

# **Modul Praktikum**

**Basis Data**



**Program Studi Sistem Komputer**

**STMIK STIKOM Indonesia**

## DAFTAR ISI

MODUL I PEMBUATAN DAN PEMELIHARAAN TABEL .....	3
MODUL II MANIPULASI DATA .....	11
MODUL III PENGAMBILAN DATA DARI BANYAK TABEL.....	17
MODUL IV FUNGSI-FUNGSI SQL .....	22
MODUL V SORTING DAN AGGREGATE .....	28
MODUL VI SUBQUERIES DAN SET OPERATION .....	33
MUDUL VII <i>VIEWS</i> DAN <i>CONTROL FLOW FUNCTION</i> .....	38
MUDUL VIII USER-DEFINED FUNCTIONS, <i>STORED PROCEDURE</i> , TRIGERS .....	45
MODUL IX CONCEPTUAL DATA MODEL DAN PHYSICAL DATA MODEL.....	53

## MODUL I

### PEMBUATAN DAN PEMELIHARAAN TABEL

#### Tujuan :

1. Mahasiswa dapat memahami perintah-perintah dalam Data Definition Language (DDL).
2. Mahasiswa dapat memahami perintah CREATE.
3. Mahasiswa dapat memahami perintah ALTER.
4. Mahasiswa dapat memahami perintah DROP.
5. Mahasiswa dapat memahami penggunaan Constraint.

#### Tugas Pendahuluan :

1. Apa Yang anda ketahui tentang Data Definition Language (DDL)?
2. Apa saja perintah-perintah yang tergolong dalam DDL?

#### DASAR TEORI

DDL atau Data Definition Language adalah bagian dari SQL yang digunakan untuk mendefinisikan data dan objek database. Apabila perintah ini digunakan, entri akan dibuat ke dalam kamus data dari SQL (Octaviani, 2010). Didalam kategori ini terdapat perintah-perintah sebagai berikut :

Tabel 1.1 DDL (Data Definition Language)

Perintah	Keterangan
CREATE DATABASE	Membuat Database
CREATE TABLE	Membuat tabel
ALTER TABLE	Mengubah atau menyisipkan kolom ke dalam tabel
DROP TABLE	Menghapus tabel dari database

#### Perintah CREATE

Sistem manajemen (DBMS) memungkinkan untuk membuat dan mengelola banyak database independent. Untuk membuat database berikut querinya.

```
CREATE DATABASE namadatabase
```

Nama database tidak boleh mengandung spasi dan tidak boleh memiliki nama yang sama antar database.

Sebelum membuat tabel dalam database pastikan terlebih dahulu database tempat anda membuat tabel sudah aktif, untuk mengaktifkan database yang anda buat dapat menggunakan statement **USE** nama\_database . Untuk membuat tabel berikut querinya.

```
CREATE TABLE nm_tabel
(nm_field1 tippedata1 [NOT NULL | NULL]
[{nm_field2 tippedata2 [NOT NULL | NULL]}...])
```

**nm\_tabel** adalah nama dari tabel yang akan di buat, sesuaikan nm\_tabel dengan entity yang diwakilinya. **nm\_field1**, **nm\_field2** adalah nama field yang harus ada dalam sebuah tabel yang mewakili element entity tersebut. Tiap field akan menampung data dengan tipe data tertentu yang ditunjukkan oleh tippedata1, tippedata2 dan seterusnya. Opsi berikutnya adalah constraint (syarat) masing-masing tabel apakah boleh kosong (NULL) atau harus diisi (NOT NULL). Nilai default dari constraint ini adalah NULL yang artinya field tersebut boleh kosong atau tidak diisi (Alam, 2005).

### Tipe Data

Tipe data adalah suatu bentuk pemodelan data yang dideklarasikan pada saat melakukan pembuatan tabel. Tipe data ini akan mempengaruhi setiap data yang akan dimasukkan ke dalam sebuah tabel. Data yang akan dimasukkan harus sesuai dengan tipe data yang dideklarasikan. Dalam MySQL terdapat beberapa tipe data, diantaranya:

Tabel 1.2 Tipe Data Numerik

Type Data	Keterangan
TINYINT	Bilangan bulat terkecil, dengan jangkauan untuk bilangan bertanda: -128 sampai dengan 127 dan untuk yang tidak bertanda : 0 s/d 255.
SMALLINT	Bilangan bulat dengan jangkauan untuk bilangan bertanda : -32768 s/d 32767 dan untuk yang tidak bertanda : 0 s/d 65535
MEDIUMINT	Bilangan bulat dengan jangkauan untuk bilangan bertanda : -8388608 s/ d 8388607 dan untuk yang tidak bertanda : 0 s/d 16777215
INT	Bilangan bulat dengan jangkauan untuk bilangan bertanda : -2147483648 s/d 2147483647 dan untuk yang tidak bertanda : 0 s/d 4294967295
BIGINT	Bilangan bulat terbesar dengan jangkauan untuk bilangan bertanda : -9223372036854775808 s/d 9223372036854775807 dan untuk yang tidak bertanda : 0 s/d 1844674473709551615
FLOAT, REAL, DOUBLE	Bilangan pecahan
DECIMAL, NUMERIC	Bilangan pecahan, misalnya DECIMAL(5,2) dapat digunakan untuk menyimpan bilangan -99,99 s/d 99,99

Tabel 2.3 Tipe Data Date and Time

Type Data	Keterangan
DATETIME	Kombinasi tanggal dan jam, dengan jangkauan dari '1000-01-01 00:00:00' s/d '9999-12-31 23:59:59'
DATE	Tanggal dengan jangkauan dari '1000-01-01' s/d '9999-12-31'
TIMESTAMP	Kombinasi tanggal dan jam, dengan jangkauan dari '1970-01-01 00:00:01' s/d '2038-01-19 03:14:07'
TIME	Waktu dengan jangkauan dari '-838:59:59' s/d '838:59:59'
YEAR	Data tahun antara 1901 s/d 2155

Tabel 2.4 Tipe Data String

Type Data	Keterangan
CHAR	Mampu menangani data hingga 255 karakter.
VARCHAR	Mampu menangani data hingga 255 karakter. Tipe data VARCHAR tidak mengharuskan untuk memasukkan data sepanjang yang telah ditentukan.
TINYBLOB, TINYTEXT	Mampu menangani data sampai $2^8-1$ data.
BLOB, TEXT	Type string yang mampu menangani data hingga $2^{16}-1$ .
MEDIUMBLOB, MEDIUMTEXT	Ukuran 16777215 byte. Mampu menyimpan data hingga $2^{24}-1$ .
LONGBLOB, LONGTEXT	Ukuran 4294967295 byte. Mampu menyimpan data hingga berukuran GIGA BYTE. Tipe data ini memiliki batas penyimpanan hingga $2^{32}-1$ .
ENUM('nilai1','nilai2',...,'nilaiN')	Ukuran 1 atau 2 byte. Tergantung jumlah nilai enumerasinya (maksimum 65535 nilai)
SET('nilai1','nilai2',...,'nilaiN')	1,2,3,4 atau 8 byte, tergantung jumlah anggota himpunan (maksimum 64 anggota)

### Constraints

Constraint merupakan batasan atau aturan yang ada pada tabel. MySQL menyediakan beberapa tipe constraint berikut:

1. NOT NULL merupakan suatu kolom yang mendefinisikan dengan constraint NOT NULL. Kolom yang berfungsi sebagai kunci primer (Primary Key) otomatis tidak boleh NULL.
2. UNIQUE mendefinisikan suatu kolom bersifat unik, artinya antara satu data dengan data lain namanya tidak boleh sama, misal alamat email.

3. PRIMARY KEY Constraint PRIMARY KEY membentuk key yang unik untuk suatu tabel.
4. FOREIGN KEY Constraint didefinisikan pada suatu kolom yang ada pada suatu tabel, dimana kolom tersebut juga dimiliki oleh tabel yang lain sebagai suatu PRIMARY KEY bisa digunakan untuk menghubungkan antara dua tabel.
5. CHECK constraint yang satu ini mendefinisikan sebuah kondisi untuk data agar dapat masuk dalam field artinya tiap pemasukan data atau editing terhadap data record, field yang dimasukkan akan selalu diperiksa apakah isinya ada diantara data-data yang dimasukkan, jika tidak ada maka SQL akan menampilkan pesan ERROR.
6. DEFAULT digunakan untuk mendefinisikan nilai default dari field yang mana ketika ada baris baru yang dimasukkan kedalam tabel nilai default dari field akan digunakan apabila tidak ada nilai yang diberikan padanya.

### Perintah ALTER

Setelah membuat tabel dalam database, dapat memodifikasi field pada tabel yang telah dibuat. Dengan perintah ALTER dapat membuat perubahan pada struktur tabel tanpa menghapus dan menciptakan. Query ALTER :

Query ini digunakan untuk menambah field pada tabel

```
ALTER TABLE namatabel ADD nama_field tipe_data (lebar)
```

Query ini digunakan untuk merubah field pada tabel

```
ALTER TABLE namatabel CHANGE COLUMN nama_field  
nama_field tipe_data (lebar)
```

Query ini digunakan untuk menghapus field pada tabel

```
ALTER TABLE namatabel DROP COLUMN nama_field.
```

### Perintah DROP

Perintah terakhir dari Data Definition Language, DROP memungkinkan untuk menghapus seluruh objek dalam database dari DBMS. Gunakan perintah ini dengan hati-hati, perlu diingat bahwa perintah DROP akan menghapus data keseluruhan struktur dari database. Querinya sebagai berikut.

Untuk menghapus database:

```
DROP DATABASE namadatabase
```

Untuk menghapus Tabel:

```
DROP TABLE namatabel
```

## KEGIATAN PRAKTIKUM

Untuk lebih memahami tentang DDL (Data Definition Language) maka buatlah database dengan nama “Toko” yang mempunyai dua tabel yaitu tabel “Barang” dan tabel “Pembelian”, isilah kedua tabel sesuai dengan ketentuan tabel dibawah ini !

### Create database

```
CREATE DATABASE toko
```

Pakai database yang sudah dibuat

```
USE toko
```

### Create table

Untuk membuat tabel Barang.

Tabel 2.5 Tabel Barang

Nama Field	Tipe data
ID_Barang	Varchar
Nama_Barang	Varchar
Tanggal_terima	Datetime
Stok_Barang	Int

```
CREATE TABLE Barang
(
  id_barang VARCHAR(50) NOT NULL,
  nama_barang VARCHAR(255) NOT NULL,
  tanggal_terima DATETIME NOT NULL,
  stok_barang INT NULL DEFAULT 0,
  CONSTRAINT pk_barang PRIMARY KEY (id_barang)
)
```

Hasilnya bisa anda lihat pada gambar dibawah ini !

<input type="checkbox"/>	<b>id_barang</b>	<b>nama_barang</b>	<b>tanggal_terima</b>	<b>stok_barang</b>
*	(NULL)	(NULL)	(NULL)	0

Gambar 1.1 Hasil dari Create tabel Barang.

Untuk membuat tabel Pembelian.

Tabel 1.6 Tabel Pembelian

Nama Field	Tipe Data
ID_Pembeli	Varchar
ID_Barang	Varchar
Tanggal_Beli	Datetime
Jumlah_Pembelian	Int
Nama_Pembelian	Varchar

```
CREATE TABLE pembelian
(
  id_pembeli VARCHAR(50) NOT NULL,
  id_barang VARCHAR(50) NULL ,
  tanggal_beli DATETIME NOT NULL,
  nama_pembeli VARCHAR (60) NOT NULL,
  jumlah_pembelian INT NULL ,
  CONSTRAINT pk_pembelian PRIMARY KEY (id_pembeli)
)
```

Hasilnya anda bisa lihat pada gambar dibawah ini!

<input type="checkbox"/>	<b>id_pembeli</b>	<b>id_barang</b>	<b>tanggal_beli</b>	<b>nama_pembeli</b>	<b>jumlah_pembelian</b>
*	(NULL)	(NULL)	(NULL)	(NULL)	(NULL)

Gambar 1.2 Hasil dari Create tabel Pembelian.

Karena belum menambahkan Foreign key pada saat membuat tabel Pembelian, maka dapat menambahkan Foreign key dengan perintah ALTER seperti queri dibawah ini.



```

ALTER TABLE pembelian
ADD CONSTRAINT
fk_pembelian_relation_barang FOREIGN
KEY (id_barang)
REFERENCES barang (id_barang)

```

Pada queri alter diatas, akan ditambahkan foreign key pada ID\_Barang menggunakan perintah ALTER TABLE. Selain itu juga bisa langsung memasukkan foreign key pada saat CREATE TABLE Pembelian.

### Alter

Contoh penggunaan Alter.

Menambahkan field “Alamat\_Pembeli” pada tabel Pembelian.

```

ALTER TABLE pembelian
ADD alamat_pembeli VARCHAR(70)

```

<input type="checkbox"/>	id_pembeli	id_barang	tanggal_beli	nama_pembeli	jumlah_pembelian	alamat_pembeli
*	(NULL)	(NULL)	(NULL)	(NULL)	(NULL)	(NULL)

Gambar 1.3 Add Alamat\_Pemeli pada tabel Pembelian.

Menghapus kolom nama pembeli pada tabel Pembelian.

```


ALTER TABLE pembelian
DROP COLUMN nama_pembeli

```

<input type="checkbox"/>	id_pembeli	id_barang	tanggal_beli	jumlah_pembelian	alamat_pembeli
*	(NULL)	(NULL)	(NULL)	(NULL)	(NULL)

Gambar 1.4 Drop kolom Nama\_Pembeli pada tabel Pembelian.

Merubah tipe data field (kolom) Nama\_Barang dari tipe data Varchar (255) menjadi tipe data Int pada tabel Barang, pada gambar dibawah ini merupakan gambar saat field Nama\_Barang belum dirubah tipe datanya.


	Field	Type
	id_barang	varchar(50) NOT NULL
	nama_barang	varchar(255) NOT NULL
	tanggal_terima	datetime NOT NULL
	stok_barang	int(11) NULL

Gambar 1.5 Tampilan sebelum merubah Nama Barang pada tabel Barang.

Untuk merubahnya tulislah queri dibawah ini!

```
ALTER TABLE barang  
CHANGE COLUMN nama_barang nama_barang INT not null
```

Setelah menuliskan queri diatas, maka anda bisa melihat hasilnya pada gambar dibawah ini!

	Field	Type
	id_barang	varchar(50) NOT NULL
	nama_barang	int(11) NOT NULL
	tanggal_terima	datetime NOT NULL
	stok_barang	int(11) NULL

Gambar 1.6 Tampilan sesudah merubah Nama Barang pada tabel Barang.

### Drop

Menghapus tabel Barang.

```
DROP TABLE barang
```

### TUGAS

1. Buatlah database baru dengan perintah SQL dengan nama “Mahasiswa” yang memiliki dua tabel yaitu tabel “Jurusan” dan tabel “Biodata”!
  - a. Isikan kedua tabel dengan field masing-masing dibawah ini!  
Tabel Jurusan : KodJur, NamaJur, Ketua\_jurusan  
Tabel Biodata : NoMhs, KodJur, Nama, Nilai, Alamat, JK
  - b. Tentukan tipe data masing-masing field
2. Modifikasi table tersebut dengan menambahkan Primary key serta Foreign Key nya agar kedua tabel saling berhubungan!

## MODUL II MANIPULASI DATA

### Tujuan :

1. Mahasiswa dapat melakukan manipulasi data yang tersimpan dalam basis data.
2. Mahasiswa dapat memahami perintah INSERT.
3. Mahasiswa dapat memahami perintah UPDATE.
4. Mahasiswa dapat memahami perintah DELETE.

### Tugas Pendahuluan :

1. Apa yang anda ketahui tentang DML?
2. Sebutkan perintah-perintah SQL yang tergolong dalam DML dan jelaskan perbedaan antar DML dan DDL!
3. Apa yang anda ketahui tentang perintah INSERT, UPDATE, dan DELETE?
4. Sebutkan macam-macam klausa dan operator pada perintah SELECT!

### DASAR TEORI

DML (Data Manipulation Language) adalah bahasa yang memungkinkan pengguna mengakses atau memanipulasi data yang berbentuk suatu model data tertentu (Silberschatz,2011). Tipe akses data adalah (Silberschatz,2011):

1. Pengambilan informasi yang disimpan dalam basis data.
2. Penempatan informasi baru dalam basis data.
3. Penghapusan informasi dari basis data.
4. Modifikasi informasi yang disimpan dalam basis data.

Adapun perintah-perintah umum dalam DML yang disediakan dalam MySQL adalah sebagai berikut:

1. **INSERT**, menyisipkan atau menambahkan data (*tuple*) baru ke dalam tabel.
2. **SELECT**, mengambil atau menampilkan data dari tabel. Perintah ini adalah perintah dasar yang digunakan untuk mengambil informasi dari basis data
3. **UPDATE**, memperbaharui data yang lama ke dalam data yang baru.
4. **DELETE**, menghapus data dalam tabel.

Untuk lebih memahami perintah-perintah DML (Data Manipulation Language), berikut ini adalah penjelasan dalam praktikum untuk masing-masing dari perintah-perintah di atas.

