

Sistem Manajemen Basis Data

Pendahuluan

Apakah basis data itu? Secara umum basis data dapat didefinisikan sebagai sekumpulan data yang berelasi, misalnya kumpulan data yg berisi tentang data film yang ada di dalam usaha rental film, sekumpulan data pegawai dalam suatu instansi, dan lain sebagainya. Berikut merupakan beberapa definisi yang intinya adalah sama dengan definisi di atas :

- Kumpulan informasi yang bermanfaat yang diorganisasikan ke dalam tatacara khusus (George Tsu-der Chou).
- Sistem berkas terpadu yang dirancang terutama untuk meminimalkan pengulangan data (Anthoni J. Fabbri & A. Robert Schwab).
- Sistem terkomputerisasi yang tujuan utamanya adalah memelihara informasi dan membuat informasi itu ada pada saat dibutuhkan (C.J. Date).
- Kumpulan data yang saling berelasi dan program yang memungkinkan user untuk mengakses dan memodifikasi data tersebut (Silberschatz dkk).

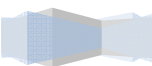
Fungsi dari sistem basis data adalah :

1. Untuk membantu dalam hal pemeliharaan dan utilitas kumpulan data dalam jumlah besar.
2. Untuk mengelola database, mulai dari membuat database itu sendiri, sampai dengan proses-proses yang berlaku dalam database tersebut, baik berupa entry, edit, hapus, query terhadap data, membuat laporan dan lain sebagainya secara efektif dan efisien.

Pengelolaan atau manajemen basis data meliputi :

1. Mendefinisikan struktur dari penyimpanan data/informasi.
2. Menyediakan mekanisme untuk memanipulasi informasi.
3. Memberikan keamanan bagi data yang tersimpan, baik terhadap kerusakan sistem maupun akses dari pihak yang tidak berhak.
4. Menghasilkan data/informasi yang tidak bersifat anomali jika data digunakan secara bersama oleh banyak pengguna.

Untuk melakukan fungsi dan manajemen basis data di atas maka digunakan suatu perangkat lunak yang disebut dengan Database Management System (DBMS). Dan karena



basis data yang digunakan entitasnya saling berhubungan (relationship), maka disebut juga Relationship DBMS (RDBMS). MySQL, Microsoft Access, PostgreSQL, dan Oracle merupakan sebagian kecil dari perangkat lunak DBMS yang banyak sekali ditawarkan.

Pemodelan Data

Pemodelan data (model data) merupakan sekumpulan perangkat konseptual untuk menggambarkan data, hubungan antar data, semantik (makna) data dan batasan data. Dengan menggunakan model data ini maka akan memberikan kemudahan untuk melakukan evaluasi/analisis, dan dilakukan perbaikan sebelum diimplementasikan ke dalam DBMS. Pemodelan data ada beberapa model, yaitu model Hirarkis, Jaringan, dan Relasional serta berorientasi objek. Yang paling banyak dipakai sekarang ini adalah model relasional.

Model relasional yang terkenal adalah Entity Relationship Diagram (ERD). Ada tiga hal yang paling penting dalam ERD, yaitu terdapat Entitas, Relationship, dan Atribut.

1. Entitas

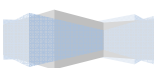
Entitas adalah objek (sesuatu) yang ada (eksis) dan dapat dibedakan dengan objek yang lain (buku, orang, liburan, absensi). Keberadaan dari entitas biasanya berdiri sendiri dan bisa nyata maupun tidak nyata. Entitas digambarkan (direpresentasikan) dengan menggunakan sekumpulan atribut, entitas orang mempunyai atribut nama, alamat, tanggal lahir dan sebagainya.

2. Atribut

Atribut merupakan penggambaran (data) dari entitas. Setiap atribut harus dijelaskan dengan suatu nilai, misalnya entitas orang mempunyai atribut nama dengan nilai “Adi Dewanto”. Nilai dari atribut tersebut juga dapat diatur, misalnya panjang karakter dari nama tidak boleh lebih dari 15 karakter. Pengaturan nilai atribut ini disebut dengan Domain.

Selain dapat diatur, nilai atribut juga dapat bernilai tunggal maupun jamak (multi-valued), sederhana (simple) atau gabungan (composite), “kosong” (Null) atau harus ada (Not Null), dan key (Primary Key atau Foreign Key) atau non key.

Dalam atribut juga terdapat istilah Stored Attribute, yaitu atribut yang langsung terlihat pada entitas (atribut nama, atribut alamat) dan Derived Attribute, yaitu merupakan atribut hasil perhitungan dari atribut yang lain (misal atribut umur dihitung dari atribut tanggal lahir).

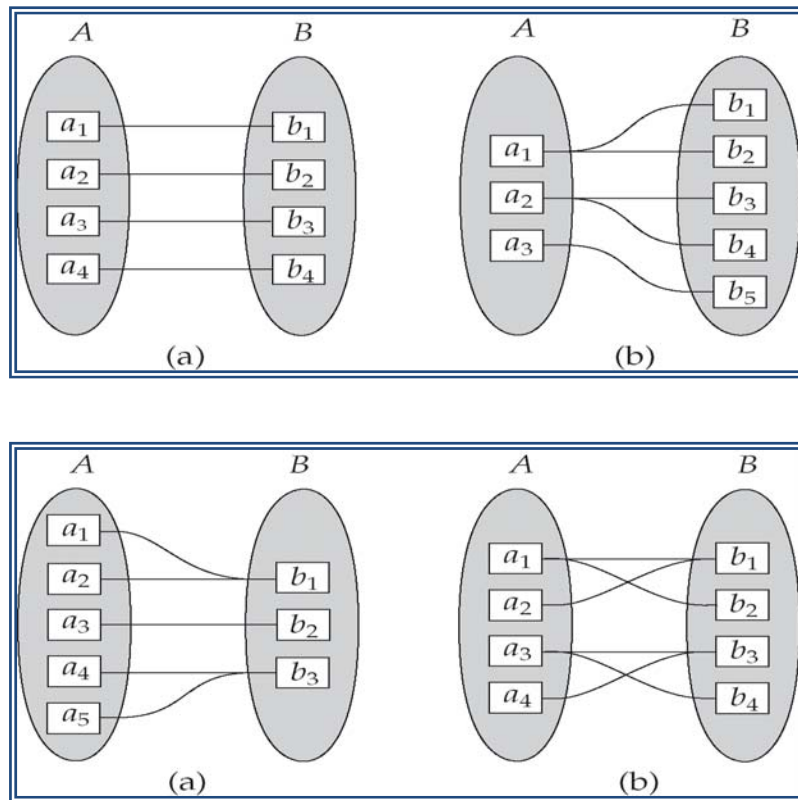


3. Relationship

Relationship menggambarkan asosiasi (hubungan) yang nyata diantara beberapa entitas. Tingkatan dari Relationship ini adalah :

- 1) Unary: Cuma ada satu Entitas.
- 2) Binary: melibatkan dua Entitas.
- 3) Ternary: melibatkan lebih dari dua Entitas (Ternary jarang sekali terjadi, paling banyak Binary).

Sedangkan derajat relasinya dapat one to one (Gambar 1.a), one to many (Gambar 1.b), many to one (Gambar 1.c), dan many to many (Gambar 1.d).



Gambar 1

Gambar 2 merupakan Entity Relationship Diagram (ERD) yang merupakan contoh basis data sederhana dari perpustakaan. Dari ERD Perpustakaan di atas terlihat bahwa hubungan antara entitas penerbit dan entitas buku adalah One to Many (satu ke banyak), yang artinya satu penerbit dapat menerbitkan banyak buku. Sebaliknya satu buku pasti diterbitkan oleh satu penerbit. Silakan diterjemahkan sendiri untuk hubungan antar entitas yang lainnya.

Hubungan antar entitas ini bisa terjadi karena adanya atribut yang berfungsi sebagai penghubung. Pada entitas penerbit dan buku terlihat ada kesamaan nama, yaitu kode_penerbit.

Kode penerbit inilah yang digunakan sebagai penghubung antara kedua entitas tersebut. Atribut kode_penerbit pada tabel penerbit dinamakan Primary Key, sedangkan pada tabel buku dinamakan Foreign Key. Ciri dari Primary Key adalah :

1. Unik

Tidak boleh ada data yang sama, contohnya NIP. Contoh di atas setiap penerbit diberi identitas unik yang berbeda satu sama lainnya.

2. Not Null

Data ini harus ada, contohnya setiap pegawai negeri pasti punya NIP.

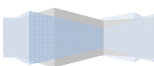


Gambar 2

Dalam implementasinya ke dalam DBMS, entitas dan atribut dalam ERD akan menjadi tabel dan kolom.

MySQL

MySQL merupakan sebuah RDBMS yang berbasiskan Database Server. Kemampuannya dalam menangani RDBMS mengakibatkan MySQL menjadi sangat terkenal dan populer pada saat ini. MySQL mampu menangani data yang sangat besar (Giga byte) sehingga cocok untuk menangani data pada perusahaan besar maupun kecil.



Praktikum Basis Data

1. Mengakses MySQL

A. Masuk ke MySQL Server

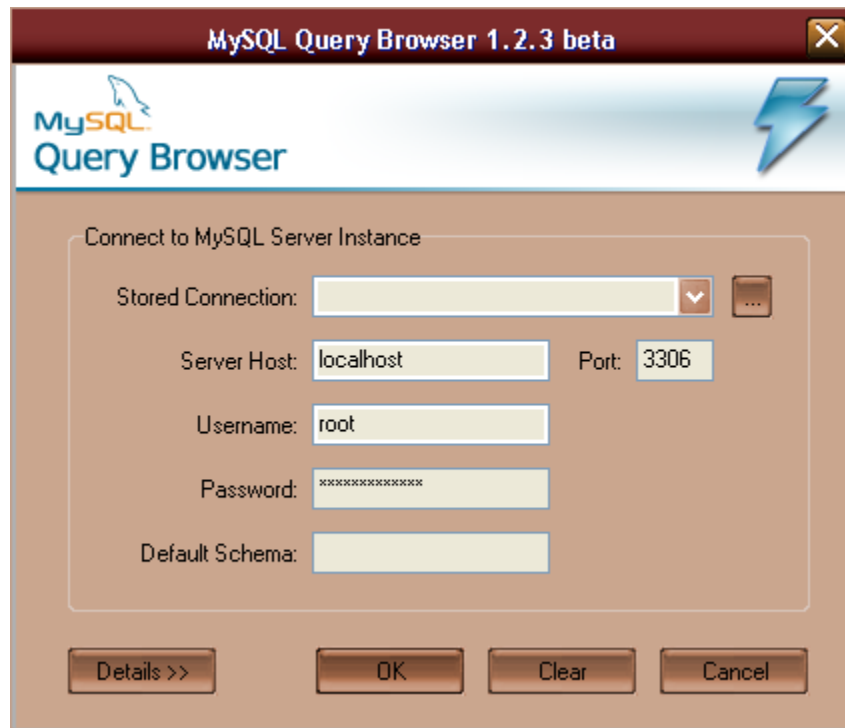
a. Melalui MS DOS

Ketik **mysql -u root -p** pada MS DOS Prompt kemudian tekan enter.

-u adalah option untuk memilih user (dalam hal ini root).

-p adalah option untuk menulis password milik user.

b. Melalui MySQL Browser



B. Melihat database yang ada dalam MySQL

show databases;

Jika menggunakan MySQL Browser jika kita berhasil Login maka secara otomatis akan terlihat semua database yang ada, sesuai dengan hak akses yang kita miliki.

