SQL Triggers with example link - https://chatgpt.com/share/68ac13b0-9014-8006-a721-843b64ec2d6c

```
CREATE DATABASE sqltriggers;
\c sqltriggers;   -- (use \c in psql, not USE like MySQL)

-- Employees table
CREATE TABLE employees (
    emp_id INT PRIMARY KEY,
    name VARCHAR(100),
    department VARCHAR(100),
    salary NUMERIC(10,2)
);

-- Audit log table
CREATE TABLE employee_audit_log (
    audit_id SERIAL PRIMARY KEY,
    emp_id INT,
    action_type VARCHAR(10),        -- INSERT, UPDATE, DELETE
    old_name VARCHAR(100),
    new_name VARCHAR(100),
    old_department VARCHAR(100),
    new_department VARCHAR(100),
    old_salary NUMERIC(10,2),
    new_salary NUMERIC(10,2),
    salary_level VARCHAR(10),
    salary_change NUMERIC,
    action_time TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

2. AFTER INSERT Trigger

```
CREATE OR REPLACE FUNCTION log_employee_insert()
RETURNS TRIGGER AS $$
BEGIN
    INSERT INTO employee_audit_log (
        emp_id, action_type, new_name, new_department, new_salary,
salary_level
    ) VALUES (
        NEW.emp_id,
        'INSERT',
        NEW.name,
        NEW.department,
        NEW.salary,
        CASE
            WHEN NEW.salary < 3000 THEN 'Low'
            WHEN NEW.salary BETWEEN 3000 AND 7000 THEN
'Medium'
            ELSE 'High'
        END
    );
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER after_insert_employee
AFTER INSERT ON employees
FOR EACH ROW
EXECUTE FUNCTION log_employee_insert();
```

3. AFTER UPDATE Trigger

```
CREATE OR REPLACE FUNCTION log_employee_update()
RETURNS TRIGGER AS $$
BEGIN
    INSERT INTO employee_audit_log (
        emp_id, action_type, old_name, new_name,
        old_department, new_department,
        old_salary, new_salary,
        salary_change, salary_level
    ) VALUES (
        OLD.emp_id,
        'UPDATE',
        OLD.name, NEW.name,
        OLD.department, NEW.department,
        OLD.salary, NEW.salary,
        NEW.salary - OLD.salary,
        CASE
            WHEN NEW.salary < 3000 THEN 'Low'
            WHEN NEW.salary BETWEEN 3000 AND 7000 THEN
'Medium'
            ELSE 'High'
        END
    );
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER after_update_employee
AFTER UPDATE ON employees
FOR EACH ROW
EXECUTE FUNCTION log_employee_update();
```

4. AFTER DELETE Trigger

```
CREATE OR REPLACE FUNCTION log_employee_delete()
RETURNS TRIGGER AS $$
BEGIN
    INSERT INTO employee_audit_log (
        emp_id, action_type, old_name, old_department, old_salary
    ) VALUES (
        OLD.emp_id,
        'DELETE',
        OLD.name,
        OLD.department,
        OLD.salary
    );
    RETURN OLD;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER after_delete_employee
AFTER DELETE ON employees
FOR EACH ROW
EXECUTE FUNCTION log_employee_delete();
```

5. BEFORE Triggers (Validation Rules)

- **Prevent negative salary (BEFORE INSERT)**

```
CREATE OR REPLACE FUNCTION check_salary_before_insert()
RETURNS TRIGGER AS $$
BEGIN
   IF NEW.salary < 0 THEN
      RAISE EXCEPTION 'Salary cannot be negative';
   END IF;
   RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER before_insert_employee
BEFORE INSERT ON employees
FOR EACH ROW
EXECUTE FUNCTION check_salary_before_insert();
```

- **Prevent salary decrease (BEFORE UPDATE)**

```
CREATE OR REPLACE FUNCTION prevent_salary_decrease()
RETURNS TRIGGER AS $$
BEGIN
   IF NEW.salary < OLD.salary THEN
      RAISE EXCEPTION 'Salary cannot be decreased';
   END IF;
   RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER before_update_employee
BEFORE UPDATE ON employees
FOR EACH ROW
EXECUTE FUNCTION prevent_salary_decrease();
```

- **Prevent deletion of HR employees (BEFORE DELETE)**

```
CREATE OR REPLACE FUNCTION prevent_hr_delete()
RETURNS TRIGGER AS $$
BEGIN
   IF OLD.department = 'HR' THEN
      RAISE EXCEPTION 'Cannot delete HR department employees';
   END IF;
   RETURN OLD;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER before_delete_employee
BEFORE DELETE ON employees
FOR EACH ROW
EXECUTE FUNCTION prevent_hr_delete();
```

6. Testing

