BAB XI
MENGELOLA TRANSAKSI

Dalam penggunaan database, sering terjadi pengaksesan sebuah database oleh beberapa user atau beberapa program pada saat yang bersamaan, oleh karena itu dibutuhkan pengaturan untuk menjaga konsistensi pengaksesan data, sehingga tidak akan terjadi katidakkonsistensian data. Transaksi dapat digunakan untuk menjaga konsistensi dan keakuratan data. Transaksi digunakan untuk mengatur hak user pada saat mengakses database.

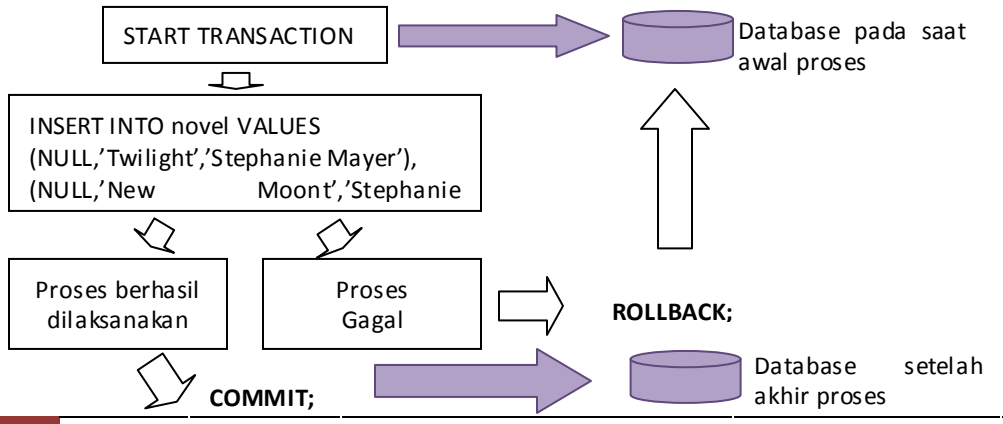
Transaksi memastikan bahwa suatu proses yang dijalankan terhadap database dilaksanakan seluruhnya atau tidak sama sekali. Jika terjadi suatu kesalahan, maka proses harus di roll back dan kondisi database kembali seperti sebelum dimulainya proses tersebut, sebaliknya jika proses berhasil, maka seluruh data dimasukkan ke dalam database. Selain itu, transaksi juga memastikan hanya boleh ada satu user yang sedang mengakses penulisan terhadap database, jika satu user sedang melakukan proses penulisan, maka user lain dibatasi dalam pengaksesan database, dengan begitu, maka data akan konsisten.

Hanya saja, tidak semua mesin mendukung proses pengelolaan transaksi. Dua mesin yang sering digunakan adalah MyISAM dan InnoDB. MyISAM lebih sering digunakan, tapi mesin ini tidak mendukung pengelolaan transaksi, sehingga untuk menggunakan pengelolaan transaksi, kita perlu menggunakan mesin InnoDB. Sebagai contoh:

```
CREATE TABLE novel
  (
  idNovel smallint unsigned auto_increment primary key,
  judulNovel varchar(50) not null,
  pengarangNovel varchar(50)
  )
  ENGINE=INNODB;
```

Keterangan mesin yang digunakan, di letakkan pada akhir pendeskripsian tabel yang dibuat dan menggunakan mesin tersebut.

Berikut ini merupakan gambar skema pelaksanaan pengelolaan transaksi:



XI.1 Statement memulai Transaksi

Untuk memulai sebuah transaksi, syntax yang digunakan adalah **START TRANSACTION**, syntax ini diletakkan pada awal proses yang akan dilaksanakan menggunakan Transaksi.

XI.2 Statement COMMIT Transaksi

Statement COMMIT digunakan untuk mengakhiri suatu proses dan menyimpan semua perubahan data yang terjadi dalam proses. Berikut ini merupakan contoh penggunaan COMMIT:

```
START TRANSACTION;
INSERT INTO novel VALUES
  (NULL,'Twilight','Stephanie Mayer'),
  (NULL,'New Moon','Stephanie Mayer');
COMMIT;
```

Statement memulai Transaksi

Proses Pengisian data ke dalam tabel novel.

