JOBSHEET

BASIS DATA LANJUT

Jurusan Teknologi Informasi

POLITEKNIK NEGERI MALANG

TAHUN AJARAN 2025/2026

PERTEMUAN 7

Transaction & Concurrency

Team Teaching Basis Data Lanjut:

- Candra Bella Vista, S.Kom., MT.
- Moch Zawaruddin Abdullah, S.ST., M.Kom.
- Yan Watequlis Syaifudin, ST., MMT., PhD.
- Yoppy Yunhasnawa, S.ST., M.Sc.

KEMENTERIAN PENDIDIKAN, KEBUDAYAAN, RISET, DAN TEKNOLOGI
POLITEKNIK NEGERI MALANG
JURUSAN TEKNOLOGI INFORMASI
Jl. Soekarno Hatta No. 9, Jatimulyo, Lowokwaru, Malang 65141
Telp. (0341) 404424 – 404425, Fax (0341) 404420
http://www.polinema.ac.id

Mata Kuliah : Basis Data Lanjut
Program Studi : D4 – Teknik Informatika / 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 digram 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
    b. jelaskan pula relasi yang ada

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.
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"

```
-- LANGKAH 1: MULAI TRANSACTION
begin;
```

3. Eksekusi query SELECT berikut dan screenshot hasilnya

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

4. Eksekusi query berikut dan catat apa yang terjadi

```
-- validasi manual
do $$
declare
    saldo_asal numeric ;
begin
    select saldo into saldo_asal
    from rekening
    where no_rekening = 'REK000138';

    if saldo_asal < 100000 then
        raise exception 'Saldo tidak cukup. Saldo: %, Butuh: %', saldo_asal, 100000;
    end if;
end $$;
```

5. Eksekusi query UPDATE berikut untuk melakukan debit rekening asal

```
-- debit rekening asal
update rekening
set saldo = saldo - 100000
where no_rekening = 'REK000138';
```

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

```
-- 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');
```

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');
```

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

```
-- catat mutasi
with rek_asal as (select rekening_id from rekening where no_rekening = 'REK000138'),
     rek_tujuan as (select rekening_id FROM rekening WHERE no_rekening = 'REK000110')
)
insert into mutasi_rekening (rekening_id, transaksi_id, jenis_mutasi, jumlah, saldo_akhir, tanggal_mutasi)
select
    (select rekening_id from rek_asal),
    currval('transaksi_transaksi_id_seq'),
    'DEBIT',
    100000,
    (select saldo from rekening where no_rekening = 'REK000138'),
    cast('15-01-2024' as date)
union all
select
    (select rekening_id from rek_tujuan),
    currval('transaksi_transaksi_id_seq'),
    'CREDIT',
    100000,
    (select  saldo from  rekening where no_rekening = 'REK000110'),
    cast ('15-01-2024' as date)
```

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

```
 -- cek hasil dalam transaksi
select
    r.no_rekening, r.saldo, m.jenis_mutasi,
    m.jumlah as mutasi_jumlah, m.saldo_akhir
from rekening r
left join mutasi_rekening m ON r.rekening_id = m.rekening_id
where r.no_rekening in ('REK000138', 'REK000110')
and m.transaksi_id = currval('transaksi_transaksi_id_seq');
```

11. COMMIT transaction

```
-- commit transaction
commit;
```

12. Cek hasil setelah COMMIT dan screenshot hasilnya

```
-- cek hasil setelah commit
select no_rekening, saldo from  rekening
where no_rekening in ('REK000138', 'REK000110');
```

13. Hasil Pengamatan

Langkah	Operasi	Hasil Sebelum	Hasil Sesudah	Apa yang terjadi?
3	Cek saldo awal	REK000138: ? REK000110: ?		
7	Cek dalam transaksi		REK000138: ? REK000110: ?	
10	Cek mutasi		Debit: ? Credit: ?	
12	Cek setelah commit		REK000138: ? REK000110: ?	

14. Pertanyaan Analisis

a. Apa yang terjadi jika berhenti di langkah 7 tanpa COMMIT?
b. Bagaimana jika terjadi error di langkah 4 atau 5?
c. Sekarang eksekusi perintah ROLLBACK, Apa yang terjadi jika menjalankan ROLLBACK setelah COMMIT?

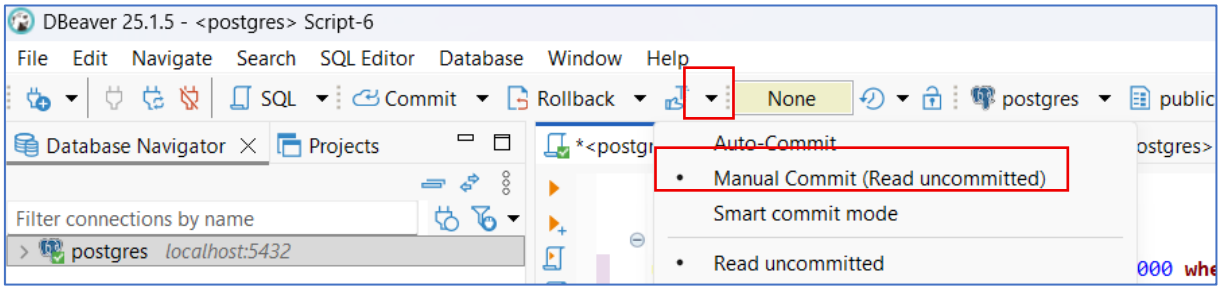
Praktikum 02 – SAVEPOINT

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:



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

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

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

