

PENGANTAR BASIS DATA DAN DATA DEFINITION LANGUAGE : CREATE, DROP, ALTE

1

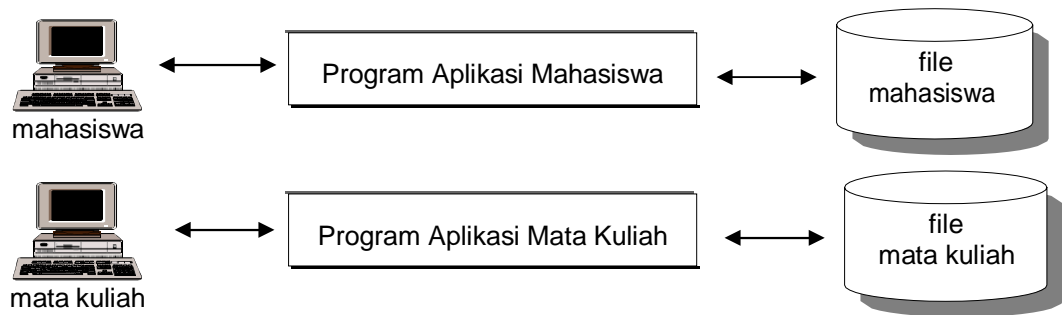
Obyektif :

1. Menjelaskan perbedaan antara file tradisional dan file manajemen basis data
 2. Menjelaskan keuntungan dan kerugian apabila menggunakan file manajemen basis data
 3. Memahami konsep basis data dan istilah yang termasuk didalamnya
 4. Mengetahui bahasa yang digunakan untuk pengoperasian basis data
 5. Mengetahui dan memahami perintah yang terdapat pada Data Definition Language
 6. Dapat menggunakan perintah CREATE, DROP, dan ALTER
-

Basis data menyediakan fasilitas atau mempermudah dalam menghasilkan informasi yang digunakan oleh pemakai untuk mendukung pengambilan keputusan. Hal inilah yang menjadikan alasan dari penggunaan teknologi basis data pada saat sekarang (dunia bisnis). Berikut ini contoh penggunaan Aplikasi basis data dalam dunia bisnis :

- Bank : Pengelolaan data nasabah, akunting, semua transaksi perbankan
- Bandara : Pengelolaan data reservasi, penjadualan
- Universitas : Pengelolaan pendaftaran, alumni
- Penjualan : Pengelolaan data customer, produk, penjualan
- Pabrik : Pengelolaan data produksi, persediaan barang, pemesanan, agen
- Kepegawaian: Pengelolaan data karyawan, gaji, pajak
- Telekomunikasi : Pengelolaan data tagihan, jumlah pulsa

Sistem Pemrosesan File



Gambar 1. Sistem pemrosesan file untuk suatu Universitas

Keterangan :

File mahasiswa : **Mhs** (npm, nama, alamat, tgl_lahir)

MataKul (kd_mk, nama_mk, sks)

File MataKuliah : **MataKul** (kd_mk, nama, sks)

Sebelumnya, sistem yang digunakan untuk mengatasi semua permasalahan bisnis, menggunakan pengelolaan data secara tradisional dengan cara menyimpan record-record pada file-file yang terpisah, yang disebut juga sistem pemrosesan file. Dimana masing-masing file diperuntukkan hanya untuk satu program aplikasi saja. Perhatikan gambar 1 mengenai suatu universitas yang mempunyai dua sistem yakni sistem yang memproses data mahasiswa dan sistem yang mengelola data mata kuliah.

Kelemahannya dari sistem pemrosesan file ini antara lain :

1. Timbulnya data rangkap (*redundancy data*) dan Ketidakkonsistensi data (*Inconsistency data*)

Karena file-file dan program aplikasi disusun oleh programmer yang berbeda, sejumlah informasi mungkin memiliki duplikasi dalam beberapa file. Sebagai contoh nama mata kuliah dan sks dari

mahasiswa dapat muncul pada suatu file memiliki record-record mahasiswa dan juga pada suatu file yang terdiri dari record-record mata kuliah. Kerangkapan data seperti ini dapat menyebabkan pemborosan tempat penyimpanan dan biaya akses yang bertambah. Disamping itu dapat terjadi inkonsistensi data. Misalnya, apabila terjadi perubahan jumlah sks mata kuliah, sedangkan perubahan hanya diperbaiki pada file mata kuliah dan tidak diperbaiki pada file mahasiswa. Hal ini dapat mengakibatkan kesalahan dalam laporan nilai mahasiswa.

2. Kesukaran dalam Mengakses Data

Munculnya permintaan-permintaan baru yang tidak diantisipasi sewaktu membuat program aplikasi, sehingga tidak memungkinkan untuk pengambilan data.

3. Data terisolir (*Isolation Data*)

Karena data tersebar dalam berbagai file, dan file-file mungkin dalam format –format yang berbeda, akan sulit menuliskan program aplikasi baru untuk mengambil data yang sesuai.

4. Masalah Pengamanan (*Security Problem*)

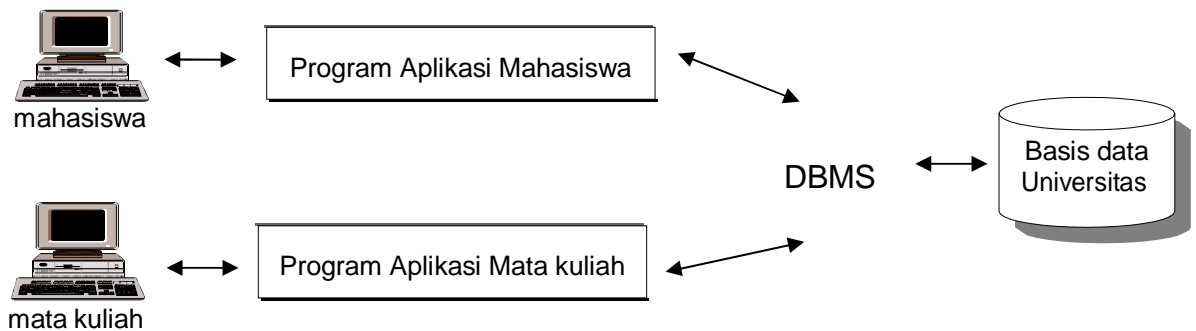
Tidak semua pemakai diperbolehkan mengakses seluruh data. Bagian Mahasiswa hanya boleh mengakses file mahasiswa. Bagian Mata kuliah hanya boleh mengakses file mata kuliah, tidak boleh mengakses file mahasiswa. Tetapi sejak program-program aplikasi ditambahkan secara ad-hoc maka sulit melaksanakan pengamanan seperti yang diharapkan.

5. Data Dependence

Apabila terjadi perubahan atau kesalahan pada program aplikasi maka pemakai tidak dapat mengakses data.

Sistem Basis data

Seiring dengan berjalannya waktu, lambat laun sistem pemrosesan file mulai ditinggalkan karena masih bersifat manual, yang kemudian dikembangkanlah sistem pemrosesan dengan pendekatan basis data.



Gambar 2. Sistem basis data untuk suatu universitas

Keterangan :

Mhs (Npm, nama, alamat, tgl_lahir)

Mt_kul (kd_mk, nama_mk,sks)

Perhatikan gambar 2 di atas. Pada sistem ini record-record data disimpan pada satu tempat yakni basis data dan diantara program aplikasi maupun pemakai terdapat DBMS (*Database Management System*).

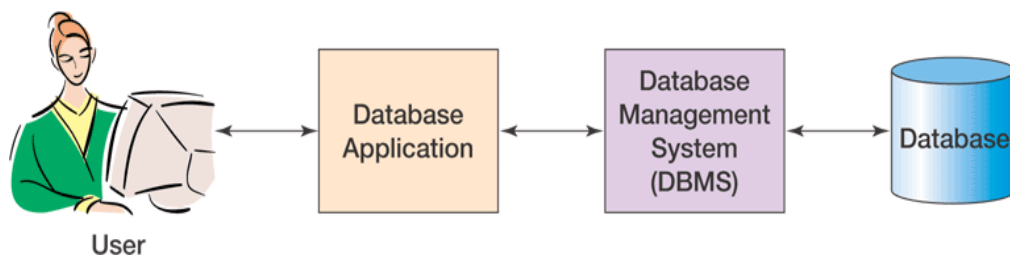
Konsep Dasar Basis Data

Data adalah representasi fakta dunia nyata yang mewakili suatu objek seperti manusia (pegawai, mahasiswa, pembeli), barang, hewan, peristiwa, konsep, keadaan, dan sebagainya yang direkam dalam bentuk angka, huruf, simbol, teks, gambar, bunyi atau kombinasinya.

Basis Data adalah sekumpulan data yang terintegrasi yang diorganisasikan untuk memenuhi kebutuhan para pemakai di dalam suatu organisasi.

DBMS (*Database Management System*) adalah Perangkat Lunak yang menangani semua pengaksesan ke basis data

Sistem Basis Data terdiri dari basis data dan DBMS.



Gambar 3. Sistem Basis Data

Istilah - Istilah Dasar Basis Data

Enterprise

Suatu bentuk organisasi seperti : bank, universitas, rumah sakit, pabrik, dsb.

Data yang disimpan dalam basis data merupakan data operasional dari suatu enterprise.

Contoh data operasional : data keuangan, data mahasiswa, data pasien

Entitas

Suatu obyek yang dapat dibedakan dari lainnya yang dapat diwujudkan dalam basis data.

Contoh Entitas dalam lingkungan bank terdiri dari : Nasabah, Simpanan, Hipotik

Contoh Entitas dalam lingkungan universitas terdiri dari : Mahasiswa, mata kuliah

Kumpulan dari entitas disebut **Himpunan Entitas**

Contoh : semua nasabah, semua mahasiswa

Atribut (Elemen Data)

Karakteristik dari suatu entitas.

Contoh : Entitas Mahasiswa atributnya terdiri dari Npm, Nama, Alamat, Tanggal lahir.

Nilai Data (Data Value)

Isi data / informasi yang tercakup dalam setiap elemen data.

Contoh Atribut Nama Mahasiswa dapat berisi Nilai Data : Diana, Sulaeman, Lina

Kunci Elemen Data (*Key Data Element*)

Tanda pengenal yang secara unik mengidentifikasikan entitas dari suatu kumpulan entitas.

Contoh Entitas Mahasiswa yang mempunyai atribut-atribut npm, nama, alamat, tanggal lahir menggunakan Kunci Elemen Data npm.

Record Data

Kumpulan Isi Elemen data yang saling berhubungan.

Contoh : kumpulan atribut npm, nama, alamat, tanggal lahir dari Entitas Mahasiswa berisikan : "10200123", "Sulaeman", "Jl. Sirsak 28 Jakarta", "8 Maret 1983".

Keuntungan Sistem Basis Data

1. Terkontrolnya kerangkapan data

Dalam basis data hanya mencantumkan satu kali saja field yang sama yang dapat dipakai oleh semua aplikasi yang memerlukannya.

2. Terpeliharanya keselarasan (kekonsistenan) data

Apabila ada perubahan data pada aplikasi yang berbeda maka secara otomatis perubahan itu berlaku untuk keseluruhan

3. Data dapat dipakai secara bersama (*shared*)
Data dapat dipakai secara bersama-sama oleh beberapa program aplikasi (secara *batch* maupun *on-line*) pada saat bersamaan.
4. Dapat diterapkan standarisasi
Dengan adanya pengontrolan yang terpusat maka DBA dapat menerapkan standarisasi data yang disimpan sehingga memudahkan pemakaian, pengiriman maupun pertukaran data.
5. Keamanan data terjamin
DBA dapat memberikan batasan-batasan pengaksesan data, misalnya dengan memberikan password dan pemberian hak akses bagi pemakai (misal : *modify, delete, insert, retrieve*)
6. Terpeliharanya integritas data
Jika kerangkapan data dikontrol dan kekonsistenan data dapat dijaga maka data menjadi akurat
7. Terpeliharanya keseimbangan (keselarasan) antara kebutuhan data yang berbeda dalam setiap aplikasi
Struktur basis data diatur sedemikian rupa sehingga dapat melayani pengaksesan data dengan cepat
8. *Data independence* (kemandirian data)
Dapat digunakan untuk bermacam-macam program aplikasi tanpa harus merubah format data yang sudah ada

Kelemahan Sistem Basis Data

- Memerlukan tenaga spesialis
- Kompleks
- Memerlukan tempat yang besar
- Mahal

Pengguna Basis Data

1. *System Engineer*

Tenaga ahli yang bertanggung jawab atas pemasangan Sistem Basis Data, dan juga mengadakan peningkatan dan melaporkan kesalahan dari sistem tersebut kepada pihak penjual

2. *Database Administrator (DBA)*

Tenaga ahli yang mempunyai tugas untuk mengontrol sistem basis data secara keseluruhan, meramalkan kebutuhan akan sistem basis data, merencanakannya dan mengaturnya.

Tugas DBA :

- Mengontrol DBMS dan *software-software*
- Memonitor siapa yang mengakses basis data
- Mengatur pemakaian basis data
- Memeriksa *security, integrity, recovery* dan *concurency*

Program Utilitas yang digunakan oleh DBA :

- *Loading Routines*
Membangun versi utama dari basis data
- *Reorganization Routines*
Mengatur / mengorganisasikan kembali basis data
- *Journaling Routines*
Mencatat semua operasi pemakaian basis data
- *Recovery Routines*
Menempatkan kembali data, sebelum terjadinya kerusakan
- *Statistical Analysis Routines*
Membantu memonitor kehandalan sistem

3. *End User* (Pemakai Akhir)

Ada beberapa jenis (tipe) pemakai terhadap suatu sistem basis data yang dapat dibedakan berdasarkan cara mereka berinteraksi terhadap sistem :

a. Programmer aplikasi

Pemakai yang berinteraksi dengan basis data melalui *Data Manipulation Language* (DML), yang disertakan (*embedded*) dalam program yang ditulis pada bahasa pemrograman induk (seperti C, pascal, cobol, dll)

b. Pemakai Mahir (*Casual User*)

Pemakai yang berinteraksi dengan sistem tanpa menulis modul program. Mereka menyatakan *query* (untuk akses data) dengan bahasa *query* yang telah disediakan oleh suatu DBMS

c. Pemakai Umum (*End User / Naïve User*)

Pemakai yang berinteraksi dengan sistem basis data melalui pemanggilan satu program aplikasi permanen (*executable program*) yang telah ditulis (disediakan) sebelumnya

d. Pemakai Khusus (*Specialized/Sophisticated User*)

Pemakai yang menulis aplikasi basis data non konvensional, tetapi untuk keperluan-keperluan khusus seperti aplikasi AI, Sistem Pakar, Pengolahan Citra, dll, yang bisa saja mengakses basis data dengan atau tanpa DBMS yang bersangkutan.

