MODUL

TRANSACTION

A. TUJUAN

a. Memahami konsep dan urgensi transaksi dalam kehidupan sehari-hari
b. Mampu mengimplementasikan transaksi basis data
c. Mampu menyelesaikan operasi-operasi sensitif dengan memanfaatkan transaksi basis data

B. DASAR TEORI

1. Transaksi Basis Data

Transaksi merupakan serangkaian kelompok dari operasi manipulasi database yang dilakukan seolah-olah sebagai satu unit kerja secara individu.

Contoh kegiatan bertransaksi salah apabila A memiliki saldo di Bank sebesar Rp 10.000.000,00 ingin mentransfer uang ke rekening B sebesar Rp 1.500.000,00. Maka dapat diilustrasikan secara sederhana proses yang terjadi adalah sebagai berikut :

a. Saldo si A dikurangi Rp 1.500.000,00 sehingga saldo akhir si A menjadi sebesar Rp 8.500.000,00
b. Saldo si B bertambah sebesar Rp 1.500.000,00

Jika di antara langkah (1) dan (2) terjadi hal-hal buruk yang tidak diinginkan, misalnya saja mesin ATM crash, terjadi pemadaman listrik, UPS gagal up, disk pada server penuh, atau ada cracker yang merusak infrastruktur jaringan. Maka dapat dipastikan bahwa saldo si A berkurang, tetapi saldo si B tidak bertambah. Hal tersebut merupakan sesuatu yang tidak diharapkan untuk terjadi. Dari ilustrasi di atas, maka dapat disimpulkan bahwa solusi yang tepat adalah memperlakukan perintah -perintah sebagai satu kesatuan operasi. Sederhananya, lakukan semua operasi atau tidak sama sekali. Konsep yang disebut transaksi basis data (database transaction) ini sebenarnya cukup sederhana, antara lain:

a. Tandai bagian awal dan akhir himpunan perintah,
b. Putuskan di bagian akhir untuk mengeksekusi (COMMIT) atau membatalkan (ROLLBACK) semua perintah.

2. Properti Transaksi Basis Data

Dalam transaksi basis data, terdapat properti-properti yang menjamin bahwa transaksi dilaksanakan dengan baik. Properti-properti ini dikenal sebagai ACID (Atomicity, Consistency, Isolation, Durability).

a. Atomicity

Transaksi dilakukan sekali dan sifatnya atomic, artinya merupakan satu kesatuan tunggal yang tidak dapat dipisah, baik itu pekerjaan yang dilaksanakan secara keseluruhan, ataupun tidak satupun, biasa dikenal dengan jargon “all or nothing”.

b. Consistency

Transaksi dilakukan Jika basis data pada awalnya dalam keadaan konsisten, maka pelaksanaan transaksi dengan sendirinya juga harus meninggalkan basis data tetap dalam status konsisten.

c. Isolation
   Isolasi memastikan bahwa secara bersamaan (konkuren) eksekusi transaksi terisolasi dari yang lain.

d. Durability
   Begitu transaksi telah dilaksanakan (di-commit), maka perubahan yang diakibatkan tidak akan hilang atau tahan lama (durable), sekalipun terdapat kegagalan sistem.

3. Penanganan Kesalahan

   Fasilitas penanganan kesalahan (error handling) biasa diperlukan untuk mengantisipasi terjadinya kesalahan pada suatu proses transaksi, sehingga programmer bisa mengatur skenario jika suatu operasi gagal sebagian atau seluruhnya. Secara default skenario dari transaksi adalah AUTO COMMIT dimana seluruh proses yang berhasil dilaksanakan akan secara otomatis secara fisik disimpan dalam database. Jika diinginkan mulai dari posisi tertentu, maka AUTO COMMIT tidak berfungsi, dapat digunakan perintah START TRANSACTION.

   Selanjutnya suatu perintah sesudah pernyataan START TRANSACTION akan ditunda untuk disimpan sampai bertemu pernyataan COMMIT yang akan menyimpan seluruh proses yang tertunda, atau bertemu pernyataan ROLLBACK yang akan membatalkan seluruh proses yang tertunda. Akan tetapi perlu diingat bahwa terdapat beberapa perintah yang tidak dapat di ROLLBACK karena mengandung fungsi COMMIT secara implisit. Perintah–peritah tersebut adalah sebagia berikut :

   - ALTER TABLE
   - BEGIN
   - CREATE TABLE
   - CREATE DATABASE
   - CREATE INDEX
   - DROP DATABASE
   - DROP TABLE
   - DROP INDEX
   - LOAD MASTER DATA
   - LOCK TABLES
   - SET AUTOCOMMIT = 1
   - START TRANSACTION
   - TRUNCATE TABLE
   - UNLOCK TABLES

