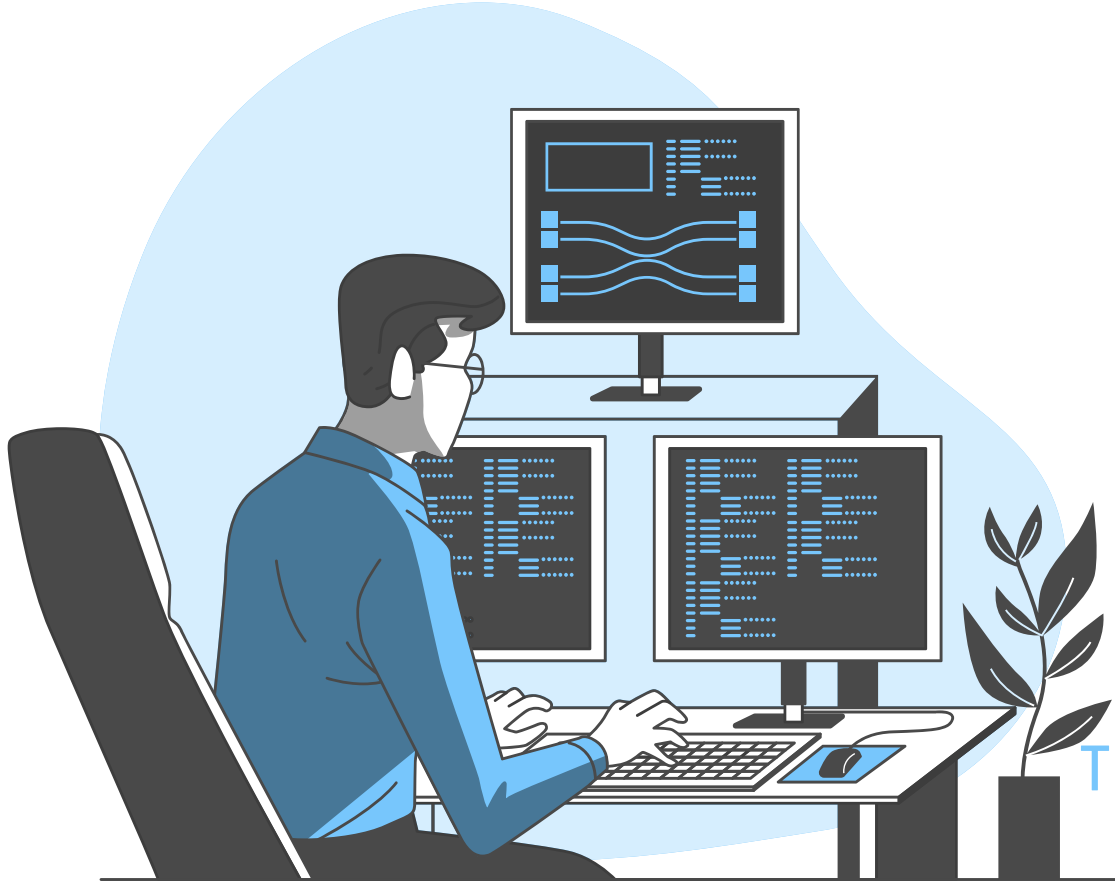
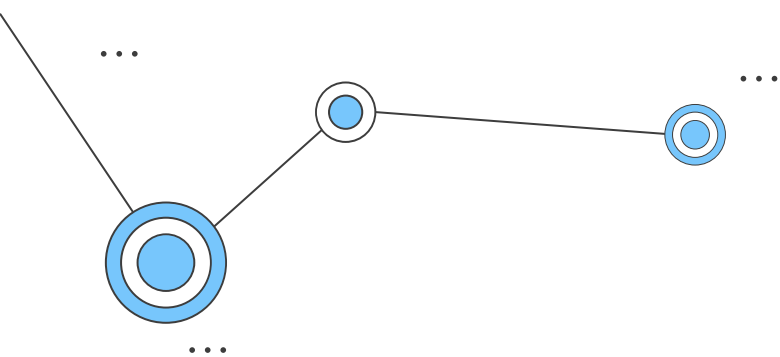


BASIS DATA LANJUT

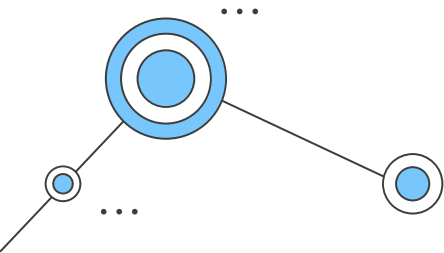
Pertemuan 07
Transaction & Concurrency





OUTLINE

- Transaction
- COMMIT, ROLLBACK
- ACID Properties
- Concurrency Problem
- Locking
- TRIGGER





