## 🎯 Learning Outcomes

By the end of this session, you will be able to:

- Understand the concept and purpose of triggers in PostgreSQL
- Create databases and tables for practical use
- Write trigger functions using PL/pgSQL
- Use NEW and OLD values effectively
- Differentiate between FOR EACH ROW and FOR EACH STATEMENT
- Implement real-world business rules using triggers
- Apply best practices for designing maintainable trigger systems

---

# 🧱 Step 1: Create and Connect Database

```
CREATE DATABASE company_db;
```

```
\c company_db;
```

---

# 📋 Step 2: Create Tables

## 1. Employees Table

```
CREATE TABLE employees (
    emp_id SERIAL PRIMARY KEY,
    name VARCHAR(100),
    salary NUMERIC,
    department VARCHAR(50)
);
```

## 2. Audit Log Table

```
CREATE TABLE emp_audit (
    audit_id SERIAL PRIMARY KEY,
    action_type VARCHAR(20),
    emp_id INT,
    old_salary NUMERIC,
    new_salary NUMERIC,
    changed_on TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

### 3. Statement Log Table

```
CREATE TABLE statement_log (
    log_id SERIAL PRIMARY KEY,
    message TEXT,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

---

## 🖍️ Step 3: Insert Sample Data

```
INSERT INTO employees (name, salary, department)
VALUES
('Ali', 50000, 'IT'),
('Sara', 60000, 'HR'),
('Ahmed', 55000, 'Finance');
```

---

## ⚡ Step 4: Trigger Examples (Real-World)

---

### 🔹 Example 1: BEFORE INSERT (Validation)

🎯 Scenario:

Prevent inserting employees with **negative salary**

Function:

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

Trigger:

```
CREATE TRIGGER before_insert_salary
BEFORE INSERT ON employees
```

```
FOR EACH ROW
EXECUTE FUNCTION validate_salary();
```

---

## 🔹 Example 2: AFTER UPDATE (Using OLD and NEW)

### 🎯 Scenario:

Track salary changes (audit system)

### Function:

```
CREATE OR REPLACE FUNCTION log_salary_update()
RETURNS TRIGGER AS $$
BEGIN
    INSERT INTO emp_audit(action_type, emp_id, old_salary, new_salary)
    VALUES ('UPDATE', OLD.emp_id, OLD.salary, NEW.salary);

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;
```

### Trigger:

```
CREATE TRIGGER after_salary_update
AFTER UPDATE ON employees
FOR EACH ROW
EXECUTE FUNCTION log_salary_update();
```

### Test:

```
UPDATE employees
SET salary = 70000
WHERE emp_id = 1;
```

---

## 🔹 Example 3: AFTER DELETE (Using OLD)

### 🎯 Scenario:

Keep record of deleted employees

### Function:

```
CREATE OR REPLACE FUNCTION log_employee_delete()
RETURNS TRIGGER AS $$
BEGIN
```

```
    INSERT INTO emp_audit(action_type, emp_id, old_salary)
    VALUES ('DELETE', OLD.emp_id, OLD.salary);

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

Trigger:

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

---

## 🔹 Example 4: BEFORE UPDATE (Using NEW)

### 🎯 Scenario:

Prevent salary from being reduced below 30000

```
CREATE OR REPLACE FUNCTION prevent_low_salary()
RETURNS TRIGGER AS $$
BEGIN
    IF NEW.salary < 30000 THEN
        RAISE EXCEPTION 'Salary cannot be less than 30000';
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;
```

```
CREATE TRIGGER before_salary_update
BEFORE UPDATE ON employees
FOR EACH ROW
EXECUTE FUNCTION prevent_low_salary();
```

---

## 🔹 Example 5: FOR EACH STATEMENT Trigger

### 🎯 Scenario:

Log bulk updates (runs once per query)

Function:

```
CREATE OR REPLACE FUNCTION log_bulk_operation()
RETURNS TRIGGER AS $$
```

```
BEGIN
    INSERT INTO statement_log(message)
    VALUES ('Bulk update executed on employees table');

    RETURN NULL;
END;
$$ LANGUAGE plpgsql;
```

Trigger:

```
CREATE TRIGGER after_bulk_update
AFTER UPDATE ON employees
FOR EACH STATEMENT
EXECUTE FUNCTION log_bulk_operation();
```

Test:

```
UPDATE employees
SET salary = salary + 1000;
```

👉 This will:

- Update all rows
- Trigger audit (row-level) multiple times
- Trigger statement log only once

---

# 🔍 Key Concepts

| Concept | Meaning |
|---|---|
| `NEW` | New values after INSERT/UPDATE |
| `OLD` | Previous values before UPDATE/DELETE |
| `FOR EACH ROW` | Runs for every affected row |
| `FOR EACH STATEMENT` | Runs once per SQL statement |

---

# 🧠 Practice Tasks

1. Create a trigger:
    - Automatically assign department = 'General' if NULL
2. Create a trigger:

- Log employee name changes

3. Create a trigger:
   - Prevent deletion of employees from "HR" department
4. Modify audit table:
   - Add column for user (`CURRENT_USER`)

---

# 🏁 Mini Project (Real-World)

## 💳 Bank System

Table:

```
CREATE TABLE accounts (
    acc_id SERIAL PRIMARY KEY,
    name VARCHAR(100),
    balance NUMERIC
);
```

Task:

- Create trigger to:
  - ❌ Prevent withdrawal if balance < 0
  - ✅ Log every transaction

---