C. Membuat database dengan nama pelatihan

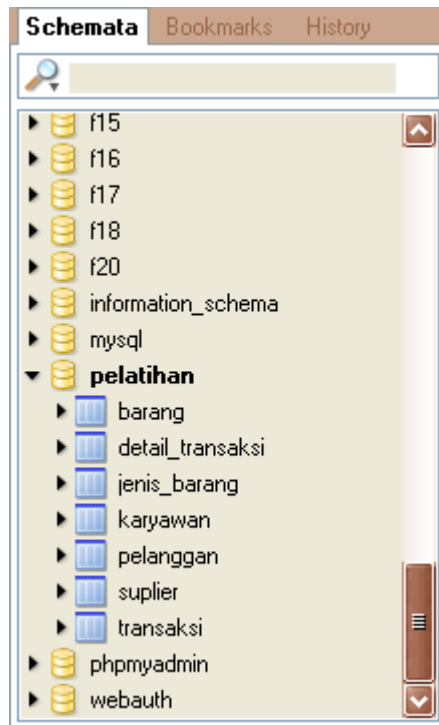
create database pelatihan;

D. Menggunakan database, misal database mysql

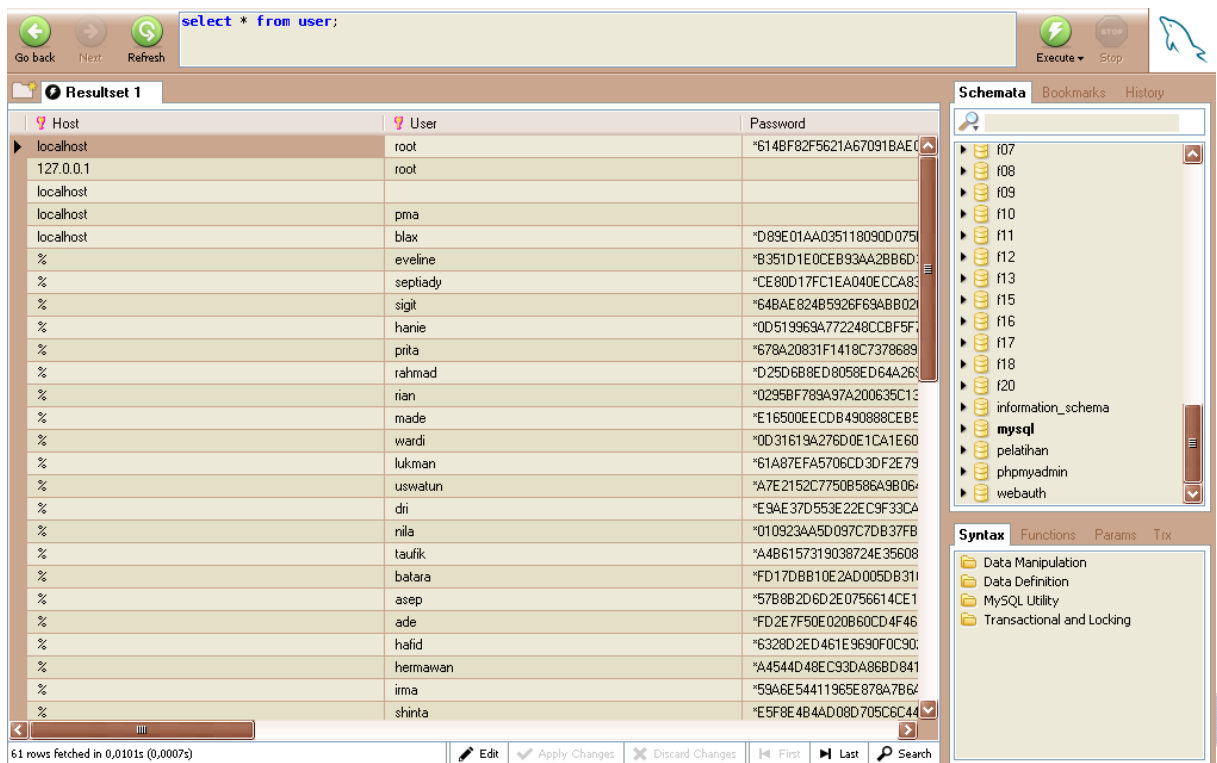
use mysql;

Untuk MySQL Browser, selain dengan cara menuliskan perintah di atas, anda juga dapat menggunakan cara klik dua kali pada database yang diinginkan.

- E. Melihat tabel dalam suatu database
show tables;



- F. Melihat seluruh isi tabel, misal tabel user pada database mysql
select * from user;



Host	User	Password
localhost	root	*614BF82F5621A67091BAE0
127.0.0.1	root	
localhost		
localhost	pma	
localhost	blax	*D89E01AA035118090D0750
%	eveline	*B351D1E0CEB93AA2BB6D0
%	septiady	*CE80D17FC1EA040ECCA80
%	sigit	*64BAE824B5926F69AB020
%	hanie	*0D519969A772248CCBF5F
%	prita	*678A20831F1418C7378689
%	rahmad	*D25D6B8ED8058ED64A260
%	rian	*0295BF789A97A200635C1E
%	made	*E16500EE CDB490888CEBE
%	wardi	*0D31619A276D0E1CA1E60
%	lukman	*61A87EFA5706CD3DF2E79
%	uswatun	*A7E2152C7750B586A9B06
%	dii	*E9AE37D553E22EC9F33CA
%	nila	*010923AA5D097C7DB37FB
%	taufik	*A4B6157319038724E35608
%	batara	*FD17DBB10E2AD005DB310
%	asep	*57B8B2D6D2E0756614CE1
%	ade	*FD2E7F50E020B60CD4F46
%	hafid	*6328D2ED461E9690F0C90
%	hermawan	*A4544D48EC93DA86BD841
%	irma	*59A6E54411965E878A7B62
%	shinta	*E5F8E4B4AD08D705CC6C44

G. Membuat user baru

Untuk membuat user, anda harus menggunakan database mysql terlebih dahulu.

Caranya adalah ketik: **use mysql;**

Kemudian gunakan perintah berikut :

```
grant create,select,insert,delete,update,create view,show view,drop,alter  
on pelatihan.*  
to 'nama_user'@'host'  
identified by 'password_user';
```

H. Menghapus user

Gunakan salah satu dari perintah berikut :

1. **drop user 'nama_user'@'host';**
2. **delete from user**
where user='nama_user';

I. Memberi privileges pada user

```
grant drop,alter  
on pelatihan.*  
to 'nama_user'@'host';
```

J. Menghapus privileges pada user

```
revoke drop,alter  
on pelatihan.*  
from 'nama_user'@'host';
```

2. Structured Query Language (SQL)

SQL adalah suatu bahasa yang digunakan untuk mengakses data pada basis data relational. Komponen SQL terdiri dari tiga bagian, yaitu DDL (Data Definition Language), DML (Data Manipulation Language), dan DCL (Data Control Language).

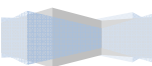
A. DDL

a. Creating Table

```
CREATE TABLE <nama_tabel>  
(<nama_kolom_1> <tipe_data>(<panjang_data>  
[UNIQUE] [NOT NULL] [PRIMARY KEY] [DEFAULT <nilai_default>]  
[referential_constraint_definition] [INDEX],  
<nama_kolom_2> <tipe_data>(<panjang_data>  
[UNIQUE] [NOT NULL] [PRIMARY KEY] [DEFAULT <nilai_default>]  
[referential_constraint_definition] [INDEX], ...);
```

Contoh:

1. **CREATE TABLE Suplier (KodeSuplier Integer *Primary Key*, NamaSuplier Char(20) NOT NULL, Alamat Char(30) NOT NULL DEFAULT 'Jl. Gejayan No. 1', Telpon Char(15));**



2. **CREATE TABLE Suplier (KodeSuplier Integer, NamaSuplier Char(20) NOT NULL, Alamat Char(30) NOT NULL DEFAULT 'Jl. Gejayan No. 1', Telpon Char(15), Primary Key (KodeSuplier));**

b. Creating Index

CREATE INDEX <nama_index> ON <nama_tabel>(<nama_kolom>);

Contoh :

CREATE INDEX Suplier_my ON Suplier(NamaSuplier);

c. Altering

ALTER TABLE <nama_tabel>

[ADD <nama_kolom_1> <tipe_data>(<panjang_data>),

<nama_kolom_2> <tipe_data>(<panjang_data>), ...;]

[ADD CONSTRAINT <nama_constraint> UNIQUE (<nama_kolom>);]

[DROP <nama_kolom_1>,...;]

[FIRST | AFTER <nama_kolom>]

Merubah nama kolom

ALTER TABLE <nama_tabel> CHANGE

<nama_kolom_lama> <nama_kolom_baru> <tipe_data>(<panjang_data>)

[FIRST | AFTER <nama_kolom>]

Merubah nama tabel

ALTER TABLE <nama_tabel>

RENAME TO <nama_tabel_baru>

Contoh :

○ **ALTER TABLE Suplier ADD Kota Char(15);**

○ **ALTER TABLE Suplier ADD Kota Char(15) AFTER alamat;**

○ **ALTER TABLE Suplier DROP Kota;**

○ **ALTER TABLE Suplier ADD CONSTRAINT NamaSupl UNIQUE (NamaSuplier);**

○ **ALTER TABLE Suplier CHANGE**

telpon telepon Char(15)

AFTER NamaSuplier;

○ **ALTER TABLE Suplier RENAME TO Supliyer;**

○ **ALTER TABLE Suplier DROP Primary Key;**

d. Dropping

DROP TABLE <nama_table>;

Contoh :

DROP TABLE Suplier;

B. DML

a. Inserting

INSERT INTO <nama_tabel> [(<nama_kolom_1>, <nama_kolom_2>, ...)]

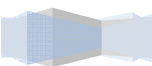
VALUES (<nilai_kolom_1>, <nilai_kolom_2>, ...);

Contoh :

INSERT INTO Suplier (NamaSuplier, KodeSuplier, Alamat, Kota, Telpon, Email)

VALUES ('Asmara Djati', 212, 'Jl. Gejayan 1', 'Yogyakarta', '0274-557689',

'asmara@yahoo.com');



b. Updating

```
UPDATE <nama_tabel>  
SET <nama_kolom_1 = nilai_kolom_1>,  
    <nama_kolom_2 = nilai_kolom_2>, .....,  
    <nama_kolom_N = nilai_kolom_N>  
[WHERE <kondisi>]
```

Contoh :

UPDATE Suplier

SET Alamat = 'Jl. Sudirman 100', Kota='Jakarta', Telpon='021-78654579'
WHERE NamaSuplier='Pungkas Mandiri';

c. Deletion

```
DELETE FROM <nama_tabel>  
WHERE <kondisi>
```

Contoh :

DELETE FROM Suplier WHERE NamaSuplier='Asmara Djati';

d. Selection

```
SELECT <nama_kolom_1, nama_kolom_2, ..., nama_kolom_N>  
FROM <nama_tabel>  
WHERE <kondisi>
```

Contoh :

- o **SELECT * FROM Suplier;**
- o **SELECT NamaSuplier, Telpon FROM Suplier;**
- o **SELECT NamaSuplier, Telpon FROM Suplier
WHERE Alamat='Yogyakarta';**

e. Creating View

```
CREATE VIEW <name> AS <SELECT statement>;
```

Contoh :

**CREATE VIEW siswa_yogya AS SELECT * FROM sisaw
WHERE kota='Yogyakarta';**

C. DCL

Untuk Administrator Database. Perintah-perintahnya antara lain :

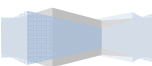
- CREATE DATABASE
- CREATE USER
- DROP USER
- GRANT PRIVILEGE
- REVOKE PRIVILEGE
- dan masih banyak lagi.