C. LATIHAN PRAKTIKUM

Sebelum masuk ke scenario, pahami definisi berikut:

a. COMMIT

Perintah COMMIT berfungsi untuk menyelesaikan transaksi dan menyimpan semua perubahan yang dilakukan selama transaksi tersebut secara permanen ke dalam database. Perubahan yang terjadi:

Semua operasi seperti INSERT, UPDATE, atau DELETE yang dilakukan sejak transaksi dimulai akan menjadi bagian dari database yang tidak dapat diubah lagi (kecuali dengan transaksi baru). Perubahan tersebut akan terlihat oleh sesi pengguna lain di dalam database. COMMIT akan melepaskan lock atau kunci yang diterapkan pada baris atau tabel selama transaksi berlangsung.

b. ROLLBACK

Perintah ROLLBACK digunakan untuk membatalkan semua perubahan yang dilakukan selama sebuah transaksi. Perubahan yang terjadi:

Semua operasi INSERT, UPDATE, atau DELETE yang belum di-commit akan dibatalkan. Database akan kembali ke keadaan semula, seperti sebelum transaksi dimulai. Perubahan yang dibatalkan tidak akan terlihat oleh sesi pengguna lain. Sama seperti COMMIT, ROLLBACK juga akan melepaskan semua kunci yang diterapkan selama transaksi.

c. SAVEPOINT

Savepoint adalah titik sementara yang dapat dibuat di dalam sebuah transaksi. Fungsinya mirip dengan "pos pemeriksaan" atau checkpoint.

Perubahan yang terjadi:

Anda dapat membatalkan (meng-rollback) transaksi hingga ke titik SAVEPOINT tertentu, bukan harus membatalkan seluruh transaksi dari awal.

Perintah ini berguna saat Anda ingin membatalkan sebagian operasi tanpa harus mengulang semua operasi yang sudah dilakukan sebelumnya dalam transaksi yang sama.

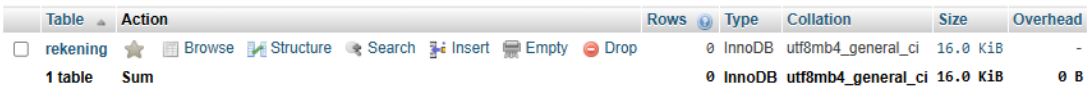
Skenario: Transfer Uang Antar Rekening

Menggunakan skenario transfer uang antar rekening untuk menunjukkan bagaimana transaksi memastikan bahwa semua operasi berhasil atau tidak sama sekali (properti atomicity dalam ACID).

1. Pertama, siapkan tabel rekening di database.

```
CREATE TABLE rekening (
    id_rekening INT PRIMARY KEY,
    nama_pemilik VARCHAR(100),
    saldo DECIMAL(10, 2)
);
```

Hasil :



2. Masukkan beberapa data awal

```
INSERT INTO rekening (id_rekening, nama_pemilik, saldo) VALUES (1, 'Alice', 1000.00);
INSERT INTO rekening (id_rekening, nama_pemilik, saldo) VALUES (2, 'Bob', 500.00);
```

3. Periksa data awal

```
SELECT * FROM rekening;
```

Hasil:

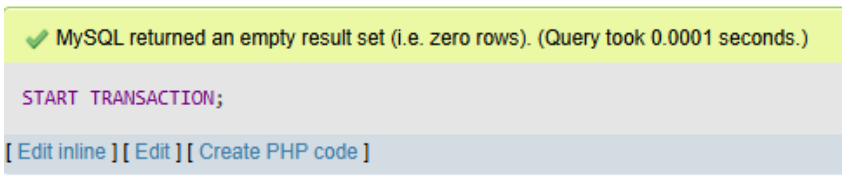
	id_rekening	nama_pemilik	saldo
Edit Copy Delete	1	Alice	1000.00
Edit Copy Delete	2	Bob	500.00

4. Implementasi dengan COMMIT dan ROLLBACK

Secara default, banyak database berjalan dalam mode AUTOCOMMIT. Untuk mengimplementasikan COMMIT dan ROLLBACK, kita harus menonaktifkan mode ini atau memulai transaksi secara eksplisit. Untuk memulai transaksi manual:

```
START TRANSACTION;
```

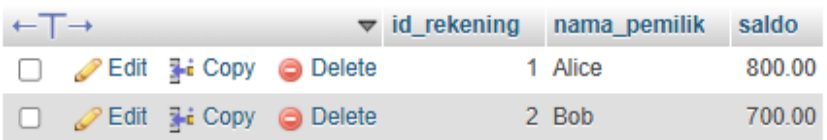
Hasil :



5. Melakukan Transfer yang Sukses (COMMIT)

```
START TRANSACTION;
-- Untuk mengurangi saldo Alice
UPDATE rekening SET saldo = saldo - 200.00 WHERE id_rekening = 1;
-- Untuk menambahkan saldo Bob
UPDATE rekening SET saldo = saldo + 200.00 WHERE id_rekening = 2;
-- Untuk menyimpan perubahan secara permanen
COMMIT;
-- Memeriksa hasilnya setelah COMMIT
SELECT * FROM rekening;
```

Hasil:



	id_rekening	nama_pemilik	saldo
Edit Copy Delete	1	Alice	800.00
Edit Copy Delete	2	Bob	700.00

Setelah menjalankan skrip no.5, saldo Alice akan menjadi 800.00 dan Bob menjadi 700.00. Perubahan ini permanen di database.

