

MODUL 8 TRANSACTION

A. TUJUAN

- ✓ Memahami konsep dan urgensi transaksi dalam kehidupan sehari-hari,
- ✓ Mampu mengimplementasikan transaksi basis data,
- ✓ Mampu menyelesaikan operasi-perasi sensitif dengan memanfaatkan transaksi basis data.

B. PETUNJUK

- Awali setiap aktivitas dengan doa, semoga berkah dan mendapat kemudahan.
- Pahami tujuan, dasar teori, dan latihan-latihan praktikum dengan baik dan benar.
- Kerjakan tugas-tugas praktikum dengan baik, sabar, dan jujur.
- Tanyakan kepada asisten/dosen apabila ada hal-hal yang kurang jelas.

C. DASAR TEORI

1. Transaksi Basis Data

Pada suatu hari, Tono ingin mentransfer uang ke rekening adiknya, Tini, sebesar Rp 9.000.000. Diilustrasikan secara sederhana, proses yang terjadi adalah sebagai berikut:

- 1) Saldo Tono dikurangi sebesar Rp 9.000.000,
- 2) Saldo Tini ditambah sebesar Rp 9.000.000.

Begitu kedua tahap di atas terlaksana dengan baik, Tono akan merasa lega, dan Tini pun bersuka cita, karena sebentar lagi dapat memiliki laptop baru yang telah lama dimimpikannya. Namun, pertimbangkan jika di antara langkah (1) dan (2) terjadi hal-hal buruk yang tidak diinginkan, misalnya saja mesin ATM *crash*, terjadi pemadaman listrik dan UPS gagal *up*, *disk* pada *server* penuh, atau ada *cracker* yang merusak infrastruktur jaringan. Maka dapat dipastikan bahwa saldo Tono berkurang, tetapi saldo Tini 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, biasa juga dikenal dengan jargon *all or nothing*.

Uniknya, konsep penyelesaian di atas sudah dikemukakan oleh para ahli sejak puluhan tahun silam. Konsep yang disebut transaksi basis data (*database transaction*) ini sebenarnya cukup sederhana, antara lain:

- Tandai bagian awal dan akhir himpunan perintah,
- 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*).

- *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.
- *Consistency*
Jika basis data pada awalnya dalam keadaan konsisten, maka pelaksanaan transaksi dengan sendirinya juga harus meninggalkan basis data tetap dalam status konsisten.
- *Isolation*
Isolasi memastikan bahwa secara bersamaan (konkuren) eksekusi transaksi terisolasi dari yang lain.
- *Durability*
Begitu transaksi telah dilaksanakan (*di-commit*), maka perubahan yang diakibatkan tidak akan hilang atau tahan lama (*durable*), sekalipun terdapat kegagalan sistem.

D. LATIHAN

1. Transaksi di MySQL

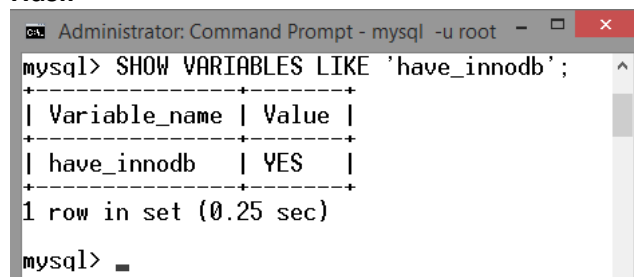
MySQL mendukung transaksi melalui storage engine InnoDB (*full ACID compliance*) dan BDB (BerkeleyDB) sejak versi 4.0. Oleh karena itu, untuk dapat mengimplementasikan transaksi, DBMS MySQL harus mendukung salah satu atau kedua *engine transactional*.

Untuk memeriksa dukungan transaksi basis data, gunakan perintah berikut ini:

MySQL Query

```
SHOW VARIABLES LIKE 'have_innodb';
```

Hasil



```
Administrator: Command Prompt - mysql -u root
mysql> SHOW VARIABLES LIKE 'have_innodb';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| have_innodb   | YES   |
+-----+-----+
1 row in set (0.25 sec)
mysql>
```

Apabila nilai yang dikembalikan **YES**, berarti dukungan transaksi basis data telah aktif. Jika tidak, pastikan tabel yang terbentuk menggunakan engine InnoDB. Idealnya, engine InnoDB akan menjadi *default engine* di MySQL.