Structure Query Language

Structure Query Language (SQL) merupakan komponen bahasa *relational database system*. SQL merupakan bahasa baku (ANSI/SQL), *non procedural*, dan berorientasi himpunan (*set-oriented language*). SQL dapat digunakan baik secara interaktif atau ditempelkan (*embedded*) pada sebuah program aplikasi.

Komponen-Komponen SQL

a. **Data Definition Language (DDL) :**

Digunakan untuk mendefinisikan data dengan menggunakan perintah : *create, drop, alter*.

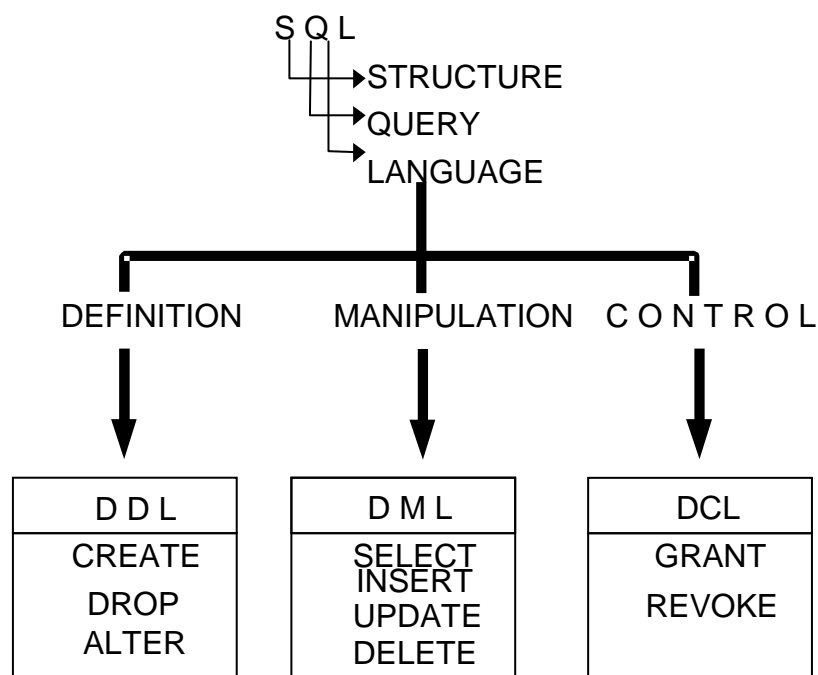
b. **Data Manipulation Language (DML) :**

Digunakan untuk memanipulasi data dengan menggunakan perintah :
select, insert, update, delete.

Data Manipulation Language merupakan bagian terpadu bahasa SQL. Perintah-perintahnya dapat dibuat secara interaktif atau ditempelkan pada sebuah program aplikasi. Pemakai hanya perlu menentukan 'APA' yang ia inginkan, DBMS menentukan 'BAGAIMANA' cara mendapatkannya.

c. **Data Control Language (DCL) :**

Digunakan untuk mengontrol hak para pemakai data dengan perintah :
grant, revoke



DATA DEFINITION LANGUAGE :

CREATE, DROP, ALTER

1. CREATE TABLE

Fungsi : membuat tabel

Sintaks : **CREATE TABLE** tbname

```
(col 1      data type  data spec,  
 col 2      data type  data spec,  
 .  
 .  
 PRIMARY KEY (col1,.....))
```

Contoh :

```
CREATE TABLE PERSONEL  
(REGNO    CHAR(10)  NOT NULL,  
 NAME     CHAR(45)  NOT NULL,  
 ADDRESS  CHAR(45),  
 BIRTH    DATE      NOT NULL WITH DEFAULT,  
 PRIMARY KEY (REGNO))
```

NULL

Spesifikasi *NULL, NOT NULL, NOT NULL WITH DEFAULT*

NULL :

dapat diinterpretasikan sebagai nilai yang tidak diketahui atau tidak tersedianya suatu nilai. Null bukan berarti kosong (blank) atau 0 (Nol)

NOT NULL :

pemakai atau program harus memberikan nilai-nilai pada saat memasukkan record

NOT NULL WITH DEFAULT :

nilai *default* disimpan pada saat record dimasukkan tanpa nilai yang ditentukan untuk kolom ini.

Nilai *default*-nya :

Nol	untuk tipe field NUMERIC
Blank	untuk tipe field CHARACTER
CURRENT DATE	untuk tipe field DATE
CURRENT TIME	untuk tipe field TIME

Pada saat membuat tabel, salah satu atribut tersebut di atas dispesifikasikan pada sebuah kolom.

2. CREATE VIEW

Fungsi : membuat tabel view.

View merupakan bentuk alternatif penyajian data dari satu atau lebih tabel. View dapat berisi semua atau sebagian kolom yang terdapat pada tabel dimana kolom tersebut didefinisikan.

Tujuan membuat view :

- Meningkatkan keamanan data
- Meningkatkan kemandirian data
- Penyederhanaan bagi end user (data yang sedikit, nama-nama kolom yang baru dan dapat dibaca dengan lebih baik)

Properti :

- Tidak terdapatnya data tambahan
- View mencakup subset kolom dan / atau baris
- View dapat berisikan data dari beberapa tabel dan / atau tabel-tabel view lainnya
- View dapat berisikan perolehan data, misal : nilai rata-rata
- Manipulasi data melalui view terbatas

Sintaks : **CREATE VIEW viewname (column1, column2,.....)
AS SELECT statement FROM tbname
[WITH CHECK OPTION]**

Keterangan :

View-name : nama view yang akan dibuat.

Column : nama atribut untuk view

Statement : atribut yang dipilih dari tabel basis data.

Tabel-name : nama tabel basis data.

Contoh :

```
CREATE VIEW VPERSON (REGNO, NAME) AS  
SELECT REGNO, NAME FROM PAUL.PERSONEL
```

3. CREATE INDEX

Fungsi : membuat index

Sintaks : **CREATE [UNIQUE] INDEX indexname**
ON nama_table (nama_kolom)

Contoh :

```
CREATE UNIQUE INDEX PRSONIDX  
ON PERSONEL(REGNO)
```

Dengan indeks memungkinkan suatu tabel diakses dengan urutan tertentu tanpa harus merubah urutan fisik dari datanya dan dapat pula diakses secara cepat melalui indeks yang dibuat berdasar nilai field tertentu. Spesifikasi UNIQUE akan menolak key yang sama dalam file.

4. DROP TABLE

Fungsi : menghapus Tabel

Sintaks : **DROP TABLE tbname**

Contoh : DROP TABLE PERSONEL

Dengan perintah itu obyek lain yang berhubungan dengan tabel tersebut otomatis akan dihapus atau tidak akan berfungsi seperti :

- semua record dalam tabel akan terhapus
- index dan view pada tabel akan hilang
- deskripsi tabel akan hilang

5. DROP VIEW

Fungsi : menghapus view

Sintaks : **DROP VIEW viewname**

Contoh : DROP VIEW VPERSON

6. DROP INDEX

Fungsi : menghapus index

Sintaks : **DROP INDEX indexname**

Contoh : DROP INDEX PRSONIDX

7. ALTER

Fungsi : merubah atribut pada suatu tabel

Sintaks : **ALTER TABLE tname**

MODIFY (nama_kolom tipe_kolom)

ADD (nama_kolom tipe_kolom [[before,
nama_kolom]])

DROP (nama_kolom tipe_kolom)

Contoh : merubah Tabel TABX dengan menambah Field D.

ALTER TABLE TABX

ADD D CHAR(3)

Contoh Kasus DDL :

- **Membuat Tabel (CREATE TABLE)**

1. CREATE TABLE S

```
(Sn Char(5) NOT NULL,  
Sname Char(20) NOT NULL,  
Status Smallint NOT NULL,  
City Char(15) NOT NULL);
```

2. CREATE TABLE P

```
(Pn Char(6) NOT NULL,  
Pname Char(20) NOT NULL,  
Color Char(6) NOT NULL,  
Weight Smallint NOT NULL);
```

3. CREATE TABLE SP

```
(Sn Char(5) NOT NULL,  
Pn Char(6) NOT NULL,  
QTY INTEGER NOT NULL);
```

4. CREATE UNIQUE INDEX Sidx ON S(Sn);
CREATE UNIQUE INDEX Pidx ON P(Pn);
CREATE INDEX Sdx ON SP(Sn);
CREATE INDEX Pdx ON SP(Pn);

- **Modifikasi Table P dengan perintah :**

```
RENAME COLUMN P.COLOR TO WARNA  
ALTER TABLE P ADD (City CHAR(15) NOT NULL)
```


- **Membuat View (CREATE VIEW)**

1. Membuat view untuk suplier yang statusnya lebih besar dari 15

```
CREATE VIEW GOOD_SUPPLIERS
AS SELECT Sn, Status, City FROM S
WHERE Status > 15;
```

2. Membuat view yang berisi supplier yang tinggal di Paris

```
CREATE VIEW Paris_Suppliers
AS SELECT * FROM Suppliers
WHERE City = ' Paris '
```

3. Membuat view dengan mengganti nama_atributnya

```
CREATE VIEW Parts (PNum, Part_Name, WT)
AS SELECT P#, Pname, Weight FROM Part
WHERE COLOR = 'Red'
```

DATA MANIPULATION LANGUAGE : INSERT, SELECT, UPDATE, DELETE

2

Obyektif :

7. Mengetahui dan memahami perintah yang terdapat pada Data Manipulation Language
 8. Dapat menggunakan perintah INSERT, SELECT, UPDATE, dan DELETE
-

1. INSERT

FUNGSI : MENAMBAH BARIS (RECORD) BARU

Sintaks : **INSERT INTO tbname**
(col1, ...) VALUES (value1, ...)

Catatan :

Sintaks tersebut dapat digunakan jika jumlah kolom = jumlah nilai, tetapi jika dalam tabel semua kolom akan diisi dapat digunakan sintaks berikut ini :

Sintaks : **INSERT INTO tbname**
VALUES (value1, value2, ...)

Nilai-nilai diisikan sebanyak kolom yang terdapat di tabel tersebut.

2. UPDATE

Fungsi : merubah record

Sintaks : **UPDATE** tbname
SET field = ekspresi
WHERE kondisi

3. DELETE

Fungsi : menghapus record

Sintaks : **DELETE FROM** tbname

WHERE kondisi

4. SELECT

Fungsi : menampilkan record

Sintaks : **SELECT** [DISTINCT] colname FROM tbname
[WHERE kondisi]
[GROUP BY kondisi]
[HAVING kondisi]
[ORDER BY kondisi]

Contoh Kasus DML :

- **Menambah record (INSERT)**

```
INSERT INTO S VALUES ('S1','Smith',20,'London');
```

```
INSERT INTO S VALUES ('S2','Jones',10,'Paris');
```

```
INSERT INTO S VALUES ('S3','Blake',30,'Paris')
```

Tabel S, P dan SP isikan dengan data-data sebagai berikut :

TABEL S

Sn	Sname	Status	City
S1	Smith	20	London
S2	Jones	10	Paris
S3	Blake	30	Paris
S4	Clark	20	London
S5	Adams	30	Athens

TABEL P

Pn	Pname	Warna	Weight	City
P1	Nut	Red	12	London
P2	Bolt	Green	17	Paris
P3	Screw	Blue	17	Rome
P4	Screw	Red	14	London
P5	Cam	Blue	12	Paris
P6	Cog	Red	19	London

TABEL SP

Sn	Pn	Qty
S1	P1	300
S1	P2	200
S1	P3	400
S1	P4	200
S1	P5	100
S1	P6	100
S2	P1	300
S2	P2	400
S3	P2	200
S4	P2	200
S4	P4	300
S4	P5	400

- **Merubah record (UPDATE)**

1. Merubah data (record) pada tabel P yang mempunyai nomor part P2, warnanya dirubah menjadi Kuning dan beratnya ditambah 5

UPDATE P

SET Warna = 'Yellow',

Weight = Weight + 5

WHERE Pn = 'P2'

2. Merubah record pada tabel S, statusnya menjadi dua kali status awal untuk supplier yang bertempat tinggal di kota London

```
UPDATE S
    SET Status = 2 * Status
    WHERE City = 'London'
```

- **Menghapus record (DELETE)**

Menghapus record pada tabel S yang nomor supplier-nya S5

```
DELETE FROM S

    WHERE Sn ='S5'
```

- **Menampilkan record (SELECT 1 tabel)**

1. Menampilkan semua data supplier

```
SELECT * FROM S
atau
SELECT Sn, Sname, Status, City FROM S
```

2. Menampilkan semua nilai Pn pada tabel SP

```
SELECT Pn FROM SP
```

3. Menampilkan nomor supplier dan status untuk supplier yang tinggal di Paris

```
SELECT Sn, Status FROM S
    WHERE City ='Paris'
```

4. Menampilkan no.supplier yang tinggal di Paris dengan status > 20

```
SELECT Sn FROM S
      WHERE City ='Paris" AND Status > 20
```

5. Menampilkan jumlah pengiriman P1

```
SELECT COUNT(*) FROM SP
      WHERE Pn = 'P1'
```

6. Perintah untuk menghindari hasil data yang sama terulang kembali (distinct)

```
SELECT DISTINCT Pn FROM SP
```

7. Menampilkan no.supplier dan status bagi supplier yang tinggal di Paris dalam urutan status menurun

```
SELECT Sn,Status FROM S
      WHERE City = 'Paris'
      ORDER BY Status desc
```

8. Menampilkan no.Part dari semua part yang dipasok oleh lebih dari seorang supplier

```
SELECT Pn FROM SP
      GROUP BY Pn
      HAVING COUNT(*) > 1
```

9. Menampilkan semua part yang nomornya dimulai dengan huruf C

```
SELECT * FROM P  
WHERE Pname LIKE 'C%'
```


DATA MANIPULATION LANGUAGE : AGGREGATE FUNCTION, GROUP BY, HAVING

3

Obyektif :

9. Mengetahui dan memahami perintah Fungsi Agregasi, Group By, dan Having
10. Dapat menggunakan perintah Fungsi Agregasi, Group By, dan Having

Selain mengambil data dalam basis data dengan kriteria tertentu, sering juga diperlukan berbagai perhitungan yang bersifat ringkasan. Agregasi dalam SQL merupakan proses untuk mendapatkan nilai dari sekumpulan data yang telah dikelompokkan. Pengelompokan data didasarkan pada kolom atau kombinasi kolom yang dipilih. Fungsi AGREGASI digunakan untuk melakukan operasi pada kelompok-kelompok baris data sehingga menghasilkan satu baris data untuk setiap kelompok baris data yang ada.