## KEGIATAN PRAKTIKUM

### INSERT

Sebelum memulai praktikum, pastikan bahwa struktur basis data yang digunakan sudah ada sebelumnya. Struktur basis data yang digunakan disini adalah struktur basis data yang dibuat dalam MODUL II sebelum dikenakan perintah ALTER. Setelah tabel dibuat beserta constraint-nya jika diperlukan, maka tabel siap digunakan untuk menampung data. Perintah pada MySQL untuk memasukkan data ke dalam tabel adalah perintah INSERT. Masukkan data pada tabel barang dan tabel pembelian. Bentuk perintah INSERT adalah sebagai berikut :

```
INSERT INTO [tabel]
([field1],[field2],[field3],...)
VALUES ([value1],[value2],[value3],...);
```

Memasukkan data pada tabel Barang

```
INSERT INTO barang
(id_barang,nama_barang,tanggal_terima)
VALUES ('BRG01','Sony','2015-04-03');
```

Untuk memasukkan beberapa data sekaligus, dapat menggunakan perintah sebagai berikut:

```
INSERT INTO barang
(id_barang,nama_barang,tanggal_terima)
VALUES ('BRG01','Sony','2015-04-03'),
('BRG02','Samsung','2015-05-03');
```

Dan data seterusnya silahkan masukkan sendiri dengan petunjuk perintah diatas berdasarkan data pada Gambar 2.1.

<input type="checkbox"/>	id_barang	nama_barang	tanggal_terima	stok_barang
<input type="checkbox"/>	BRG01	Sony	2015-04-03 00:00:00	0
<input type="checkbox"/>	BRG02	Samsung	2015-05-03 00:00:00	0
<input type="checkbox"/>	BRG03	HTC	2015-03-08 00:00:00	0
<input type="checkbox"/>	BRG04	Microsoft	2015-03-10 00:00:00	0
<input type="checkbox"/>	BRG05	Motorola	2015-03-15 00:00:00	0
<input type="checkbox"/>	BRG06	Xiaomi	2015-03-17 00:00:00	0
*	(NULL)	(NULL)	(NULL)	0

Gambar 2.1 Hasil Penambahan data pada tabel barang

Kolom yang tidak disebutkan pada pernyataan INSERT secara otomatis akan diisi dengan NULL. Pada Stok\_Barang nilainya akan terisi 0 karena pada saat membuat tabel Barang memberikan tipe data int, default value untuk int adalah 0.

Memasukkan data pada tabel pembelian.

```
INSERT INTO pembelian
(id_pembeli, id_barang, tanggal_beli, nama_pembeli,
jumlah_pembelian)
VALUES ('P01', 'BRG03', '2015-04-07', 'Mahadewi Istirani', 2);
```

Dan data seterusnya silahkan masukkan sendiri dengan petunjuk perintah diatas berdasarkan data pada Gambar 2.2.

<input type="checkbox"/>	id_pembeli	id_barang	tanggal_beli	nama_pembeli	jumlah_pembelian
<input type="checkbox"/>	P01	BRG03	2015-03-04 00:00:00	Putu	2
<input type="checkbox"/>	P02	BRG02	2015-04-06 00:00:00	Made	2
<input type="checkbox"/>	P03	BRG06	2015-05-08 00:00:00	Komang	2
<input type="checkbox"/>	P04	BRG06	2015-06-10 00:00:00	Ketut	1
*	(NULL)	(NULL)	(NULL)	(NULL)	(NULL)

Gambar 2.2 Hasil penambahan data pada tabel pembelian

Memasukkan data dari tabel lain juga dimungkinkan dalam perintah insert. Untuk mempraktekannya buatlah satu tabel lagi pada basis data “toko” dengan nama tabel “pelanggan” yang memiliki *field* ID\_Pelanggan dan Nama\_Pelanggan. Untuk membuat tabel pelanggan gunakan perintah dibawah ini:

```
CREATE TABLE pelanggan
(id_pelanggan VARCHAR(10),
nama_pelanggan VARCHAR(80));
```

Setelah tabel pelanggan dibuat, kemudian jalankan perintah dibawah ini untuk mengisi tabel “Pelanggan” dengan data dari tabel pembelian.

```
INSERT INTO pelanggan
(id_pelanggan, nama_pelanggan)
SELECT id_pembeli, nama_pembeli FROM pembelian;
```

Untuk memeriksa kebenarannya, gunakan perintah dibawah ini dan lihat hasilnya!

```
SELECT * FROM pelanggan;
```

<input type="checkbox"/>	id_pelanggan	nama_pelanggan
<input type="checkbox"/>	P01	Putu
<input type="checkbox"/>	P02	Made
<input type="checkbox"/>	P03	Komang
<input type="checkbox"/>	P04	Ketut
*	(NULL)	(NULL)

Gambar 2.3 Hasil dari penambahan pada tabel Pelanggan.

## UPDATE

Perintah UPDATE digunakan untuk memodifikasi data pada tabel dalam basis data.

```
UPDATE [tabel]
SET [field] = '[value baru]'
WHERE [field acuan] = '[value]';
```

Keterangan :

SET : Untuk menentukan kolom yang akan diubah dan isi penggantinya.

WHERE : Menentukan kondisi atau syarat dari baris-baris yang akan diganti.

Berikut ini perintah untuk meng-*update field* nama barang pada tabel barang dengan nama “LG” yang awalnya bernama “HTC”.

```
UPDATE barang
SET nama_barang='LG'
WHERE id_barang='BRG03';
```

<input type="checkbox"/>	^ id_barang	nama_barang	tanggal_terima	stok_barang
<input type="checkbox"/>	BRG01	Sony	2015-04-03 00:00:00	0
<input type="checkbox"/>	BRG02	HTC	0000-00-00 00:00:00	0
<input type="checkbox"/>	BRG03	LG	2015-03-08 00:00:00	0
<input type="checkbox"/>	BRG04	Microsoft	2015-03-10 00:00:00	0
<input type="checkbox"/>	BRG05	Motorola	2015-03-15 00:00:00	0
<input type="checkbox"/>	BRG06	Xiaomi	2015-03-17 00:00:00	0
<input type="checkbox"/>	* (NULL)	(NULL)	(NULL)	0

Gambar 2.4 Hasil perintah UPDATE pada tabel barang

## DELETE

Pernyataan DELETE digunakan untuk menghapus baris pada tabel, bentuk perintahnya sebagai berikut.

```
DELETE FROM [tabel]
WHERE [field] = '[values]';
```

Pada contoh ini akan menghapus *record* dengan ID\_Barang = 'BRG01' pada tabel pembelian. Untuk mempraktekkannya tulislah perintah dibawah ini!

```
DELETE FROM pembelian
WHERE id_barang = 'BRG03';
```

<input type="checkbox"/> id_pembeli	id_barang	tanggal_beli	nama_pembeli	jumlah_pembelian
<input type="checkbox"/> P02	BRG02	2015-04-06 00:00:00	Made	2
<input type="checkbox"/> P03	BRG06	2015-05-08 00:00:00	Komang	2
<input type="checkbox"/> P04	BRG06	2015-06-10 00:00:00	Ketut	1
* (NULL)	(NULL)	(NULL)	(NULL)	(NULL)

Gambar 2.5 Hasil dari perintah DELETE pada tabel pembelian.

Dalam perintah DELETE jika ingin menghapus semua data pada tabel tanpa menghapus tabel, maka klausa WHERE tidak perlu disebutkan, berikut perintahnya.

```
DELETE FROM [tabel];
atau
DELETE * FROM [tabel];
atau
TRUNCATE [tabel];
```

Untuk mengambil, mengubah, atau menghapus data, dapat digunakan where untuk mencari beberapa record saja sesuai kriteria yang ditetapkan. Misal

```
SELECT * FROM barang WHERE id_barang='BRG03';
```

Untuk mendapatkan data yang memenuhi beber nilai untuk sebuah kriteria, dapat digunakan in

```
SELECT * FROM barang WHERE id_barang IN ('BRG04', 'BRG05');
```

Untuk mendapatkan data dengan kriteria tidak sama persis, digunakan like, contohnya:

```
SELECT * FROM pelanggan WHERE nama_pembeli LIKE %tu%;
```

Tanda persen (%) dapat diartikan sebagai apa saja, jadi dengan menulis '%tu%' berarti akan mencari semua yang mengandung 'tu' apapun huruf di depan dan belakangnya. Namun, jika ditulis '%tu' maka akan dicari hanya yang berakhiran dengan 'tu'.

**TUGAS**

1. Isilah tabel jurusan dan tabel biodata yang anda buat dengan ketentuan sebagai berikut:

Misalkan isi tabel jurusan:

Kode Jurusan	Nama Jurusan	Ketua Jurusan
KD01	Sistem Informasi	Harnaningrum,S.Si
KD02	Teknik Informatika	EnnySela,S.Kom.,M.Kom
KD03	Tekhnik Komputer	Berta Bednar,S.Si,M.T.

Isi tabel biodata:

No Mahasiswa	Kode Jurusan	Nama Mahasiswa	Alamat	IPK
210089	KD01	Rina Gunawan	Denpasar	3
210090	KD03	Gani Suprpto	Singaraja	3.5
210012	KD02	Alexandra	Nusa dua	3
210099	KD02	Nadine	Gianyar	3.2
210002	KD01	Rizal Samurai	Denpasar	3.7

2. Masukkan data baru pada tabel biodata dengan kode jurusan “KD04”!
3. Dengan menggunakan perintah UPDATE, cobalah merubah isi tabel jurusan dan biodata pada basis data mahasiswa yang anda buat dengan ketentuan sebagai berikut :
  - Mengganti nama mahasiswa pada tabel biodata “Rina Gunawan” menjadi “Rina Gunawan Astuti”!
  - Mengganti kode jurusan pada tabel jurusan “KD01” menjadi “KM01”!
  - Mengganti no mahasiswa pada tabel biodata “210089” menjadi “210098”!
  - Mengganti nilai pada tabel biodata “3” menjadi “3.3”!
  - Mengganti kode jurusan pada tabel biodata “KD03” menjadi “KD05”!
4. Buatlah kesimpulan mengenai soal no.2 dan no.3!



## MODUL III

### PENGAMBILAN DATA DARI BANYAK TABEL

#### Tujuan :

1. Mahasiswa memahami perintah-perintah SQL untuk pengambilan data dari banyak tabel.
2. Mahasiswa mampu memahami tipe-tipe join.
3. Mahasiswa mampu memahami tentang cartesian product.
4. Mahasiswa mampu memahami tentang penggabungan tabel.

#### Tugas Pendahuluan :

1. Apa yang anda ketahui tentang JOIN?
2. Sebutkan macam-macam JOIN !
3. Apakah perbedaan Left JOIN dengan Right JOIN?

### DASAR TEORI

#### Tipe-tipe join

Di dalam database, ada kalanya membutuhkan data dari beberapa tabel yang saling berhubungan. Untuk mendapatkan data dari beberapa tabel tersebut dapat digunakan perintah JOIN pada perintah SQL (Alam, 2005). Fungsi dari operator ini adalah untuk menggabungkan data-data dari dua buah tabel atau lebih. Operator JOIN ini berlaku pada tabel biasa ataupun VIEW. Untuk mengambil data dari kombinasi dua tabel atau lebih bisa menggunakan JOIN. MySQL menyediakan fasilitas penggabungan dan JOIN tabel antara lain [5]:

- *Cartesian product* (CROSS JOIN)
- INNER JOIN
- LEFT JOIN (LEFT OUTER JOIN)
- RIGHT JOIN (RIGHT OUTER JOIN)

#### a. Cartesian Products

*Cartesian product* atau disebut pula sebagai *cross join* akan menciptakan hasil yang didasarkan pada semua kemungkinan kombinasi *tuple* dalam kedua set data. Dengan demikian, jumlah *tuple* yang dikembalikan adalah  $N \times M$ , dimana  $N$  adalah jumlah *tuple* dalam tabel A dan  $M$  jumlah *tuple* dalam tabel B. Bentuk perintah dari CROSS JOIN pada MySQL adalah :

```
SELECT [field1], [field2], [...] FROM [tabel_A]
      CROSS JOIN [tabel_B];
atau
SELECT [field1], [field2], [...] FROM [tabel_A], [tabel_B];
```

**b. INNER JOIN**

INNER JOIN ini merupakan perintah JOIN yang paling umum yang dapat digunakan pada semua basis data. Penggabungan hanya dilakukan pada dua buah tabel yang telah merelasikan *field*-nya. Maksudnya adalah data pada tabel pertama akan dihubungkan dengan data pada tabel kedua apabila nilai *field* yang dijadikan patokan relasi kedua tabel memiliki nilai yang sama.

Perbedaan antara EQUI JOIN dan INNER JOIN terletak pada tuple yang dikembalikan (*return*). EQUI JOIN mengembalikan tuple yang memiliki isi yang sama pada *field* tertentu. Sedangkan dua atau lebih tabel yang digabungkan pada INNER JOIN hanya mengembalikan tuple yang memenuhi syarat penggabungan. Bentuk baku perintah INNER JOIN :

```
SELECT [field1], [field2], [...], [fieldn]
FROM [tabel_A]
INNER JOIN [tabel_B]
ON [tabel_A].[field key] = [tabel_B].[field_key];
```

**c. LEFT JOIN**

LEFT JOIN (*left outer join*) akan menampilkan data pada tabel kiri walaupun tidak memiliki relasi pada tabel di bagian kanan, bentuk perintahnya sebagai berikut:

```
SELECT [field1], [field2], [...], [fieldn]
FROM [tabel_A]
LEFT JOIN [tabel_B]
ON [tabel_A].[field key] = [tabel_B].[field_key];
```

Atau

```
SELECT [field1], [field2], [...], [fieldn]
FROM [tabel_A]
LEFT OUTER JOIN [tabel_B]
ON [tabel_A].[field key] = [tabel_B].[field_key];
```

**d. RIGHT JOIN**

RIGHT JOIN (*right outer join*) akan menampilkan data pada tabel disebelah kanan walaupun tidak mempunyai data yang berhubungan dengan tabel disebelah kirinya. Bentuk perintahnya sebagai berikut:

```
SELECT [field1], [field2], [...], [fieldn]
FROM [tabel_A]
RIGHT JOIN [tabel_B]
ON [tabel_A].[field key] = [tabel_B].[field_key];
```

Atau

```
SELECT [field1], [field2], [...], [fieldn]
FROM [tabel_A]
RIGHT OUTER JOIN [tabel_B]
ON [tabel_A].[field key] = [tabel_B].[field_key];
```

## KEGIATAN PRAKTIKUM

Agar lebih memperdalam pemahaman tentang fungsi-fungsi MySQL diatas, silahkan coba perintah dibawah ini pada basis data “toko” yang anda buat.

### a. Cartesian Products

Untuk mendapatkan hasil *cartesian product* dapat digunakan 2 perintah berikut:

```
SELECT barang.ID_Barang, pembelian.ID_Pembeli,
       pembelian>Nama_Pembeli, barang>Nama_Barang
FROM barang
CROSS JOIN pembelian
```

Atau

```
SELECT barang.id_barang, pembelian.id_pembeli,
       pembelian.nama_pembeli, barang.nama_barang
FROM barang, pembelian
```

Hasil dari kedua perintah tersebut sama-sama *cartesian product*. Dapat diperhatikan pula bahwa CROSS JOIN tidak memerlukan perintah ON, karena tidak diperlukan pencocokan *field*. Gambar 4.1 merupakan hasil kedua perintah diatas.