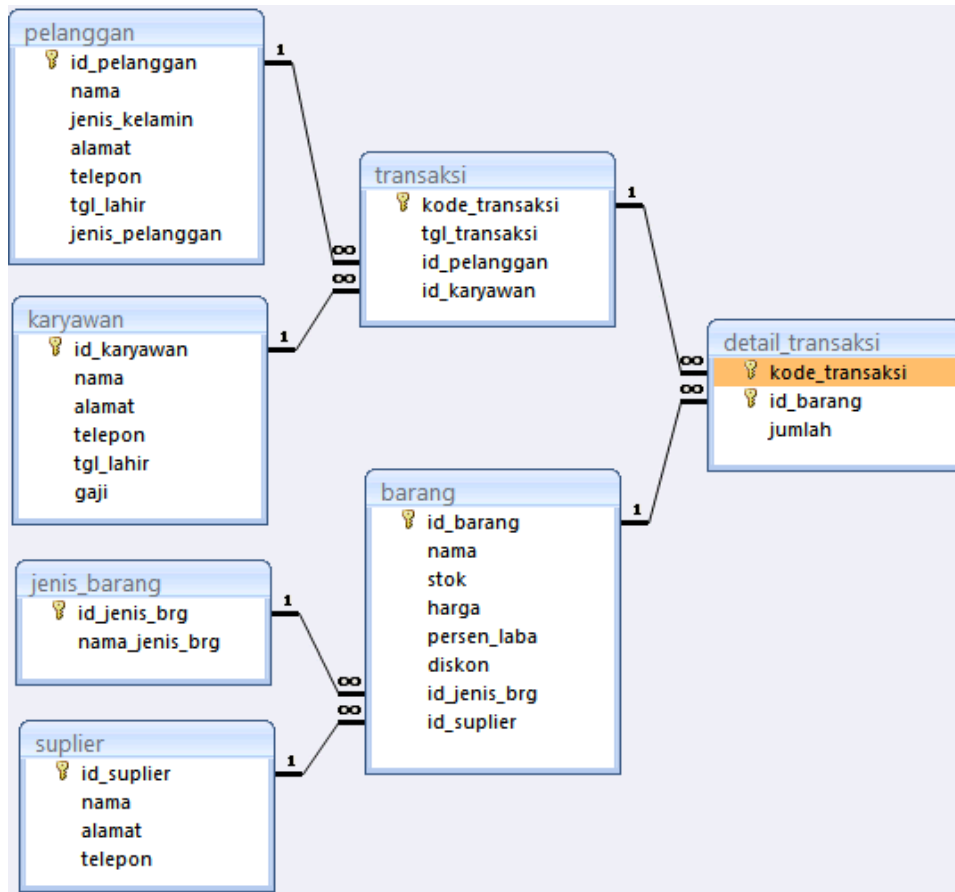
3. Pemetaan Model Data

Pemetaan model data secara konseptual yang berupa ERD ke model data DBMS dalam basis data dilakukan dengan menggunakan perintah SQL yang masuk dalam bagian DDL yaitu Creating Table, Creating Index, Altering, dan Dropping Table.

Sedangkan untuk memasukkan dan mengambil data dilakukan dengan menggunakan perintah SQL yang masuk dalam bagian DML yaitu Inserting, Updating, Deletion, Selection, dan Creating View.

Berikut adalah model data konseptual yang digunakan dalam pelatihan :





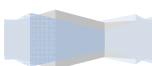
Sedangkan kamus datanya adalah sebagai berikut :

1. pelanggan

Field	Type	Null	Key	Default	Keterangan
id_pelanggan	char(4)	No	PK		Primary Key Format = P999
nama	varchar(20)	No			
jenis_kelamin	enum('L','P')	No		L	L=laki-laki P=perempuan
alamat	varchar(50)	No			
telepon	varchar(15)	Yes			
tgl_lahir	date	No			
jenis_pelanggan	enum('G','S')	No		S	G=Gold S=Silver

2. karyawan

Field	Type	Null	Key	Default	Keterangan
id_karyawan	char(4)	No	PK		Primary Key Format = K999
nama	varchar(20)	No			
alamat	varchar(50)	No			
telepon	varchar(15)	Yes			
tgl_lahir	date	No			
gaji	double	No			



3. transaksi

Field	Type	Null	Key	Default	Keterangan
kode_transaksi	char(4)	No	PK		Primary Key Format = J999
tgl_transaksi	datetime	No			
id_pelanggan	char(4)	No	FK		Foreign Key
id_karyawan	char(4)	No	FK		Foreign Key

4. jenis_barang

Field	Type	Null	Key	Default	Keterangan
id_jenis_brg	tinyint	No	PK		Primary Key, Auto Increment
nama_jenis_brg	varchar(15)	No			

5. suplier

Field	Type	Null	Key	Default	Keterangan
id_suplier	char(4)	No	PK		Primary Key Format = S999
nama	varchar(20)	No			
alamat	varchar(50)	No			
telepon	varchar(15)	No			

6. barang

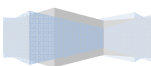
Field	Type	Null	Key	Default	Keterangan
id_barang	char(4)	No	PK		Primary Key Format = B999
nama	varchar(20)	No			
stok	smallint	No			
harga	float	No			
persen_laba	float	No		5.0	
diskon	float	No		0.0	
id_jenis_brg	tinyint	No	FK		Foreign Key
id_suplier	char(4)	No	FK		Foreign Key

7. detail_transaksi

Field	Type	Null	Key	Default	Keterangan
kode_transaksi	char(4)	No	PK,FK		Primary Key, Foreign Key Format = T999
id_barang	char(4)	No	PK,FK		Primary Key, Foreign Key
jumlah	smallint	No		1	

Pemetaan model data (ERD) dan kamus data ke DBMS MySQL adalah :

1. create table pelanggan(id_pelanggan char(4) primary key, nama varchar(20) not null, jenis_kelamin enum('P','W') not null default 'P', alamat varchar(50) not null, telepon varchar(15), tgl_lahir date not null, jenis_pelanggan enum('G','S') not null default 'S') engine=innodb;



2. create table karyawan(id_karyawan char(4) primary key, nama varchar(20) not null, jenis_kelamin enum('P','W') not null default 'P', alamat varchar(50) not null, telepon varchar(15), tgl_lahir date not null, gaji double not null) engine=innodb;
3. create table transaksi(kode_transaksi char(4) primary key, tgl_transaksi datetime not null, id_pelanggan char(4) not null, id_karyawan char(4) not null, foreign key (id_pelanggan) references pelanggan(id_pelanggan) on delete cascade on update cascade, foreign key (id_karyawan) references karyawan(id_karyawan) on delete cascade on update cascade) engine=innodb;
4. create table jenis_barang(id_jenis_brg tinyint primary key auto_increment, nama_jenis_brg varchar(20) not null) engine=innodb;
5. create table supplier(id_supplier char(4) primary key, nama varchar(20) not null, alamat varchar(50) not null, telepon varchar(15) not null) engine=innodb;
6. create table barang(id_barang char(4) primary key, nama varchar(20) not null, stok smallint not null, harga float not null, persen_laba float not null default 5.0, diskon float not null default 0.0, id_jenis_brg tinyint not null, id_supplier char(4) not null, foreign key (id_jenis_brg) references jenis_barang(id_jenis_brg) on delete cascade on update cascade, foreign key (id_supplier) references supplier(id_supplier) on delete cascade on update cascade) engine=innodb;
7. create table detail_transaksi(kode_transaksi char(4), id_barang char(4), jumlah smallint not null, primary key (kode_transaksi, id_barang), foreign key (kode_transaksi) references transaksi(kode_transaksi) on delete cascade on update cascade, foreign key (id_barang) references barang(id_barang) on delete cascade on update cascade) engine=innodb;

4. DML

Perintah-perintah yang termasuk dalam DML adalah insert, update, delete, select, dan view.

1. INSERT

Sintak untuk perintah insert adalah :

```
INSERT INTO <nama_tabel> [(<nama_kolom_1>, <nama_kolom_2>, ...)]
VALUES (<nilai_kolom_1>, <nilai_kolom_2>, ...);
```

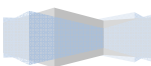
Sintak [(<nama_kolom_1>, <nama_kolom_2>, ...)] mengidentifikasi bahwa sintak tersebut boleh ditulis maupun tidak. Sintak tersebut harus ditulis jika kita ingin memasukkan record dimana tidak semua field kita isi datanya dan atau pemasukan data pada field tidak berurutan (tidak sesuai dengan urutan kolom pada tabel). Sedangkan sintak tersebut tidak perlu ditulis jika kita ingin memasukkan record dengan data field yang terurut (sesuai dengan urutan kolom pada tabel).

Contoh :

Perhatikan struktur dari tabel pelanggan yang telah anda buat pada praktikum sebelumnya. Urutan kolom pada tabel pelanggan tersebut adalah id_pelanggan, nama, jenis_kelamin, alamat, telepon, tgl_lahir, dan jenis_pelanggan. Jalankan perintah SQL berikut ini :

- a. INSERT INTO pelanggan


```
VALUES ('P011','Debimpri','P','Jl. Balikpapan No.8','0858732620',
      '1975-3-11','G');
```



b. INSERT INTO pelanggan (nama, id_pelanggan, jenis_kelamin, tgl_lahir, alamat, telepon, jenis_pelanggan)
VALUES ('Hardiansyah', 'P012',DEFAULT, '1970-12-8','Jl. Bondowoso No.95',NULL, 'G');

Untuk memasukkan lebih dari satu record, maka sintak (<nilai_kolom_1>, <nilai_kolom_2>, ...) ditulis secara berulang kali dimana record satu dengan yang lainnya dipisahkan dengan tanda koma (,).

Contoh :

```
INSERT INTO pelanggan (nama,alamat,id_pelanggan,tgl_lahir) VALUES ('Ibrahim','Jl. Kacer 2 No.19','P013','1974-6-5'), ('Gatot','Jl. Bukit Menoreh No.66','P014','1984-10-27');
```

Sintak VALUES pada perintah insert dapat diganti dengan sintak SET yang merupakan sintak alternatif. Sintak alternatif ini hanya dapat untuk memasukkan satu buah record saja.

Contoh :

```
INSERT INTO pelanggan SET nama='Adita', alamat='Jl. Giri Loka 3 No.28', id_pelanggan='P015', tgl_lahir='1983-5-17', jenis_kelamin='P', handphome=NULL, jenis_pelanggan='G';
```

Selain menggunakan perintah insert, kita juga dapat menggunakan perintah replace untuk memasukkan data. Penggunaan perintah replace sama dengan perintah insert, perbedaan antara keduanya adalah kita tidak dapat memasukkan suatu record dengan perintah insert jika nilai data field dari primary key dan atau field lain yang unik sama dengan record yang sudah ada. Akan tetapi jika kita memakai perintah replace, maka record yang lama akan dihapus dan diganti dengan record yang baru.