```
-- Insert
INSERT INTO employees VALUES (1, 'John Doe', 'IT', 5000);

-- Update
UPDATE employees SET salary = 6000 WHERE emp_id = 1;

-- Delete
DELETE FROM employees WHERE emp_id = 1;

-- View logs
SELECT * FROM employee_audit_log;
SELECT * FROM employees;
```

1. BEFORE INSERT → Prevent Negative Salary

```
CREATE OR REPLACE FUNCTION check_salary_before_insert()
RETURNS TRIGGER AS $$
BEGIN
   IF NEW.salary < 0 THEN
      RAISE EXCEPTION 'Salary cannot be negative';
   END IF;
   RETURN NEW;   -- allow insert if valid
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER before_insert_employee
BEFORE INSERT ON employees
FOR EACH ROW
EXECUTE FUNCTION check_salary_before_insert();
```

2. BEFORE UPDATE → Prevent Salary Decrease

```
CREATE OR REPLACE FUNCTION prevent_salary_decrease()
RETURNS TRIGGER AS $$
BEGIN
   IF NEW.salary < OLD.salary THEN
      RAISE EXCEPTION 'Salary cannot be decreased';
   END IF;
   RETURN NEW;   -- allow update if valid
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER before_update_employee
BEFORE UPDATE ON employees
FOR EACH ROW
EXECUTE FUNCTION prevent_salary_decrease();
```

3. BEFORE DELETE → Prevent Deletion of HR Employees

```
CREATE OR REPLACE FUNCTION prevent_hr_delete()
RETURNS TRIGGER AS $$
BEGIN
    IF OLD.department = 'HR' THEN
        RAISE EXCEPTION 'Cannot delete HR department employees';
    END IF;
    RETURN OLD;   -- allow delete if not HR
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER before_delete_employee
BEFORE DELETE ON employees
FOR EACH ROW
EXECUTE FUNCTION prevent_hr_delete();
```

✅ Testing the BEFORE Triggers

```
-- Test 1: Negative salary insert (should fail)
INSERT INTO employees VALUES (10, 'Test User', 'Sales', -2000);

-- Test 2: Decrease salary (should fail)
INSERT INTO employees VALUES (1, 'John Doe', 'IT', 5000);
UPDATE employees SET salary = 3000 WHERE emp_id = 1;

-- Test 3: Delete HR employee (should fail)
INSERT INTO employees VALUES (2, 'Alice', 'HR', 7000);
DELETE FROM employees WHERE emp_id = 2;

-- Successful insert
INSERT INTO employees VALUES (3, 'Bob', 'Finance', 8000);

-- Check results
SELECT * FROM employees;
```

Updated BEFORE Triggers with Logging
1. BEFORE INSERT → Block Negative Salary + Log

```
CREATE OR REPLACE FUNCTION check_salary_before_insert()
RETURNS TRIGGER AS $$
BEGIN
   IF NEW.salary < 0 THEN
      INSERT INTO employee_audit_log (
         emp_id, action_type, new_name, new_department, new_salary,
action_time
      ) VALUES (
         NEW.emp_id,
         'BLOCKED_INSERT',
         NEW.name,
         NEW.department,
         NEW.salary,
         CURRENT_TIMESTAMP
      );
      RAISE EXCEPTION 'Salary cannot be negative';
   END IF;
   RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER before_insert_employee
BEFORE INSERT ON employees
FOR EACH ROW
EXECUTE FUNCTION check_salary_before_insert();
```

2. BEFORE UPDATE → Block Salary Decrease + Log

```
CREATE OR REPLACE FUNCTION prevent_salary_decrease()
RETURNS TRIGGER AS $$
BEGIN
   IF NEW.salary < OLD.salary THEN
      INSERT INTO employee_audit_log (
         emp_id, action_type, old_salary, new_salary, salary_change,
action_time
      ) VALUES (
         OLD.emp_id,
         'BLOCKED_UPDATE',
         OLD.salary,
         NEW.salary,
         NEW.salary - OLD.salary,
         CURRENT_TIMESTAMP
      );
      RAISE EXCEPTION 'Salary cannot be decreased';
   END IF;
   RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER before_update_employee
BEFORE UPDATE ON employees
FOR EACH ROW
EXECUTE FUNCTION prevent_salary_decrease();
```

3. BEFORE DELETE → Block Deletion of HR Employees + Log