```
-- operation 1: transfer a → b (sukses)
update rekening set saldo = saldo - 150000 where no_rekening = 'REK000098';
update rekening set saldo = saldo + 150000 where no_rekening = 'REK000149';

-- buat savepoint
savepoint after_transfer_1;
```

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

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

```
-- operation 2: transfer c → d
update rekening set saldo = saldo - 2500000 where no_rekening = 'REK000190';
update rekening set saldo = saldo + 2500000 where no_rekening = 'REK000160';

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

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

```
--Rollback ke savepoint
rollback to savepoint after_transfer_1;

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

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

```
-- commit transaction
commit;

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

8. Hasil Pengamatan

Langkah	Operasi	Hasil Sebelum	Hasil Sesudah	Apa yang terjadi?
3	Cek saldo awal	REK000098: ? REK000149: ? REK000160: ? REK000095: ?		
4	Cek setelah savepoint 1		REK000098: ? REK000149: ? REK000160: ? REK000095: ?	
5	Cek setelah operation 2		REK000098: ? REK000149: ? REK000160: ? REK000095: ?	
6	Cek setetelah rollback to savepoint 1		REK000098: ? REK000149: ? REK000160: ? REK000095: ?	
7	Cek setelah commit		REK000098: ? REK000149: ? REK000160: ? REK000095: ?	

9. Pertanyaan Analisis

a. Bagaimana jika terjadi error di langkah 5?
b. Bagaimana jika kita menggunakan ROLLBACK tanpa menyebut savepoint?

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
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.

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

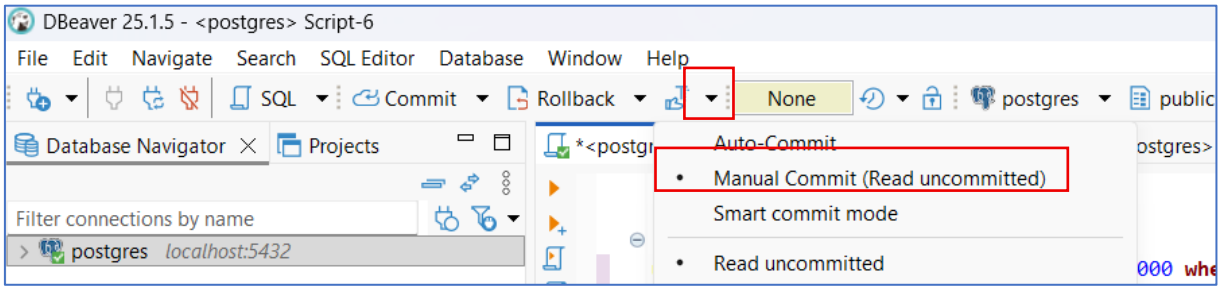
Terjadi ketika transaksi membaca **sekumpulan baris dengan kondisi tertentu**, lalu transaksi lain **menambah/menghapus** baris sehingga jumlah hasil query berubah saat dibaca ulang.

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)	-

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

```
BEGIN;
SET TRANSACTION ISOLATION LEVEL READ COMMITTED;
```

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

```
-- Baca saldo pertama kali
SELECT no_rekening, saldo FROM rekening
WHERE no_rekening IN('REK000101', 'REK000173');
```

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

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');
```

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

```
-- Lakukan transfer yang mengubah saldo
UPDATE rekening SET saldo = saldo - 1000000 WHERE no_rekening = 'REK000101';
UPDATE rekening SET saldo = saldo + 1000000 WHERE no_rekening = 'REK000173';

-- Catat transaksi
INSERT INTO transaksi (rekening_asal, rekening_tujuan, jenis_transaksi, jumlah, berita, tanggal_transaksi)
VALUES ('REK000101', 'REK000183', 'TRANSFER', 1000000, 'Percobaan isolation level', CURRENT_DATE);

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

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

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

8. **Hasil Pengamatan**

Langkah	Operasi	Hasil Sebelum	Hasil Sesudah	Apa yang terjadi?
4	Cek saldo awal window 1	REK000101: ? REK000173: ?		
5	Cek saldo awal window 2		REK000101: ? REK000173: ?	
6	Cek saldo setelah commit window 1		REK000101: ? REK000173: ?	
7	Cek saldo akhir window 1		REK000101: ? REK000173: ?	

