PERTEMUAN 5

INDEPENDENSI DATA DAN *DATABASE LANGUAGE*

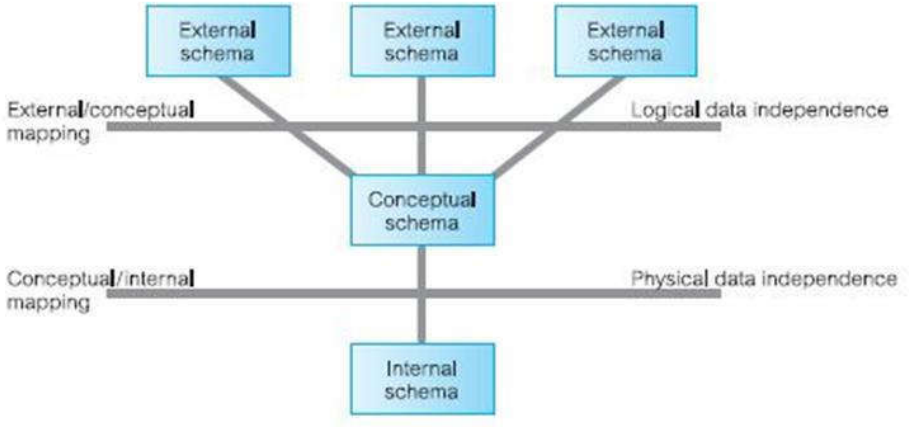
A. TUJUAN PEMBELAJARAN

Setelah mempelajari materi ini, diharapkan mahasiswa mampu memahami Independensi data, dan memahami *database* language.

B. URAIAN MATERI

1. Indepedensi Data

Independensi data adalah gagasan bahwa data yang dihasilkan dan disimpan harus disimpan terpisah dari aplikasi yang menggunakan data untuk komputasi dan presentasi. Tujuan utama arsitektur tiga tingkat adalah untuk memberikan independensi data, yang berarti bahwa tingkat atas tidak terpengaruh oleh perubahan ke tingkat yang lebih rendah.



Gambar 0.1 Struktur Independensi Data

Ada dua jenis independensi data: *logical* dan *physical*. Pada jenis *physical* program aplikasi secara logis tidak akan terpengaruh atau berubah ketika metode akses fisik atau struktur penyimpanan diubah. Contohnya :

a. Menggunakan perangkat penyimpanan yang baru seperti hard drive atau magnetic tapes
b. Memodifikasi Teknik organisasi file di *database*
c. Beralih ke struktur data yang berbeda
d. Mengubah metode akses
e. Mengubah indeks
f. Perubahan pada Teknik kompresi atau algoritma hashing
g. Perubahan lokasi *database* misalkan dari drive C ke drive D

Pada jenis *logical*, program aplikasi secara logis tidak akan terpengaruh atau berubah ketika perubahan dilakukan pada struktur tabel yang mempertahankan nilai tabel yang asli (mengubah urutan kolom atau memasukkan kolom). Contohnya :

a. Menambah/mengubah/menghapus atribut, entitas atau hubungan yang baru dimungkinkan tanpa penulisan ulang program aplikasi yang ada.
b. Menggabungkan dua record menjadi satu.
c. Memecah record yang ada menjadi dua atau lebih.

Perbedaan antara *physical* dan *logical* independensi data :

Tabel 0.1 Perbedaan *Physical* dan *Logical* Independensi Data

Independensi data *logical*	**Independensi data *physical***
Independensi Data Logis terutama berkaitan dengan struktur atau perubahan definisi data.	Terutama berkaitan dengan penyimpanan data.
Sulit karena pengambilan data terutama tergantung pada struktur logis data.	Mudah untuk dikembalikan.
Dibandingkan dengan logika independensi *physical*, lebih sulit untuk mendapatkan indepedensi data *logical*	Dibandingkan dengan logika independensi *logical*, sangat mudah untuk mendapatkan indepedensi data *physical*
Anda perlu membuat perubahan dalam program Aplikasi jika bidang baru	Perubahan pada *level* fisik biasanya tidak membutuhkan perubahan pada

