DATABASE MANAGEMENT SYSTEM

NOTES

RAJENDRAN S

DATA DEFINITION LANGUAGE

- Click here to view the notes for DDL

This topic contains the following sub-topics:

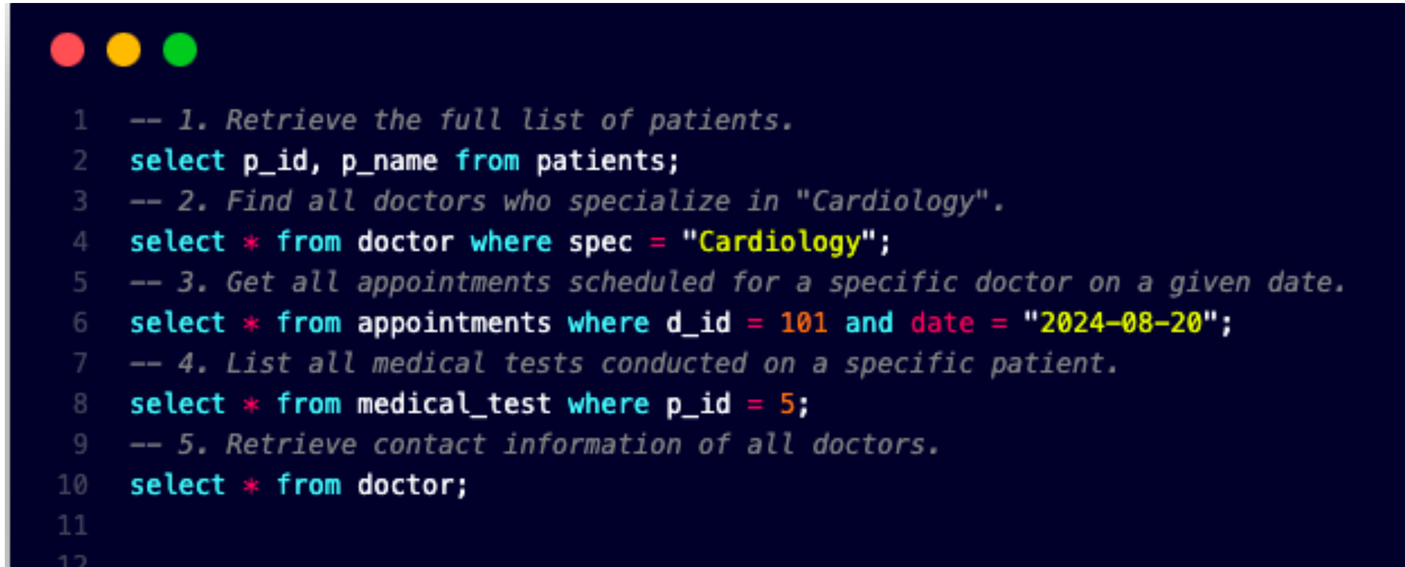
A. Creating a Table
B. Inserting Values into a Table
C. Updating the Values in a table
D. Deleting a Table
E. Alter Operations

DATA MANIPULATION LANGUAGE (DML)

SELECT

□□□SELECT Queries:

□□ Retrieve the full list of patients.
□□ Find all doctors who specialize in 'Cardiology'.
□□ Get all appointments scheduled for a specific doctor on a given date.
□□ List all medical tests conducted on a specific patient.
□□ Retrieve contact information of all doctors.



```sql
-- 1. Retrieve the full list of patients.
select p_id, p_name from patients;
-- 2. Find all doctors who specialize in "Cardiology".
select * from doctor where spec = "Cardiology";
-- 3. Get all appointments scheduled for a specific doctor on a given date.
select * from appointments where d_id = 101 and date = "2024-08-20";
-- 4. List all medical tests conducted on a specific patient.
select * from medical_test where p_id = 5;
-- 5. Retrieve contact information of all doctors.
select * from doctor;
```

DATA MANIPULATION LANGUAGE (DML)

INSERT

□□□INSERT Queries:

- □□ Add a new patient to the `Patients` table.
- □□ Insert a new appointment for a patient with a specific doctor on a given date and time.
- □□ Record a new medical test for a patient.