Contoh :

- a. INSERT INTO pelanggan
VALUES ('P009','Andi Ikhsan',DEFAULT,'Jl. Makasar 13',NULL,'1969-7-19','G');
- b. REPLACE INTO pelanggan
VALUES ('P009','Andi Ikhsan',DEFAULT,'Jl. Makasar 13', '08164228112', '1969-7-19','G');
- c. REPLACE INTO pelanggan
SET id_pelanggan='P015', nama='Adita Putri', alamat=' Jl. Giri Loka 3 No.28', handphome='08164622383', jenis_kelamin='P', jenis_pelanggan=DEFAULT, tgl_lahir='1988-2-2';

2. UPDATE

Sintak untuk perintah update adalah :

```
UPDATE <nama_tabel>  
SET <nama_kolom_1 = nilai_kolom_1>,  
    <nama_kolom_2 = nilai_kolom_2>, ...,  
    <nama_kolom_N = nilai_kolom_N>  
[WHERE <kondisi>]
```

Sintak [WHERE <kondisi>] mengidentifikasi bahwa sintak tersebut boleh ditulis maupun tidak. Jika sintak tersebut tidak ditulis maka semua record yang ada akan berubah. Tetapi jika kita hanya ingin mengubah satu atau lebih record yang sesuai dengan kondisi atau syarat yang kita tentukan, maka sintak tersebut harus ditulis.

Contoh :

Tampilkan semua data (record) pada tabel barang dan perhatikan jumlah stoknya. Kemudian jalankan perintah SQL berikut ini dan amati serta diskusikan perubahan yang terjadi dengan teman anda :

- a. UPDATE barang SET stok=stok+1;
- b. UPDATE barang SET stok=stok+5, harga=13750 WHERE nama='Ember';
- c. UPDATE barang SET harga=harga+125 WHERE stok<60;

3. DELETE

Sintak untuk perintah delete (telah diberikan pada modul 2) adalah :

```
DELETE FROM <nama_tabel>  
WHERE <kondisi>
```

Sintak WHERE <kondisi> harus ditulis karena kalau tidak maka semua data akan terhapus semua. Dengan demikian untuk menghapus suatu data, anda harus tahu kondisi atau syarat dari penghapusan agar tidak terjadi kesalahan.

Contoh :

Tampilkan semua data (record) pada tabel pelanggan, kemudian jalankan perintah SQL berikut ini dan amati serta diskusikan perubahan yang terjadi dengan teman anda :

- a. Masukkan data berikut :
INSERT INTO pelanggan VALUES
('P020','Gatot','P','Jl. Ring Road Selatan 111',NULL,'1982-12-12','S');
- b. DELETE FROM pelanggan WHERE nama='Gatot';
- c. Untuk contoh berikut ini perhatikan tabel transaksi dan cari kode transaksi J012. Transaksi ini dilakukan oleh pelanggan yang bernama Meiliana dengan id pelanggan P008. Kemudian jalankan perintah SQL berikut :

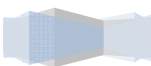
```
DELETE FROM pelanggan WHERE nama='Meiliana';
```

Pelanggan dengan nama Meiliana dan transaksi dengan kode_transaksi J012 akan terhapus. Hal ini dapat terjadi karena pada waktu kita membuat tabel transaksi, kita menggunakan sintak ON DELETE CASCADE. Sebaliknya jika pada waktu kita membuat tabel transaksi tidak menggunakan sintak ON DELETE CASCADE, maka penghapusan tersebut tidak akan diperkenankan karena pelanggan Meiliana dengan kode pelanggan P008 direferensi (terkait) oleh tabel lain, yaitu tabel transaksi.

4. Diskusi

1. Pungkas Mandiri sebagai salah satu dari supplier yang memasok barangnya ke toko telah menaikkan harga barangnya sebesar 10%. Buatlah perintah SQL-nya.
2. Jalankan perintah SQL berikut :
 - o INSERT INTO supplier VALUES
('S009','Dapur Ngepul','Jl. Dapurmu No.666','08122855666');
 - o INSERT INTO supplier VALUES
('S010','Patmo','Jl. Perjuangan No.45','08122954545');
 - o INSERT INTO barang VALUES
('B014','Wafer Tanggo','103','2500','5.0','2.5','1','S009');
 - o INSERT INTO barang VALUES
('B015','Wafer Nissin','73','2400','4.0','2.5','1','S010');

Bagaimana perintah SQL untuk menghapus supplier yang memasok Wafer Tanggo dan Wafer Nissin?



3. Jalankan perintah SQL berikut :
 - INSERT INTO jenis_barang SET nama_jenis_brg='Makanan';
 - INSERT INTO supplier VALUES ('S011','Sridewi','Jl. Goa Selarong No.2','0817565656');
 - INSERT INTO barang VALUES ('B016','Nasi Rawon','5','7750','0','0','5','S011');

Bagaimana perintah SQL untuk menghapus supplier yang memasok makanan?

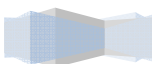
5. DML Menampilkan Data

Untuk mengambil data yang sesuai dengan syarat tertentu, perintah query yang sering digunakan adalah klausa WHERE diikuti dengan operator-operator perbandingan dan logika.

1) WHERE : sintaknya adalah WHERE [Search Condition].

2) COMPARISONS OPERATORS

Operator	Description
=	Evaluates to true if both arguments are equal, unless both conditions are NULL.
<=>	Evaluates to true if both arguments are equal, even if both conditions are NULL.
<>, !=	Evaluates to true if the two arguments are not equal.
<	Evaluates to true if the value of the first argument is less than the value of the second argument.
<=	Evaluates to true if the value of the first argument is less than or equal to the value of the second argument.
>	Evaluates to true if the value of the first argument is greater than the value of the second argument.
>=	Evaluates to true if the value of the first argument is greater than or equal to the value of the second argument.
IS NULL	Evaluates to true if the argument equals a null value.
IS NOT NULL	Evaluates to true if the argument does not equal a null value.
BETWEEN	Evaluates to true if the value of the argument falls within the range specified by the BETWEEN clause.
NOT BETWEEN	Evaluates to true if the value of the argument does not fall within the range specified by the NOT BETWEEN clause.
IN	Evaluates to true if the value of the argument is specified within the IN clause.
NOT IN	Evaluates to true if the argument is not specified within the NOT IN clause.
LIKE	Evaluates to true if the value of the argument is not specified by the LIKE construction.
NOT LIKE	Evaluates to true if the value of the argument is not specified by the NOT LIKE construction.



Operator	Description
REGEXP	Evaluates to true if the value of the argument is specified by the REGEXP construction.
NOT REGEXP	Evaluates to true if the value of the argument is not specified by the NOT REGEXP construction.

3) LOGICAL OPERATORS

Operator	Description
AND	Evaluates to true if both of the two arguments or expressions evaluate to true. You can use double ampersands (&&) in place of the AND operator.
OR	Evaluates to true if either of the two arguments or expressions evaluates to true. You can use the double vertical pipes () in place of the OR operator.
XOR	Evaluates to true if exactly one of the two arguments or expressions evaluates to true.
NOT, !	Evaluates to true if the argument or expression evaluates to false. You can use an exclamation point (!) in place of the NOT operator.

Jalankan contoh-contoh perintah SQL berikut ini.

a. Klausa where

- 1) SELECT nama,alamat,jenis_pelanggan FROM pelanggan WHERE jenis_pelanggan='G';
- 2) SELECT nama,alamat,jenis_kelamin FROM pelanggan WHERE jenis_kelamin='P';

b. Klausa untuk perbandingan

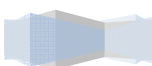
- 1) SELECT nama,stok,harga FROM barang WHERE stok>100;
- 2) SELECT nama,stok,harga FROM barang WHERE harga<=2500;
- 3) SELECT nama,jenis_kelamin FROM karyawan WHERE jenis_kelamin<>'P';

c. Klausa is null dan is not null

Tambahlah kolom email untuk tabel suplier dimana email setiap suplier tidak boleh sama. Kemudian isi data email untuk suplier sebagai berikut :

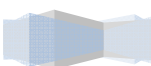
- Pungkas Mandiri : pt_puma@puma.com
- Kiat Sejahtera : sejahtera.kiat@yahoo.co.id
- Isaku Niki : yoiki_isakudab@gmail.com
- Indo Mandiri : mandirindo@gomandiri.org

- 1) SELECT nama,alamat,email FROM suplier WHERE email=NULL;
- 2) SELECT nama,alamat,email FROM suplier WHERE email<=>NULL;
- 3) SELECT nama,alamat,email FROM suplier WHERE email IS NULL;
- 4) SELECT nama,alamat,email FROM suplier WHERE email IS NOT NULL;



- d. Klausa between
- 1) `SELECT nama,id_karyawan FROM karyawan WHERE id_karyawan BETWEEN 'K003' AND 'K007';`
 - 2) `SELECT nama,gaji FROM karyawan WHERE gaji BETWEEN 400000 AND 650000;`
 - 3) `SELECT nama,gaji FROM karyawan WHERE gaji>400000 AND gaji<650000;`
- e. Klausa in
- 1) `SELECT nama,harga,persen_laba FROM barang WHERE persen_laba IN (3,6,7);`
 - 2) `SELECT nama,harga,persen_laba FROM barang WHERE nama IN ('chitato');`
 - 3) `SELECT nama,harga,persen_laba,id_suplier FROM barang WHERE id_suplier IN ('s001');`
- f. Klausa like
- 1) `SELECT nama,tgl_lahir FROM pelanggan WHERE nama LIKE 'me%';`
 - 2) `SELECT nama,tgl_lahir FROM pelanggan WHERE nama LIKE '%anto';`
 - 3) `SELECT nama,tgl_lahir FROM pelanggan WHERE nama LIKE '%san%';`
 - 4) `SELECT nama,tgl_lahir FROM pelanggan WHERE nama LIKE 'susan_';`
 - 5) `SELECT nama,tgl_lahir FROM pelanggan WHERE nama LIKE 'susan__';`
 - 6) `SELECT nama,tgl_lahir FROM pelanggan WHERE nama LIKE '_e%';`
- g. Klausa untuk logical operator
- 1) `SELECT nama,stok,harga,persen_laba FROM barang WHERE persen_laba=6 AND stok<100;`
 - 2) `SELECT nama,stok,harga,persen_laba FROM barang WHERE persen_laba<5 OR stok>100;`
 - 3) `SELECT nama,stok,harga,persen_laba FROM barang WHERE persen_laba NOT LIKE 6;`
- h. Klausa regexp dan not regexp
Option yang digunakan dalam klausa ini adalah :

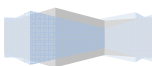
Options	Meaning	Example	Acceptable Values
<value>	The tested value must contain the specified value.	'bo'	about, book, abbot, boot
<^>	The tested value must <i>not</i> contain the specified value.	'^bo'	abut, took, amount, root



Options	Meaning	Example	Acceptable Values
.	The tested value can contain any individual character represented by the period (.).	'b.'	by, be, big, abbey
[<characters>]	The tested value must contain at least one of the characters listed within the brackets.	'[xz]'	dizzy, zebra, x-ray, extra
[<range>]	The tested value must contain at least one of the characters listed within the range of values enclosed by the brackets.	'[1-5]'	15, 3, 346, 50, 22, 791
^	The tested value must begin with the value preceded by the caret (^) symbol.	'^b'	book, big, banana, bike
\$	The tested value must end with the value followed by the dollar sign (\$) symbol.	'st\$'	test, resist, persist
*	The tested value must include zero or more of the character that precedes the asterisk (*).	'^b.*e\$'	bake, be, bare, battle