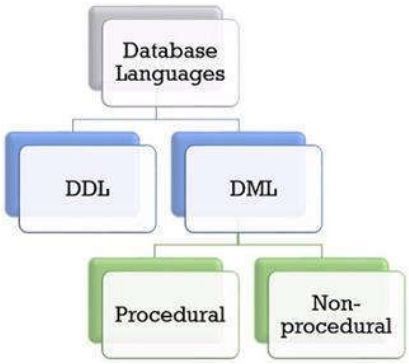
ditambahkan atau dihapus dari *database*.	*level* program Aplikasi.
Modifikasi pada tingkat logis menjadi signifikan setiap kali struktur logis dari *database* diubah.	Modifikasi yang dilakukan di tingkat *internal* mungkin diperlukan atau mungkin tidak diperlukan untuk meningkatkan kinerja struktur.
Berkaitan dengan *schema* konseptual	Berkaitan dengan *schema internal*
Contoh: tambah/ubah/hapus atribut baru	Contoh: mengubah Teknik kompresi, hashing, algortima, perangkat penyimpanan, dll

Keuntungan dari independensi data yaitu:

a. Membantu untuk meningkatkan kualitas data
b. Pemeliharaan sistem *database* menjadi terjangkau
c. Penegakan standar dan peningkatan keamanan *database*
d. Tidak perlu mengubah struktur data di program aplikasi
e. Izinkan pengembang untuk fokus pada struktur umum *Database* daripada mengkhawatirkan implementasi *internal*
f. Memungkinkan untuk meningkatkan keadaan yang tidak rusak atau tidak terbagi
g. Ketidaksesuaian *database* sangat berkurang.
h. Mudah melakukan modifikasi pada *level* fisik diperlukan untuk meningkatkan kinerja *system*.

2. *Database* Languages

Bahasa *Database* adalah sekumpulan pernyataan, yang digunakan untuk mendefinisikan dan memanipulasi *database*. Didalam Bahasa *database* terdapat 2 jenis, yaitu Data Definition Language (DDL) dan Data Manipulation Language (DML). Kedua Bahasa tersebut biasa digunakan dalam Bahasa SQL.



Gambar 0.2 Struktur *Database* Language

a. Data Definition Language (DDL)

Data Definition Language (DDL) mendefinisikan pernyataan untuk mengimplementasikan skema *database*. Jika pemisahan yang jelas antara *level* logis (konseptual) dan fisik (*internal*) tidak ada, maka DDL mendefinisikan skema logis dan fisik dan juga mendefinisikan pemetaan antara skema logis dan fisik.

Jika ada pemisahan yang jelas antara skema logis dan fisik, maka Storage Definition Language (SDL) digunakan untuk menentukan skema fisik. Tapi hari ini, sebagian besar DBMS relasional tidak menggunakan SDL untuk menentukan skema fisik. Alih-alih, skema fisik ditentukan menggunakan kombinasi fungsi dan parameter yang memungkinkan DBA memetakan data ke penyimpanan.

Setelah menerapkan skema logis dan fisik, sekarang waktunya untuk menentukan skema tampilan (eksternal). Untuk itu View Definition Language (VDL) digunakan, yang juga memetakan skema tampilan ke skema logis. Tapi hari ini di sebagian besar DBMS, DDL menjalankan peran VDL.

Kumpulan perintah di DDL yang digunakan untuk mengimplementasikan skema *database* adalah sebagai berikut:

a) CREATE: Perintah ini digunakan untuk membangun relasi (tabel) dalam *database*. Contoh dari create table.

```
CREATE TABLE operator(
    id VARCHAR (20) NOT NULL,
    nama VARCHAR (50) NOT NULL,
    password VARCHAR(100) NOT NULL,
    created_at DATETIME NOT NULL,
    updated_at TIMESTAMP,
    PRIMARY KEY (id)
);
```

Gambar 0.3 Contoh CREATE (DDL)

b) ALTER : Perintah ini digunakan untuk merekonstruksi data di *database*. Berikut contoh dari Alter:

```
ALTER TABLE barang ADD jenis varchar(10);
```

```
ALTER TABLE barang DROP jenis;
```

```
ALTER TABLE barang CHANGE Satuan Model varchar(6);
```

Gambar 0.4 Contoh Perintah ALTER ADD, DROP, dan CHANGE (DDL)

c) DROP : Perintah ini digunakan untuk menghapus relasi dalam *database* atau seluruh *database*.

```
DROP DATABASE Toko;
```

```
DROP TABLE Barang;
```

Gambar 0.5 Contoh Perintah Drop (DDL)

d) TRUNCATE : Perintah ini menghapus semua entri dari relasi tetapi menjaga struktur relasi tetap aman dalam *database*.

```
TRUNCATE TABLE mahasiswa;
```

