

Minggu 9-Database Manajemen Sistem Lanjut

MySQL Transaction Query

Dalam SQL terkadang kita membutuhkan untuk menjalankan lebih dari 1 query untuk mendapatkan hasil tertentu. Dimana beberapa query ini adalah menjadi 1 kebutuhan. Contohnya, dalam proses transfer saldo dari rekening A ke rekening B. Proses transfer ini melibatkan paling tidak 2 buah query, yaitu query pertama mengurangi jumlah saldo A dan query kedua menambahkan saldo rekening B.

Hal seperti ini disebut sebagai atomic transaction, atomic berarti bagian terkecil yang sudah tidak dapat dipisahkan lagi. 2 buah query yang dibutuhkan untuk proses pemindahan saldo itu harus berjalan secara menyeluruh, bayangkan ketika proses pemotongan saldo A berhasil, namun penambahan saldo B gagal, pasti si A marah besar.

Secara gampangnya, saldo A harus terpotong apabila saldo B berhasil ditambahkan, atau query pertama akan disimpan apabila query kedua (dan seterusnya) berhasil. Jika tidak, maka perubahan yang terjadi harus dibatalkan. Perubahan disini bukan berarti menambahkan kembali ke rekening A, melainkan membatalkan perubahan yang ada.

Query untuk menjalankan hal ini disebut sebagai TRANSACTION. Hasil dari Transaction ini akan disimpan bila seluruh query yang terlibat berhasil dijalankan. di MySQL, Transaction harus dimulai dengan query BEGIN, kemudian diakhiri dengan COMMIT apabila ingin menyimpan perubahan, dan ROLLBACK apabila ingin membatalkan perubahan yang terjadi setelah BEGIN tadi.

Contoh penerapan dalam MySQL (dan juga database SQL standard lainnya) adalah berikut

```
1  mysql> DESC customer;
2  +-----+-----+-----+-----+-----+-----+
3  | FIELD | TYPE   | NULL | KEY | DEFAULT | Extra      |
4  +-----+-----+-----+-----+-----+-----+
5  | id    | INT(11) | NO   | PRI | NULL    | AUTO_INCREMENT |
6  | name  | VARCHAR(40) | YES |     | NULL    |                |
7  | saldo | INT(11)  | YES |     | NULL    |                |
8  +-----+-----+-----+-----+-----+-----+
9  3 ROWS IN SET (0.00 sec)
10
11 mysql> SELECT * FROM customer;
12 +-----+-----+-----+-----+-----+-----+
```

```

13 | id | name   | saldo |
14 +---+-----+-----+
15 | 1 | John Doe | 100 |
16 | 2 | Will Smith | 100 |
17 +---+-----+-----+
18 2 ROWS IN SET (0.00 sec)
19
20 mysql> BEGIN;
21 Query OK, 0 ROWS affected (0.01 sec)
22
23 mysql> UPDATE customer SET saldo = saldo - 20 WHERE id = 1;
24 Query OK, 1 ROW affected (0.00 sec)
25 ROWS matched: 1 Changed: 1 Warnings: 0
26
27 mysql> SELECT * FROM customer;
28 +---+-----+-----+
29 | id | name   | saldo |
30 +---+-----+-----+
31 | 1 | John Doe | 80 |
32 | 2 | Will Smith | 100 |
33 +---+-----+-----+
34 2 ROWS IN SET (0.00 sec)
35
36 mysql> ROLLBACK;
37 Query OK, 0 ROWS affected (0.10 sec)
38
39 mysql> SELECT * FROM customer;
40 +---+-----+-----+
41 | id | name   | saldo |
42 +---+-----+-----+
43 | 1 | John Doe | 100 |
44 | 2 | Will Smith | 100 |
45 +---+-----+-----+
46 2 ROWS IN SET (0.00 sec)

```

Contoh query diatas memberikan gambaran ketika saldo customer 1 diubah, namun kemudian dilakukan perintah ROLLBACK. Data yang ada akan dikembalikan sama seperti keadaan sebelum BEGIN.

