# PL/SQL

- Pl/SQL stands for "Procedural Language extension of SQL" that is used in Oracle.

- pl/SQL is a block structured language that can have multiple blocks in it.

- A line of PL/SQL text contains s groups of characters known as lexical units. It can be classified as follows:

```
DELIMITER
   // BEGIN
         ............
  END
// DELIMITER ;
```

## MySQL Stored Function with IF...ELSEIF...ELSE Control Statement

## Database Context :-

**USE student;**

Switches the active database to student so all queries/functions run in this DB.

## System Variable Setup

**SET GLOBAL log_bin_trust_function_creators = ON;**

- MySQL by default doesn't allow creating functions when binary logging is enabled (for replication safety).

- Setting log_bin_trust_function_creators to ON allows function creation without SUPER privilege.
- Note: Requires proper permissions to set globally.

## Function Creation

```
DELIMITER //

CREATE FUNCTION check_if(a INT)
RETURNS VARCHAR(30)
DETERMINISTIC
BEGIN
  DECLARE income VARCHAR(30);
  IF a = 100 THEN
    SET income = ‘your income is 100’;
  ELSEIF a = 200 THEN
    SET income = ‘your income is 200’;
  ELSEIF a = 300 THEN
    SET income = ‘your income is 300’;
  ELSE
    SET income = ‘invalid income’;
  END IF;

  RETURN income;

END //

DELIMITER ;
```

## Key Points:

1. **DELIMITER //**
   - Changes the default statement terminator from ; to // so MySQL treats the entire function as one block.
2. **CREATE FUNCTION check_if(a INT)**
   - Defines a stored function named check_if that accepts one integer parameter a.
3. **RETURNS VARCHAR(30)**
   - Specifies the return type (up to 30 characters).
4. **DETERMINISTIC**
   - Means for the same input value, the function will always return the same output.
5. **DECLARE income VARCHAR(30)**
   - Creates a local variable income to store the message.
6. **Control Flow (IF...ELSEIF...ELSE)**
   - Checks the value of a and assigns a corresponding message to income.
   - If no match, sets "invalid income".
7. **RETURN income**
   - Sends the message back to the caller.
8. **DELIMITER ;**
   - Resets delimiter back to the default.

## Function Execution

```
SELECT check_if(100);  -- Returns: 'your income is 100'

SELECT check_if(400);  -- Returns: 'invalid income'
```

### Learning Takeaways

- Stored functions can encapsulate logic and return single values.
- IF...ELSEIF...ELSE helps branch execution based on conditions.
- Always handle default cases to avoid unexpected NULL results.
- Changing the delimiter is important when writing multi-line procedural code in MySQL.

## 1. Using LOOP + ITERATE + LEAVE

```
USE student;
SET GLOBAL log_bin_trust_function_creators = 1;

DELIMITER //
CREATE FUNCTION cal_income(val INT)
RETURNS INT
BEGIN
    DECLARE income INT;
    SET income = 0;

    label1: LOOP
        SET income = income + val;
        IF income < 3000 THEN
            ITERATE label1; -- Go to the next loop cycle
        END IF;
        LEAVE label1; -- Exit loop
    END LOOP label1;

    RETURN income;
END; //
DELIMITER ;

SELECT cal_income(100);
```

**Explanation:**

- Purpose: Add val repeatedly to income until it reaches at least 3000.
- LOOP: Executes repeatedly until explicitly stopped.

- ITERATE label1: Skips remaining code in the current loop and starts the next iteration.
- LEAVE label1: Breaks the loop entirely.
- Flow:
    1. Start at 0.
    2. Keep adding val to income.
    3. If still under 3000 → repeat.
    4. If ≥ 3000 → exit and return value.

## 2. Using WHILE Loop

```
USE college;
SET GLOBAL log_bin_trust_function_creators = 1;

DELIMITER //
CREATE FUNCTION CalcIncome(starting_value INT)
RETURNS INT
BEGIN
    DECLARE income INT;
    SET income = 0;

    label1: WHILE income <= 3000 DO
        SET income = income + starting_value;
    END WHILE label1;

    RETURN income;
END; //
DELIMITER ;



SELECT CalcIncome(100);
```

### Explanation:

- Purpose: Same end goal — keep adding until income exceeds 3000.
- WHILE: Continues execution as long as the condition (income <= 3000) is true.
- No ITERATE or LEAVE: The loop exits naturally when the condition fails.
- Flow:
    1. Start at 0.
    2. Keep adding starting_value to income while it's ≤ 3000.
    3. When income becomes greater than 3000 → loop ends, return value.