Fungsi Agregasi adalah berikut ini :

Fungsi	Deskripsi
AVG	Menghitung rata-rata nilai ekspresi yang tidak NULL. B.U: SELECT AVG(kolom) FROM tabel
SUM	Menghitung total nilai ekspresi yang tidak NULL. B.U: SELECT SUM(kolom) FROM tabel
COUNT	Menghitung banyaknya baris yang dipilih secara query. B.U: SELECT COUNT(kolom) FROM tabel

	Mencari nilai maksimal dari suatu kolom.
MAX	B.U: SELECT MAX(kolom) FROM tabel
	Mencari nilai minimal dari suatu kolom.
MIN	B.U: SELECT MIN(kolom) FROM tabel
	Menampilkan nilai dari record yang pertama dari suatu kolom/field.
FIRST	B.U: SELECT FIRST(kolom) AS [ekspresi] FROM tabel
	Menampilkan nilai dari record terakhir dari suatu kolom/field.
LAST	B.U: SELECT LAST(kolom) AS [ekspresi] FROM tabel

Fungsi COUNT mengabaikan adanya data yang sifatnya NULL VALUE.

Misalnya untuk menghitung jumlah pegawai, rata-rata gaji, dan total gaji pegawai dari tabel Pegawai. Maka perintahnya adalah :

```
SELECT  COUNT(*),
        AVG(gaji),
        SUM(gaji)
FROM    Pegawai;
```

- **Fungsi agregasi — GROUP BY**

Digunakan untuk membentuk group/kelompok dari tupel-tupel yang memiliki nilai yang sama. Hasil pengelompokan dirutkan secara ascending.

Misalnya untuk menghitung jumlah pegawai, rata-rata gaji, total gaji untuk setiap kelompok golongan pegawai. Maka sintaks perintahnya adalah :

```
SELECT  COUNT(*),
        AVG(gaji),
        SUM(gaji)
FROM    Pegawai
GROUP BY golongan;
```

- **Fungsi agregasi – HAVING**

Klausa HAVING digunakan untuk membatasi baris yang dihasilkan oleh fungsi agregasi GROUP BY.

Misalnya untuk menghitung rata-rata gaji dan total gaji dalam tabel Pegawai untuk setiap kelompok golongan pegawai yang rata-rata gaji > 1000000. Maka perintahnya adalah :

```
SELECT  AVG(gaji),
        SUM(gaji)
FROM    Pegawai
GROUP BY golongan
HAVING  AVG(gaji)>1000000;
```

- **Fungsi agregasi – ORDER BY**

Digunakan untuk mengurutkan query dengan nilai yang berisi dalam satu atau lebih kolom. Secara *default* data diurutkan secara ascending (menaik).

Ascending adalah pengurutan dari yang terkecil ke yang terbesar.

Descending adalah pengurutan yang terbesar ke yang terkecil.

Contoh :

- (1) Mengurutkan data berdasarkan nama pegawai. Maka sintaks perintahnya adalah:

```
SELECT      *
FROM        Pegawai
ORDER BY    nama;
```

- (2) Mengurutkan data pegawai berdasarkan nama, dan diurutkan lagi berdasarkan golongan secara menurun. Maka sintaks perintahnya adalah :

```
SELECT      *
FROM        Pegawai
ORDER BY    nama,
            golongan DESC;
```

Contoh Kasus :

Tabel PERSON

Nama	Umur
Cynthia	24
Kevin	32
Dave	45
Jenny	20
Michael	24
David	

Dengan menggunakan tabel di atas,

1. Untuk menampilkan umur tertua (maksimal) :

SELECT MAX(umur) FROM PERSON;

Jawab : **45**

2. Untuk menampilkan umur termuda (minimal) :

SELECT MIN(umur) FROM PERSON;

Jawab : **20**

3. Untuk menghitung banyaknya data :

SELECT COUNT(*) FROM PERSON; —→ data dari seluruh record

Jawab : **6**

SELECT COUNT(umur) FROM PERSON; —→ data di kolom umur, tidak termasuk yang null

Jawab : **5**

4. Untuk menghitung total umur :

SELECT SUM(umur) FROM PERSON;

Jawab : **135**

5. Untuk mengurutkan data :

SELECT * FROM PERSON ORDER BY nama;

Jawab :

Tabel PERSON

Nama	Umur
Cynthia	24
Dave	45
David	
Jenny	20
Kevin Michael	32
Michael	24

→ Diurutkan secara
menaik (ascending)

SELECT * FROM PERSON ORDER BY nama DESC;

(mengurutkan data berdasarkan nama secara
menurun/descending)

6. Untuk menampilkan record pertama:

**SELECT FIRST(umur) AS umur_termuda
FROM PERSON ORDER BY umur;**

Jawab : **20**

7. Untuk menampilkan record terakhir :

**SELECT LAST(umur) AS umur_tertua
FROM PERSON ORDER BY umur;**

Jawab : **45**

DATA CONTROL LANGUAGE : GRANT DAN REVOKE

4

Obyektif :

11. Mengetahui dan memahami perintah Data Control Language
 12. Dapat menggunakan perintah Grant dan Revoke
-

Data Control Language (DCL) merupakan perintah-perintah yang dapat digunakan untuk menjaga keamanan basis data. Perintah tersebut dapat dipakai untuk menentukan akses basis data hanya dapat dilakukan oleh orang-orang tertentu dan dengan macam akses yang dibatasi pula. Adapun Objek-Objek DCL dalam Mysql diantaranya :

a. Tabel USER dari database MySQL

Tabel user adalah tabel yang ada dalam database MySQL. Tabel user hanya diperuntukkan bagi seorang Administrator (root). Tabel user bersifat global, apapun perubahan yang terjadi pada tabel ini akan mempengaruhi jalannya keseluruhan system MySQL.

Dari kolom yang didapat dari tabel user untuk MySQL yang embeded pada PHP Instant adalah sebagai berikut :

Field	Type	Null	Key	Default	Extra
Host	Varchar(60) binary		Pri		
User	Varchar(16) binary		Pri		
Password	Varchar(16)				
Select_priv	Enum('N','Y')			N	
Insert_priv	Enum('N','Y')			N	
Update_priv	Enum('N','Y')			N	
Delete_priv	Enum('N','Y')			N	
Drop_priv	Enum('N','Y')			N	
Reload_priv	Enum('N','Y')			N	
Shutdown_priv	Enum('N','Y')			N	
Process_priv	Enum('N','Y')			N	
File_priv	Enum('N','Y')			N	
Grant_priv	Enum('N','Y')			N	
References_priv	Enum('N','Y')			N	
Index_priv	Enum('N','Y')			N	
Alter_priv	Enum('N','Y')			N	
Show_db_priv	Enum('N','Y')			N	
Create_priv	Enum('N','Y')			N	

Penjelasan fungsi :

Field	Penjelasan
Host	Field ini berisi alamat host komputer user. Field ini dapat diisi "localhost" jika hanya mengizinkan user yang bersangkutan hanya dapat mengakses dari komputer yang sama dengan server MySQLnya Dapat juga diisi dengan "%" jika user MySQL dapat mengakses dari

	komputer lain dalam jaringan. Jika hanya mengizinkan user dapat mengakses dari satu komputer saja, maka masukkan nilainya dengan nomor IP address pada komputer klien sehingga user yang bersangkutan tidak dapat mengakses server MySQL dari komputer lain termasuk komputer tempat MySQL terinstall, kecuali hanya dari komputer yang nomor IP-nya sesuai dengan yang dimasukkan pada kolom host.
User	Kolom ini berisi username dari user yang terdaftar. Username tersebut digunakan untuk melakukan login pada server MySQL sebagai user biasa. Username dapat berupa pencampuran karakter (huruf, angka, simbol karakter)
Password	Berisi password dari username yang terdaftar Isi dari kolom ini akan di-enkripsi dengan standard password yang dimiliki oleh MySQL server
Select_priv	Berfungsi untuk memberikan hak user untuk menampilkan data (menggunakan perintah SELECT) Jika diisi 'Y' berarti user diijinkan untuk menggunakan perintah SELECT Jika diisi 'T' berarti user tidak diijinkan untuk menggunakan perintah SELECT
Insert_priv	Berfungsi untuk memberikan hak user untuk memasukkan data (menggunakan perintah INSERT) Jika diisi 'Y' berarti user diijinkan untuk menggunakan perintah INSERT Jika diisi 'T' berarti user tidak diijinkan untuk menggunakan perintah INSERT
Update_priv	Berfungsi untuk memberikan hak user untuk meremajakan / memperbarui (menggunakan perintah UPDATE) Jika diisi 'Y' berarti user diijinkan untuk menggunakan perintah UPDATE Jika diisi 'T' berarti user tidak diijinkan untuk menggunakan perintah UPDATE
Delete_priv	Berfungsi untuk memberikan hak user untuk menghapus / mengosongkan data (menggunakan perintah DELETE) Jika diisi 'Y' berarti user diijinkan untuk menggunakan perintah DELETE Jika diisi 'T' berarti user tidak diijinkan untuk menggunakan perintah DELETE
Drop_priv	Berfungsi untuk memberikan hak user untuk menghapus database, tabel atau kolom (menggunakan perintah DROP) Jika diisi 'Y' berarti user diijinkan untuk menggunakan perintah DROP Jika diisi 'T' berarti user tidak diijinkan untuk menggunakan perintah DROP
Reload_priv	Berfungsi untuk memberikan hak user untuk melakukan refresh server (menggunakan perintah RELOAD) Jika diisi 'Y' berarti user diijinkan untuk menggunakan perintah RELOAD Jika diisi 'T' berarti user tidak diijinkan untuk menggunakan perintah RELOAD
Shutdown_priv	Berfungsi untuk memberikan hak user untuk dapat mematikan daemon server MySQL Jika diisi 'Y' berarti user diijinkan untuk menghentikan server Jika diisi 'T' berarti user tidak diijinkan untuk menghentikan server
Process_priv	Berfungsi untuk memberikan hak user untuk dapat melihat proses

- **Sintaks Umum**

1. Menampilkan data pada kolom host, user dan password saja

```
SELECT host, user, password FROM user;
```

2. Menghapus semua user tanpa identitas

```
DELETE FROM user WHERE user='';
```

3. Mengganti password

```
UPDATE user SET password=password('xxxxxxxxxx')  
WHERE user='root';
```

4. Perintah FLUSH.

```
FLUSH PRIVILEGES;
```

Perintah FLUSH PRIVILEGES adalah suatu perintah untuk mengaktifkan perubahan-perubahan yang terjadi pada user, seperti hak akses, penggantian password pada user, dsb. Perintah FLUSH PRIVILEGES ini hukumnya wajib dilaksanakan setelah Anda melakukan perubahan (apapun juga) secara langsung ke dalam tabel user atau ke dalam database mysql.

- **Contoh penggunaan :**

1. Menggunakan database mysql

```
use mysql;
```

2. Menambah user ='Winda' dan password='w1nd'

insert into user (User,Password) values('Winda','w1nd');

3. Menambah user ='Febe' dan password='4567'

insert into user (User,Password) values('Febe','4567');

4. Menambah user ='Elfrida' dan password='9812'

insert into user (User,Password) values('Elfrida','9812');

5. menampilkan user dan password tabel user

select user,password from user;

6. Menghapus semua user tanpa identitas

Delete from user where user='';

7. Menampilkan semua isi dari tabel user

Select * from user;

8. Mengubah password untuk user winda dengan 'w1nd4'

Update user set password ='w1nd4' where user='Winda';

9. Mengubah password untuk user Febe dengan 'f3b3'

Update user set password ='f3b3' where user='Febe';

10. Mengubah password untuk user Elfrida dengan 'fr1d4'

Update user set password =' fr1d4' where user='Elfrida';

b. Tabel Tables_Priv dari database MySQL

Tabel_priv berfungsi mengatur tabel apa saja yang dapat diakses oleh seorang user, berikut jenis izin aksesnya. Tingkat akses hanya untuk tabel. Pada prinsipnya hanya bekerja seperti db table, kalau tidak digunakan untuk tabel sebagai ganti database.