Berikut contoh penerapan COMMIT.

```

1 mysql> SELECT * FROM customer;
2 +---+-----+-----+
3 | id | name   | saldo |
4 +---+-----+-----+

```

```

5 | 1 | John Doe | 100 |
6 | 2 | Will Smith | 100 |
7 +---+-----+-----+
8 2 ROWS IN SET (0.01 sec)
9
10 mysql> BEGIN;
11 Query OK, 0 ROWS affected (0.00 sec)
12
13 mysql> UPDATE customer SET saldo = saldo - 30 WHERE id = 1;
14 Query OK, 1 ROW affected (0.00 sec)
15 ROWS matched: 1 Changed: 1 Warnings: 0
16
17 mysql> SELECT * FROM customer;
18 +---+-----+-----+
19 | id | name | saldo |
20 +---+-----+-----+
21 | 1 | John Doe | 70 |
22 | 2 | Will Smith | 100 |
23 +---+-----+-----+
24 2 ROWS IN SET (0.00 sec)
25
26 mysql> COMMIT;
27 Query OK, 0 ROWS affected (0.05 sec)
28
29 mysql> SELECT * FROM customer;
30 +---+-----+-----+
31 | id | name | saldo |
32 +---+-----+-----+
33 | 1 | John Doe | 70 |
34 | 2 | Will Smith | 100 |
35 +---+-----+-----+
36 2 ROWS IN SET (0.00 sec)
37
38 mysql> BEGIN;
39 Query OK, 0 ROWS affected (0.00 sec)
40
41 mysql> UPDATE customer SET saldo = saldo - 20;
42 Query OK, 2 ROWS affected (0.01 sec)
43 ROWS matched: 2 Changed: 2 Warnings: 0
44
45 mysql> SELECT * FROM customer;
46 +---+-----+-----+
47 | id | name | saldo |
48 +---+-----+-----+
49 | 1 | John Doe | 50 |

```

```

50 | 2 | Will Smith | 80 |
51 +---+-----+-----+
52 2 ROWS IN SET (0.00 sec)
53
54 mysql> ROLLBACK;
55 Query OK, 0 ROWS affected (0.18 sec)
56
57 mysql> SELECT * FROM customer;
58 +---+-----+-----+
59 | id | name      | saldo |
60 +---+-----+-----+
61 | 1  | John Doe  | 70    |
62 | 2  | Will Smith| 100   |
63 +---+-----+-----+
64 2 ROWS IN SET (0.00 sec)

```

Pada tampilan diatas,apa yang terjadi adalah

1. Saldo awal adalah 100
2. BEGIN, kita mulai SQL Transaction
3. Kita kurangi saldo customer dengan 1 sebanyak 30
4. Saldo customer 1 adalah 70
5. COMMIT, simpan perubahan secara permanen
6. Saldo customer 1 adalah 70, setelah commit secara permanen
7. BEGIN, kita mulai SQL Transaction kedua
8. Kurangi kembali saldo customer 1 sebanyak 20
9. Lihat bahwa saldo customer 1 adalah 50, yang mana adalah benar $70 - 20 = 50$;
10. kita lakukan ROLLBACK, batalkan seluruh perubahan setelah BEGIN
11. Saldo customer 1 kembali menjadi 70.

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

- 1) Saldo si A dikurangi Rp 1.500.000,00 sehingga saldo akhir si A menjadi sebesar Rp 8.500.000,00
- 2) 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. 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, biasa dikenal dengan jargon “*all or nothing*”.
- *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.

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-perintah tersebut adalah sebagai 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

B. LATIHAN PRAKTIKUM

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 apakah tabel sudah menggunakan *engine* InnoDB dapat dilihat pada struktur tabel.