<input type="checkbox"/> id_barang	id_pembeli	nama_pembeli	nama_barang
<input type="checkbox"/> BRG01	P02	Made	Sony
<input type="checkbox"/> BRG01	P03	Komang	Sony
<input type="checkbox"/> BRG01	P04	Ketut	Sony
<input type="checkbox"/> BRG02	P02	Made	HTC
<input type="checkbox"/> BRG02	P03	Komang	HTC
<input type="checkbox"/> BRG02	P04	Ketut	HTC
<input type="checkbox"/> BRG03	P02	Made	LG
<input type="checkbox"/> BRG03	P03	Komang	LG
<input type="checkbox"/> BRG03	P04	Ketut	LG
<input type="checkbox"/> BRG04	P02	Made	Microsoft
<input type="checkbox"/> BRG04	P03	Komang	Microsoft
<input type="checkbox"/> BRG04	P04	Ketut	Microsoft
<input type="checkbox"/> BRG05	P02	Made	Motorola
<input type="checkbox"/> BRG05	P03	Komang	Motorola
<input type="checkbox"/> BRG05	P04	Ketut	Motorola
<input type="checkbox"/> BRG06	P02	Made	Xiaomi
<input type="checkbox"/> BRG06	P03	Komang	Xiaomi
<input type="checkbox"/> BRG06	P04	Ketut	Xiaomi

Gambar 3.1 Hasil dari CROSS JOIN

**b. INNER JOIN**

```

SELECT barang.id_barang, pembelian.id_pembeli,
       pembelian.nama_pembeli, barang.nama_barang
FROM barang
INNER JOIN pembelian
      ON barang.id_barang = pembelian.id_barang

```

<input type="checkbox"/> id_barang	id_pembeli	nama_pembeli	nama_barang
<input type="checkbox"/> BRG02	P02	Made	HTC
<input type="checkbox"/> BRG06	P03	Komang	Xiaomi
<input type="checkbox"/> BRG06	P04	Ketut	Xiaomi

Gambar 3.2 Hasil INNER JOIN pada tabel pembelian

**c. LEFT JOIN**

```

SELECT barang.id_barang, pembelian.id_pembeli,
       pembelian.nama_pembeli, barang.nama_barang
FROM barang
LEFT JOIN pembelian
      ON barang.id_barang = pembelian.id_barang

```

<input type="checkbox"/> id_barang	id_pembeli	nama_pembeli	nama_barang
<input type="checkbox"/> BRG02	P02	Made	HTC
<input type="checkbox"/> BRG06	P03	Komang	Xiaomi
<input type="checkbox"/> BRG06	P04	Ketut	Xiaomi
<input type="checkbox"/> BRG01	(NULL)	(NULL)	Sony
<input type="checkbox"/> BRG03	(NULL)	(NULL)	LG
<input type="checkbox"/> BRG04	(NULL)	(NULL)	Microsoft
<input type="checkbox"/> BRG05	(NULL)	(NULL)	Motorola

Gambar 3.3 Hasil LEFT JOIN pada tabel pembelian

**d. RIGHT JOIN**

```

SELECT barang.id_barang, pembelian.id_pembeli,
       pembelian.nama_pembeli, barang.nama_barang
FROM barang
RIGHT JOIN pembelian
      ON barang.id_barang = pembelian.id_barang

```

<input type="checkbox"/> id_barang	id_pembeli	nama_pembeli	nama_barang
<input type="checkbox"/> BRG02	P02	Made	HTC
<input type="checkbox"/> BRG06	P03	Komang	Xiaomi
<input type="checkbox"/> BRG06	P04	Ketut	Xiaomi

Gambar 3.4 RIGHT JOIN pada tabel pembelian

**TUGAS**

1. Cobalah masing-masing perintah join yang sudah anda praktikkan dengan menggunakan data base Mahasiswa yang anda buat!
2. Buatlah kesimpulan tentang perbedaan klausa antaran Right join dan Left join!

## MODUL IV FUNGSI-FUNGSI SQL

### Tujuan:

1. Mahasiswa dapat memahami fungsi-fungsi SQL.
2. Mahasiswa dapat memahami single row functions dalam SQL.
3. Mahasiswa dapat memahami character functions dalam SQL.

### Tugas Pendahuluan:

1. Apa yang anda ketahui tentang fungsi-fungsi SQL?
2. Sebutkan macam-macam dari fungsi SQL!
3. Apa saja yang tergolong dalam single row function?

## DASAR TEORI

### Single row functions

Secara garis besar function dibagi menjadi dua bagian yaitu: Single row functions dan group function, perbedaan kedua fungsi ini yaitu pada Single row functions memproses satu baris data pada satu proses dan memberikan satu output untuk setiap satu baris data masukan.

Salah satu contoh single-row functions adalah UPPER yang berfungsi mengubah data input menjadi huruf kapital. Sedangkan group function memproses multi-row data pada saat bersamaan dan memberikan satu output.

Contoh group function adalah SUM untuk menghitung nilai total. Namun yang akan dibahas adalah Single row function. Berdasarkan tipe data yang diproses, single-row function dibagi menjadi lima jenis, yaitu:

1. **Character Function** digunakan untuk memproses data karakter.
2. **Numeric Function** digunakan untuk memproses data numerik.
3. **Date Function** digunakan untuk memproses data tanggal.
4. **Convension Function** digunakan untuk melakukan konversi data.
5. **General Function** merupakan function yang bisa digunakan untuk memproses semua.

### Character functions

Fungsi karakter menerima input berupa karakter dan mengembalikan nilai yang bisa berupa karakter maupun angka. Beberapa contoh penggunaan dalam character function.

LOWER : Menjadikan huruf kecil.

Queri :

```
SELECT LOWER ('character')
```

**UPPER** : Menjadikan huruf kapital.

Queri :

```
SELECT UPPER ('character')
```

**SUBSTRING** : Mengambil karakter mulai dari posisi m sebanyak n, jika n tidak dituliskan, maka semua karakter mulai posisi m sampai terakhir akan diambil.

Queri :

```
SELECT SUBSTRING ('character',  
nilai_character_awal, Jumlah_karakter) FROM  
nm_tabel WHERE nm_filed='character'
```

**LTRIM** : Digunakan untuk menghilangkan spasi kosong disebelah kiri string didalam kurung.

Queri :

```
SELECT LTRIM (character)
```

**RTRIM** : Digunakan untuk menghilangkan spasi kosong disebelah kanan string didalam kurung.

Queri :

```
SELECT RTRIM (character)
```

**RIGHT** : Fungsi ini akan mengembalikan nilai string yang berasal dari sebelah kanan string dengan jumlah yang telah ditentukan.

Queri :

```
SELECT RIGHT ( String, jumlah karakter)
```

**LEFT** : Fungsi ini mengembalikan string sepanjang (png) karakter dari sebelah kiri strings.

Queri :

```
SELECT LEFT ( String, jumlah karakter)
```

**CHAR** : Digunakan untuk mengkonversi kode ASCII menjadi karakter.

Queri :

```
SELECT CHAR (Expresi Integer)
```

**LENGTH** : Fungsi ini mengembalikan nilai integer panjang string x termasuk spasi kosong.

Queri :

```
SELECT LENGTH (character)
```

**REVERSE** : Fungsi ini digunakan untuk melakukan pembalikan string yang disertakan.

Queri :

```
SELECT REVERSE (character)
```

**SPACE** : Fungsi ini akan memberikan spasi sejumlah yang telah ditetapkan.

Queri :

```
SELECT SPACE (Jumlah spasi)
```

**KEGIATAN PRAKTIKUM**

Untuk lebih memahami fungsi-fungsi SQL, tuliskan statement-statement berikut pada database Toko yang anda buat!

**Pernyataan Single row functions**

Contoh

**1. LOWER**

Merubah nama barang menjadi huruf kecil semua.

```
SELECT LOWER (nama_barang) AS huruf_kecil  
FROM barang
```

<input type="checkbox"/>	huruf_kecil
<input type="checkbox"/>	sony
<input type="checkbox"/>	htc
<input type="checkbox"/>	lg
<input type="checkbox"/>	microsoft
<input type="checkbox"/>	motorola
<input type="checkbox"/>	xiaomi

Gambar 4.1 Contoh Penggunaan Lower.

**2. UPPER**

Merubah nama barang menjadi huruf besar semua.

```
SELECT UPPER (nama_barang) AS huruf_besar  
FROM barang
```

<input type="checkbox"/>	huruf_besar
<input type="checkbox"/>	SONY
<input type="checkbox"/>	HTC
<input type="checkbox"/>	LG
<input type="checkbox"/>	MICROSOFT
<input type="checkbox"/>	MOTOROLA
<input type="checkbox"/>	XIAOMI

Gambar 4.2 Contoh Penggunaan Upper.

**3. SUBSTRING**

Mengambil nama barang yang dimulai dari huruf kedua sebanyak lima huruf.

```
SELECT SUBSTRING (nama_barang, 2, 5) AS ambil_karakter  
FROM barang
```



<input type="checkbox"/>	ambil_karakter
<input type="checkbox"/>	ony
<input type="checkbox"/>	TC
<input type="checkbox"/>	G
<input type="checkbox"/>	icros
<input type="checkbox"/>	otoro
<input type="checkbox"/>	iaomi

Gambar 4.3 Contoh Penggunaan Substring.

## 4. LTRIM

Menghilangkan spasi disebelah kiri.

```
SELECT LTRIM('   Samsung') AS hapus_spasi;
```

<input type="checkbox"/>	hapus_spasi
<input type="checkbox"/>	Samsung

Gambar 4.4 Contoh Penggunaan LTRIM.

## 5. RTRIM

Menghilangkan spasi disebelah kanan.

```
SELECT RTRIM('MOTOROLA   ') AS hapus_spasi;
```

<input type="checkbox"/>	hapus_spasi
<input type="checkbox"/>	MOTOROLA

Gambar 4.5 Contoh Penggunaan RTRIM.

## 6. RIGHT

Mengambil sebanyak 3 karakter nama barang dari sebelah kanan.

```
SELECT RIGHT(nama_barang,3) AS ambil_karakter
FROM barang;
```

<input type="checkbox"/>	ambil_karakter
<input type="checkbox"/>	ony
<input type="checkbox"/>	HTC
<input type="checkbox"/>	LG
<input type="checkbox"/>	oft
<input type="checkbox"/>	ola
<input type="checkbox"/>	omi

Gambar 4.6 Contoh Penggunaan RIGHT.

## 7. LEFT

Mengambil sebanyak 3 karakter nama barang dari sebelah kiri.

```
SELECT LEFT(nama_barang, 3) AS ambil_karakter
FROM barang;
```

<input type="checkbox"/>	ambil_karakter
<input type="checkbox"/>	Son
<input type="checkbox"/>	HTC
<input type="checkbox"/>	LG
<input type="checkbox"/>	Mic
<input type="checkbox"/>	Mot
<input type="checkbox"/>	Xia

Gambar 4.7 Contoh Penggunaan LEFT.

## 8. LENGTH

Menghitung banyak karakter pembentuk nama barang termasuk spasinya.

```
SELECT nama_barang, LENGTH(nama_barang) AS panjang
FROM barang;
```

<input type="checkbox"/>	nama_barang	panjang
<input type="checkbox"/>	Sony	4
<input type="checkbox"/>	HTC	3
<input type="checkbox"/>	LG	2
<input type="checkbox"/>	Microsoft	9
<input type="checkbox"/>	Motorola	8
<input type="checkbox"/>	Xiaomi	6

Gambar 4.8 Contoh Penggunaan LENGTH.

## 9. REVERSE

Membalik nama barang.

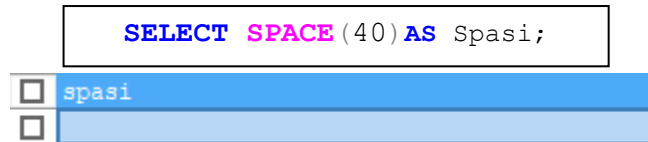
```
SELECT nama_barang, REVERSE(nama_barang) AS balik
FROM barang;
```

<input type="checkbox"/>	nama_barang	balik
<input type="checkbox"/>	Sony	ynoS
<input type="checkbox"/>	HTC	CTH
<input type="checkbox"/>	LG	GL
<input type="checkbox"/>	Microsoft	tfosorciM
<input type="checkbox"/>	Motorola	alorotoM
<input type="checkbox"/>	Xiaomi	imoaiX

Gambar 4.9 Contoh Penggunaan REVERSE.

## 10. SPACE

Memberikan spasi sebanyak 40 spasi.



Gambar 4.10 Contoh Penggunaan SPACE.

## TUGAS

1. Buatlah perintah SQL yang memanfaatkan single row function (Masing-masing 1)!

## MODUL V

### SORTING DAN AGGREGATE

#### Tujuan :

1. Mahasiswa mampu memahami suatu fungsi pengurutan data dengan klausa ORDER BY.
2. Mahasiswa mampu memahami pengambilan data dari basis data menggunakan fungsi-fungsi aggregate.
3. Mahasiswa mampu memahami fungsi-fungsi aggregate.
4. Mahasiswa mampu memahami klausa GROUP BY.
5. Mahasiswa mampu memahami klausa HAVING.

#### Tugas Pendahuluan :

1. Apa yang anda ketahui tentang Fungsi Aggregate?
2. Sebutkan pembagian fungsi-fungsi aggregate!

### DASAR TEORI

#### Sorting data

Pada SQL sorting digunakan untuk mengurutkan data, terdapat dua macam metode untuk mengurutkan data, yaitu :

1. ASC digunakan untuk urutan naik yang secara default digunakan.
2. DESC digunakan untuk mengurutkan data secara menurun.

#### Klausa ORDER BY

Penggunaan klausa Order By harus diikuti dengan ASC atau DESC karena klausa ORDER BY ini digunakan untuk mengurutkan data, jika ingin menampilkan data dalam tabel berdasarkan kriteria yang ditentukan, query nya sebagai berikut (Octaviani, 2010):

```
SELECT * FROM namatabel  
  
ORDER BY namakolom ekspresi (ASC/DESC)
```

#### Fungsi-Fungsi Aggregate

Fungsi *aggregate* merupakan fungsi yang mengambil set atau multi set nilai sebagai input dan menghasilkan nilai tunggal sebagai output (Silberschatz, 2011). Berikut ini merupakan fungsi-fungsi Aggregate yang tersedia dalam SQL:

## 1. AVG

Untuk menghitung nilai rata-rata dari suatu kolom tertentu yang telah definisikan dalam perintah select. Perintah:

```
SELECT AVG ([field]) FROM [tabel];
```

## 2. COUNT

Untuk menghitung jumlah baris dalam sebuah tabel. Perintah:

```
SELECT COUNT ([field]) FROM [tabel];
```

## 3. SUM

Untuk menjumlahkan suatu kolom tertentu yang telah didefinisikan dalam perintah **SELECT**.

Perintah:

```
SELECT SUM ([field]) FROM [tabel];
```

## 4. MAX

Untuk mengetahui nilai terbesar dari sebuah kolom tertentu dalam perintah **SELECT**.

Perintah:

```
SELECT MAX ([field]) FROM [tabel];
```

## 5. MIN

Untuk mengetahui nilai terkecil dari sebuah kolom tertentu dalam perintah select. Perintah:

```
SELECT MIN ([field]) FROM [tabel];
```

**GROUP BY**

Dalam SQL terdapat banyak kata kunci yang digunakan untuk melengkapi statement **SELECT** untuk memperoleh tampilan data yang dikehendaki, diantaranya yaitu **GROUP BY**. **GROUP BY** ini merupakan kata kunci yang digunakan untuk mengelompokkan satu atau lebih *field* yang memiliki nilai yang sama untuk membentuk satu kelompok.

Perintah:

```
SELECT [field], [aggregate] ([field]) FROM [tabel]
WHERE [field] [operator] [value]
GROUP BY [field];
```

**HAVING**

Kata kunci ini tidak termasuk fungsi, tetapi kata kunci ini berfungsi untuk melengkapi statement **SELECT**. Kegunaannya adalah mendefinisikan sebuah kondisi yang kemudian diterapkan pada sekelompok data pada beberapa *field* yang kemudian ditampilkan sebagai hasil perintah.

Perintah:

```

SELECT [field], [aggregate] ([field])
FROM [tabel]
WHERE [field] [operator] [value]
GROUP BY [field]
HAVING [aggregate] ([field]) [operator] [value];

```

## KEGIATAN PRAKTIKUM

### Pernyataan ORDER BY

Menampilkan data pada tabel Barang dengan urutan ASC atau terurut secara abjad.

```

SELECT * FROM barang ORDER BY nama_barang ASC;

```

<input type="checkbox"/> id_barang	nama_barang	tanggal_terima	stok_barang
<input type="checkbox"/> BRG02	HTC	0000-00-00 00:00:00	0
<input type="checkbox"/> BRG03	LG	2015-03-08 00:00:00	0
<input type="checkbox"/> BRG04	Microsoft	2015-03-10 00:00:00	0
<input type="checkbox"/> BRG05	Motorola	2015-03-15 00:00:00	0
<input type="checkbox"/> BRG01	Sony	2015-04-03 00:00:00	0
<input type="checkbox"/> BRG06	Xiaomi	2015-03-17 00:00:00	0

Gambar 5.1 Contoh Penggunaan Order By (asc) pada tabel Barang.

Menampilkan data pada tabel Barang dengan urutan DESC atau terurut secara abjad terbalik.

```

SELECT * FROM barang ORDER BY nama_barang DESC;

```

<input type="checkbox"/> id_barang	nama_barang	tanggal_terima	stok_barang
<input type="checkbox"/> BRG06	Xiaomi	2015-03-17 00:00:00	0
<input type="checkbox"/> BRG01	Sony	2015-04-03 00:00:00	0
<input type="checkbox"/> BRG05	Motorola	2015-03-15 00:00:00	0
<input type="checkbox"/> BRG04	Microsoft	2015-03-10 00:00:00	0
<input type="checkbox"/> BRG03	LG	2015-03-08 00:00:00	0
<input type="checkbox"/> BRG02	HTC	0000-00-00 00:00:00	0

Gambar 5.2 Contoh Penggunaan Order By (desc) pada tabel Barang.

Fungsi-fungsi SQL seperti Character function bisa dipadukan dengan query Order By, seperti contoh-contoh dibawah ini.

Untuk lebih memahami fungsi-fungsi *aggregate*, cobalah perintah-perintah MySQL dibawah ini pada basis data “Toko” yang anda buat!

