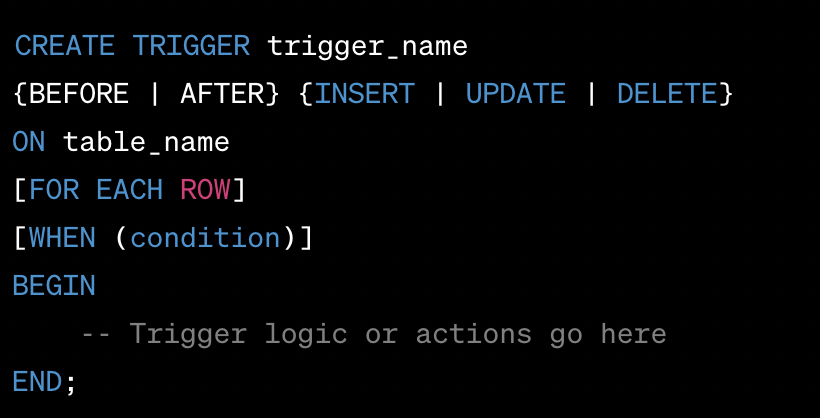
**Relation Database (SQL)**

1. **Trigger**

🡪A relational database trigger is a set of instructions that are automatically executed in response to specific events or actions occurring within a database. These events can include actions such as inserting, updating, or deleting data in a table. Triggers are commonly used to enforce data integrity, implement business rules, perform data validation, or automate certain tasks within a database system.

**Syntax:**



**Example 1: Enforcing a Constraint:**

CREATE TRIGGER prevent\_customer\_deletion

BEFORE DELETE

ON customers

FOR EACH ROW

BEGIN

DECLARE order\_count INT;

SELECT COUNT(\*) INTO order\_count FROM orders WHERE customer\_id = OLD.customer\_id;

IF order\_count > 0 THEN

SIGNAL SQLSTATE '45000'

SET MESSAGE\_TEXT = 'Cannot delete customer with associated orders';

END IF;

END;

**Example 2: Auditing Changes:**

🡪This trigger logs any updates made to the "employees" table in an audit table called "employee\_audit".

CREATE TRIGGER employee\_audit\_trigger

AFTER UPDATE

ON employees

FOR EACH ROW

BEGIN

INSERT INTO employee\_audit (employee\_id, old\_salary, new\_salary, updated\_at)

VALUES (OLD.employee\_id, OLD.salary, NEW.salary, CURRENT\_TIMESTAMP);

END;

**Example 3: Calculating Aggregates:**

🡪This trigger updates the "orders" table with the total order amount whenever a new order is inserted or an existing order is updated.

CREATE TRIGGER update\_order\_total

AFTER INSERT, UPDATE

ON order\_items

FOR EACH ROW

BEGIN

UPDATE orders

SET total\_amount = (

SELECT SUM(quantity \* price) FROM order\_items WHERE order\_id = NEW.order\_id

)

WHERE order\_id = NEW.order\_id;

END;

**Example 4: Generating Unique Identifier:**

🡪This trigger automatically generates a unique invoice number for each newly inserted row in the "invoices" table.

CREATE TRIGGER generate\_invoice\_number

BEFORE INSERT

ON invoices

FOR EACH ROW

BEGIN

SET NEW.invoice\_number = CONCAT('INV-', UUID());

END;

**Example 5: Denormalizing Data:**

🡪This trigger denormalizes the "employees" table by updating the "department\_name" column whenever a corresponding department row is updated.

CREATE TRIGGER update\_employee\_department

AFTER UPDATE

ON departments

FOR EACH ROW

BEGIN

IF NEW.department\_id = OLD.department\_id THEN

UPDATE employees

SET department\_name = NEW.department\_name

WHERE department\_id = NEW.department\_id;

END IF;

END;

1. **Stored Procedure:**

🡪 A stored procedure is a set of precompiled SQL statements that are stored in a database. It allows you to encapsulate complex database operations and logic into a single unit, which can be executed repeatedly without rewriting the code. Stored procedures are widely used in relational databases to improve performance, maintain data integrity, and enhance security. The syntax for creating a stored procedure varies slightly depending on the specific database management system (DBMS) being used. I'll provide an example using the SQL syntax commonly used in many relational database systems.

**Syntax:**

CREATE PROCEDURE procedure\_name ([parameter1 datatype1 [, parameter2 datatype2 [,...]]])