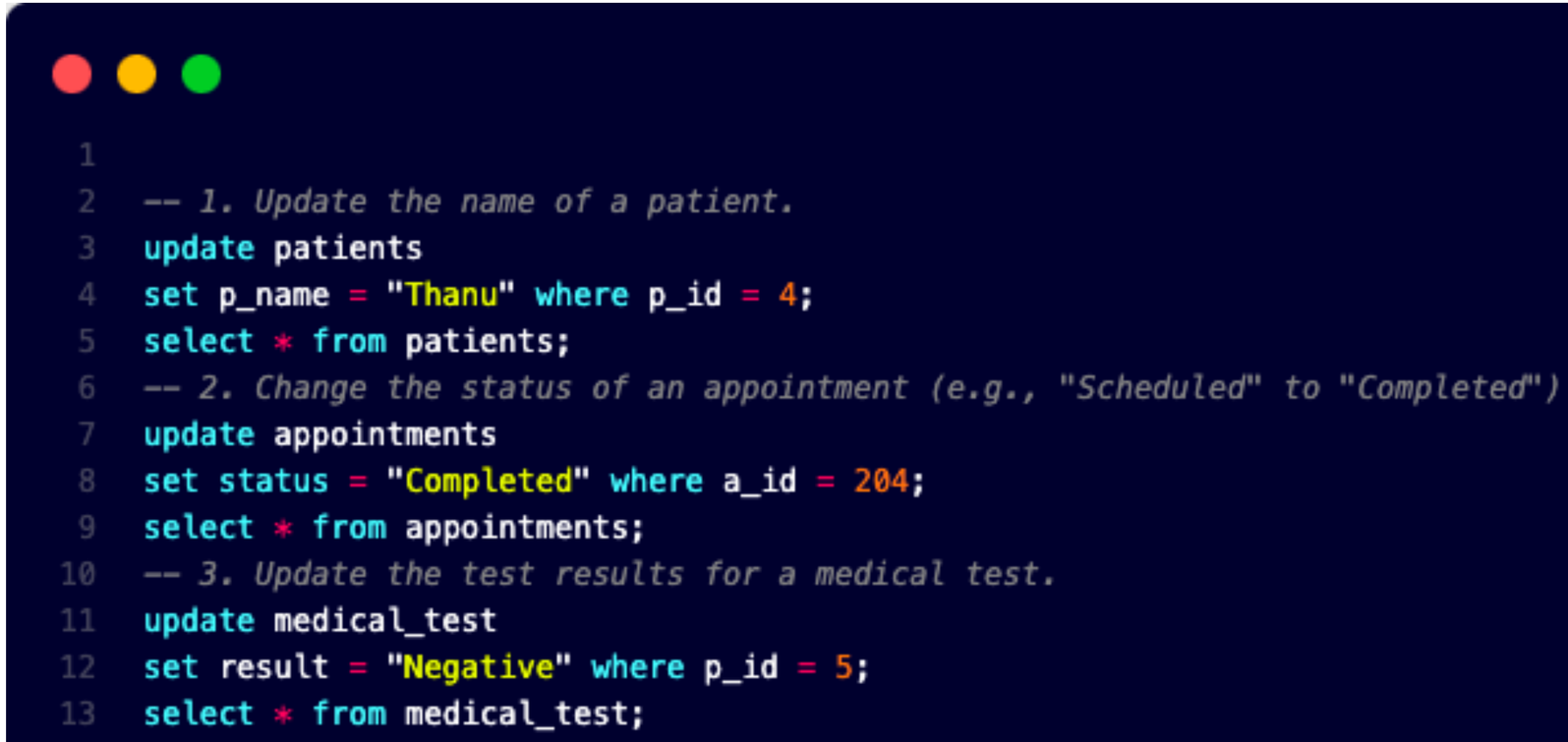
```sql
-- 1. Add a new patient to the Patients table.
insert into patients values(13,103,303,"Mike Johnson");
select * from patients;
-- 2. Insert a new appointment for a patient with a specific doctor on a given date and time.
insert into appointments values(304,5,102,"2024-07-23","15:00","Cancelled");
select * from appointments;
-- 3. Record a new medical test for a patient.
insert into medical_test values(14,"Positive");
select * from medical_test;
```

DATA MANIPULATION LANGUAGE (DML)

UPDATE

□□□UPDATE Queries:

□□ Update the contact number of a patient.
□□ Change the status of an appointment (e.g., from 'Scheduled' to 'Completed').
□□ Update the test results for a medical test.



```
1
2  -- 1. Update the name of a patient.
3  update patients
4  set p_name = "Thanu" where p_id = 4;
5  select * from patients;
6  -- 2. Change the status of an appointment (e.g., "Scheduled" to "Completed")
7  update appointments
8  set status = "Completed" where a_id = 204;
9  select * from appointments;
10 -- 3. Update the test results for a medical test.
11 update medical_test
12 set result = "Negative" where p_id = 5;
13 select * from medical_test;
```

DATA MANIPULATION LANGUAGE (DML)

DELETE

□□□DELETE Queries:

□□ Remove a patient record from the database.
□□ Delete an appointment for a specific patient.

```sql
-- 1. Remove a patient record from the database.
delete from patients where p_id = 1;
select * from patients;
-- 2. Delete an appointment for a specific patient.
delete from appointments where p_id = 1;
select * from appointments;
```

AGGREGATE FUNCTIONS

Sales Database: Monthly Sales Analysis

You are analysing a sales database to understand monthly sales performance.

Question 1: Write a SQL query to find the total sales amount for each month in the current year.

Question 2: Calculate the average sales amount per transaction for the last quarter.

Question 3: Identify the month with the highest sales and the corresponding sales amount.

```sql
-- Sales Database: Monthly Sales Analysis
-- You are analysing a sales database to understand monthly sales performance.
-- Question 1: Write a SQL query to find the total sales amount for each month in the current year
select MONTH(sale_date),sum(sale_amount) from sales where year(sale_date) = year(curdate()) group by MONTH(sale_date) ;
-- Question 2: Calculate the average sales amount per transaction for the last quarter.
select transaction_id, avg(transaction_amount) from banks where MONTH(curdate()) - MONTH(transaction_date) <= 3 and MONTH(curdate()) - MONTH(transaction_date) >=0  group by transaction_id ;
-- Question 3: Identify the month with the highest sales and the corresponding sales amount.
select month(sale_date), sum(sale_amount) as total_sales from sales group by month(sale_date) order by total_sales desc limit 1;

```

Cont.

Banking Database: Customer Transactions

You are analysing a banking database to understand customer transactions.

Question 4: Write a SQL query to find the total amount of deposits made by each customer in the last six months.

Question 5: Calculate the average transaction amount for withdrawals in the current month.

Question 6: Determine the customer with the highest total transaction amount in the last year.

```sql
-- Banking Database: Customer Transactions
-- You are analysing a banking database to understand customer transactions.
-- Question 4: Write a SQL query to find the total amount of deposits made by each customer in the
-- last six months.
select * from banks;
select customer_id, sum(transaction_amount) as total from banks
where transaction_type = "deposit"
AND TIMESTAMPDIFF(MONTH, transaction_date, CURDATE()) <= 6
AND TIMESTAMPDIFF(MONTH, transaction_date, CURDATE()) >= 0
group by customer_id;
-- Question 5: Calculate the average transaction amount for withdrawals in the current month.
select avg(transaction_amount) from banks
 where transaction_type = "withdrawal" and month(transaction_date) = month(curdate());
-- Question 6: Determine the customer with the highest total transaction amount in the last year.
select customer_id, sum(transaction_amount) as total from banks
 group by customer_id order by total desc limit 1;
```

Cont.

Library Database: Book Loans

You are working with a library database to analyse book loan records.

Question 7: Write a SQL query to find the total number of books loaned out in the last year.

Question 8: Calculate the average number of books loaned out per member in the last month.

Question 9: Determine the book with the highest number of loans in the current semester.

```sql
-- Library Database: Book Loans
-- You are working with a library database to analyse book loan records.
-- Question 7: Write a SQL query to find the total number of books loaned out in the last year.
select * from library;
select count(loan_id) as total from library where timestampdiff(year, loan_date, curdate()) <= 1;
-- Question 8: Calculate the average number of books loaned out per member in the last month.
select avg(total) from( select count(book_id) as total from library where timestampdiff(month, loan_date, curdate()) <= 1 group by member_id) as subquery;
-- Question 9: Determine the book with the highest number of loans in the current semester.
select book_id, count(*) as total from library where timestampdiff(month, loan_date, curdate())<=3 group by book_id order by total desc limit 1;
```

Cont.

