



MODUL PRAKTIKUM BASIS DATA

TIM ASISTEN
LABORATORIUM REKAYASA PERANGKAT LUNAK
DEPARTEMEN MATEMATIKA FAKULTAS MIPA
UNIVERSITAS HASANUDDIN

MODUL PRAKTIKUM
BASIS DATA



Program Studi Sistem Informasi
Departemen Matematika
Fakultas Matematika dan Pengetahuan Alam
Universitas Hasanuddin
Makassar
2022

Kata Pengantar

Dengan menyebut nama Allah SWT yang Maha Pengasih lagi Maha Penayang, Kami panjatkan puja dan puji syukur atas kehadiran-Nya, yang telah melimpahkan rahmat, hidayah, dan inayah-Nya kepada kami, sehingga kami dapat menyelesaikan modul sistem basis data.

Modul ini telah kami susun dengan maksimal dan mendapatkan bantuan dari berbagai pihak. Untuk itu kami menyampaikan banyak terima kasih kepada semua pihak yang telah berkontribusi dalam pembuatan modul ini.

Terlepas dari semua itu, Kami menyadari sepenuhnya bahwa masih ada kekurangan baik dari segi susunan kalimat maupun tata bahasanya. Oleh karena itu dengan tangan terbuka kami menerima segala saran dan kritik dari pembaca agar kedepannya modul ini dapat semakin baik.

Akhir kata kami berharap semoga modul sistem basis data ini dapat memberikan manfaat maupun inspirasi bagi pembaca.

Makassar, September 2022

DAFTAR ISI

Kata Pengantar	ii
DAFTAR ISI	iii
BAB 1 PENGANTAR	1
1.1 MYSQL.....	1
BAB 2 DATA DEFINITION LANGUAGE (DDL)	6
2.1 CREATE	6
2.2 SHOW	9
2.3 ALTER.....	10
2.4 DROP.....	11
TUGAS PRAKTIKUM	13
BAB 3 QUERY DASAR MYSQL.....	15
3.1 SELECT	15
3.2 WHERE	16
3.3 ORDER BY	18
3.4 LIMIT	18
3.5 DISTINCT	19
TUGAS PRAKTIKUM	21
BAB 4 DATA MANIPULATION LANGUAGE (DML)	22
4.1.INSERT	22
4.2. UPDATE.....	24
4.3. DELETE.....	26
4.4. Hal Penting dalam DML	27
TUGAS PRAKTIKUM	28
BAB 5 JOIN	29
5.1 INNER JOIN	29
5.2 OUTER JOIN	32
TUGAS PRAKTIKUM	36
BAB 6 OPERATOR	37
6.1 Operator Aritmatika	37
6.2 Operator Bitwise	38
6.3 Operator Pembanding.....	38
6.4 Operator Logika	39
6.5 Predicate	42
6.6 Fungsi (Function).....	42
Tugas Praktikum	44

BAB 7 FUNCTION DAN GROUPING.....	45
7.1 AGGREGATE FUNCTION	45
7.2 COMPARISON FUNCTION	53
7.3 DATE FUNCTION.....	55
7.4 STRING FUNCTION	56
TUGAS PRAKTIKUM	59
BAB 8 SUBQUERY SQL.....	60
8.1 Pengenalan <i>Subquery</i>	60
8.2 Penggunaan <i>Subquery</i> dalam WHERE.....	60
8.3 Penggunaan <i>Subquery</i> dalam FROM	61
8.4 Penggunaan <i>subquery</i> dalam SELECT.....	61
8.5. Penggunaan IN/NOT IN dalam <i>subquery</i>	62
8.6. Penggunaan EXIST/NOT EXIST dalam <i>subquery</i>	63
8.7. <i>Correlated Subquery</i>	64
TUGAS PRAKTIKUM	65
BAB 9 UNION, INTERSECT & EXCEPT	66
9.1 Union	66
9.2 INTERSECT	68
9.3 EXCEPT	68
TUGAS PRAKTIKUM	69
BAB 10 TRANSACTIONS.....	70
10.1 Pengertian Database Transaction	70
10.2 Cara menggunakan perintah COMMIT	70
10.3 Cara menggunakan perintah ROLLBACK.....	71
TUGAS PRAKTIKUM	73
BAB 11 INDEX, CONTROL FLOW	74
11.1 INDEX	74
11.2 CONTROL FLOW	75
TUGAS PRAKTIKUM	76

