A. Trigger

In SQL, a trigger is a special type of stored procedure that automatically executes or "fires" in response to specific events or changes in a database. Triggers are used to enforce business rules, automate tasks, and maintain data integrity.

Features

1. Automatic Execution:
   Triggers are executed automatically in response to events like `INSERT`, `UPDATE`, or `DELETE` on a table.
2. Event-Driven:
   They are triggered by specific database events and can be set to run before or after the event occurs.
3. Maintain Data Integrity:
   Triggers can enforce rules and constraints to ensure data consistency and integrity.
4. Complex Logic:
   They allow for the execution of complex logic that cannot be easily handled with constraints alone.

Types

1. **BEFORE** Triggers:
   Executed before an `INSERT`, `UPDATE`, or `DELETE` operation on a table.
2. **AFTER** Triggers:
   Executed after an `INSERT`, `UPDATE`, or `DELETE` operation on a table.
3. **INSTEAD OF** Triggers:
   Executed in place of the triggering operation, allowing for custom logic to replace the standard operation.

Syntax

```
CREATE FUNCTION trigger_function()
RETURNS TRIGGER
AS $$
BEGIN
   -- trigger logic here
   RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trigger_name
BEFORE INSERT ON table_name
FOR EACH ROW
EXECUTE FUNCTION trigger_function();
```

Example

```
CREATE OR REPLACE FUNCTION log_salary_changes()
RETURNS TRIGGER AS $$
BEGIN
   -- Insert a record into salary_log for each update
   INSERT INTO salary_log (employee_id, old_salary, new_salary)
   VALUES (OLD.employee_id, OLD.salary, NEW.salary);

   -- Return the new row
   RETURN NEW;
END;
$$ LANGUAGE plpgsql;
```

```
CREATE TRIGGER trigger_salary_changes
AFTER UPDATE OF salary
ON employees
FOR EACH ROW
EXECUTE FUNCTION log_salary_changes();
```

B. Stored Procedures

A stored procedure is a set of SQL statements that perform a specific task, which is stored in the database and can be executed by calling the procedure. Unlike functions, stored procedures do not return a value but can perform operations such as modifying data or managing transactions.

Syntax

```
CREATE PROCEDURE procedure_name (param1 type1, param2 type2, ...)
LANGUAGE plpgsql
AS $$

DECLARE
   Var1 DATATYPE;
   var2 DATATYPE;
   …
   var_n DATATYPE;

BEGIN
   -- Procedure logic here
END;
$$;
```

Example

```
CREATE PROCEDURE update_department_average(department_name VARCHAR)
LANGUAGE plpgsql
AS $$

DECLARE
   total_salary DECIMAL;
   employee_count INT;
   avg_salary DECIMAL;

BEGIN
   -- Calculate total salary and employee count for the department
   SELECT SUM(salary), COUNT(*)
   INTO total_salary, employee_count
   FROM employees
   WHERE department = department_name;

   -- Calculate average salary
   IF employee_count > 0 THEN
      avg_salary := total_salary / employee_count;
   ELSE
      avg_salary := 0;
   END IF;
```

```
    -- Update department table with average salary (assuming there's a department table)
    UPDATE departments
    SET average_salary = avg_salary
    WHERE department_name = department_name;
END;
$$;

-- Calling the procedure
CALL update_department_average('Engineering');
```

Parameters

1. IN Parameters: Used to pass values into the procedure.
2. OUT Parameters: Used to return values from the procedure to other procedure.
3. INOUT Parameters: Can be used to pass values in and return values out.

C. Functions

A function in SQL is a stored routine that returns a value. Functions can be used in SQL statements and can perform computations, return results, and be used in SELECT, WHERE, or ORDER BY clauses.

Syntax

```
CREATE FUNCTION function_name (param1 type1, param2 type2, ...)
RETURNS return_type
LANGUAGE plpgsql
AS $$

DECLARE
    var1DATATYPE;
    var2 DATATYPE;
    …
    var_n DATATYPE;

BEGIN
    -- Function logic here
    RETURN return_value;
END;
$$;
```

Example

```
CREATE FUNCTION get_total_salary(department_name VARCHAR)
RETURNS DECIMAL
LANGUAGE plpgsql
AS $$
DECLARE
    total_salary DECIMAL;
BEGIN
    -- Calculate total salary for the department
    SELECT SUM(salary)
    INTO total_salary
    FROM employees
    WHERE department = department_name;

    -- Return the total salary
```

```
    RETURN total_salary;
END;
$$;

-- Calling the function
SELECT get_total_salary('Engineering');
```

Difference between Function and Procedure

1. **Return Value:**
   Function: Returns a single value or a set of values (e.g., scalar, table).
   Procedure: Does not return a value directly; can use `OUT` parameters to return values.

2. **Usage in SQL Statements:**
   Function: Can be used within SQL statements (e.g., `SELECT`, `WHERE`).
   Procedure: Cannot be used within SQL statements; called using `CALL`.

3. **Purpose:**
   Function: Typically used for computations and data retrieval.
   Procedure: Used for performing operations, including data modification and managing transactions.

4. **Transactional Control:**
   Function: Limited control over transactions (cannot commit or roll back transactions within the function).
   Procedure: Can include explicit transaction control (e.g., `COMMIT`, `ROLLBACK`).

5. **Side Effects:**
   Function: Ideally should not have side effects (e.g., modifying database state).
   Procedure: Often used to perform operations that change the database state.