Human Resources Database: Employee Salaries

You are analysing an HR database to understand employee salaries.

Question 10: Write a SQL query to find the total salary paid to employees in each department.

Question 11: Calculate the average salary of employees in the "IT" department.

Question 12: Identify the department with the highest average salary.

```sql
-- Human Resources Database: Employee Salaries
-- You are analysing an HR database to understand employee salaries.
select * from humanresource;
-- Question 10: Write a SQL query to find the total salary paid to employees in each department.
select department_name, sum(Salary) from humanresource group by department_name;
-- Question 11: Calculate the average salary of employees in the &quot;IT&quot; department.
select avg(salary)  as averageSalary from humanresource
 where department_name = "IT" group by department_name;
-- Question 12: Identify the department with the highest average salary.
select department_name, sum(Salary)  as total from humanresource
 group by department_name order by total desc limit 1;
```

Cont.

Retail Database: Product Inventory

You are managing a retail database and need to analyse product inventory.

Question 13: Write a SQL query to find the total number of products in each category.

Question 14: Calculate the average price of products in the "Electronics" category.

Question 15: Identify the category with the highest total inventory value.

```sql
-- Retail Database: Product Inventory
select * from retail;
-- You are managing a retail database and need to analyse product inventory.
-- Question 13: Write a SQL query to find the total number of products in each category.
select product_id, count(*) from retail group by product_id;
-- Question 14: Calculate the average price of products in the &quot;Electronics&quot; category.
select avg(price) from retail where category_name = "Electronics" group by category_id;
-- Question 15: Identify the category with the highest total inventory value.
select category_name, SUM(price*quantity_in_stock) as total from retail
group by category_name order by total desc limit 1;

```

SUB QUERIES

```sql
-- Write a query to find all employees whose salary is above the average salary of the company.
select * from employee;
select employee_name, salary from employee
where salary > (select avg(salary) from employee);
-- Find customers who have placed orders with a total amount greater than the average order amount.
select * from orderss;
select customer_id, order_amount from orderss
where order_amount > (select avg(order_amount) from orderss);
-- Identify employees who do not manage any departments.
select employee_name from employee
where employee_name not in (select manager_id from department);
-- List all products that have not been sold yet.
select * from product;
select * from orderss;
select product_name from product
where product_id not in (select order_id from orderss);
-- Find the department with the highest average employee salary.
SELECT department_id, average
FROM (
    SELECT department_id, AVG(salary) AS average
    FROM employee
    GROUP BY department_id
) AS depavg
ORDER BY average DESC
LIMIT 1;
```

JOINS

```sql
-- Joins
-- Join the "Orders" and "Customers" tables to retrieve the order details along with customer information.
 select
    c.customer_id,
    c.customer_name,
    o.order_id,
    o.order_amount,
    c.email
FROM customer c
JOIN orderss o
ON c.customer_id = o.customer_id;


-- Combine the "Employees" and "Departments" tables to display employee details along with their respective department names.

 select
    e.employee_id,
    e.employee_name,
    d.department_name,
    e.salary
FROM employee e
JOIN department d
ON e.department_id = d.department_id;

-- Join the "Products" and "Sales" tables to get a list of products along with their sales information.
select
    p.product_id,
    p.product_name,
    s.quantity_sold,
    s.sale_amount,
    s.sale_Date
from product p
JOIN sales s
on p.product_id = s.product_id;
```

```sql
-- Retrieve all orders placed by customers with order amount greater
--  than 100 by joining the "Orders" and "Customers" tables.
select
    c.customer_id,
    c.customer_name,
    o.order_amount
FROM customer c
JOIN orderss o
ON c.customer_id = o.customer_id
WHERE o.order_amount >100;
-- Join the "Employees" and "Managers" tables to find out the managers of each department.
select
    d.department_id,
    d.department_name,
    e.employee_name
FROM department d
JOIN employee e
ON d.manager_id = e.employee_id;
```

SUBQUERIES & JOINS

- List all departments along with the count of employees in each department.
- Retrieve the details of employees who earn more than the average salary in their respective departments.

```sql
-- List all departments along with the count of employees in each department.
select
  d.department_name,
    IFNULL(c.total, 0) AS total
FROM department d
LEFT JOIN
  (select department_id, count(*) as total from employee group by department_id) c
ON d.department_id = c.department_id;

-- Retrieve the details of employees who earn more than the average salary in their respective departments.
select
  e.employee_id,
    e.employee_name,
    d.department_name
FROM employee e
JOIN department d
ON e.department_id = d.department_id
WHERE
  e.salary > (select avg(salary) from employee e2 where e2.department_id = e.department_id);