- **Sintaks Umum :**

SELECT, INSERT, UPDATE, DELETE, CREATE, DROP, GRANT, REFERENCES, INDEX, ALTER, CREATE VIEW, SHOW VIEW

- **Contoh penggunaan :**

1. Insert into tables_priv(User,table_name,table_priv)
values('Febe,Elfrida','Point_Of_Sales.jenis','select');
2. Insert into tables_priv(User,table_name,table_priv)
values('Winda','Point_Of_Sales.jenis','select,insert');
3. Insert into tables_priv(User,table_name,table_priv)
values('Elfrida','Point_Of_Sales.item','select,insert,update');
4. Insert into tables_priv (User,Table_name,Table_priv)
values ('Admin','Point_Of_Sales.user , Point_Of_Sales.Jenis,
Point_Of_Sales.item, Point_Of_Sales.JualGlobal,,
Point_Of_Sales.JualDetail', 'SELECT, INSERT, UPDATE, DELETE,
CREATE, DROP, GRANT, REFERENCES, INDEX, ALTER,
CREATE VIEW, SHOW VIEW');
5. Insert into tables_priv (User,Table_name,Table_priv)
values('kasir','Point_Of_Sales.jualDetail','select,insert,show view');

6. `Insert into tables_priv (User,Table_name,Table_priv)
values('Cewek','Point_Of_Sales.jenis, Point_Of_Sales.item,
Point_Of_Sales.jualGlobal, Point_Of_Sales.jualDetail',
'select,insert,update,show view');`
7. `update tables_priv set host='localhost' where user='Admin';`
8. `update tables_priv set host='192.168.10.2' where user='Cewek';`
9. `update tables_priv set grantor ='root@localhost' where user='Admin';`
10. `delete from tables_priv where user='cewek';`

c. GRANT

Grant berfungsi untuk membuat user baru dan memberikan hak istimewa. Grant adalah salah satu privilege untuk tabel. Grant digunakan untuk memberikan privilege kepada tabel yang didefinisikan kepada pemakai lain. Privilege untuk pemakai dalam perintah grant didefinisikan dengan menggunakan nama-nama privilege. Nama privilege memudahkan administrator untuk dapat memberikan privilege tanpa harus tahu apa nama field dan tabel yang harus diisi.

Perintah grant secara otomatis akan menambah data pemakai apabila data nama pemakai yang disertakan pada perintah tersebut belum ada dalam tabel user. Perintah grant memudahkan administrator untuk tidak perlu melakukan perintah pendefinisian privilege dengan menggunakan sql. Karena dengan menggunakan sql, kita harus hafal nama tabel yang harus diisi, field apa saja yang harus diisi, jumlah field yang harus diisi. Kesalahan mudah dilakukan dengan menggunakan perintah sql karena

ketidaktelitian atau ketidakhafalan nama tabel dan nama field yang harus diisi.

- **Sintak Umum :**

- ✓ GRANT *hak_akses* ON *nama_tabel* TO *pemakai*;
- ✓ GRANT ALL PRIVILEGES ON *database_name.** TO 'myuser' IDENTIFIED BY 'mypassword';

- **Contoh Penggunaan :**

1. GRANT SELECT ON Point_Of_Sales.jenis TO Febe;
2. GRANT SELECT ON Point_Of_Sales.jenis TO Winda;
3. GRANT SELECT ON Point_Of_Sales.item TO Elfrida;
4. GRANT ALL PRIVILEGES ON Point_Of_Sales.User TO Admin;
5. GRANT ALL ON Point_Of_Sales.jualDetail TO Admin
6. SHOW GRANTS FOR root@localhost;
7. SHOW GRANTS FOR Admin;
8. GRANT SELECT,INSERT ON Point_Of_Sales.jualDetail TO kasir;
9. GRANT SELECT(Kode,Nama) ON Point_Of_Sales.jenis TO Elfrida;
10. GRANT UPDATE(kodeItem,NmlItem,kategori,Harga) ON Point_Of_Sales.item TO Elfrida;

d. REVOKE

Untuk menghapus batasan hak akses yang telah diatur dengan menggunakan perintah GRANT, digunakan perintah **REVOKE**.

- **Sintak umum :**

- ✓ `REVOKE hak_akses ON nama_tabel FROM
namaAccount@namaHost;`

Atau menghapus batasan hak akses untuk database & tabel :

- ✓ `REVOKE hak_akses ON nama_database.nama_tabel FROM user;`

Atau menghapus batasan hak akses untuk kolom tertentu :

- ✓ `REVOKE hak_akses(field1,field2, field3,...) ON
nama_database.nama_tabel FROM user;`

Penulisan perintah REVOKE :

- ✓ **HakAkses(field)** : kita harus memberikan sedikitnya satu hak akses. Untuk setiap hak akses yang diberikan, dapat juga diberikan daftar field yang diletakkan dalam kurung, dan dipisahkan dengan tanda koma. Contoh : `REVOKE select (nim, nama), update, insert(nim), ...`

- ✓ **NamaTabel** : merupakan nama tabel yang dikenal hak akses tersebut. Harus ada sedikitnya satu nama tabel. Dapat menggunakan simbol asterik (*) untuk mewakili semua tabel pada database aktif. Penulisan namaTabel dapat juga diikuti oleh nama database diikuti nama tabel yang dipisahkan dengan tanda titik. Menggunakan simbol *.* berarti semua database dan semua tabel yang dikenai hak akses tersebut.

- ✓ **namaAccount@namaHost** : jika nama account tidak ada, tidak pernah diberikan hak akses dengan perintah GRANT sebelumnya maka akan terjadi error.

- **Contoh Penggunaan :**

1. REVOKE SELECT ON Point_Of_Sales.jenis FROM Febe;
2. REVOKE SELECT ON Point_Of_Sales.jenis FROM Winda;
3. REVOKE SELECT ON Point_Of_Sales.item FROM Elfrida;
4. REVOKE ALL PRIVILEGES ON Point_Of_Sales.user FROM Admin;
5. REVOKE ALL ON Point_Of_Sales.jualDetail FROM Admin;
6. REVOKE SELECT,INSERT ON Point_Of_Sales.jualDetail FROM kasir;
7. REVOKE SELECT(Kode,Nama) ON Point_Of_Sales.jenis FROM Elfrida;
8. REVOKE UPDATE(kodeItem,NmlItem,kategori,Harga) ON Point_Of_Sales.item FROM Elfrida;
9. REVOKE INSERT ON Point_Of_Sales.jenis FROM Winda;
10. REVOKE ALL ON Point_Of_Sales.item FROM PUBLIC;

e. SHOW PROCESSLIST

Digunakan untuk menampilkan kegiatan apa saja yang terjadi pada MySQL server atau menampilkan informasi mengenai thread yang dieksekusi di server. Bila terdapat kegiatan yang membahayakan kita sebagai admin dapat menghentikan dengan perintah KILL atau MySQLAdmin

- Sintak umum :

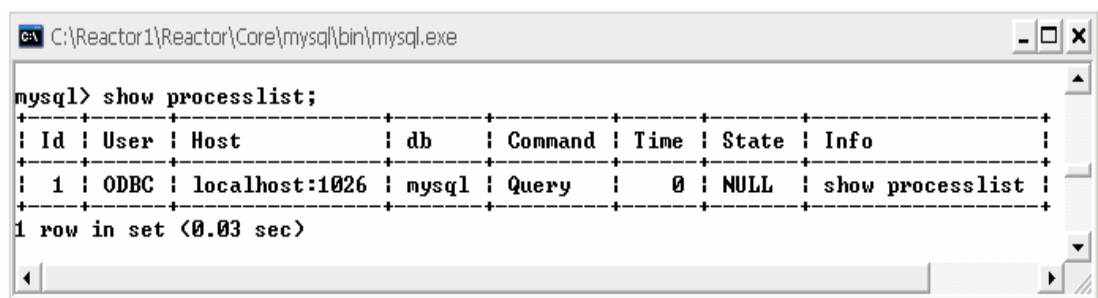
SHOW PROCESSLIST;

Id	User	Host	db	command	time	state	Info
1	ODBC	127.0.0.1:1030	Null	Sleep	6		Null
2	ODBC	127.0.0.1:1031	Pendaftaran	Query	0	Null	Show Processlist

2 rows in set (0.00 set)

- Contoh penggunaan :

SHOW PROCESSLIST;



```
C:\Reactor1\Reactor\Core\mysql\bin\mysql.exe

mysql> show processlist;
+----+-----+-----+-----+-----+-----+-----+-----+
| Id | User | Host           | db    | Command | Time | State | Info          |
+----+-----+-----+-----+-----+-----+-----+-----+
| 1  | ODBC | localhost:1026 | mysql | Query   | 0    | NULL  | show processlist |
+----+-----+-----+-----+-----+-----+-----+-----+
1 row in set (0.03 sec)
```

f. KILL

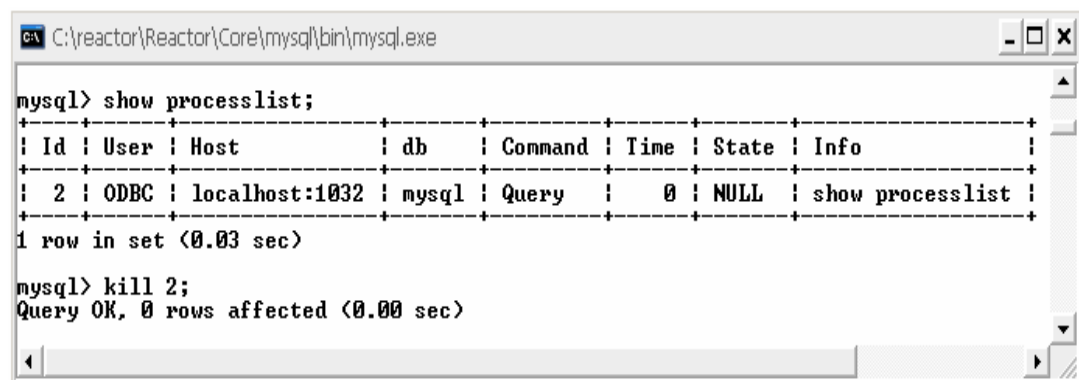
Kill berfungsi menghentikan thread server / untuk membunuh proses yang sedang berjalan

- **Sintak Umum :**

KILL nomor_Id;

- **Contoh penggunaan :**

Kill 2;



```
C:\reactor\Reactor\Core\mysql\bin\mysql.exe

mysql> show processlist;
+-----+-----+-----+-----+-----+-----+-----+-----+
| Id | User | Host          | db   | Command | Time | State | Info          |
+-----+-----+-----+-----+-----+-----+-----+-----+
| 2  | ODBC | localhost:1032 | mysql | Query   | 0    | NULL  | show processlist |
+-----+-----+-----+-----+-----+-----+-----+-----+
1 row in set (0.03 sec)

mysql> kill 2;
Query OK, 0 rows affected (0.00 sec)
```

g. OPTIMIZE TABLE

- ✓ Tabel yang sering mengalami proses penghapusan dan penambahan akan menyebabkan struktur yang tidak teratur secara fisik atau telah terjadi fragmentasi.
- ✓ Penghapusan data dalam jumlah besar mempunyai peluang terjadinya fragmentasi.
- ✓ Terutama untuk data bertipe VARCHAR, TEXT atau BLOB

- ✓ Tidak semua DBMS dapat melakukan fragmentasi, kita dapat melihat dukungan setiap DBMS
- ✓ Untuk mengatasi masalah fragmentasi solusinya adalah melakukan OPTIMIZE TABLE
- ✓ Untuk MySQL versi 3.23 keatas mendukung fasilitas OPTIMIZE TABLE.
- ✓ Perlu diketahui pada saat OPTIMIZE TABLE dikerjakan, semua tabel akan di-lock (terkunci)
- ✓ Proses fragmentasi sebaiknya dilakukan secara berkala, misalnya setiap minggu atau setiap bulan.

▪ **Sintak umum :**

OPTIMIZE TABLE *tabel_1, tabel_2, tabel_3, tabel_n;*

Table	Op	Msg_Type	Mst_Text
STMIK.Mhs	Optimize	Status	OK
STMIK.Absen	Optimize	Status	OK
STMIK.Nilai	Optimize	Status	Table is already up to date

6 rows in set (1.37 sec)

• **Contoh penggunaan :**

OPTIMIZE TABLE

**Point_Of_Sales.User,Point_Of_Sales.item,
Point_Of_Sales.jualGlobal;**

MODEL DATA RELASIONAL

5

Obyektif :

13. Mengetahui bahasa-bahasa yang digunakan pada model data relasional

- **Pengertian Basis Data Relasional**

Pada model relasional, basis data akan “disebar” atau dipilah-pilah ke dalam berbagai tabel dua dimensi. Setiap tabel selalu terdiri atas lajur mendatar yang disebut *baris data (row / record)* dan lajur vertikal yang biasa disebut dengan *kolom (column / field)*. Basis Data Relasional menggunakan tabel dua dimensi yang terdiri atas baris dan kolom untuk memberi gambaran sebuah berkas data.

Contoh Tabel dan keterhubungannya :

MHS

NPM	Nama	Alamat
10296832	Nurhayati	Jakarta
10296126	Astuti	Jakarta
31296500	Budi	Depok
41296525	Prananingrum	Bogor
50096487	Pipit	Bekasi
21196353	Quraish	Bogor

MKUL

KDMK	MTKULIAH	SKS
KK021	P. Basis Data	2
KD132	SIM	3
KU122	Pancasila	2

NILAI

NPM	KDMK	MID	FINAL
10296832	KK021	60	75
10296126	KD132	70	90
31296500	KK021	55	40
41296525	KU122	90	80
21196353	KU122	75	75
50095487	KD132	80	0
10296832	KD132	40	30

- **Keuntungan Basis Data Relasional**

1. Bentuknya sederhana
2. Mudah melakukan berbagai operasi data

- **Istilah dalam Basis Data Relasional :**

Relasi

Relasi merupakan sebuah tabel yang terdiri dari beberapa kolom dan beberapa baris. Relasi menunjukkan adanya hubungan diantara sejumlah entitas yang berasal dari himpunan entitas yang berbeda. Entitas merupakan individu yang mewakili sesuatu yang nyata dan dapat dibedakan dengan yang lainnya.

