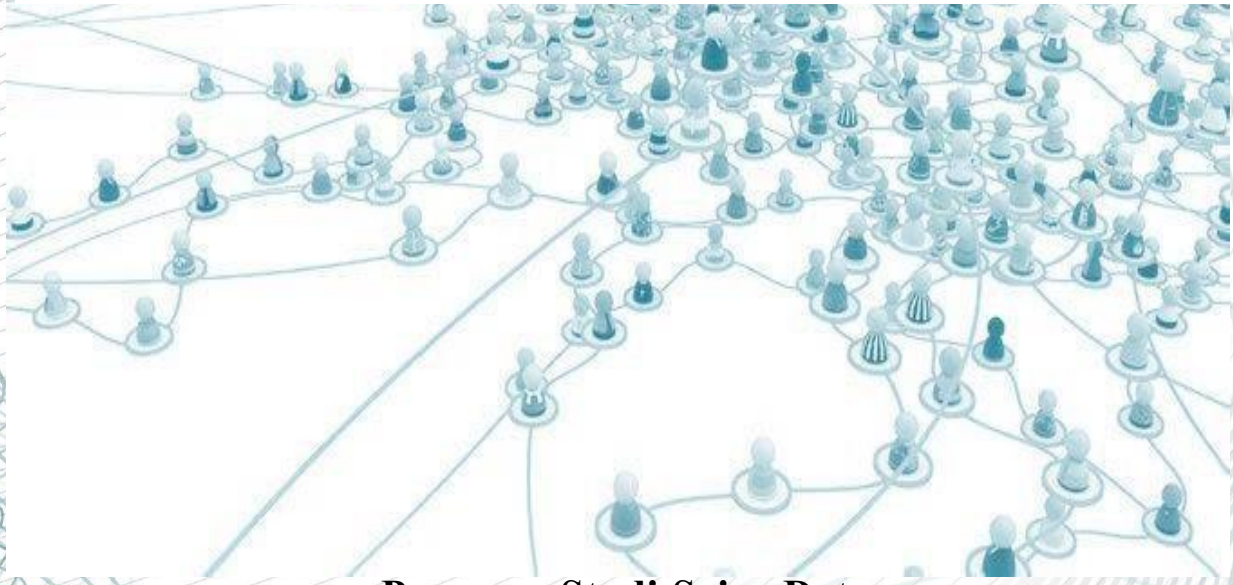
MODUL PRAKTIKUM

SD25-31002-Teknologi Basis Data



Program Studi Sains Data
Fakultas Sains
Institut Teknologi Sumatera

2025

MODUL 5

Concurrency Control
(KONTROL KONKURENSI)

Concurrency Control
(KONTROL KONKURENSI)

1. Tujuan Praktikum

a. Mahasiswa mampu menjelaskan konsep dasar kontrol konkurensi pada sistem basis data multiuser.
b. Mahasiswa mampu mengidentifikasi permasalahan yang dapat muncul akibat transaksi bersamaan, seperti lost update, dirty read, non-repeatable read, dan phantom read.
c. Mahasiswa mampu menerapkan mekanisme locking (shared lock, exclusive lock) dan isolation level menggunakan MySQL di XAMPP.

2. Konsep Dasar

Kontrol konkurensi adalah koordinasi pelaksanaan transaksi simultan dalam sistem database multiuser. Tujuan dari concurrency control adalah untuk memastikan serialisasi transaksi dalam kontrol environment ketika terjadi banyak transaksi pengguna basis data (multi user transactions). Serializability dalam konteks transaksi basis data adalah sebuah konsep untuk memastikan bahwa urutan eksekusi transaksi secara paralel (konkurensi) menghasilkan hasil yang setara dengan menjalankan transaksi tersebut secara berurutan (serial), tanpa adanya konflik atau inkonsistensi dalam data.

Concurrency database penting karena eksekusi simultan dalam sistem multiuser (contoh: banyak user mengakses tabel atau field yang sama) melalui database dapat menciptakan beberapa integritas data dan konsistensi.
masalah utama yang mungkin timbul adalah:

a) update hilang,
b) data uncommitted,
c) retrievals tidak konsisten

Kondisi data berkaitan konkurensi dan penguncian data

1. **Lost updates**

   Terjadi ketika dua transaksi atau lebih membaca data yang sama, lalu masing-masing memperbarui data tersebut, tetapi pembaruan transaksi pertama hilang karena ditimpa oleh transaksi berikutnya.