**AVG**

Menampilkan nilai rata-rata Jumlah pembelian pada tabel Pembelian.

```
SELECT AVG(jumlah_pembelian) AS 'nilai_rata'  
FROM pembelian;
```

	nilai_rata
	1.6667

Gambar 5.3 Hasil AVG pada tabel pembelian

**COUNT**

Menjumlahkan jumlah data pada tabel Barang.

```
SELECT COUNT(*) AS 'jumlah_data'  
FROM barang;
```

	jumlah_data
	6

Gambar 5.4 Hasil COUNT pada tabel barang

**SUM**

Menjumlahkan jumlah pembelian pada tabel Pembelian.

```
SELECT SUM(jumlah_pembelian) AS 'total_pembelian'  
FROM pembelian;
```

	total_pembelian
	5

Gambar 5.5 Hasil SUM pada tabel pembelian

**MAX**

Menampilkan pembelian paling banyak pada tabel pembelian.

```
SELECT MAX(jumlah_pembelian) AS 'max_pembelian'  
FROM pembelian;
```

	max_pembelian
	2

Gambar 5.6 Hasil MAX pada tabel pembelian

**MIN**

Menampilkan pembelian paling sedikit pada tabel pembelian.

```
SELECT MIN(jumlah_pembelian) AS 'min_pembelian'  
FROM pembelian;
```

	min_pembelian
	1

Gambar 5.7 Hasil MIN pada tabel pembelian

## GROUP BY

Menampilkan nama barang serta jumlahnya berdasarkan nama barang pada tabel barang.

```
SELECT nama_barang, COUNT(nama_barang) AS jumlah  
FROM barang GROUP BY nama_barang
```

<input type="checkbox"/>	nama_barang	jumlah
<input type="checkbox"/>	HTC	1
<input type="checkbox"/>	LG	1
<input type="checkbox"/>	Microsoft	1
<input type="checkbox"/>	Motorola	1
<input type="checkbox"/>	Sony	1
<input type="checkbox"/>	Xiaomi	1

Gambar 5.8 Hasil GROUP BY pada tabel barang

## HAVING

Menampilkan nama pembeli yang memiliki rata-rata pembelian lebih dari satu.

```
SELECT nama_pembeli, AVG(jumlah_pembelian) AS rata  
FROM pembelian GROUP BY id_pembeli  
HAVING AVG(jumlah_pembelian) > 1
```

<input type="checkbox"/>	nama_pembeli	rata
<input type="checkbox"/>	Made	2.0000
<input type="checkbox"/>	Komang	2.0000

Gambar 5.9 Hasil HAVING pada tabel barang

## TUGAS

1. Buatlah perintah SQL yang menggunakan fungsi Aggregate (Masing-masing 1)!
2. Buatlah perintah SQL dengan klausa Order By, Group By, dan Having!



## MODUL VI

### SUBQUERIES DAN SET OPERATION

#### Tujuan :

1. Mahasiswa dapat memahami dan membuat subqueri serta penggunaan set operation dalam SQL.
2. Memahami tentang subqueries dan dapat menerapkan subqueries.
3. Memahami penggunaan operator UNION.
4. Memahami penggunaan operator INTERSECT.
5. Memahami penggunaan operator EXCEPT.

#### Tugas Pendahuluan

1. Apa yang anda ketahui tentang subqueries?
2. Apa perbedaan antara subqueri dengan queri biasa?

#### DASAR MATERI

##### Subqueries

*Subqueries* merupakan perintah SQL SELECT-FROM-WHERE yang terdapat dalam perintah SQL lain (Silberschatz,2011). Perintah ini digunakan untuk mengambil data dari lebih dari satu tabel. *Subqueri* biasanya terdiri dari dua atau lebih perintah SQL. Perintah SQL yang pertama disebut dengan perintah SQL utama dan perintah SQL yang lain disebut *subqueri*. Untuk lebih jelasnya mengenai *subqueri* berikut perintahnya:

```
SELECT [field_A] FROM [tabel_A]
      WHERE [expresi operator]
            (SELECT [field_B] FROM [tabel_B]);
```

##### Set Operation

Set operation pada SQL merupakan implementasi dari operasi matematika *union* ( $\cup$ ), *intersect* ( $\cap$ ), dan *except* ( $-$ ) (Silberschatz,2011). Pada MySQL, hanya fasilitas operator UNION yang disediakan. Untuk mendapatkan hasil INTERSECT dan EXCEPT pada MySQL, dapat menggunakan kombinasi fungsi UNION dengan fungsi lainnya.

##### UNION

Operator UNION digunakan untuk mendapatkan tabel (himpunan) gabungan dari dua buah tabel yang ada. Bila dilakukan penggabungan dua tabel maka yang didapatkan adalah semua *field* dari kedua tabel tersebut. Perintahnya sebagai berikut:

```
SELECT [field_A] FROM [tabel_A]
      UNION SELECT [field_B] FROM [tabel_B];
```

Misalkan pada basis data "toko" mempunyai tabel bernama "barang" dan "barang1" dengan nama *field* kedua tabel adalah sama yaitu ID\_Barang, Nama\_barang, Tanggal\_terima, Stok\_Barang, maka untuk menampilkan semua data pada tabel "barang" dan "barang1" hanya menggunakan perintah berikut:

```
SELECT Nama FROM barang
      UNION SELECT Nama FROM barang1;
```

## INTERSECT

INTERSECT merupakan operator yang digunakan untuk memperoleh data dari dua buah perintah dimana data yang ditampilkan adalah yang memenuhi kedua perintah tersebut dengan ketentuan jumlah, nama, dan tipe kolom dari masing-masing tabel yang akan ditampilkan datanya harus sama. Perintahnya sebagai berikut:

```
SELECT [field1], [field2], .., [field n] FROM [table]
      INTERSECT SELECT [field1], [field2], .., [field n];
```

## EXCEPT

EXCEPT merupakan operator yang memiliki fungsi untuk mengembalikan sekumpulan nilai yang ada di tabel pertama, tetapi tidak ada pada tabel kedua (Alam, 2005). Perintahnya sebagai berikut:

```
SELECT [field1], [field2], .., [field n] FROM [table]
      EXCEPT SELECT [field1], [field2], .., [field n]
      FROM [table];
```

## KEGIATAN PRAKTIKUM

Untuk meningkatkan pengetahuan tentang *subquery* dan *set operation*, cobalah perintah-perintah dibawah ini pada database toko yang anda buat.

### Subquery

Menampilkan ID\_Barang, Tanggal\_Beli, Nama\_Pembeli, dan Jumlah\_Pembelian dimana jumlah pembeliannya yang terbanyak pada tabel pembelian.

```
SELECT id_barang, tanggal_beli, nama_pembeli, jumlah_pembelian
FROM pembelian
WHERE jumlah_pembelian = (SELECT MAX(jumlah_pembelian)
FROM pembelian);
```

<input type="checkbox"/>	id_barang	tanggal_beli	nama_pembeli	jumlah_pembelian
<input type="checkbox"/>	BRG02	2015-04-06 00:00:00	Made	2
<input type="checkbox"/>	BRG06	2015-05-08 00:00:00	Komang	2

Gambar 6.1 Hasil *subquery* pada tabel pembelian

Menampilkan data pada tabel pembelian dimana jumlah pembeliannya yang paling sedikit.

```
SELECT * FROM pembelian
WHERE jumlah_pembelian = (SELECT MIN(jumlah_pembelian)
FROM pembelian);
```

<input type="checkbox"/>	id_pembeli	id_barang	tanggal_beli	nama_pembeli	jumlah_pembelian
<input type="checkbox"/>	P04	BRG06	2015-06-10 00:00:00	Ketut	1

Gambar 7.2 Hasil *subquery* pada tabel pembelian.

Menampilkan Nama\_Barang, Tanggal\_terima, dan Stok\_Barang pada tabel barang dengan Jumlah pembelian lebih dari satu berdasarkan tabel pembelian.

```
SELECT nama_barang, tanggal_terima, stok_barang
FROM barang Where id_barang
IN (SELECT id_barang
FROM pembelian WHERE jumlah_pembelian >= 1);
```

<input type="checkbox"/>	nama_barang	tanggal_terima	stok_barang
<input type="checkbox"/>	HTC	0000-00-00 00:00:00	0
<input type="checkbox"/>	Xiaomi	2015-03-17 00:00:00	0

Gambar 6.3 Hasil *subquery* pada tabel barang

## UNION

Menampilkan Nama\_Pembeli, dan Jumlah\_Pembelian pada tabel pembelian dan Nama\_Barang, Stok\_Barang pada tabel barang.

```
SELECT nama_pembeli, jumlah_pembelian
FROM pembelian
UNION
SELECT nama_barang, stok_barang
FROM barang;
```

nama_pembeli	jumlah_pembelian
Made	2
Komang	2
Ketut	1
Sony	0
HTC	0
LG	0
Microsoft	0
Motorola	0
Xiaomi	0

Gambar 6.4 Hasil UNION pada tabel pembelian dan tabel barang

Menampilkan Nama\_Pembeli, dan Jumlah\_Pembelian pada tabel pembelian dan Nama\_Barang, Stok\_Barang pada tabel barang.

```

SELECT nama_pembeli, jumlah_pembelian
      FROM pembelian
UNION ALL
SELECT nama_barang, stok_barang
      FROM barang;

```

nama_pembeli	jumlah_pembelian
Made	2
Komang	2
Ketut	1
Sony	0
HTC	0
LG	0
Microsoft	0
Motorola	0
Xiaomi	0

Gambar 6.5 Hasil UNION ALL pada tabel pembelian dan tabel barang

Menampilkan Tanggal\_Beli, Nama\_Pembeli, dan Jumlah\_Pembelian pada tabel pembelian dimana Jumlah\_Pembelian = 1 dan menampilkan Tanggal\_Beli, Nama\_Pembeli, Jumlah\_Pembelian pada tabel pembelian berdasarkan Nama\_Pembeli yang mengandung huruf 'ma'.

```

SELECT tanggal_beli, nama_pembeli, jumlah_pembelian
      FROM pembelian WHERE jumlah_pembelian='1'
UNION ALL
SELECT tanggal_beli, nama_pembeli, jumlah_pembelian
      FROM pembelian WHERE nama_pembeli LIKE '%ma%';

```

<input type="checkbox"/>	tanggal_beli	nama_pembeli	jumlah_pembelian
<input type="checkbox"/>	2015-06-10 00:00:00	Ketut	1
<input type="checkbox"/>	2015-04-06 00:00:00	Made	2
<input type="checkbox"/>	2015-05-08 00:00:00	Komang	2

Gambar 6.6 Hasil UNION ALL pada taebel pembelian

**TUGAS**

1. Buatlah perintah MySQL masing-masing pada basis data mahasiswa dengan menggunakan *subqueries* dan set operation!
2. Buatlah perintah MySQL yang menghasilkan *output* INTERSECT dan EXCEPT dengan menggunakan fungsi UNION!
3. Perhatikan *output*-nya dan buatlah kesimpulan mengenai perbedaan antara perintah masing-masing!

## MUDUL VII

### VIEWS DAN CONTROL FLOW FUNCTION

#### Tujuan :

1. Mahasiswa dapat memahami konsep serta penggunaan *view* dalam basis data
2. Mahasiswa dapat memahami *control flow function*.
3. Mahasiswa dapat memahami penerapan konsep *view* dalam SQL.
4. Mahasiswa dapat memahami pembuatan *view*.
5. Mahasiswa dapat memahami modifikasi *view*.
6. Mahasiswa dapat memahami penggunaan variabel dalam fungsi SQL.
7. Mahasiswa dapat memahami penggunaan *control flow function* MySQL.
8. Mahasiswa dapat memahami penggunaan tabel *temporary*.

#### Tugas Pendahuluan :

1. Apa yang anda ketahui tentang *View*?
2. Apa yang anda ketahui tentang *control flow function* MySQL?

### DASAR TEORI

#### Konsep View

*View* merupakan tabel yang tidak tampak, hanya di memori saja. *View* sering digunakan oleh para pengembang basis data untuk mempermudah menampilkan data dengan kriteria tertentu yang diambil dari satu atau beberapa tabel sekaligus. Selain menampilkan hasil, *view* juga dapat digunakan sebagai sumber data saat perintah dijalankan. *View* juga berguna untuk membatasi akses basis data, membuat perintah kompleks secara mudah, mengizinkan independensi data dan untuk menampilkan bentuk data yang berbeda dari data yang sama. *View* juga disebut sebagai relasi virtual (Silberschatz, 2011).

#### Pembuatan view

Untuk membuat *view* gunakan perintah dibawah ini:

```
CREATE VIEW [nm_view] ([daftar_field])  
AS [statement_select];
```

#### Modifikasi *view*

Sama seperti halnya objek basis data, *view* juga dapat diedit dengan perintah ALTER VIEW. Perintahnya sebagai berikut.

```
ALTER VIEW [nm_view] AS
SELECT [field1], [field2], [...], [fieldn]
FROM [tabel1], [tabel2], [...], [tabeln]
WHERE [table1]([key]) = [table2]([key]);
```

Sedangkan untuk menghapus *view* yang sudah tidak diperlukan, dapat menggunakan perintah sebagai berikut:

```
DROP VIEW [nm_view];
```

### User-Defined Variables

Seperti bahasa pemrograman, pada MySQL juga terdapat variabel yang dapat digunakan dalam perintahnya. *User-Defined Variables* merupakan variabel *session-specific*. Dengan kata lain, *user-defined variables* yang dideklarasikan oleh salah satu client tidak dapat dilihat oleh client lainnya. Semua variabel yang diberikan ke client akan hilang apabila client tersebut keluar.

```
SET
@ [variabel] = [value] ,
@ [variabel] = [value] ,
[...],
@ [variabeln] = [value] ;
```

### Control Flow Function

MySQL memiliki statement-statement untuk mengatur jalannya alur program. Statement-statement tersebut adalah sebagai berikut :

#### IF

Ini adalah statement pengendali alur program yang paling umum dan sangat sering digunakan pada bahasa-bahasa pemrograman lainnya. Bentuk IF yang paling sederhana adalah sebagai berikut:

```
SELECT IF ([kondisi],
          [Statement Benar],
          [Statement Salah]);
```

Pada contoh perintah dibawah ini, jika kondisi yang ada setelah IF bernilai False maka statement yang ada dibawahnya akan dihindari atau tidak dilaksanakan.

```
SELECT IF (2 = 4,  
          'Benar',  
          'Salah');
```

Hasilnya yang dicetak adalah Salah, karena 2 tidak sama dengan 4.

### IFNULL

IFNULL menggunakan dua ekspresi dimana jika ekspresi pertama tidak dalam kondisi NULL maka MySQL akan mengembalikan ekspresi pertama. Jika ekspresi pertama dalam kondisi NULL, maka ekspresi kedua yang akan dikembalikan.

```
SELECT IFNULL  
      ([ekspresi1], [ekspresi2]);
```

### NULLIF

NULLIF mengembalikan nilai NULL saat ekspresi pertama sama dengan ekspresi kedua. Apabila ekspresi tidak sama, maka NULLIF mengembalikan ekspresi pertama.

```
SELECT NULLIF  
      ([ekspresi1], [ekspresi2]);
```

### CASE

Pernyataan CASE merupakan alternatif lain untuk IF dengan kondisi yang bertingkat. Berikut bentuk perintahnya:

```
SELECT CASE  
      WHEN [kondisi] THEN [ekspresi]  
      WHEN [kondisi] THEN [ekspresi]  
      [...]   
      ELSE [ekspresi]  
END;
```

### Temporary table

Tabel *temporary* merupakan tabel sementara yang terbentuk ketika perintah dieksekusi. Fungsi tabel *temporary* digunakan untuk menyimpan data dari rangkaian proses untuk memperoleh informasi yang diinginkan. Pada umumnya tabel *temporary* diciptakan karena proses yang dilakukan tidak bisa diselesaikan dalam satu kali eksekusi perintah.



Keuntungan penggunaan tabel *temporary* adalah tidak membebani besarnya *file*, keuntungan lainnya adalah tabel *temporary* dapat digunakan untuk proses secara bersamaan pada waktu yang sama dengan pengguna yang berbeda-beda atau lebih mudahnya tabel *temporary* sangat membantu untuk aplikasi program berbasis *client server* tanpa perlu membuat tabel bantu pada masing-masing *client*. Berikut perintahnya:

```
CREATE TEMPORARY TABLE [nama tabel]
([nama field] [tipe data] ([ukuran]),
..);
```

Untuk menghapus tabel *temporary* dapat digunakan perintah DROP diikuti dengan nama tabelnya.

```
DROP [nama_temp_tabel];
```

Tabel *temporary* dapat dikategorikan menjadi :

a. *Local Temporary Table*

*Local temporary table* hanya berlaku didalam sebuah *stored procedure*. Apabila sebuah *stored procedure* dieksekusi, maka secara otomatis tabel akan dibuat dan dibuang apabila *stored procedure* tersebut telah selesai dieksekusi.

b. *Global Temporary Table*

Berbeda dengan *local temporary table*, *global temporary table* ini dapat diakses kapan saja selama tabel *temporary* ini belum dibuang.

