount for amounts over 500

**else**

**set discount = 0;** -- No discount for amounts below 500

**end if;**

**return discount;**

**end $$**

**delimiter ;**

// The provided SQL code creates a scalar user-defined function that takes a total\_amount as input and returns a discount based on the value of total\_amount. If the total amount is greater than or equal to 1000, the function applies a 10% discount. For amounts between 500 and 999, it applies a 5% discount. If the total amount is below 500, no discount is applied. The function uses conditional logic to determine the appropriate discount and then returns the calculated discount value.

**select calculate\_discount(1500) as discount\_amount;**

****

**select calculate\_discount(600) as discount\_amount;**

****

**select calculate\_discount(300) as discount\_amount;**



**Note :** In MySQL, a user-defined function (UDF) cannot return multiple result sets directly. We can use stored procedures for returning multiple values. To return multiple values, you can use a stored procedure with either out parameters or a temporary table to simulate table-valued results.