Latar Belakang

- Banyak user mengakses basis data secara bersamaan
- Jika tidak diatur akan timbul masalah: inconsistency, update hilang dll
- Mekanisme transaction melindungi *shared resources* (data) terhadap akses simultan oleh beberapa proses



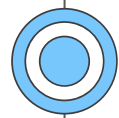
Transaction

- **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

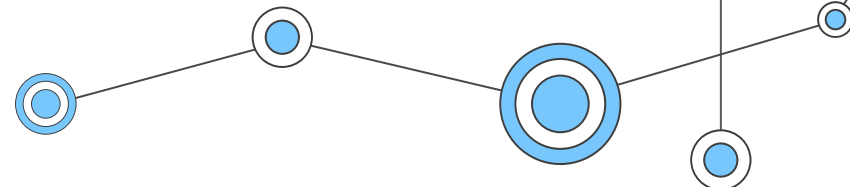
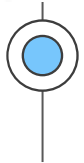
Perintah	Keterangan
BEGIN	memulai transaksi, semua query setelah BEGIN akan dianggap satu kesatuan transaksi sampai ada COMMIT atau ROLLBACK.
COMMIT	Menyimpan semua perubahan transaksi ke database secara permanen. Setelah COMMIT, data tidak bisa dibatalkan dengan ROLLBACK.
ROLLBACK	Membatalkan semua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

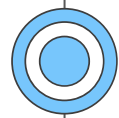
- **COMMIT**

```
begin;  
-- perintah insert/update/delete  
update rekening  
set saldo = saldo + 100000  
where no_rekening = 'REK000110';  
commit;
```



- **ROLLBACK**

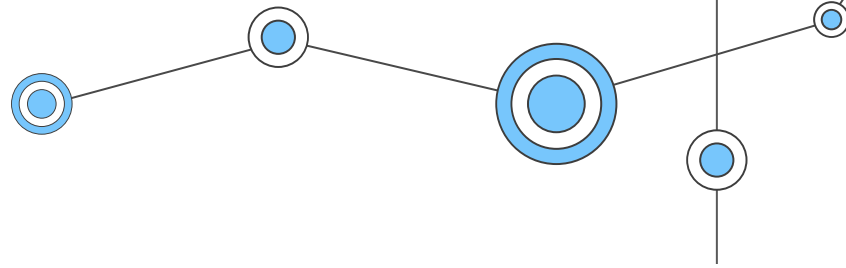
```
begin;  
-- perintah insert/update/delete  
update rekening  
set saldo = saldo + 100000  
where no_rekening = 'REK000110';  
rollback;
```

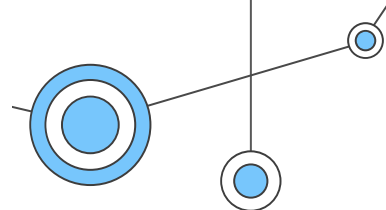


Transaction : SAVEPOINT

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

```
begin;  
-- 1. Operasi pertama  
update ...;  
savepoint s1;  
  
-- 2. Operasi kedua  
update ...;  
savepoint s2;  
  
-- 3. Operasi ketiga (error)  
update...; -- gagal  
  
-- 4. Rollback ke savepoint s2 (batalan langkah 3 saja)  
rollback to savepoint s2;  
  
-- 5. Lanjutkan transaksi  
commit;
```





Transaction : SAVEPOINT

```
BEGIN;  
select  
(  
  -- Operation 1: Transfer A → B (SUKSES)  
  UPDATE rekening SET saldo = saldo - 150000 WHERE nomor_rekening = 'REK000098';  
  UPDATE rekening SET saldo = saldo + 150000 WHERE nomor_rekening = 'REK000149';  
  
  -- Buat savepoint  
  SAVEPOINT after_transfer_1;  
  
  -- Operation 2: Transfer C → D (SUKSES)  
  UPDATE rekening SET saldo = saldo - 250000 WHERE nomor_rekening = 'REK000190';  
  UPDATE rekening SET saldo = saldo + 250000 WHERE nomor_rekening = 'REK000160';  
  
  -- Buat savepoint  
  SAVEPOINT after_transfer_2;  
  
  -- Operation 3: Transfer dengan error (akan rollback ke savepoint)  
  UPDATE rekening SET saldo = saldo - 10000000 WHERE nomor_rekening = 'REK000095'; -- Error  
  
  -- Jika error, rollback ke savepoint  
  ROLLBACK TO SAVEPOINT after_transfer_2;  
  
  RAISE NOTICE 'Rollback ke after_transfer_2, operation 3 dibatalkan';  
  
  -- Commit operation 1 dan 2 saja  
  COMMIT;
```