Atribut

Atribut merupakan kolom pada sebuah relasi. Setiap entitas pasti memiliki atribut yang mendeskripsikan karakter dari entitas tersebut. Penentuan atau pemilihan atribut-atribut yang relevan bagi sebuah entitas merupakan hal penting dalam pembentukan model data.

Tuple

Tuple merupakan baris pada sebuah relasi atau kumpulan elemen-elemen yang saling berkaitan menginformasikan tentang suatu entitas secara lengkap. Satu record mewakili satu data atau informasi tentang seseorang, misalnya : NPM, nama mahasiswa, alamat, kota, dll.

Domain :

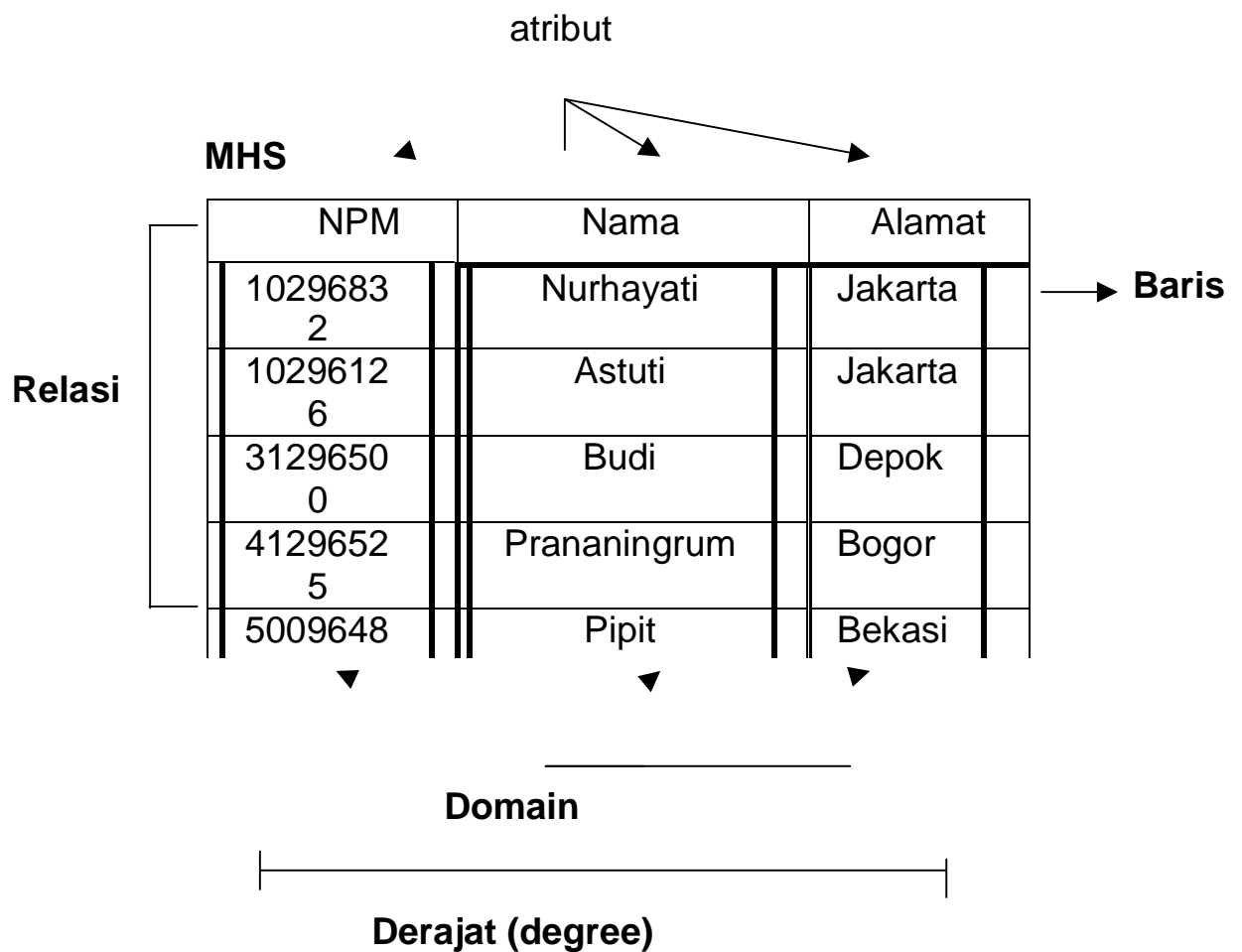
Kumpulan nilai yang valid untuk satu atau lebih atribut

Derajat (degree) :

Jumlah atribut dalam sebuah relasi

Cardinality :

Jumlah tupel dalam sebuah relasi



- **Relational Key**

Super key

Satu atribut / kumpulan atribut yang secara unik mengidentifikasi sebuah tuple di dalam relasi

Candidate key

Suatu atribut atau satu set minimal atribut yang mengidentifikasikan secara unik suatu kejadian spesifik dari entitas. Atribut di dalam relasi yang biasanya mempunyai nilai unik. Satu set minimal dari atribut menyatakan secara tak langsung dimana kita tidak dapat membuang beberapa atribut dalam set tanpa merusak kepemilikan yang unik.

Primary key

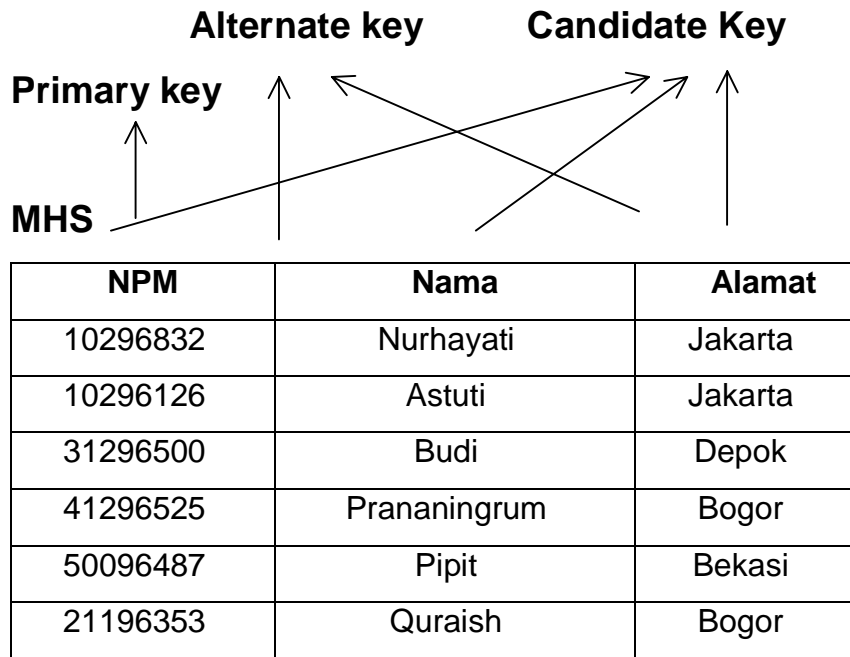
Merupakan satu atribut atau satu set minimal atribut yang tidak hanya mengidentifikasikan secara unik suatu kejadian spesifik, tapi juga dapat mewakili setiap kejadian dari suatu entitas. *Candidate key* yang dipilih untuk mengidentifikasikan tuple secara unik dalam relasi. Setiap kunci *candidate key* punya peluang menjadi *primary key*, tetapi sebaiknya dipilih satu saja yang dapat mewakili secara menyeluruh terhadap entitas yang ada.

Alternate key

Merupakan *candidate key* yang tidak dipakai sebagai *primary key* atau *Candidate key* yang tidak dipilih sebagai *primary key*.

Foreign key (Kunci Tamu)

Atribut dengan domain yang sama yang menjadi kunci utama pada sebuah relasi tetapi pada relasi lain atribut tersebut hanya sebagai atribut biasa. Kunci tamu ditempatkan pada entitas anak dan sama dengan *primary key* induk direlasikan.



- **Relational Integrity Rules**

1. Null

Nilai suatu atribut yang tidak diketahui dan tidak cocok untuk baris (tuple) tersebut.

Nilai (konstanta) Null digunakan untuk menyatakan / mengisi atribut-atribut yang nilainya memang belum siap/tidak ada.

2. Entity Integrity

Tidak ada satu komponen primary key yang bernilai null.

3. Referential Integrity

Suatu domain dapat dipakai sebagai kunci primer bila merupakan atribut tunggal pada domain yang bersangkutan.

- **Bahasa Pada Basis data Relational**

Menggunakan bahasa query ① pernyataan yang diajukan untuk mengambil informasi. Bahasa Query (*Query Language*) lebih ditekankan pada aspek pencarian data dari dalam tabel. Aspek pencarian ini sedemikian penting karena merupakan inti dari upaya untuk pengelolaan data.

Bahasa query terbagi 2 :

1. Bahasa Formal

Bahasa query yang diterjemahkan dengan menggunakan simbol-simbol matematis.

Bahasa query rasional formal merupakan bahasa untuk meminta informasi dari sebuah database/basisdata tanpa harus menghiraukan kerumitan algoritma pengambilannya (sebagaimana sering dijumpai dalam bahasa pemrograman konvensional).

Contoh penggunaan Bahasa query relasional formal yang menggunakan basis data yaitu seperti SQL dikonversi menjadi bahasa relasional formal sehingga didapatkan sekumpulan informasi untuk memperoleh query paling efisien.

Contoh :

- **Bahasa query prosedural** □ pemakai menspesifikasikan data apa yang dibutuhkan dan bagaimana untuk mendapatkannya.

Contohnya : Aljabar Relasional

Aljabar Relasional

Aljabar relasional adalah sebuah bahasa query prosedural yang terdiri dari sekumpulan operasi dimana masukannya adalah satu atau dua relasi dan keluarannya adalah sebuah relasi baru sebagai hasil dari operasi tersebut.

Terdapat lima operasi dasar dalam aljabar relasional, yaitu:

1. Selection (σ)

Operasi selection berfungsi untuk menyeleksi tuple-tuple yang memenuhi predikat yang diberikan dari sebuah tabel relasi. Simbol sigma " σ " digunakan untuk menunjukkan operasi select. Predikat muncul sebagai subscript dari σ dan kondisi yang diinginkan yang ditulis dalam predikat. Argumen diberikan dalam tanda kurung yang mengikuti σ dan berisi tabel relasi yang dimaksud.

2. Projection (π)

Operasi projection berfungsi untuk memilih nilai atribut-atribut tertentu saja dari sebuah tabel relasi. Simbol phi " π " digunakan untuk menunjukkan operasi projection. Predikat muncul sebagai subscript dari π dan hanya nama atribut yang diinginkan yang ditulis dalam predikat. Argumen

diberikan dalam tanda kurung yang mengikuti π dan berisi tabel relasi yang dimaksud.

3. Cartesian – product (\times , juga disebut sebagai cross product)

Operasi cartesian product berfungsi untuk mengkombinasikan informasi yang ada dalam 2 tabel relasi dan menghasilkan sebuah tabel relasi yang baru. Simbol " \times " digunakan untuk menunjukkan operasi cartesian product.

4. Union (\cup)

Operasi union berfungsi untuk mendapatkan gabungan nilai atribut dari sebuah tabel relasi dengan nilai atribut dari tabel relasi lainnya. Simbol " \cup " digunakan untuk menunjukkan operasi union. Operasi union bernilai benar bila terpenuhi 2 kondisi, yaitu : Derajat dari 2 tabel relasi yang dioperasikan harus sama dan domain dari atribut yang dioperasikan juga harus sama.

5. Set – difference ($-$)

Operasi *set difference* berfungsi untuk mendapatkan nilai yang ada dalam sebuah tabel relasi, tapi tidak ada dalam tabel relasi lainnya. Simbol " $-$ " digunakan untuk menunjukkan operasi *set difference*.

6. Rename (ρ)

Dalam operasi himpunan Cross – Product, bisa menimbulkan terjadinya **Konflik Penamaan**, karena Cross – Product bisa menghasilkan suatu relasi dari 2 relasi dengan skema yang sama, sehingga skema hasil akan muncul field dengan nama yang sama. Operator Renaming (ρ) digunakan untuk menghindari terjadinya Konflik Penamaan tersebut.

Operasi – operasi turunan dari operasi – operasi dasar tersebut adalah:

1. Set intersection (\cap)

Set intersection / Intersection (\cap) termasuk ke dalam operator tambahan, karena operator ini dapat diderivikasi dari operator dasar seperti berikut:

$$A \cap B = A - (A - B), \text{ atau } A \cap B = B - (B - A)$$

Operasi *set intersection* berfungsi untuk mendapatkan nilai yang ada dalam sebuah tabel relasi dan juga ada dalam tabel relasi lainnya. Simbol " \cap " digunakan untuk menunjukkan operasi *set intersection*.

2. Theta join (θ)

Operasi *theta join* berfungsi untuk mengkombinasikan tupel dari 2 tabel relasi dimana kondisi dari kombinasi tersebut tidak hanya nilai dari 2 atribut bernama sama, tetapi kondisi yang diinginkan juga bisa menggunakan operator relasional (\leq , $<$, $=$, $>$, \geq). Operasi *theta join* merupakan ekstensi dari natural join.

3. Natural-join ()

Operasi *natural join* berfungsi untuk menggabungkan operasi *selection* dan *cartesian product* menjadi hanya 1 operasi saja. Simbol " \bowtie " digunakan untuk menunjukkan operasi *natural join*. Operasi *natural join* hanya menghasilkan tupel yang mempunyai nilai yang sama pada 2 atribut yang bernama sama pada 2 tabel relasi yang berbeda.