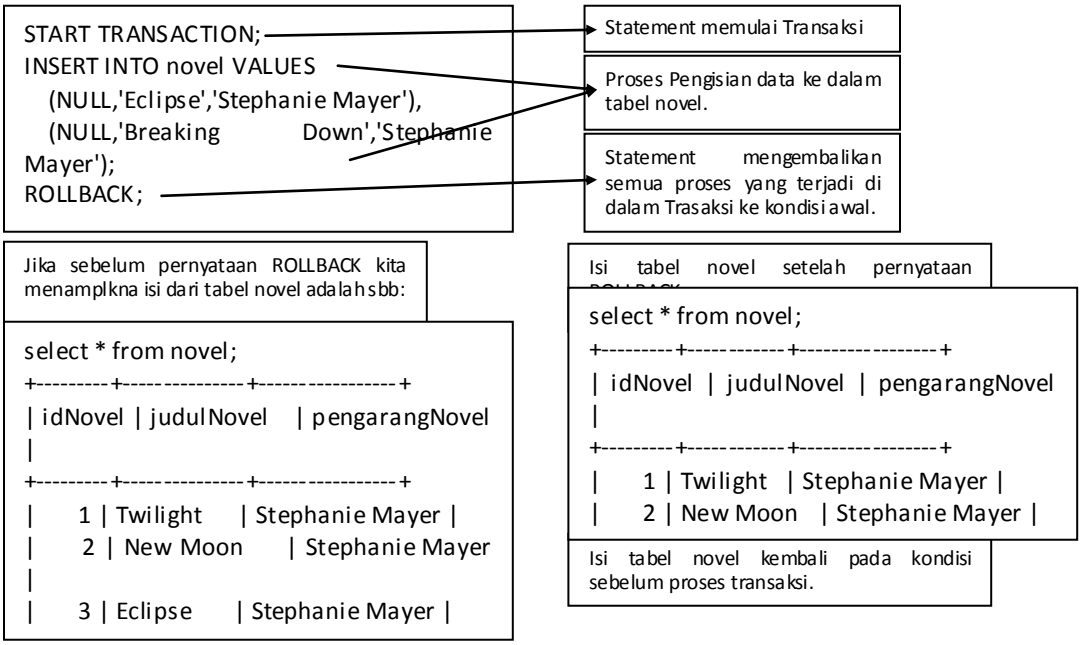
Statement mengcommit semua proses yang terjadi di dalam Trasaksi.

Hasilnya adalah:

```
select * from novel;
+---------+------------+-----------------+
| idNovel | judulNovel | pengarangNovel  |
+---------+------------+-----------------+
|       1 | Twilight   | Stephanie Mayer |
|       2 | New Moon   | Stephanie Mayer |
+---------+------------+-----------------+
```

XI.3 Statement ROLLBACK Transaksi

Seperti statement COMMIT, statement ROLLBACK juga digunakan untuk mengakhiri suatu proses, hanya saja, jika kita menggunakan ROLLBACK, maka semua proses yang dilakukan tidak akan disimpan, dan kondisi di kembalikan seperti pada awal proses. Berikut ini adalah contoh penggunaan ROLLBACK:



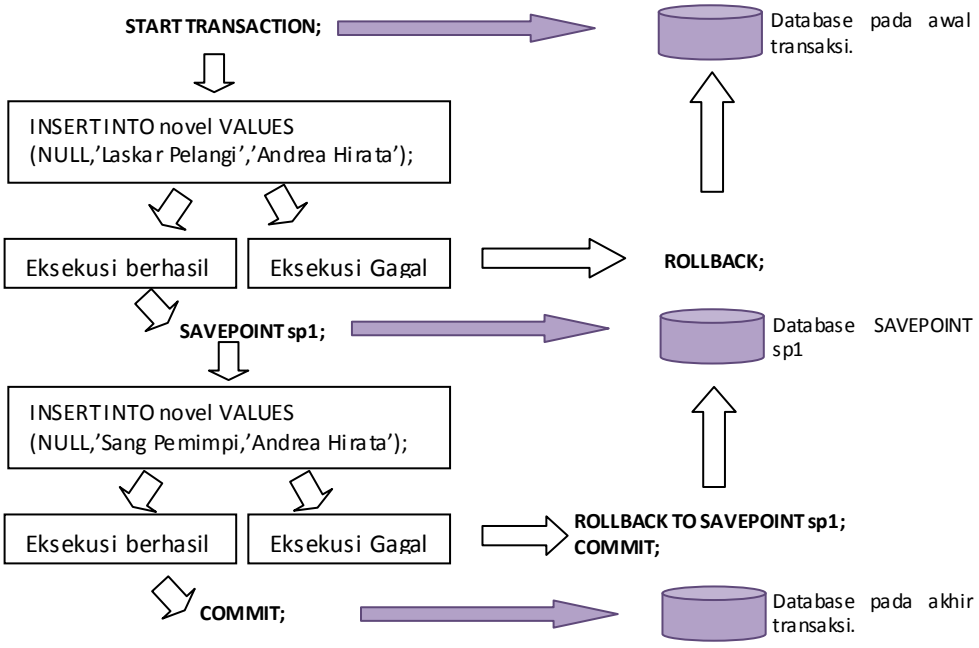
XI.4 Statement yang dapat mengakhiri Transaksi