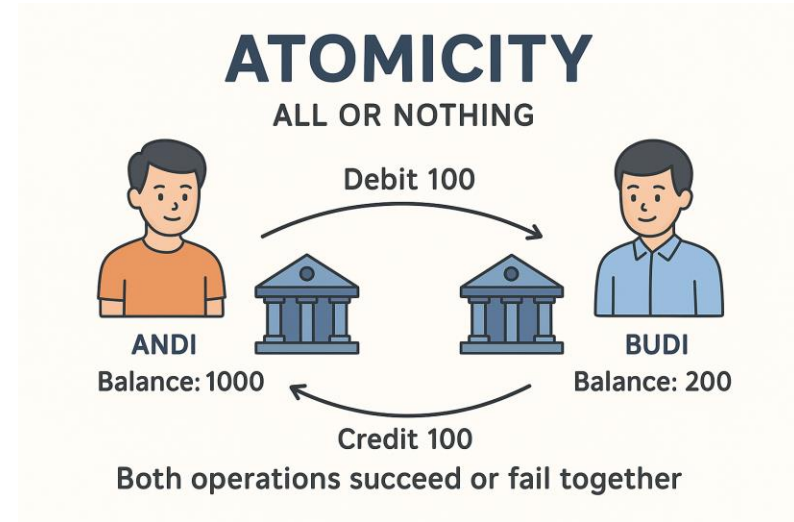
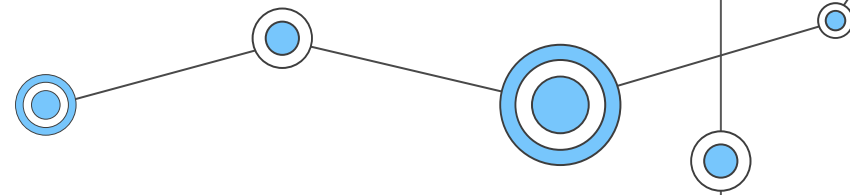



ACID Properties

- 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
- ACID: Atomicity, Consistency, Isolation, Durability

ACID Properties: Atomicity

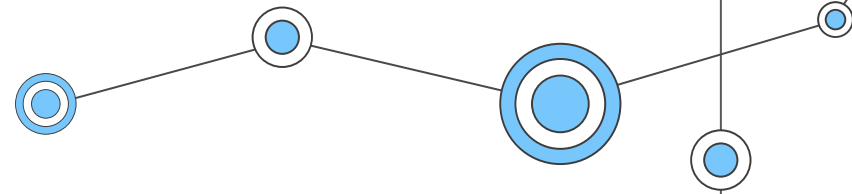
- Memastikan bahwa seluruh operasi dalam satu transaksi dianggap sebagai satu kesatuan yang utuh (***all – or – nothing***)
- Jika semua berhasil, maka transaksi COMMIT
- Jika ada yang gagal, maka semua perubahan dibatalkan (ROLLBACK)