Contoh penggunaan regexp :

- 1) SELECT nama,alamat FROM pelanggan
WHERE nama REGEXP 'o' ORDER BY nama;
- 2) SELECT nama,tgl_lahir FROM pelanggan
WHERE nama REGEXP 'ah' ORDER BY nama;
- 3) SELECT nama,alamat FROM pelanggan
WHERE nama REGEXP '^[a-d]' ORDER BY nama;
- 4) SELECT nama,alamat FROM pelanggan
WHERE nama REGEXP '^[c-m]' ORDER BY nama;
- 5) SELECT nama,tgl_lahir FROM supplier
WHERE nama REGEXP 'ri\$' ORDER BY nama;
- 6) SELECT nama,handphone FROM supplier
WHERE nama REGEXP '^k.*h\$' ORDER BY nama;
- 7) SELECT nama,tgl_lahir FROM pelanggan
WHERE nama REGEXP '^.....\$' ORDER BY nama;
- 8) SELECT nama, tgl_lahir FROM pelanggan
WHERE nama REGEXP '^.{5}\$' ORDER BY nama;



4) DISKUSI

Buatlah perintah SQL untuk mencari informasi mengenai :

1. Pelanggan yang lahir pada tahun 1983.
2. Pelanggan yang lahir pada bulan mei.
3. Pelanggan yang melakukan transaksi pembelian pada bulan juni.
4. Pelanggan yang membeli Chitato.
5. Barang apa saja yang dibeli oleh Charles pada bulan mei.
6. Suplier yang mempunyai alamat email .com.
7. Suplier yang menggunakan nomor handphone dari telkomsel.
8. Suplier yang mensuplai minuman dan beralamat di jalan merapi dengan menggunakan klausa LIKE dan REGEXP.

6. Perhitungan Data dan Fungsi

Dapat membuat perintah-perintah SQL yang berhubungan dengan perhitungan data dalam basis data dan menggunakan fungsi-fungsi yang ada dalam MySQL.

1) ARITHMETIC OPERATORS

Operator	Description
+ (addition)	Adds the two arguments together.
- (subtraction)	Subtracts the second argument from the first argument.
- (unary)	Changes the sign of the argument.
* (multiplication)	Multiplies the two arguments together.
/ (division)	Divides the first argument by the second argument.
% (modulo)	Divides the first argument by the second argument and provides the remainder from that operation.

Contoh :

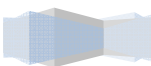
Buat tabel hitungan dengan data berupa angka seperti tabel berikut :

kolom1	kolom2	kolom3
3	7	5
4	12	4
6	5	7
7	9	8
9	8	15
10	11	9

```
SELECT (kolom1+kolom2) 'kolom1 + kolom2',(kolom3*2) 'kolom3 * 2',  
       (kolom2-2) 'kolom2 - 2', (kolom1/2) 'kolom1 / 2',  
       (kolom3%kolom1) 'kolom3 % kolom1'  
FROM hitungan;
```

2) AGGREGATE FUNCTIONS

Fungsi Aggregate : digunakan untuk mengambil data tunggal hasil dari perhitungan data yang tersimpan dalam suatu kolom.



Function	Description
AVG ()	Used to return the average of values stored in a column.
COUNT (*)	Used to count the rows in a table including NULL values.
COUNT (ColumnName)	Used to count the rows in a column excluding NULL values.
MAX ()	Used to return the highest value stored in a column.
MIN ()	Used to return the lowest value stored in a column.
SUM ()	Used to return the sum of values stored in a column.

Contoh :

- a) SELECT AVG(kolom1) 'Rata2', SUM(kolom2) 'Jumlah',
MAX(kolom2) 'Data Terbesar', MIN(kolom2) 'Data Terkecil'
FROM hitungan;
- b) SELECT MAX(nama) 'Nama Besar', MIN(nama) 'Nama Terkecil'
FROM karyawan;
- c) SELECT COUNT(*) 'Menghitung jumlah baris termasuk Null',
COUNT(kolom2) 'Menghitung jumlah baris, Null tidak dihitung'
FROM hitungan;
- d) SELECT COUNT(*) 'Menghitung jumlah baris termasuk Null',
COUNT(email) 'Menghitung jumlah baris, Null tidak dihitung'
FROM suplier;

3) STRING FUNCTION

- a) CHAR_LENGTH(), CHARACTER_LENGTH(), dan LENGTH()
CHAR_LENGTH() dan CHARACTER_LENGTH() untuk menghitung jumlah karakter sedangkan LENGTH() yang dihitung adalah jumlah byte per karakter, bukan karakternya.

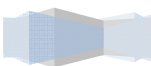
Contoh :

- 1) SELECT nama, CHAR_LENGTH(nama) 'banyak karakter' FROM pelanggan;
- 2) SELECT nama, CHARACTER_LENGTH(nama) 'banyak karakter' FROM pelanggan;
- 3) SELECT nama, LENGTH(nama) 'banyak karakter' FROM pelanggan;

- b) CONCAT() dan CONCAT_WS()

Contoh :

- 1) SELECT CONCAT(nama, ' --> ', alamat) 'nama dan alamat' FROM pelanggan;
- 2) SELECT CONCAT_WS(' --> ', nama, alamat) 'nama dan alamat' FROM pelanggan;



c) LCASE(), LOWER(), UCASE(), dan UPPER()

Contoh :

- 1) SELECT LCASE(nama) 'nama_hrf_kecil' FROM pelanggan;
- 2) SELECT LOWER(nama) 'nama_hrf_kecil' FROM pelanggan;
- 3) SELECT UCASE(nama) 'nama_hrf_besar' FROM pelanggan;
- 4) SELECT UPPER(nama) 'nama_hrf_besar' FROM pelanggan;

d) LEFT() dan RIGHT()

Contoh :

```
SELECT alamat,LEFT(alamat,10),RIGHT(alamat,10) FROM pelanggan;
```

e) REPEAT() dan REVERSE()

Contoh :

```
SELECT REPEAT(nama,3),REVERSE(nama) FROM pelanggan;
```

f) SUBSTRING()

Contoh :

- 1) SELECT alamat,SUBSTRING(alamat,7) FROM pelanggan;
- 2) SELECT alamat,SUBSTRING(alamat FROM 7) FROM pelanggan;
- 3) SELECT alamat,SUBSTRING(alamat,7,5) FROM pelanggan;
- 4) SELECT alamat,SUBSTRING(alamat FROM 7 FOR 5) FROM pelanggan;
- 5) SELECT alamat,SUBSTRING(alamat FROM 1 FOR 10)
FROM pelanggan;
- 6) SELECT alamat,SUBSTRING(alamat FROM -5 FOR 5)
FROM pelanggan;
- 7) SELECT alamat,SUBSTRING(alamat FROM -11 FOR 5)
FROM pelanggan;
- 8) SELECT alamat,SUBSTRING(alamat FROM -11 FOR 11)
FROM pelanggan;

g) MID()

Contoh :

```
SELECT alamat,MID(alamat,1,10) 'Mid alamat' FROM pelanggan;
```

4) NUMERIC FUNCTIONS

a) CEIL(), CEILING(), dan FLOOR()

Contoh :

- 1) SELECT CEIL(19.987);
- 2) SELECT CEILING(19.987);
- 3) SELECT FLOOR(19.987);

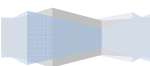
b) MOD()

Contoh :

```
SELECT MOD(10,3);
```

c) PI()

```
SELECT PI();
```



d) POW() dan POWER() → keduanya sama
SELECT POW(3,2);

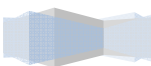
e) ROUND() dan TRUNCATE()
1) SELECT ROUND(4.27943, 2);
2) SELECT TRUNCATE(4.27943, 2);

f) SQRT()
SELECT SQRT(36);

5) DATE/TIME FUNCTIONS

- a) ADDDATE(), DATE_ADD(), SUBDATE(), DATE_SUB(), dan EXTRACT()
- o Fungsi ADDDATE() dan DATE_ADD() adalah sama. Sintak-nya adalah :
ADDDATE(<date>, INTERVAL <expression> <type>)
 - o Fungsi SUBDATE() dan DATE_SUB() adalah sama. Sintak-nya adalah :
SUBDATE(<date>, INTERVAL <expression> <type>)

<type>	<expression> format
MICROSECOND	<microseconds>
SECOND	<seconds>
MINUTE	<minutes>
HOURL	<hours>
DAY	<days>
MONTH	<months>
YEAR	<years>
SECOND_MICROSECOND	'<seconds>.<microseconds>'
MINUTE_MICROSECOND	'<minutes>.<microseconds>'
MINUTE_SECOND	'<minutes>:<seconds>'
HOURL_MICROSECOND	'<hours>.<microseconds>'
HOURL_SECOND	'<hours>:<minutes>:<seconds>'
HOURL_MINUTE	'<hours>:<minutes>'
DAY_MICROSECOND	'<days>.<microseconds>'
DAY_SECOND	'<days> <hours>:<minutes>:<seconds>'
DAY_MINUTE	'<days> <hours>:<minutes>'
DAY_HOUR	'<days> <hours>'
YEAR_MONTH	'<years>-<months>'



Contoh :

- 1) SELECT tgl_lahir,ADDDATE(tgl_lahir, INTERVAL 1 DAY) FROM pelanggan;
- 2) SELECT tgl_lahir,ADDDATE(tgl_lahir, INTERVAL '1:2' YEAR_MONTH) FROM pelanggan;
- 3) SELECT tgl_lahir,ADDDATE(tgl_lahir, INTERVAL 3 MONTH) FROM pelanggan;
- 4) SELECT tgl_transaksi,ADDDATE(tgl_transaksi, INTERVAL '10:20' HOUR_MINUTE) FROM transaksi;
- 5) SELECT tgl_lahir,SUBDATE(tgl_lahir, INTERVAL 1 DAY) FROM pelanggan;
- 6) SELECT tgl_lahir,SUBDATE(tgl_lahir, INTERVAL '1:2' YEAR_MONTH) FROM pelanggan;
- 7) SELECT tgl_lahir,SUBDATE(tgl_lahir, INTERVAL 3 MONTH) FROM pelanggan;
- 8) SELECT tgl_transaksi,SUBDATE(tgl_transaksi, INTERVAL '10:20' HOUR_MINUTE) FROM transaksi;

- o EXTRACT() mempunyai sintak :
EXTRACT(<type> FROM <date>)

Contoh :

- 1) SELECT tgl_lahir,EXTRACT(YEAR_MONTH FROM tgl_lahir) FROM pelanggan;
- 2) SELECT tgl_transaksi,EXTRACT(HOUR_MINUTE FROM tgl_transaksi) FROM transaksi;

- b) CURDATE(), CURRENT_DATE(), CURTIME(), CURRENT_TIME(), CURRENT_TIMESTAMP(), dan NOW()

Fungsi CURDATE() sama dengan fungsi CURRENT_DATE(), fungsi CURTIME() sama dengan CURRENT_TIME() dan fungsi CURRENT_TIMESTAMP() sama dengan NOW().