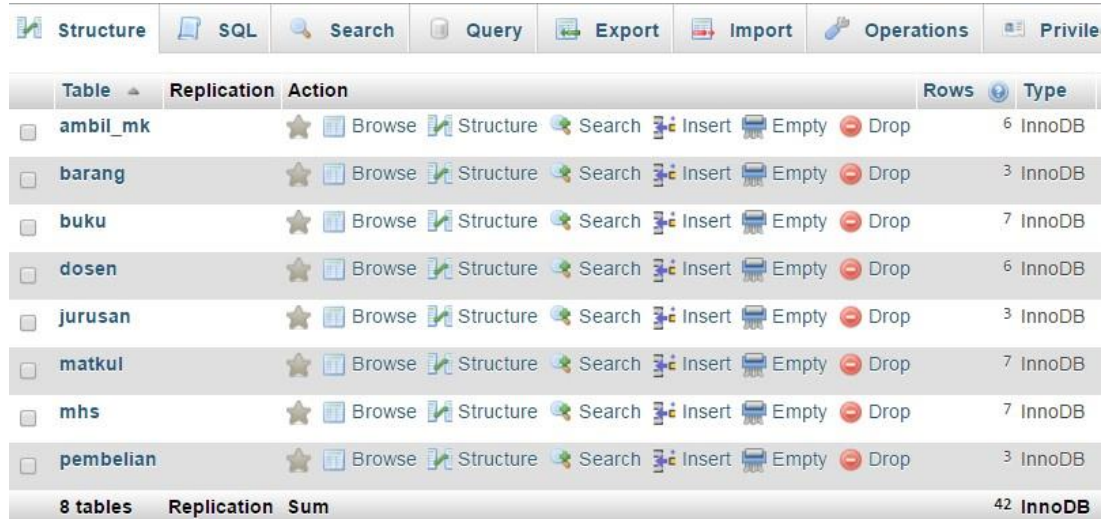


Table	Replication	Action	Rows	Type
ambil_mk		Browse Structure Search Insert Empty Drop	6	InnoDB
barang		Browse Structure Search Insert Empty Drop	3	InnoDB
buku		Browse Structure Search Insert Empty Drop	7	InnoDB
dosen		Browse Structure Search Insert Empty Drop	6	InnoDB
jurusan		Browse Structure Search Insert Empty Drop	3	InnoDB
matkul		Browse Structure Search Insert Empty Drop	7	InnoDB
mhs		Browse Structure Search Insert Empty Drop	7	InnoDB
pembelian		Browse Structure Search Insert Empty Drop	3	InnoDB
8 tables	Replication	Sum	42	InnoDB



Perlu sekali diperhatikan, *engine non-transactional* (seperti MyISAM) tidak dapat digunakan untuk mengimplementasikan transaksi basis data

2. Tabel Transaksi

Membuat sebuah tabel transaksi sebagai berikut :

- Membuat tabel di PhpMyAdmin

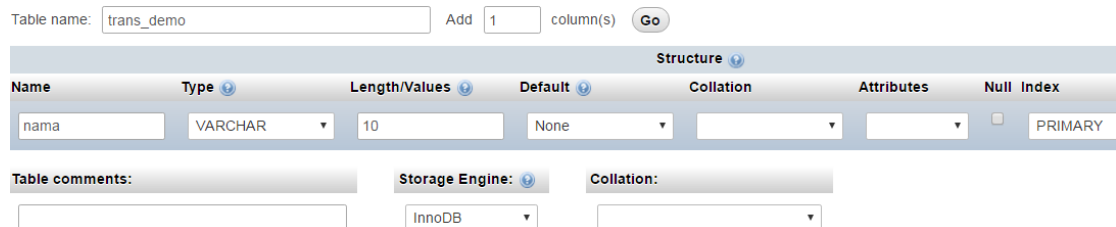


Table name: Add column(s)

Name	Type	Length/Values	Default	Collation	Attributes	Null	Index
nama	VARCHAR	10	None			<input type="checkbox"/>	PRIMARY

Table comments:

Storage Engine:

Collation:

- Membuat tabel dengan SQL Query

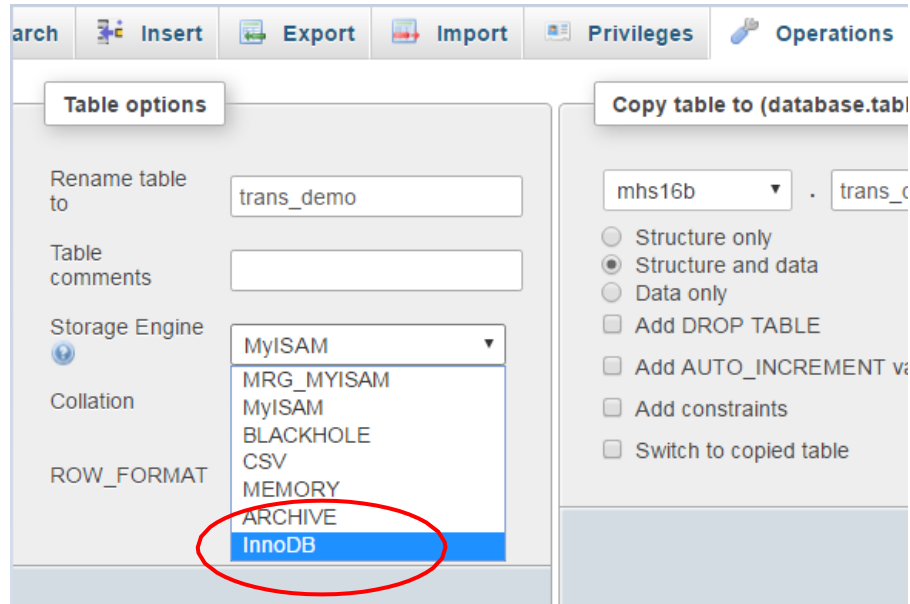
```
CREATE TABLE trans_demo (  
    nama VARCHAR(10) NOT NULL,  
    PRIMARY KEY (nama)  
) ENGINE = INNODB;
```



Perhatikan tipe atau *storage engine* nya HARUS InnoDB

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

- PhpMyAdmin



- SQL Query

```
ALTER TABLE trans_demo Engine = "InnoDB";
```

3. Implementasi Transaksi

Transaksi di MySQL diinisiasi dengan menggunakan pernyataan START TRANSACTION atau BEGIN dan diakhiri dengan COMMIT untuk menerapkan semua transaksi. Untuk membuat suatu kegiatan transaksi dalam PhpMyAdmin dapat menggunakan Stored Procedure.

a. COMMIT

- 1) Membuat stored procedur dengan nama “trans1” yang berisikan perintah seperti di bawah ini. Dimana ditambahkan dua baris data ke dalam tabel trans_demo.

```
BEGIN
START TRANSACTION;
INSERT INTO trans_demo VALUES("MySQL"),("Oracle");
SELECT * FROM trans_demo;
END
```

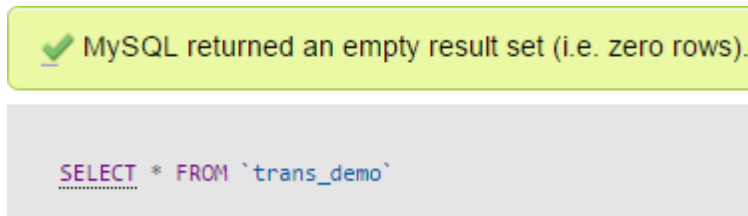
- 2) Panggil stored procedure dengan perintah CALL

```
CALL trans1;
```

Maka akan muncul dua baris data yang telah ditambahkan.

nama
MySQL
Oracle

3) Periksa data pada tabel trans_demo.



Dapat diperhatikan pada gambar di atas, bahwa tabel trans_demo kosong. Hal tersebut dikarenakan tidak diterapkannya transaksi dengan memanggil perintah COMMIT. Sehingga mengakibatkan transaksi di-rollback secara implisit.