Terdapat beberapa statement yang jika kita letakkan di dalam proses yang menggunakan Transaksi dapat secara otomatis menghentikan Transaksi, kemudian pernyataan tersebut dijalankan. Jika hal ini terjadi, maka semua proses dalam Transaksi akan di COMMIT dan tidak akan bisa di ROLLBACK. Pernyataan-pernyataan tersebut antara lain:

Pernyataan	Keterangan
ALTER TABLE	Mengubah definisi tabel
CREATE INDEX	Membuat indeks di dalam tabel
DROP DATABASE	Menghapus database dari mysql server
DROP INDEX	Menghapus indeks didalam tabel
DROP TABLE	Menghapus tabel dari database
LOCK TABLES	Menghalangi akses bersamaan pada tabel
RENAME TABLES	Mengubah nama tabel
SET AUTOCOMMIT=1	Menyetel autocommit ke dalam mode on
START TRANSACTION	Memulai Transaksi baru
TRUNCATE TABLE	Menghapus data dari tabel
UNLOCK TABLES	Menghilangkan LOCK TABLES

XI.5 SAVEPOINT

Jika dalam sebuah proses yang panjang, kita memerlukan beberapa titik untuk melakukan penyimpanan data. Titik itu dapat kita sebut SAVEPOINT. Beriku ini merupakan skema peletakkan SAVEPOINT didalam Transaksi:



XI.5.1 Pernyataan SAVEPOINT

Untuk menyatakan savepoint, pernyataannya adalah:

SAVEPOINT <nama savepoint>;

Dengan menggunakan pernyataan savepoint, maka semua proses yang terjadi sebelum pernyataan ini akan di commit, kemudian melanjutkan ke proses selanjutnya, tentunya pernyataan ini harus didalam Transaksi.