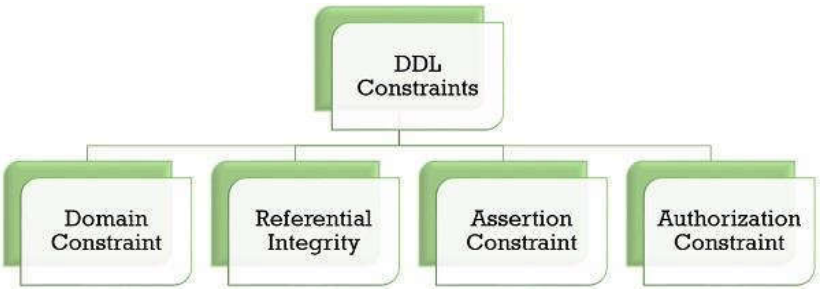
Gambar 0.6 Contoh Perintah TRUNCATE (DDL)

e) RENAME : Perintah ini mengganti nama relasi dalam *database*.

```
RENAME TABLE mahasiswa to siswa
```

Gambar 0.7 Contoh Perintah RENAME (DDL)

DDL juga mendefinisikan beberapa batasan konsistensi pada data, yang disimpan dalam *database*. Di bawah ini adalah daftar batasan yang ditentukan oleh DDL:



Gambar 0.8 Struktur Data Definition language (DDL)

1) *Domain Constraint* : Sebuah *domain* dengan nilai yang memungkinkan harus dikaitkan dengan setiap atribut (misalnya, tipe integer, tipe karakter, tipe tanggal / waktu). Mendeklarasikan atribut sebagai *domain* tertentu bertindak sebagai batasan pada nilai yang dapat diambil. Batasan *domain* adalah bentuk paling dasar dari batasan integritas. Mereka diuji dengan mudah oleh sistem setiap kali item data baru dimasukkan ke dalam *database*.
2) *Referential Integrity* : Ada kasus di mana kami ingin memastikan bahwa nilai yang muncul dalam satu relasi untuk himpunan atribut tertentu juga muncul dalam himpunan atribut tertentu di relasi lain (integritas referensial). Misalnya, departemen yang terdaftar untuk setiap kursus harus benar-benar ada. Lebih tepatnya, nilai nama departemen dalam catatan mata kuliah harus muncul dalam atribut nama departemen dari beberapa catatan hubungan departemen. Modifikasi *database* dapat menyebabkan pelanggaran integritas referensial. Jika batasan integritas referensial dilanggar, prosedur normalnya adalah menolak tindakan yang menyebabkan pelanggaran.

3) Assertion *Constraint* : Assertion adalah kondisi apa pun yang harus selalu dipenuhi oleh *database*. Batasan *domain* dan batasan integritas referensial adalah bentuk pernyataan khusus. Namun demikian, ada banyak kendala yang tidak dapat kita ungkapkan hanya dengan menggunakan bentuk-bentuk khusus ini. Misalnya, “Setiap jurusan harus memiliki minimal lima mata kuliah yang ditawarkan setiap semester” harus dinyatakan sebagai pernyataan. Saat pernyataan dibuat, sistem akan mengujinya untuk validitas. Jika pernyataan tersebut valid, maka setiap modifikasi di masa mendatang ke *database* diizinkan hanya jika tidak menyebabkan pernyataan tersebut dilanggar.
4) Authorization *Constraint* : untuk membedakan di antara pengguna sejauh mana jenis akses mereka diizinkan pada berbagai nilai data dalam *database*. Diferensiasi ini diekspresikan dalam istilah otorisasi, yang paling umum adalah: otorisasi baca, yang memungkinkan pembacaan, tetapi bukan modifikasi data; masukkan otorisasi, yang memungkinkan penyisipan data baru, tetapi tidak dapat mengubah data yang sudah ada; memperbarui otorisasi, yang memungkinkan modifikasi, tetapi tidak menghapus, data; dan hapus otorisasi, yang memungkinkan penghapu san data. Kami dapat menetapkan pengguna semua, tidak ada, atau kombinasi dari jenis otorisasi ini.