☞ Langkah ini diperlukan untuk memastikan tipe engine tabel yang terbentuk adalah InnoDB, karena engine non-transactional (seperti MyISAM) tidak dapat digunakan untuk mengimplementasikan transaksi basis data.

2. Tabel Transaksi

Sebelum memulai implementasi transaksi basis data, terlebih dahulu buat *database* sebagai berikut:

MySQL Query

```
CREATE DATABASE dtransaksi;
```

Hasil

```
MariaDB [(none)]> CREATE DATABASE dtransaksi;  
Query OK, 1 row affected (0.01 sec)
```

Setelah *database* dtransaksi “ telah berhasil dibuat, berikutnya adalah gunakan *database* tersebut sebagai berikut:

MySQL Query

```
USE dtransaksi;
```

Hasil

```
MariaDB [(none)]> use dtransaksi;  
Database changed
```

Setelah database “dtransaksi” telah berhasil dibuka/digunakan, berikutnya adalah membuat sebuah tabel sebagai berikut:

MySQL Query

```
CREATE TABLE trans_demo(  
    nama VARCHAR(10) NOT NULL,  
    PRIMARY KEY(nama)  
) ENGINE = InnoDB;
```

Hasil

```
MariaDB [dtransaksi]> CREATE TABLE trans_demo(  
-> nama VARCHAR(10) NOT NULL,  
-> PRIMARY KEY(nama)  
-> ) ENGINE = InnoDB;  
Query OK, 0 rows affected (0.13 sec)
```

☞ Perhatikan tipe atau storage engine-nya, **HARUS** InnoDB.

Jika sudah memiliki tabel *non-transactional* dan ingin mengubahnya menjadi *transactional*, gunakan perintah `ALTER TABLE`. Sebagai contoh, perintah berikut akan mengubah *engine* tabel `non_trans` menjadi InnoDB:

MySQL Query

```
ALTER TABLE non_trans ENGINE = InnoDB;
```

3. Implementasi Transaksi

Transaksi di MySQL diinisiasi dengan menggunakan pernyataan `START TRANSACTION` atau `BEGIN` dan diakhiri dengan `COMMIT` untuk menerapkan semua transaksi.

Sebagai ilustrasi, ikuti dan pahami contoh kasus berikut ini:

- 1) Aktifkan transaksi basis data,

MySQL Query

```
START TRANSACTION;
```

Hasil

```
MariaDB [dtransaksi]> START TRANSACTION;  
Query OK, 0 rows affected (0.00 sec)
```

- 2) Tambahkan dua baris data ke tabel `trans_demo`, misalnya `mysql` dan `oracle`,

MySQL Query

```
INSERT INTO trans_demo  
VALUES("mysql"), ("oracle");
```

Hasil

```
MariaDB [dtransaksi]> INSERT INTO trans_demo  
-> VALUES("mysql"), ("oracle");  
Query OK, 2 rows affected (0.02 sec)  
Records: 2 Duplicates: 0 Warnings: 0
```

- 3) Periksa hasil penambahan data,

MySQL Query

```
SELECT *  
FROM trans_demo;
```

Hasil

```
MariaDB [dtransaksi]> SELECT *  
-> FROM trans_demo;  
+-----+  
| nama   |  
+-----+  
| mysql  |  
| oracle |  
+-----+  
2 rows in set (0.00 sec)
```

- 4) Keluar dari terminal,

MySQL Query

```
EXIT;
```

Hasil

```
MariaDB [dtransaksi]> exit;  
Bye
```

```
C:\xampp\mysql\bin>
```

- 5) Login kembali ke basis data yang sama, kemudian periksa isi tabel 'trans_demo'.