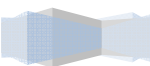
Contoh :

- 1) SELECT CURDATE();
- 2) SELECT CURTIME();
- 3) INSERT INTO transaksi VALUES ('J017',now(tgl_transaksi),'P0019','K003');
- 4) INSERT INTO transaksi VALUES ('J018',CURRENT_TIMESTAMP(tgl_transaksi),'P0019','K003');

- c) DATE(), MONTH(), MONTHNAME(), dan YEAR()

Contoh :

- 1) SELECT nama,tgl_lahir FROM pelanggan WHERE DATE(tgl_lahir)='1980-06-12';
- 2) SELECT nama,tgl_lahir FROM pelanggan WHERE MONTH(tgl_lahir)=3;
- 3) SELECT nama,tgl_lahir FROM pelanggan WHERE MONTHNAME(tgl_lahir)='June';
- 4) SELECT nama,tgl_lahir FROM pelanggan WHERE YEAR(tgl_lahir)=1983;



d) DATEDIFF() dan TIMEDIFF()

Contoh :

- 1) SELECT DATEDIFF('2009-01-01 23:59:59','2008-01-01 23:59:59');
- 2) SELECT TIMEDIFF('2009-01-01 23:59:59','2008-01-01 23:59:59');

e) DAY(), DAYOFMONTH(), DAYNAME(), DAYOFWEEK(), dan DAYOFYEAR()

Contoh :

- 1) SELECT tgl_lahir, DAY(tgl_lahir) FROM pelanggan;
- 2) SELECT tgl_lahir, DAYNAME(tgl_lahir), DAYOFWEEK(tgl_lahir), DAYOFYEAR(tgl_lahir) FROM pelanggan;

f) SECOND(), MINUTE(), HOUR(), dan TIME()

Contoh :

```
SELECT tgl_transaksi, SECOND(tgl_transaksi), MINUTE(tgl_transaksi),  
HOUR(tgl_transaksi), TIME(tgl_transaksi) FROM transaksi;
```

6) DISKUSI

Buatlah perintah SQL untuk mencari :

1. Pelanggan yang lahir pada bulan mei (dengan fungsi).
2. Pelanggan yang melakukan transaksi pembelian pada bulan juni (dengan fungsi).
3. Barang apa saja yang dibeli oleh Charles pada bulan mei.
4. Berapa umur masing-masing pelanggan pada hari ini? (tahun, bulan, hari)
5. Berapa total harga yang harus dibayar oleh Andi untuk kode transaksi J001 (tanpa memperhitungkan diskon).
6. Berapa keuntungan perusahaan pada bulan Juni 2006 (penjualan tanpa ada diskon)?
7. Berapa uang yang harus kita bayar pada PT. Pungkas Mandiri pada bulan mei 2006 karena barangnya berhasil kita jual?

7. Grouping Data

Digunakan bersama-sama dengan fungsi Aggregate untuk menyatukan dua atau lebih grup data kedalam suatu fungsi data tunggal.

Fungsi Aggregate digunakan untuk mengambil data tunggal hasil dari perhitungan data yang tersimpan dalam suatu kolom.

Contoh :

1. Fungsi Aggregate

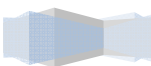
```
SELECT COUNT (nama) AS jumlah_baris  
FROM pelanggan;
```

→ hasilnya merupakan data tunggal yang menunjukkan jumlah baris (atau jumlah supplier) dalam tabel supplier.

2. Group By

Menampilkan data pelanggan dan berapa banyak transaksi pembelian barang yang telah dilakukannya.

- 1) SELECT id_pelanggan, COUNT (id_pelanggan) 'banyak_pembelian'
FROM transaksi GROUP BY id_pelanggan;
- 2) SELECT p.nama, t.id_pelanggan, COUNT (t.id_pelanggan) 'banyak_pembelian'
FROM pelanggan p, transaksi t
WHERE p.id_pelanggan=t.id_pelanggan
GROUP BY p.nama;




```
3) SELECT p.nama,t.id_pelanggan, COUNT (t.id_pelanggan) 'banyak_pembelian'  
FROM pelanggan p,transaksi t  
WHERE p.id_pelanggan=t.id_pelanggan  
GROUP BY t.id_pelanggan;
```

Syarat penggunaan GROUP BY adalah :

1. GROUP BY hanya dapat digunakan dalam Query yang mengandung paling sedikit satu fungsi Aggregate. Dengan demikian jika tidak ada fungsi Aggregate tidak perlu menggunakan GROUP BY.
2. Data harus dikelompokkan dalam atribut tertentu (bisa lebih dari satu) yang disebutkan dalam perintah query (setelah klausa SELECT) yaitu dengan menuliskan atribut tersebut dalam GROUP BY.

Dalam contoh di atas, pengelompokan dapat dilakukan berdasarkan nama pelanggan atau identitas pelanggan atau keduanya.

Dalam DBMS tertentu, misalnya Microsoft Access, semua atribut yang ditulis dalam query (setelah klausa SELECT) harus ditulis semua.

(Catatan : atribut atau nama kolom yang disebut dalam fungsi atau alias tidak perlu ditulis).

3. GROUP BY WITH ORDER BY

GROUP BY dapat ditambah dengan ORDER BY untuk mengurutkan hasil Query dengan syarat :

1. ORDER BY tidak dapat digunakan pada Query yang hanya mengandung fungsi Aggregate, yaitu tanpa ada GROUP BY.
2. GROUP BY harus ditulis sebelum ORDER BY.

Contoh :

```
SELECT p.nama,t.id_pelanggan, COUNT(t.id_pelanggan) 'banyak_pembelian'  
FROM pelanggan p,transaksi t  
WHERE p.id_pelanggan=t.id_pelanggan  
GROUP BY p.nama  
ORDER BY banyak_pembelian DESC;
```

4. HAVING

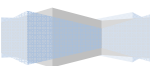
Digunakan dengan GROUP BY untuk mengkondisikan suatu group data hasil perhitungan dari fungsi Aggregate.

HAVING mempunyai fungsi dan sintak yang sama dengan WHERE.

Contoh :

Menampilkan data pelanggan yang melakukan transaksi pembelian barang minimal 2 kali.

```
SELECT p.nama,t.id_pelanggan, COUNT(t.id_pelanggan) 'banyak_pembelian'  
FROM pelanggan p,transaksi t  
WHERE p.id_pelanggan=t.id_pelanggan  
GROUP BY p.nama  
HAVING banyak_pembelian>=2  
ORDER BY banyak_pembelian DESC;
```



5. Diskusi

Buatlah perintah SQL untuk menampilkan :

1. Siapa pelanggan yang paling banyak melakukan transaksi pembelian barang!
2. Besar transaksi pembelian yang dilakukan oleh setiap pelanggan dan kapan transaksi pembelian tersebut tersebut dilakukan (tidak ada diskon)!
Urutkan dari transaksi yang terbesar.
3. Besar keuntungan yang diperoleh perusahaan setiap bulannya! (tidak ada diskon).
4. Berapa jumlah uang yang harus kita bayarkan (setor) ke setiap suplier yang barangnya berhasil kita jual pada tahun 2006?
5. Jenis barang apa yang paling banyak pembeliannya (paling laku)?

8. Basic Join

Mengambil data dari banyak tabel dimana nama-nama tabel yang terlibat ditulis semua dengan menggunakan koma sebagai pemisah antar tabel.

Contoh :

Diketahui permasalahan sebagai berikut :

Menampilkan nama pelanggan yang pernah membeli Sapu (tampilkan nama dan tanggal pembeliannya).

Solusi :

Untuk memperoleh nama pelanggan, kita harus melibatkan tabel pelanggan karena nama pelanggan adanya di tabel tersebut. Sedangkan untuk menampilkan tanggal transaksi kita harus melibatkan tabel transaksi. Terakhir nama barang, dalam hal ini Sapu, kita perlu menggunakan tabel barang. Tabel pelanggan terhubung dengan tabel transaksi (lihat ERD dalam modul 3). Sedangkan untuk dapat mengakses tabel barang, tabel transaksi harus melalui tabel detail transaksi baru dapat terhubung dengan tabel barang.

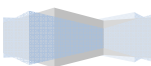
Jadi dalam perintah SQL kita harus melibatkan keempat tabel tersebut (pelanggan, transaksi, detail_transaksi, barang).

Perintah SQL-nya (basic join) :

```
SELECT pelanggan.nama,transaksi.tgl_transaksi  
FROM pelanggan,transaksi,detail_transaksi,barang  
WHERE pelanggan.id_pelanggan=transaksi.id_pelanggan AND  
transaksi.kode_transaksi=detail_transaksi.kode_transaksi AND  
detail_transaksi.id_barang=barang.id_barang AND barang.nama='Sapu'  
ORDER BY pelanggan.nama;
```

Perintah SQL di atas dapat disederhanakan dengan menggunakan alias untuk nama tabel sebagai berikut :

```
SELECT a.nama,b.tgl_transaksi  
FROM pelanggan a,transaksi b,detail_transaksi c,barang d  
WHERE a.id_pelanggan=b.id_pelanggan AND  
b.kode_transaksi=c.kode_transaksi AND  
c.id_barang=d.id_barang AND d.nama='Sapu' ORDER BY a.nama;
```



1) INNER dan CROSS JOIN

Perintah SQL untuk permasalahan yang sama dengan di atas dengan menggunakan INNER JOIN adalah :

```
SELECT pelanggan.nama,transaksi.tgl_transaksi
FROM pelanggan INNER JOIN transaksi ON
pelanggan.id_pelanggan=transaksi.id_pelanggan
INNER JOIN detail_transaksi ON
transaksi.kode_transaksi=detail_transaksi.kode_transaksi
INNER JOIN barang ON detail_transaksi.id_barang=barang.id_barang
WHERE barang.nama='Sapu' ORDER BY pelanggan.nama;
```

Anda juga dapat menggunakan alias untuk nama tabel seperti halnya pada Basic Join :

```
SELECT p.nama,t.tgl_transaksi
FROM pelanggan p INNER JOIN transaksi t ON p.id_pelanggan=t.id_pelanggan
INNER JOIN detail_transaksi d ON t.kode_transaksi=d.kode_transaksi
INNER JOIN barang b ON d.id_barang=b.id_barang
WHERE b.nama='Sapu' ORDER BY p.nama;
```

Atau untuk lebih efisien gunakan USING untuk menggantikan klausa ON seperti berikut ini :

```
SELECT p.nama,t.tgl_transaksi
FROM pelanggan p INNER JOIN transaksi t USING (id_pelanggan)
INNER JOIN detail_transaksi d USING (kode_transaksi)
INNER JOIN barang b USING (id_barang)
WHERE b.nama='Sapu' ORDER BY p.nama;
```

Atau hilangkan klausa INNER seperti berikut ini :

```
SELECT p.nama,t.tgl_transaksi
FROM pelanggan p JOIN transaksi t USING (id_pelanggan)
JOIN detail_transaksi d USING (kode_transaksi)
JOIN barang b USING (id_barang)
WHERE b.nama='Sapu' ORDER BY p.nama;
```

Klausa INNER JOIN dapat diganti dengan CROSS JOIN sehingga kedua perintah tersebut dapat saling menggantikan :

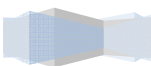
```
SELECT p.nama,t.tgl_transaksi
FROM pelanggan p CROSS JOIN transaksi t USING (id_pelanggan)
CROSS JOIN detail_transaksi d USING (kode_transaksi)
CROSS JOIN barang b USING (id_barang)
WHERE b.nama='Sapu' ORDER BY p.nama;
```

2) LEFT OUTER JOIN dan RIGHT OUTER JOIN

Perhatikan dan diskusikan dengan teman anda hasil dari ketiga perintah SQL berikut ini dan bandingkan dengan BASIC JOIN, INNER JOIN, dan CROSS JOIN.