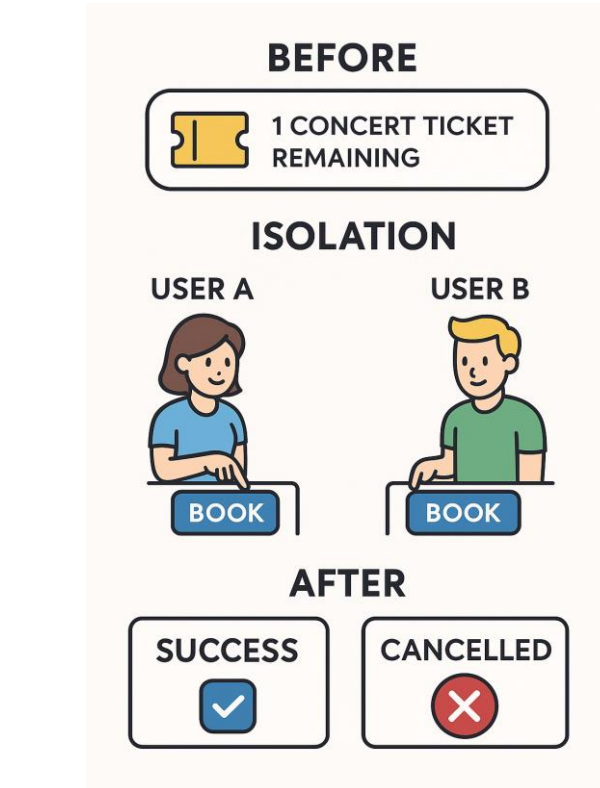
ACID Properties: Consistency

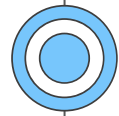
- Transaksi harus membawa database dari satu **state valid** ke **state valid lainnya** sesuai aturan bisnis, constraint, dan integritas data.
- Mencegah data keluar dari aturan integritas (misalnya saldo tidak boleh negatif, constraint primary key harus unik).



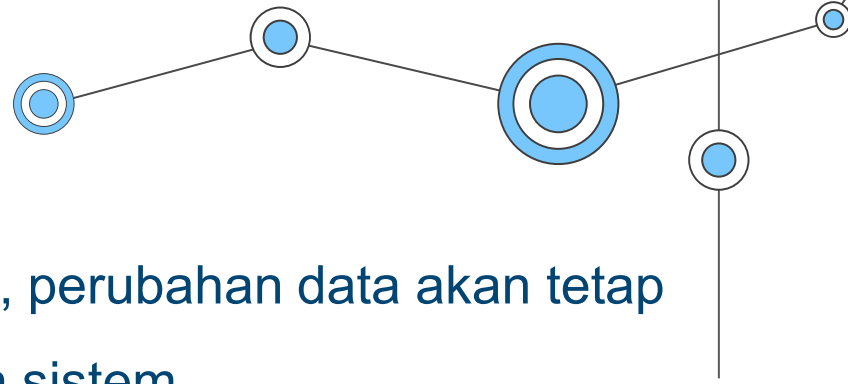
ACID Properties: Isolation

- Transaksi berjalan **seolah-olah sendiri**, tanpa gangguan transaksi lain
- Menghindari anomali
concurrency: dirty read, non-repeatable read, phantom read, lost update





ACID Properties: Durability



- Setelah transaksi berhasil (commit), perubahan data akan tetap tersimpan meskipun ada kegagalan sistem.
- Menjamin data yang sudah di-commit tidak hilang





Concurrency

- 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

Masalah Concurrency: 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 uncommitted value written by Transaction B	

Masalah Concurrency: Lost Update

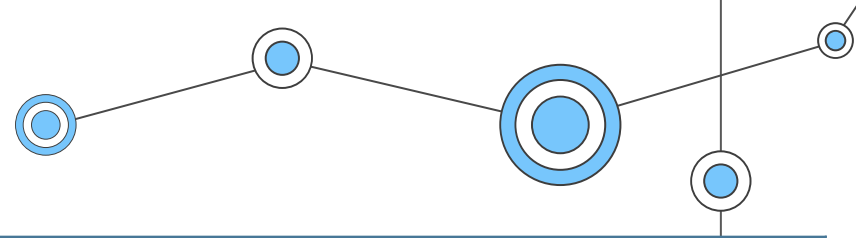
Terjadi ketika dua transaksi membaca data yang sama, lalu sama-sama mengubahnya, tetapi update terakhir menimpa hasil update pertama

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	

Masalah Concurrency: 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	



Masalah Concurrency: 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)	-



Isolation Level

- Disebut juga kebijakan transaksi
- Sejauh mana satu transaksi boleh “melihat” perubahan dari transaksi lain yang sedang berjalan.
- Tujuannya menentukan seberapa ketat isolasi data antar transaksi untuk menghindari anomali seperti: Dirty Read, Non-repeatable Read, Phantom Read



