# **Buku Pembelajaran: Triggers**

*Advanced Database - Pertemuan 5*

## **Daftar Isi**

1. [BEFORE/AFTER Triggers](https://markdownlivepreview.com/#1-beforeafter-triggers)
2. [NEW/OLD Records dalam PostgreSQL](https://markdownlivepreview.com/#2-newold-records-dalam-postgresql)
3. [Trigger Functions](https://markdownlivepreview.com/#3-trigger-functions)

## **Pengantar: Database yang Reaktif dan Intelligent**

Database triggers seperti sistem keamanan rumah yang otomatis - ketika ada perubahan data, triggers bereaksi: validasi, logging, update related tables, atau kirim notifikasi.

Triggers mengubah database dari storage pasif menjadi sistem intelligent yang proaktif. Kita akan menguasai triggers dari konsep timing hingga audit trail professional.

## **1. BEFORE/AFTER Triggers**

### **Mengapa Trigger Timing Penting?**

Seperti memasak - ada bumbu yang ditambah sebelum memasak (meresap) dan setelah masak (aroma). Database triggers punya timing yang sama pentingnya.

BEFORE triggers bekerja sebelum data berubah - untuk validasi dan transformasi. AFTER triggers bekerja setelah data berubah - untuk logging dan cascade operations.

### **BEFORE Triggers: Prevention dan Preparation**

BEFORE triggers seperti security guard yang memeriksa sebelum masuk gedung. Functions:

* Validasi data sebelum disimpan
* Transformasi data format
* Auto-generate timestamps dan defaults

-- BEFORE trigger untuk validasi dan transformasi

CREATE OR REPLACE FUNCTION validate\_student\_data()

RETURNS TRIGGER AS $$

BEGIN

-- Validasi NIM format

IF NEW.nim !~ '^[0-9]{7}$' THEN

RAISE EXCEPTION 'NIM harus 7 digit angka';

END IF;

-- Transformasi data

NEW.nama := INITCAP(LOWER(NEW.nama));

NEW.email := COALESCE(NEW.email, NEW.nim || '@university.edu');

NEW.created\_at := CURRENT\_TIMESTAMP;

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

CREATE TRIGGER trg\_validate\_student

BEFORE INSERT OR UPDATE ON students

FOR EACH ROW EXECUTE FUNCTION validate\_student\_data();

### **AFTER Triggers: Reaction dan Recording**

AFTER triggers seperti reporter yang mendokumentasi kejadian. Functions:

* Audit logging - mencatat siapa mengubah apa dan kapan
* Cascade operations - update related tables
* Notifications - kirim notifikasi

-- AFTER trigger untuk audit logging

CREATE OR REPLACE FUNCTION audit\_student\_changes()

RETURNS TRIGGER AS $$

BEGIN

INSERT INTO audit\_log (

table\_name, operation, record\_id,

old\_values, new\_values, changed\_by

) VALUES (

'students', TG\_OP, COALESCE(NEW.id, OLD.id),

CASE WHEN TG\_OP = 'DELETE' THEN to\_jsonb(OLD) ELSE NULL END,

CASE WHEN TG\_OP = 'DELETE' THEN NULL ELSE to\_jsonb(NEW) END,

current\_user

);

RETURN COALESCE(NEW, OLD);

END;

$$ LANGUAGE plpgsql;

CREATE TRIGGER trg\_audit\_student

AFTER INSERT OR UPDATE OR DELETE ON students

FOR EACH ROW EXECUTE FUNCTION audit\_student\_changes();

### **Trigger Event Types**

INSERT Triggers - Setup default values, validation UPDATE Triggers - Track modifications, prevent invalid changes DELETE Triggers - Soft delete implementation, cascade cleanup TRUNCATE Triggers - Backup before mass deletion

-- INSERT: Setup defaults

CREATE OR REPLACE FUNCTION setup\_defaults()

RETURNS TRIGGER AS $$

BEGIN

NEW.status := COALESCE(NEW.status, 'ACTIVE');

NEW.created\_at := CURRENT\_TIMESTAMP;

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

-- UPDATE: Track changes

CREATE OR REPLACE FUNCTION track\_updates()

RETURNS TRIGGER AS $$

BEGIN

IF OLD.nim != NEW.nim THEN

RAISE EXCEPTION 'NIM cannot be changed';

END IF;

NEW.updated\_at := CURRENT\_TIMESTAMP;

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

-- DELETE: Soft delete

CREATE OR REPLACE FUNCTION soft\_delete()

RETURNS TRIGGER AS $$

BEGIN

UPDATE students SET status = 'DELETED', deleted\_at = CURRENT\_TIMESTAMP

WHERE id = OLD.id;

RETURN NULL; -- Cancel hard delete

END;

$$ LANGUAGE plpgsql;

Tahukah Anda? TRUNCATE triggers hanya bisa BEFORE/AFTER (tidak FOR EACH ROW), karena TRUNCATE menghapus semua rows sekaligus.

