✅ 1. What is a Trigger?

A **trigger** is a special database object that automatically runs **before or after** an event on a table.

✓ A trigger runs automatically when:

- A row is **INSERTED**
- A row is **UPDATED**
- A row is **DELETED**

✓ Why triggers are used?

- Auto-update timestamps
- Maintain logs
- Enforce business rules
- Automatically insert related data
- Validate data before saving
- Send notifications (via functions)

✨ 2. How Triggers Work (Easy Explanation)

A trigger has **two parts**:

1️⃣ Trigger Function

A PL/pgSQL function containing the logic.

2️⃣ Trigger

Defines *when* and *on which table* the function should run.

⭐ 3. Types of Triggers

🔵 By Timing

Type	Runs When
BEFORE	Before the row is inserted/updated/deleted
AFTER	After the row action completes

🟢 By Event

Event	Meaning
INSERT	When new row is added

Event	Meaning
UPDATE	When existing row changes
DELETE	When row is removed

🟣 Row-level vs Statement-level

Type	Meaning
ROW	Runs once per affected row
STATEMENT	Runs once per SQL statement

⭐ 4. Creating a Simple Trigger Example (dvdrental)

🎯 Example 1: Auto-update last_update column before updating a film

Step 1 — Create Trigger Function

```
CREATE OR REPLACE FUNCTION update_film_timestamp()
RETURNS TRIGGER
LANGUAGE plpgsql
AS $$
BEGIN
    NEW.last_update := NOW(); -- change the last_update column
    RETURN NEW;
END;
$$;
```

Step 2 — Create Trigger

```
CREATE TRIGGER trg_update_film_timestamp
BEFORE UPDATE ON film
FOR EACH ROW
EXECUTE FUNCTION update_film_timestamp();
```

✓ Result:

Whenever you update a film's title, rating, rental rate, etc., the **last_update** column automatically updates.

⭐ 5. Trigger for Logging Changes (Audit Log)

🎯 Example 2: Log deleted rentals into a log table

Step 1 — Create a log table

```
CREATE TABLE rental_log (
    rental_id INT,
    customer_id INT,
    rental_date TIMESTAMP,
    action_time TIMESTAMP DEFAULT NOW()
);
```

Step 2 — Create Trigger Function

```
CREATE OR REPLACE FUNCTION log_rental_delete()
RETURNS TRIGGER
LANGUAGE plpgsql
AS $$
BEGIN
    INSERT INTO rental_log(rental_id, customer_id, rental_date)
    VALUES (OLD.rental_id, OLD.customer_id, OLD.rental_date);

    RETURN OLD;
END;
$$;
```

Step 3 — Create Trigger

```
CREATE TRIGGER trg_log_rental_delete
AFTER DELETE ON rental
FOR EACH ROW
EXECUTE FUNCTION log_rental_delete();
```

✓ Result:

Whenever a rental row is deleted, it is recorded in **rental_log**.

⭐ 6. Trigger to Prevent Invalid Data

🎯 Example 3: Stop inserting customers without email

Step 1 — Create Function

```
CREATE OR REPLACE FUNCTION validate_customer_email()
RETURNS TRIGGER
LANGUAGE plpgsql
AS $$
BEGIN
```

```
    IF NEW.email IS NULL OR NEW.email = '' THEN
        RAISE EXCEPTION 'Email cannot be empty!';
    END IF;

    RETURN NEW;
END;
$$;
```

Step 2 — Create Trigger

```
CREATE TRIGGER trg_validate_customer_email
BEFORE INSERT ON customer
FOR EACH ROW
EXECUTE FUNCTION validate_customer_email();
```

✓ Result:

The database prevents incomplete customer records.

⭐ 7. Trigger After Insert (Notification Example)

🎯 Example 4: Print a message when a new customer is added

Function

```
CREATE OR REPLACE FUNCTION notify_new_customer()
RETURNS TRIGGER
LANGUAGE plpgsql
AS $$
BEGIN
    RAISE NOTICE 'New customer added: % %', NEW.first_name, NEW.last_name;
    RETURN NEW;
END;
$$;
```

Trigger

```
CREATE TRIGGER trg_notify_new_customer
AFTER INSERT ON customer
FOR EACH ROW
EXECUTE FUNCTION notify_new_customer();
```

⭐ 8. Update Trigger (Example on Payment Table)

🎯 Example 5: Prevent payment amount less than 0

```
CREATE OR REPLACE FUNCTION validate_payment_amount()
RETURNS TRIGGER
LANGUAGE plpgsql
AS $$
BEGIN
    IF NEW.amount <= 0 THEN
        RAISE EXCEPTION 'Payment amount must be greater than zero';
    END IF;
    RETURN NEW;
END;
$$;

CREATE TRIGGER trg_validate_payment_amount
BEFORE INSERT OR UPDATE ON payment
FOR EACH ROW
EXECUTE FUNCTION validate_payment_amount();
```

🎓 Summary

Concept	Meaning
Trigger	Auto-runs on INSERT/UPDATE/DELETE
Trigger Function	Contains logic in PL/pgSQL
BEFORE Trigger	Validates/changes data before save
AFTER Trigger	Used for logs, history, messages
NEW	The new row after insert/update
OLD	The old row before update/delete
FOR EACH ROW	Runs once per affected row