Selanjutnya ubah isi perintah pada procedure trans1 menjadi seperti di bawah ini. Dimana telah ditambahkan perintah COMMIT setelah diberikan perintah penambahan data.

```
BEGIN
  START TRANSACTION;
  INSERT INTO trans_demo VALUES("MySQL"), ("Oracle");
  COMMIT;
END
```

Panggil stored procedure dengan perintah CALL

```
CALL trans1;
```

Maka akan muncul dua baris data yang telah ditambahkan.

nama
MySQL
Oracle

Periksa data pada tabel trans_demo.

nama
MySQL
Oracle

Ketika telah ditambahkan perintah COMMIT, maka data telah tersimpan secara permanen (tidak di-rollback).

b. 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 telah aktif. Dengan kata lain, setiap perintah akan langsung diterapkan secara permanen.

- 1) Modifikasi isi perintah procedure trans1 dengan mengganti START TRANSACTION dengan SET seperti di bawah ini.

```

BEGIN
    SET @@autocommit = 0;
    INSERT INTO trans_demo VALUES("MongoDB");
    SELECT * FROM trans_demo;
END

```

- 2) Panggil procedure

```
CALL trans1;
```

- 3) Periksa pada data tabel trans_demo

	nama
<input type="checkbox"/> Edit <input type="checkbox"/> Copy <input type="checkbox"/> Delete	MongoDB
<input type="checkbox"/> Edit <input type="checkbox"/> Copy <input type="checkbox"/> Delete	MySQL
<input type="checkbox"/> Edit <input type="checkbox"/> Copy <input type="checkbox"/> Delete	Oracle

Data yang baru ditambahkan akan disimpan secara permanen, karena telah mengaktifkan transaksi basis data.

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, ROLLBACK juga dapat dilakukan secara eksplisit dengan memanggil pernyataan ROLLBACK.

- 1) Membuat procedure “roll” yang berisikan perintah seperti di bawah ini

```

BEGIN
    START TRANSACTION;
    INSERT INTO trans_demo VALUES("PostgreSQL");
    ROLLBACK;
    SELECT * FROM trans_demo;
END

```

- 2) Panggil procedure dengan perintah CALL

```
CALL roll;
```

nama
MongoDB
MySQL
Oracle

Dapat dilihat bahwa data yang ditambahkan tidak akan muncul karena adanya pernyataan ROLLBACK yang akan menggugurkan rangkaian perintah.

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) Membuat procedure dengan nama “cekpoint” yang berisikan perintah seperti di bawah ini.

```
BEGIN
  START TRANSACTION;
  INSERT INTO trans_demo VALUES("PostgreSQL");
  SAVEPOINT point1;

  INSERT INTO trans_demo VALUES("NoSQL");
  ROLLBACK TO SAVEPOINT point1;

  INSERT INTO trans_demo VALUES("FireBird");
  COMMIT;

  SELECT * FROM trans_demo;
END
```

- 2) Panggil dengan menggunakan perintah CALL dan amati hasil penambahan data pada tabel trans_demo.

CALL checkpoint;

Hasil

nama		nama
FireBird	<input type="checkbox"/> Edit <input type="checkbox"/> Copy <input type="checkbox"/> Delete	FireBird
MongoDB	<input type="checkbox"/> Edit <input type="checkbox"/> Copy <input type="checkbox"/> Delete	MongoDB
MySQL	<input type="checkbox"/> Edit <input type="checkbox"/> Copy <input type="checkbox"/> Delete	MySQL
Oracle	<input type="checkbox"/> Edit <input type="checkbox"/> Copy <input type="checkbox"/> Delete	Oracle
PostgreSQL	<input type="checkbox"/> Edit <input type="checkbox"/> Copy <input type="checkbox"/> Delete	PostgreSQL

Dapat dilihat bahwa data yang ditambahkan sebelum ditandai dengan perintah SAVEPOINT akan disimpan secara permanen. Sedangkan jika ada penambahan data diantara perintah SAVEPOINT dan perintah ROLLBACK, maka data tidak akan tersimpan permanen. Data akan tersimpan secara permanen jika sudah ada pernyataan COMMIT.

TUGAS MINGGU 9

Praktekkan semua materi di atas, screenshot setiap tahapan dan hasilnya, simpan dengan nama file **SMBDL20211-REG-KELAS-M9-NOABSEN-NAMA.docx**

Upload file dokumentasi ke elearning sebelum petemuan berikutnya.