MySQL Query

```
SELECT *  
FROM trans_demo;
```

Hasil

```
MariaDB [dtransaksi]> SELECT *  
-> FROM trans_demo;  
Empty set (0.00 sec)
```

Dapat diperhatikan pada gambar di atas, bahwa tabel 'trans_demo' kosong. Hal tersebut dikarenakan tidak diterapkannya transaksi dengan memanggil COMMIT. Adapun penutupan *prompt* mysql mengakibatkan transaksi di-rollback secara implisit.

Sekarang ulangi langkah nomor 2, namun pada langkah nomor 4, ketikkan pernyataan COMMIT sebagai berikut:

MySQL Query

```
COMMIT;
```

Hasil

```
MariaDB [dtransaksi]> COMMIT;  
Query OK, 0 rows affected (0.00 sec)  
MariaDB [dtransaksi]> exit;  
Bye
```

```
C:\xampp\mysql\bin>
```

```
MariaDB [dtransaksi]> USE dtransaksi;  
Database changed  
MariaDB [dtransaksi]> SELECT *  
-> FROM trans_demo;  
+-----+  
| nama   |  
+-----+  
| mysql  |  
| oracle |  
+-----+  
2 rows in set (0.00 sec)
```

Dari hasil di atas, telah dilakukan penutupan *prompt* mysql dan dibuka kembali. Ketika telah digunakan COMMIT, maka setelah *prompt* mysql yang telah ditutup dibuka kembali, data telah tersimpan (tidak di-rollback).

Autocommit Mode

Selain menggunakan pernyataan START TRANSACTION, juga dapat menggunakan pernyataan SET untuk mengatur nilai variabel *autocommit*. Nilai *default autocommit* adalah 1, yang menyatakan bahwa transaksi basis data tidak aktif. Dengan kata lain, setiap perintah akan langsung diterapkan secara permanen.

- 1) Terlebih dahulu periksa nilai variabel *autocommit*,

MySQL Query

```
SELECT @@autocommit;
```

Hasil

```
MariaDB [dtransaksi]> SELECT @@autocommit;
+-----+
| @@autocommit |
+-----+
|             1 |
+-----+
1 row in set (0.00 sec)
```

- 2) Tetapkan nilai *autocommit* menjadi 0 (mode transaksi *on*),

MySQL Query

```
SET @@autocommit = 0;
```

Hasil

```
MariaDB [dtransaksi]> SET @@autocommit = 0;
Query OK, 0 rows affected (0.00 sec)

MariaDB [dtransaksi]> SELECT @@autocommit;
+-----+
| @@autocommit |
+-----+
|             0 |
+-----+
1 row in set (0.00 sec)
```

- 3) Tambahkan data berikut ini pada tabel *trans_demo* , „

MySQL Query

```
INSERT INTO trans_demo
VALUES ('db2');
```

Hasil

```
MariaDB [dtransaksi]> INSERT INTO trans_demo
-> VALUES ('db2');
Query OK, 1 row affected (0.02 sec)
```

- 4) Keluar dari *prompt* *mysql*, kemudian *login* kembali dan periksa hasil penambahan data. Seharusnya, hasil penambahan tidak akan diterapkan secara permanen di basis data,
- 5) Amati apa yang terjadi ketika melakukan langkah nomor 3 dengan menampilkan semua data pada tabel *trans_demo* , „
- 6) Periksa nilai variabel *autocommit*,
- 7) Lakukan langkah nomor 4, kemudian periksa nilai variabel *autocommit*,
- 8) Berikan kesimpulan dari langkah nomor 5 sampai dengan langkah nomor 7 pada laporan.

Selama *autocommit* belum dikembalikan ke 1, maka mode transaksi basis data akan selalu aktif, sehingga tidak diperlukan lagi pemanggilan COMMIT.

☞ Normalnya, pengaturan variabel *autocommit* berlaku untuk satu sesi login. Jadi, login ulang meskipun di terminal yang sama, maka akan mengakibatkan

4. Rollback Transaksi

Akhir pernyataan transaksi dapat berupa COMMIT atau ROLLBACK, tergantung pada kondisinya. Pernyataan ROLLBACK digunakan untuk menggugurkan rangkaian perintah. ROLLBACK akan dilakukan manakala ada satu atau lebih perintah yang gagal dilaksanakan. Di samping itu, juga ROLLBACK dapat dilakukan secara eksplisit dengan memanggil pernyataan ROLLBACK.