   Skenario:

   - Ada pegawai dengan EMP_NO = 10001 dan gaji sebesar 88.958 USD di tabel **SALARIES**.
   - **Transaksi A (HRD):** ingin menambahkan tunjangan sebesar 1.000 USD.
   - **Transaksi B (Manajer):** ingin menaikkan gaji sebesar 10.000 USD.
   - Keduanya membaca gaji lama (88.958 USD), lalu melakukan pembaruan secara

bersamaan.

No	Transaksi A	Transaksi B	Hasil
T1	SELECT salary FROM salaries WHERE emp_no = 10001;		88.958
T2		SELECT salary FROM salaries WHERE emp_no = 10001;	88.958
T3	UPDATE salaries SET salary = 89958 WHERE emp_no = 10001;		89.958
T4		UPDATE salaries SET salary = 98958 WHERE emp_no = 10001;	98.958

Hasil akhir: gaji = 98.958 USD, padahal seharusnya: 88.958 + 1.000 + 10.000 = 99.958 USD.
Update yang dilakukan oleh HRD hilang.

2. **Uncommitted Data (Dirty Read)**
   Terjadi ketika suatu transaksi membaca data yang **belum di-commit** oleh transaksi lain. Jika transaksi lain itu melakukan rollback, maka data yang sudah terbaca tadi **tidak valid**.
   Skenario:
   - **Transaksi A:** sedang mengupdate nama departemen di tabel **DEPARTMENTS** dari *Finance → Financial Management*, tetapi belum commit.
   - **Transaksi B:** membaca data tersebut sebelum A melakukan commit.
   - Lalu **Transaksi A** melakukan rollback.

No	Transaksi A	Transaksi B	Nama departments
T1	UPDATE departments SET dept_name = 'Financial Management' WHERE dept_no = 'd002';		Financial Management
T2		SELECT dept_name FROM departments WHERE dept_no = 'd002';	Financial Management
T3	ROLLBACK;		Finance

3. **Unrepeatable Reads**

Terjadi ketika dalam **satu transaksi**, kita membaca data yang sama dua kali, tetapi hasilnya berbeda karena transaksi lain **memodifikasi dan commit data tersebut** di antara dua pembacaan.

Skenario:

- **Transaksi A:** ingin melihat gaji pegawai 10004.
- **Transaksi B:** di tengah-tengah, menaikkan gaji pegawai tersebut dan commit.
- **Transaksi A:** membaca ulang gaji pegawai, hasilnya berubah.

No	Transaksi A	Transaksi B	Nama departments
T1	SELECT salary FROM salaries WHERE emp_no = 10004;		74,057
T2		UPDATE salaries SET salary = 5000000 WHERE emp_no = 10004; COMMIT;	50,000
T3	SELECT salary FROM salaries WHERE emp_no = 10004;		50,000

Dalam **satu transaksi**, Transaksi A mendapatkan hasil yang **berbeda** ketika membaca data yang sama lebih dari sekali.

Fenomena ini disebut ***Unrepeatable Read***, yaitu ketika data yang dibaca oleh suatu transaksi berubah karena telah dimodifikasi dan dikomit oleh transaksi lain di antara dua pembacaan tersebut. Sehingga dalam satu transaksi, Transaksi A mendapat hasil berbeda **→ unrepeatable read.**

4. **Phantom Reads**

Terjadi ketika suatu transaksi membaca sekumpulan data dengan kondisi tertentu, lalu transaksi lain **menambah atau menghapus** baris data yang memenuhi kondisi itu, sehingga pada pembacaan berikutnya, hasil query **berbeda jumlah barisnya**.

Skenario:

- **Transaksi A:** menjalankan query untuk melihat semua pegawai di departemen d003 (Human Resources).
- **Transaksi B:** menambahkan pegawai baru ke departemen d003 lalu commit.
- **Transaksi A:** menjalankan query yang sama lagi, jumlah baris hasil berubah.