DDL, seperti bahasa pemrograman lainnya, mendapat masukan beberapa instruksi (pernyataan) dan menghasilkan beberapa keluaran. Keluaran DDL ditempatkan di kamus data, yang berisi metadata, yaitu, data tentang data. Kamus data dianggap sebagai jenis tabel khusus yang hanya dapat diakses dan diperbarui oleh sistem basis data itu sendiri (bukan pengguna biasa). Sistem *database* berkonsultasi dengan kamus data sebelum membaca atau memodifikasi data aktual.

b. Data Manipulation *Language* (DML)

Data Manipulation *Language* (DML) memiliki sekumpulan pernyataan yang memungkinkan pengguna untuk mengakses dan memanipulasi data dalam *database*. Menggunakan pernyataan DML pengguna dapat retrieve, insert, *delete* atau modify informasi dalam *database*.

Data Manipulation *Language* memiliki 2 jenis yaitu :

1) Procedural DML : DML prosedural dianggap sebagai bahasa tingkat rendah, dan mereka menentukan data apa yang dibutuhkan dan bagaimana mendapatkan data tersebut. DML prosedural juga disebut DML satu-per-waktu karena menerima dan memproses setiap catatan secara terpisah.

2) DML Non-prosedural adalah bahasa tingkat tinggi, dan mereka secara tepat menentukan data apa yang diperlukan tanpa menentukan cara untuk mengaksesnya. DML non-prosedural juga disebut DML set-a-time; ini karena DML non-prosedural dapat mengambil beberapa catatan menggunakan satu perintah DML. DML non-prosedural juga disebut bahasa deklaratif. Karena hanya mendeklarasikan data apa yang diperlukan alih-alih menentukan bagaimana cara mendapatkannya. Umumnya, pengguna akhir menggunakan DML tingkat tinggi (non-prosedural) untuk menentukan persyaratan mereka.

   Berikut adalah beberapa pernyataan dari DML :

   a) Select : perintah ini digunakan untuk membaca atau mengambil data dari *database*.

```
Select * from namatabel;
```

```
Select * value1,value2,value3 from namatabel;
```

Gambar 0.9 Contoh Perintah SELECT (DML)

   b) Insert : Perintah ini digunakan untuk menambahkan data baru ke *database*.

```
Insert into namatabel values ('value1','value2','...');
```

Gambar 0.10 Contoh Perintah INSERT (DML)

   c) *Update* : perintah ini digunakan untuk mengubah data dari *database*.

```
Update namatabel set field1='value1' where kondisi and kondisi;
```

```
Update namatabel set field1='value1',field2='value2';
```

```
Update namatabel set field1='value1' where kondisi;
```

Gambar 0.11 Contoh Perintah UPDATE (DML)

d) *Delete* : perintah ini digunakan untuk menghapus data dari *database*.

```
Delete from namatabel;
```

```
Delete from namatabel where kondisi;
```

Gambar 0.12 Contoh Perintah DELETE (DML)

c. Data *Control Language* (DCL)

Data *Control Language* atau yang bisa disebut DCL adalah salah satu sub Bahasa dari SQL yang berfungsi untuk melakukan pengontrolan pada data dan server dari *database*, seperti manipulasi pengguna dan hak aksesnya. Ada dua perintah yang termasuk dari DCL, yaitu *GRANT* dan *REVOKE*.

a) *GRANT* : Perintah ini digunakan untuk memberikan hak akses dari admin ke *user* atau pengguna. Hak akses tersebut bisa berupa hak untuk CREATE, SELECT, *DELETE* atau *UPDATE*, dan hak khusus lainnya yang berhubungan dengan *database*. Contoh *syntax* dari *GRANT* :

```
GRANT priviliges ON tbname TO user;
```

Gambar 0.13 Contoh Perintah GRANT (DCL)

b) *REVOKE* : Perintah ini digunakan untuk mencabut hak akses yang telah diberikan ke pengguna atau *user*. Contoh *syntax* dari *REVOKE* :

```
REVOKE priviliges ON tbname from user;
```

Gambar 0.14 Contoh Perintah REVOKE (DCL)

d. Transactional Control Language (TCL)

*Transaction*al *Control Language* atau yang bisa disebut TCL adalah suatu Bahasa yang digunakan untuk mengelola dan mengontrol transaksi dalam *database*. Transaksi mewakili setiap perubahan dalam *database*. Bahasa ini juga memungkinkan pernyataan untuk dikelompokkan bersama menjadi transaksi *logical*. *Transaction*al *Control Language* atau TCL memiliki tiga perintah, yaitu *COMMIT*, *SAVEPOINT*, dan *ROLLBACK*.