Isolation Level: READ UNCOMMITTED

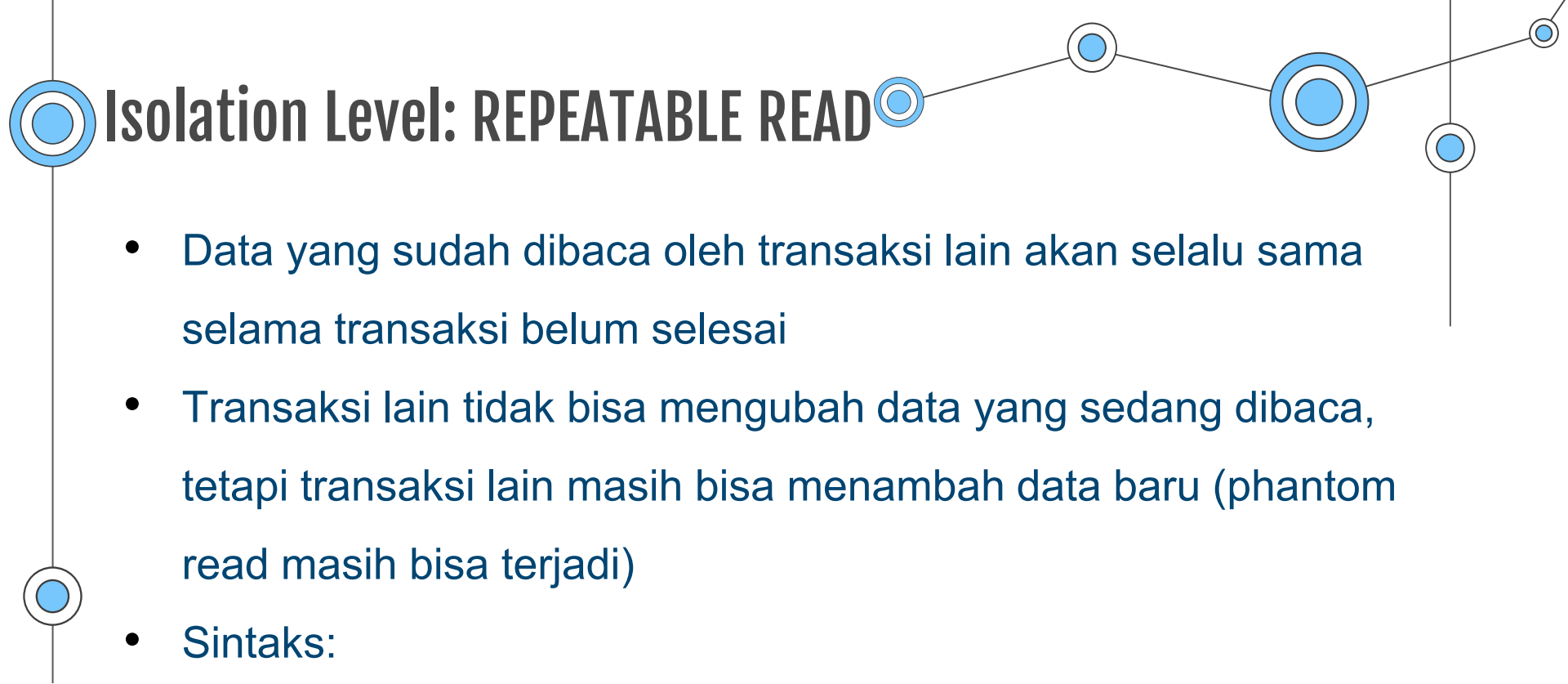
- Isolation level terendah
- Transaksi boleh **membaca data yang belum di-commit** oleh transaksi lain → bisa muncul **dirty read**.
- PostgreSQL tidak mendukung jenis isolation level ini



Isolation Level: READ COMMITTED

- Transaksi hanya bisa membaca data yang sudah di-commit
- Tapi jika baris data dibaca dua kali, nilai bisa berubah (non-repeatable read)
- Sintaks:

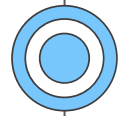
```
SET TRANSACTION ISOLATION LEVEL READ COMMITTED;
```



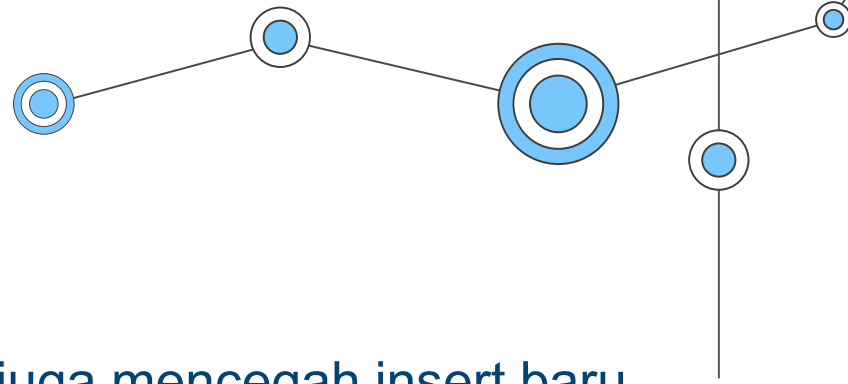
Isolation Level: REPEATABLE READ

- Data yang sudah dibaca oleh transaksi lain akan selalu sama selama transaksi belum selesai
- Transaksi lain tidak bisa mengubah data yang sedang dibaca, tetapi transaksi lain masih bisa menambah data baru (phantom read masih bisa terjadi)
- Sintaks:

```
SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;
```