No	Transaksi A	Transaksi B	Nama departments
T1	SELECT * FROM		17,786

	dept_emp WHERE dept_no = 'd003';		
T2		INSERT INTO dept_emp (emp_no, dept_no, from_date, to_date) VALUES (11000, 'd003', '2025-01-01', '9999-01-01'); COMMIT;	
T3	SELECT * FROM dept_emp WHERE dept_no = 'd003';		

Hasil query berubah jumlah barisnya → **phantom read.**

SOLUSI

Kondisi Data	Penyebab	Solusi / Isolation Level	Solusi / Isolation Level
Dirty Read	Terjadi ketika transaksi membaca data yang sedang diubah oleh transaksi lain yang belum di-commit.	READ COMMITTED atau gunakan Shared Lock (S-Lock).	Isolation level ini memastikan transaksi hanya bisa membaca data yang sudah di-commit, sehingga tidak ada pembacaan terhadap data sementara yang mungkin dibatalkan.
Non-Repeatable Read	Terjadi ketika transaksi membaca data yang sama lebih dari sekali, tetapi data tersebut telah diubah atau dihapus oleh transaksi lain yang sudah commit.	REPEATABLE READ atau gunakan Shared Lock selama transaksi.	Isolation level ini menjaga agar baris yang sudah dibaca tetap terkunci (read lock) sampai transaksi selesai, sehingga tidak bisa diubah oleh transaksi lain di tengah proses.
Phantom Read	Terjadi ketika transaksi membaca sekumpulan data, kemudian	SERIALIZABLE atau gunakan Range Lock.	Level ini mengunci rentang data (range) yang dibaca, bukan hanya baris tertentu, sehingga tidak ada baris baru yang bisa

	transaksi lain menambah atau menghapus baris baru yang memenuhi kondisi tersebut.		ditambahkan atau dihapus selama transaksi berlangsung.
Lost Update	Terjadi ketika dua transaksi membaca nilai yang sama dan melakukan update berdasarkan nilai awal yang sama. Salah satu hasil update menimpa yang lain.	Exclusive Lock (X-Lock) atau Optimistic Concurrency Control (OCC).	Exclusive lock mencegah dua transaksi mengubah data yang sama secara bersamaan. Sementara OCC memeriksa apakah data telah berubah sebelum update dilakukan, untuk menghindari penimpaan data

Mekanisme Kontrol Konkruensi

- **Optimis –** Penundaan pengecekan apakah transaksi memenuhi isolasi dan aturan integritas lainnya (misalnya *serializability* dan *recovery*) dilakukan hingga akhir, tanpa memblokir transaksi yang membaca atau menulis. Jika aturan yang diinginkan dilanggar, maka transaksi akan dibatalkan untuk mencegah pelanggaran sebelum dilakukan *commit*. Sebuah transaksi yang dibatalkan akan segera di-*restart* dan dijalankan kembali, yang dapat menimbulkan *overhead* karena harus mengeksekusi ulang hingga selesai dan hanya sekali. Jika jumlah transaksi yang dibatalkan tidak terlalu banyak, maka mekanisme optimis biasanya menjadi strategi yang baik
- **Pesimis** – Operasi transaksi diblokir jika berpotensi menimbulkan pelanggaran aturan, dan akan tetap diblokir hingga kemungkinan pelanggaran tersebut hilang. Pemblokiran operasi ini biasanya berdampak pada penurunan kinerja.
- **Semi-optimis** – Operasi diblokir pada beberapa transaksi jika berpotensi menyebabkan pelanggaran terhadap aturan tertentu, namun tidak diblokir dalam situasi lain. Pemeriksaan aturan (jika diperlukan) akan ditunda hingga akhir transaksi, seperti yang dilakukan pada mekanisme optimis.

