MYSQL Stored Procedures

Stored procedures are a feature in database management systems that allow you to encapsulate a set of SQL statements and logic into reusable routines. These routines are stored and executed on the database server, providing several benefits for database development and management.

Benefits of Stored Procedures:

- Modularity and Reusability: Stored procedures promote modular programming by allowing developers to encapsulate frequently used sets of SQL statements and logic into a single unit. This improves code organization and facilitates code reuse across different parts of an application.
- Performance Optimization: Stored procedures can enhance performance by reducing network traffic between the application and the database server. Instead of sending multiple SQL statements to the server, the application can execute a single stored procedure, reducing overhead.
- Improved Security: Stored procedures can help enforce security policies by allowing controlled access to data. Applications can interact with the database through predefined stored procedures, limiting direct access to tables and views. This reduces the risk of SQL injection attacks and unauthorized access to sensitive data.
- Enhanced Maintainability: Centralizing business logic within stored procedures simplifies maintenance and updates. Changes to database schema or business rules can be made in the stored

procedures without modifying the application code, reducing the risk of introducing errors.

- Transaction Management: Stored procedures enable developers to define transaction boundaries within the database. This ensures data integrity by grouping multiple SQL statements into atomic operations, allowing for consistent and reliable database updates.
- Reduced Network Load: By executing logic on the database server, stored procedures can minimize data transfer between the server and client applications. This is particularly beneficial for applications deployed across distributed networks with limited bandwidth

Drawbacks of Stored Procedures:

- Portability: Stored procedures are specific to the database management system (DBMS) in which they are created. Migrating stored procedures between different DBMS platforms may require significant effort and may not be straightforward.
- Debugging and Testing: Testing and debugging stored procedures can be more complex compared to application code. Tools for debugging stored procedures may be limited, leading to longer development cycles and potentially higher maintenance costs.
- Version Control: Managing version control for stored procedures can be challenging, especially in environments with multiple developers working concurrently. Changes to stored procedures may not be tracked effectively, leading to inconsistencies and conflicts.

- Limited Language Features: The procedural language used in stored procedures (e.g., PL/SQL in Oracle, T-SQL in SQL Server) may have limitations compared to general-purpose programming languages. This can restrict the complexity and flexibility of logic that can be implemented within stored procedures.

A. Stored procedures without parameters based on your employee table:

1. Procedure to fetch all employees:

```
DELIMITER $$
CREATE PROCEDURE `GetAllEmployees`()
BEGIN
     SELECT * FROM employees;
END$$
DELIMITER ;

  ★ Calling stored procedures without parameter
CALL GetAllEmployees();
```

2. Procedure to fetch the employee with the highest salary:

```
DELIMITER $$
CREATE PROCEDURE GetEmployeeWithHighestSalary ()
BEGIN
   SELECT * FROM employee ORDER BY salary DESC LIMIT 1;
END$$
DELIMITER ;

  ★ Calling stored procedures without parameter
CALL GetEmployeeWithHighestSalary();
```

B. Stored procedures with IN parameters based on your employee table:

1. Procedure to fetch employee details based on employee ID:

```
DELIMITER $$
CREATE PROCEDURE GetEmployeeByID (IN empID INT)
BEGIN
    SELECT * FROM employee WHERE emp_id = empID;
END$$
DELIMITER ;

  ★ Calling stored procedures with parameter
CALL GetEmployeeByID(4)
```

2. Procedure to fetch employees in a specific department:

```
DELIMITER$$
CREATE PROCEDURE GetEmployeesByDepartment (IN deptName
VARCHAR(255))
BEGIN
    SELECT * FROM employee WHERE department = deptName;
END$$
DELIMITER ;
```

3. Procedure to fetch employees based on salary range:

```
DELIMITER $$
CREATE PROCEDURE GetEmployeesBySalaryRange (IN
minSalary DECIMAL(10,2), IN maxSalary DECIMAL(10,2))
BEGIN
        SELECT * FROM employee WHERE salary BETWEEN
minSalary AND maxSalary;
END$$
DELIMITER ;
```