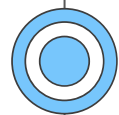
Isolation Level: SERIALIZABLE



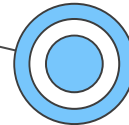
- Level tertinggi (paling ketat)
- Selain mencegah perubahan data, juga mencegah insert baru di range yang dibaca
- Sintaks:

```
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;
```





Perbandingan Isolation Level

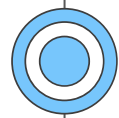


Isolation Level	Dirty Read	Non Repeatable Read	Phantom Read	Kinerja	Keamanan Data
Read Uncommitted	V	V	V	Paling cepat	Rendah
Read Committed (PostgreSQL default)	X	V	V	Lebih cepat dari read uncommitted	Cukup aman
Repeatable Read	X	X	V	Lebih cepat dari read committed	Lebih aman
Serializable	X	X	X	Paling lambat	Sangat aman



Locking

- **Locking** adalah mekanisme untuk **mencegah konflik antar transaksi** yang mengakses data yang sama pada saat bersamaan.
- Row-level Lock dan Tabel-Level Lock



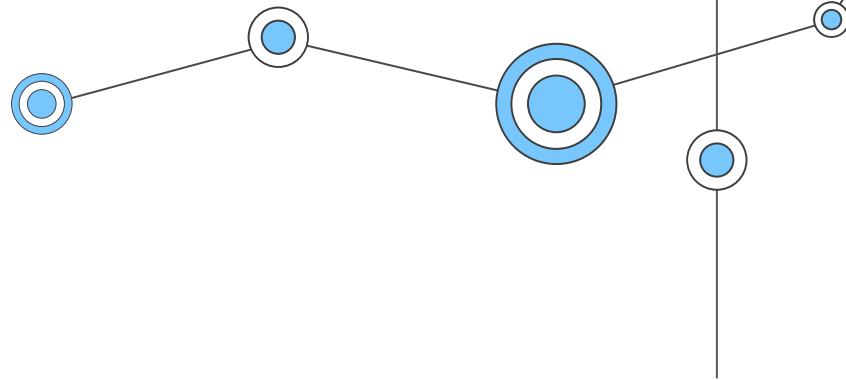
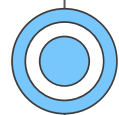
Locking: Row – Level Lock

- Mengunci baris tertentu dalam tabel, Namun transaksi lain tetap bisa mengakses baris lain di tabel yang sama
- Digunakan saat operasi DML: UPDATE, DELETE, SELECT
- Kelebihan: efisien, cocok untuk banyak transaksi kecil
- Kekurangan: bisa terjadi deadlock kalau dua transaksi saling menunggu baris berbeda



Locking: Row – Level Lock (1)

Jenis Lock	Tujuan	Keterangan
FOR UPDATE	Mengunci baris untuk update/delete	Transaksi lain tidak bisa ubah/hapus baris
FOR NO KEY UPDATE	Seperti FOR UPDATE, lebih ringan (tidak melindungi foreign key)	Cocok untuk mengubah nilai non-key
FOR SHARE	Mengunci baris agar tidak dihapus, tapi masih bisa dibaca	Cocok untuk membaca data yang pasti digunakan
FOR KEY SHARE	Paling ringan, hanya mencegah penghapusan baris	Bisa dipakai bersama FOR NO KEY UPDATE



Locking: Row – Level Lock (2)

- **Session A**

```
BEGIN;
```

```
SELECT * FROM products WHERE id = 5 FOR UPDATE;
```

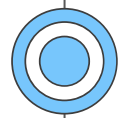
```
-- Baris id=5 dikunci session A, tidak bisa  
update/delete
```

- **Session B**

```
BEGIN;
```

```
UPDATE accounts SET balance=balance+200 WHERE id=1;
```

```
-- Akan "menunggu" sampai Session A COMMIT/ROLLBACK
```