a. Concurrency Control Method : Optimistic

- Penundaan pengecekan apakah transaksi memenuhi isolasi dan aturan integritas lainnya (misalnya serializability dan recovery) sampai akhir,tanpa memblokir transaksi yang membaca atau menulis.
- Membatalkan transaksi untuk mencegah pelanggaran,jika aturan yang diinginkan harus dilanggar atas yang COMMIT

b. Concurrency Control Method : Pessimistic

- Melakukan blok operasi transaksi jika dapat menimbulkan pelanggaran aturan, sampai kemungkinan pelanggaran menghilang.
- Memblokir operasi biasanya akan berakibat pada pengurangan kinerja.

c. Concurrency Control Method : Semi Optimistic

- Melakukan blok operasi dalam beberapa transaksi, jika transaksi tersebut dapat menyebabkan pelanggaran beberapa aturan.
- Tidak memblokir dalam situasi lain menggunakan aturan yang akan menunda pemeriksaan (jika diperlukan) sampai akhir transaksi ini, seperti yang dilakukan dengan mekanisme optimis.

Dalam konteks development aplikasi yang scalable, implementasi mekanisme locking menjadi semakin penting untuk memastikan correctness and consistency data. Tanpa mekanisme ini, terdapat risiko tinggi terjadinya race condition, di mana multiple request saling berebut untuk mengakses sumber daya/resource yang sama secara bersamaan. **Mekanisme locking** dapat diterapkan pada berbagai tingkatan dalam arsitektur aplikasi. Salah satu contoh penggunaannya adalah pada level database, di mana locking digunakan untuk memastikan bahwa operasi yang dilakukan oleh satu transaksi tidak terganggu oleh transaksi lain yang berusaha mengakses atau memodifikasi data yang sama. Dalam hal ini, mekanisme locking digunakan untuk mengontrol akses konkurensi ke data dan memastikan bahwa operasi transaksi dilakukan secara terurut dan terisolasi.

Selain itu, mekanisme locking juga sering digunakan dalam sistem caching untuk menghindari masalah yang disebabkan oleh konsistensi data. Dalam skenario caching, ketika sebuah data diminta, sistem akan mencoba mencarinya di cache terlebih dahulu sebelum mengakses sumber data asli, seperti database. Jika data tersebut ada di cache, maka proses pengambilan data menjadi lebih cepat. Namun, jika beberapa request datang secara bersamaan dan mencoba mengambil data yang sama yang belum tersedia di cache, maka mekanisme locking akan digunakan untuk memastikan hanya satu request yang mengakses sumber data asli dan melakukan proses caching. Hal ini menghindari duplikasi pengambilan data dan memastikan konsistensi data yang ditampilkan kepada pengguna.

3. Latihan Praktikum

A. Soal 1 – Lost Update

Seorang pegawai bernama Andi memiliki gaji sebesar 8.000.000.
Dua transaksi dilakukan bersamaan:
Transaksi A (HRD) ingin menaikkan gaji Andi sebesar 500.000.
Transaksi B (Manajer) ingin menaikkan gaji Andi sebesar 1.000.000.

B. Soal 2 – Dirty Read

Transaksi A mengubah gaji pegawai Budi dari 8.500.000 menjadi 10.000.000, tetapi belum melakukan commit.
Transaksi B membaca data gaji Budi dengan isolation level READ UNCOMMITTED.

Kemudian, Transaksi A melakukan rollback.

C. Soal 3 – Non-Repeatable Read

Transaksi A ingin membaca data gaji pegawai Citra dua kali dalam satu transaksi.
Sementara itu, Transaksi B mengubah gaji Citra dari 9.000.000 menjadi 9.500.000 dan melakukan commit di tengah transaksi A.

D. Soal 4 – Phantom Read

Transaksi A membaca semua pegawai dengan gaji di atas 8.000.000.
Di tengah proses, Transaksi B menambahkan pegawai baru bernama Dewi dengan gaji 8.500.000 dan melakukan commit.

Adapun langkah - langkah praktikum sebagai berikut.