a) *COMMIT* : Perintah *COMMIT* digunakan untuk menyimpan transaksi data secara permanen di *database*. Pada saat melakukan perintah seperti *UPDATE*, INSERT atau *DELETE* transaksi sebenarnya belum dilakukan secara permanen. Yang mana artinya operasi atau perintah tersebut masih bisa di *rollback*/ dibatalkan. Berikut adalah contoh untuk menggunakan perintah *COMMIT* :

Masukkan data terlebih dahulu.

```
INSERT INTO mahasiswa
VALUES
(21400200,"faqih","bandung"),
(21400201,"ina","jakarta"),
(21400202,"anto","semarang"),
(21400203,"dani","padang");
```

Selanjutnya cek terlebih dahulu data yang telah di*input*kan.

```
> SELECT * FROM mahasiswa;
+----------+-------+----------+
| nim      | nama  | alamat   |
+----------+-------+----------+
| 21400200 | faqih | bandung  |
| 21400201 | ina   | jakarta  |
| 21400202 | anto  | semarang |
| 21400203 | dani  | padang   |
+----------+-------+----------+
4 rows in set (0.00 sec)
```

Untuk memulai menggunakan *COMMIT* dimulai dengan *syntax* berikut.

```
START TRANSACTION;
```

```
> START TRANSACTION;
Query OK, 0 rows affected (0.00 sec)

> INSERT INTO mahasiswa VALUES (21400210,"Jaka","Kalimantan");
Query OK, 1 row affected (0.00 sec)

> COMMIT;
Query OK, 0 rows affected (0.07 sec)

> SELECT * FROM mahasiswa;
+----------+-------+------------+
| nim      | nama  | alamat     |
+----------+-------+------------+
| 21400200 | faqih | bandung    |
| 21400201 | ina   | jakarta    |
| 21400202 | anto  | semarang   |
| 21400203 | dani  | padang     |
| 21400210 | Jaka  | Kalimantan |
+----------+-------+------------+
5 rows in set (0.00 sec)
```

Gambar 0.15 Contoh Perintah COMMIT (TCL)

b) *SAVEPOINT* : Perintah ini digunakan untuk menyimpan sementara transaksi sehingga Anda dapat melakukan *rollback* ke titik itu kapan pun diperlukan. *Syntax* untuk menggunakan perintah *SAVEPOINT* yaitu :

```
SAVEPOINT SAVEPOINT_NAME;
```

Gambar 0.16 Contoh Perintah SAVEPOINT (TCL)

Perintah ini hanya berfungsi dalam pembuatan *SAVEPOINT* di antara semua pernyataan transaksional. Perintah *ROLLBACK* digunakan untuk

membatalkan sekelompok transaksi. *Syntax* untuk memutar kembali ke *SAVEPOINT* seperti yang ditunjukkan di bawah ini :

```
ROLLBACK TO SAVEPOINT_NAME;
```

Gambar 0.17 Contoh Perintah ROLLBACK to SAVEPOINT (TCL)

c) *ROLLBACK* : Perintah ini digunakan untuk mengembalikan *database* ke bentuk awal, *ROLLBACK* juga bisa digunakan untuk melompat ke suatu titik tertentu yang didefinisikan didalam *SAVEPOINT*. *SAVEPOINT* adalah tanda khusus didalam transaksi yang memungkinkan semua perintah yang sudah dijalankan setelah dibuat untuk di *ROLLBACK*, mengembalikan status transaksi dari *database* ke keadaan pada saat *SAVEPOINT*. Untuk menggunakan *ROLLBACK* harus dimulai dengan *syntax* berikut :

```
START TRANSACTION;
```

Lalu dilanjutkan dengan perintah-perintah berikut :

```
> START TRANSACTION;
Query OK, 0 rows affected (0.00 sec)

> SELECT * FROM mahasiswa;
+----------+-------+------------+
| nim      | nama  | alamat     |
+----------+-------+------------+
| 21400200 | faqih | bandung    |
| 21400201 | ina   | jakarta    |
| 21400202 | anto  | semarang   |
| 21400203 | dani  | padang     |
| 21400210 | Jaka  | Kalimantan |
+----------+-------+------------+
5 rows in set (0.00 sec)

> INSERT INTO mahasiswa VALUES (21400211,"Fitri","Surabaya");
Query OK, 1 row affected (0.00 sec)

> SELECT * FROM mahasiswa;
+----------+-------+------------+
| nim      | nama  | alamat     |
+----------+-------+------------+
| 21400200 | faqih | bandung    |
| 21400201 | ina   | jakarta    |
| 21400202 | anto  | semarang   |
| 21400203 | dani  | padang     |
| 21400210 | Jaka  | Kalimantan |
| 21400211 | Fitri | Surabaya   |
+----------+-------+------------+
6 rows in set (0.00 sec)

> ROLLBACK;
Query OK, 0 rows affected (0.11 sec)

> SELECT * FROM mahasiswa;
+----------+-------+------------+
| nim      | nama  | alamat     |
+----------+-------+------------+
| 21400200 | faqih | bandung    |
| 21400201 | ina   | jakarta    |
| 21400202 | anto  | semarang   |
| 21400203 | dani  | padang     |
| 21400210 | Jaka  | Kalimantan |
+----------+-------+------------+
5 rows in set (0.00 sec)
```