BAB 1 | PENGANTAR

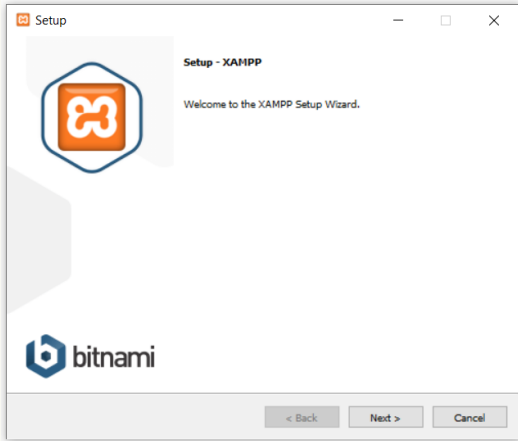
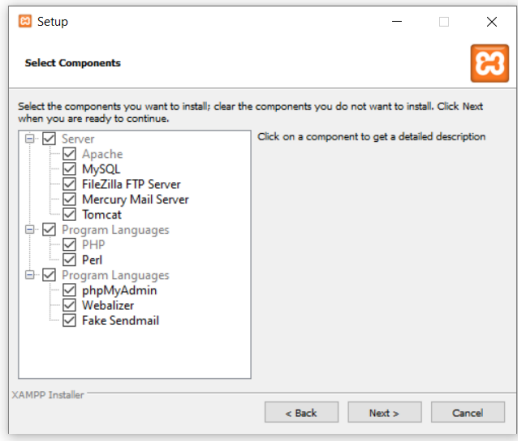
1.1 MYSQL

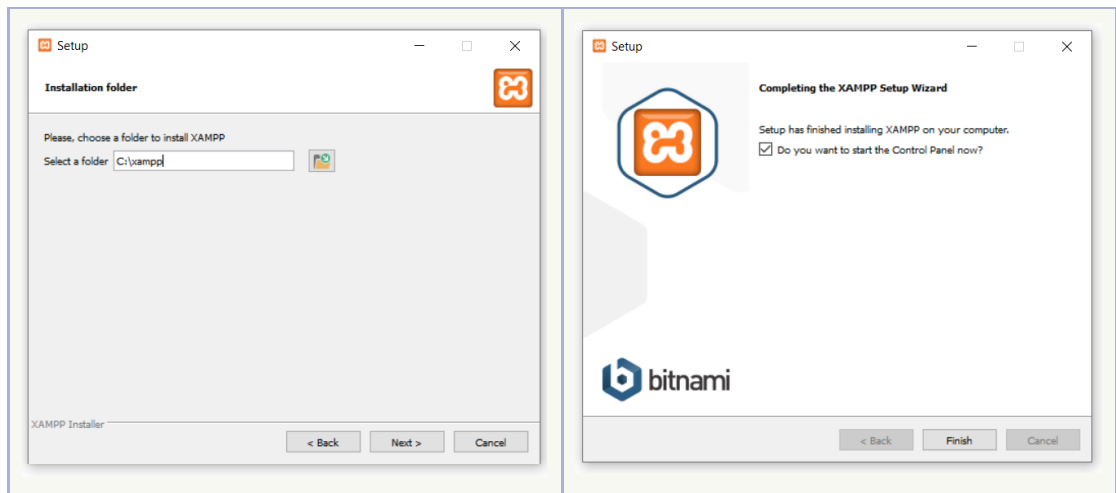
MySQL adalah sebuah DBMS (Database Management System) menggunakan perintah SQL (Structured Query Language) yang banyak digunakan saat ini dalam pembuatan aplikasi berbasis website. MySQL termasuk ke dalam RDBMS (Relational Database Management System). Sehingga, menggunakan tabel, kolom, baris, di dalam struktur databasenya.

XAMPP adalah software berbasis web server yang mendukung banyak sistem operasi. XAMPP digunakan sebagai server lokal (localhost), yang terdiri dari beberapa program antara lain : Apache, HTTP Server, MYSQL database, dan penerjemah bahasa pemrograman PHP dan Perl. Karena didalam XAMPP sudah termuat MYSQL database maka setelah menginstall XAMPP kita bisa langsung menggunakan MYSQL.

