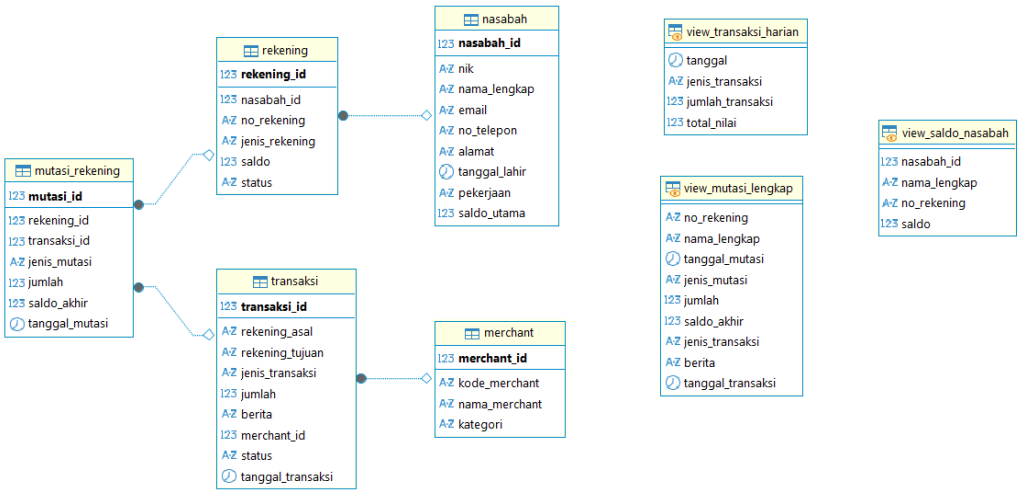
Nama : Mochammad Tanggaq Dirat Saputra
Kelas : SIB 2D
NIM : 244107060126
Absen : 08
Mata Kuliah : Basis Data Lanjut
Program Studi : D4 – Sistem Informasi Bisnis
Semester : 3 (tiga)
Pertemuan ke- : 7

JOBSHEET 07

Transaction & Concurrency

Praktikum 00 – Menyiapkan file Studi Kasus (db_mobile_banking)

1. Download file `db_mobile_banking.sql` yang berisi **data dummy** pada contoh desain basis data transaksi mobile banking.
2. Buka aplikasi dBeaver, lalu buat database baru bernama `db_mobile_banking`
3. Kemudian klik kanan `Tools  Execute Script `
4. Kemudian arahkan ke file db_akademik.sql yang sudah ada
5. Selanjutnya klik `Next  Start`
6. Jika sudah selesai, cek diagram ERD melalui klik kanan `Shemas/public  View Diagram`
7. **Pertanyaan 1**
   a. Screenshot hasil diagram yang sudah ada, dan berikan hasil analisis kalian mengenai desain basis data pada db_mobile_banking



- Nasabah menyimpan data pelanggan yaitu id, nik, nama lengkap, kontak, alamat, tanggal lahir, pekerjaan, saldo utama)
- Rekening menyimpan data rekening yang dimiliki nasabah yaitu id, id nasabah, nomor rekening, jenis rekening, status
- Transaksi mencatat detail setiap perpindahan dana yaitu id, rekening asal, rekening tujuan, jenis transaksi, jumlah, berita, id merchant status, tanggal transaksi
- Mutase_rekening mencatat perubahan saldo pada rekening yaitu id mutase, id rekening, id transaksi, jenis mutasi, saldo akhir, jumlah mutase
- Merchant menyimpan data pihak penerima atau penyedia barang yaitu id merchant, kode merchant, nama merchant, kategori

b. jelaskan pola relasi yang ada
- nasabah => rekening , satu nasabah dapat memiliki banyak rekening
- rekening => mutase_rekening , satu rekening dapat memiliki banyak catatan mutase\
- transaksi => mutase_rekening , satu transaksi dapat memicu banyak catatan mutase
- merchant => transaksi , satu merchant dapat terlibat dalam banyak transaksi
- rekening => transaksi , satu rekening dapat menjadi asal atau tujuan bagi banyak transaksi

Praktikum 01 – Basic Transaction

Transaction pada DBMS merupakan sekumpulan operasi database yang dianggap sebagai satu kesatuan (*atomic unit of work*). Tujuannya menjaga agar data tetap konsisten walaupun ada banyak operasi atau kegagalan. Transaction harus memenuhi prinsip **ACID (Atomicity, Consistency, Isolation, Durability)**. Perintah dasar transaction terdiri dari:

Perintah	Keterangan
BEGIN	memulai transaksi, semua query setelah BEGIN akan dianggap satu kesatuan transaksi sampai ada COMMIT atau ROLLBACK.
COMMIT	Menyimpan semua perubahan transaksi ke database secara permanen. Setelah COMMIT, data tidak bisa dibatalkan dengan ROLLBACK.

13. Eksekusi query berikut untuk simulasi payment saldo yang memenuhi kentutan (transaksi sukses), screenshot hasilnya

ROLLBACK	Membatalkan semua perubahan sejak transaksi dimulai (kembali ke kondisi sebelum BEGIN).
SAVEPOINT	Membuat titik checkpoint dalam transaksi. Bisa melakukan ROLLBACK TO SAVEPOINT tanpa membatalkan seluruh transaksi.

Sintaks dasar dari transaction adalah:

```
begin;
 --set of statements
commit|rollback;
```

ACID merupakan sekumpulan karakteristik yang harus dimiliki sebuah transaksi basis data agar data tetap **benar, konsisten,** dan **andal,** meskipun ada transaksi yang berjalan bersamaan atau terjadi kegagalan.

1. *Atomicity*
   Memastikan bahwa seluruh operasi dalam satu transaksi dianggap sebagai satu kesatuan yang utuh (***all – or – nothing***). Jika semua berhasil, maka transaksi COMMIT Jika ada yang gagal, maka semua perubahan dibatalkan (ROLLBACK).
2. *Consistency*
   Transaksi harus membawa database dari satu **state valid** ke **state valid lainnya** sesuai aturan bisnis, constraint, dan integritas data. Mencegah data keluar dari aturan integritas (misalnya saldo tidak boleh negatif, constraint primary key harus unik).
3. *Isolation*
   Transaksi berjalan **seolah-olah sendiri**, tanpa gangguan transaksi lain. Menghindari anomali concurrency: dirty read, non-repeatable read, phantom read, lost update.
4. *Durability*
   Setelah transaksi berhasil (commit), perubahan data akan tetap tersimpan meskipun ada kegagalan sistem. Menjamin data yang sudah di-commit tidak hilang.

Pada praktikum ini, kita akan mempraktikkan simulasi transaction:

1. Kita buka SQL Editor dengan cara klik kanan `Schemas/public  SQL Editor  New SQL Script`
2. Langkah pertama adalah memulai transaksi dengan eksekusi perintah “BEGIN”

13. Eksekusi query berikut untuk simulasi payment saldo yang memenuhi kentutan (transaksi sukses), screenshot hasilnya

```
begin;
```

Statistics 1

Name	Value
Updated Rows	0
Execute time	0.0s
Start time	Tue Oct 21 22:14:57 WIB 2025
Finish time	Tue Oct 21 22:14:57 WIB 2025
Query	begin