- 1) Aktifkan transaksi basis data,

MySQL Query

```
START TRANSACTION;
```

Hasil

```
MariaDB [dtransaksi]> START TRANSACTION;
Query OK, 0 rows affected (0.00 sec)
```

- 2) Terlebih dahulu periksa nilai di tabel trans_demo ,

MySQL Query

```
SELECT *
FROM trans_demo;
```

Hasil

```
MariaDB [dtransaksi]> SELECT *
-> FROM trans_demo;
+-----+
| nama  |
+-----+
| mysql |
| oracle|
+-----+
2 rows in set (0.00 sec)
```

- 3) Tambahkan baris data berikut ini,

MySQL Query

```
INSERT INTO trans_demo
VALUES ("sybase");
```

Hasil

```
MariaDB [dtransaksi]> INSERT INTO trans_demo
-> VALUES ('sybase');
Query OK, 1 row affected (0.01 sec)
```

- 4) Tambahkan lagi baris data, namun dengan nilai yang sama,

MySQL Query

```
INSERT INTO trans_demo
VALUES ("sybase");
```

Hasil

```
MariaDB [dtransaksi]> INSERT INTO trans_demo
-> VALUES('sybase');
ERROR 1062 (23000): Duplicate entry 'sybase' for key 'PRIMARY'
MariaDB [dtransaksi]>
MariaDB [dtransaksi]> SELECT *
-> FROM trans_demo;
+-----+
| nama  |
+-----+
| mysql |
| oracle|
| sybase|
+-----+
3 rows in set (0.00 sec)
```

- 5) Berikan pernyataan ROLLBACK untuk membatalkan rangkaian perintah dalam satu transaksi,

MySQL Query

```
ROLLBACK;
```

Hasil

```
3 rows in set (0.00 sec)

MariaDB [dtransaksi]> ROLLBACK;
Query OK, 0 rows affected (0.01 sec)

MariaDB [dtransaksi]>
```

- 6) Sampai langkah ini, seharusnya tidak ada penambahan data baru yang tersimpan.

MySQL Query

```
SELECT *
FROM trans_demo;
```

Hasil

```
MariaDB [dtransaksi]> SELECT *
-> FROM trans_demo;
+-----+
| nama  |
+-----+
| mysql |
| oracle|
+-----+
2 rows in set (0.00 sec)
```

Pemanggilan `START TRANSACTION` diakhir transaksi yang tidak ditutup, misal menggunakan `COMMIT`, maka akan mengakibatkan dipanggilnya `COMMIT` secara implisit. Dengan demikian, tidak dapat lagi memaksa pembatalan melalui pernyataan `ROLLBACK`.

5. Checkpointing

Idealnya, `ROLLBACK` akan menggugurkan keseluruhan perintah dalam blok transaksi. Kondisi ini terkadang tidak dikehendaki, misal terdapat tiga perintah, namun kita hanya ingin menggugurkan perintah setelah perintah kedua (perintah pertama masih ada). Dalam kasus ini, kita bisa memanfaatkan fitur *checkpointing*.

- 1) Aktifkan transaksi basis data,

MySQL Query

```
START TRANSACTION;
```

Hasil

```
MariaDB [dtransaksi]> START TRANSACTION;  
Query OK, 0 rows affected (0.00 sec)
```

- 2) Terlebih dahulu periksa nilai di tabel "trans_demo",

MySQL Query

```
SELECT *  
FROM trans_demo;
```

Hasil

```
MariaDB [dtransaksi]> SELECT *  
-> FROM trans_demo;  
+-----+  
| nama  |  
+-----+  
| mysql |  
| oracle|  
+-----+  
2 rows in set (0.00 sec)
```

- 3) Tambahkan baris data berikut,

MySQL Query

```
INSERT INTO trans_demo  
VALUES ("sybase");
```

Hasil

```
MariaDB [dtransaksi]> INSERT INTO trans_demo  
-> VALUES ('sybase');  
Query OK, 1 row affected (0.01 sec)
```