### **Performance Considerations**

-- ❌ SALAH: Nested loops lambat

FOR rec IN SELECT \* FROM table LOOP

UPDATE other\_table SET count = count + 1 WHERE id = rec.id;

END LOOP;

-- ✅ BENAR: Set-based operation efisien

UPDATE other\_table

SET count = count + 1

WHERE id IN (SELECT related\_id FROM related\_table WHERE student\_id = NEW.id);

### **Rangkuman Key Points**

* BEFORE triggers untuk validasi, transformasi, prevention
* AFTER triggers untuk logging, cascade operations, notifications
* Event types: INSERT, UPDATE, DELETE, TRUNCATE berbeda behavior
* Performance: Hindari nested loops, gunakan set-based operations
* TRUNCATE triggers bekerja table level, bukan row level

## **2. NEW/OLD Records dalam PostgreSQL**

### **Mengapa NEW/OLD Records Penting?**

Seperti editor yang mereview artikel - butuh versi lama (OLD) untuk tahu perubahan, dan versi baru (NEW) untuk hasil akhir. NEW/OLD records memberikan akses ke "before and after" snapshots data.

Seperti "track changes" di Microsoft Word, NEW/OLD memungkinkan triggers membandingkan, memvalidasi, dan bereaksi terhadap perubahan spesifik.

### **Konsep NEW Record**

NEW record berisi nilai setelah operasi DML. Available untuk INSERT dan UPDATE.

#### **NEW dalam INSERT dan UPDATE**

-- INSERT: NEW berisi data yang akan disimpan

CREATE OR REPLACE FUNCTION log\_new\_student()

RETURNS TRIGGER AS $$

BEGIN

RAISE NOTICE 'New student: % (NIM: %)', NEW.nama, NEW.nim;

INSERT INTO registration\_log (student\_nim, student\_name, registration\_date)

VALUES (NEW.nim, NEW.nama, NEW.created\_at);

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

-- UPDATE: NEW = nilai baru, OLD = nilai lama

CREATE OR REPLACE FUNCTION track\_gpa\_changes()

RETURNS TRIGGER AS $$

BEGIN

IF NEW.gpa IS DISTINCT FROM OLD.gpa THEN

INSERT INTO gpa\_history (student\_nim, old\_gpa, new\_gpa, change\_date)

VALUES (NEW.nim, OLD.gpa, NEW.gpa, CURRENT\_TIMESTAMP);

-- Log significant changes

IF ABS(NEW.gpa - OLD.gpa) > 0.5 THEN

RAISE NOTICE 'Major GPA change: % -> %', OLD.gpa, NEW.gpa;

END IF;

END IF;

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

### **Konsep OLD Record**

OLD record berisi nilai sebelum operasi. Available untuk UPDATE dan DELETE.

-- DELETE: OLD berisi data yang akan dihapus (NEW tidak available)

CREATE OR REPLACE FUNCTION archive\_deleted\_student()

RETURNS TRIGGER AS $$

BEGIN

INSERT INTO students\_archive (original\_id, nim, nama, gpa, deleted\_at)

VALUES (OLD.id, OLD.nim, OLD.nama, OLD.gpa, CURRENT\_TIMESTAMP);

RAISE NOTICE 'Student archived: %', OLD.nama;

RETURN OLD;

END;

$$ LANGUAGE plpgsql;

### **Conditional Processing berdasarkan Changes**

-- Smart processing hanya untuk field yang berubah

CREATE OR REPLACE FUNCTION smart\_notification\_trigger()

RETURNS TRIGGER AS $$

BEGIN

-- GPA significant change

IF TG\_OP = 'UPDATE' AND OLD.gpa IS DISTINCT FROM NEW.gpa THEN

IF ABS(OLD.gpa - NEW.gpa) >= 0.5 THEN

INSERT INTO notifications (student\_id, message, priority)

VALUES (NEW.id, format('GPA changed: %s → %s', OLD.gpa, NEW.gpa), 'HIGH');

END IF;

END IF;

-- Status change tracking

IF TG\_OP = 'UPDATE' AND OLD.status IS DISTINCT FROM NEW.status THEN

INSERT INTO status\_history (student\_id, old\_status, new\_status, change\_date)

VALUES (NEW.id, OLD.status, NEW.status, CURRENT\_TIMESTAMP);

END IF;

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

### **Rangkuman Key Points**

* NEW record berisi nilai after operation (INSERT, UPDATE)
* OLD record berisi nilai before operation (UPDATE, DELETE)
* Field-level access dengan NEW.column\_name dan OLD.column\_name
* NULL handling gunakan IS DISTINCT FROM untuk comparison
* Record variables untuk complex data processing
* Conditional processing berdasarkan field changes untuk efficiency

## **3. Trigger Functions**

### **Mengapa Trigger Functions Berbeda?**

Trigger functions seperti "special agents" - punya akses khusus (NEW, OLD, TG\_OP) dan aturan khusus (RETURNS TRIGGER). Berbeda dengan function biasa, trigger functions hanya dipanggil otomatis saat database events.