a) SELECT DISTINCT(p.nama),t.kode_transaksi
FROM pelanggan p INNER JOIN transaksi t USING (id_pelanggan)
ORDER BY t.kode_transaksi;

b) SELECT DISTINCT(p.nama),t.kode_transaksi
FROM pelanggan p LEFT OUTER JOIN transaksi t USING(id_pelanggan)
ORDER BY t.kode_transaksi;



c) `SELECT DISTINCT(p.nama),t.kode_transaksi
FROM transaksi t RIGHT OUTER JOIN pelanggan p USING (id_pelanggan)
ORDER BY t.kode_transaksi;`

Berikut ini merupakan contoh aplikasi dari LEFT dan RIGHT OUTER JOIN :

a) `SELECT DISTINCT(p.nama),t.kode_transaksi
FROM pelanggan p LEFT OUTER JOIN transaksi t USING(id_pelanggan)
WHERE t.kode_transaksi IS NULL ORDER BY t.kode_transaksi;`

b) `SELECT DISTINCT(p.nama),t.kode_transaksi
FROM transaksi t RIGHT OUTER JOIN pelanggan p USING (id_pelanggan)
WHERE t.kode_transaksi IS NULL ORDER BY t.kode_transaksi;`

c) `SELECT DISTINCT(k.nama),t.kode_transaksi
FROM transaksi t RIGHT OUTER JOIN karyawan k USING (id_karyawan)
WHERE t.kode_transaksi IS NULL ORDER BY t.kode_transaksi;`

3) NATURAL JOIN

Bandingkan perintah SQL dengan NATURAL JOIN berikut ini dengan BASIC JOIN, INNER JOIN, CROSS JOIN, LEFT OUTER JOIN, dan RIGHT OUTER JOIN.

a) `SELECT DISTINCT(p.nama),t.kode_transaksi
FROM pelanggan p NATURAL JOIN transaksi t
ORDER BY t.kode_transaksi;`

b) `SELECT DISTINCT(p.nama),t.kode_transaksi
FROM pelanggan p NATURAL LEFT JOIN transaksi t
ORDER BY t.kode_transaksi;`

c) `SELECT DISTINCT(p.nama),t.kode_transaksi
FROM pelanggan p NATURAL LEFT OUTER JOIN transaksi t
ORDER BY t.kode_transaksi;`

d) `SELECT DISTINCT(p.nama),t.kode_transaksi
FROM transaksi t NATURAL RIGHT JOIN pelanggan p
ORDER BY t.kode_transaksi;`

e) `SELECT DISTINCT(p.nama),t.kode_transaksi
FROM transaksi t NATURAL RIGHT OUTER JOIN pelanggan p
ORDER BY t.kode_transaksi;`

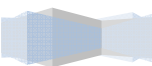
4) UPDATE dan DELETE

Masukkan data baru pada tabel supplier sebagai berikut :

```
INSERT INTO supplier VALUES  
( 'S009','Dapur Ngepul','Jl. Dapurmu No.666','08122855666',NULL),  
( 'S010','Patmo','Jl. Perjuangan No.45','08122954545','45@yahoo.co.id');
```

Masukkan data baru pada tabel barang sebagai berikut :

```
INSERT INTO barang VALUES  
( 'B014','Wafer Tanggo','103','2500','5.0','2.5','1','S009'),  
( 'B015','Wafer Nissin','73','2400','4.0','2.5','1','S010');
```



Salah satu cara melakukan perubahan harga barang yang dipasok oleh supplier Patmo sebesar 5% adalah :

```
UPDATE barang b JOIN supplier s USING(id_supplier)
SET b.harga=b.harga+(b.harga*0.05)
WHERE s.nama='Patmo';
```

Berikut beberapa cara untuk melakukan penghapusan data supplier yang memasok makanan “wafer” adalah :

- a) Apakah perintah SQL ini bisa dieksekusi?
DELETE supplier FROM supplier s, barang b
WHERE s.id_supplier=b.id_supplier AND b.nama REGEXP 'wafer';
- b) Bagaimana dengan keempat perintah SQL berikut ini?
 1. DELETE supplier FROM supplier, barang
WHERE supplier.id_supplier=barang.id_supplier
AND barang.nama REGEXP 'wafer';
 2. DELETE s FROM supplier s, barang b
WHERE s.id_supplier=b.id_supplier AND b.nama REGEXP 'wafer';
 3. DELETE s FROM supplier s JOIN barang b USING (id_supplier)
WHERE b.nama REGEXP 'wafer';
 4. DELETE FROM s USING supplier s, barang b
WHERE s.id_supplier=b.id_supplier AND b.nama REGEXP 'wafer';

5) SUBQUERIES

Merupakan perintah SQL yang terdiri dari lebih dari satu perintah SQL dan digunakan untuk mengambil data dari lebih dari satu tabel. Subquery biasanya terdiri dari dua perintah SQL. Perintah SQL pertama disebut dengan perintah SQL utama dan perintah SQL kedua disebut subquery. Berikut contoh perintah SQL dengan subquery yang digunakan untuk menampilkan supplier yang memasok “sapu” (a) dan supplier yang memasok jenis barang “minuman” (b).

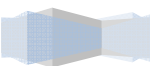
- a) SELECT nama FROM supplier WHERE id_supplier=
(SELECT id_supplier FROM barang WHERE nama='sapu');
- b) SELECT nama FROM supplier WHERE id_supplier<>
(SELECT id_supplier FROM barang WHERE nama='sapu');
- c) SELECT DISTINCT(s.nama) FROM supplier s INNER JOIN barang b USING
(id_supplier) WHERE b.id_jenis_brg=
(SELECT id_jenis_brg FROM jenis_barang WHERE nama_jenis_brg='Minuman');
- d) SELECT DISTINCT(s.nama) FROM supplier s INNER JOIN barang b USING
(id_supplier) WHERE b.id_jenis_brg<>
(SELECT id_jenis_brg FROM jenis_barang WHERE nama_jenis_brg='Minuman');

Seringkali subquery menggunakan operator IN, EXISTS, ANY, SOME, NOT, ALL dan atau operator pembandingan (=, >, <, <>, >=, <=) yang digunakan untuk dihubungkan kedua perintah SQL.

IN dan NOT IN digunakan untuk membandingkan data pada suatu tabel terhadap data pada tabel yang lain.

Contoh :

1. Menampilkan barang-barang yang dibeli oleh pelanggan.
SELECT nama FROM barang WHERE id_barang IN
(SELECT id_barang FROM detail_transaksi);



Bandungkan perintah SQL di atas dengan kedua perintah SQL berikut ini :

- a. `SELECT DISTINCT(b.nama) FROM barang b JOIN detail_transaksi d USING(id_barang);`
 - b. `SELECT DISTINCT(b.nama) FROM barang b NATURAL JOIN detail_transaksi d;`
2. Menampilkan barang-barang yang belum pernah dibeli oleh pelanggan.
Masukan dahulu data barang berikut ini :
- ```
INSERT INTO barang VALUES
('B014','Green Tea','180','2450','2.0','1','2','S003'),
('B015','Sikat','11','1500','4.0','0.0','4','S006');
```

```
SELECT nama FROM barang WHERE id_barang NOT IN
(SELECT id_barang FROM detail_transaksi);
```

Bandungkan perintah SQL di atas dengan perintah SQL berikut ini :

```
SELECT DISTINCT(b.nama) FROM barang b LEFT OUTER JOIN detail_transaksi d
USING(id_barang) WHERE d.id_barang IS NULL;
```

**EXISTS dan NOT EXISTS** digunakan untuk mengecek atau mengetahui keberadaan suatu data dalam suatu tabel tertentu.

Contoh :

- a. Menampilkan barang-barang yang dibeli oleh pelanggan.  
`SELECT nama FROM barang WHERE EXISTS
(SELECT id_barang FROM detail_transaksi WHERE
barang.id_barang=detail_transaksi.id_barang);`
- b. Menampilkan barang-barang yang belum pernah dibeli oleh pelanggan.  
`SELECT nama FROM barang WHERE NOT EXISTS
(SELECT id_barang FROM detail_transaksi WHERE
barang.id_barang=detail_transaksi.id_barang);`

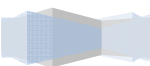
**ANY dan SOME** dapat digunakan bergantian. ANY dan SOME hampir sama dengan IN tetapi ANY dan SOME menggunakan operator perbandingan, sedangkan IN tidak.

Contoh :

- a. `SELECT nama FROM barang WHERE persen_laba > ANY
(SELECT b.persen_laba FROM barang b NATURAL JOIN jenis_barang j
WHERE nama_jenis_brg='elektronik');`
- b. `SELECT nama FROM barang WHERE persen_laba > SOME
(SELECT b.persen_laba FROM barang b NATURAL JOIN jenis_barang j
WHERE nama_jenis_brg='elektronik');`

Subquery di atas akan menghasilkan persen laba 7 dan 8. Perintah SQL di atas sama dengan perintah berikut ini :

```
SELECT nama FROM barang
WHERE persen_laba > 7 OR persen_laba > 8;
```



**ALL** digunakan untuk mengambil nilai dari perintah SQL utama yang cocok dengan semua nilai yang ada pada perintah Sub Query.

Contoh :

```
SELECT nama FROM barang WHERE persen_laba > ALL
(SELECT b.persen_laba FROM barang b NATURAL JOIN jenis_barang j
WHERE nama_jenis_brg='elektronik');
```

Subquery di atas akan menghasilkan persen laba 7 dan 8. Perintah SQL di atas sama dengan perintah berikut ini :

```
SELECT nama FROM barang
WHERE persen_laba > 7 AND persen_laba > 8;
```

- 6) Sub Query juga bisa dimasukkan di dalam suatu perintah SQL, bukan diluar seperti pada contoh-contoh sebelumnya. Subquery yang seperti ini akan diproses terlebih dahulu, baru kemudian perintah SQL utamanya.

Contoh :

- a. Menampilkan data kapan terakhir kali "charles" melakukan transaksi pembelian.

```
SELECT nama,id_pelanggan,
(SELECT MAX(tgl_transaksi) FROM transaksi
WHERE pelanggan.id_pelanggan=transaksi.id_pelanggan) 'Transaksi terakhir'
FROM pelanggan WHERE nama='charles';
```

- b. Menampilkan banyak transaksi yang pernah dilakukan oleh pelanggan (bandingkan dengan contoh pada modul ke-7 untuk kasus yang sama, yaitu GROUP BY).

```
SELECT nama,id_pelanggan,
(SELECT COUNT(id_pelanggan) FROM transaksi
WHERE pelanggan.id_pelanggan=transaksi.id_pelanggan) 'Banyak transaksi'
FROM pelanggan;
```

## 7) DISKUSI

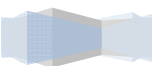
Buatlah perintah SQL untuk menampilkan :

1. Data pelanggan (nama, kode transaksi, tanggal transaksi, dan total pembayaran transaksinya) pada bulan Juli 2006.
2. Berapa rata-rata transaksi yang terjadi pada bulan Juli 2006?
3. Data pelanggan (nomor 1) yang transaksinya lebih kecil dari rata-rata transaksi pada bulan Juli 2006 (nomor 2).

## 9. View

Tujuan dari View adalah untuk :

- Menurunkan Network Traffic (beban Network).  
Menyimpan suatu perintah SQL (terutama yang kompleks) dimana perintah tersebut sering digunakan dan diakses.
- Keamanan: mencegah user untuk dapat mengakses suatu tabel sepenuhnya.  
Misal user dapat mengakses nama dan nomor telepon tetapi tidak bisa mengakses tanggal lahir dan gaji.