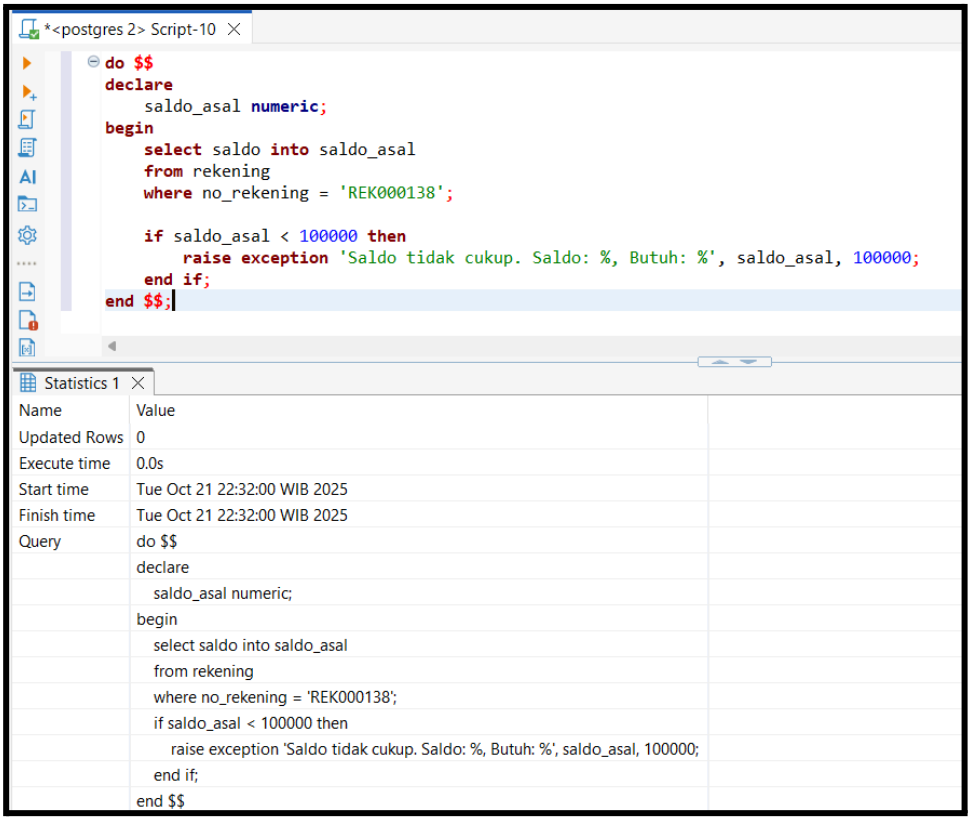
3. Eksekusi query SELECT berikut dan screenshot hasilnya

```
select no_rekening, saldo from rekening
where no_rekening in ('REK000138', 'REK000110');
```

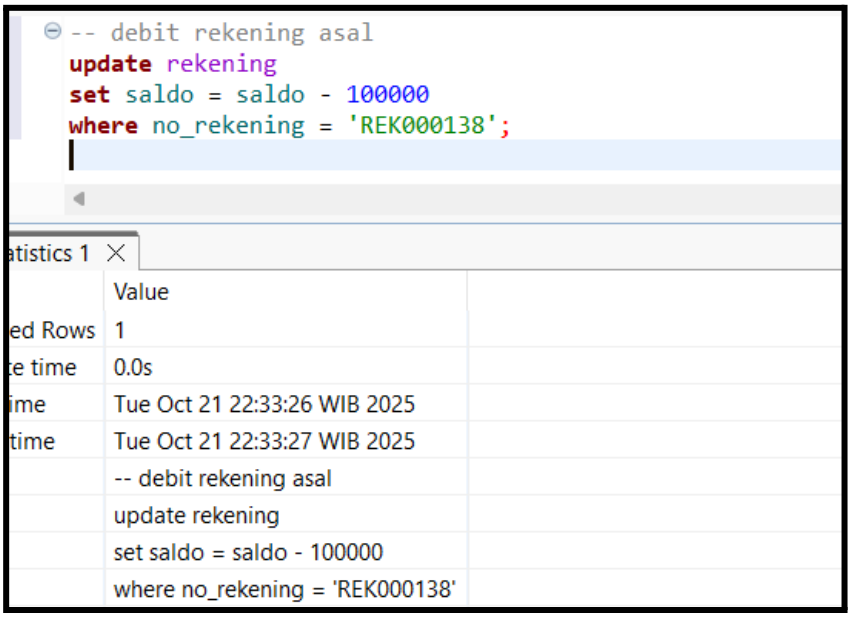
no_rekening	saldo
REK000138	959,089.22
REK000110	1,286,029.6

4. Eksekusi query berikut dan catat apa yang terjadi

13. Eksekusi query berikut untuk simulasi payment saldo yang memenuhi kentutan (transaksi sukses), screenshot hasilnya



5. Eksekusi query UPDATE berikut untuk melakukan debit rekening asal



13. Eksekusi query berikut untuk simulasi payment saldo yang memenuhi kentutan (transaksi sukses), screenshot hasilnya

6. Eksekusi query UPDATE berikut untuk melakukan credit ke rekening tujuan

```
-- credit rekening tujuan
update rekening
set saldo = saldo + 100000
where no_rekening = 'REK000110';
```

	Value
Rows	1
ime	0.0s
e	Tue Oct 21 22:34:23 WIB 2025
ne	Tue Oct 21 22:34:23 WIB 2025
	-- credit rekening tujuan
	update rekening
	set saldo = saldo + 100000
	where no_rekening = 'REK000110'

7. Cek saldo dalam transaksi dan screenshot hasilnya

```
-- cek di dalam transaksi
select no_rekening, saldo from rekening
where no_rekening in ('REK000138', 'REK000110');
```

o_rekening, saldo from rekening whe | Enter a SQL expression to fi

no_rekening	saldo
REK000138	859,089.22
REK000110	1,386,029.6

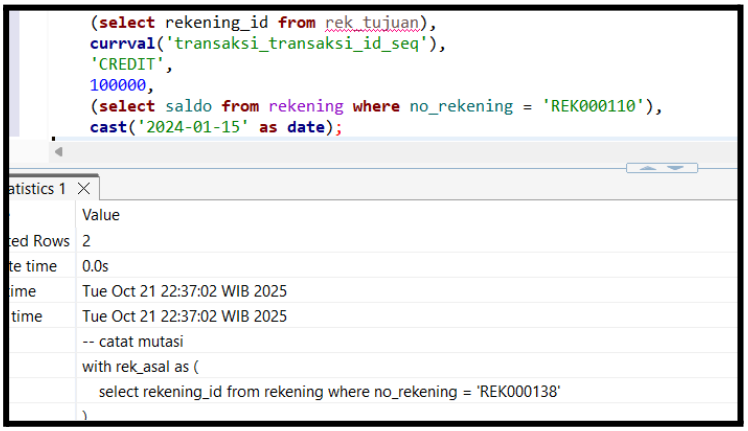
8. Catat transaksi ke dalam tabel transaksi menggunakan query INSERT berikut

```
-- catat transaksi
insert into transaksi
    (rekening_asal, rekening_tujuan, jenis_transaksi, jumlah, berita, tanggal_transaksi, status)
values
    ('REK000138', 'REK000110', 'TRANSFER', 100000,
     'Transfer manual tanpa trigger', '2025-10-07', 'SUCCESS');
```

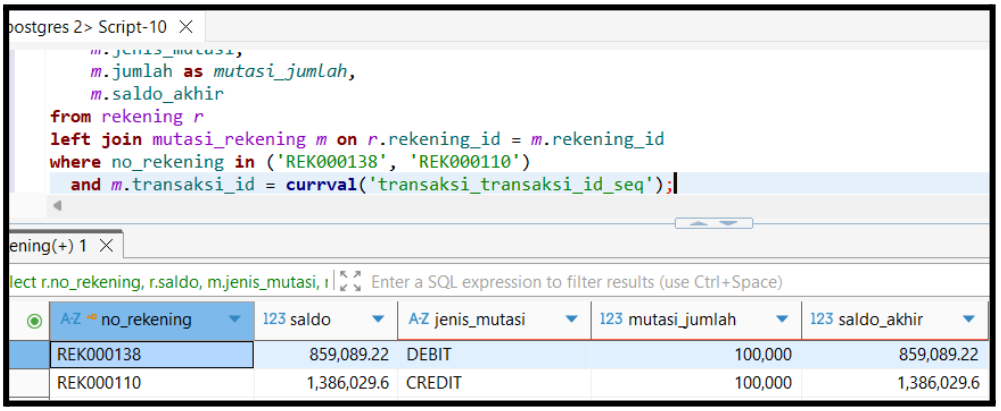
	Value
ows	1
ne	0.0s
	Tue Oct 21 22:35:49 WIB 2025
	Tue Oct 21 22:35:49 WIB 2025
	-- catat transaksi
	insert into transaksi
	(rekening_asal, rekening_tujuan, jenis_transaksi, jumlah, berita, tanggal_transaksi, status)
	values
	('REK000138', 'REK000110', 'TRANSFER', 100000,
	'Transfer manual tanpa trigger', '2025-10-07', 'SUCCESS')