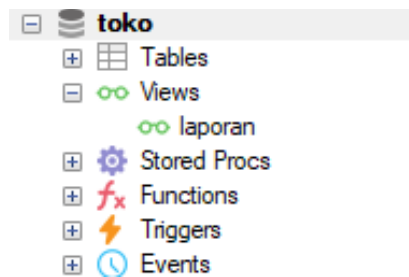
## PRAKTIKUM

### View

Untuk lebih memahami tentang penggunaan *View* maka akan membuat *View* pada databse “Toko”, buatlah *view* dengan nama “Laporan”, tulislah perintah dibawah ini!

```
CREATE VIEW laporan
AS SELECT pembelian.id_pembeli, pembelian.id_barang,
barang.nama_barang, pembelian.nama_pembeli,
pembelian.tanggal_beli, pembelian.jumlah_pembelian
FROM pembelian, barang
WHERE pembelian.id_barang=barang.id_barang;
```

Apabila pembuatan *View* ini sukses, maka pada navigator anda akan melihat tampilan yang sama seperti gambar dibawah ini.



Gambar 7.1 Tampilan navigator *view* Laporan pada basis data Toko

Untuk dapat menggunakan *view* yang telah dibuat dengancara sebagai berikut.

```
SELECT * FROM laporan;
```

<input type="checkbox"/>	id_pembeli	id_barang	nama_barang	nama_pembeli	tanggal_beli	jumlah_pembelian
<input type="checkbox"/>	P02	BRG02	HTC	Made	2015-04-06 00:00:00	2
<input type="checkbox"/>	P03	BRG06	Xiaomi	Komang	2015-05-08 00:00:00	2
<input type="checkbox"/>	P04	BRG06	Xiaomi	Ketut	2015-06-10 00:00:00	1

Gambar 7.2 Hasil *view* Laporan pada tabel pembelian

### DROP View

Untuk menghapus *view* Laporan guankan perintah dibawah ini!

```
DROP VIEW laporan;
```

### User-Defined Variables

Untuk lebih memahai tentang penggunaan *user-defined variables*, tulislsh perintah dibawah ini!

```
SET @t1=1, @t2=2, @t3:=4;
SELECT @t1, @t2, @t3, @t4 := @t1+@t2+@t3;
```

Bila anda eksekusi perintah diatas maka hasil yang anda dapatkan tampak seperti gambar dibawah ini.

<input type="checkbox"/>	@t1	@t2	@t3	@t4 := @t1+@t2+@t3
<input type="checkbox"/>	1	2	4	7

Gambar 7.3 Hasil *user-defined variables*

### Control Flow Function

Untuk memahami konsep *control flow function*, mari mempraktikkan contoh penggunaannya satu persatu pada praktikum kali ini.

#### a. IF

```
SELECT IF (2 = 4, 'Benar', 'Salah');
```

Hasil yang akan dicetak adalah Salah, karena 2 tidak sama dengan 4.

<input type="checkbox"/>	IF (2 = 4, 'Benar', 'Salah')
<input type="checkbox"/>	Salah

Gambar 7.5 Contoh Penggunaan IF

**b. IFNULL**

Contoh penggunaan IFNULL, coba anda tuliskan perintah dibawah ini !

```
SELECT IFNULL
  ((SELECT nama_barang FROM barang
    WHERE id_barang = 'a'),
  'Tidak Ditemukan') AS hasil;
```

Hasilnya dapat anda lihat pada gambar dibawah ini.

<input type="checkbox"/>	hasil
<input type="checkbox"/>	Tidak Ditemukan

Gambar 7.6 Contoh Penggunaan IFNULL

**c. NULLIF**

Penggunaan perintah NULLIF dibawah menggunakan perintah yang hampir sama dengan contoh IFNULL. Perhatikan perbedaannya.

```
SELECT NULLIF
  ((SELECT nama_barang FROM barang
    WHERE id_barang = 'BRG01'),
  'Sony') AS hasil;
```

<input type="checkbox"/>	hasil
<input type="checkbox"/>	(NULL)

Gambar 7.7 Hasil dari NULLIF

**d. CASE**

Contoh penggunaan CASE.

```
SELECT id_barang , jumlah_pembelian,
CASE
  WHEN jumlah_pembelian=1 THEN 'Cuma Satu'
  WHEN jumlah_pembelian>1 THEN 'Banyak'
  ELSE 'Kosong'
END AS jum
FROM pembelian
```

Hasilnya anda bisa lihat pada gambar dibawah ini.

<input type="checkbox"/> id_barang	jumlah_pembelian	jum
<input type="checkbox"/> BRG02		2 Banyak
<input type="checkbox"/> BRG06		2 Banyak
<input type="checkbox"/> BRG06		1 Cuma Satu

Gambar 7.9 contoh penggunaan CASE.

### Tabel Temporary

Untuk lebih memahami tentang pembuatan tabel *temporary*, silahkan mencoba perintah dibawah ini!

```
CREATE TEMPORARY TABLE templatihan (
    id INT PRIMARY KEY,
    nama VARCHAR(80),
    alamat VARCHAR(50),
    hobby VARCHAR(70)
);
```

Setelah tabel *temporary* terbentuk, anda dapat memperlakukan tabel tersebut seperti tabel biasa.

Untuk mengetahui apakah tabel sudah terbentuk, dapat digunakan perintah SELECT seperti berikut:

```
SELECT * FROM templatihan;
```

<input type="checkbox"/> ID	Nama	Alamat	Hobby
-----------------------------	------	--------	-------

Gambar 7.10 Hasil dari tabel *temporary*

### TUGAS

1. Sebutkan fungsi dari *View*!
2. Buatlah *view* pada basis data “Mahasiswa” yang anda buat!
3. Buatlah *control flow function* dan tabel *temporary* pada basis data “Mahasiswa” dengan menggunakan ekspresi berbentuk perintah DML SELECT!

## MUDUL VIII

### USER-DEFINED FUNCTIONS, *STORED PROCEDURE*, TRIGGERS

#### Tujuan :

1. Mahasiswa memahami lebih mendalam mengenai fungsi programmable object .
2. Mahasiswa memahami penggunaan user-defined functions dalam MySQL.
3. Mahasiswa memahami penggunaan *stored procedures* dalam MySQL.
4. Mahasiswa memahami penggunaan triggers dalam MySQL.

#### Tugas Pendahuluan :

1. Apa yang anda ketahui tentang *stored procedures*?
2. Apa yang anda ketahui tentang triggers?

### DASAR TEORI

#### User-defined functions

Merupakan suatu program yang terdiri dari sekumpulan perintah yang tersimpan sebagai suatu objek didalam basis data dengan pengambilan nilai. Pada User Defined Function (UDF) ini MySQL mengizinkan pengguna membuat fungsinya sendiri seperti *stored prosedure*. Untuk membuat sebuah fungsi adalah sebagai berikut:

```
CREATE FUNCTION [nama_fungsi] ([parameter] [tipe data])  
RETURNS [tipe data return]  
BEGIN  
    [perintah deklarasi]  
    [perintah eksekusi]  
END;
```

Menghapus fungsi:

```
DROP FUNCTION ([nama_fungsi]);
```

Kata kunci RETURNS mendefinisikan tipe data yang akan menampung hasil atau nilai yang akan dikembalikan oleh fungsi ke sistem. Sedangkan RETURN akan mengembalikan hasil kerja fungsi kepada sistem. Statement-statement yang diperbolehkan dalam fungsi adalah sebagai berikut.

- SET
- WHILE
- IF
- DECLARE
- SELECT

- INSERT
- UPDATE
- DELETE

Untuk melakukan pemanggilan pada fungsi yang dibuat bisa menggunakan perintah dibawah ini:

```
SELECT [nama fungsi] ([value]);
```

Terdapat perbedaan antara *stored procedure* dengan fungsi adalah *stored procedure* tidak mengembalikan nilai, sedangkan fungsi harus mengembalikan nilai. Fungsi dan *stored procedure* dapat digunakan dalam perintah MySQL.

### Stored Procedures

*Stored procedure* merupakan sekumpulan statement yang disusun sedemikian rupa untuk menjalankan tugas tertentu. *Stored procedure* digunakan untuk mempermudah pengolahan data dengan mendekatkan kode program dengan datanya. *Stored procedure* biasanya disimpan dalam sebuah nama jadi dapat diprekompilasi (Kadir, 2002).

Adapun keuntungan-keuntungan dari penggunaan *stored procedure* adalah sebagai berikut :

- Sebuah *stored procedure* dikompilasi dan lebih cepat dalam mengeksekusi batch atau perintah.
- Memproses data lewat *stored procedure* dilakukan pada server sehingga mengurangi intensitas lalu lintas data network
- *Stored procedure* menawarkan pemrograman modular hal ini karena sebuah *stored procedure* dapat memanggil *stored procedure* yang lain.
- *Stored procedure* bisa jadi adalah salah satu komponen penting dalam keamanan database. Jika semua akses user melalui *stored procedure* maka semua akses langsung ke tabel dan data dapat dikontrol.

Pada *stored procedure* dikenal dua macam tipe *stored procedure* yaitu *stored procedure* yang sudah ada pada sistem dan *stored procedure* yang dibuat oleh user. Perintah untuk membuat *stored procedure* adalah sebagai berikut.

```
CREATE PROCEDURE [nama_procedure] ([parameter] [tipe data])  
BEGIN  
    [perintah deklarasi]  
    [perintah eksekusi]  
END;
```

Untuk menjalankan *stored procedure* dapat memanggilnya dengan perintah:

```
CALL [nama procedure];
```

Pada *stored procedure* terdapat istilah parameter, parameter ini dapat melewatkan data untuk diolah oleh *stored procedure* tersebut. Ini menjadikan *stored procedure* sangat fleksibel (Kadir, 2002).

### Triggers

*Trigger* merupakan sebuah mekanisme kerja yang dipanggil ketika ada sebuah aksi yang terjadi pada sebuah tabel. Aksi yang dikenali pada *trigger* dapat berupa statement DML bisa seperti INSERT, UPDATE, dan DELETE atau statement DDL. Biasanya yang dieksekusi oleh *trigger* adalah *stored procedure*.

1. Membuat sebuah *trigger* dapat menggunakan perintah CREATE TRIGGER, bentuk perintahnya adalah sebagai berikut:

```
CREATE TRIGGER [nm_trigger]
    [trigger_time] [trigger_event]
ON [nm_tabel]/[nm_view] FOR EACH ROW
[isi trigger]
```

2. Memperbaiki trigger atau memodifikasi trigger yang ada, maka anda akan dapat menggunakan perintah ALTER TRIGGER.
3. Menghapus trigger yang tidak diperlukan, dapat menggunakan perintah DROP TRIGGER.

```
DROP TRIGGER [nm_trigger]
```

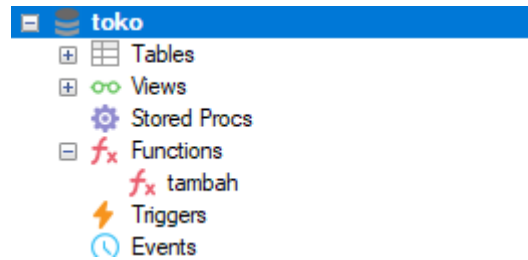
## PRAKTIKUM

Praktikkan *user-defined functions*, *stored procedures* dan pembuatan *trigger*.

### User-defined functions

Untuk lebih memahami tentang User defined function, tulislah perintah berikut!

```
DELIMITER //  
CREATE FUNCTION tambah (angka1 INT, angka2 INT)  
RETURNS INT  
BEGIN  
    RETURN angka1 + angka2;  
END; //  
DELIMITER ;
```

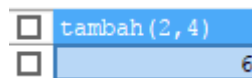


Gambar 8.1 Fungsi tambah

Untuk melakukan pemanggilan fungsi, lakukan perintah berikut ini :

```
SELECT tambah(2,4) AS Penjumlahan;
```

Hasilnya dapat anda lihat pada gambar dibawah ini:



Gambar 8.2 User-defined functions

Perbedaan antara *stored procedure* dan fungsi adalah *stored procedure* tidak mengembalikan nilai, sedangkan fungsi harus mengembalikan nilai. Fungsi dan *stored procedure* dapat digunakan dibagian perintah.

### Stored Procedures

Untuk lebih memahami tentang *Stored procedure*, tulislah perintah dibawah ini !

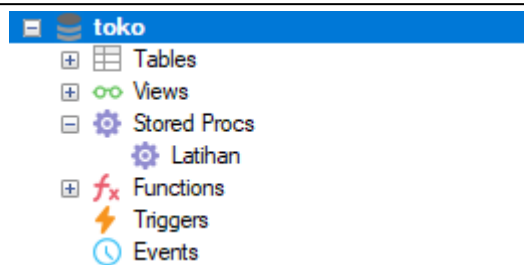
Pertama buat *Stored procedure* dengan nama Latihan.



```

DELIMITER //
CREATE procedure Latihan ( OUT nilai INT )
BEGIN
    DECLARE nilai_a INT;
    SET nilai_a = 50;
    label1: WHILE nilai_a <= 200 DO
        SET nilai_a = nilai_a * 2;
    END WHILE label1;
    SET nilai = nilai_a;
END; //
DELIMITER ;

```



Gambar 8.3 Store Procedure

Setelah selesai membuat *Stored procedure*, anda dapat menjalankan *perintah* diatas dengan perintah sebagai berikut:

```

CALL Latihan (@variabel);
SELECT @variabel;

```

Anda akan melihat hasilnya pada gambar dibawah ini.

@variabel	
	400

Gambar 8.4 Stored procedure.

Selanjutnya akan mencoba mempraktekkan penggunaan parameter pada *stored procedure* dengan menggunakan basis data "Toko", pertama tulislah perintah dibawah ini !

```

DELIMITER //
CREATE PROCEDURE cr_nm_barang (stok INT)
BEGIN
    SELECT * FROM toko.barang
        WHERE Stok_Barang = stok;
END; //
DELIMITER ;

```

Selanjutnya bisa mengeksekusi *stored procedure*nya dengan perintah dibawah ini.

```
CALL cr_nm_barang (0);
```

Pada perintah diatas akan terelihat data pada tabel Pembelian dengan Jumlah\_Barang yang berjumlah 1 sampai 2, dan lihatlah hasilnya pada gambar dibawah ini.

<input type="checkbox"/>	id_barang	nama_barang	tanggal_terima	stok_barang
<input type="checkbox"/>	BRG01	Sony	2015-04-03 00:00:00	0
<input type="checkbox"/>	BRG02	HTC	0000-00-00 00:00:00	0
<input type="checkbox"/>	BRG03	LG	2015-03-08 00:00:00	0
<input type="checkbox"/>	BRG04	Microsoft	2015-03-10 00:00:00	0
<input type="checkbox"/>	BRG05	Motorola	2015-03-15 00:00:00	0
<input type="checkbox"/>	BRG06	Xiaomi	2015-03-17 00:00:00	0

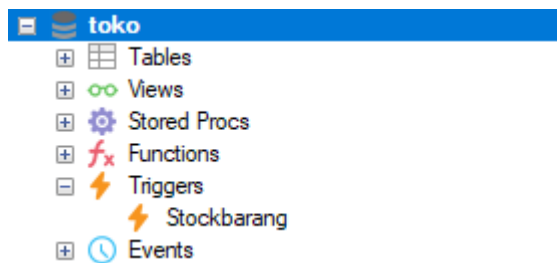
Gambar 8.5 Hasil dari *Stored procedure* pada tabel barang

## Trigger

Buat sebuah trigger menggunakan basis data Toko. Untuk membuat sebuah trigger gunakan perintah dibawah ini!

```
DELIMITER //
CREATE TRIGGER Stockbarang
AFTER INSERT
ON pembelian FOR EACH ROW
BEGIN
    UPDATE barang SET barang.Stok_Barang = barang.Stok_Barang
    + NEW.Jumlah_Pembelian
    WHERE barang.ID_Barang=NEW.ID_Barang;
END;
```

Apabila pembuatan Trigger ini sukses, maka pada navigator anda akan melihat tampilan yang sama seperti gambar berikut ini.



Gambar 8.4 Tampilan navigator hasil pembuatan *trigger*.

Pada perintah diatas akan dibuat sebuah trigger dengan nama Stockbarang dimana *trigger* tersebut akan terpicu jika ada perubahan di tabel Pembelian. Pada pembuatan *trigger* ini bertujuan untuk merubah nilai Stok\_Barang pada tabel Barang dengan menambahkan nilai Stok\_Barang yang

sekarang dengan jumlah barang yang dibeli (Jumlah\_Pembelian). Perhatikan pada perintah diatas terdapat kata **inserted**, ini merupakan logika yang digunakan untuk menyimpan data yang memicu terjadinya *trigger*, dalam hal ini nilai data yang dimasukkan (insert) kedalam tabel pembelian, selain **inserted**, tabel logika lainnya adalah **deleted**, tabel logika ini digunakan untuk *trigger* yg terpicu dengan kejadian **delete**.

Perhatikan gambar dibawah ini, gambar dibawah ini merupakan gambar pada saat menginserskan data pada tabel Barang saat trigger belum dibuat, yaitu kondisikan nilai Stok\_Barang dalam kondisi default 0.

<input type="checkbox"/>	id_barang	nama_barang	tanggal_terima	stok_barang
<input type="checkbox"/>	BRG01	Sony	2015-04-03 00:00:00	0
<input type="checkbox"/>	BRG02	HTC	0000-00-00 00:00:00	0
<input type="checkbox"/>	BRG03	LG	2015-03-08 00:00:00	0
<input type="checkbox"/>	BRG04	Microsoft	2015-03-10 00:00:00	0
<input type="checkbox"/>	BRG05	Motorola	2015-03-15 00:00:00	0
<input type="checkbox"/>	BRG06	Xiaomi	2015-03-17 00:00:00	0
*	(NULL)	(NULL)	(NULL)	0

Gambar 8.5 Tabel barang saat *trigger* belum dibuat.