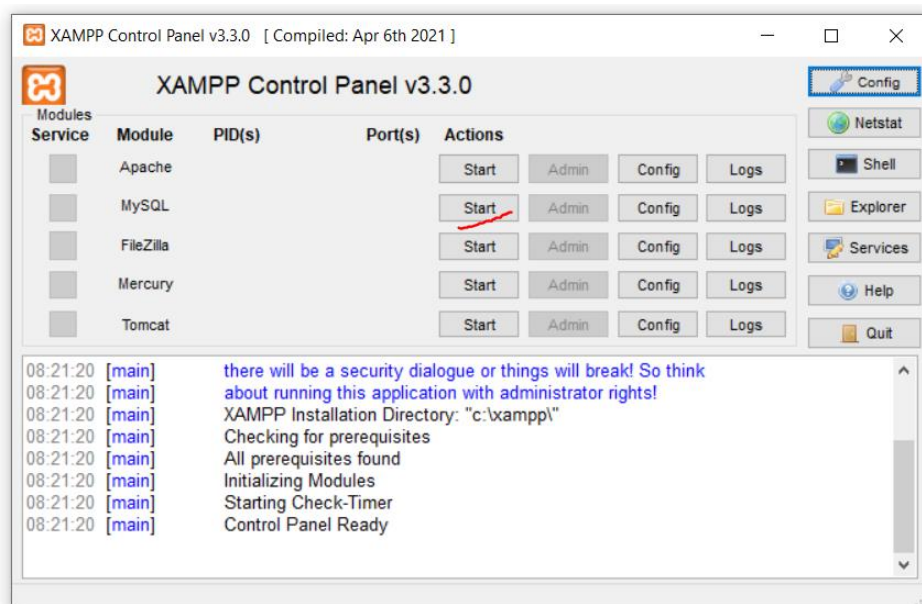
Cara Install XAMPP

1. Download XAMPP (versi terbaru yang menggunakan PHP 8) pada link berikut <https://www.apachefriends.org/>
2. Buka Instalasi XAMPP
3. Lakukan Proses Instalasi Sampai Selesai

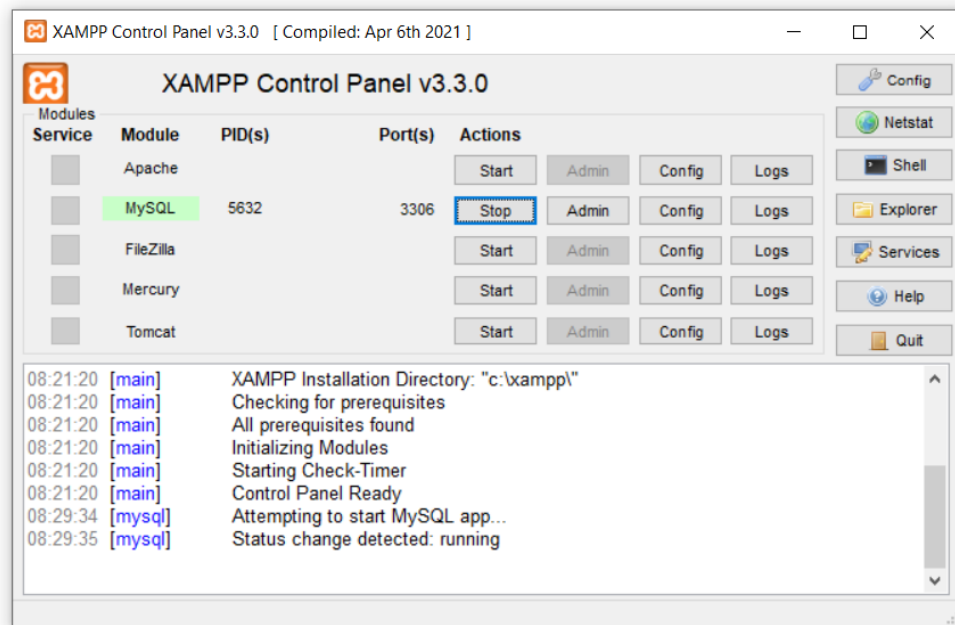
Tampilan Setup Instalasi	Disini bisa dipilih komponen yang akan dipasang
	
Perhatikan direktori tempat XAMPP	Lanjutkan sampai proses instalasi selesai



Setelah proses instalasi selesai kita bisa langsung menjalankan XAMPP, berikut tampilan Control Panel dari XAMPP



Untuk menggunakan MYSQL pastikan terlebih dahulu telah mengaktifkan service MYSQL.