- 4) Gunakan pernyataan `SAVEPOINT` untuk menandai perintah pertama,

MySQL Query

```
SAVEPOINT mypoint1;
```

Hasil

```
MariaDB [dtransaksi]> SAVEPOINT mypoint1;  
Query OK, 0 rows affected (0.00 sec)
```

- 5) Tambahkan lagi baris baru,

MySQL Query

```
INSERT INTO trans_demo  
VALUES ("sqlite");
```

Hasil

```
MariaDB [dtransaksi]> INSERT INTO trans_demo  
-> VALUES('sqlite');  
Query OK, 1 row affected (0.00 sec)
```

```
MariaDB [dtransaksi]> SELECT *  
-> FROM trans_demo;
```

```
+-----+  
| nama  |  
+-----+  
| mysql |  
| oracle|  
| sqlite|  
| sybase|  
+-----+  
4 rows in set (0.00 sec)
```

- 6) Berikan perintah ROLLBACK ke SAVEPOINT **my_point1**,

MySQL Query

```
ROLLBACK TO SAVEPOINT mypoint1;
```

Hasil

```
MariaDB [dtransaksi]> ROLLBACK TO SAVEPOINT mypoint1;  
Query OK, 0 rows affected (0.00 sec)
```

- 7) Tambahkan lagi sebuah baris baru,

MySQL Query

```
INSERT INTO trans_demo  
VALUES ("db2");
```

Hasil

```
MariaDB [dtransaksi]> INSERT INTO trans_demo  
-> VALUES('db2');  
Query OK, 1 row affected (0.00 sec)
```

- 8) Terapkan transaksi,

MySQL Query

```
COMMIT;
```

Hasil

```
MariaDB [dtransaksi]> COMMIT;  
Query OK, 0 rows affected (0.09 sec)
```

9) Lihat hasilnya.

MySQL Query

```
SELECT *  
FROM trans_demo;
```

Hasil

```
MariaDB [dtransaksi]> SELECT *  
-> FROM trans_demo;  
+-----+  
| nama   |  
+-----+  
| db2    |  
| mysql  |  
| oracle |  
| sybase |  
+-----+  
4 rows in set (0.00 sec)
```

Sampai di sini seharusnya Anda dapat memahami fungsi *checkpointing*. Jika Anda masih belum memahami fungsi *checkpointing*, ulangi lagi langkah-langkah di atas hingga benar-benar paham.

E. TUGAS PRAKTIKUM

Untuk menyelesaikan tugas praktikum, gunakan tabel tabungan dengan struktur sebagai berikut:

```
CREATE TABLE tabungan(  
    no_rek INT(15) NOT NULL,  
    jumlah DOUBLE NOT NULL,  
    trans_id INT NOT NULL  
) ENGINE = InnoDB;
```

- 1) Definisikan *stored procedure* untuk menangani transfer antar rekening dengan mengimplementasikan transaksi basis data. Setelah transfer uang berhasil, tampilkan sisa saldo rekening pengirim!
- 2) Definisikan *stored procedure* untuk menangani penarikan tabungan dengan mengimplementasikan transaksi basis data. Skenarionya adalah penarikan hanya dapat dilakukan jika:
 - Saldo mencukupi,
 - Menyisakan saldo minimal Rp 50.000, dan
 - Jumlah (nominal) penarikan minimal Rp 50.000 dan maksimal Rp 500.000.

- 3) Definisikan *stored procedure* untuk menangani penarikan berulang. Artinya, penarikan tabungan dengan nominal tertentu yang dispesifikasikan akan dilakukan sebanyak iterasi yang dispesifikasikan juga. Aturan penarikan tabungan pada soal nomor 2 masih berlaku di sini.

Sebagai ilustrasi, misal Tono memiliki saldo Rp 300.000, kemudian melakukan penarikan melalui *stored procedure* berulang dengan nominal Rp 100.000 sebanyak 3 kali, maka *stored procedure* hanya akan meng- *commit* penarikan Rp 200.000 (2 x Rp 100.000). Sisa Rp 100.000 tidak dapat diambil, karena harus menyisakan saldo Rp 50.000.