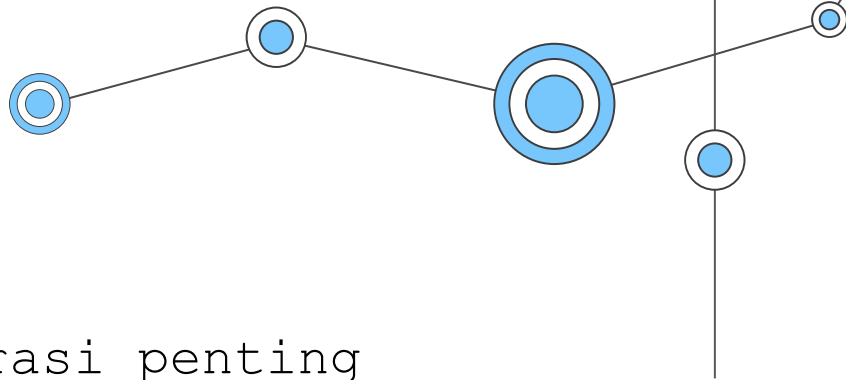
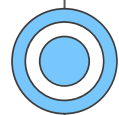
Locking: Table – Level Lock

- Mengunci seluruh tabel selama transaksi berjalan, sehingga transaksi lain tidak bisa membaca atau menulis ke tabel
- Digunakan di beberapa operasi DDL (ALTER, DROP)
- Kelebihan: stabil untuk operasi besar, data sangat aman
- Kekurangan: lambat jika banyak transaksi parallel, sistem tidak responsif



Locking: Table – Level Lock (1)

Jenis Lock	Tujuan	Keterangan
ACCESS SHARE	Semua boleh baca	Digunakan untuk SELECT
ROW EXCLUSIVE	Boleh baca, tidak boleh ubah struktur	Digunakan untuk INSERT/UPDATE/DELETE
EXCLUSIVE	Tidak boleh ada transaksi lain	Operasi besar
ACCESS EXCLUSIVE	Tidak boleh baca/tulis	ALTER/DROP/TRUNCATE



Locking: Table – Level Lock (2)

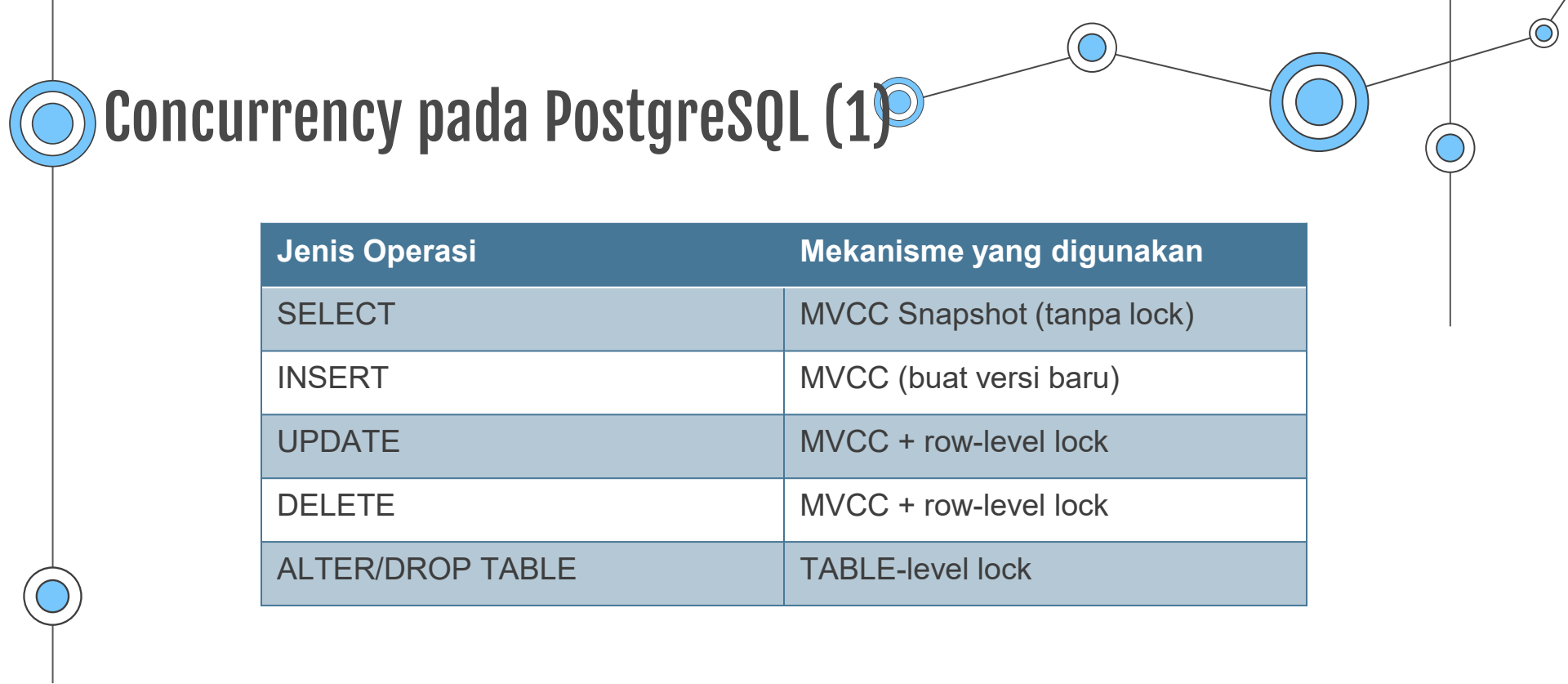
```
BEGIN;  
-- Kunci tabel penuh untuk operasi penting  
LOCK TABLE transactions IN ACCESS EXCLUSIVE MODE;  
  
-- Lakukan perubahan besar  
DELETE FROM transactions WHERE created_at<'2024-01-01';  
COMMIT;
```