13. Eksekusi query berikut untuk simulasi payment saldo yang memenuhi kentutan (transaksi sukses), screenshot hasilnya

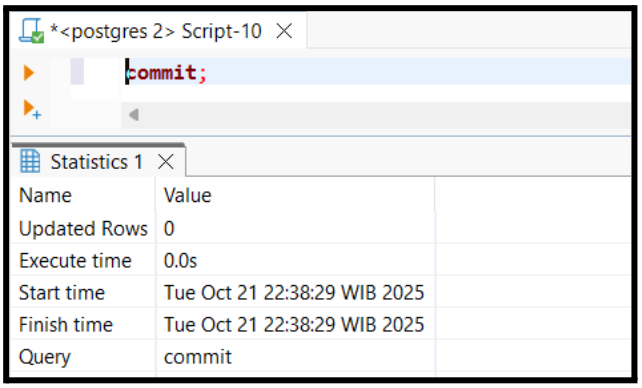
9. Dengan menggunakan CTE, buat catatan mutasi menggunakan query berikut:



10. Cek hasil dalam transaksi dengan query berikut dan screenshot hasilnya

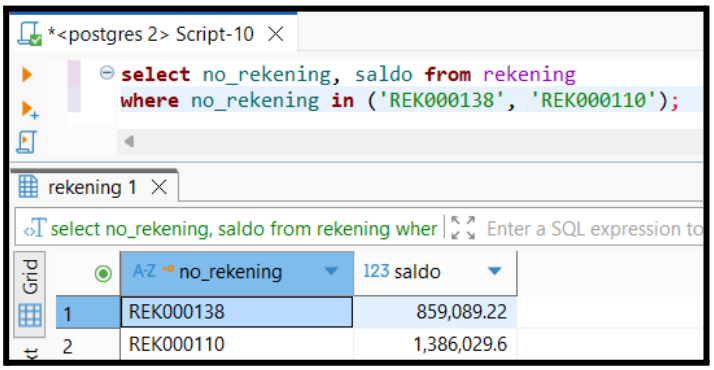


11. COMMIT transaction



13. Eksekusi query berikut untuk simulasi payment saldo yang memenuhi kentutan (transaksi sukses), screenshot hasilnya

12. Cek hasil setelah COMMIT dan screenshot hasilnya



13. Hasil Pengamatan

Langkah	Operasi	Hasil Sebelum	Hasil Sesudah	Apa yang terjadi?
3	Cek saldo awal	REK000138: 959.089 REK000110: 1.286.029		Belum ada ada perubahan pada saldo kedua rekening
7	Cek dalam transaksi		REK000138: 859.089 REK000110: 1.386.029	Saldo terupdate secara sementara
10	Cek mutasi		Debit: 859.089 Credit: 1.386.029	Memverifikasi mutasi berhasil di insert
12	Cek setelah commit		REK000138: 859.089 REK000110: 1.386.029	Seluruh perubahan saldo dan catatan mutasi disimpan secara permanen ke database

14. Pertanyaan Analisis

a. Apa yang terjadi jika berhenti di langkah 7 tanpa COMMIT?

- Transaksi pending, dan belum tersimpan di database

13. Eksekusi query berikut untuk simulasi payment saldo yang memenuhi kentutan (transaksi sukses), screenshot hasilnya

b. Bagaimana jika terjadi error di langkah 4 atau 5?

- Transaksi akan dihentikan, perintah raise exception di langkah 4 atau error fatal di langkah 5 akan memicu rollback otomatis

c. Sekarang eksekusi perintah ROLLBACK, Apa yang terjadi jika menjalankan ROLLBACK setelah COMMIT?

- Tidak terjadi apa apa pada data, karena perintah commit telah menyimpan perubahan secara permanen. Rollback hanya membatalkan transaksi yang belum di commit

Praktikum 02 – SAVEPOINT

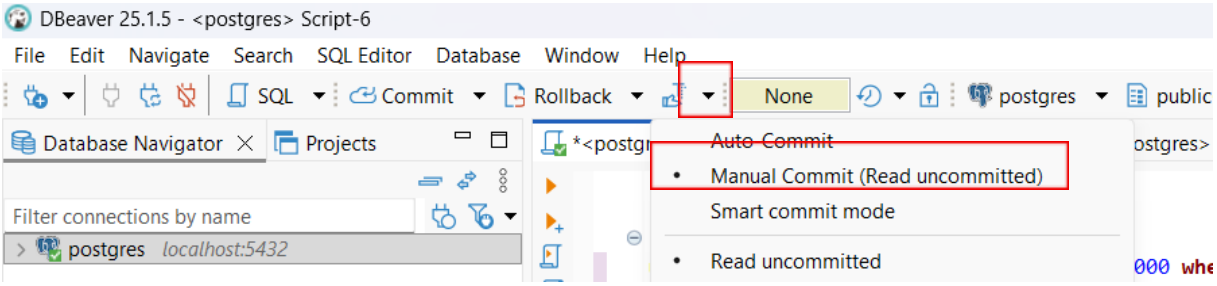
SAVEPOINT adalah titik penyimpanan sementara di dalam satu transaksi, yang memungkinkan kamu melakukan **rollback sebagian** tanpa membatalkan seluruh transaksi. Sintaks SAVEPOINT adalah sebagai berikut:

```
begin;
-- 1. Operasi pertama
update ...;
savepoint s1;

-- 2. Operasi kedua
update ...;
savepoint s2;

-- 3. Operasi ketiga (error)
update...; -- gagal

-- 4. Rollback ke savepoint s2 (batalkan langkah 3 saja)
rollback to savepoint s2;

-- 5. Lanjutkan transaksi
commit;
```

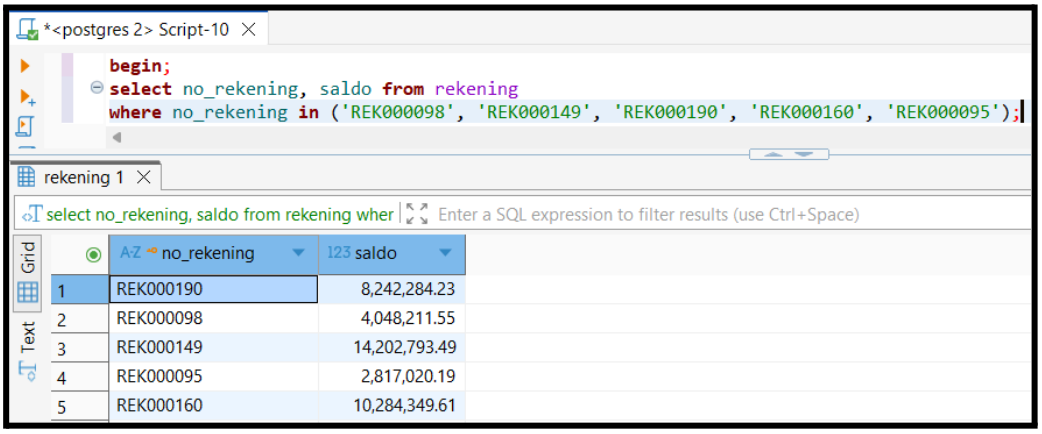
Pada praktikum ini, kita akan mempraktikkan simulasi transaction dengan SAVEPOINT:

1. Kita buka SQL Editor klik kanan `Schemas/public  SQL Editor  New SQL Script`
2. Pastikan **menonaktifkan fitur auto-commit** pada **dBeaver** dengan cara berikut:

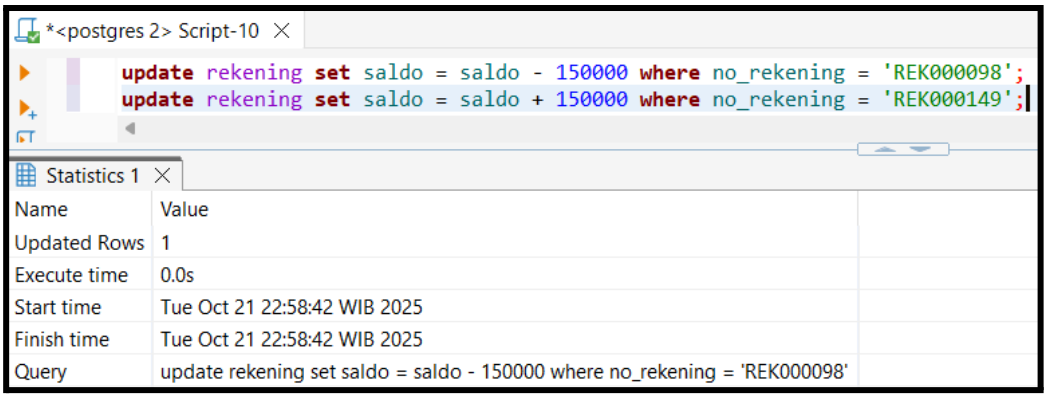
13. Eksekusi query berikut untuk simulasi payment saldo yang memenuhi kentutan (transaksi sukses), screenshot hasilnya



3. Jalankan query berikut untuk memulai transaksi dan mengecek saldo awal, screenshot hasilnya



4. Jalankan query berikut untuk mengeksekusi operation 1, membuat savepoint after_transfer_1, dan Cek saldo ditengah transaksi savepoint after_transfer_1. Screenshot hasilnya



13. Eksekusi query berikut untuk simulasi payment saldo yang memenuhi kentutan (transaksi sukses), screenshot hasilnya

```
savepoint after_transfer_1;
select no_rekening, saldo from rekening
where no_rekening in ('REK000098', 'REK000149', 'REK000190', 'REK000160', 'REK000095');
```

no_rekening	saldo
REK000190	8,242,284.23
REK000095	2,817,020.19
REK000160	10,284,349.61
REK000149	14,502,793.49
REK000098	3,898,211.55

5. Jalankan query berikut untuk mengeksekusi operation 2, membuat savepoint `after_transfer_2`, dan Cek saldo ditengah transaksi savepoint `after_transfer_2`. Screenshot hasilnya

```
update rekening set saldo = saldo - 2500000 where no_rekening = 'REK000190';
update rekening set saldo = saldo + 2500000 where no_rekening = 'REK000160';
```

Statistics 1

Name	Value
Updated Rows	1
Execute time	0.0s
Start time	Tue Oct 21 23:01:33 WIB 2025
Finish time	Tue Oct 21 23:01:33 WIB 2025
Query	update rekening set saldo = saldo + 2500000 where no_rekening = 'REK000160'

```
select no_rekening, saldo from rekening
where no_rekening in ('REK000098', 'REK000149', 'REK000190', 'REK000160', 'REK000095');
```

no_rekening	saldo
REK000190	8,242,284.23
REK000095	2,817,020.19
REK000149	14,502,793.49
REK000098	3,898,211.55
REK000160	12,784,349.61

13. Eksekusi query berikut untuk simulasi payment saldo yang memenuhi kentutan (transaksi sukses), screenshot hasilnya

6. ROLLBACK transaksi ke SAVEPOINT `after_transfer_1` dan cek saldo. Screenshot hasilnya

```
rollback to savepoint after_transfer_1;

select no_rekening, saldo from rekening
where no_rekening in ('REK000098', 'REK000149', 'REK000190', 'REK000160', 'REK000095');
```

no_rekening	saldo
REK000190	8,242,284.23
REK000095	2,817,020.19
REK000149	14,502,793.49
REK000098	3,898,211.55
REK000160	12,784,349.61

7. Lalu COMMIT transaction, dan cek kembali saldo akhir, catat perubahannya

```
commit;

select no_rekening, saldo from rekening
where no_rekening in ('REK000098', 'REK000149', 'REK000190', 'REK000160', 'REK000095');
```

	no_rekening	saldo
1	REK000190	8,242,284.23
2	REK000095	2,817,020.19
3	REK000149	14,502,793.49
4	REK000098	3,898,211.55
5	REK000160	12,784,349.61

8. **Hasil Pengamatan**

Langkah	Operasi	Hasil Sebelum	Hasil Sesudah	Apa yang terjadi?
3	Cek saldo awal	REK000098:8.242.284,23 REK000149: 14.202.793,49 REK000160: 10.284.349,61 REK000095: 2.817.020,19		Tidak berubah karena hanya select untuk menampilkan saldo awal
4	Cek setelah savepoint 1		REK000098: 3.898.211,55 REK000149: 14.352.793,49 REK000160: 10.284.349,61 REK000095: 2.817.020,19	Perubahan saldo berhasil dibuat, dan menandai savepoint pertama

13. Eksekusi query berikut untuk simulasi payment saldo yang memenuhi kentutan (transaksi sukses), screenshot hasilnya

5	Cek setelah operation 2		REK000098: 3.898.211,55 REK000149: 14.352.793,49 REK000160: 10.534.349,61 REK000095: 2.817.020,19	Perubahan saldo kedua berhasil dibuat
6	Cek setetelah rollback to savepoint 1		REK000098: 3.898.211,55 REK000149: 14.352.793,49 REK000160: 10.284.349,61 REK000095: 2.817.020,19	Membatalkan semua perubahan yang dibuat setelah savepoint 1,yaitu transfer kedua
7	Cek setelah commit		REK000098: 3.898.211,55 REK000149: 14.352.793,49 REK000160: 10.284.349,61 REK000095: 2.817.020,19	commit menyimpan hasil akhir

9. Pertanyaan Analisis

a. Bagaimana jika terjadi error di langkah 5?

- Postgre akan menandai transaksi sebagai failed, sehingga semua query berikutnya tidak akan bisa dijalankan sampai melakukan rollback, sehingga bisa rollback ke savepoint 1 di langkah 4 tanpa menghapus transaksi LANGKAH 4

b. Bagaimana jika kita menggunakan ROLLBACK tanpa menyebut savepoint?

- Seluruh langkah transaksi dari begin akan di batalkan

Praktikum 03 – Concurrent Transaction

Concurrency adalah kemampuan DBMS untuk menjalankan **banyak transaksi secara bersamaan (parallel).** Tujuannya memaksimalkan kinerja sistem menangani akses data oleh banyak user di saat bersamaan. Tantangannya mencegah masalah seperti: dirty read, lost update, phantom read.

1. Dirty Read

Transaksi A membaca data yang sedang diubah oleh Transaksi B, padahal B belum commit. Jika B rollback, maka A membaca data “kotor” (tidak valid).

Time	Transaction A	Transaction B

13. Eksekusi query berikut untuk simulasi payment saldo yang memenuhi kentutan (transaksi sukses), screenshot hasilnya

t1	BEGIN	-
t2	UPDATE accounts SET balance = 2000 WHERE id = 1;	SELECT balance FROM accounts WHERE id = 1 -- Output 2000
t3	ROLLBACK	-
t4	Transaction A reads an uncommited value written by Transaction B	

2. *Lost Update*

Terjadi ketika dua transaksi membaca data yang sama, lalu sama-sama mengubahnya, tetapi update terakhir menimpa hasil update pertama.