Seperti alarm mobil otomatis - trigger functions adalah "automated responders" yang bereaksi terhadap perubahan data.

### **CREATE FUNCTION untuk Trigger Functions**

-- Basic trigger function structure

CREATE OR REPLACE FUNCTION nama\_trigger\_function()

RETURNS TRIGGER AS $$

DECLARE

variable\_name data\_type;

BEGIN

-- Trigger logic here

RETURN NEW; -- Must return NEW, OLD, or NULL

END;

$$ LANGUAGE plpgsql;

### **RETURNS TRIGGER dan Return Values**

-- ✅ BENAR: Trigger function

CREATE OR REPLACE FUNCTION student\_audit()

RETURNS TRIGGER AS $$ -- WAJIB RETURNS TRIGGER

BEGIN

INSERT INTO audit\_log (operation, table\_name, record\_id)

VALUES (TG\_OP, TG\_TABLE\_NAME, COALESCE(NEW.id, OLD.id));

RETURN COALESCE(NEW, OLD); -- WAJIB return NEW/OLD/NULL

END;

$$ LANGUAGE plpgsql;

-- ❌ SALAH: Function biasa RETURNS INTEGER tidak bisa jadi trigger

Return Values:

* RETURN NEW - Proceed dengan changes (untuk INSERT/UPDATE)
* RETURN OLD - Keep original data (untuk DELETE)
* RETURN NULL - Cancel operation completely

-- Example: Conditional processing

CREATE OR REPLACE FUNCTION smart\_trigger()

RETURNS TRIGGER AS $$

BEGIN

-- Validation

IF TG\_OP IN ('INSERT', 'UPDATE') AND (NEW.gpa < 0 OR NEW.gpa > 4.0) THEN

RAISE EXCEPTION 'Invalid GPA: %', NEW.gpa;

END IF;

-- Transform data

IF TG\_OP IN ('INSERT', 'UPDATE') THEN

NEW.email := LOWER(TRIM(NEW.email));

RETURN NEW;

END IF;

-- For DELETE

IF OLD.protected = TRUE THEN

RETURN NULL; -- Cancel delete

END IF;

RETURN OLD;

END;

$$ LANGUAGE plpgsql;

### **TG\_\* Variables: Trigger Context Information**

TG\_OP - Operation type (INSERT/UPDATE/DELETE/TRUNCATE)  
TG\_WHEN - Timing (BEFORE/AFTER)  
TG\_TABLE\_NAME - Source table name

-- Universal trigger menggunakan TG\_\* variables

CREATE OR REPLACE FUNCTION universal\_trigger()

RETURNS TRIGGER AS $$

BEGIN

-- Handle different operations dengan TG\_OP

CASE TG\_OP

WHEN 'INSERT' THEN

INSERT INTO audit\_log (operation, table\_name, new\_data)

VALUES ('CREATE', TG\_TABLE\_NAME, to\_jsonb(NEW));

RETURN NEW;

WHEN 'UPDATE' THEN

INSERT INTO audit\_log (operation, table\_name, old\_data, new\_data)

VALUES ('MODIFY', TG\_TABLE\_NAME, to\_jsonb(OLD), to\_jsonb(NEW));

RETURN NEW;

WHEN 'DELETE' THEN

INSERT INTO audit\_log (operation, table\_name, old\_data)

VALUES ('REMOVE', TG\_TABLE\_NAME, to\_jsonb(OLD));

RETURN OLD;

END CASE;

RETURN NULL;

END;

$$ LANGUAGE plpgsql;

-- Timing-aware trigger dengan TG\_WHEN

CREATE OR REPLACE FUNCTION timing\_trigger()

RETURNS TRIGGER AS $$

BEGIN

IF TG\_WHEN = 'BEFORE' THEN

-- Pre-processing: validation, transformation

NEW.updated\_at := CURRENT\_TIMESTAMP;

RETURN NEW;

ELSE

-- Post-processing: logging, notifications

INSERT INTO change\_log (table\_name, operation, timestamp)

VALUES (TG\_TABLE\_NAME, TG\_OP, CURRENT\_TIMESTAMP);

RETURN COALESCE(NEW, OLD);

END IF;

END;

$$ LANGUAGE plpgsql;

### **Multiple Triggers dan Reusability**

Trigger Execution Order: Alphabetical by trigger name

-- Gunakan prefix numerik untuk control order

CREATE TRIGGER trg\_01\_validate

BEFORE INSERT OR UPDATE ON students

FOR EACH ROW EXECUTE FUNCTION validate\_data();

CREATE TRIGGER trg\_02\_enrich

BEFORE INSERT OR UPDATE ON students

FOR EACH ROW EXECUTE FUNCTION enrich\_data();

CREATE TRIGGER trg\_03\_audit

AFTER INSERT OR UPDATE OR DELETE ON students

FOR EACH ROW EXECUTE FUNCTION audit\_changes();

Tahukah Anda? Triggers execute dalam alphabetical order. Gunakan naming convention dengan prefix numerik (01\_, 02\_) untuk control execution order.