Untuk mengetahui apakah trigger yang dibuat tadi bisa berjalan, coba menginserskan data pada tabel Pembelian.

```
INSERT INTO pembelian
(id_pembeli, id_barang, tanggal_beli, nama_pembeli,
jumlah_pembelian)
VALUES ('P08', 'BRG03', '2011-03-11', 'Wayan', 1);
```

Pada saat menuliskan perintah diatas maka akan terlihat message dibagian bawah seperti gambar diatas yang artinya ketika menuliskan perintah diatas maka secara otomatis akan menginserskan data sekaligus kedalam tabel Pembelian dan tabel Barang.

Anda akan melihat hasil pada tabel Pembelian dan tabel Barang seperti gambar dibawah ini.

<input type="checkbox"/>	id_pembeli	id_barang	tanggal_beli	nama_pembeli	jumlah_pembelian
<input type="checkbox"/>	P02	BRG02	2015-04-06 00:00:00	Made	2
<input type="checkbox"/>	P03	BRG06	2015-05-08 00:00:00	Komang	2
<input type="checkbox"/>	P04	BRG06	2015-06-10 00:00:00	Ketut	1
<input type="checkbox"/>	P08	BRG03	2011-03-11 00:00:00	Wayan	1
*	(NULL)	(NULL)	(NULL)	(NULL)	(NULL)

Gambar 8.6 Hasil dari proses Insert trigger tabel pembelian

<input type="checkbox"/>	id_barang	nama_barang	tanggal_terima	stok_barang
<input type="checkbox"/>	BRG01	Sony	2015-04-03 00:00:00	0
<input type="checkbox"/>	BRG02	HTC	0000-00-00 00:00:00	0
<input type="checkbox"/>	BRG03	LG	2015-03-08 00:00:00	1
<input type="checkbox"/>	BRG04	Microsoft	2015-03-10 00:00:00	0
<input type="checkbox"/>	BRG05	Motorola	2015-03-15 00:00:00	0
<input type="checkbox"/>	BRG06	Xiaomi	2015-03-17 00:00:00	0
*	(NULL)	(NULL)	(NULL)	0

Gambar 8.7 Hasil dari proses Insert trigger tabel barang

Pada gambar diatas terlihat nilai pada Stok\_Barang sudah terisi sesuai dengan jumlah pembelian yang dimasukkan pada perintah Insert tabel Pembelian diatas.

## TUGAS

1. Buatlah sebuah *stored procedure* pada basis data “Mahasiswa” yang anda buat untuk menampilkan semua data mahasiswa berdasarkan ketua jurusanannya!
2. Buatlah *stored procedure* yang akan mencari nama ketua jurusan dengan nama tertentu!
3. Buatlah sebuah trigger pada basis data “Mahasiswa”!
4. Apakah perbedaan antara *stored procedure* dengan fungsi?

## MODUL IX

### CONSEPTUAL DATA MODEL DAN PHYSICAL DATA MODEL

#### Tujuan :

1. Mahasiswa mampu menjelaskan tentang CDM dan PDM
2. Mahasiswa mampu membuat CDM
3. Mahasiswa mampu membuat PDM
4. Mahasiswa mampu membaca PDM dan menterjemahkan kedalam bentuk SQL.

#### Tugas Pendahuluan :

1. Apa Yang anda ketahui tentang Conceptual Data Model (CDM)?
2. Apa Yang anda ketahui tentang Physical Data Model (PDM)?

### DASAR TEORI

#### Conceptual Data Model (CDM)

CDM (Conceptual Data Model) atau model konsep data merupakan konsep yang berkaitan dengan pandangan pemakai terhadap data yang disimpan dalam basis data. CDM dibuat sudah dalam bentuk tabel-tabel tanpa tipe data yang menggambarkan relasi antar tabel untuk keperluan implementasi ke basis data (Rosa dan Shalahuddin, 2013).

#### Simbol-Symbol pada CDM

Tabel 9.1 Simbol pada Conceptual Data Model

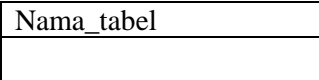
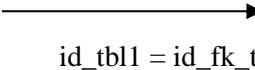
Simbol	Deskripsi
<div style="border: 1px solid black; padding: 5px; display: inline-block;">           Entitas / Tabel            Nama_tabel         </div>	Entitas atau tabel yang menyimpan data dalam basis data.
<div style="border: 1px solid black; padding: 5px; display: inline-block;">           Relasi            1..*              Nama Relasi              1..*         </div>	

#### Physical Data Model (PDM)

Model Relasional atau Physical Data Model (PDM) adalah model yang menggunakan sejumlah tabel untuk menggambarkan data serta hubungan antara data. Setiap tabel mempunyai sejumlah kolom di mana setiap kolom memiliki nama yang unik beserta tipe datanya. PDM merupakan konsep yang menerangkan detail dari bagaimana data disimpan di dalam basis data. PDM sudah merupakan bentuk fisik perancangan basis data yang sudah siap diimplementasikan ke dalam DBMS sehingga nama tabel juga sudah merupakan nama asli tabel yang akan diimplementasikan ke dalam DBMS (Rosa dan Shalahuddin, 2013).

## Simbol-Symbol pada PDM

Tabel 9.2 Simbol pada Physical Data Model

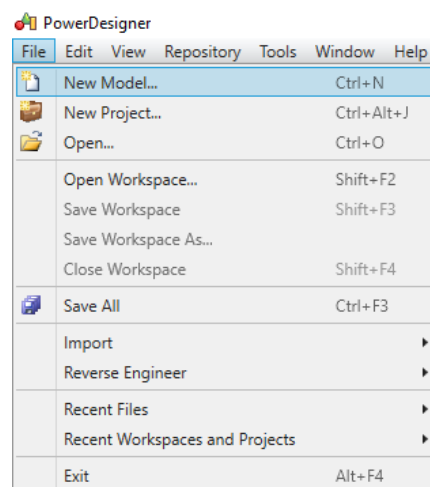
Simbol	Deskripsi
Tabel 	Tabel yang menyimpan data dalam basis data
Relasi 	Relasi antar tabel yang terdiri dari persamaan antara primary key (kunci primer) tabel yang di acu dengan kunci yang menjadi referensi acuan di tabel lain.

## KEGIATAN PRAKTIKUM

Salah satu perangkat lunak yang dapat digunakan untuk membuat CDM dan PDM adalah Power Designer. Power Designer akan memudahkan dalam membuat dan mengecek model yang telah dibuat. Misal terdapat permasalahan seperti berikut:

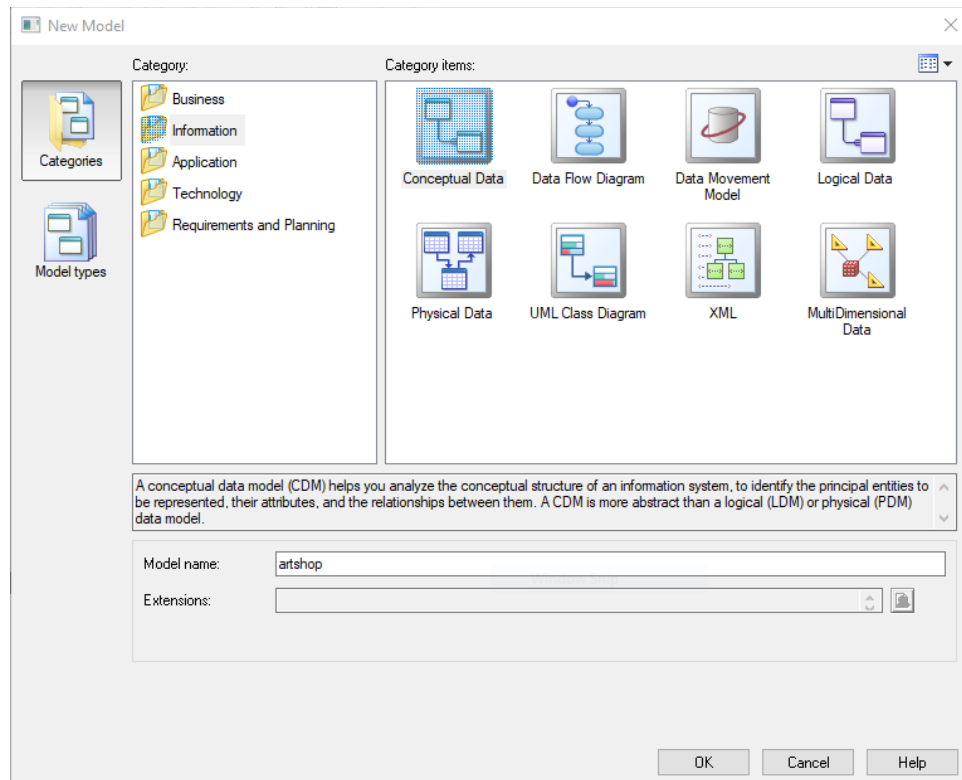
Sebuah artshop menjual berbagai barang kerajinan. Ketika ada *customers* yang datang untuk melakukan pembelian barang, kasir Artshop harus melakukan pencatatan transaksi menggunakan buku besar. Artshop melakukan pengadaan barang dengan cara membeli barang dari beberapa supplier. Ketika barang di terima dari suppliers kasir harus langsung menambahkan stok barang ke dalam buku besar.

1. Buka Power Designer, klik File, New Model, atau langsung tekan Ctrl+N di keyboard



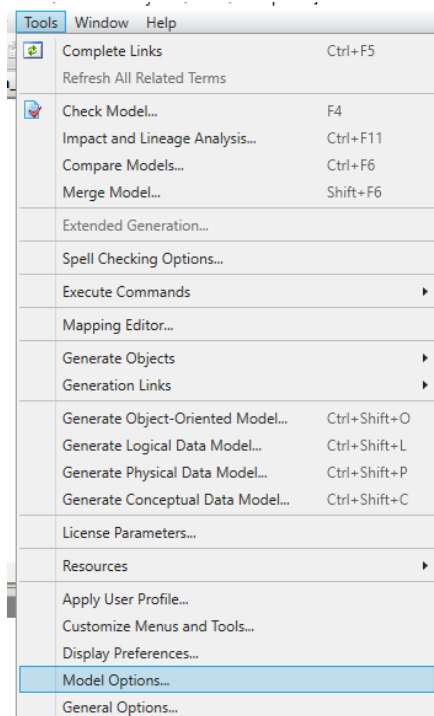
Gambar 9.1 Menu File

- Pilih Information, pilih Conceptual Data, kemudian masukkan nama model yang akan dibuat kedalam kolom Model name. Misal disini masukkan artshop.



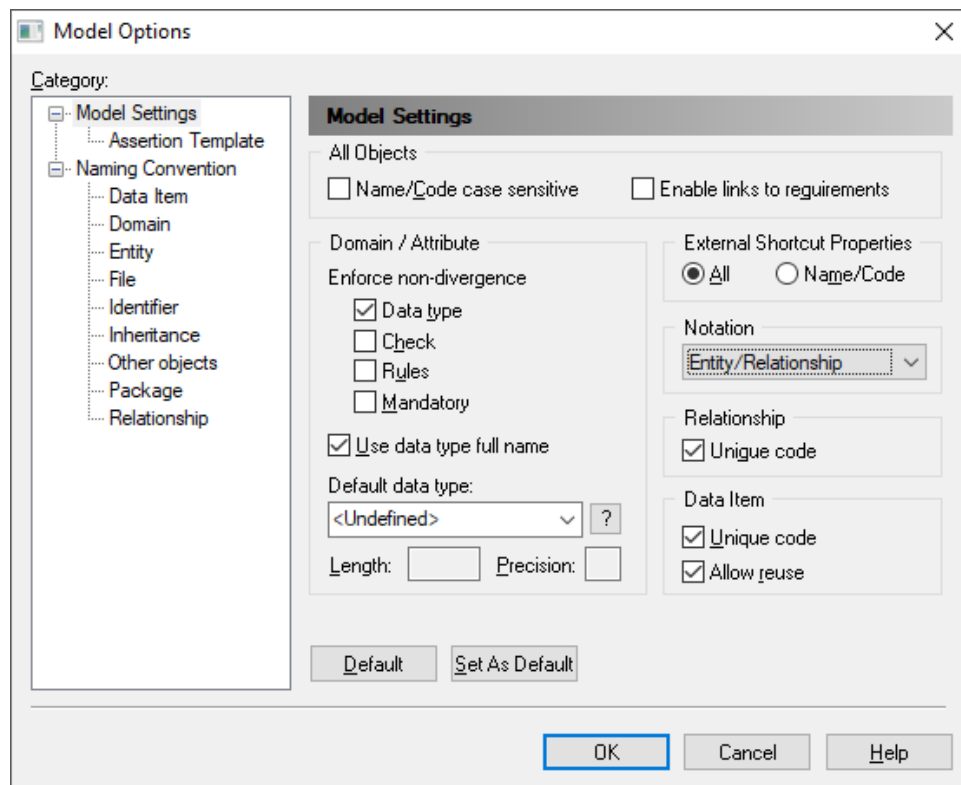
Gambar 9.2 New Model

- Rubah notasi CDM dengan klik pada Tools, kemudian klik Model Options



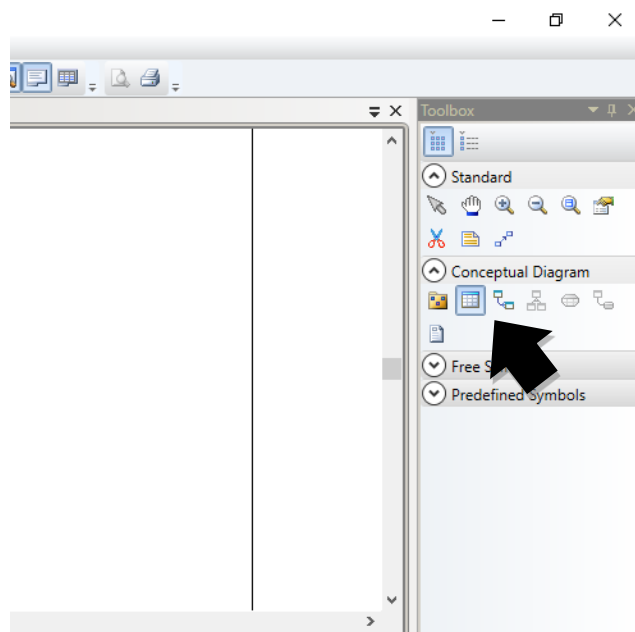
Gambar 9.3 Menu Tools

4. Pada Model Option pilih Notation menjadi Entity/Relation



Gambar 9.4 Model Option

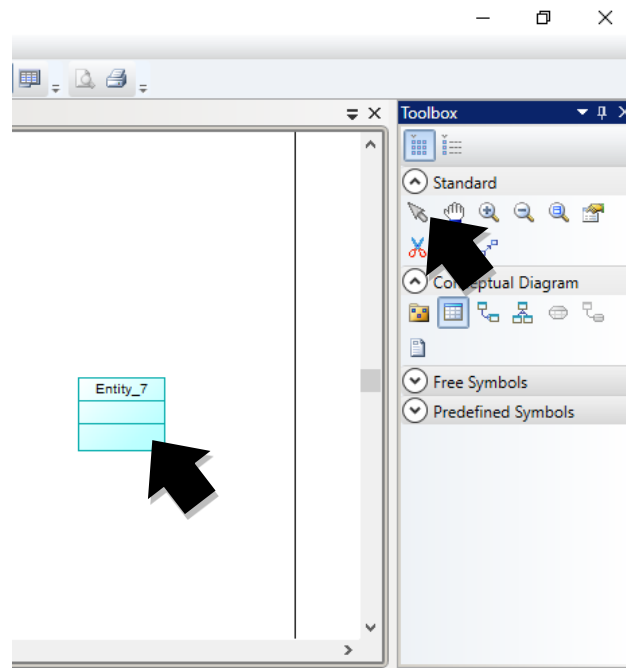
5. Untuk membuat entitas, pada toolbox, klik Entity



Gambar 9.5 Tools Box

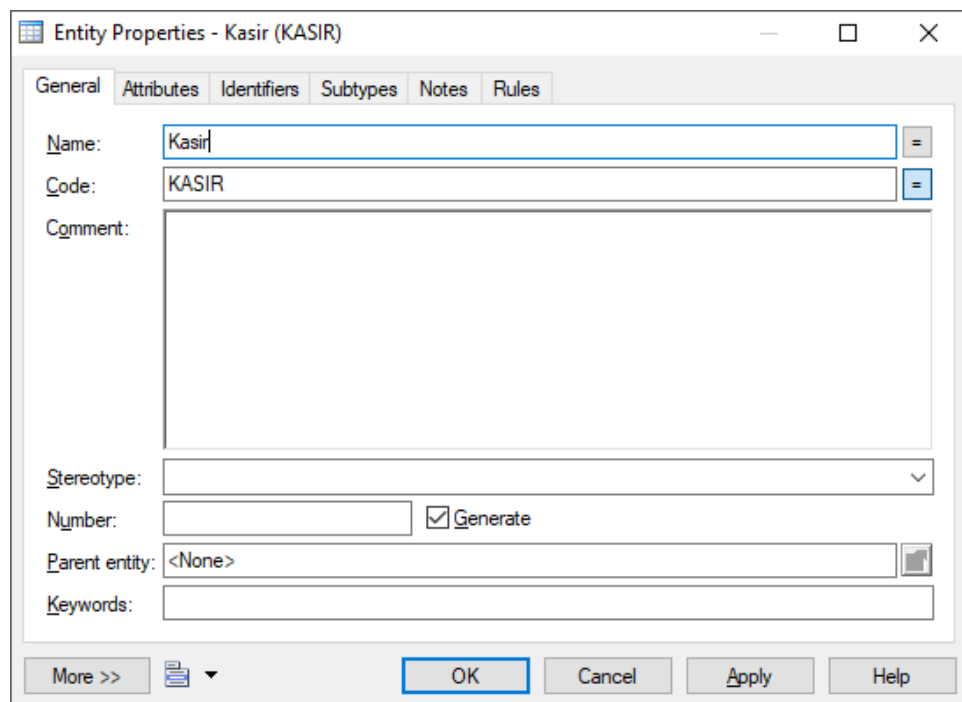
6. Klik pada area kosong, akan terbentuk sebuah entitas. Setelah terbentuk entitas, pada toolbox klik pada pointer. Kemudian klik duakali pada entitas yang terbentuk.