4. Procedure to fetch employees hired after a specific date:

```
DELIMITER $$
CREATE PROCEDURE GetEmployeesHiredAfterDate (IN
hireDate DATE)
BEGIN
    SELECT * FROM employee WHERE hiredate > hireDate;
END$$
DELIMITER ;
```

5. Procedure to fetch employees in a specific city:

```
DELIMITER $$
CREATE PROCEDURE GetEmployeesByCity (IN cityName
VARCHAR(255))
BEGIN
    SELECT * FROM employee WHERE city = cityName;
END$$
DELIMITER ;
```

6. Procedure to fetch employees based on experience:

```
DELIMITER $$
CREATE PROCEDURE GetEmployeesByExperience (IN expYears
INT)
BEGIN
    SELECT * FROM employee WHERE experience_years >=
expYears;
END$$
DELIMITER ;
```

7. Procedure to fetch employees holding a specific position:

```
DELIMITER $$
CREATE PROCEDURE GetEmployeesByPosition (IN
positionName VARCHAR(255))
BEGIN
    SELECT * FROM employee WHERE position =
positionName;
END$$
DELIMITER ;
```

8. Procedure to fetch employees based on gender:

```
DELIMITER $$
CREATE PROCEDURE GetEmployeesByGender (IN empGender
VARCHAR(20))
BEGIN
    SELECT * FROM employee WHERE gender = empGender;
END$$
DELIMITER ;
```

D. Stored procedures with OUT parameters based on your employee table:

9. Procedure to get the count of employees in a specific department:

```
DELIMITER $$
CREATE PROCEDURE GetEmployeeCountByDepartment (IN
deptName VARCHAR(255), OUT empCount INT)
BEGIN
    SELECT COUNT(*) INTO empCount FROM employee WHERE
department = deptName;
END//
DELIMITER ;
```

★ Calling stored procedures with parameter

```
CALL GetEmployeeCountByDepartment('HR', @empCount);
SELECT @empCount;
```

10. Procedure to get the average salary in a specific department:

```
DELIMITER //
CREATE PROCEDURE GetAverageSalaryByDepartment (IN
deptName VARCHAR(255), OUT avgSalary DECIMAL(10,2))
BEGIN
      SELECT  AVG(salary)  INTO  avgSalary  FROM  employee
WHERE department = deptName;
END//
DELIMITER ;

  ★ Calling stored procedures with parameter
CALL GetAverageSalaryByDepartment('IT', @avgSalary);
SELECT @avgSalary;
```

11. Procedure to get the maximum salary in the company

```
DELIMITER //
CREATE PROCEDURE GetMaxSalary (OUT maxSalary DECIMAL(10,2))
BEGIN
  SELECT MAX(salary) INTO maxSalary FROM employee;
END//
DELIMITER ;

  ★ Calling stored procedures with parameter
CALL GetMaxSalary(@maxSalary);
SELECT @maxSalary;
```

12. Procedure to get the latest hire date in the company:

```
DELIMITER //
CREATE PROCEDURE GetLatestHireDate (OUT latestHireDate DATE)
BEGIN
   SELECT MAX(hiredate) INTO latestHireDate FROM employee;
END//
DELIMITER ;
```

★ Calling stored procedures with parameter

```
CALL GetLatestHireDate(@latestHireDate);
SELECT @latestHireDate;
```

13. Procedure to get the count of employees in a specific city:

```
DELIMITER //
CREATE PROCEDURE GetEmployeeCountByCity (IN cityName
VARCHAR(255), OUT cityEmployeeCount INT)
BEGIN
      SELECT COUNT(*) INTO cityEmployeeCount FROM employee
WHERE city = cityName;
END//
DELIMITER
```

★ Calling stored procedures with parameter

```
CALL GetEmployeeCountByCity(@cityEmployeeCount);
SELECT @cityEmployeeCount;
```