```

GROUP BY CLAUSE

```sql
-- Write a query to find the total sales amount for each product category from the "Sales" table.
select product_id,  sum(Sale_amount) from sales group by product_id;
-- Calculate the average salary of employees in each department from the "Employees" table
select department_id, avg(salary) from employee group by department_id;
-- Extract the year from the order date and find the total sales amount for each year from the "Sales" table.
select year(sale_date), sum(sale_amount) from sales group by year(sale_date);
-- Calculate the number of orders placed by each customer from the "Orders" table.
select customer_id, count(*) from orderss group by customer_id;
-- Find the average salary of employees in each department for the year 2024 from the "Employees" table.
select department_id, avg(salary) from employee group by department_id;
-- 1.Find the top 5 highest-selling products along with their categories for each year from the "Sales" table.
select product_id, sum(quantity_sold) as total from sales group by product_id  order by total desc limit 5;
-- Group the sales data by product category and month-year combination
-- and calculate the total sales amount for each category-month pair from the "Sales" table.


```

MISCELLANEOUS

- DISTINCT
- ORDER BY
- WILDCARD CHARACTERS
- IN AND NOT IN
- BETWEEN CLAUSE
- ALIASING

```sql
-- DISTINCT: returns only the unique values from the specified column
select distinct product_id from sales;

SELECT COUNT(DISTINCT product_id) AS total
FROM sales;

-- ORDER BY: used to sort the output
select * from sales order by sale_amount desc;

-- WILDCARD CHARACTERS: use d to manipulate the data
select * from sales where sale_amount like "4%";
select * from sales where sale_date like "2024-__-15";
select * from sales where sale_date like "%03%";
-- return all two and more digit sale amounts
select * from sales where sale_amount like "__%";
-- return only two digit sale amounts
select * from sales where sale_amount like "__";

-- Alternative method: Usage of IN
select * from sales where sale_amount in  (40, 20, 30);
-- BETWEEEN clause
select * from sales where sale_amount between 20 and 60;

-- Aliasing
select sale_amount as "Amount sold" from sales;
```

PL/SQL PROCEDURES

DEFINITION & SYNTAX

- A procedure is a named PL/SQL block that performs a specific task. Unlike functions, procedures do not return a value directly.
- Instead, they perform actions such as updating a database, performing computations, or managing transactions.

Key Concepts

1. **Parameters**: Procedures can have parameters to pass values in and out. There are three types:
   - **IN**: Passes a value into the procedure (read-only).
   - **OUT**: Returns a value from the procedure (write-only).
   - **IN OUT**: Passes a value into the procedure and can return a modified value.
2. **Declarative Part**: This section is where you define any variables, constants, cursors, and exceptions that the procedure will use.
3. **Executable Part**: This section contains the PL/SQL statements that execute when the procedure is called. This is where the main logic of the procedure is implemented.
4. **Exception Handling Part**: Handles exceptions (errors) that occur during execution. This part is optional but recommended for robust error handling.

Basic Syntax for Creating a Stored Procedure in MySQL

sql Copy code

```sql
DELIMITER //

CREATE PROCEDURE procedure_name (
    [IN | OUT | IN OUT] parameter_name parameter_type,
    [IN | OUT | IN OUT] parameter_name parameter_type,
    ...
)
BEGIN
    -- Declarative part: Variable declarations (optional)
    DECLARE variable_name variable_type;

    -- Executable part: SQL statements and procedural logic
    statement1;
    statement2;
    ...

    -- Optional exception handling
    -- DECLARE CONTINUE HANDLER FOR SQLSTATE 'error_code'
    -- BEGIN
    --     -- Error handling code
    -- END;
END //

DELIMITER ;
```

Procedures (Cont.)