Semua transaksi lain yang ingin baca/ubah tabel transaction harus menunggu sampai commit



Concurrency pada PostgreSQL

- Mekanisme concurrency pada PostgreSQL menggunakan Locking dan MVCC (Multi-version Concurrency Control)
- Dengan MVCC, setiap transaksi di PostgreSQL **melihat snapshot** (versi data sesuai waktunya)
- Saat transaksi update, PostgreSQL tidak menimpa baris lama, tapi membuat versi baru baris itu, versi lama akan di VACUUM jika tidak dibutuhkan lagi
- PostgreSQL menggunakan MVCC untuk menghindari blocking saat *read*, dan menggunakan locking untuk menjaga integritas saat *write*.



Concurrency pada PostgreSQL (1)

Jenis Operasi	Mekanisme yang digunakan
SELECT	MVCC Snapshot (tanpa lock)
INSERT	MVCC (buat versi baru)
UPDATE	MVCC + row-level lock
DELETE	MVCC + row-level lock
ALTER/DROP TABLE	TABLE-level lock



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.

TRIGGER (1)

- **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 (1)

- 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)

Sintaks Trigger

```
create or function nama_function()  
returns trigger as $$  
begin  
    -- Logika trigger di sini  
    return [new | old];  
end;  
$$ language plpgsql;
```

```
create trigger nama_trigger{ before | after | instead of }  
{ insert | update | delete | truncate } on  
nama_tabel[ for each row | for each statement ]  
execute function nama_function();
```

TRIGGER untuk Logging INSERT




```
-- step 1: buat tabel log
create table audit_log (
    id serial primary key, action text, table_name text,
    changed_at timestamp default now());

-- step 2: buat function untuk trigger
create or replace function log_insert()
returns trigger as $$
begin
    insert into audit_log(action, table_name)
    values ('insert', tg_table_name);
return new; -- return new artinya data tetap masuk ke tabel utama
end;$$ language plpgsql;

-- step 3: buat trigger
create trigger trg_log_insert
after insert on users
for each row execute function log_insert();
```



TRIGGER VALIDASI (mencegah saldo negative)



```
-- step 1: buat function validasi
create or replace function prevent_negative_balance()
returns trigger as $$
begin
    if new.balance < 0 then
        raise exception 'saldo tidak boleh negatif!';
    end if;
    return new; -- return new untuk update data
end;
$$ language plpgsql;

-- step 2: buat trigger
create trigger trg_prevent_negative
before update on accounts
for each row
execute function prevent_negative_balance();
```

Kesimpulan

- ❑ Transaction harus memenuhi prinsip ACID (Atomicity, Consistency, Isolation, Durability)
- ❑ Perintah dasar transaction BEGIN, COMMIT, ROLLBACK, SAVEPOINT
- ❑ Masalah concurrency: Dirty Read, Non-repeatable Read, Phantom Read, diatasi oleh isolation level: read uncommitted, read committed, repeatable read, serializable
- ❑ Concurrency pada PostgreSQL menggunakan MVCC dan locking (row-level dan table-level)
- ❑ Trigger adalah prosedur otomatis yang dijalankan oleh database saat event tertentu terjadi pada tabel atau view

Thanks!

Do you have any questions?



Team Teaching Matakuliah Basis Data Lanjut
JTI POLINEMA