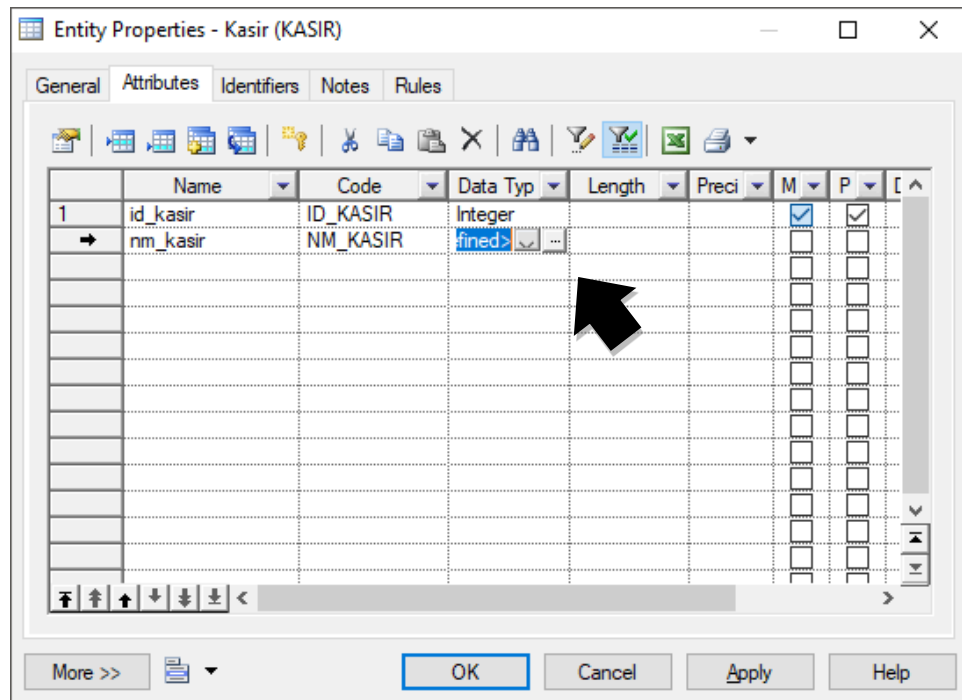
Gambar 9.6 Entitas

7. Pada halaman Entity Property, di tab General berikan nama entitas pada kolom Name.



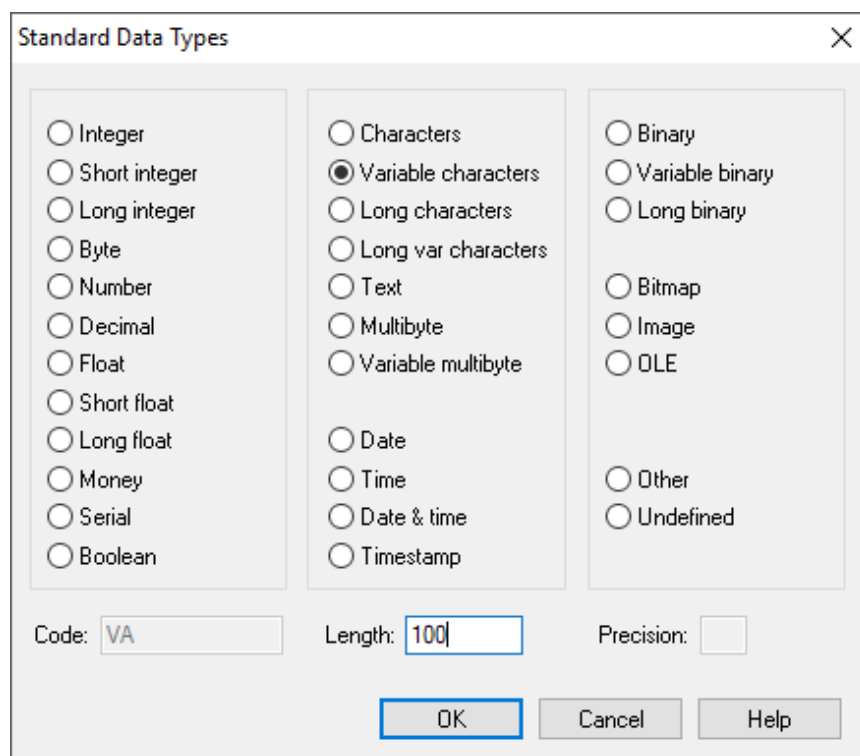
Gambar 9.7 Entity Property tab General

8. Pindah ke tab Atributtes dengan mengklik nama tab nya, kemudian isikan nama atribut. Untuk atribut yang merupakan primary key, berikan tanda centang pada kolom P. Untuk merubah tipe data, klik ... yang terpadap pada kolom Data Type



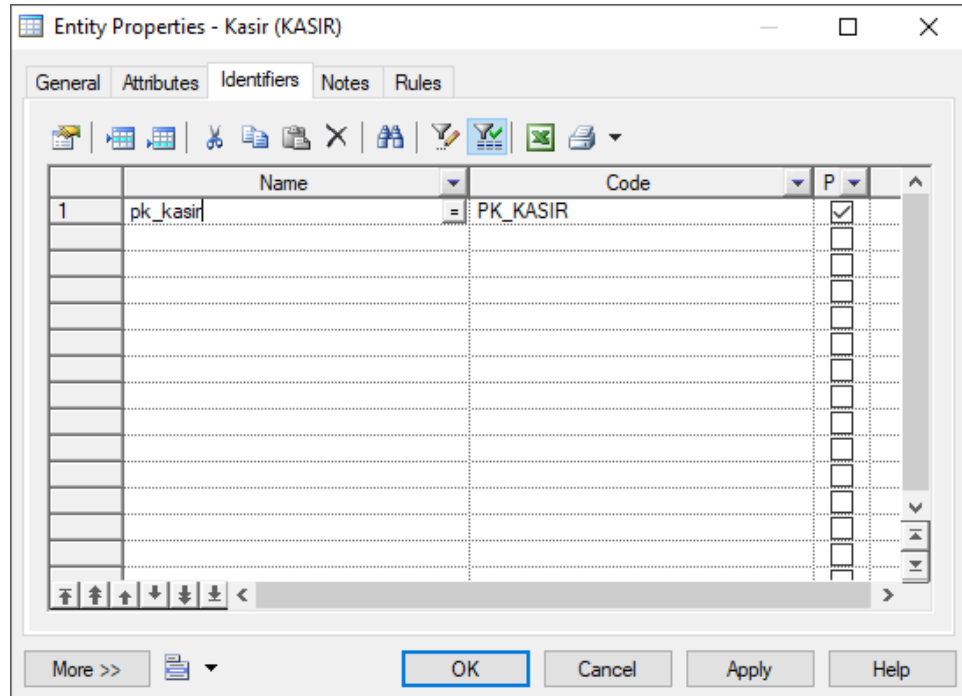
Gambar 9.8 Entity Property tab Atributtes

9. Pilih Tipe data yang bersesuaian. Pada contoh ini dipilih Variable characters. Karena pada Variable characters memerlukan Panjang, maka harus dimasukkan pada kolom length yang berada di bawah. Untuk tipe data yang tidak memerlukan Panjang, kolom ini tidak akan aktif.



Gambar 9.9 Pemilihan Tipe Data

10. Pindah ke tab Identifiers dengan mengklik nama tab nya, kemudian masukkan nama identifiernya. Kemudian klik OK



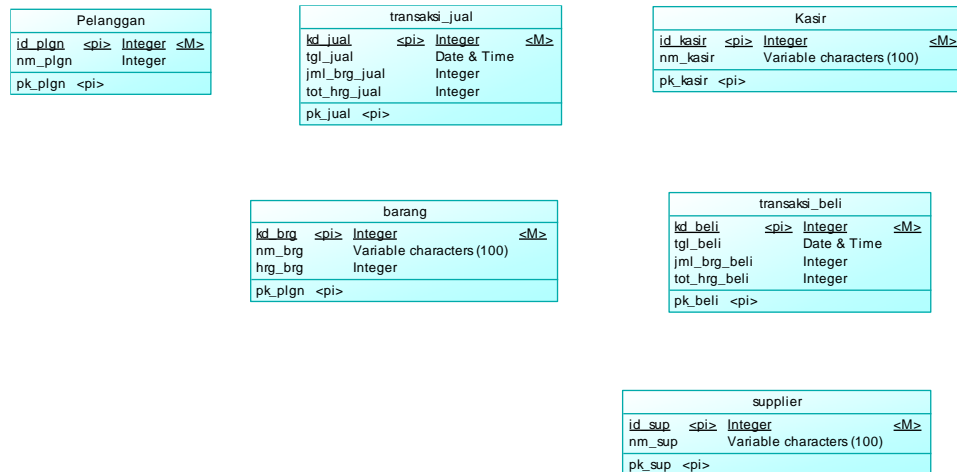
Gambar 9.10 Entity Property tab Identifiers

Sampai pada tahap ini sudah berhasil dibuat sebuah entitas. Ulangi langkah tersebut untuk membuat entitas lainnya.

Kasir				
id_kasir	<pi>	Integer		<M>
nm_kasir		Variable characters (100)		
pk_kasir	<pi>			

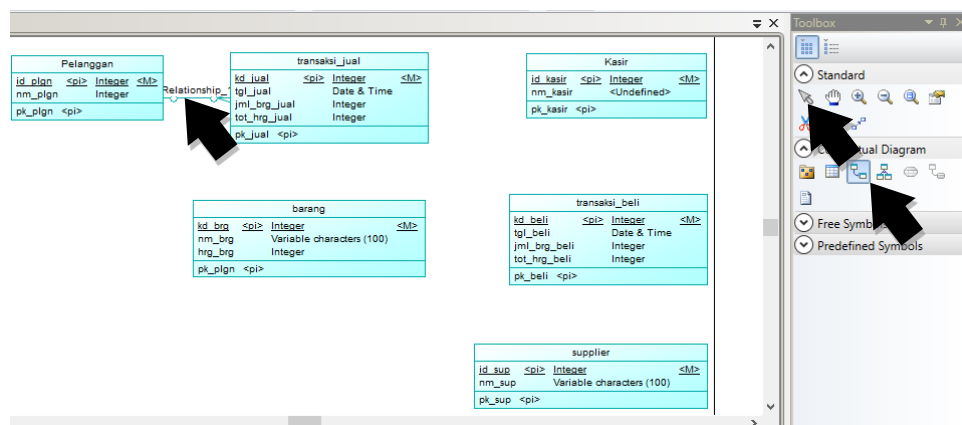
Gambar 9.11 Sebuah Entitas

Hasil akhirnya akan tampak sebagai berikut:



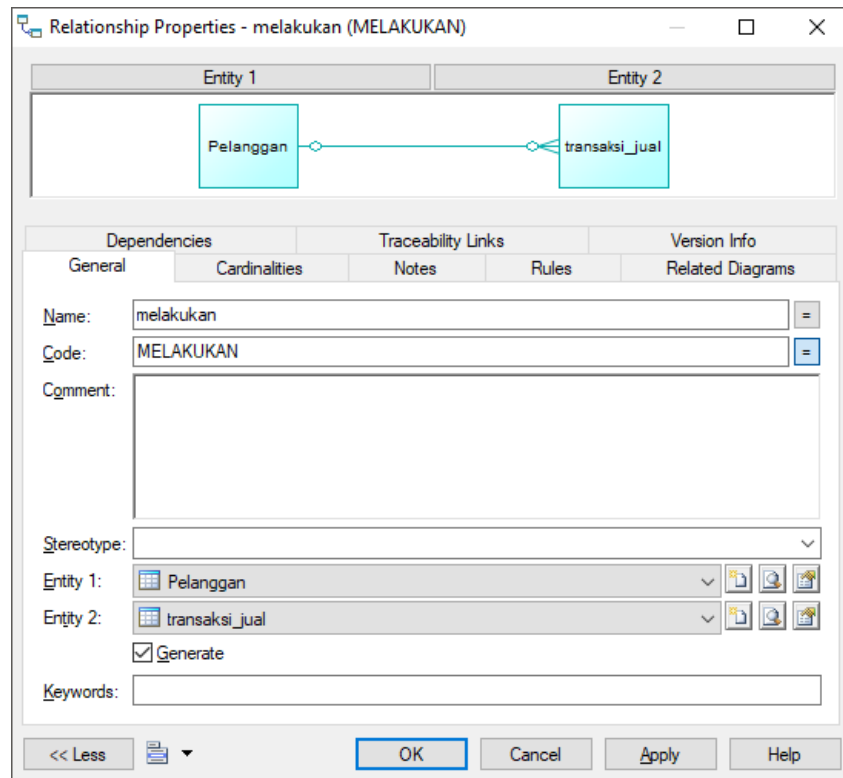
Gambar 9.12 Semua Entitas

11. Tahap seanjutnya adalah menambahkan relasi. Pada toolbox klik Relationship, kemudian Tarik dari atas sebuah entitas, ke entitas lainnya. Klik pada Pointer, kemudian klik dua kali pada relasi yang telah terbentuk



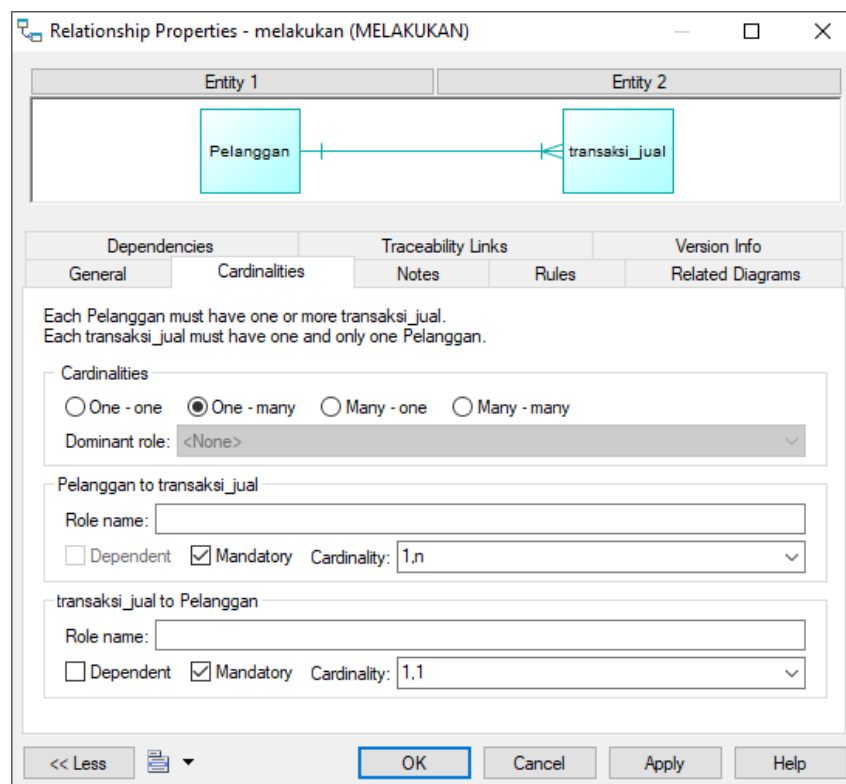
Gambar 9.13 Pembuatan Relasi

12. Pada halaman Relationship Properties, di tab General, masukkan nama relasi pada kolom Name.



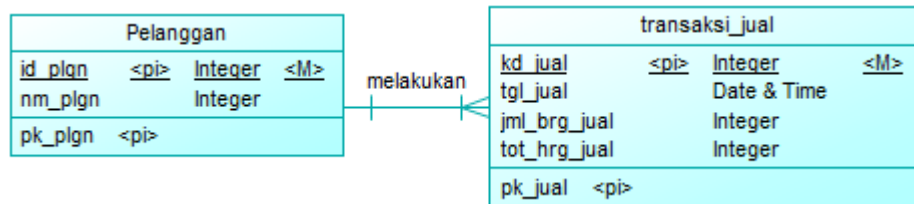
Gambar 9.14 Relationship Properties tab General

13. Pindah ke tab Cardinalities dengan mengklik nama tab nya, kemudian Sesuaikan dengan relasi yang ingin dibuat. Kemudian klik OK



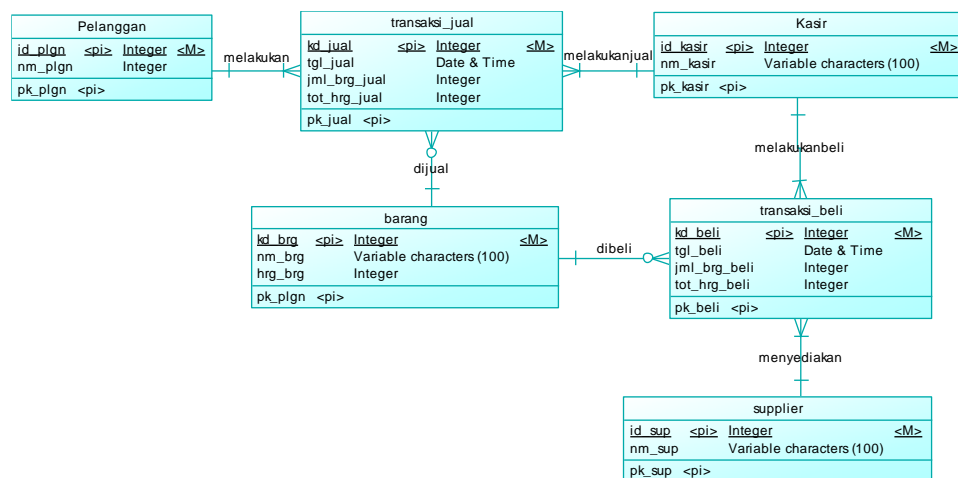
Gambar 9.15 Relationship Properties tab Cardinalities

Sampai tahap ini telah meng dibuat sebuah relasi untuk dua buah entitas. Lakukan hal yang serupa untuk membuat relasi yang lain.