EXAMPLE

sql Copy code

```sql
DELIMITER //

CREATE PROCEDURE update_salary (
    IN ID INT,
    IN percentage DECIMAL(5,2)
)
BEGIN
    -- Update the quantity by a percentage if the percentage is positive
    IF percentage > 0 THEN
        UPDATE Employee
        SET quantity = quantity + (quantity * (percentage / 100))
        WHERE emp_ID = ID;

        -- Check if the update affected any rows
        IF ROW_COUNT() = 0 THEN
            SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'No employee found with the
        END IF;
    ELSE
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Percentage must be positive';
    END IF;
END //

DELIMITER ;
```

sql Copy code

```sql
CALL update_salary(104, 10);
```

NOTE:

- Incase of safe mode, use the following command

```sql
SET SQL_SAFE_UPDATES = 0;
```

- Incase of replacing a procedure, you have to drop it and redefine it

```sql
DROP PROCEDURE IF EXISTS update_salary;
```

Procedures (Cont.)

Lab Exercises

```sql
-- 1. Write a PL/SQL procedure that accepts an employee ID as input and
  --  updates the salary of that employee by a given percentage increase.
DELIMITER //
create procedure update_salary(IN ID INT, IN percentage DECIMAL(5,2))
begin
  update Employee
    set salary = salary + (salary * (percentage/100))
    where emp_ID = ID;
end  //
DELIMITER ;

CALL update_salary(104, 10);
```

```sql
-- 2. Create a PL/SQL procedure that takes a department ID as input and
  -- returns the average salary of employees in that department.

delimiter //
create procedure departmentAverage( IN ID INT, OUT avg_salary DECIMAL(10,2) )
begin
  select avg(salary) INTO avg_salary from Employee where dept_ID = ID;
end
// delimiter ;

-- Caling a procedure with an OUT variable
set @avg_salary = 0;
call departmentAverage(2, @avg_salary);
select @avg_salary as Average_Salary;
```

```sql
-- 3. Develop a PL/SQL procedure that accepts a product ID and quantity as input parameters.
  -- This procedure should decrease the stock quantity of the product by the given quantity if it's available;
    -- otherwise, it should raise an appropriate exception.


DELIMITER //
create procedure decreaseQuantity(IN ID INT, IN quantity_reduce INT)
begin
  declare current_quantity INT;

    select quantity into current_quantity
    FROM product
  where ID = prdt_ID;

   IF current_quantity < quantity_reduce THEN
    SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT =  'Stock is not sufficient!';
  ELSE
    update Product
    set quantity = quantity - quantity_reduce
    where ID = prdt_ID;
        SELECT 'Quantity Updated' as Message;
    END IF;
end
// DELIMITER ;


CALL decreaseQuantity(51, 3);
```

PL/SQL Functions

- A PL/SQL function is a named PL/SQL block that performs a specific task and returns a value. Functions in PL/SQL are similar to procedures, but a function must return a value, whereas a procedure doesn't have to.

sql Copy code

```sql
DELIMITER $$

CREATE FUNCTION function_name(parameter_name datatype)
RETURNS return_datatype
[DETERMINISTIC | NOT DETERMINISTIC]
BEGIN
    -- Declare variables
    DECLARE variable_name datatype DEFAULT value;

    -- Function logic goes here
    RETURN value;  -- Returns the result
END$$

DELIMITER ;
```

sql Copy code

```sql
DELIMITER $$

CREATE FUNCTION calculate_factorial(n INT)
RETURNS INT
DETERMINISTIC
BEGIN
    DECLARE result INT DEFAULT 1;
    DECLARE i INT DEFAULT 1;

    -- Loop to calculate factorial
    WHILE i <= n DO
        SET result = result * i;
        SET i = i + 1;
    END WHILE;

    -- Return the result
    RETURN result;
END$$

DELIMITER ;
```

sql Copy code

```sql
SELECT calculate_factorial(5);
```

```sql
-- Write a PL/SQL function named calculate_salary that takes in two parameters:
-- hours_worked (number) and hourly_rate (number), and returns the calculated salary (number).

DELIMITER $$
CREATE FUNCTION calculate_salary(hours_worked INT, hourly_rate INT)
RETURNS INT
DETERMINISTIC
BEGIN
    DECLARE Salary INT DEFAULT 0;
    SET Salary = hours_worked * hourly_rate;
    RETURN Salary;
END $$
DELIMITER ;

SELECT calculate_salary(10, 50);
```

```sql
-- Develop a function named get_employee_name that takes an employee ID as input and
-- returns the employee's name. You can use a cursor to fetch the name from the employee table
-- based on the provided ID.

DELIMITER $$

CREATE FUNCTION get_employee_name(employee_ID INT)
RETURNS VARCHAR(100)
DETERMINISTIC
BEGIN
    DECLARE employee_name VARCHAR(100);
    Select emp_name INTO employee_name FROM employee WHERE emp_ID = employee_ID;
    RETURN employee_name;
END $$
DELIMITER ;

SELECT get_employee_name(105);
```

