MySQL: Stored Procedures vs Functions

Stored Procedures

Definition

- Pre-compiled SQL statements stored in the database.
- Designed to perform **multiple operations**.

Key Features

- Can **accept input parameters**
- Can **return multiple values or result sets**
- Can include **control flow** (IF, WHILE, etc.)
- Can **modify database state**
- Stored in the database server and invoked with CALL

Syntax Example

sql

CopyEdit

```
DELIMITER //

CREATE PROCEDURE GetEmployeesByDepartment(IN dept_name VARCHAR(50))
BEGIN
  SELECT * FROM employees WHERE department = dept_name;
END //

DELIMITER ;
```

Usage:

sql

CopyEdit

```
CALL GetEmployeesByDepartment('Marketing');
```

Functions

Definition

- Pre-compiled SQL block that **must return a single value**
- Cannot modify database state (only READ/DETERMINISTIC)

Key Features

- Used inside SQL expressions (e.g., in SELECT clause)
- Cannot have side-effects (e.g., no INSERT/UPDATE)
- Great for **reusable logic** and **calculations**

Syntax Example

sql

CopyEdit

```
DELIMITER //

CREATE FUNCTION CalculateBonus(salary DECIMAL(10,2), performance_score INT)
RETURNS DECIMAL(10,2)
DETERMINISTIC
BEGIN
  DECLARE bonus DECIMAL(10,2);
  SET bonus = salary * (performance_score / 100);
  RETURN bonus;
END //

DELIMITER ;
```

Usage:

sql

CopyEdit

```
SELECT employee_name, CalculateBonus(salary, performance_score) AS bonus
FROM employees;
```

Stored Procedures vs Functions

Feature	**Stored Procedures**	**Functions**
Return Value	Multiple values or none	Must return **a single value**
Modify DB State	Yes	No (except UDFs with limitations)
Use in SQL SELECT	No	Yes
Control Flow	(IF, WHILE, etc.)	Limited
Purpose	Perform actions	Return a computed value

When to Use Stored Procedures

- **Complex operations** involving multiple SQL statements
- **Data integrity** needs multiple coordinated steps
- **Security**, to restrict direct table access
- **Performance**, reduces network calls
- **Maintainability**, centralizes business logic

When to Use Functions

- **Reusable calculations**
- **Data formatting/transformation**
- **Custom aggregations**
- **Encapsulating logic** used in SELECT or WHERE

Best Practices

- **Naming**: Use clear, descriptive names
- **Input Validation**: Prevent SQL injection, ensure clean inputs
- **Error Handling**: Use DECLARE ... HANDLER or validations
- **Documentation**: Comment all logic for future reference

- **Performance**: Avoid unnecessary operations, use indexes

FAQs

1. **What's the difference between a stored procedure and a function?**
   → Procedures can return multiple values & modify data; functions return one value & cannot modify data.
2. **When to prefer stored procedure over function?**
   → Use for multi-step operations, data modification, or centralizing business logic.
3. **Can a function modify data?**
   → No. Functions are read-only (unless UDF with READS SQL DATA etc.)
4. **How to optimize stored procedures?**
   → Reduce SQL calls, use indexing, bundle operations, avoid unnecessary logic.
5. **Security tips for stored procedures/functions?**
   → Use parameterized queries, validate inputs, apply role-based access control.
6. **When to avoid both?**
   → For **very simple queries** or **when logic is better handled in application code**.

Conclusion

- **Stored Procedures**: Best for multi-step, secure, and modifiable logic blocks.
- **Functions**: Best for reusable, lightweight, read-only computations.

Choose based on use-case complexity, performance needs, and desired reusability