4. Outer-join ()

5. Division (\div)

Operasi yang digunakan dalam query yang mencangkup frase "setiap" atau "untuk semua", operasi ini juga merupakan pembagian atas tuple-tuple dari dua relasi.

- **Bahasa query non-prosedural** @ pemakai menspesifikasikan data apa yang dibutuhkan tanpa menspesifikasikan bagaimana untuk mendapatkannya.

Contohnya : Kalkulus Relasional

Kalkulus Relasional

Dalam kalkulus relasional tidak ada penjabaran bagaimana mengevaluasi querinya, hanya menspesifikkan apa yang harus ditampilkan bukan bagaimana menampilkan. Memungkinkan user menggambarkan apa yang mereka inginkan, tidak pada pada bagaimana cara melakukan komputasi terhadap apa yang mereka inginkan tersebut. (tidak bersifat operasional, tapi

bersifat deklaratif). Memahami aljabar dan kalkulus relasional adalah kunci memahami SQL.

Terbagi 2 :

1. Kalkulus Relasional Tupel

Kalkulus relasional tupel adalah bahasa query yang non prosedural. Bahasa ini mendeskripsikan informasi yang diinginkan tanpa memberi prosedurnya secara detil untuk mendapatkan informasi tersebut. Konsep dasar kalkulus relasional tupel adalah konsep variable tupel. Variable ini merepresentasikan tupel – tupel pada relasi dan digunakan untuk mengekstrak data dari relasi.

Bentuk umum dari kalkulus relasional tuple adalah:

TupleVariabel1 operator[TupleVariabel2 | constant]

Kalkulus relasional tupel merupakan basis untuk bahasa query QUEL.

Sintaks:

$\{ t \mid P(t) \}$ artinya, semua tuple t sedemikian sehingga predikat P adalah benar untuk t .

t : tuple variables

$P(t)$: formula

2. Kalkulus Relasional Domain

Kalkulus relasional domain juga adalah bahasa query yang non prosedural dan karenanya berhubungan dekat dengan kalkulus relasional tupel. Berbeda dengan kalkulus relasional tupel, bahasa ini menggunakan variabel domain yang mengambil nilai dari domain atribut, bukan dari nilai seluruh tupel. Kalkulus relasional domain merupakan basis untuk bahasa query QBE.

Dalam kalkulus relasional ada 2 notasi yang penting.

1. “terdapat beberapa (there exists)” yang ditulis :

$\exists t \in r (Q(t))$, artinya, terdapat beberapa tuple t anggota relasi r sedemikian sehingga

bahwa predikat $Q(t)$ adalah benar.

2. “untuk seluruh (for all)” yang ditulis :

$\forall t \in r(Q(t))$, artinya, untuk seluruh tupel t anggota relasi r sedemikian sehingga bahwa predikat $Q(t)$ adalah benar.

Ciri-ciri relasi kalkulus :

– First order calculus menggunakan simbol-simbol predikat dan simbol-simbol fungsi.

Untuk kaitannya dengan basis data : simbol fungsi tidak diperlukan dan predikat diinterpretasikan sebagai relasi.

– Formula pada first order calculus dapat dibedakan ke dalam dua kelas :

1. Open formula (free variable)

Didefinisikan sebagai himpunan tuples elemen dari kondisi secara keseluruhan, yang dapat menghasilkan formula “TRUE”.

2. Closed Formula atau Sentences yang memiliki variable terbatas.

- Karena kalkulus dipergunakan sebagai bahasa query dan basis data bertujuan untuk instant maupun relasi lainnya, maka closed formulas tidak diperhatikan.

- Dalam kalkulus relasional tupel digunakan variabel dari tupelnya.

- Variabel dari suatu tupel adalah daerah yang terdefinisi sebagai nama dari suatu relasi.

Simbol yang muncul pada formula terdiri dari :

❑ Konstan (elemen-elemen domain D)

❑ Variabel (elemen-elemen dari himpunan berhingga V yang dihubungkan dengan domain D),

❑ Nama relasi (tabel) dan atribut (berdasarkan skema basis data),

❑ Operator perbandingan ($=, \neq, >, \geq, <, \leq$),

❑ Penghubung logika (\wedge (dan / konjungsi)

❑ (\vee atau/disjungsi), \neg (not/negasi), ada/beberapa (\exists), dan semua (\forall).

2. Bahasa Komersial

Bahasa Query yang dirancang sendiri oleh programmer menjadi suatu program aplikasi agar pemakai lebih mudah menggunakannya (user friendly). Contoh :

- QUEL

 Berbasis pada bahasa kalkulus relasional

- QBE

 Berbasis pada bahasa kalkulus relasional

- SQL

 Berbasis pada bahasa kalkulus relasional dan aljabar relasional

- **Contoh-contoh Basis Data Relasional :**

- DB2 ® IBM
- ORACLE ® Oracle
- SYBASE ® Powersoft
- INFORMIX ® Informix
- Microsoft Access ® Microsoft

NORMALISASI

6

Obyektif :

14. Mengetahui dan memahami perintah Normalisasi
 15. Dapat menggunakan perintah Normalisasi
-

Pengertian

Normalisasi database merupakan suatu pendekatan sistematis untuk meminimalkan redundansi data pada suatu database agar database tersebut dapat bekerja dengan optimal.

Tujuan Normalisasi Database

Tujuan normalisasi database adalah untuk menghilangkan dan mengurangi redundansi data dan tujuan yang kedua adalah memastikan dependensi data (Data berada pada tabel yang tepat).

Jika data dalam database tersebut belum di normalisasi maka akan terjadi 3 kemungkinan yang akan merugikan sistem secara keseluruhan.

1. INSERT Anomali : Situasi dimana tidak memungkinkan memasukkan beberapa jenis data secara langsung di database.
2. DELETE Anomali: Penghapusan data yang tidak sesuai dengan yang diharapkan, artinya data yang harusnya tidak terhapus mungkin ikut terhapus.
3. UPDATE Anomali: Situasi dimana nilai yang diubah menyebabkan inkonsistensi database, dalam artian data yang diubah tidak sesuai dengan yang diperintahkan atau yang diinginkan.

Normalisasi Database

Normalisasi Database terdiri dari banyak bentuk, dalam ilmu basis data ada setidaknya 9 bentuk normalisasi yang ada yaitu 1NF, 2NF, 3NF, EKNF, BCNF, 4NF, 5NF, DKNF, dan 6NF. Namun dalam prakteknya dalam dunia industri bentuk normalisasi ini yang paling sering digunakan ada sekitar 5 bentuk.

Tahap Normalisasi Database

1. Unnormalized Form (UNF)

Merupakan bentuk tidak normal berdasarkan data yang diperoleh dan mengandung kerangkapan data.

2. First Normal Form (1NF)

Entitas yang atributnya memiliki tidak lebih dari satu nilai untuk contoh tunggal entitas tersebut.

3. Second Normal Form (2NF)

Entitas yang atribut non-primary key-nya hanya tergantung pada full primary key.

4. Third Normal Form (3NF)

Entitas yang atribut non-primary key-nya tidak tergantung pada atribut nonprimary key yang lain.

5. Boyce Code Normal Form (BCNF)

Dilakukan remove multivalued dependent. BCNF terjadi jika masih terdapat anomaly pada bentuk 3NF dikarenakan relasi memiliki lebih dari satu candidate key.

6. Fifth Normal Form (5NF)

Tahapan ini dilakukan untuk mengatasi terjadinya join dependent pemecahan relasi menjadi dua sehingga relasi tersebut tidak dapat digabungkan kembali menjadi satu.

Unnormalized Form (UNF)

Bentuk ini memiliki ciri-ciri, yaitu merupakan kumpulan data yang akan direkam, dapat saja data tidak lengkap atau terduplikasi, dan data dikumpulkan apa adanya sesuai dengan kedatangannya.

Contoh Normal Form

" Honda Jaya Raya" AHASS 06488 Jatimulya - Bekasi Timur Telp. 021-82432162			No Faktur : 05103214 Tanggal : 25/10/2005		
BON PEMBELIAN					
No Polisi : B3117LB, Warna : Biru Merek : Supra X, Tahun : 2005 Mekanik ID : DDE, Nama : Djoko Dewanto					
Kode Parts	Nama Parts	Kuantum	Harga (*)	Discount	Jumlah Rp.
20W501000CC	Oli Top 1 1000cc	2	27,000	1,000	52,000
SERV001	Engine Tune Up	1	25,000	2,000	23,000
				Potongan	2,000
				Total Rp.	73,000

(*) Harga tersebut sudah termasuk PPN

Lembar ke-1 : Pelanggan

Lembar ke-2 : Accounting

Normalisasi Database Form (Rudiawan16)

1. First Normal Form (1NF)

Bentuk normal yang pertama atau 1NF mensyaratkan beberapa kondisi dalam sebuah database, berikut adalah fungsi dari bentuk normal pertama ini.

- Menghilangkan duplikasi kolom dari tabel yang sama.
- Buat tabel terpisah untuk masing-masing kelompok data terkait dan mengidentifikasi setiap baris dengan kolom yang unik (primary key).

Syarat bentuk normal pertama (1NF):

- Tidak ada set atribut yang berulang atau bernilai ganda, setiap atribut yang dimilikinya bersifat atomic (bernilai tunggal) untuk setiap baris.
- Telah ditentukannya primary key untuk tabel atau relasi.
- Tiap atribut hanya memiliki satu pengertian.
- Tiap atribut yang dapat memiliki banyak nilai sebenarnya menggambarkan entitas atau relasi yang terpisah.

Contoh Normalisasi Database 1NF

BON PEMBELIAN

No Faktur	Tanggal	No Polisi	Warna	Merek	Tahun	Mekanik ID	Nama Mekanik	Kode Parts	
05103214	25/10/2005	B3117LB	Biru	Supra X	2005	DDE	Dioko Dewanto	20W501000CC	..
05103214	25/10/2005	B3117LB	Biru	Supra X	2005	DDE	Dioko Dewanto	SERV001	..
05103215	25/10/2005	B2121AA	Merah	Supra X	2005	DDE	Dioko Dewanto	SERV001	..

Nama Parts	Kuantum	Harga	Discount	Jumlah	Potongan	Total
Oil Top 1 000cc	2	27000	1000	52000	2000	73000
Engine Tune Up	1	25000	2000	23000	2000	73000
Engine Tune Up	1	25000	2000	23000	0	23000

Normalisasi Database 1NF (Rudiawan16)

dari manual bon pembelian diatas kita dapat menjadi bentuk normal pertama dengan memisah-misahkan data pada atribut-atribut yang tepat dan bernilai atomik, juga seluruh record / baris harus lengkap adanya.

2. Second Normal form (2NF)

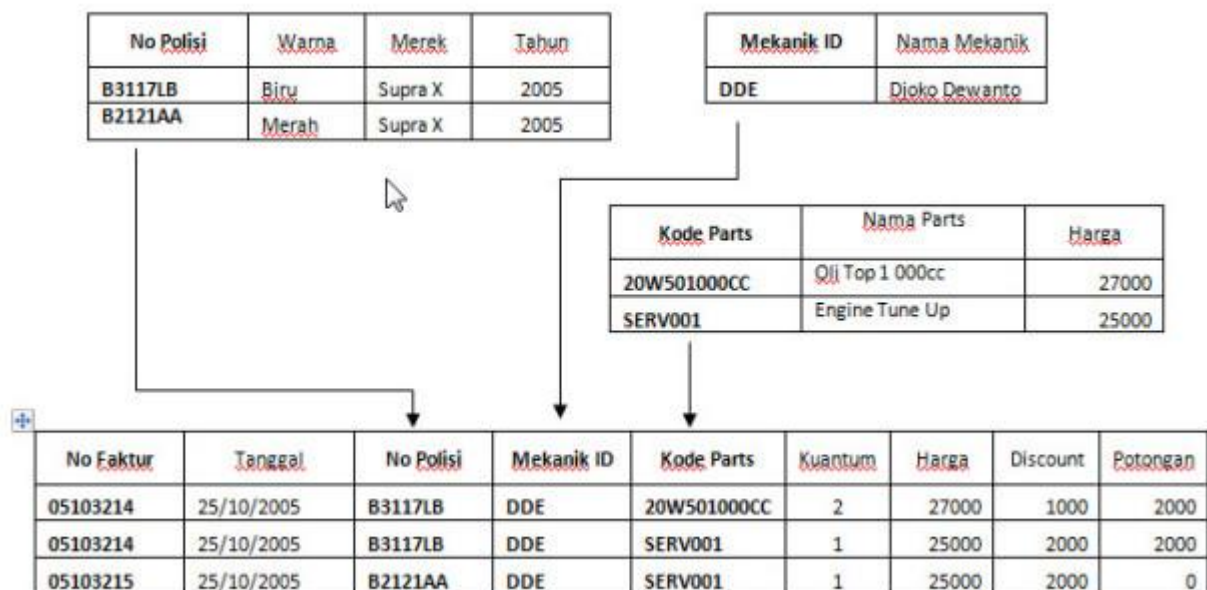
Syarat untuk menerapkan normalisasi bentuk kedua ini adalah data telah dibentuk dalam 1NF, berikut adalah beberapa fungsi normalisasi 2NF.

- Menghapus beberapa subset data yang ada pada tabel dan menempatkan mereka pada tabel terpisah.
- Menciptakan hubungan antara tabel baru dan tabel lama dengan menciptakan foreign key.
- Tidak ada atribut dalam tabel yang secara fungsional bergantung pada candidate key tabel tersebut.

Syarat dari bentuk normal kedua (2NF):

- Bentuk data telah memenuhi kriteria bentork normal ke satu.
- Atribut bukan kunci (non - key atribut) haruslah memiliki ketergantungan fungsional sepenuhnya pada primary key.
- Kunci primer hanya mengandung satu atribut.

