**Pre-Test: Triggers — Advanced Database Systems**

**Nama:** Raihan Muhammad Riswandi

**NIM:** 24110500001

Bagian A — Jawaban Pilihan Ganda

* BEFORE trigger dieksekusi sebelum operasi DML, AFTER trigger sesudah operasi.
* Hanya OLD.
* NEW, OLD, atau NULL.
* Untuk validasi dan transformasi data sebelum disimpan.
* Operasi DML dibatalkan.
* Jenis operasi (INSERT/UPDATE/DELETE/TRUNCATE).

**Bagian B — Implementasi Praktis (SQL / PostgreSQL)**

Skrip berikut dibuat untuk PostgreSQL. Jalankan dengan hak yang cukup (CREATE FUNCTION, CREATE TRIGGER, INSERT). Pastikan students dan audit\_log

*1) Trigger function: validate\_student\_data() (BEFORE INSERT OR UPDATE)*

**CREATE** **OR** **REPLACE** **FUNCTION** validate\_student\_data()  
RETURNS **trigger** **AS** $$  
**BEGIN**  
 *-- Validasi NIM: harus 7 digit angka*  
 **IF** **NEW**.nim **IS** **NULL** **OR** **NOT** (**NEW**.nim ~ '^\d{7}$') **THEN**  
 RAISE **EXCEPTION** 'Invalid NIM: must be 7 digits. Given: %', **NEW**.nim;  
 **END** **IF**;  
  
 *-- Validasi email sederhana*  
 **IF** **NEW**.email **IS** **NULL** **OR** **NOT** (**NEW**.email ~ '^[A-Za-z0-9.\_%+-]+@[A-Za-z0-9.-]+\.[A-Za-z]{2,}$') **THEN**  
 RAISE **EXCEPTION** 'Invalid email format: %', **NEW**.email;  
 **END** **IF**;  
  
 *-- Set updated\_at otomatis*  
 **NEW**.updated\_at := NOW();  
  
 **RETURN** **NEW**;  
**EXCEPTION**  
 **WHEN** others **THEN**  
 *-- Propagate clear error*  
 RAISE;  
**END**;  
$$ LANGUAGE plpgsql;  
  
*-- Attach trigger to students*  
**CREATE** **TRIGGER** students\_validate\_before  
**BEFORE** **INSERT** **OR** **UPDATE** **ON** students  
**FOR** **EACH** **ROW**  
**EXECUTE** **FUNCTION** validate\_student\_data();

*2) Trigger function: audit\_student\_changes() (AFTER INSERT/UPDATE/DELETE)*

**CREATE** **OR** **REPLACE** **FUNCTION** audit\_student\_changes()  
RETURNS **trigger** **AS** $$  
**DECLARE**  
 v\_old jsonb;  
 v\_new jsonb;  
 v\_record\_id integer;  
**BEGIN**  
 **IF** TG\_OP = 'INSERT' **THEN**  
 v\_old := **NULL**;  
 v\_new := to\_jsonb(**NEW**);  
 v\_record\_id := **NEW**.**id**;  
 **ELSIF** TG\_OP = 'UPDATE' **THEN**  
 v\_old := to\_jsonb(**OLD**);  
 v\_new := to\_jsonb(**NEW**);  
 v\_record\_id := **NEW**.**id**;  
 **ELSIF** TG\_OP = 'DELETE' **THEN**  
 v\_old := to\_jsonb(**OLD**);  
 v\_new := **NULL**;  
 v\_record\_id := **OLD**.**id**;  
 **ELSE**  
 *-- safety*  
 v\_old := **NULL**;  
 v\_new := **NULL**;  
 v\_record\_id := **NULL**;  
 **END** **IF**;  
  
 **INSERT** **INTO** audit\_log(table\_name, record\_id, operation, old\_values, new\_values, changed\_at, changed\_by)  
 **VALUES**('students', v\_record\_id, TG\_OP, v\_old, v\_new, NOW(), current\_user);  
  
 **RETURN** **NULL**; *-- AFTER trigger*  
**EXCEPTION**  
 **WHEN** others **THEN**  
 *-- Log atau re-raise untuk memastikan audit tidak tumpang tindih*  
 RAISE;  
**END**;  
$$ LANGUAGE plpgsql;  
  
*-- Attach trigger*  
**CREATE** **TRIGGER** students\_audit\_after  
**AFTER** **INSERT** **OR** **UPDATE** **OR** **DELETE** **ON** students  
**FOR** **EACH** **ROW**  
**EXECUTE** **FUNCTION** audit\_student\_changes();

**Bagian C — Problem Solving: Student Data Update Trigger**

Additional tables (as provided)

*-- student\_stats*  
**CREATE** **TABLE** **IF** **NOT** **EXISTS** student\_stats (  
 student\_id INTEGER **PRIMARY** **KEY** **REFERENCES** students(**id**),  
 total\_courses INTEGER **DEFAULT** 0,  
 avg\_grade DECIMAL(3,2) **DEFAULT** 0,  
 last\_updated TIMESTAMP **DEFAULT** CURRENT\_TIMESTAMP  
);  
  
*-- grades (if not already created in environment)*  
**CREATE** **TABLE** **IF** **NOT** **EXISTS** grades (  
 **id** SERIAL **PRIMARY** **KEY**,  
 student\_id INTEGER **REFERENCES** students(**id**),  
 course\_code VARCHAR(10),  
 grade DECIMAL(3,2) **CHECK** (grade >= 0 **AND** grade <= 4.0),  
 created\_at TIMESTAMP **DEFAULT** CURRENT\_TIMESTAMP  
);

Trigger function: update\_student\_statistics()

Fungsi ini akan menangani INSERT, UPDATE, DELETE pada grades.