1. Langkah 1 Persiapan

- Jalankan XAMPP Control Panel, aktifkan Apache dan MySQL
- Buka PHPMyAdmin di browser: http://localhost/phpmyadmin
- Pastikan halaman utama PHPMyAdmin terbuka.
- Pilih tab **Database** → klik **Create Database**, beri nama:

```
db_konkruensi
```

- Klik **Create**.
- Pastikan database db_konkruensi aktif.
- Klik tab **SQL** → salin dan jalankan kode berikut:

```
CREATE TABLE pegawai (
    id INT AUTO_INCREMENT PRIMARY KEY,
    nama VARCHAR(50),
    gaji DECIMAL(10,2)
);
```

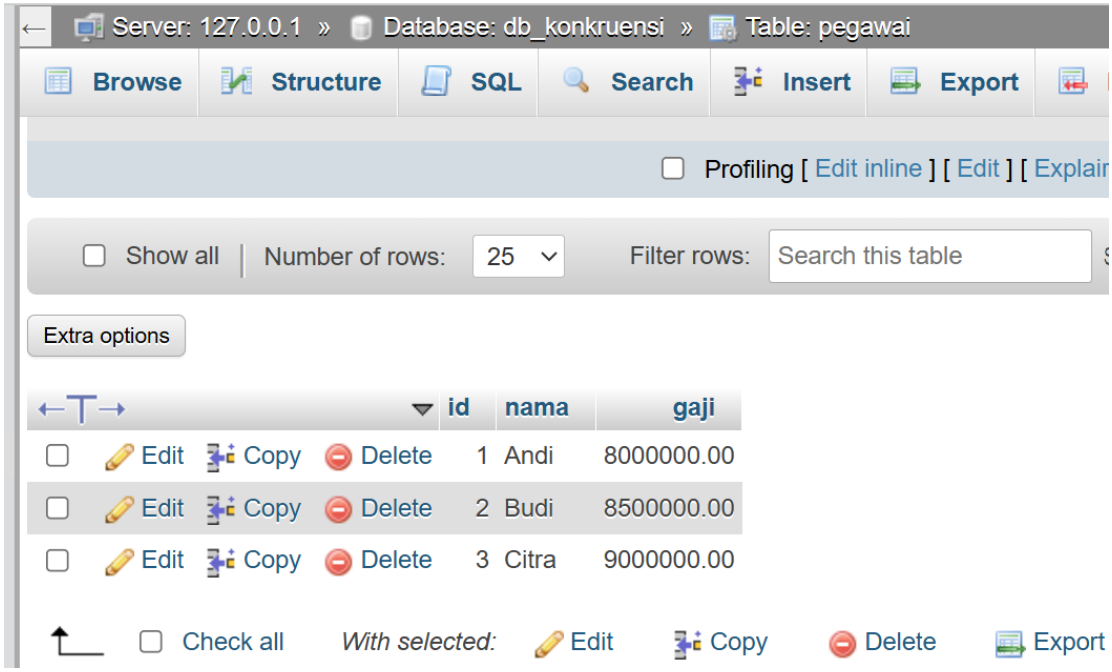
- Setelah tabel berhasil dibuat, masukkan data awal:

```
INSERT INTO pegawai (nama, gaji) VALUES
('Andi', 8000000),
('Budi', 8500000),
('Citra', 9000000);
```

- Jalankan query:

```
SELECT * FROM pegawai;
```

Hasil awal:



- **Menyiapkan Dua Tab Transaksi**
  Untuk mensimulasikan kontrol konkurensi, kita membutuhkan **dua koneksi berbeda** (dua tab MySQL aktif):
  a) Klik kanan tab browser PHPMyAdmin → pilih **Duplicate Tab**.
  b) Ubah nama tab agar mudah membedakan:
  - Tab 1: **Transaksi A**
  - Tab 2: **Transaksi B**

  Kedua tab ini akan digunakan untuk menjalankan transaksi secara bersamaan.

2. **Langkah 2 – Simulasi Lost Update**
   Mensimulasikan kondisi Lost Update, yaitu ketika dua transaksi membaca data yang sama, memperbarui secara bersamaan, dan salah satu pembaruan hilang karena tumpang tindih commit.
   Skenario Kasus:
   Pegawai bernama Andi memiliki gaji awal sebesar Rp8.000.000.
   - Transaksi A (HRD) menaikkan gaji Andi sebesar Rp500.000.
   - Transaksi B (Manajer) menaikkan gaji Andi sebesar Rp1.000.000.
     Kedua transaksi dijalankan bersamaan tanpa locking.