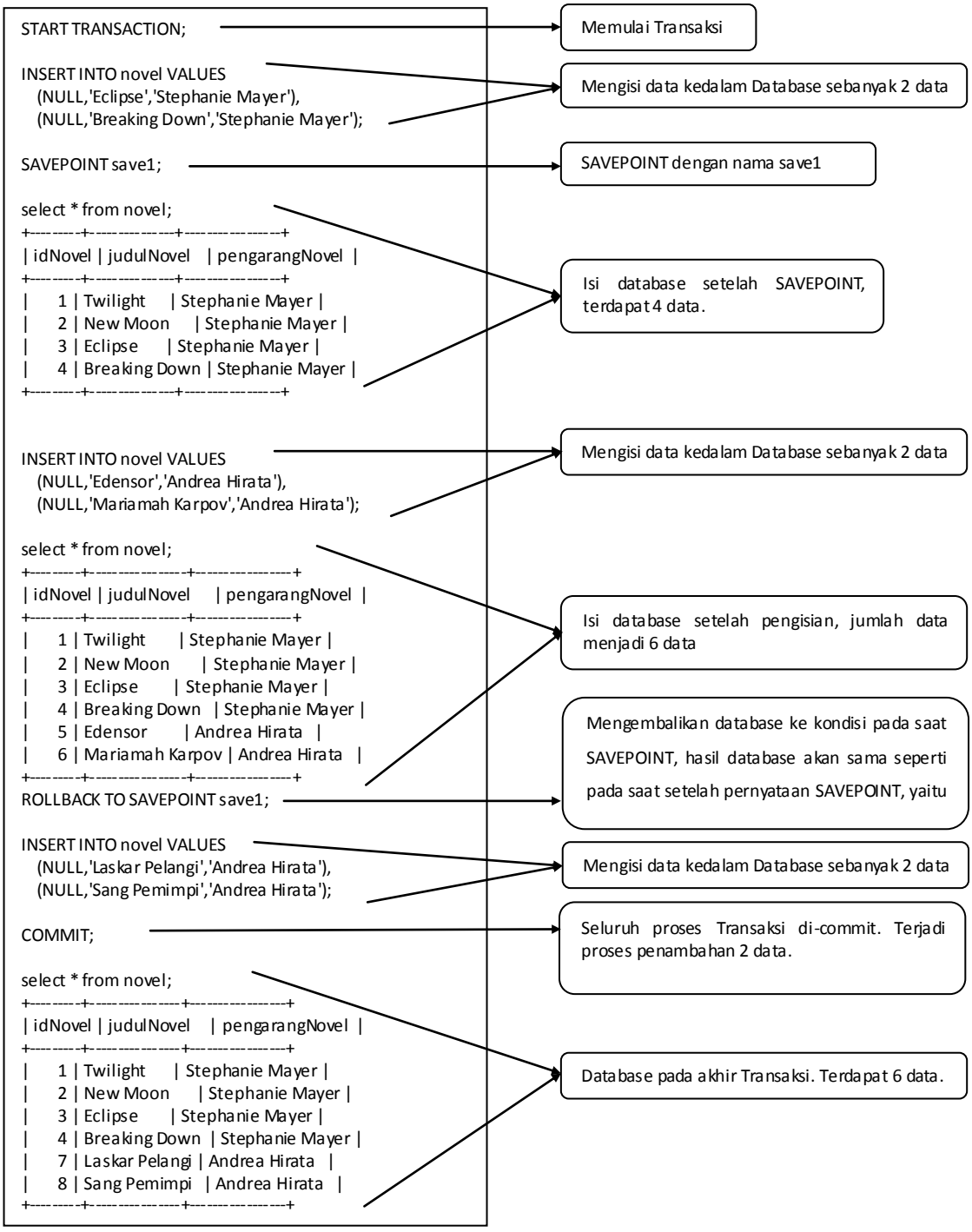
XI.5.2 Mengembalikan proses ke titik savepoint

Untuk mengembalikan kondisi database ke savepoint, maka digunakan pernyataan:

ROLLBACK TO SAVEPOINT <nama savepoint>;

Dengan ini, maka semua proses setelah savepoint sampai pernyataan ini di remove dan kondisi database kembali seperti pada saat stelah savepoint.

Berikut ini contoh penggunaan SAVEPOINT dan ROLLBACK TO SAVEPOINT:

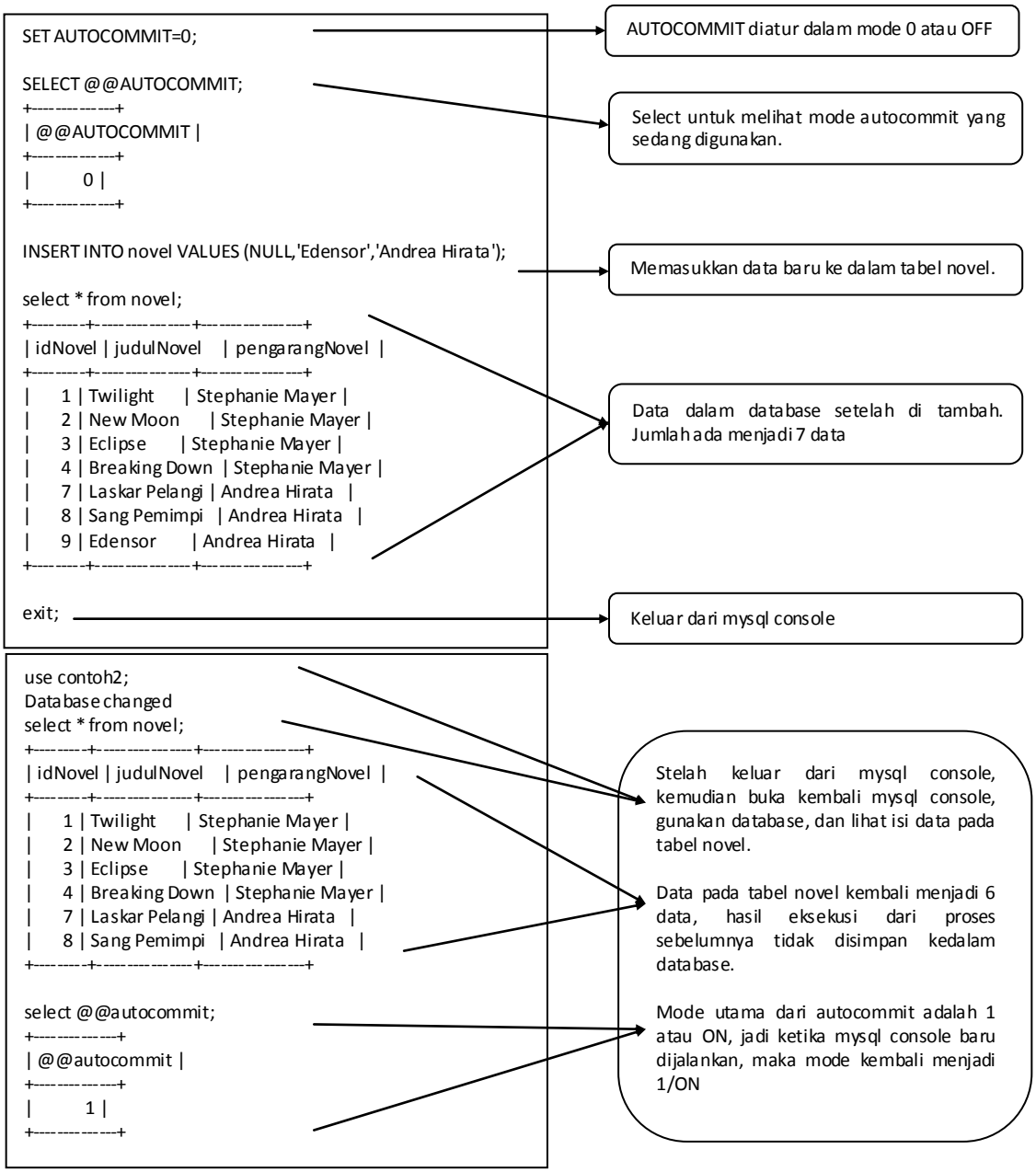

START TRANSACTION;
Memulai Transaksi
INSERT INTO novel VALUES
(NULL,'Eclipse','Stephanie Mayer'),
(NULL,'Breaking Down','Stephanie Mayer');
Mengisi data kedalam Database sebanyak 2 data
SAVEPOINT save1;
SAVEPOINT dengan nama save1
select * from novel;
+---------+---------------+-----------------+
| idNovel | judulNovel | pengarangNovel |
+---------+---------------+-----------------+
1	Twilight	Stephanie Mayer
2	New Moon	Stephanie Mayer
3	Eclipse	Stephanie Mayer
4	Breaking Down	Stephanie Mayer
+---------+---------------+-----------------+
Isi database setelah SAVEPOINT, terdapat 4 data.
INSERT INTO novel VALUES
(NULL,'Edensor','Andrea Hirata'),
(NULL,'Mariamah Karpov','Andrea Hirata');
Mengisi data kedalam Database sebanyak 2 data
select * from novel;
+---------+---------------+-----------------+
| idNovel | judulNovel | pengarangNovel |
+---------+---------------+-----------------+
1	Twilight	Stephanie Mayer
2	New Moon	Stephanie Mayer
3	Eclipse	Stephanie Mayer
4	Breaking Down	Stephanie Mayer
5	Edensor	Andrea Hirata
6	Mariamah Karpov	Andrea Hirata
+---------+---------------+-----------------+
Isi database setelah pengisian, jumlah data menjadi 6 data
ROLLBACK TO SAVEPOINT save1;
Mengembalikan database ke kondisi pada saat SAVEPOINT, hasil database akan sama seperti pada saat setelah pernyataan SAVEPOINT, yaitu
INSERT INTO novel VALUES
(NULL,'Laskar Pelangi','Andrea Hirata'),
(NULL,'Sang Pemimpi','Andrea Hirata');
Mengisi data kedalam Database sebanyak 2 data
COMMIT;
Seluruh proses Transaksi di-commit. Terjadi proses penambahan 2 data.
select * from novel;
+---------+---------------+-----------------+
| idNovel | judulNovel | pengarangNovel |
+---------+---------------+-----------------+
1	Twilight	Stephanie Mayer
2	New Moon	Stephanie Mayer
3	Eclipse	Stephanie Mayer
4	Breaking Down	Stephanie Mayer
7	Laskar Pelangi	Andrea Hirata
8	Sang Pemimpi	Andrea Hirata
+---------+---------------+-----------------+
Database pada akhir Transaksi. Terdapat 6 data.