```
CREATE OR REPLACE FUNCTION prevent_hr_delete()
RETURNS TRIGGER AS $$
BEGIN
   IF OLD.department = 'HR' THEN
      INSERT INTO employee_audit_log (
         emp_id, action_type, old_name, old_department, old_salary,
action_time
      ) VALUES (
         OLD.emp_id,
         'BLOCKED_DELETE',
         OLD.name,
         OLD.department,
         OLD.salary,
         CURRENT_TIMESTAMP
      );
      RAISE EXCEPTION 'Cannot delete HR department employees';
   END IF;
   RETURN OLD;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER before_delete_employee
BEFORE DELETE ON employees
FOR EACH ROW
EXECUTE FUNCTION prevent_hr_delete();
```

Testing

```
-- Will log + fail
INSERT INTO employees VALUES (10, 'Test User', 'Sales', -2000);

-- Will log + fail
INSERT INTO employees VALUES (1, 'John Doe', 'IT', 5000);
UPDATE employees SET salary = 3000 WHERE emp_id = 1;

-- Will log + fail
INSERT INTO employees VALUES (2, 'Alice', 'HR', 7000);
DELETE FROM employees WHERE emp_id = 2;


-- Will succeed
INSERT INTO employees VALUES (3, 'Bob', 'Finance', 8000);

-- Check audit log (see blocked attempts too!)
SELECT * FROM employee_audit_log;

-- Check employees table
SELECT * FROM employees;
```

2. Updated BEFORE Triggers with Reason Logging

◈ BEFORE INSERT → Prevent Negative Salary

```
CREATE OR REPLACE FUNCTION check_salary_before_insert()
RETURNS TRIGGER AS $$
BEGIN
  IF NEW.salary < 0 THEN
    INSERT INTO employee_audit_log (
      emp_id, action_type, new_name, new_department, new_salary,
action_time, reason
    ) VALUES (
      NEW.emp_id,
      'BLOCKED_INSERT',
      NEW.name,
      NEW.department,
      NEW.salary,
      CURRENT_TIMESTAMP,
      'Salary cannot be negative'
    );
    RAISE EXCEPTION 'Salary cannot be negative';
  END IF;
  RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER before_insert_employee
BEFORE INSERT ON employees
FOR EACH ROW
EXECUTE FUNCTION check_salary_before_insert();
```

◈ BEFORE UPDATE → Prevent Salary Decrease

```
CREATE OR REPLACE FUNCTION prevent_salary_decrease()
RETURNS TRIGGER AS $$
BEGIN
  IF NEW.salary < OLD.salary THEN
    INSERT INTO employee_audit_log (
      emp_id, action_type, old_salary, new_salary, salary_change,
action_time, reason
    ) VALUES (
            OLD.emp_id,
            'BLOCKED_UPDATE',
            OLD.salary,
            NEW.salary,
            NEW.salary - OLD.salary,
            CURRENT_TIMESTAMP,
            'Salary decrease not allowed'
        );
        RAISE EXCEPTION 'Salary cannot be decreased';
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER before_update_employee
BEFORE UPDATE ON employees
FOR EACH ROW
EXECUTE FUNCTION prevent_salary_decrease();
```

� BEFORE DELETE → Prevent Deletion of HR Employees

```
CREATE OR REPLACE FUNCTION prevent_hr_delete()
RETURNS TRIGGER AS $$
BEGIN
    IF OLD.department = 'HR' THEN
        INSERT INTO employee_audit_log (
            emp_id, action_type, old_name, old_department, old_salary,
action_time, reason
        ) VALUES (
            OLD.emp_id,
            'BLOCKED_DELETE',
            OLD.name,
            OLD.department,
            OLD.salary,
            CURRENT_TIMESTAMP,
            'Cannot delete HR department employees'
        );
        RAISE EXCEPTION 'Cannot delete HR department employees';
    END IF;
    RETURN OLD;
END;
```

```
$$ LANGUAGE plpgsql;

CREATE TRIGGER before_delete_employee
BEFORE DELETE ON employees
FOR EACH ROW
EXECUTE FUNCTION prevent_hr_delete();
```

✅ **Test Again**

```
-- Will log + fail (reason: Salary < 0)
INSERT INTO employees VALUES (10, 'Test User', 'Sales', -2000);

-- Will log + fail (reason: Salary decrease not allowed)
INSERT INTO employees VALUES (1, 'John Doe', 'IT', 5000);
UPDATE employees SET salary = 3000 WHERE emp_id = 1;

-- Will log + fail (reason: Cannot delete HR employees)
INSERT INTO employees VALUES (2, 'Alice', 'HR', 7000);
DELETE FROM employees WHERE emp_id = 2;

-- Successful insert
INSERT INTO employees VALUES (3, 'Bob', 'Finance', 8000);

-- View logs (reasons included)
SELECT * FROM employee_audit_log;
```

BEFORE INSERT (validate salary)