   a) Langkah 2.1 – Cek Data Awal
   Buka salah satu tab (boleh Transaksi A atau B):

```
SELECT * FROM pegawai WHERE nama='Andi';
```

b) Langkah 2.2 – Jalankan Transaksi A (HRD)
Di Tab 1 (Transaksi A) jalankan:

```
START TRANSACTION;
SELECT gaji FROM pegawai WHERE nama='Andi';
-- hasil: 8000000
UPDATE pegawai SET gaji = 8500000 WHERE nama='Andi';
-- belum di-COMMIT
```

Transaksi A telah memperbarui nilai di memori transaksi, tapi belum disimpan ke database (belum commit).

c) Langkah 2.3 – Jalankan Transaksi B (Manajer)
Tanpa menunggu Transaksi A selesai, buka Tab 2 (Transaksi B)**:**

```
START TRANSACTION;
SELECT gaji FROM pegawai WHERE nama='Andi';
-- hasil: 8000000 (masih nilai lama)
UPDATE pegawai SET gaji = 9000000 WHERE nama='Andi';
COMMIT;
```

Transaksi B selesai dan menyimpan data ke database → gaji Andi sekarang 9.000.000.

d) Langkah 2.4 – Lanjutkan Transaksi A
Kembali **ke** Tab 1 (Transaksi A)**:**

```
COMMIT;
```

Transaksi A menyimpan hasil yang masih berdasarkan nilai awal (8.000.000), bukan nilai terbaru.
Gaji Andi kini menjadi 8.500.000 → update Transaksi B tertimpa.

e) Langkah 2.5 – Cek Hasil Akhir

```
SELECT * FROM pegawai WHERE nama='Andi';
```

Hasil akhir: Andi memiliki gaji sebesar Rp8.500.000
Seharusnya hasil benar adalah Rp9.500.000 (gabungan dua update), tetapi update Transaksi B hilang (tertimpa oleh A).

3. Langkah 3 – Simulasi Dirty Read
Skenario Kasus:
Pegawai bernama Budi memiliki gaji awal Rp8.500.000.

- Transaksi A (HRD) sedang mengubah gaji Budi menjadi Rp10.000.000, tetapi belum commit (masih dalam proses editing).
- Transaksi B (Keuangan) membaca gaji Budi sebelum Transaksi A melakukan commit.
- Transaksi A kemudian membatalkan perubahan (rollback). Namun, Transaksi B sudah terlanjur membaca nilai yang belum valid → Dirty Read.

a) Langkah 3.1 – Cek Data Awal

Buka salah satu tab (Transaksi A atau B) di PHPMyAdmin, lalu jalankan:

```
SELECT * FROM pegawai WHERE nama='Budi';
```

b) Langkah 3.2 – Jalankan Transaksi A (HRD)

Di Tab 1 (Transaksi A) jalankan:

-
```
START TRANSACTION;
UPDATE pegawai SET gaji = 10000000 WHERE nama='Budi';
-- belum di-commit
```

Transaksi ini belum disimpan ke database utama — data "sementara" hanya tersimpan di buffer transaksi A.

c) Langkah 3.3 – Jalankan Transaksi B (Keuangan)

Tanpa menunggu Transaksi A commit, buka Tab 2 (Transaksi B):

```
SET TRANSACTION ISOLATION LEVEL READ
UNCOMMITTED;
START TRANSACTION;
SELECT gaji FROM pegawai WHERE nama='Budi';
COMMIT;
```

Transaksi B membaca nilai Rp10.000.000, padahal Transaksi A belum menyimpan perubahan tersebut.

d) Langkah 3.4 – Transaksi A Membatalkan Perubahan

Kembali ke Tab 1 (Transaksi A):

```
ROLLBACK;
```

Perubahan dibatalkan, nilai gaji Budi kembali ke 8.500.000 di database utama.

e) Langkah 3.5 – Cek Hasil Akhir

Cek kembali data Budi (boleh dari tab mana pun):

```
SELECT * FROM pegawai WHERE nama='Budi';
```

Hasil akhir: Budi memiliki gaji sebesar Rp8.500.000

4. Langkah 4 – Simulasi Non-Repeatable Read

Skenario Kasus:

Pegawai bernama Citra memiliki gaji awal Rp9.000.000.

