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.