6. Melakukan Transfer yang Gagal (ROLLBACK)

```
START TRANSACTION;
-- Mengurangi saldo Alice
UPDATE rekening SET saldo = saldo - 100.00 WHERE id_rekening = 1;
-- Karena ada masalah, batalkan semua perubahan dalam transaksi ini
ROLLBACK;
-- Memeriksa hasilnya setelah ROLLBACK
SELECT * FROM rekening;
```

Hasil:

	id_rekening	nama_pemilik	saldo
Edit Copy Delete	1	Alice	800.00
Edit Copy Delete	2	Bob	700.00

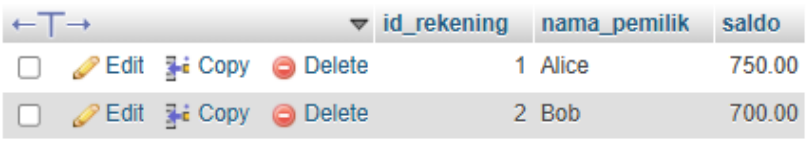
Pada skenario no.6, saldo Alice tidak akan berubah. Perintah ROLLBACK membatalkan semua operasi sejak START TRANSACTION, mengembalikan kondisi database seperti semula.

7. Implementasi dengan SAVEPOINT
Menggunakan SAVEPOINT untuk Pembatalan Sebagian. SAVEPOINT berguna ketika ingin membatalkan hanya sebagian dari transaksi, bukan keseluruhan.

```
START TRANSACTION;
-- Operasi 1: Mengurangi saldo Alice (akan disimpan)
UPDATE rekening SET saldo = saldo - 50.00 WHERE id_rekening = 1;
-- Menentukan SAVEPOINT
SAVEPOINT setelah_kurangi_alice;
-- Operasi 2: Menambahkan saldo Bob (akan dibatalkan)
UPDATE rekening SET saldo = saldo + 50.00 WHERE id_rekening = 2;
-- Membatalkan transaksi hingga SAVEPOINT
ROLLBACK TO SAVEPOINT setelah_kurangi_alice;
-- Menyimpan transaksi yang tersisa secara permanen
COMMIT;
-- Memeriksa hasilnya
SELECT * FROM rekening;
```

Saldo Alice berkurang 50, tetapi saldo Bob tidak bertambah. Hanya operasi setelah SAVEPOINT yang dibatalkan, sedangkan yang sebelumnya tetap di-commit.

Hasil :



	id_rekening	nama_pemilik	saldo
Edit Copy Delete	1	Alice	750.00
Edit Copy Delete	2	Bob	700.00

8. Implementasi dengan AUTOCOMMIT
   Pada mode AUTOCOMMIT yang aktif (biasanya ini adalah mode default), setiap pernyataan SQL dieksekusi sebagai transaksi terpisah, yang otomatis di-commit setelah selesai.

```
-- Memastikan mode AUTOCOMMIT aktif
SET AUTOCOMMIT = 1;
-- Melakukan UPDATE
UPDATE rekening SET saldo = saldo - 10.00 WHERE id_rekening = 1;
-- Secara otomatis akan di-COMMIT, tidak perlu perintah COMMIT atau START TRANSACTION
-- Perubahan tidak bisa di-ROLLBACK lagi, jika mencoba ROLLBACK, tidak akan ada efeknya pada UPDATE
ROLLBACK;
-- Periksa hasilnya. Saldo Alice sudah terpengaruh.
SELECT * FROM rekening;
```

   Hasil :

		id_rekening	nama_pemilik	saldo
☐	Edit Copy Delete	1	Alice	740.00
☐	Edit Copy Delete	2	Bob	700.00

9. Menonaktifkan AUTOCOMMIT
   Untuk mengontrol transaksi secara manual, perlu menonaktifkan AUTOCOMMIT.

```
SET AUTOCOMMIT = 0;
-- Transaksi akan dimulai secara implisit
UPDATE rekening SET saldo = saldo - 10.00 WHERE id_rekening = 1;
-- Perubahan tidak permanen sampai ada COMMIT, jika memanggil ROLLBACK, perubahan akan dibatalkan
ROLLBACK;
-- Periksa hasilnya. Saldo Alice tidak berubah.
SELECT * FROM rekening;
```

		id_rekening	nama_pemilik	saldo
☐	Edit Copy Delete	1	Alice	740.00
☐	Edit Copy Delete	2	Bob	700.00