- Transaksi A (HRD) ingin membaca gaji Citra dua kali dalam satu transaksi (pembacaan pertama dan kedua).
- Di tengah proses, Transaksi B (Manajer) memperbarui gaji Citra menjadi Rp9.500.000 dan melakukan commit.
- Ketika Transaksi A membaca ulang, hasilnya berubah.

Hal ini menunjukkan bahwa data yang dibaca tidak konsisten selama transaksi berlangsung.

a) Langkah 4.1 – Cek Data Awal

Buka salah satu tab (Transaksi A atau B) di PHPMyAdmin, lalu jalankan:

```
SELECT * FROM pegawai WHERE nama='Citra';
```

b) Langkah 4.2 – Jalankan Transaksi A (HRD)

Di Tab 1 (Transaksi A) jalankan:

```
SET TRANSACTION ISOLATION LEVEL READ COMMITTED;
START TRANSACTION;
SELECT gaji FROM pegawai WHERE nama='Citra';
-- Hasil pertama: 9000000
```

Transaksi A belum melakukan **commit**. Ia menyimpan nilai gaji saat pertama kali dibaca.

c) Langkah 4.3 – Jalankan Transaksi B (Manajer)

Buka Tab 2 (Transaksi B) dan jalankan:

```
START TRANSACTION;
UPDATE pegawai SET gaji = 9500000 WHERE nama='Citra';
COMMIT;
```

Transaksi B telah menyimpan nilai baru ke database utama (commit berhasil).

d) Langkah 4.4 – Transaksi A Membaca Ulang

Kembali ke Tab 1 (Transaksi A):

```
SELECT gaji FROM pegawai WHERE nama='Citra';
COMMIT;
```

Hasil:

- Pembacaan pertama: 9.000.000
- Pembacaan kedua: 9.500.000

Data berubah di tengah transaksi → **Non-Repeatable Read terjadi.**

5. Langkah 5 – Simulasi Phantom Read

Skenario Kasus:

Dalam tabel pegawai, terdapat tiga data awal:

id	nama	gaji
1	Andi	8.500.000
2	Budi	8.500.000
3	Citra	9.500.000

Transaksi A (HRD) ingin menampilkan daftar semua pegawai dengan gaji di atas Rp8.000.000. Sementara Transaksi B (Admin) menambahkan pegawai baru bernama Dewi dengan gaji Rp8.500.000 di tengah proses transaksi A.

Ketika Transaksi A menjalankan query ulang, hasilnya bertambah satu baris tanpa mengubah query → **terjadi Phantom Read.**

a) Langkah 5.1 – Cek Data Awal

Jalankan perintah berikut di salah satu tab:

```
SELECT * FROM pegawai;
```

b) Langkah 5.2 – Jalankan Transaksi A (HRD)

Di Tab 1 (Transaksi A) jalankan:

-
```
SET TRANSACTION ISOLATION LEVEL REPEATABLE
READ;
START TRANSACTION;
SELECT * FROM pegawai WHERE gaji > 8000000;
-- Hasil awal: 3 baris (Andi, Budi, Citra)
```

Transaksi A belum melakukan commit. Ia membaca data pegawai dengan gaji > 8.000.000, dan menyimpannya sementara di buffer transaksi.

c) Langkah 5.3 – Jalankan Transaksi B (Admin)

Tanpa menunggu Transaksi A selesai, buka Tab 2 (Transaksi B) dan jalankan:

```
START TRANSACTION;
INSERT INTO pegawai (nama, gaji) VALUES ('Dewi', 8500000);
COMMIT;
```

Transaksi B telah menambahkan baris baru ke database utama.

d) Langkah 5.4 – Jalankan Query Ulang di Transaksi A

Kembali ke **Tab 1 (Transaksi A):**

```
SELECT * FROM pegawai WHERE gaji > 8000000;
COMMIT;
```

Hasil : Terjadi penambahan baris baru (Dewi) padahal query-nya tidak berubah. Fenomena ini disebut Phantom Read, karena “baris baru” muncul di antara dua pembacaan.