9. **Pertanyaan Analisis**
   a. Bagaimana saldo awal dan akhir yang dilihat oleh transaksi pada window 1?
   b. Apa yang dapat kamu simpulkan dari isolation level READ COMMITED?

Praktikum 04 – Trigger

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

```
--tambahkan constraint default pada tabel transaksi
alter table transaksi
alter column tanggal_transaksi set default current_date;
```

3. Buat function untuk validasi saldo sebelum transaksi

```
-- 1. buat function validasi saldo
create or replace function validate_saldo_sebelum_transaksi()
returns trigger as $$
declare
    saldo_sekarang decimal(15,2);
begin
    if new.jenis_transaksi = 'payment' then
        select saldo into saldo_sekarang
        from rekening
        where no_rekening = new.rekening_asal;

        raise notice 'validasi: saldo % = %, transaksi = %', new.rekening_asal, saldo_sekarang, new.jumlah;

        if (saldo_sekarang - new.jumlah) < 100000 then
            raise exception 'transaksi ditolak: saldo tidak boleh kurang dari 100rb';
        end if;
    end if;
    return new;
end;
$$ language plpgsql;
```

4. Buat Function untuk update saldo rekening

```
-- 2. buat function update saldo
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;

        if new.jenis_transaksi = 'payment' then
            update rekening set saldo = saldo - new.jumlah
            where no_rekening = new.rekening_asal;
        end if;

        if new.jenis_transaksi = 'topup' then
            update rekening set saldo = saldo + new.jumlah
            where no_rekening = new.rekening_asal;
        end if;
    end if;
    return new;
end;
$$ language plpgsql;
```

5. Buat trigger menggunakan query berikut

```
-- 3. buat trigger
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();
```

6. Pastikan trigger aktif dengan menjalankan query berikut

```
-- cek trigger yang aktif
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';
```

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

```
-- persiapan saldo awal
update rekening set saldo = 1000000
where no_rekening in ('REK000114', 'REK000180');

-- verifikasi saldo awal
select no_rekening, saldo from rekening
where no_rekening in ('REK000114', 'REK000180');
```

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

```
-- simulasi topup saldo cukup
insert into transaksi (rekening_asal, rekening_tujuan, jenis_transaksi, jumlah, berita, merchant_id, status
values ('REK000180', 'REK000180', 'topup', 200000, 'test topup', null, 'success');

-- cek hasil (jalankan query satu per satu)
select no_rekening, saldo from rekening where no_rekening = 'REK000180';

select * from transaksi where rekening_asal = 'REK000180' order by transaksi_id desc limit 1;
```

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

```
-- payment dengan saldo cukup
insert into transaksi (rekening_asal, rekening_tujuan, jenis_transaksi, jumlah, berita, merchant_id, status)
values ('REK000114', null, 'payment', 150000, 'pembayaran aman', 3, 'success');

-- cek hasil (jalankan query satu per satu)
select no_rekening, saldo from rekening where no_rekening = 'REK000114';
select * from transaksi where rekening_asal = 'REK000114' order by tanggal_transaksi desc;
```

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

```
-- PAYMENT dengan saldo < 100rb
insert  into transaksi (rekening_asal, rekening_tujuan, jenis_transaksi, jumlah, berita, merchant_id, status)
values  ('REK000114', NULL, 'PAYMENT', 950000, 'Payment berbahaya', 3, 'SUCCESS');
```

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

```
-- cek saldo akhir
select no_rekening, saldo from rekening  where no_rekening in ('REK000114', 'REK000180');

-- cek transaksi
select  rekening_asal, jenis_transaksi, jumlah, status, tanggal_transaksi
from  transaksi
where rekening_asal in ('REK000114', 'REK000180')
order by tanggal_transaksi desc;
```

12. **Hasil Pengamatan**

Langkah	Operasi	Hasil Sebelum	Hasil Sesudah	Apa yang terjadi?
6	Cek trigger aktif			
7	Set saldo awal	REK000114: ? REK000180: ?	REK000114: ? REK000180: ?	
8	Cek saldo topup berhasil		REK000101: ? REK000173: ?	
	Cek transaksi topup berhasil			
9	Cek saldo payment berhasil		REK000101: ? REK000173: ?	
	Cek transaksi payment berhasil			
10	Payment gagal			
11	Cek saldo akhir		REK000114: ? REK000180: ?	
	Cek transaksi akhir			

13. **Pertanyaan Analisis**
    a. Apa perbedaan fundamental antara trigger BEFORE dan AFTER dalam konteks praktikum ini?
        - Mengapa validasi saldo menggunakan BEFORE INSERT?
        - Mengapa update saldo menggunakan AFTER INSERT?
    b. Buat flowchart yang menunjukkan execution flow dari insert transaksi hingga update saldo, termasuk semua trigger yang terlibat!

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