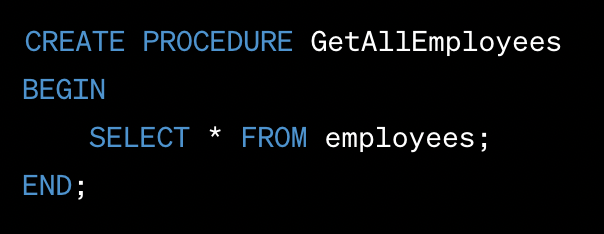
[RETURNS return\_datatype]

BEGIN

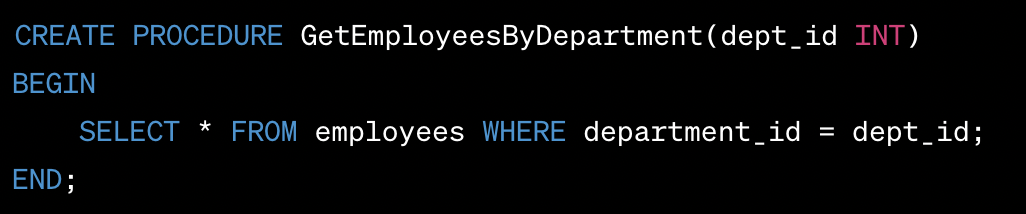
-- SQL statements and logic go here

END;

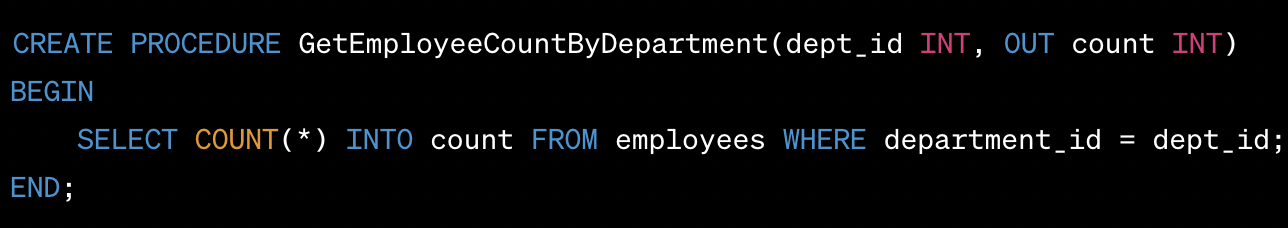
**Example 1: Creating simple procedure without parameter.**



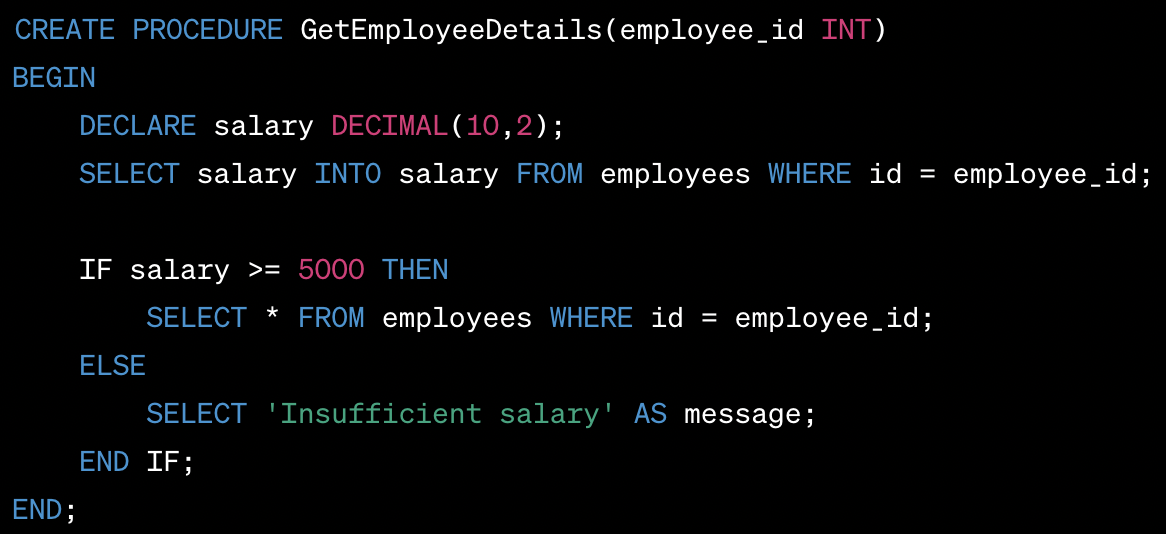
**Example 2: Creating a stored procedure with input parameter.**



**Example 3: Creating a stored procedure with output parameter.**



**Example 4: Creating a stored procedure with conditional Statement.**



**Complex Example:**

CREATE PROCEDURE ProcessOrder(order\_id INT)

BEGIN

DECLARE EXIT HANDLER FOR SQLEXCEPTION

BEGIN

ROLLBACK;

SELECT 'An error occurred. Transaction rolled back.' AS message;

END;

START TRANSACTION;

-- Retrieve order details

SELECT \* FROM orders WHERE id = order\_id;

-- Retrieve customer details

SELECT \* FROM customers WHERE id = (SELECT customer\_id FROM orders WHERE id = order\_id);