PL/SQL Cursor

- In MySQL, cursors allow you to iterate over a result set row-by-row.
- Cursors are primarily used in stored procedures and functions when you need to process query results sequentially, typically when performing more complex operations on each row.

```
DELIMITER $$

CREATE PROCEDURE fetch_employee_names()
BEGIN
    -- Declare a variable to hold employee names
    DECLARE emp_name VARCHAR(100);

    -- Declare the cursor to select employee names
    DECLARE emp_cursor CURSOR FOR
        SELECT name FROM employees;

    -- Open the cursor
    OPEN emp_cursor;

    -- Fetch the first row
    FETCH emp_cursor INTO emp_name;

    -- Loop until no more rows are found
    WHILE emp_name IS NOT NULL DO
        -- Output the employee name
        SELECT emp_name;

        -- Fetch the next row
        FETCH emp_cursor INTO emp_name;
    END WHILE;

    -- Close the cursor
    CLOSE emp_cursor;
END$$

DELIMITER ;
```

```
DELIMITER $$

CREATE PROCEDURE procedure_name()
BEGIN
    -- Declare variables to store the data from the cursor
    DECLARE variable_name datatype;

    -- Declare the cursor
    DECLARE cursor_name CURSOR FOR
        SELECT column_name FROM table_name;

    -- Open the cursor
    OPEN cursor_name;

    -- Fetch data from the cursor into the variable
    FETCH cursor_name INTO variable_name;

    -- Use a loop to continue fetching rows until there are no more
    WHILE condition DO
        -- Do something with the data (e.g., SELECT it or process it)

        -- Fetch the next row from the cursor
        FETCH cursor_name INTO variable_name;
    END WHILE;

    -- Close the cursor after use
    CLOSE cursor_name;
END$$

DELIMITER ;
```

```sql
-- 1. Create a cursor named EmployeeCursor that fetches the employee
 -- names from the Employees table. Print the names of all the employees
 -- using the cursor.

DELIMITER $$
CREATE PROCEDURE Get_Employee_Names()
BEGIN
    DECLARE e_name VARCHAR(100);
    DECLARE done INT DEFAULT 0;

    DECLARE EmployeeCursor CURSOR FOR
        Select emp_name from employee;

    DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = 1;

    OPEN EmployeeCursor;

    FETCH EmployeeCursor INTO e_name;

    WHILE done = 0 DO
        SELECT e_name;
        FETCH EmployeeCursor INTO e_name;
    END WHILE;

    CLOSE EmployeeCursor;
END $$
DELIMITER ;

CALL Get_Employee_Names();
```

```sql
-- 3. create table for this,
-- Create a cursor named "ProductCursor" that retrieves the ProductID and
-- Quantity from the "Products" table.Loop through the cursor and delete all
 -- products that have q quantity less than 10.

 DELIMITER $$
 CREATE PROCEDURE DeleteProducts()
 BEGIN
  DECLARE prodID INT;
    DECLARE Qnty INT;
    DECLARE done INT DEFAULT 0;

    DECLARE ProductCursor CURSOR FOR
    SELECT ProductID, Quantity FROM Products;

  DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = 1;

    OPEN ProductCursor;

    FETCH ProductCursor INTO prodID, Qnty;

    WHILE done = 0 DO
    IF Qnty < 10 THEN
      DELETE FROM Products WHERE ProductID = ProdID;
    END IF;
        FETCH ProductCursor INTO prodID, Qnty;
  END WHILE;

    CLOSE ProductCursor;
END $$
DELIMITER ;


CALL DeleteProducts();
```

```sql
-- 2.Write a cursor named "OrderCursor" that retrieves the OrderId and
 -- OrderDate from the "Orders" table. Iterate through the cursor and updat
 -- the OrderDate to the current date for each other.



DELIMITER $$
CREATE PROCEDURE Get_Employee_Details()
BEGIN
  DECLARE ID INT;
    DECLARE o_name DATETIME;
    DECLARE done INT DEFAULT 0;

    DECLARE OrderCursor CURSOR FOR