Sintak dari View adalah :

```
CREATE [OR REPLACE] [<algorithm attributes>]
VIEW [database.]< name> [(<columns>)]
AS <SELECT statement> [<check options>];
```

Contoh :

a) Membuat View :

```
CREATE VIEW pelanggan_simpati AS
SELECT nama,alamat,tgl_lahir,telepon
FROM pelanggan WHERE telepon REGEXP '^081[23]'
ORDER BY nama;
```

b) Cara mengaksesnya :

- SELECT \* FROM pelanggan\_simpati;
- SELECT nama,alamat FROM pelanggan\_simpati;

1) Algorithm Attributes

Terdiri dari tiga atribut, yaitu MERGE, TEMPTABLE, dan UNDEFINED dimana default-nya adalah UNDEFINED.

Amati dan diskusikan dengan teman anda perintah SQL berikut ini:

#### **Contoh MERGE**

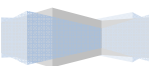
- CREATE ALGORITHM = MERGE VIEW email\_supplier AS  
SELECT nama,alamat,telepon,email  
FROM supplier WHERE email IS NOT NULL;
- SELECT \* FROM email\_supplier;
- UPDATE email\_supplier SET email='puma@puma.com'  
WHERE nama LIKE 'pungkas%';
- SELECT nama,email FROM supplier;
- INSERT INTO email\_supplier VALUES  
( 'Paman','Jl. Mahameru 10', '0812229999','paman@uny.ac.id');
- SELECT \* FROM supplier;

#### **Contoh TEMPTABLE**

- CREATE ALGORITHM = TEMPTABLE VIEW email\_supplier1 AS  
SELECT nama,alamat,telepon,email  
FROM supplier WHERE email IS NOT NULL;
- UPDATE email\_supplier1 SET email='indomandiri@gomandiri.org'  
WHERE nama LIKE '%mandiri';

#### **Contoh UNDEFINED**

- CREATE ALGORITHM = UNDEFINED VIEW email\_supplier2 AS  
SELECT nama,alamat,telepon,email  
FROM supplier WHERE email IS NOT NULL;
- UPDATE email\_supplier2 SET alamat='Jl. Muria No. 25'  
WHERE nama='Indo Mandiri';
- SELECT \* FROM email\_supplier2;





## 2) Columns

Digunakan untuk mengganti tampilan nama kolom pada view (nama kolom berbeda dengan nama kolom pada tabel asal).

Contoh :

- a) CREATE VIEW pelanggan\_simpati1 (name,address,birth,phone) AS  
SELECT nama,alamat,tgl\_lahir,telepon FROM pelanggan  
WHERE telepon REGEXP '^081[23]' ORDER BY nama;
- b) SELECT \* FROM pelanggan\_simpati1;
- c) CREATE VIEW pelanggan\_simpati4 AS  
SELECT nama 'name', alamat 'address', tgl\_lahir 'birth', telepon 'phone'  
FROM pelanggan WHERE telepon REGEXP '^081[23]'  
ORDER BY nama;
- d) SELECT \* FROM pelanggan\_simpati4;

## 3) Check Option

Check Option **hanya** dapat digunakan untuk Updating View. Tujuan dari Check View adalah untuk mengecek apakah perintah SQL yang memanggil View merupakan bagian dari View atau tidak (**sesuai dengan klausa WHERE pada View**), jika tidak maka perintah ditolak.

Ada dua pilihan yang dapat digunakan dalam Check Option, yaitu LOCAL dan CASCADING.

### a. Local

**WITH LOCAL CHECK OPTION** untuk menyakinkan bahwa data yang sedang di-update (INSERT, DELETE, UPDATE) merupakan bagian dari View itu sendiri, bukan bagian dari View lainnya.

### b. Cascading

**WITH CASCADED CHECK OPTION** merupakan kebalikan dari LOCAL, yaitu memeriksa juga apakah merupakan bagian dari View lainnya.

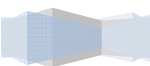
Jika dalam pembuatan View hanya ditulis **WITH CHECK OPTION**, maka artinya View menggunakan CASCADING karena DEFAULT dari Check Option adalah CASCADING. Tetapi jika dalam pembuatan View tidak menggunakan Check Option, maka View tersebut menggunakan LOCAL.

## 1) View tanpa Check Option

- a) CREATE VIEW pelanggan\_simpati2 AS  
SELECT nama,alamat,tgl\_lahir,telepon  
FROM pelanggan WHERE telepon REGEXP '^081[23]'  
ORDER BY nama;
- b) UPDATE pelanggan\_simpati2  
SET telepon='08175469672' WHERE nama='Merry';
- c) SELECT \* FROM pelanggan\_simpati2;

## 2) View dengan Check Option

- a) CREATE VIEW pelanggan\_simpati3 AS  
SELECT nama,alamat,tgl\_lahir,telepon FROM pelanggan  
WHERE telepon REGEXP '^081[23]'  
ORDER BY nama WITH CHECK OPTION;
- b) UPDATE pelanggan\_simpati3  
SET telepon='08175469718' WHERE nama='Bobby';



- 3) View dengan Local Check Option
  - a) CREATE ALGORITHM = MERGE VIEW pelanggan\_silver AS  
SELECT nama,jenis\_kelamin,jenis\_pelanggan  
FROM pelanggan WHERE jenis\_pelanggan='S';
  - b) SELECT \* FROM pelanggan\_silver;
  - c) CREATE ALGORITHM = MERGE VIEW pelanggan\_silver\_edit AS  
SELECT nama,jenis\_kelamin,jenis\_pelanggan  
FROM pelanggan\_silver WHERE jenis\_kelamin='L'  
WITH LOCAL CHECK OPTION;
  - d) SELECT \* FROM pelanggan\_silver\_edit;
  - e) UPDATE pelanggan\_silver\_edit  
SET jenis\_pelanggan='G' WHERE nama='Andi';
  - f) SELECT \* FROM pelanggan\_silver\_edit;
  - g) SELECT \* FROM pelanggan\_silver;
  
- 4) View dengan Cascading Check Option
  - a) CREATE ALGORITHM = MERGE VIEW pelanggan\_silver\_edit1 AS  
SELECT nama,jenis\_kelamin,jenis\_pelanggan  
FROM pelanggan\_silver WHERE jenis\_kelamin='L'  
WITH CASCADED CHECK OPTION;
  - b) SELECT \* FROM pelanggan\_silver\_edit1;
  - c) UPDATE pelanggan\_silver\_edit1  
SET jenis\_pelanggan='G' WHERE nama='Anton';

#### 4) Joining Table

Contoh :

```
CREATE VIEW pel_jumlah_beli AS
SELECT p.nama,t.id_pelanggan,COUNT(t.id_pelanggan) 'banyak_pembelian'
FROM pelanggan p NATURAL JOIN transaksi t
GROUP BY p.nama;
```

#### 5) Union Table

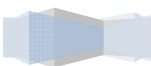
Digunakan untuk menggabungkan dua atau lebih View dengan syarat mempunyai field (kolom) yang sama.

Contoh :

- a) CREATE VIEW pelanggan\_xl AS  
SELECT nama,alamat,tgl\_lahir,telepon  
FROM pelanggan WHERE telepon REGEXP '^081[7-9]'  
ORDER BY nama;
- b) CREATE VIEW simpati\_xl AS  
SELECT \* FROM pelanggan\_simpati UNION  
SELECT \* FROM pelanggan\_xl;

#### 6) Managing View

- a) Melihat sintak View  
SHOW CREATE VIEW pelanggan\_xl;
- b) Melihat kolom dan tipe data dari View
  - DESCRIBE pelanggan\_xl;
  - DESC pelanggan\_xl;



## 7) Changing View

- a) ALTER VIEW pelanggan\_xl (name,address,birth,phone) AS  
SELECT nama,alamat,tgl\_lahir,telepon  
FROM pelanggan WHERE telepon REGEXP '^081[7-9]'  
ORDER BY nama;
- b) ALTER VIEW pelanggan\_xl AS SELECT nama,alamat,telepon  
FROM pelanggan WHERE telepon REGEXP '^081[7-9]'  
ORDER BY nama;

## 8) Removing View

Sintaknya adalah :

**DROP VIEW [IF EXISTS] [<database>.<name>**

Contoh :

DROP VIEW email\_supplier;

## 9) CONTROL FLOW FUNCTION

Merupakan fungsi untuk perbandingan, yaitu membandingkan kondisi suatu data. Ada tiga macam fungsi, yaitu if(), ifnull(), nullif(), dan case().

### 1) if() function

Sintaknya :

**IF(<expression1>, <expression2>, <expression3>)**

**<expression1> : untuk ekspresi perbandingan data (data yang ingin dicek)**

**<expression2> : bila perbandingan bernilai true**

**<expression3> : bila perbandingan bernilai false**

Contoh :

```
SELECT nama,IF(jenis_kelamin='L','Laki-laki','Perempuan') 'jenis kelamin'
FROM pelanggan;
```

### 2) ifnull()

Fungsi ifnull() digunakan untuk mengetahui atau mengevaluasi nilai data yang NULL.

Sintaknya :

**IFNULL(<expression1>, <expression2>)**

**<expression1> : untuk ekspresi perbandingan data (data yang ingin dicek)**

**<expression2> : bila perbandingan bernilai true**

Contoh :

- SELECT nama,IFNULL(telepon, 'belum punya telepon') 'telepon'  
FROM pelanggan;
- SELECT nama,IFNULL(email, ' belum punya email') 'email'  
FROM supplier;

### 3) nullif()

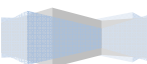
Fungsi nullif() akan menghasilkan nilai NULL jika ekspresi1 sama dengan ekspresi2. Jika tidak maka akan menghasilkan ekspresi1.

Sintaknya :

**NULLIF(<expression1>, <expression2>)**

Contoh :

- SELECT NULLIF(10\*20, 20\*10);
- SELECT NULLIF(10\*20, 20\*5);



4) case()

Sintaknya :

```
CASE WHEN <expression> THEN <result>
 [{WHEN <expression> THEN <result>}...]
 [ELSE <result>]
 END
```

Contoh :

```
SELECT nama,telepon,
CASE
 WHEN telepon REGEXP '081[23]' THEN 'Simpati'
 WHEN telepon REGEXP '081[56]' THEN 'Mentari'
 WHEN telepon REGEXP '081[7-9]' THEN 'XL'
 WHEN telepon REGEXP '085' THEN 'IM3'
END 'provider'
FROM pelanggan;
```

10) DISKUSI

Buatlah View untuk :

1. Menghitung total pembayaran tiap transaksi yang dilakukan oleh pelanggan.
2. Menggunakan View yang anda buat dari nomor 1, buat perintah SQL untuk mengetahui jumlah kupon undian yang diperoleh setiap pelanggan jika setiap kelipatan 10 ribu akan memperoleh kupon satu buah. Urutkan mulai dari yang memperoleh kupon paling banyak.
3. Menggunakan View yang anda buat dari nomor 1, buat perintah SQL untuk mengetahui hadiah apa yang diterima oleh pelanggan dimana jika mempunyai kupon dengan :
  - Tidak punya kupon maka dia tidak memperoleh hadiah.
  - 2 : Buku
  - 5 : Weker
  - 10 : Jam Dinding
  - 50 : Radio
  - 100 : Radio Tape