Time	Transaction A	Transaction B
t1	BEGIN SELECT balance FROM account WHERE id = 1; -- Output: 1000	BEGIN SELECT balance FROM account WHERE id = 1; -- Output: 1000
t2	UPDATE accounts SET balance = balance + 200 WHERE id= 1; -- Output yang diharapkan 1200	-
t3	-	UPDATE accounts SET balance = balance-300 WHERE id= 1; COMMIT; -- Output 700
t4	SELECT balance FROM account WHERE id = 1; -- Output: 700	

3. Non repeatable read

Terjadi ketika dalam satu transaksi, baris data yang sama dibaca dua kali, tetapi hasilnya berbeda karena ada transaksi lain yang melakukan UPDATE atau DELETE dan COMMIT di antara dua pembacaan tersebut.

13. Eksekusi query berikut untuk simulasi payment saldo yang memenuhi kentutan (transaksi sukses), screenshot hasilnya

Time	Transaction A	Transaction B
t1	BEGIN SELECT balance FROM account WHERE id = 1; -- Output: 1000	
t2		UPDATE accounts SET balance = balance-300 WHERE id= 1; COMMIT; -- Output 700
t3	SELECT balance FROM account WHERE id = 1; -- Output: 700	

4. Phantom read

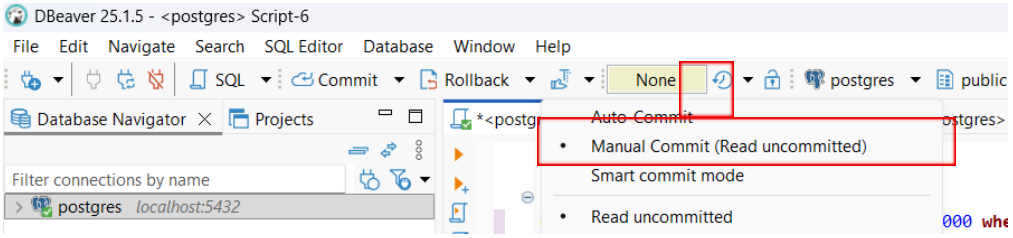
Terjadi ketika transaksi membaca **sekumpulan baris dengan kondisi tertentu**, lalu transaksi lain **menambah/menghapus** baris sehingga jumlah hasil query berubah saat dibaca ulang.

Time	Transaction A	Transaction B
t1	BEGIN SELECT * FROM orders WHERE total > 500; -- Output: Adi (600), Budi (800)	-
t2	-	BEGIN INSERT INTO orders VALUES (‘Chika’, 700); COMMIT;
t3	BEGIN SELECT * FROM orders WHERE total > 500; -- Output: Adi (600), Budi (800), Chika (700)	-

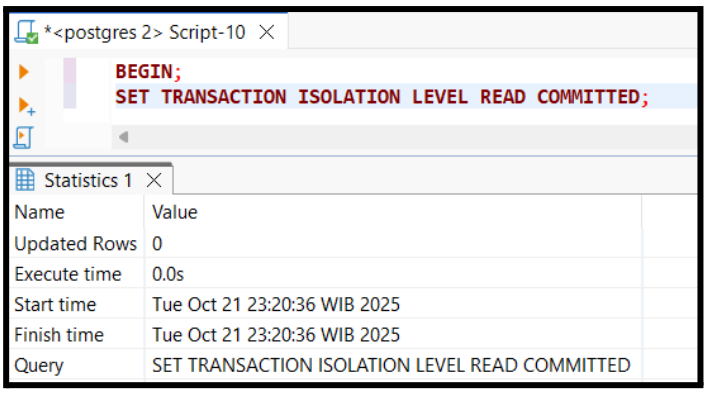
13. Eksekusi query berikut untuk simulasi payment saldo yang memenuhi kentutan (transaksi sukses), screenshot hasilnya

Langkah praktikum Isolation Level:

1. Buka 2 window SQL Editor
2. Pastikan **menonaktifkan fitur auto-commit** di kedua window dengan cara berikut:

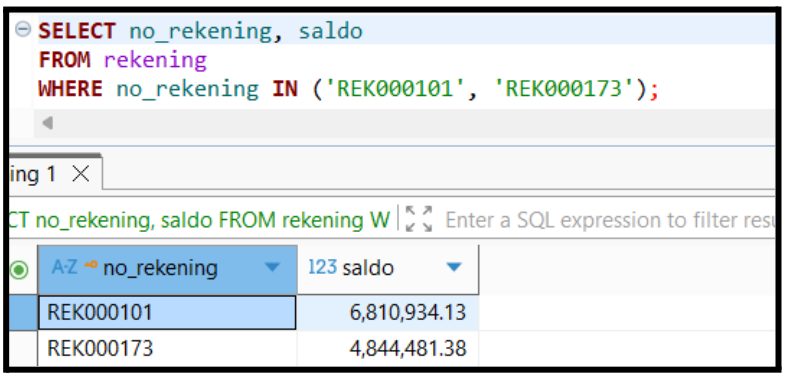


3. Pada window 1, eksekusi query berikut untuk memulai transaksi dan set isolation level READ COMMITED, tetapi jangan COMMIT dulu



4. Pada window 1, eksekusi query untuk menampilkan saldo awal seperti berikut, screenshot hasilnya

   Transaksi pada window 1 **jangan** di-commit



13. Eksekusi query berikut untuk simulasi payment saldo yang memenuhi kentutan (transaksi sukses), screenshot hasilnya

5. Pada window 2, eksekusi query berikut untuk memulai transaksi, set isolation level READ COMMITED, dan melihat saldo awal. screenshot hasilnya

```
BEGIN;
SET TRANSACTION ISOLATION LEVEL READ COMMITTED;
SELECT no_rekening, saldo
FROM rekening
WHERE no_rekening IN ('REK000101', 'REK000173');
```

no_rekening	saldo
REK000101	6,810,934.13
REK000173	4,844,481.38

6. Pada window 2, eksekusi berikut untuk update saldo dan insert mutasi. Screenshot hasilnya

```
UPDATE rekening SET saldo = saldo - 1000000 WHERE no_rekening = 'REK000101';
UPDATE rekening SET saldo = saldo + 1000000 WHERE no_rekening = 'REK000173';

INSERT INTO transaksi (rekening_asal, rekening_tujuan, jenis_transaksi, jumlah, berita, tanggal_transaksi)
VALUES ('REK000101', 'REK000173', 'TRANSFER', 1000000, 'Percobaan isolation level', CURRENT_DATE);

COMMIT;
SELECT no_rekening, saldo FROM rekening WHERE no_rekening IN ('REK000101', 'REK000173');
```

no_rekening	saldo
REK000101	6,810,934.13
REK000173	5,844,481.38

7. Kembali ke window 1, cek ulang saldo dan catat hasilnya

```
SELECT no_rekening, saldo FROM rekening
WHERE no_rekening IN ('REK000101', 'REK000173');
```

	no_rekening	saldo
1	REK000101	6,810,934.13
2	REK000173	5,844,481.38

13. Eksekusi query berikut untuk simulasi payment saldo yang memenuhi kentutan (transaksi sukses), screenshot hasilnya

8. **Hasil Pengamatan**

Langkah	Operasi	Hasil Sebelum	Hasil Sesudah	Apa yang terjadi?
4	Cek saldo awal window 1	REK000101: 6.810.934,13 REK000173: 4.844.481,38		Membaca saldo awal
5	Cek saldo awal window 2		REK000101: 6.810.934,13 REK000173: 4.844.481,38	Membaca saldo awal
6	Cek saldo setelah commit window 1		REK000101: 5.810.934,13 REK000173: 5.844.481,38	Windo 2 melakukan update saldo dan commit
7	Cek saldo akhir window 1		REK000101: 5.810.934,13 REK000173: 5.844.481.38	Membaca data baru yang sudah di commit window 2