    Select OrderID, OrderDate From Orders;

    DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = 1;

    OPEN OrderCursor;

    FETCH OrderCursor into ID, o_name;

    WHILE done = 0 DO

        UPDATE Orders
        SET OrderDate = now()
        WHERE OrderId = ID;

    SELECT ID, o_name; -- optional
        FETCH OrderCursor INTO ID, o_name;
    END WHILE;

    CLOSE OrderCursor;

END $$
DELIMITER ;


CALL Get_Employee_Details();
```

```sql
-- 4. You have “Customers” table with the following columns :
-- CustomerID,CustomerName, and TotalOrders. Write a cursor named
-- “CustomerCursor” that calculates the total number of orders for each
-- customer and updates the “TotalOrders” column accordingly.

DELIMITER $$
CREATE PROCEDURE UpdateTotalOrders()
BEGIN
  DECLARE CID INT;
    DECLARE Total INT;
    DECLARE done INT DEFAULT 0;

    DECLARE CustomerCursor CURSOR FOR
    SELECT CustomerID FROM Customers;

    DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = 1;

    OPEN CustomerCursor;

    FETCH CustomerCursor INTO CID;

    WHILE done = 0 DO

    SELECT count(*) INTO Total FROM Orders WHERE CustomerID = CID;

    UPDATE Customers SET TotalOrders = Total WHERE CustomerID = CID;
    FETCH CustomerCursor INTO CID;
    END WHILE;
    CLOSE CustomerCursor;
END $$
DELIMITER ;


CALL UpdateTotalOrders();
```

```sql

-- 5. You have an “Inventory” table with the following columns:
-- ProductID,Quantity and Recorderlevel. Write a cursor name
-- “RecorderCursor” that iterates through each product in the inventory and
-- checks if the quantity is below the reorder level. For products that need to
-- be recorded, insert a record into the “ReorderItems” table with the ProductID and Quantity.

DELIMITER $$

CREATE PROCEDURE CheckReorderLevel()
BEGIN
    DECLARE prodID INT;
    DECLARE qty INT;
    DECLARE reorderLevel INT;
    DECLARE done INT DEFAULT 0;

    DECLARE ReorderCursor CURSOR FOR
        SELECT ProductID, Quantity, ReorderLevel FROM Inventory;

    DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = 1;
    OPEN ReorderCursor;

     FETCH ReorderCursor INTO prodID, qty, reorderLevel;

    WHILE done = 0 DO

        IF qty < reorderLevel THEN
            INSERT INTO ReorderItems (ProductID, Quantity)
            VALUES (prodID, qty);
        END IF;
        FETCH ReorderCursor INTO prodID, qty, reorderLevel;
    END WHILE;

    CLOSE ReorderCursor;
END $$

DELIMITER ;


CALL CheckReorderLevel();


```

```sql
-- 6. You have an “Employees” table with the following columns:
-- EmployeeID,EmployeeName, and salary. Write a cursor named
-- “SalaryCursor” that retrieves the salary of each employee and calculates
-- the average salary of all employees.Print the employee name and salary
-- for those employees whose salary is above the average.

DELIMITER $$
CREATE PROCEDURE AverageSalary()
BEGIN
  DECLARE empID INT;
    DECLARE empName VARCHAR(100);
    DECLARE empSalary DECIMAL(10, 2);
    DECLARE avgSalary DECIMAL(10, 2);
    DECLARE done INT DEFAULT 0;

  DECLARE SalaryCursor CURSOR FOR
        SELECT emp_Id, emp_name, salary FROM Employee;

  DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = 1;

    SELECT AVG(salary) INTO avgSalary FROM Employee;

    OPEN SalaryCursor;
  FETCH SalaryCursor INTO empID, empName, empSalary;

  WHILE done = 0 DO
    IF empSalary > avgSalary THEN
        SELECT empName AS emp_name, empSalary AS salary;
      END IF;
    FETCH SalaryCursor INTO empID, empName, empSalary;
  END WHILE;

CLOSE SalaryCursor;
END $$

DELIMITER ;


CALL AverageSalary();
```