Gambar 9.16 Relasi Dua Entitas

Hasilnya akan tampak sebagai berikut:

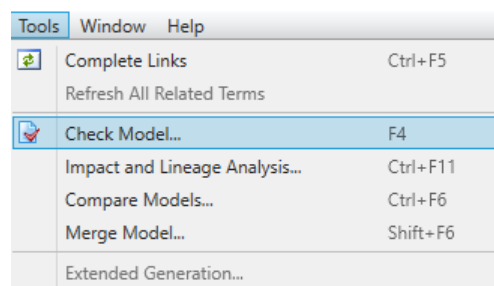


Gambar 9.17 Relasi Seluruh Entitas

14. Simpan model ini dengan menekan Ctrl + s pada keyboard.

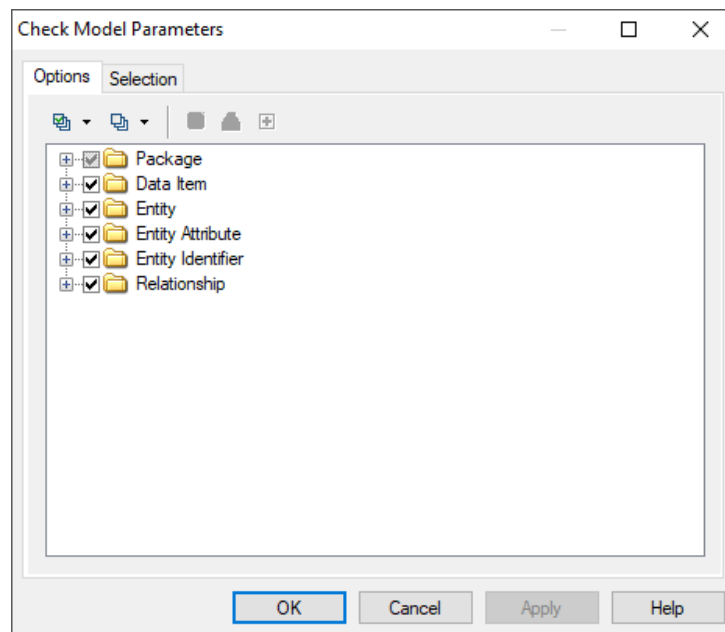
Smpai pada tahap ini, proses pembuatan CDM sudah selesai, namu untuk memastikan model yang dibuat sudah benar, maka perlu dilakukan pengecekan terhadap model yang dibuat. Power Designer menyediakan tools untuk melakukan pengecekan, caranya adalah

1. Klik Pada Tools, Kemudian Klik pada Check Model, atau dengan menekan F4 pada keyboard.



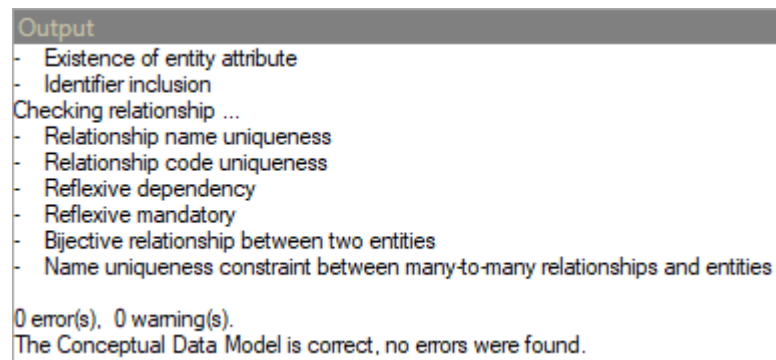
Gambar 9.18 Relasi Seluruh Entitas

2. Pada halaman Check Model Parameters, pilih opsi pengecekan yang ingin dilakukan, kemudian klik OK



Gambar 9.19 Check Model Parameters

3. Hasil pengecekan akan ditampilkan pada bagian output, di kiri bawah. Jika tertulis 0 error dan 0 warning berarti secara teknis sudah tidak ada masalah pada model yang dibuat.

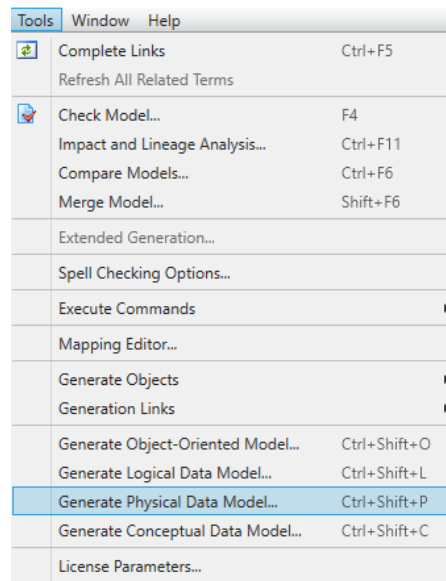


Gambar 9.20 Output Pengecekan Model

### Membuat Physical Data Model dari CDM

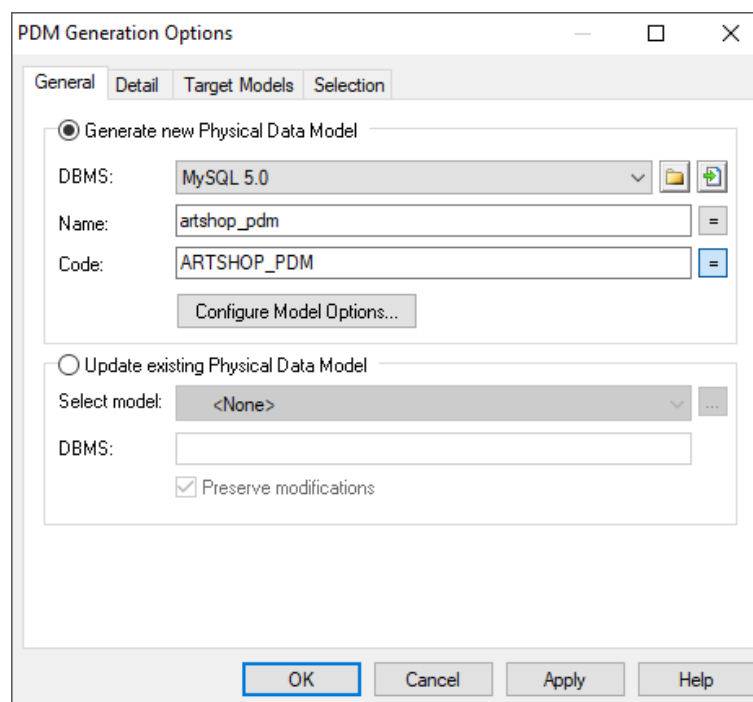
Power Designer menyediakan tools untuk membuat PDM dari CDM. Untuk dapat membuat PDM dari CDM, harus dipastikan dulu CDM sudah dicek dan tidak terdapat masalah. Untuk membuat PDM dari CDM dapat dilakukan dengan cara berikut:

1. Klik tools, klik Generate Physical Data Model, atau tekan Ctrl + Shift + P pada keyboard



Gambar 9.21 Menu Tools

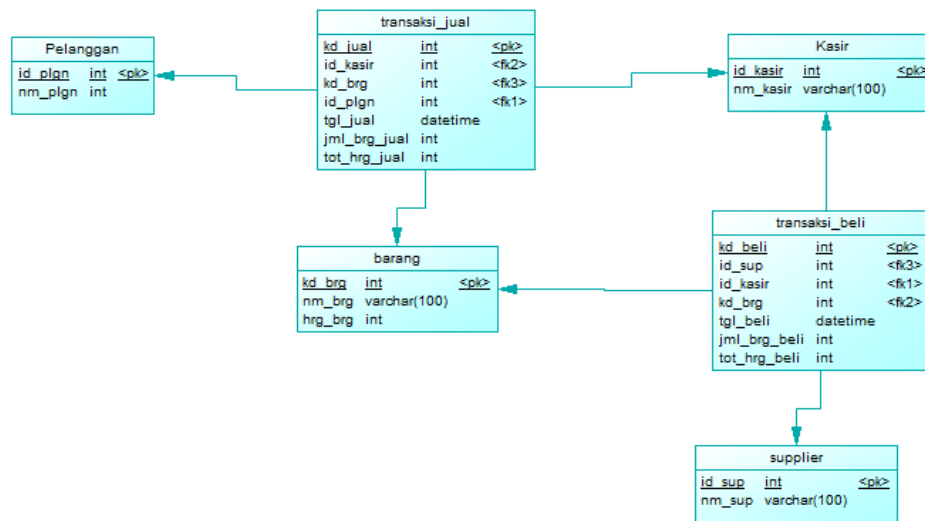
2. Pada halaman PDM Generation Option pilih Generate Physical Data Model, kemudian pilih DBMS yang sesuai, pada contoh ini digunakan MySQL. Beri nama pada kolom Name. Terakhir, klik OK. Tunggu beberapa saat, PDM akan tercipta.



Gambar 9.22 PDM Generation Option



Hasil PDM akan tampak sebagai berikut



Gambar 9.23 PDM yang Dihasilkan

Untuk memastikan tidak terdapat masalah teknis pada PDM ini maka lakukan pengecekan, seperti pada CDM (Tekan F4 pada keyboard), jika sudah terdapat 0 error dan 0 warning, maka CDM tidak memiliki masalah secara teknis.

```

Output
- Key inclusion
Checking reference ...
- Reference name uniqueness
- Reference code uniqueness
- Reflexive and mandatory reference
- Existence of reference join
- Reference code maximum length
- Incomplete join
- Join order

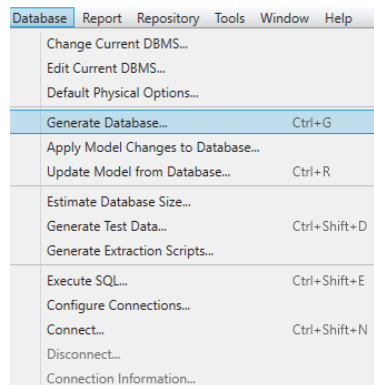
0 error(s), 0 warning(s).
The Physical Data Model is correct, no errors were found.
  
```

Gambar 9.24 Hasil Pengecekan Model

### Membuat SQL dari PDM

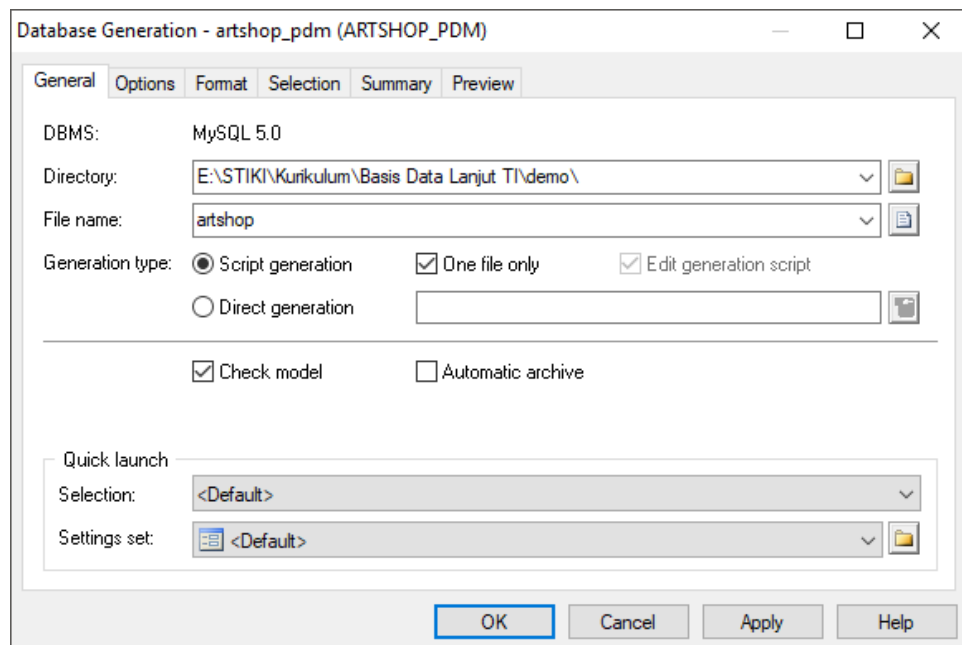
Query SQL untuk membuat table dari seluruh entitas yang ada pada PDM dapat dilakukan dengan otomatis. Power designer telah menyediakan fasilitas untuk itu. Caranya sebagai berikut:

1. Klik Database, Klik Generate Database atau tekan Ctrl + G pada Keyboard.



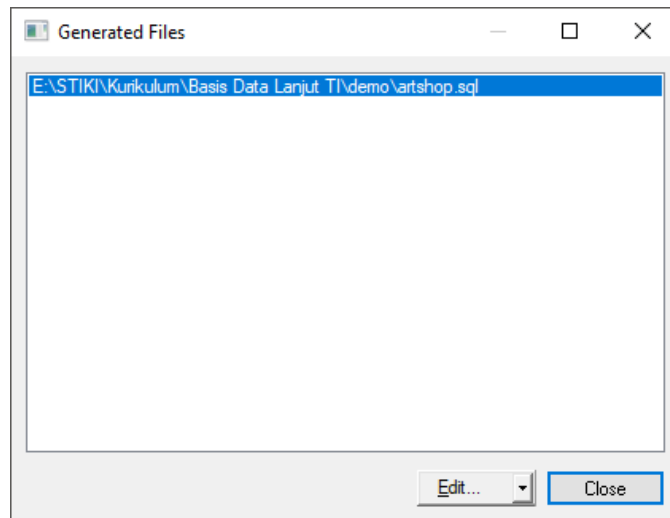
Gambar 9.25 Hasil Pengecekan Model

2. Tentukan lokasi penyimpanan dan nama file. Kemudian klik OK.



Gambar 9.26 Hasil Pengecekan Model

3. Setelah uery berhasil dibuat akan muncul informasi lokasi file yg dibuat. Klik Close.



Gambar 9.27 Lokasi File SQL

Sampai pada tahap ini query SQL sudah berhasil terbentuk, silahkan buka lokasi penyimpanan file.

## **TUGAS**

Cari sebuah permasalahan nyata, paparkan masalahnya, kemudian bangun CDM PDM dan query untuk membuat tabelnya.

## DAFTAR PUSTAKA

- Alam.J, M. Agus. 2005, *Pemrograman Transact-SQL pada SQL Server 2005*, Elexmedia Komputindo, Bandung;
- Kadir, Abdul. 2002. *Penuntun Praktis Belajar SQL*, Penerbit Andi Yogyakarta.
- Octaviani, HS. 2010, *SQL Server 2008 Express*, Penerbit Andi, Yogyakarta
- Rosa dan Shalahuddin, M. 2013. *Rekayasa Perangkat Lunak Terstruktur Dan Berorientasi Objek*. Informatika. Bandung
- Silberschatz, A., Korth, H., dan Sudarshan, S., 2011, *Database System Concepts 6th Edition*, McGraw-Hill