9. **Pertanyaan Analisis**

a. Bagaimana saldo awal dan akhir yang dilihat oleh transaksi pada window 1?
   - Window 1 melihat data yang sudah di commit terbaru, saat pengecekan awal data yang terlihat adalah data awal yang belum dirubah, lalu saat pengecekan terakhir, maka menampilkan data yang sudah dirubah sebelumnya oleh window 2

b. Apa yang dapat kamu simpulkan dari isolation level READ COMMITED?
   - Pada level read committed, setiap perintah select di dalam satu transaksi selalu membaca data terakhir yang sudah di commit oleh transaksi lain

Praktikum 04 – Trigger

13. Eksekusi query berikut untuk simulasi payment saldo yang memenuhi kentutan (transaksi sukses), screenshot hasilnya

Trigger adalah prosedur otomatis yang dijalankan oleh database saat event tertentu terjadi pada tabel atau view. Trigger di PostgreSQL biasanya terdiri dari:

- Event
  kapan trigger berjalan (INSERT, UPDATE, DELETE, TRUNCATE).
- Timing
  sebelum atau sesudah event terjadi (BEFORE, AFTER, INSTEAD OF).
- Table/View
  objek yang dipantau trigger.
- Function
  kode (biasanya PL/pgSQL) yang dieksekusi.

Jenis trigger bervariasi bergantung pada waktu eksekusinya. Berikut ini adalah jenis-jenis trigger:

Jenis	Keterangan
BEFORE	Dijalankan sebelum operasi terjadi, bisa validasi/modifikasi data
AFTER	Dijalankan setelah operasi terjadi, cocok untuk logging
INSTEAD OF	Digunakan pada view, mengganti aksi default
Row-level trigger	Aktif untuk setiap baris yang terpengaruh
Statement-level trigger	Aktif sekali untuk seluruh query, tidak peduli berapa banyak baris yang terpengaruhi

Trigger juga memiliki aturan atau constraint. Berikut ini adalah constraint trigger:

Constraint	Keterangan
FOR EACH ROW	Trigger jalan per baris yang berubah
FOR EACH STATEMENT	Trigger jalan sekali per query
RETURN NEW	Data baru (INSERT/UPDATE)
RETURN OLD	Data lama (UPDATE/DELETE)

Langkah praktikum trigger

1. Buka SQL Editor

13. Eksekusi query berikut untuk simulasi payment saldo yang memenuhi kentutan (transaksi sukses), screenshot hasilnya

2. Eksekusi query berikut ini menambahkan constraint default pada tabel transaksi

*<postgres 2> Script-10

```
ALTER TABLE transaksi
ALTER COLUMN tanggal_transaksi SET DEFAULT CURRENT_DATE;
```

Statistics 1

Name	Value
Updated Rows	0
Execute time	0.0s
Start time	Tue Oct 21 23:49:02 WIB 2025
Finish time	Tue Oct 21 23:49:02 WIB 2025
Query	ALTER TABLE transaksi
	ALTER COLUMN tanggal_transaksi SET DEFAULT CURRENT_DATE

3. Buat function untuk validasi saldo sebelum transaksi

*<postgres 2> Script-10

```
CREATE OR REPLACE FUNCTION validate_saldo_sebelum_transaksi()
RETURNS TRIGGER AS $$
DECLARE
    saldo_sekarang DECIMAL(15,2);
BEGIN
    -- Hanya lakukan validasi jika transaksi adalah payment
    IF new.jenis_transaksi = 'payment' THEN
        SELECT saldo INTO saldo_sekarang
        FROM rekening
        WHERE no_rekening = new.rekening_asal;
```

Statistics 1

Name	Value
Updated Rows	0
Execute time	0.0s
Start time	Tue Oct 21 23:50:10 WIB 2025
Finish time	Tue Oct 21 23:50:10 WIB 2025
Query	CREATE OR REPLACE FUNCTION validate_saldo_sebelum_transaksi()
	RETURNS TRIGGER AS $$
	DECLARE
	saldo_sekarang DECIMAL(15,2);
	BEGIN
	-- Hanya lakukan validasi jika transaksi adalah payment
	IF new.jenis_transaksi = 'payment' THEN
	SELECT saldo INTO saldo_sekarang
	FROM rekening
	WHERE no_rekening = new.rekening_asal;

13. Eksekusi query berikut untuk simulasi payment saldo yang memenuhi kentutan (transaksi sukses), screenshot hasilnya

4. Buat Function untuk update saldo rekening

*<postgres 2> Script-10

```
-- 2. Buat function update saldo rekening
CREATE OR REPLACE FUNCTION update_rekening_saldo()
RETURNS TRIGGER AS $$
BEGIN
    IF new.status = 'success' THEN
        RAISE NOTICE 'Memproses update saldo untuk: %', new.jenis_transaksi;

        -- Jika transaksi adalah payment, saldo dikurangi
        IF new.jenis_transaksi = 'payment' THEN
            UPDATE rekening
            SET saldo = saldo - new.jumlah
```

Statistics 1

Name	Value
Updated Rows	0
Execute time	0.0s
Start time	Tue Oct 21 23:52:13 WIB 2025
Finish time	Tue Oct 21 23:52:13 WIB 2025
Query	-- 2. Buat function update saldo rekening
	CREATE OR REPLACE FUNCTION update_rekening_saldo()
	RETURNS TRIGGER AS $$
	BEGIN
	IF new.status = 'success' THEN
	RAISE NOTICE 'Memproses update saldo untuk: %', new.jenis_transaksi;

5. Buat trigger menggunakan query berikut

*<postgres 2> Script-10

```
CREATE TRIGGER trigger_validate_saldo_sebelum_transaksi
BEFORE INSERT ON transaksi
FOR EACH ROW
EXECUTE FUNCTION validate_saldo_sebelum_transaksi();

CREATE TRIGGER trigger_update_rekening_saldo
AFTER INSERT ON transaksi
FOR EACH ROW
EXECUTE FUNCTION update_rekening_saldo();
```

Statistics 1

Name	Value
Updated Rows	0
Execute time	0.0s
Start time	Tue Oct 21 23:53:44 WIB 2025
Finish time	Tue Oct 21 23:53:44 WIB 2025
Query	CREATE TRIGGER trigger_update_rekening_saldo
	AFTER INSERT ON transaksi
	FOR EACH ROW
	EXECUTE FUNCTION update_rekening_saldo()

13. Eksekusi query berikut untuk simulasi payment saldo yang memenuhi kentutan (transaksi sukses), screenshot hasilnya

6. Pastikan trigger aktif dengan menjalankan query berikut

```
SELECT
    trigger_name,
    event_object_table AS table_name,
    action_timing AS timing,
    event_manipulation AS event
FROM information_schema.triggers
WHERE trigger_schema = 'public';
```

trigger_name	table_name	timing	event
trigger_update_rekening_saldo	transaksi	AFTER	INSERT

7. Eksekusi query berikut untuk mengatur saldo awal dan menampilkannya. Screenshot hasilnya

```
UPDATE rekening SET saldo = 1000000
WHERE no_rekening IN ('REK000114', 'REK000180');

SELECT no_rekening, saldo FROM rekening
WHERE no_rekening IN ('REK000114', 'REK000180');
```