XI.6 Mengatur mode AUTOCOMMIT

Pada dasarnya, tanpa adanya pernyataan COMMIT, setiap proses yang terjadi langsung di tulis ke dalam tabel. Untuk itu, kita perlu mengatur autocommit untuk tidak melakukan penulisan secara otomatis, sehingga proses yang terjadi tidak akan langsung ditulis ke tabel hingga ada pernyataan COMMIT. Untuk mengatur autocommit, pernyataan yang digunakan adalah:

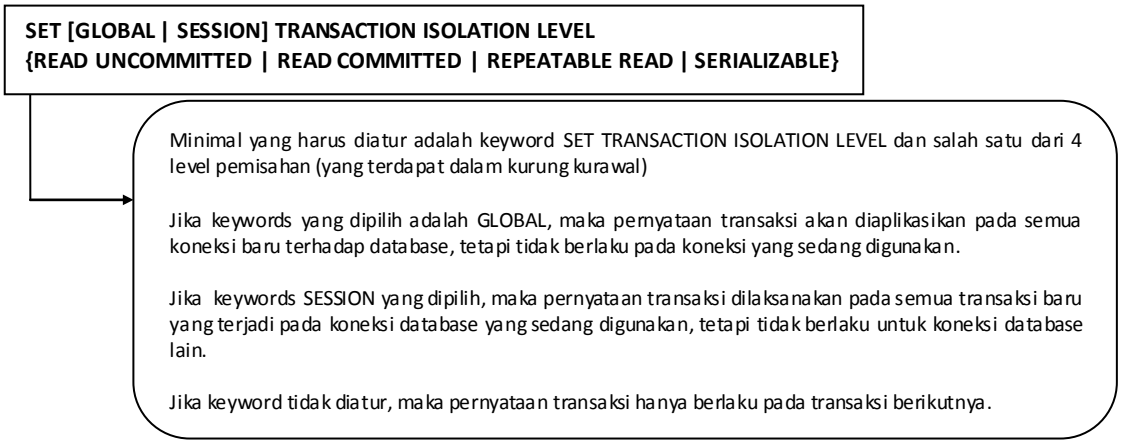
Pernyataan	Keterangan
SET AUTOCOMMIT=0	Untuk mematikan autocommit(OFF)
SET AUTOCOMMIT=1	Untuk menyalakan autocommit(ON)
@@AUTOCOMMIT;	Untuk melihat mode autocommit yang sedang digunakan

Berikut ini contoh pengaturan autocommit:


SET AUTOCOMMIT=0;
AUTOCOMMIT diatur dalam mode 0 atau OFF
SELECT @@AUTOCOMMIT;
+--------------+
| @@AUTOCOMMIT |
+--------------+
| 0 |
+--------------+
Select untuk melihat mode autocommit yang sedang digunakan.
INSERT INTO novel VALUES (NULL,'Edensor','Andrea Hirata');
Memasukkan data baru ke dalam tabel novel.
select * from novel;
+---------+---------------+-----------------+
| idNovel | judulNovel | pengarangNovel |
+---------+---------------+-----------------+
1	Twilight	Stephanie Mayer
2	New Moon	Stephanie Mayer
3	Eclipse	Stephanie Mayer
4	Breaking Down	Stephanie Mayer
7	Laskar Pelangi	Andrea Hirata
8	Sang Pemimpi	Andrea Hirata
9	Edensor	Andrea Hirata
+---------+---------------+-----------------+
Data dalam database setelah di tambah. Jumlah ada menjadi 7 data
exit;
Keluar dari mysql console
use contoh2;
Database changed
select * from novel;
+---------+---------------+-----------------+
| idNovel | judulNovel | pengarangNovel |
+---------+---------------+-----------------+
1	Twilight	Stephanie Mayer
2	New Moon	Stephanie Mayer
3	Eclipse	Stephanie Mayer
4	Breaking Down	Stephanie Mayer
7	Laskar Pelangi	Andrea Hirata
8	Sang Pemimpi	Andrea Hirata
+---------+---------------+-----------------+
Stelah keluar dari mysql console, kemudian buka kembali mysql console, gunakan database, dan lihat isi data pada tabel novel.
Data pada tabel novel kembali menjadi 6 data, hasil eksekusi dari proses sebelumnya tidak disimpan kedalam database.
select @@autocommit;
+--------------+
| @@autocommit |
+--------------+
| 1 |
+--------------+
Mode utama dari autocommit adalah 1 atau ON, jadi ketika mysql console baru dijalankan, maka mode kembali menjadi 1/ON

XI.7 Mengatur Pemisahan Level Transaksi

Selain dapat mengatur autocommit, mysql juga mengizinkan pengaturan pemisahan level Transaksi dengan menggunakan perintah sebagai berikut:



4 Level Pemisahan Transaksi:

Level Pemisahan Transaksi	Keterangan
READ UNCOMMITED	Mengizinkan dirty reads, nonrepeatable reads dan phantom read. Level ini lebih baik hanya digunakan pada database dengan data yang umum.
READ COMMITED	Hanya mengizinkan nonrepeatable read dan phantom read, tetapi tidak untuk dirty read.
REPEATABLE READ	Hanya mengizinkan phantom read,, merupakan level pemisahan transaksi utama pada mesin InnoDB.
SERIALIZABLE	Tidak mengizinkan adanya dirty read, nonrepeatable read dan phantom read. Jika level ini digunakan, maka database di kunci dari transaksi lain dan berjalan secara berurutan. Hanya level ini yang tersedia untuk tabel BDB.

Penjelasan:

- **Dirty Reads**

  Merupakan masalah yang terjadi ketika terdapat beberapa transaksi yang mencoba untuk mengakses satu tabel pada waktu yang sama ataupun hampir bersamaan.