## MySQL Stored Function Using CASE Statement

```
USE student;

SET GLOBAL log_bin_trust_function_creators = 1;


DELIMITER //

CREATE FUNCTION check_case(val INT)

RETURNS VARCHAR(20)

BEGIN

   DECLARE income_level VARCHAR(20);


   CASE val

      WHEN 1000 THEN

         SET income_level = 'Low Income';

      WHEN 5000 THEN

         SET income_level = 'Avg Income';

      ELSE

            SET income_level = 'High Income';

   END CASE;
```

```
    RETURN income_level;
END; //
DELIMITER ;


SELECT check_case(1000); -- 'Low Income'
SELECT check_case(5000); -- 'Avg Income'
SELECT check_case(7000); -- 'High Income'
```

## Explanation

- **CASE statement: Similar to switch-case in other languages; compares val against fixed values.**
- **Matching:**
  - **If val = 1000 → "Low Income"**
  - **If val = 5000 → "Avg Income"**
  - **If no match → "High Income" (default case via ELSE).**
- **DECLARE income_level: Local variable to store the result.**
- **RETURN income_level: Sends the message back to the caller.**

## Key Points

- Purpose: Categorizes income levels based on a fixed set of input values.
- Advantages of CASE:
  - Cleaner than multiple IF...ELSEIF statements when comparing to constants.
  - Easier to read and maintain.
- Syntax Tip:
  - CASE <expression> → compare against values using WHEN.
  - ELSE → default action when no match is found.
  - Always end with END CASE;.

## Learning Takeaway

- **Use CASE when you have multiple discrete values to compare.**
- **For range-based checks, IF...ELSE might be better.**
- **Always provide an ELSE to handle unexpected input values.**

## Procedure

- It is just like procedures in other programming languages.
- **Header:**The header contains the name of the procedure and the parameters or variables passed to the procedure.
- Ways to pass parameter

  **IN OUT**

- **Body**:The body contains a declaration section, execution section and exception section similar to a general PL/SQL block.

### Syntax

```
DELIMITER //

CREATE PROCEDURE procedure_name (parameters…)

BEGIN

 ......statements

END //

DELIMITER ;
```

## Topic: MySQL Stored Procedure for Data Insertion

```
DELIMITER //
CREATE PROCEDURE insert_user
(IN p_id INT(10), IN p_name VARCHAR(100))
BEGIN
  INSERT INTO user(id, name) VALUES(p_id, p_name);
END //
DELIMITER ;
```

## Explanation

- CREATE PROCEDURE insert_user
  - Defines a stored procedure named insert_user.
- Parameters:
  - p_id → Integer input (user ID).
  - p_name → String input (user name).
- INSERT INTO user(id, name) VALUES(...)
  - Inserts the given parameter values into the user table.
- DELIMITER //
  - Changes the statement delimiter so MySQL reads the whole procedure definition as one block.

**Calling the Procedure:**

```
CALL insert_user(1, 'John Doe');
```

## Key Points

- Procedures vs. Functions:
  - Procedures do not return values directly; they perform actions like inserts/updates.
  - Functions return a value and can be used in SELECT statements.
- Benefits of Stored Procedures:
  - Encapsulate frequently used SQL logic.
  - Improve code reusability and maintainability.
  - Can include complex business logic (loops, conditions, etc.).

## Function in MySQL

### Definition

- **A Function is similar to a Procedure, but it must always return a value.**
- **Procedure: May or may not return a value.**
- **Function: Must return exactly one value.**

## 📜 Syntax

```
DELIMITER //
CREATE FUNCTION function_name
(parameters…)
RETURNS datatype
BEGIN
  -- statements
  RETURN value;
END //
DELIMITER ;
```

## Example

```
DELIMITER //
CREATE FUNCTION my_square(val INT)
RETURNS INT
BEGIN
   DECLARE result INT;
   SET result = 0;
   SET result = val * val;
   RETURN result;
END //
DELIMITER ;
```

## 💡 Usage

```
SELECT my_square(5); -- Output: 25
```

## 🔍 Key Points

- Always specify the return type using RETURNS datatype.
- Use RETURN to send the value back to the caller.
- Functions can be used in SELECT, WHERE, or other SQL expressions.
- Must be deterministic if they produce the same result for the same inputs.

# 📌 Trigger in MySQL

## Definition

- A Trigger is a stored program that is automatically executed (fired) when a specific event occurs in a table.
- Used to enforce rules, log activity, generate derived values, or implement security checks.

---

## 🔍 Key Uses

- Automatically generate derived column values.
- Event logging and auditing table access.
- Enforcing security authorizations.

---

## 🧾 Events That Can Fire a Trigger

- **INSERT** → BEFORE INSERT, AFTER INSERT
- **UPDATE** → BEFORE UPDATE, AFTER UPDATE
- **DELETE** → BEFORE DELETE, AFTER DELETE

## 📜 Syntax

```
DELIMITER //

CREATE TRIGGER trigger_name

BEFORE | AFTER INSERT | UPDATE | DELETE

ON table_name

FOR EACH ROW

BEGIN

  -- statements to execute
END //
DELIMITER ;
DELIMITER //
CREATE TRIGGER before_insert_money
BEFORE INSERT ON money
FOR EACH ROW
BEGIN
   UPDATE employee SET salary = 'Credited';
END //
DELIMITER ;

INSERT INTO money VALUES('100');
```

## 💡 How It Works

1. When an INSERT is made into the money table,
   the trigger automatically updates the employee table's salary column to 'Credited'.
2. No manual execution is required — it fires automatically.

## ⚠ Key Points

- BEFORE triggers run before the event changes the table.
- AFTER triggers run after the event changes the table.
- FOR EACH ROW means the trigger runs once for every row affected by the event.
- Triggers are linked to specific tables and specific events.