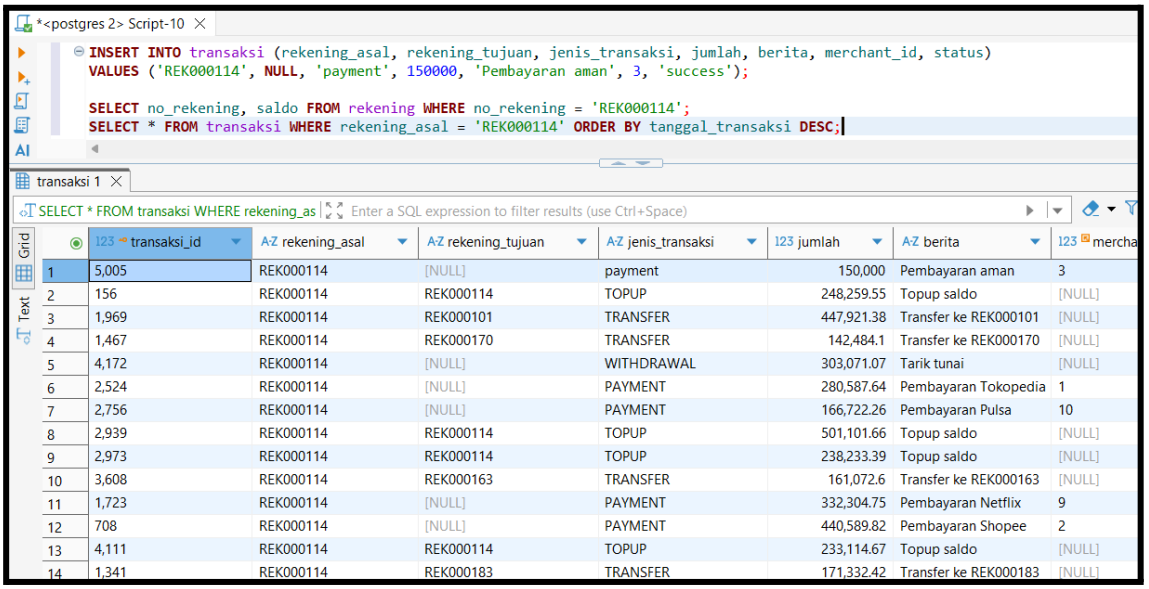
	no_rekening	saldo
1	REK000180	14,421,875.51
2	REK000114	7,339,251.96

8. Eksekusi query berikut untuk simulasi topup saldo yang memenuhi kentutan (transaksi sukses), screenshot hasilnya

```
INSERT INTO transaksi (rekening_asal, rekening_tujuan, jenis_transaksi, jumlah, berita, merchant_id, status)
VALUES ('REK000180', 'REK000180', 'topup', 200000, 'Test topup', NULL, 'success');

SELECT no_rekening, saldo FROM rekening WHERE no_rekening = 'REK000180';
SELECT * FROM transaksi WHERE rekening_asal = 'REK000180' ORDER BY transaksi_id DESC LIMIT 1;
```

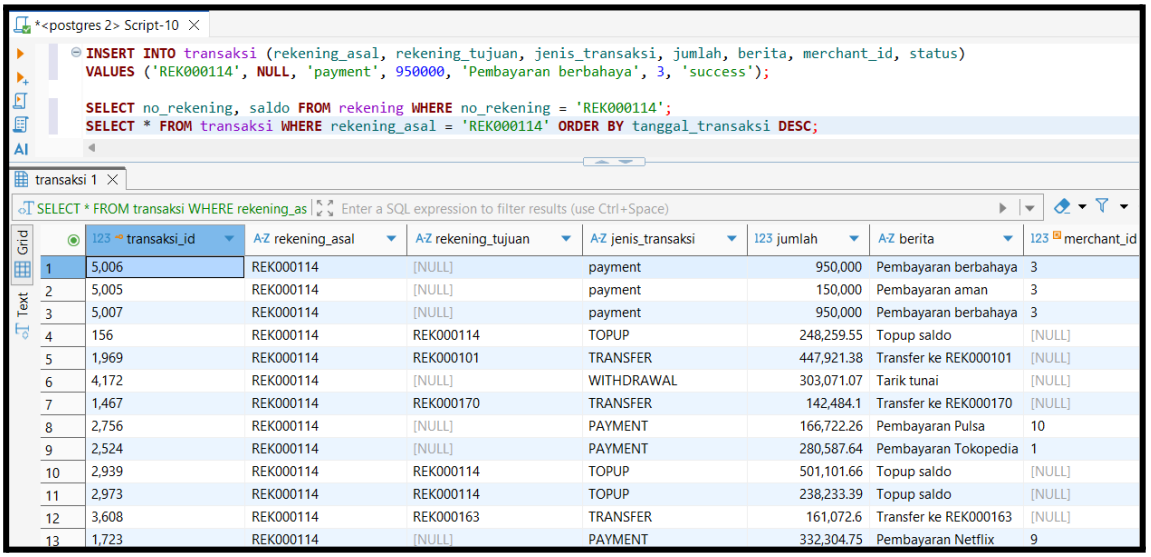
transaksi_id	rekening_asal	rekening_tujuan	jenis_transaksi	jumlah	berita	merchant_id
5,004	REK000180	REK000180	topup	200,000	Test topup	[NULL]

13. Eksekusi query berikut untuk simulasi payment saldo yang memenuhi kentutan (transaksi sukses), screenshot hasilnya



	transaksi_id	rekening_asal	rekening_tujuan	jenis_transaksi	jumlah	berita	merchant_id
1	5,005	REK000114	[NULL]	payment	150,000	Pembayaran aman	3
2	156	REK000114	REK000114	TOPUP	248,259.55	Topup saldo	[NULL]
3	1,969	REK000114	REK000101	TRANSFER	447,921.38	Transfer ke REK000101	[NULL]
4	1,467	REK000114	REK000170	TRANSFER	142,484.1	Transfer ke REK000170	[NULL]
5	4,172	REK000114	[NULL]	WITHDRAWAL	303,071.07	Tarik tunai	[NULL]
6	2,524	REK000114	[NULL]	PAYMENT	280,587.64	Pembayaran Tokopedia	1
7	2,756	REK000114	[NULL]	PAYMENT	166,722.26	Pembayaran Pulsa	10
8	2,939	REK000114	REK000114	TOPUP	501,101.66	Topup saldo	[NULL]
9	2,973	REK000114	REK000114	TOPUP	238,233.39	Topup saldo	[NULL]
10	3,608	REK000114	REK000163	TRANSFER	161,072.6	Transfer ke REK000163	[NULL]
11	1,723	REK000114	[NULL]	PAYMENT	332,304.75	Pembayaran Netflix	9
12	708	REK000114	[NULL]	PAYMENT	440,589.82	Pembayaran Shopee	2
13	4,111	REK000114	REK000114	TOPUP	233,114.67	Topup saldo	[NULL]
14	1,341	REK000114	REK000183	TRANSFER	171,332.42	Transfer ke REK000183	[NULL]

9. Eksekusi query berikut untuk simulasi payment saldo yang tidak memenuhi kentutan (transaksi gagal), screenshot hasilnya



	transaksi_id	rekening_asal	rekening_tujuan	jenis_transaksi	jumlah	berita	merchant_id
1	5,006	REK000114	[NULL]	payment	950,000	Pembayaran berbahaya	3
2	5,005	REK000114	[NULL]	payment	150,000	Pembayaran aman	3
3	5,007	REK000114	[NULL]	payment	950,000	Pembayaran berbahaya	3
4	156	REK000114	REK000114	TOPUP	248,259.55	Topup saldo	[NULL]
5	1,969	REK000114	REK000101	TRANSFER	447,921.38	Transfer ke REK000101	[NULL]
6	4,172	REK000114	[NULL]	WITHDRAWAL	303,071.07	Tarik tunai	[NULL]
7	1,467	REK000114	REK000170	TRANSFER	142,484.1	Transfer ke REK000170	[NULL]
8	2,756	REK000114	[NULL]	PAYMENT	166,722.26	Pembayaran Pulsa	10
9	2,524	REK000114	[NULL]	PAYMENT	280,587.64	Pembayaran Tokopedia	1
10	2,939	REK000114	REK000114	TOPUP	501,101.66	Topup saldo	[NULL]
11	2,973	REK000114	REK000114	TOPUP	238,233.39	Topup saldo	[NULL]
12	3,608	REK000114	REK000163	TRANSFER	161,072.6	Transfer ke REK000163	[NULL]
13	1,723	REK000114	[NULL]	PAYMENT	332,304.75	Pembayaran Netflix	9

13. Eksekusi query berikut untuk simulasi payment saldo yang memenuhi kentutan (transaksi sukses), screenshot hasilnya

10. Cek saldo akhir dan catatan transaksi terbaru dengan menjalankan query berikut

```
SELECT no_rekening, saldo
FROM rekening
WHERE no_rekening IN ('REK000114', 'REK000180');
```

	no_rekening	saldo
1	REK000180	14,621,875.51
2	REK000114	5,289,251.96

```
SELECT rekening_asal, jenis_transaksi, jumlah, status, tanggal_transaksi
FROM transaksi
WHERE rekening_asal IN ('REK000114', 'REK000180')
ORDER BY tanggal_transaksi DESC;
```

rekening_asal	jenis_transaksi	jumlah	status	tanggal_transaksi
REK000114	payment	950,000	success	2025-10-21
REK000114	payment	150,000	success	2025-10-21
REK000180	topup	200,000	success	2025-10-21
REK000114	payment	950,000	success	2025-10-21
REK000180	WITHDRAWAL	167,125.75	SUCCESS	2025-10-05
REK000114	TOPUP	248,259.55	SUCCESS	2025-10-04
REK000114	PAYMENT	166,722.26	SUCCESS	2025-10-03
REK000114	TRANSFER	447,921.38	SUCCESS	2025-10-03
REK000114	TRANSFER	142,484.1	SUCCESS	2025-10-03
REK000114	WITHDRAWAL	303,071.07	SUCCESS	2025-10-03
REK000114	PAYMENT	280,587.64	SUCCESS	2025-10-03
REK000114	TOPUP	501,101.66	SUCCESS	2025-10-02
REK000180	PAYMENT	145,068.52	SUCCESS	2025-10-01
REK000114	TOPUP	238,233.39	SUCCESS	2025-09-30

11. **Hasil Pengamatan**

Langkah	Operasi	Hasil Sebelum	Hasil Sesudah	Apa yang terjadi?
6	Cek trigger aktif			Setiap transaksi otomatis divalidasi dan saldo akan terupdate
7	Set saldo awal	REK000114: - REK000180: -	REK000114: 1.000.000 REK000180: 1.000.000	Saldo di set ke 1 juta pada rekening yang ditentukan

13. Eksekusi query berikut untuk simulasi payment saldo yang memenuhi kentutan (transaksi sukses), screenshot hasilnya

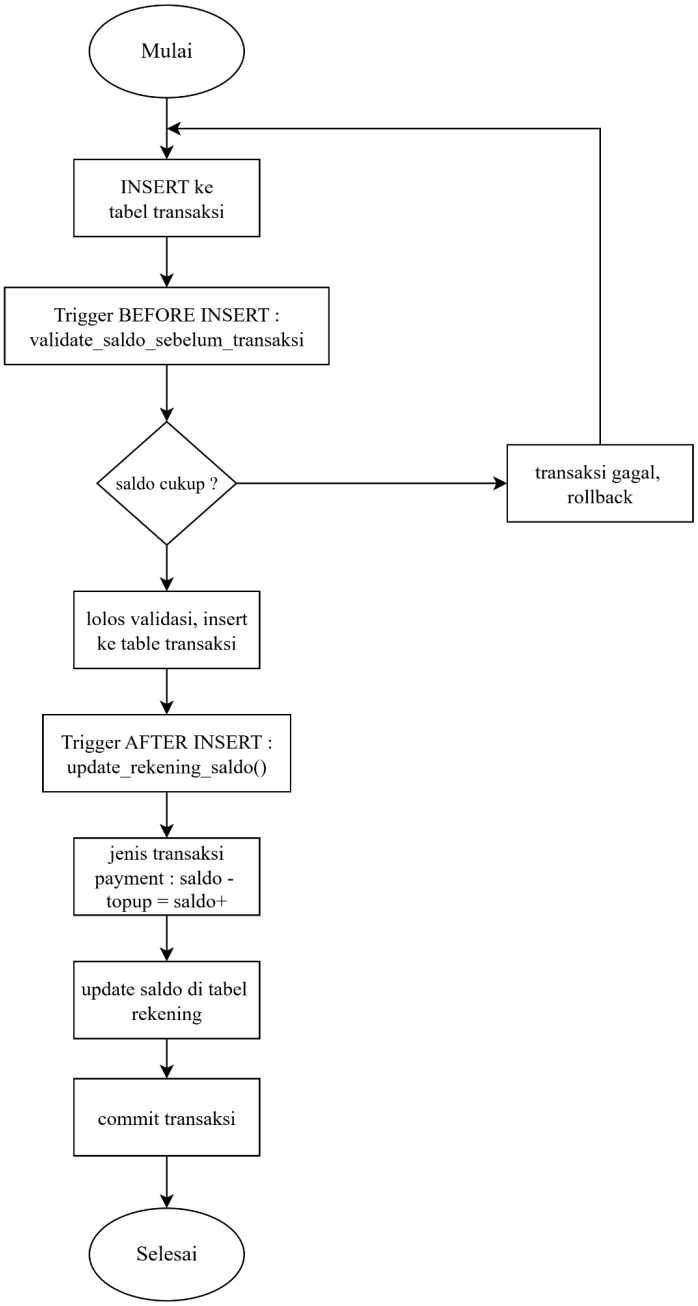
Langkah	Operasi	Hasil Sebelum	Hasil Sesudah	Apa yang terjadi?
8	Cek saldo topup berhasil		REK000114: 1.000.000 REK000180: 1.200.000	Saat top up sebesar 200.000, status success trigger after inser menambah saldo rekening asal,
	Cek transaksi topup berhasil			transaksi juga tercatat di tabel transaksi
9	Cek saldo payment berhasil		REK000114: 850.000 REK000180: 1.200.000	Transaksi payment sebesar 150.000 berhasil karena saldo >100.000. trigger after insert otomatis mengurangi saldo
	Cek transaksi payment berhasil			Transaksi juga tercatat di tabel transaksi
10	Payment gagal			transaksi payment sebesar 950.000 ditolak oleh trigger validate_saldo_sebelum_ transaksi, dikarenakan saldo akhir akan <100.000, sehingga muncul erorr dan transaksi dibatalkan
11	Cek saldo akhir		REK000114: 850.000 REK000180: 1.200.000	Hasil akhir menunjukkan top up dan payment pertama berhasil, payment kedua gagal
	Cek transaksi akhir			Transaksi juga tercatat di tabel transaksi

13. Eksekusi query berikut untuk simulasi payment saldo yang memenuhi kentutan (transaksi sukses), screenshot hasilnya

12. **Pertanyaan Analisis**
    a. Apa perbedaan fundamental antara trigger BEFORE dan AFTER dalam konteks praktikum ini?
        - Mengapa validasi saldo menggunakan BEFORE INSERT?
    - Trigger before cocok untuk validasi sebelum data masuk ke sistem
        - Mengapa update saldo menggunakan AFTER INSERT?
    - Trigger after cocok untuk aksi pasca transaksi, seperti update saldo

    b. Buat flowchart yang menunjukkan execution flow dari insert transaksi hingga update saldo, termasuk semua trigger yang terlibat!

13. Eksekusi query berikut untuk simulasi payment saldo yang memenuhi kentutan (transaksi sukses), screenshot hasilnya



**** Sekian, dan selamat belajar ****

13. Eksekusi query berikut untuk simulasi payment saldo yang memenuhi kentutan (transaksi sukses), screenshot hasilnya