-- Update order status

UPDATE orders SET status = 'Processed' WHERE id = order\_id;

-- Decrease product quantity and calculate total price

DECLARE total\_price DECIMAL(10,2) DEFAULT 0;

DECLARE product\_id INT;

DECLARE product\_qty INT;

DECLARE product\_price DECIMAL(10,2);

DECLARE cur CURSOR FOR SELECT id, quantity, price FROM order\_items WHERE order\_id = order\_id;

OPEN cur;

FETCH cur INTO product\_id, product\_qty, product\_price;

WHILE (product\_id IS NOT NULL) DO

UPDATE products SET quantity = quantity - product\_qty WHERE id = product\_id;

SET total\_price = total\_price + (product\_qty \* product\_price);

FETCH cur INTO product\_id, product\_qty, product\_price;

END WHILE;

CLOSE cur;

-- Update customer's total spent amount

UPDATE customers SET total\_spent = total\_spent + total\_price WHERE id = (SELECT customer\_id FROM orders WHERE id = order\_id);

COMMIT;

SELECT 'Order processed successfully.' AS message;

END;

1. **Functions:**

🡪 In SQL, a function is a pre-defined or user-defined operation that can be invoked to perform specific tasks or calculations. Functions can be used to manipulate data, perform calculations, format values, and more. There are two main types of functions in SQL:

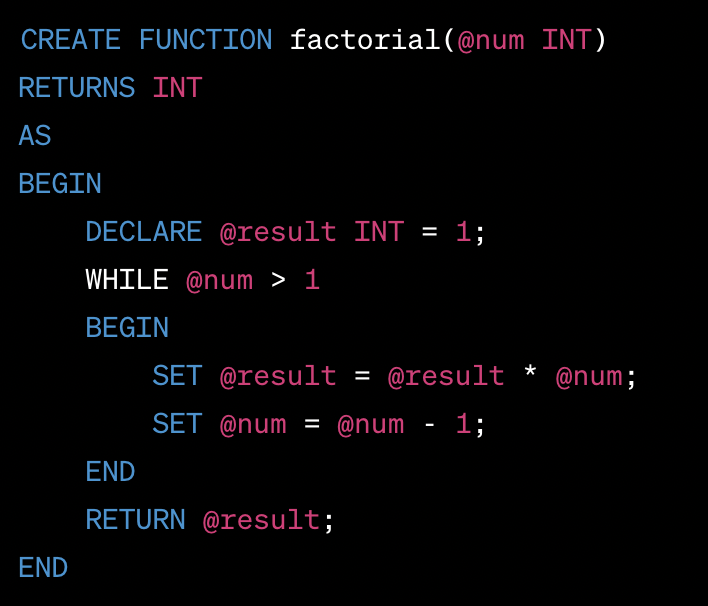
**Built-in Functions:** These functions are provided by the database management system (DBMS) and are available for use without any additional setup. Common built-in functions include:

* Mathematical Functions: ABS, ROUND, CEIL, FLOOR, POWER, SQRT, etc.
* String Functions: CONCAT, LENGTH, SUBSTRING, UPPER, LOWER, TRIM, etc.
* Date and Time Functions: NOW, DATE, TIME, YEAR, MONTH, DAY, HOUR, MINUTE, SECOND, etc.
* Aggregate Functions: SUM, AVG, MIN, MAX, COUNT, etc.

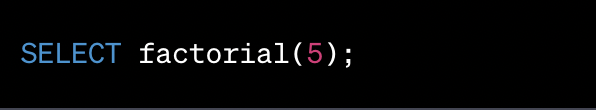
**User-defined Functions:** These functions are created by users and can be used to encapsulate specific logic or calculations. User-defined functions are defined using the CREATE FUNCTION statement and can be invoked like built-in functions. They can accept parameters and return a value. There are two types of user-defined functions:

* Scalar Functions: These functions return a single value and are used in expressions or SELECT statements.
* Table-Valued Functions: These functions return a table as a result and can be used in the FROM clause of a SELECT statement.

Here's **an example** of a user-defined scalar function that calculates the factorial of a given number:



Once the function is created, you can use it in a SQL query like this:



1. **Transaction Processing ( yet to be completed ):**

 Introduction to Transaction Processing, Transaction and System concepts, Desirable properties of Transactions, Characterizing schedules based on recoverability, Characterizing Schedules based on Serializability, Transaction support in SQL.

**Concurrency Control in Databases:** Two-phase locking techniques for Concurrency Control, Concurrency control based on Timestamp ordering, Multiversion Concurrency control techniques, Validation Concurrency control techniques, Granularity of Data items and Multiple Granularity Locking. Introduction to Database Recovery Protocols: Recovery Concepts, NO-UNDO/REDO recovery based on Deferred update, Recovery techniques based on the immediate update, Shadow paging, Database backup and recovery from catastrophic failures