```
CREATE OR REPLACE FUNCTION check_salary_before_insert()
RETURNS TRIGGER AS $$
BEGIN
   IF NEW.salary < 0 THEN
      RAISE EXCEPTION 'Salary cannot be negative';
   END IF;
   RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER before_insert_employee
BEFORE INSERT ON employees
FOR EACH ROW
EXECUTE FUNCTION check_salary_before_insert();
```

BEFORE UPDATE (block salary decrease)

```
CREATE OR REPLACE FUNCTION prevent_salary_decrease()
RETURNS TRIGGER AS $$
BEGIN
   IF NEW.salary < OLD.salary THEN
      RAISE EXCEPTION 'Salary cannot be decreased';
   END IF;
   RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER before_update_employee
BEFORE UPDATE ON employees
FOR EACH ROW
EXECUTE FUNCTION prevent_salary_decrease();
```

BEFORE DELETE (block HR deletions)

```
CREATE OR REPLACE FUNCTION prevent_hr_delete()
RETURNS TRIGGER AS $$
BEGIN
   IF OLD.department = 'HR' THEN
      RAISE EXCEPTION 'Cannot delete HR department employees';
   END IF;
```

```
    RETURN OLD;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER before_delete_employee
BEFORE DELETE ON employees
FOR EACH ROW
EXECUTE FUNCTION prevent_hr_delete();
```

✅ Key differences from MySQL:

- No DELIMITER $$ in PostgreSQL.
- Use SERIAL (or GENERATED ALWAYS AS IDENTITY) instead of AUTO_INCREMENT.
- Error handling uses RAISE EXCEPTION, not SIGNAL.
- Database switch is done with \c dbname (in psql), not USE dbname.

BEFORE INSERT → Prevent Negative Salary

```
-- Trigger to prevent inserting an employee with a negative salary
CREATE OR REPLACE FUNCTION check_salary_before_insert()
RETURNS TRIGGER AS $$
BEGIN
    IF NEW.salary < 0 THEN
        RAISE EXCEPTION 'Salary cannot be negative';
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER before_insert_employee
BEFORE INSERT ON employees
FOR EACH ROW
EXECUTE FUNCTION check_salary_before_insert();
```

� BEFORE UPDATE → Prevent Salary Decrease

```
-- Trigger to prevent updating an employee record with a lower salary than before
CREATE OR REPLACE FUNCTION prevent_salary_decrease()
RETURNS TRIGGER AS $$
BEGIN
   IF NEW.salary < OLD.salary THEN
      RAISE EXCEPTION 'Salary cannot be decreased';
   END IF;
   RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER before_update_employee
BEFORE UPDATE ON employees
FOR EACH ROW
EXECUTE FUNCTION prevent_salary_decrease();
```

� BEFORE DELETE → Prevent Deletion of HR Employees

```
-- Trigger to prevent deleting employees from the HR department
CREATE OR REPLACE FUNCTION prevent_hr_delete()
RETURNS TRIGGER AS $$
BEGIN
   IF OLD.department = 'HR' THEN
      RAISE EXCEPTION 'Cannot delete HR department employees';
   END IF;
   RETURN OLD;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER before_delete_employee
BEFORE DELETE ON employees
FOR EACH ROW
EXECUTE FUNCTION prevent_hr_delete();
```

1. Create a Sample Table

```
CREATE TABLE orders (
    order_id SERIAL PRIMARY KEY,
    customer_name VARCHAR(100),
    product_line VARCHAR(50),
    quantity INT,
    unit_price NUMERIC(10,2)
);
```

---

2. Insert Sample Data

```
INSERT INTO orders (customer_name, product_line, quantity,
unit_price) VALUES
('Alice', 'Electronics', 2, 400),
('Bob', 'Furniture', 1, 150),
('Charlie', 'Clothing', 5, 50),
('Diana', 'Electronics', 1, 800),
('Ethan', 'Clothing', 3, 120);
```

---

3. Use CASE in a Query

```
SELECT
    order_id,
    customer_name,
    product_line,
    quantity,
    unit_price,
    (quantity * unit_price) AS total,

    -- CASE example 1: Discount eligibility
    CASE
        WHEN (quantity * unit_price) > 500 THEN 'Eligible for Discount'
        ELSE 'Not Eligible'
    END AS discount_status,

    -- CASE example 2: Product category group
    CASE
        WHEN product_line = 'Electronics' THEN 'Tech'
        WHEN product_line = 'Furniture' THEN 'Home'
        WHEN product_line = 'Clothing' THEN 'Apparel'
        ELSE 'Other'
    END AS product_category,

    -- CASE example 3: Bulk vs Single order
    CASE
        WHEN quantity >= 5 THEN 'Bulk Order'
        WHEN quantity = 1 THEN 'Single Item'
        ELSE 'Standard Order'
    END AS order_type

FROM orders;
```