Contoh normalisasi database bentuk 2NF



Normalisasi Database 2NF (Rudiawan16)

Bentuk normal kedua dengan melakukan dekomposisi tabel diatas menjadi beberapa tabel dan mencari kunci primer dari tiap-tiap tabel tersebut dan atribut kunci haruslah unik.

3. Third Normal Form (3NF)

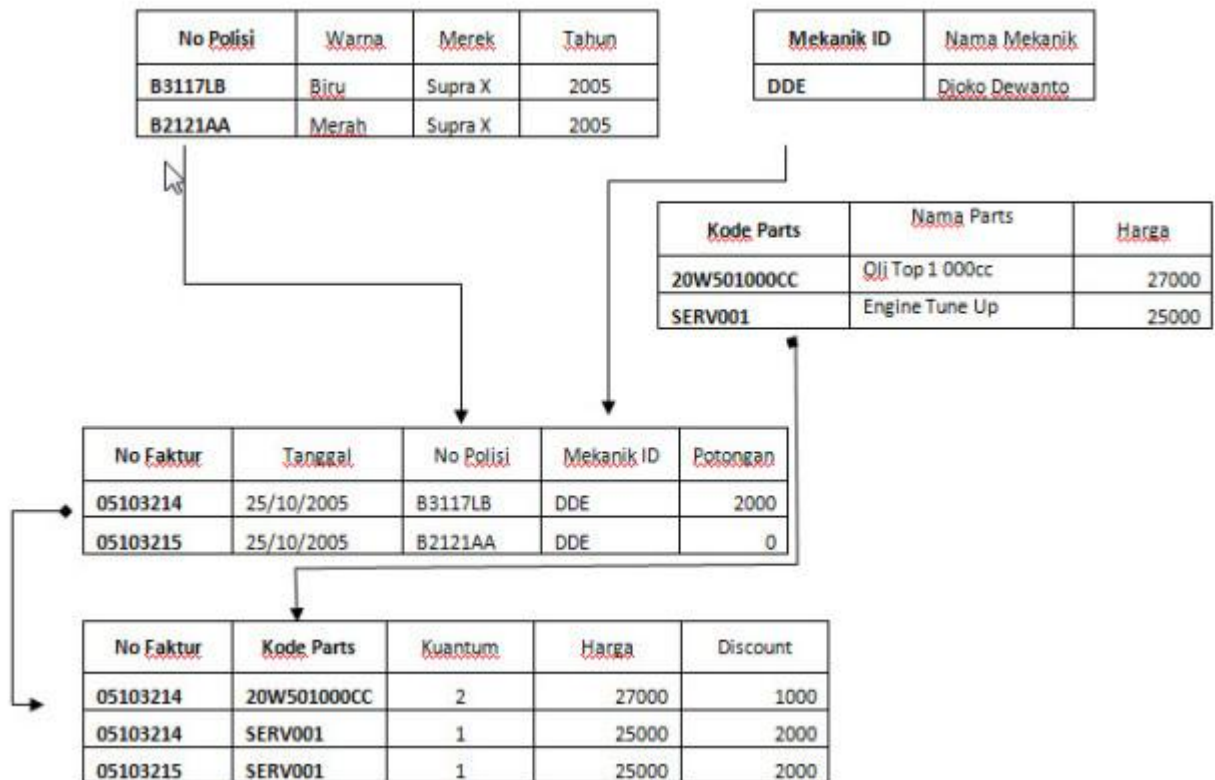
Normalisasi database dalam bentuk 3NF bertujuan untuk menghilangkan seluruh atribut atau field yang tidak berhubungan dengan primary key. Dengan demikian tidak ada ketergantungan transitif pada setiap kandidat key.

Syarat dari bentuk normal ketiga atau 3NF adalah :

- Memenuhi semua persyaratan dari bentuk normal kedua.
- Menghapus kolom yang tidak tergantung pada primary key.

Contoh Normalisasi Database Bentuk 3NF

Bentuk normal ketiga mempunyai syarat, setiap relasi tidak mempunyai atribut yang bergantung transitif, harus bergantung penuh pada kunci utama dan harus memenuhi bentuk normal kedua (2 NF).



Normalisasi Database 3NF (Rudiawan16)

4. BCNF Boyce–Codd normal form

Merupakan sebuah teknik normalisasi database yang sering disebut 3.5NF, memiliki hubungan yang sangat erat dengan bentuk 3NF. Pada dasarnya adalah untuk menghandle anomali dan overlooping yang tidak dapat di handle dalam bentuk 3NF. Normalisasi database bentuk ini tergantung dari kasus yang disediakan, tidak semua tabel wajib di normalisasi dalam bentuk BCNF.

Syarat dari bentuk normal BCNF adalah:

- Semua anomali (kesalahan data) yang tersisa dari hasil penyempurnaan kebergantungan fungsional telah dihilangkan.

5. Fourth Normal Form (4NF)

Syarat bentuk normal keempat (4NF) adalah:

- Bila dan hanya bila telah berada dalam bentuk BCNF dan tidak ada multivalued dependency nontrivial.
- Multivalued Dependency Nontrivial (MVD) dipakai dalam 4NF.
- Dependency ini dipakai untuk menyatakan hubungan satu (one to many).

6.Fifth Normal Form (5NF)

Syarat bentuk normal kelima (5NF):

- Semua anomali (kesalahan data) yang tertinggal telah dihilangkan.

Membuat database **Pembelian**

```
create database pembelian
use pembelian
```

Membuat table

```
create table kendaraan (
no_polisi char(10),
warna char(20),
merek char(30),
tahun char(5),
primary key (no_polisi)
)
sp_help kendaraan
```

```
create table mekanik (  
  
mekanik_id char(5),  
  
nama_mekanik varchar(50),  
  
primary key (mekanik_id)  
  
)
```

```
sp_help mekanik
```

```
create table parts (  
  
kode_parts char(20),  
  
nama_parts varchar(50),  
  
harga int,  
  
primary key (kode_parts)  
  
)
```

```
sp_help parts
```

```
create table bon_pembelian (  
  
no_faktur char(10),  
  
tanggal datetime,  
  
no_polisi char(10),  
  
mekanik_id char(5),  
  
potongan int,
```

```

primary key (no_faktur),

constraint FK_nopolisi foreign key (no_polisi) references kendaraan(no_polisi),

constraint FK_mekanik foreign key (mekanik_id) references mekanik(mekanik_id)

)

sp_help bon_pembelian


create table transaksi_parts (

no_faktur char(10),

kode_parts char(20),

qty int,

harga int,

discount int,

primary key (no_faktur,kode_parts),

constraint FK_nofaktur foreign key (no_faktur) references bon_pembelian(no_faktur),

constraint FK_kodeparts foreign key (kode_parts) references parts(kode_parts)

)

sp_help transaksi_parts

```

Mengisi data pada table

```

/*jawaban no.3 */

/** Isi data table kendaraan **/

INSERT INTO kendaraan VALUES('B3117LB','Biru','Supra X','2005')

```

```

INSERT INTO kendaraan VALUES('B2121AA','Merah','Supra X','2005')

/** isi data table mekanik **/

INSERT INTO mekanik VALUES('DDE','Djoko Dewanto')

/** isi data table parts **/

INSERT INTO parts VALUES('20W501000CC','Oli Top 1 000cc',27000)

INSERT INTO parts VALUES('SERV001','Engine Tune Up',25000)

/** isi data table bon_pembelian **/

INSERT INTO bon_pembelian VALUES('05103214',GETDATE(),'B3117LB','DDE',2000)

INSERT INTO bon_pembelian VALUES('05103215',GETDATE(),'B2121AA','DDE',0)

/** isi data table transaksi_parts **/

INSERT INTO transaksi_parts (no_faktur,kode_parts,qty,harga,discount)

select '05103214','20W501000CC',2,harga,1000 FROM parts where
kode_parts='20W501000CC'

INSERT INTO transaksi_parts (no_faktur,kode_parts,qty,harga,discount)

select '05103214','SERV001',1,harga,2000 FROM parts where kode_parts='SERV001'

INSERT INTO transaksi_parts (no_faktur,kode_parts,qty,harga,discount)

select '05103215','SERV001',1,harga,2000 FROM parts where kode_parts='SERV001'

```

Menampilkan data dari table yang telah kita isi

```

/*jawaban no.3 */

select * from kendaraan

select * from mekanik

```



```
select * from parts
```

```
select * from bon_pembelian
```

```
select * from transaksi_parts
```

```
/** Relasi antar table hingga terbentuk 1NF **/
```

```
select a.no_faktur,  
a.tanggal,a.no_polisi,e.warna,e.merek,e.tahun,a.mekanik_id,d.nama_mekanik,  
  
b.kode_parts,c.nama_parts,b.qty,b.harga,b.discount,(b.qty*b.harga)-(b.qty*b.discount) as  
jumlah,a.potongan,  
  
(select sum((qty*harga)-(qty*discount))-a.potongan from transaksi_parts where  
no_faktur=a.no_faktur) as total  
  
from bon_pembelian as a  
  
join transaksi_parts as b ON a.no_faktur=b.no_faktur  
  
join parts as c ON b.kode_parts=c.kode_parts  
  
join mekanik as d ON a.mekanik_id=d.mekanik_id  
  
join kendaraan as e ON a.no_polisi=e.no_polisi
```

JOIN & SUBQUERY

7

Obyektif :

- 16. Mengetahui dan memahami perintah JOIN
- 17. Dapat menggunakan perintah JOIN
- 18. Mengetahui dan memahami perintah Subquery
- 19. Dapat menggunakan perintah Subquery
- 20. Memproduksi hasil yang dapat digunakan pada seleksi berikutnya.

Perintah **JOIN** digunakan untuk menampilkan suatu output yang berasal dari beberapa tabel (lebih dari satu tabel).

1. Equi Join

Sebuah equi join adalah jenis tertentu dari komparator berbasis join, yang hanya menggunakan perbandingan kesetaraan dalam predikat join. Menggunakan operator perbandingan lainnya (seperti <) mendiskualifikasi bergabung sebagai equi join. Equi join menggunakan tanda equal (=) untuk membandingkan operatornya.

Contoh :

```
SELECT * FROM employee, department WHERE  
employee.DepartmentID = department.DepartmentID;
```

Jika kolom dalam equi-join memiliki nama yang sama, SQL-92 menyediakan notasi singkatan opsional untuk mengekspresikan equi join, dengan cara menggunakan construct

Contoh :

```
SELECT * FROM employee INNER JOIN department USING (DepartmentID);
```

Non Equi Join : Join antar dua tabel memakai tanda selain (=)

Contoh :

```
Select first_name,salary,jobs.jobs_title,jobs.min_salary,jobs.max_salary from employees  
JOIN departments ON employees.salary between jobs.min_salary and jobs.max_salary  
JOIN departments ON employees.salary between jobs.min_salary and jobs.max_salary
```

2. Inner Join

Inner join pada dasarnya adalah menemukan persimpangan (intersection) antara dua buah tabel. Penggunaan relasi INNER JOIN adalah untuk menampilkan kedua tabel yang direlasikan dengan menampilkan record – record yang bersesuaian saja.

Hanya bisa menampilkan data yang sesuai atau memenuhi kondisi dan jika ada data yang tidak punya pasangan di tabel lawannya maka tidak akan muncul. Ada dua cara yang berbeda untuk mengekspresikan sintaksis join: menggunakan “explicit join notation” dan “implicit Join notation”.

Syntax dari INNER JOIN adalah sebagai berikut :

```
table_reference [INNER] JOIN table_factor [join_condition]
```

“eksplisit join notation” menggunakan join keyword untuk menentukan tabel untuk bergabung, dan ON keyword untuk menentukan predikat untuk bergabung, seperti dalam contoh berikut :

```
SELECT * FROM employee INNER JOIN department ON  
employee.DepartmentID = department.DepartmentID;
```

“implicit join notation” hanya berisi daftar tabel untuk bergabung, dalam klausa FROM dari pernyataan SELECT, menggunakan koma untuk memisahkan mereka.

Oleh karena itu menentukan cross join, dan klausa WHERE tambahan mungkin berlaku filter-predikat.

Contoh berikut ini adalah setara dengan yang sebelumnya, tapi kali ini menggunakan implicit join notation :

```
SELECT * FROM employee, department  
WHERE employee.DepartmentID = department.DepartmentID;
```

Query yang diberikan pada contoh di atas akan bergabung dengan tabel Employee dan Department menggunakan kolom DepartmentId dari kedua tabel.

Dimana DepartmentId ini pertandingan tabel (yaitu join-predikat puas), query akan menggabungkan kolom LastName, DepartmentId dan DepartmentName dari dua tabel menjadi baris hasil.

Jika DepartmentId tidak cocok, tidak ada baris hasil yang dihasilkan.

3. Left Outer Join

Menampilkan semua data yang ada di tabel kiri dan hanya data yang bersesuaian di tabel kanan, jika tabel kiri tidak mempunyai lawan di tabel kanan maka tabel kanan akan diisi dengan null.

Contoh :

```
SELECT * FROM employee LEFT OUTER JOIN department ON  
employee.DepartmentID = department.DepartmentID;
```

Oracle mendukung alternatif syntax, yaitu :

```
SELECT * FROM employee, department WHERE  
employee.DepartmentID = department.DepartmentID(+)
```

4. Right Outer Join

Menampilkan semua data yang ada di tabel kanan dan hanya data yang sesuai di tabel kiri, jika tabel kanan tidak mempunyai lawan di tabel kiri maka tabel kiri akan diisi dengan null.

Contoh :

```
SELECT * FROM employee RIGHT OUTER JOIN department ON  
employee.DepartmentID = department.DepartmentID;
```

Oracle mendukung alternatif syntax, yaitu :

```
SELECT * FROM employee, department WHERE  
employee.DepartmentID(+) = department.DepartmentID
```

5. Full Join

Gabungan dari left outer join dan right outer join. Jika pada right join akan membuat sebuah parameter di sebelah kanan dan pada left join akan membuat sebuah parameter dari table sebelah kiri, maka full join membuat sebuah parameter pada kanan dan kiri. Jika ada data atau record yang kosong atau tidak berelasi maka akan berisi NULL di sebelah kanan dan disebelah kiri.