**CREATE** **OR** **REPLACE** **FUNCTION** update\_student\_statistics()  
RETURNS **trigger** **AS** $$  
**DECLARE**  
 v\_student\_old integer;  
 v\_student\_new integer;  
 v\_total integer;  
 v\_avg numeric(3,2);  
**BEGIN**  
 *-- Tentukan student yang perlu di-recalc*  
 **IF** TG\_OP = 'INSERT' **THEN**  
 v\_student\_new := **NEW**.student\_id;  
 *-- recalc for NEW.student\_id*  
 **SELECT** COALESCE(COUNT(\*),0), COALESCE(AVG(grade),0)  
 **INTO** v\_total, v\_avg  
 **FROM** grades  
 **WHERE** student\_id = v\_student\_new;  
  
 **INSERT** **INTO** student\_stats(student\_id, total\_courses, avg\_grade, last\_updated)  
 **VALUES** (v\_student\_new, v\_total, ROUND(v\_avg::numeric,2), NOW())  
 **ON** CONFLICT (student\_id) DO **UPDATE**  
 **SET** total\_courses = EXCLUDED.total\_courses,  
 avg\_grade = EXCLUDED.avg\_grade,  
 last\_updated = NOW();  
  
 **ELSIF** TG\_OP = 'UPDATE' **THEN**  
 v\_student\_old := **OLD**.student\_id;  
 v\_student\_new := **NEW**.student\_id;  
  
 *-- If student\_id changed, recalc for both*  
 **IF** v\_student\_old **IS** **NOT** **NULL** **AND** v\_student\_old <> v\_student\_new **THEN**  
 **SELECT** COALESCE(COUNT(\*),0), COALESCE(AVG(grade),0)  
 **INTO** v\_total, v\_avg  
 **FROM** grades  
 **WHERE** student\_id = v\_student\_old;  
  
 **INSERT** **INTO** student\_stats(student\_id, total\_courses, avg\_grade, last\_updated)  
 **VALUES** (v\_student\_old, v\_total, ROUND(v\_avg::numeric,2), NOW())  
 **ON** CONFLICT (student\_id) DO **UPDATE**  
 **SET** total\_courses = EXCLUDED.total\_courses,  
 avg\_grade = EXCLUDED.avg\_grade,  
 last\_updated = NOW();  
 **END** **IF**;  
  
 *-- Recalc for new student*  
 **SELECT** COALESCE(COUNT(\*),0), COALESCE(AVG(grade),0)  
 **INTO** v\_total, v\_avg  
 **FROM** grades  
 **WHERE** student\_id = v\_student\_new;  
  
 **INSERT** **INTO** student\_stats(student\_id, total\_courses, avg\_grade, last\_updated)  
 **VALUES** (v\_student\_new, v\_total, ROUND(v\_avg::numeric,2), NOW())  
 **ON** CONFLICT (student\_id) DO **UPDATE**  
 **SET** total\_courses = EXCLUDED.total\_courses,  
 avg\_grade = EXCLUDED.avg\_grade,  
 last\_updated = NOW();  
  
 **ELSIF** TG\_OP = 'DELETE' **THEN**  
 v\_student\_old := **OLD**.student\_id;  
  
 **SELECT** COALESCE(COUNT(\*),0), COALESCE(AVG(grade),0)  
 **INTO** v\_total, v\_avg  
 **FROM** grades  
 **WHERE** student\_id = v\_student\_old;  
  
 **IF** v\_total = 0 **THEN**  
 *-- No grades left: set defaults*  
 **INSERT** **INTO** student\_stats(student\_id, total\_courses, avg\_grade, last\_updated)  
 **VALUES** (v\_student\_old, 0, 0, NOW())  
 **ON** CONFLICT (student\_id) DO **UPDATE**  
 **SET** total\_courses = 0,  
 avg\_grade = 0,  
 last\_updated = NOW();  
 **ELSE**  
 **INSERT** **INTO** student\_stats(student\_id, total\_courses, avg\_grade, last\_updated)  
 **VALUES** (v\_student\_old, v\_total, ROUND(v\_avg::numeric,2), NOW())  
 **ON** CONFLICT (student\_id) DO **UPDATE**  
 **SET** total\_courses = EXCLUDED.total\_courses,  
 avg\_grade = EXCLUDED.avg\_grade,  
 last\_updated = NOW();  
 **END** **IF**;  
 **END** **IF**;  
  
 **RETURN** **NULL**; *-- AFTER trigger*  
**EXCEPTION**  
 **WHEN** others **THEN**  
 RAISE **EXCEPTION** 'Error updating student statistics: %', SQLERRM;  
**END**;  
$$ LANGUAGE plpgsql;  
  
*-- Attach trigger to grades*  
**CREATE** **TRIGGER** grades\_update\_stats\_after  
**AFTER** **INSERT** **OR** **UPDATE** **OR** **DELETE** **ON** grades  
**FOR** **EACH** **ROW**  
**EXECUTE** **FUNCTION** update\_student\_statistics();

**Penjelasan singkat tentang penanganan edge cases**

* **First-time grades**: INSERT ... ON CONFLICT membuat record student\_stats jika belum ada.
* **Grades deleted resulting zero rows**: ketika COUNT = 0, avg\_grade diset 0 dan total\_courses = 0.
* **UPDATE dengan perubahan student\_id**: fungsi menghitung ulang statistik untuk OLD.student\_id dan NEW.student\_id agar konsistensi terjaga.
* **Error handling**: semua exception di-catch dan dilempar ulang sebagai RAISE EXCEPTION agar transaksi dapat dibatalkan jika terjadi masalah integritas.