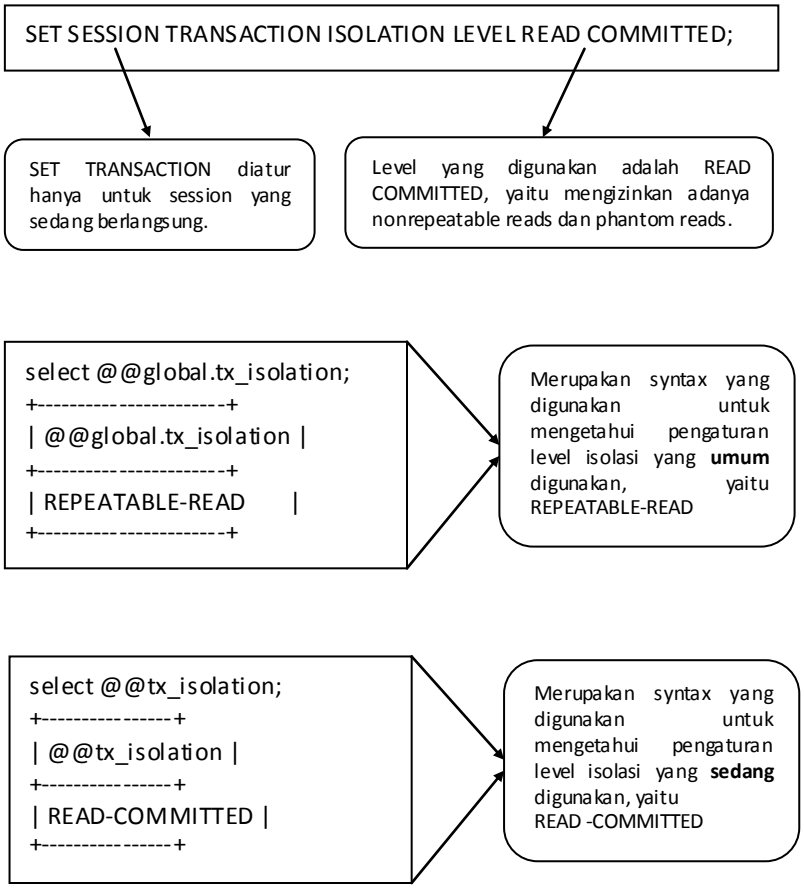
- **Nonrepeatable Reads**

  Merupakan masalah yang terjadi ketika ada sebuah transaksi yang sedang mengakses pembacaan terhadap suatu database, kemudian ketika proses tersebut masih berlangsung, ada proses lain yang mengupdate isi database. Hal ini menyebabkan ada data yang tidak terbaca oleh transaksi yang sedang membaca.

- **Phantom Reads**

  Merupakan masalah yang terjadi ketika transaksi pertama membaca database, kemudian transaksi kedua memprbaharui database, kemudian transaksi pertama membaca kembali database, maka transaksi pertama akan mendapatkan hasil yang berbeda.

Contoh Penggunaan SET TRANSACTION:

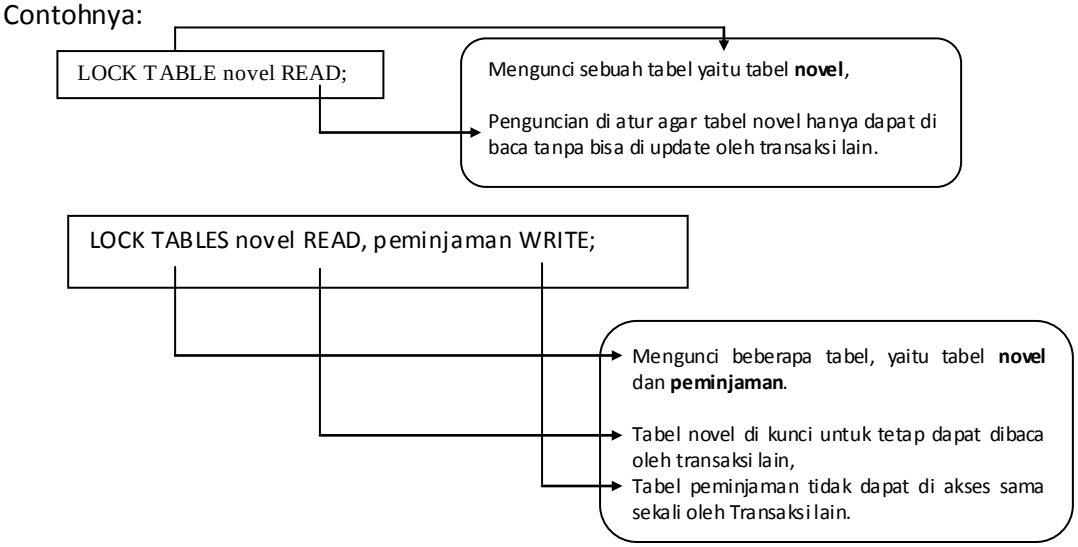


XI.8 Penguncian Tabel yang tidak menggunakan Transaksi

Transaksi hanya dapat digunakan pada tabel yang menggunakan mesin INNODB dan BDB tabel. Untuk melakukan penguncian atau pengaturan agar tidak terjadi **dirty reads, nonrepeatable reads** dan **phantom reads** ada syntax yang dapat digunakan, yaitu:

```
LOCK {TABLE | TABLES}
<table name> [AS <alias>] {READ [LOCAL] | [LOW_PRIORITY] WRITE}
[{, <table name> [AS <alias>] {READ [LOCAL] | [LOW_PRIORITY] WRITE}}...]
```

Contohnya:



XI.9 Pelepasan Kunci terhadap Tabel

Jika tebel telah di kunci, maka setelah proses selesai kunci harus dilepaskan agar transaksi lain bisa mengakses tabel. Syntax yang digunakan adalah:

