**functions** are named blocks that **perform a task and return a single value**. They are similar to procedures but differ primarily in that functions **must return a value**, while procedures **may not**.

**Syntax of a PL/SQL Function (Oracle syntax)**

sql

CopyEdit

CREATE [OR REPLACE] FUNCTION function\_name (

parameter1 [IN | OUT | IN OUT] datatype,

parameter2 ...

)

RETURN return\_datatype

IS

-- Variable declarations

BEGIN

-- Function logic

RETURN return\_value;

EXCEPTION

-- Exception handling

END function\_name;

## Example: Simple Function to Add Two Numbers

CREATE OR REPLACE FUNCTION add\_numbers (

num1 IN NUMBER,

num2 IN NUMBER

)

RETURN NUMBER

IS

result NUMBER;

BEGIN

result := num1 + num2;

RETURN result;

END;

**Usage-**

**DECLARE**

**res NUMBER;**

**BEGIN**

**res := add\_numbers(10, 20);**

**DBMS\_OUTPUT.PUT\_LINE('Result: ' || res);**

**END;**

**🔹 Key Points about PL/SQL Functions**

| **Feature** | **Description** |
| --- | --- |
| **RETURN** | Required in functions to return a value |
| **Parameters** | Can be IN, OUT, or IN OUT (most commonly IN for functions) |
| **Exception Block** | Optional, but useful for handling errors |
| **Stored Function** | Can be stored in the database and reused |
| **Can Be Used In** | SELECT statements (if deterministic), procedures, packages, triggers |

## 🔹 Example: Function to Get Employee Name by ID

Assume there's a table employees(emp\_id NUMBER, emp\_name VARCHAR2).

CREATE OR REPLACE FUNCTION get\_emp\_name (

p\_emp\_id IN NUMBER

)

RETURN VARCHAR2

IS

v\_emp\_name VARCHAR2(100);

BEGIN

SELECT emp\_name INTO v\_emp\_name

FROM employees

WHERE emp\_id = p\_emp\_id;

RETURN v\_emp\_name;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

RETURN 'Employee not found';

END;

**🔹 Calling a Function in SQL (If it does not use DML)**

SELECT get\_emp\_name(101) FROM dual;

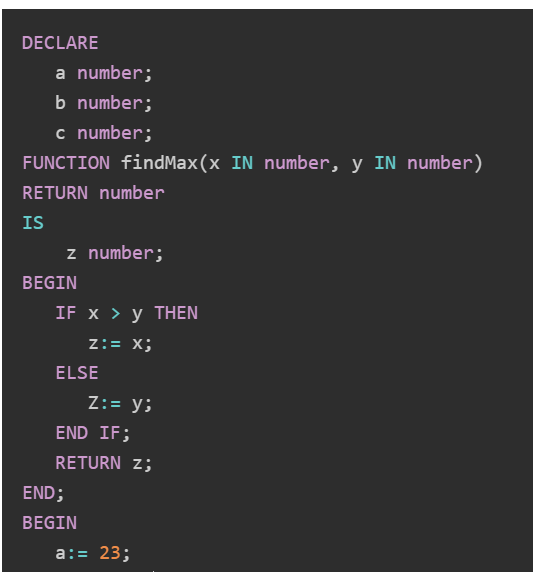
**Note:** You cannot call a function that performs DML (like INSERT, UPDATE) directly in a SQL query.

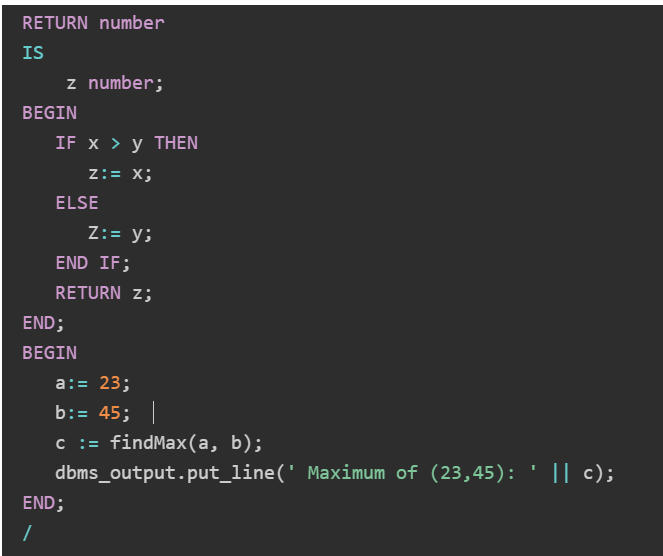
**🔹 Differences Between Function and Procedure**

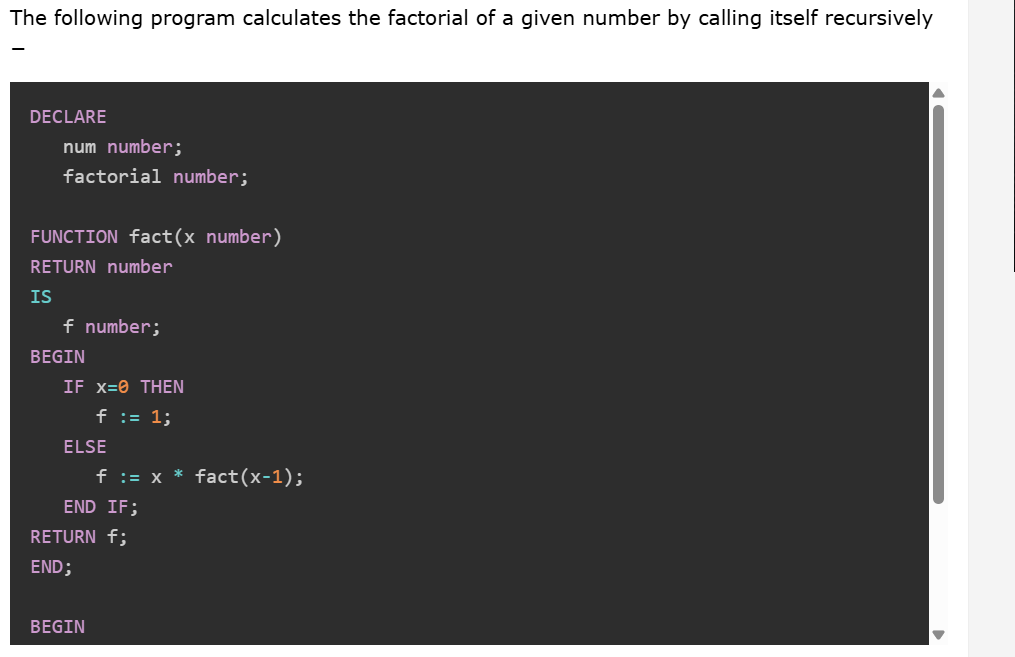
| **Feature** | **Function** | **Procedure** |
| --- | --- | --- |
| **Returns Value** | Yes (mandatory via RETURN) | No (optional, use OUT param) |
| **Used in SQL?** | Yes (if no side effects) | No |
| **Primary Use** | Computations, returning a value | Performing actions |

### **Example**

The following example demonstrates Declaring, Defining, and Invoking a Simple PL/SQL Function that computes and returns the maximum of two values.

****

****

****

****

**MYSQL-**

### ✅ **Function Syntax in MySQL**

In **MySQL**, a **function** is a stored program that accepts parameters, performs actions, and **returns a single value**. Functions can be used **in SQL statements** like SELECT, WHERE, etc.

## 🔹 ****Basic Syntax to Create a Function****

DELIMITER $$

CREATE FUNCTION function\_name (

param1 datatype,

param2 datatype

)

RETURNS return\_datatype

DETERMINISTIC -- or NOT DETERMINISTIC

BEGIN

-- Declare variables (if needed)

-- Perform operations

RETURN some\_value;

END$$

DELIMITER ;

## 🔹 ****Key Clauses Explained****

| **Clause** | **Description** |
| --- | --- |
| CREATE FUNCTION | Starts the function definition |
| param1 datatype | Input parameters with their data types |
| RETURNS | Specifies the return type (e.g., INT, VARCHAR, DECIMAL) |
| DETERMINISTIC | Means function always returns same result for same input (can affect caching) |
| RETURN | The value the function returns |

## 🧠 ****Example: Add Two Numbers****

DELIMITER $$

CREATE FUNCTION add\_numbers(a INT, b INT)

RETURNS INT

DETERMINISTIC

BEGIN

RETURN a + b;

END$$

DELIMITER ;

### ✅ **Call the Function:**

sql

CopyEdit

SELECT add\_numbers(10, 20); -- Output: 30

## 🧠 ****Example: Get Employee Full Name****

Assume a table employees(emp\_id, first\_name, last\_name):

\

DELIMITER $$

CREATE FUNCTION get\_full\_name(empid INT)

RETURNS VARCHAR(100)

DETERMINISTIC

BEGIN

DECLARE full\_name VARCHAR(100);

SELECT CONCAT(first\_name, ' ', last\_name)

INTO full\_name

FROM employees

WHERE emp\_id = empid;

RETURN full\_name;

END$$

DELIMITER ;

### ✅ Usage:

sql

CopyEdit

SELECT get\_full\_name(101);

## 🔸 Notes & Limitations

* MySQL **functions cannot perform DML** (INSERT, UPDATE, DELETE).
* Use **procedures** for complex logic that modifies data.
* You must have **CREATE ROUTINE** privilege.

## 🔧 Drop a Function

DROP FUNCTION IF EXISTS function\_name;

**Triggers-**

### ✅ **Triggers in MySQL**

A **trigger** in MySQL is a **database object** that is **automatically executed (or "triggered")** when a specific event (like INSERT, UPDATE, or DELETE) occurs on a table.

## 🔹 ****Basic Syntax of a Trigger****

sql

CopyEdit

CREATE TRIGGER trigger\_name

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

ON table\_name

FOR EACH ROW

BEGIN

-- Trigger logic here

END;

## 🔸 Trigger Timing:

| **Timing** | **Meaning** |
| --- | --- |
| BEFORE | Trigger fires **before** the operation |
| AFTER | Trigger fires **after** the operation |

## 🔸 Trigger Events:

| **Event** | **Description** |
| --- | --- |
| INSERT | Trigger when a new row is inserted |
| UPDATE | Trigger when a row is updated |
| DELETE | Trigger when a row is deleted |

## 🧠 ****Example 1: BEFORE INSERT Trigger****

Let’s say we want to **automatically set the created date** when a new record is inserted into a users table.

sql

CopyEdit

DELIMITER $$

CREATE TRIGGER before\_insert\_users

BEFORE INSERT ON users

FOR EACH ROW

BEGIN

SET NEW.created\_at = NOW();

END$$

DELIMITER ;

🟢 NEW is a keyword that refers to the **new row being inserted**.

## 🧠 ****Example 2: AFTER UPDATE Trigger****

Let’s say you want to **log changes** to an orders table in an orders\_log table after an update.

sql

CopyEdit

DELIMITER $$

CREATE TRIGGER after\_update\_orders

AFTER UPDATE ON orders

FOR EACH ROW

BEGIN

INSERT INTO orders\_log(order\_id, old\_status, new\_status, changed\_on)

VALUES (OLD.id, OLD.status, NEW.status, NOW());

END$$

DELIMITER ;

🟢 OLD refers to the **previous value** before update.

## 🧠 ****Example 3: AFTER DELETE Trigger****

Log deleted rows from a products table:

sql

CopyEdit

DELIMITER $$

CREATE TRIGGER after\_delete\_products

AFTER DELETE ON products

FOR EACH ROW

BEGIN

INSERT INTO deleted\_products\_log(product\_id, product\_name, deleted\_on)

VALUES (OLD.id, OLD.name, NOW());

END$$

DELIMITER ;

## 🔹 Viewing Triggers

sql

CopyEdit

SHOW TRIGGERS;

## 🔹 Dropping a Trigger

sql

CopyEdit

DROP TRIGGER IF EXISTS trigger\_name;

## 🔸 Limitations in MySQL Triggers

| **Limitation** | **Description** |
| --- | --- |
| No INSTEAD OF triggers | Only available in some databases like SQL Server |
| Cannot trigger another trigger recursively | Prevents infinite loops |
| No transaction control (COMMIT/ROLLBACK) | Cannot commit/rollback inside a trigger |