Contoh :

```
Select first_name, department_name From employees FULL OUTER JOIN  
departments ON employees.department_id = departments.department_id;
```

6. Self Join

Join yang dilakukan antar kolom dalam satu tabel.

Contoh :

```
Select pegawai.first_name as pegawai, manager.first_name as manager  
From employees pegawai INNER JOIN employees manager ON  
pegawai.manager_id=manager.employee_id;
```

7. Natural Join

Sebuah natural join adalah jenis equi join dimana predikat bergabung timbul implisit

dengan membandingkan semua kolom di kedua tabel yang memiliki kolom yang mempunyai nama yang sama dalam tabel yang digabung. Tabel bergabung dihasilkan hanya berisi satu kolom untuk setiap pasangan kolom yang mempunyai nama sama.

Contoh di atas untuk inner join dapat dinyatakan sebagai natural join dengan cara berikut :

```
SELECT * FROM employee NATURAL JOIN department;
```

8. Cross Join

Merupakan operasi perkalian, dengan mengkombinasikan setiap baris di tabel sebelah kiri dengan isi tabel sebelah kanan.

Cross join mengembalikan produk cartesian dari baris dari tabel dalam bergabung.

Dengan kata lain, hal itu akan menghasilkan baris yang menggabungkan setiap baris dari tabel pertama dengan setiap baris dari tabel kedua.

Contoh :

```
SELECT * FROM employee CROSS JOIN department;
```

9. STRAIGHT_JOIN

STRAIGHT_JOIN merupakan pengganti keyword JOIN pada MySQL yang digunakan untuk "memaksa" proses join table dari kiri (LEFT) ke kanan (RIGHT).

Contoh Penggunaan :

```
SELECT ms_cabang.*, ms_kota.nama_kota FROM ms_cabang STRAIGHT_JOIN ms_kota ON ms_cabang.kode_kota = ms_kota.kode_kota
```

Union

MySQL Union adalah statemen yang mengkombinasikan dua buah atau lebih resulset dari beberapa table dengan statemen SELECT sehingga menjadi satu buah resulset.

Union Statemen memiliki beberapa ketentuan sebagai berikut :

- Jumlah kolom/field dari setiap statemen SELECT harus sama.
- Tipe data kolom/field dari setiap statemen SELECT harus kompatibel.

Jika kita perlu memperoleh informasi lebih dari satu table, pilihannya adalah :

- menggunakan subquery, atau
- menggunakan join.

Contoh :

TABEL S

Sn	Sname	Status	City
S1	Smith	20	London
S2	Jones	10	Paris
S3	Blake	30	Paris
S4	Clark	20	London

TABEL P

Pn	Pname	Warna	Weight	City
P1	Nut	Red	12	London
P2	Bolt	Green	17	Paris
P3	Screw	Blue	17	Rome
P4	Screw	Red	14	London
P5	Cam	Blue	12	Paris
P6	Cog	Red	19	London

TABEL SP

Sn	Pn	Qty
S1	P1	300
S1	P2	200
S1	P3	400
S1	P4	200
S1	P5	100
S1	P6	100
S2	P1	300
S2	P2	400
S3	P2	200
S4	P2	200
S4	P4	300
S4	P5	400

- **Menampilkan record (SELECT lebih dari satu tabel / JOIN)**

1. Menampilkan semua supplier dan part yang keduanya bertempat tinggal pada kota yang sama

```
SELECT Sn, Sname, S.tatus, S.City , Pn, Pname, Warna, Weight
FROM S,P
WHERE S.City = P.City
```

2. Menampilkan nama supplier yang memasok barang dengan nomor part P2

```
SELECT Sname FROM S, SP
WHERE S.Sn = SP.Sn AND SP.Pn = 'P2'
```

3. Menampilkan nama supplier yang memasok part berwarna merah

```
SELECT Sname FROM S, SP, P
WHERE S.Sn = SP.Sn
      AND SP.Pn = P.Pn
      AND P.COLOR = 'RED'
```

- **Menampilkan record (SELECT lebih dari satu tabel / SELECT Bertingkat)**

1. Menampilkan nama supplier yang memasok barang dengan nomor part P2

```
SELECT Sname FROM S WHERE Sn IN
      (SELECT Sn FROM SP WHERE Pn = 'P2')
```

atau

```
SELECT Sname FROM S WHERE Sn = ANY
      (SELECT Sn FROM SP WHERE Pn = 'P2')
```

2. Menampilkan nama supplier yang memasok part berwarna merah

```
SELECT Sname FROM S WHERE Sn IN
      (SELECT Sn FROM SP WHERE Pn IN
        (SELECT Pn FROM P WHERE Warna = 'Red'))
```

3. Menampilkan no.supplier dengan nilai status lebih kecil daripada nilai maksimum status yang ada pada tabel S

```
SELECT Sn FROM S WHERE Status <
      (SELECT MAX(Status) FROM S)
```


4. Menampilkan nama supplier yang tidak memasok barang dengan nomor part P2

```
SELECT Sname FROM S WHERE Sn NOT IN  
      (SELECT Sn FROM SP WHERE Pn = 'P2')
```

5. Menampilkan semua nomor supplier yang sama lokasinya dengan S1

```
SELECT Sn FROM S WHERE CITY =  
      (SELECT CITY FROM S WHERE Sn = 'S1')
```

Subquery (Subselect) adalah pernyataan SELECT yang merupakan bagian dari pernyataan lain, misal : INSERT. Pernyataan ORDER BY, FOR UPDATE OF, UNION, INTERSECT atau EXCEPT tidak termasuk dalam pernyataan ini.

Subquery menghasilkan sebuah tabel yang merupakan bagian dari tabel atau view yang diidentifikasi pada klausa FROM. Pembagian ini dapat digambarkan seperti urutan operasi, dimana hasil dari suatu operasi adalah input bagi operasi lain.

Subquery diperlukan pada saat hasil query tidak berhasil dilakukan dengan hanya melalui satu tabel saja, juga pada saat hasil suatu query digunakan pada klausa WHERE query lainnya. Hasil yang diperoleh dari SUBSELECT tidak dapat ditampilkan oleh “main” SELECT.

SUBQUERY Adalah subselect yang dapat digunakan di klausa WHERE dan HAVING dipernyataan select luar untuk menghasilkan tabel akhir. Aturan-aturan untuk membuat subquery, yaitu :

1. Klausa Order By tidak boleh digunakan di subquery, Order By hanya dapat digunakan di pernyataan Select luar.
2. Klausa subquery Select harus berisi satu nama kolom tunggal atau ekspresi kecuali untuk subquery-subquery menggunakan kata kunci EXIST
3. Secara default nama kolom di subquery mengacu ke nama tabel di klausa FROM dari subquery tersebut.
4. Saat subquery adalah salah satu dua operan dilibatkan di perbandingan, subquery harus muncul disisi kanan perbandingan.

Kegunaan-kegunaan Subquery dalam memanipulasi data:

- § Meng-copy data dari satu tabel ke tabel lain
- § Menerima data dari inline view
- § Mengambil data dari tabel lain untuk kemudian di update ke tabel yang

dituju

§ Menghapus baris dari satu tabel berdasarkan baris dari tabel lain.

Penggunaan sub query dapat diterapkan pada pernyataan SELECT, UPDATE, DELETE, dan INSERT.

Single Row Subquery

Single row subquery memberikan hasil hanya satu baris pada bagian subquery. Untuk single row subquery ini yang digunakan adalah operator pembandingan: , >, >=, <, <= atau <>

Contoh:

Menampilkan data karyawan yang memiliki jabatan sama dengan Smith.

```
SELECT last_name, title FROM employee WHERE title = ( SELECT title  
FROM employee WHERE last_name = 'Smith' );
```

Multiple Rows Subquery

Multiple Row Subquery adalah subquery yang menghasilkan lebih dari satu baris data. Untuk multiple row subquery ini yang digunakan adalah operator pembandingan IN, ANY atau ALL

Operator EXIST dan NOT EXIST

Kata kunci EXIST dan NOT EXIST dirancang hanya untuk digunakan di subquery. Kata kunci-kata kunci ini menghasilkan nilai TRUE atau FALSE. EXIST akan mengirim nilai TRUE jika dan hanya jika terdapat sedikitnya satu baris di table hasil yang dikirim oleh subquery. EXIST mengirim nilai FALSE jika subquery mengirim table kosong. NOT EXIST kebalikan dan EXIST. Karena EXIST dan NOT EXIST hanya memeriksa keberadaan baris-baris di table hasil subquery.

Penggunaan ANY dan ALL

Jika subquery diawali kata kunci ALL, syarat hanya akan bernilai TRUE jika dipenuhi semua nilai yang dihasilkan subquery itu. Jika subquery diawali kata kunci ANY, syaratnya akan bernilai TRUE jika dipenuhi

sedikitnya satu nilai yang dihasilkan subquery tersebut. Standar ISO mengizinkan digunakan kata kunci SOME untuk menggantikan ANY.

- ANY dan ALL dapat digunakan dengan subqueries yang menghasilkan satu kolom tunggal.
- Jika subquery bernilai kosong (empty), ALL mengembalikan nilai benar (true), dan ANY mengembalikan nilai salah (false).
- SOME dapat digunakan sebagai pengganti ANY.

Ada beberapa bahasan terkait penggunaan Sub Query ini antara lain :

1. Sub query dengan IN

Jika operator '=' hanya digunakan untuk hasil yang tepat satu, maka jika ingin menampilkan yang memiliki hasil lebih dari satu maka menggunakan perintah IN. Dan struktur Query yang digunakan dalam hal ini adalah

SELECT namakolom FROM namatabel WHERE kondisi operatorperbandingan IN (subquery);

2. Sub query dengan ALL

Command ALL diikuti dengan operator perbandingan digunakan memiliki arti menampilkan nilai jika perbandingan bernilai benar untuk semua data. Operator perbandingan tersebut berupa (<, >, =, !=). Berikut adalah query dasar dari sub query all

SELECT namakolom FROM namatabel WHERE kondisi operatorperbandingan ALL (subquery);

3. Sub query dengan ANY

Command ANY diikuti dengan operator perbandingan memiliki arti menampilkan nilai yang sesuai dengan apapun yang dihasilkan oleh sub query. ANY berbeda dengan IN, jika IN itu semua data, sedangkan ANY hanya beberapa data. Contoh query dasar dari sub query ANY

SELECT namakolom FROM namatabel WHERE kondisi operatorperbandingan ANY (subquery);

4. Sub query dengan EXISTS

Perintah EXISTS disini berguna untuk mengatur penampilan hasil query, Query Utama akan dijalankan jika Sub Query bernilai TRUE (ada hasilnya) jika hasilnya kosong maka Query utama tidak akan dijalankan. Lawan dari statement EXISTS adalah NOT EXISTS. Query yang digunakan adalah
SELECT namakolom FROM namatabel WHERE EXIST / NOT EXIST (subquery);

5. Insert Into Statement

Untuk meng-copy data dari suatu tabel ke tabel lain kita bisa menggunakan perintah INSERT INTO. Tetapi sebelum kita menggunakan perintah INSERT INTO ini kita harus membuat tabel baru yang jumlah field dan urutannya sama dengan tabel field sebelumnya, nama dari field tidak harus sama. Query untuk perintah INSERT INTO adalah sebagai berikut :

INSERT INTO namatabel2 SELECT namakolom FROM namatabel1 ;

Urutan operasi pada *Subquery* adalah :

1. klausa FROM
2. klausa WHERE
3. klausa GROUP BY
4. klausa HAVING
5. klausa SELECT

- **Sintaks :**

```
SELECT    ( * | (ekspresi_kolom) ....  
FROM      (nama_tabel) ....  
[WHERE    kondisi ]  
[GROUP BY (nama_kolom) .... ]  
[HAVING   kondisi_having ]
```

- **SUBQUERY – Coding**

Fungsi :

Kita dapat menggunakan subquery tidak hanya dalam klausa WHERE, namun juga klausa HAVING. Pada klausa kondisi (WHERE atau HAVING), akses lain seperti SELECT dapat melibatkan beberapa tabel. Ada beberapa cara untuk menggabungkan SELECT tambahan pada klausa SELECT atau HAVING :

- Perbandingan aritmatik (=, >, <)
- ANY (dikombinasikan dengan =, >=, <=)
- SOME (dikombinasikan dengan =, >=, <=)
- IN

Hasil *Subquery* menentukan key word manakah yang dapat digunakan (ANY, SOME, IN). UNION tidak diperkenankan untuk digunakan dalam SUBSELECT. Sedangkan aritmatik dapat digunakan.

Contoh :

1. *Subquery* dengan Perbandingan Aritmatik

```
SELECT EMPNO, LASTNAME, M_SALARY  
FROM OWNER_ID.EMP  
WHERE  
        M_SALAR  
Y >  
        (SELECT  
        AVG(M_SALARY) FROM  
        OWNER_ID.EMP)
```

Subquery menghasilkan nilai tunggal. Oleh karena itu perbandingan dalam klausa WHERE cukup dilakukan dengan operator aritmatik yang sederhana.

2. *Subquery* dengan IN

```
SELECT EMPNO, FIRSTNME, M_SALARY  
FROM OWNER_ID.EMP  
WHERE    M_SALARY > 2500 AND JOBID IN  
        (SELECT JOBID FROM  
        OWNER_ID.JOB  
        WHERE JOB_NAME LIKE 'SYSTEM%')
```

Subquery ini menghasilkan sekumpulan nilai tetapi hasilnya masih dalam satu kolom. Key word khusus dibutuhkan untuk menggabungkan nilai-nilai tersebut dalam “main” SELECT. =ANY atau =SOME dapat digunakan sebagai pengganti IN.