Gambar 0.18 Contoh Perintah ROLLBACK (TCL)

Perbedaan antara *COMMIT* dan *ROLLBACK*, yaitu :

Tabel 0.2 Perbedaan antara COMMIT dan ROLLBACK

COMMIT	*ROLLBACK*
COMMIT secara permanen menyimpan perubahan yang dibuat oleh transaksi saat ini.	*ROLLBACK* membatalkan perubahan yang dibuat oleh transaksi saat ini.
Transaksi tidak dapat membatalkan perubahan setelah eksekusi *COMMIT*.	Transaksi mencapai keadaan sebelumnya setelah *ROLLBACK*.
Ketika transaksi berhasil, *COMMIT* diterapkan.	Saat transaksi dibatalkan, *ROLLBACK* terjadi.

C. SOAL LATIHAN/TUGAS

1. Ada berapa jenis dari independensi data? Jelaskan!
2. Apa saja keuntungan dari independensi data?
3. Buatlah sebuah *database* mahasiswa lalu jalankan perintah-perintah dari DDL dan DML!
4. Apa saja Batasan dari DDL?

D. REFERENSI

Connolly, T., & Begg, C. (2005). *Database System: A Practical Approach to Design, Implementation, and Management, Fourth Edition.* Harlow: Pearson Education Limited.

Coronel, C., & Morris, S. (2017). *Database System: Design, Implementation, & Management, 13th Edition.* Boston: Cengage Learning, Inc.

Silberschatz, A., Korth, H. F., & Sudarshan, S. (2011). *Database System Concept ; Sixth Edition.* New York: McGraw-Hill.

GLOSARIUM

Storage Definition Language adalah untuk menspesifikasikan *internal* skema. Bila digunakan 2 skema (*conceptual* dan *internal*), maka DDL hanya menspesifikasikan skema *conceptual* dan diperlukan bahasa SDL untuk menspesifikasikan *internal* skema.

View Definition Language adalah untuk menspesifikasikan *user view*s dan memetakan (mapping) ke skema *nonceptual*. Bila digunakan 3 skema (*view*, *conceptual* dan *internal*).

***Domain* constrant** adalah Sebuah *domain* dengan nilai yang memungkinkan harus dikaitkan dengan setiap atribut (misalnya, tipe integer, tipe karakter, tipe tanggal / waktu).

Referential integrity adalah Ada kasus di mana kami ingin memastikan bahwa nilai yang muncul dalam satu relasi untuk himpunan atribut tertentu juga muncul dalam himpunan atribut tertentu di relasi lain (integritas referensial).

Assertion *constraint* adalah kondisi apa pun yang harus selalu dipenuhi oleh *database*.

Authorization *constraint* adalah untuk membedakan di antara pengguna sejauh mana jenis akses mereka diizinkan pada berbagai nilai data dalam *database*.

Grant adalah Perintah ini digunakan untuk memberikan hak akses dari admin ke *user* atau pengguna.

Revoke adalah Perintah ini digunakan untuk mencabut hak akses yang telah diberikan ke pengguna atau *user*.

Commit adalah Perintah *COMMIT* digunakan untuk menyimpan transaksi data secara permanen di *database*.

Save Point adalah Perintah ini digunakan untuk menyimpan sementara transaksi sehingga Anda dapat melakukan *rollback* ke titik itu kapan pun diperlukan.

Rollback adalah Perintah ini digunakan untuk mengembalikan *database* ke bentuk awal, *ROLLBACK* juga bisa digunakan untuk melompat ke suatu titik tertentu yang didefinisikan didalam *SAVEPOINT*