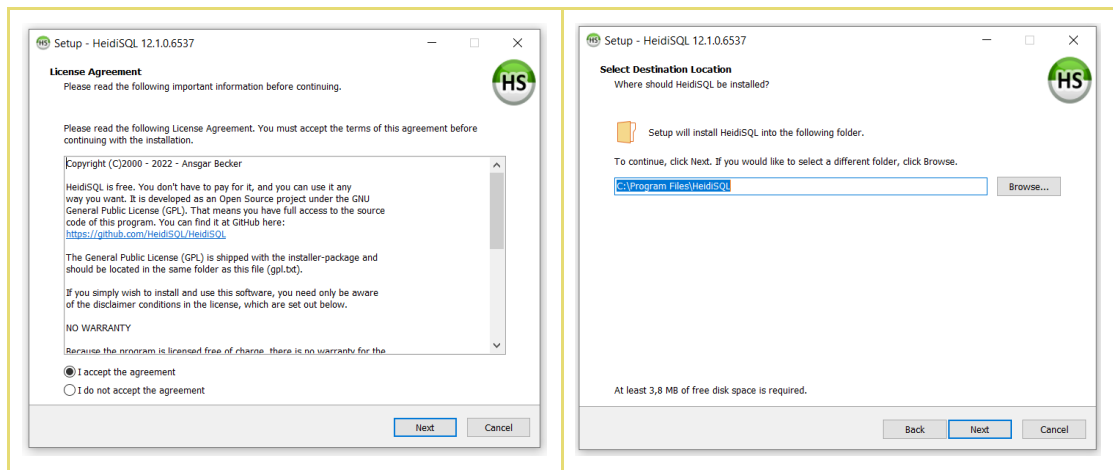


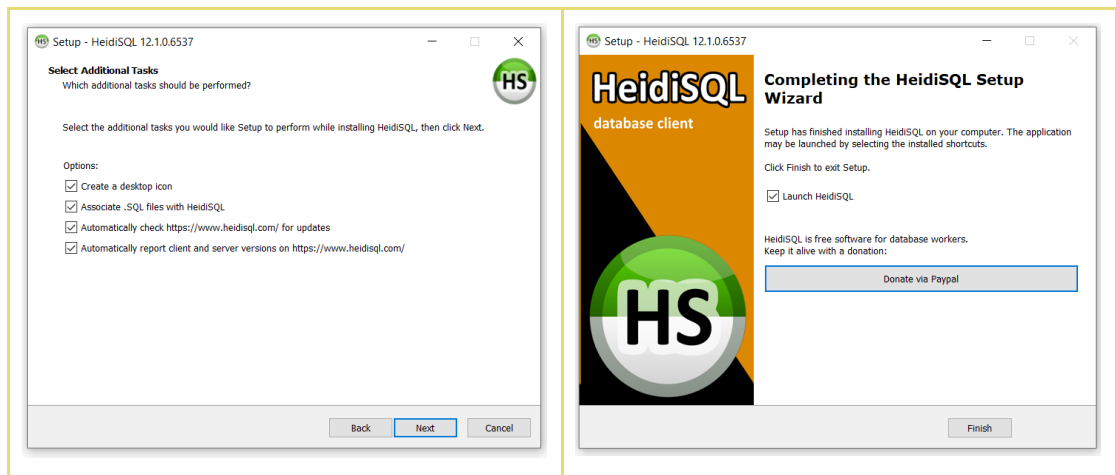
Jika Service MySQL telah jalan maka tampilannya akan seperti diatas.

Untuk melakukan atau menjalankan query MYSQL terdapat beberapa cara yang bisa digunakan mulai dari penggunaan CLI, ataupun menggunakan aplikasi berbasis GUI seperti phpMyAdmin, Heidi SQL, BeeKeeper, DataGrip, dll. Pada modul ini sendiri akan ditunjukkan cara install dan setting awal Heidi SQL.

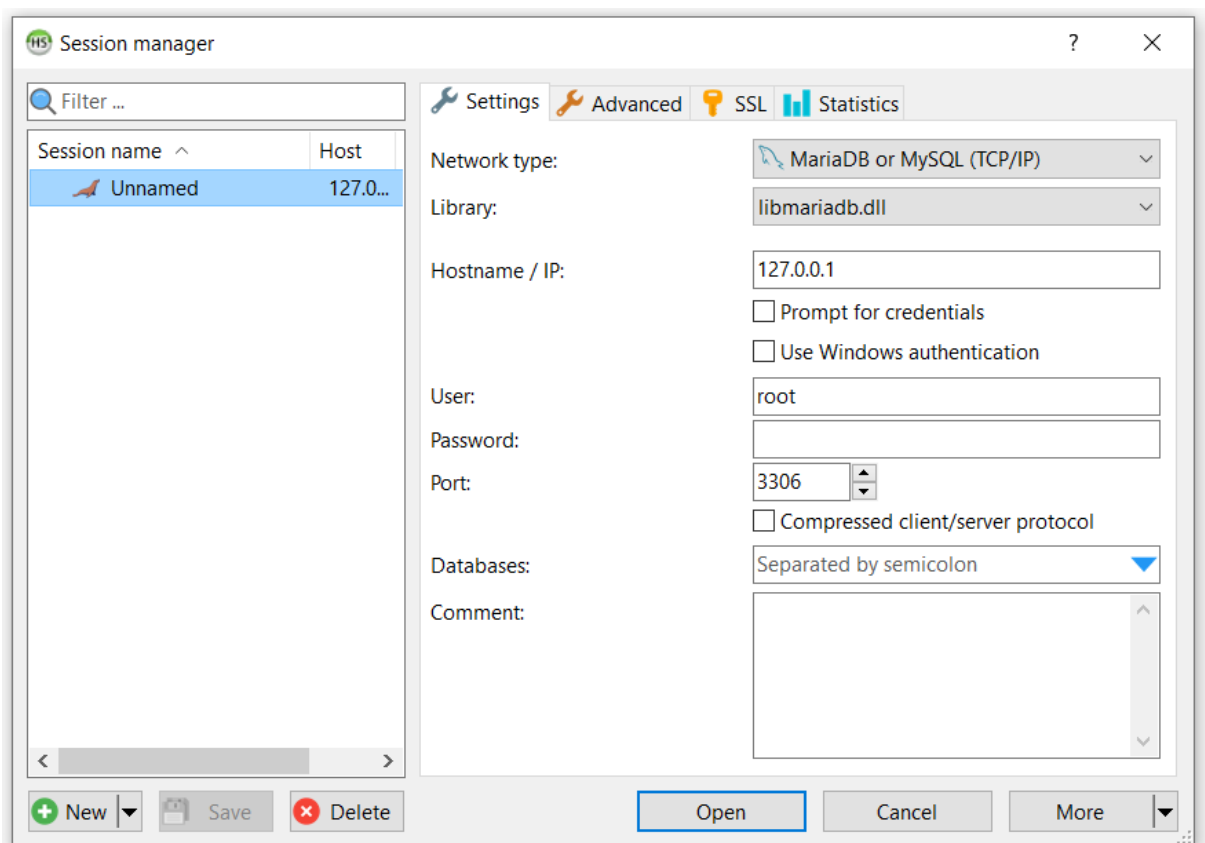
1. Download Heidi SQL pada link berikut

<https://www.heidisql.com/download.php>



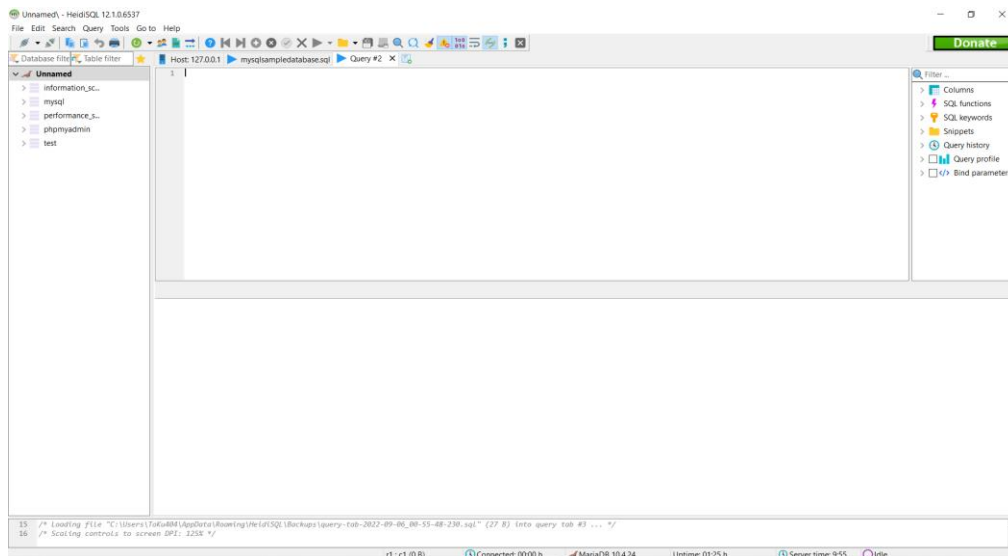


2. Setelah Download Selesai jalankan Heidi SQL maka tampilannya akan seperti berikut



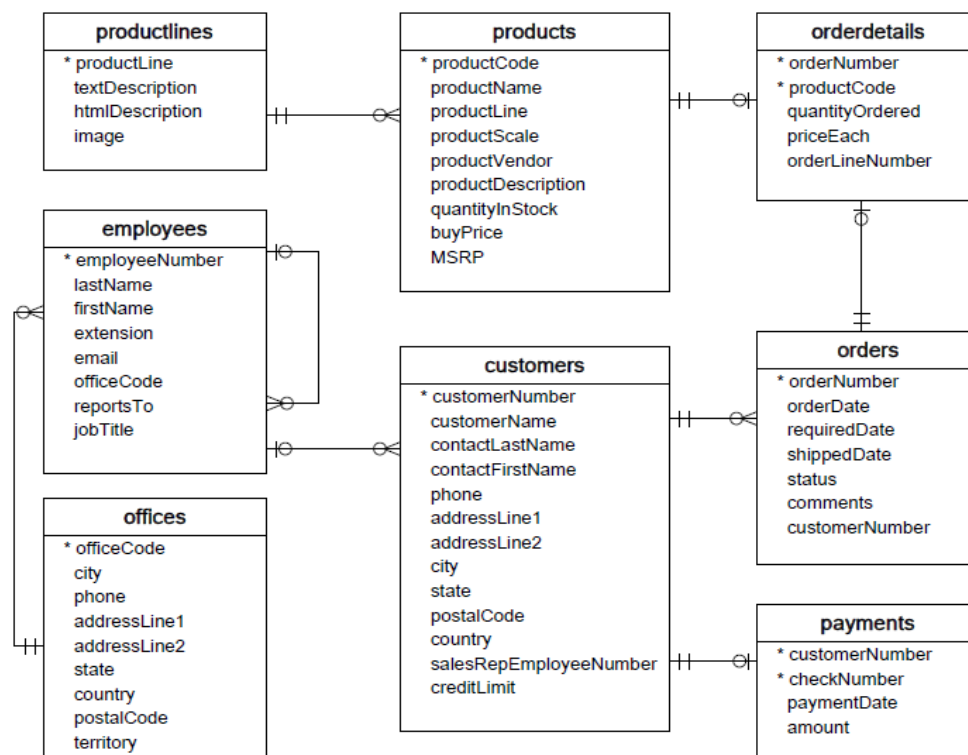
Pada halaman ini silahkan atur konfigurasi untuk server MYSQL yang telah diaktifkan sebelumnya.

3. Heidi SQL siap digunakan, silahkan lakukan perintah query MYSQL



Pada Modul ini sebagian besar latihan akan menggunakan database *classicmodels*.
Silahkan klik link berikut untuk mendownload database tersebut
<https://www.mysqltutorial.org/wp-content/uploads/2018/03/mysqlsampledatabase.zip>.

Berikut tampilan ERD dari Database Classic Models



Selain classic models, database yang akan banyak digunakan dimodul ini adalah database appseminar yang bisa didownload di link berikut :
<https://raw.githubusercontent.com/ToKu404/Lab-BasisData/main/appseminar.sql>

BAB 2 | DATA DEFINITION LANGUAGE (DDL)

DDL adalah kepanjangan dari *Data Definition Language* yang merupakan bahasa komputer untuk membuat dan memodifikasi sebuah struktur objek *database*. Hal yang dimaksud objek *database* tersebut adalah tampilan, tabel, skema, indeks, dan lainnya.

2.1 CREATE

Membuat Database Dengan CREATE

Untuk membuat basis data digunakan sintaks CREATE DATABASE diikuti nama database. Untuk membuat database digunakan syntax

CREATE DATABASE nama_database;

Contoh:

```
CREATE DATABASE mydatabase;
```

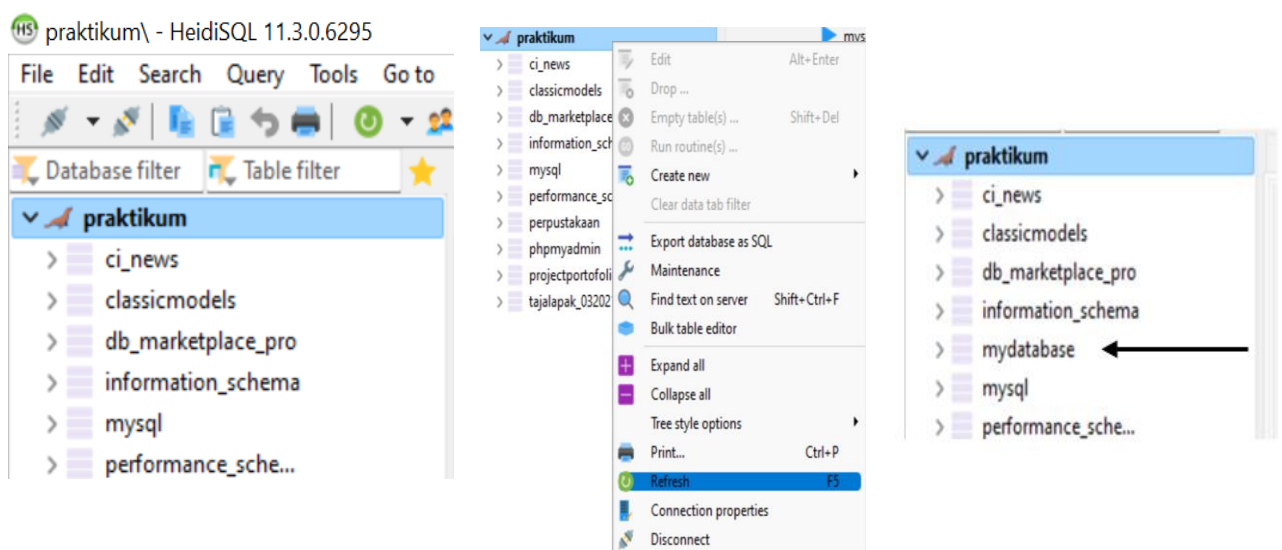
Maka akan membuat database bernama mydatabase. Untuk memilih dan menggunakan basis data mydatabase, gunakan syntax

USE nama_database;

Catatan: Pada heidisql untuk menjalankan query terbaru, maka query sebelumnya harus di nonaktifkan/ coment. Bisa dilakukan dengan menambahkan # pada query atau bisa dengan block query, klik kanan dan pilih un/coment

```
# CREATE DATABASE mydatabase;  
USE mydatabase;
```

Setelah create database maka hasilnya akan muncul disini, jika belum muncul klik kanan pada database praktikum setelah itu klik refresh.



Membuat Tabel Dengan CREATE Table Sederhana

```
CREATE TABLE mytable (  
  nama varchar(100)  
)
```

Perintah diatas akan membuat table mytable dengan satu atribut bernama nama dengan tipe data varchar(100).

Tipe data digunakan untuk mendefinisikan tipe dari *field* di *table*. Berikut beberapa tipe data yang sering digunakan

Tipe Data	Keterangan
INT	Menyimpan nilai integer
FLOAT	Menyimpan nilai float
VARCHAR	Menyimpan nilai string
CHAR	Menyimpan nilai satu karakter
DATE	Menyimpan nilai waktu
TEXT	Menyimpan nilai teks

Angka dibelakang tipe data merupakan maksimal 10 digit atau karakter yang dapat diisikan pada kolom tersebut.

Table dengan *Default Value*

pada tabel mahasiswa berikut *field* nilai diisi dengan nilai *default* yang berupa 0. Nantinya saat menambah record baru, apabila field nilai tidak diisi maka secara otomatis isinya adalah 0.

```
CREATE TABLE mahasiswa (  
  nim int(10) NOT NULL,  
  nama varchar(100),  
  nilai int default 0  
);
```

Table dengan *NOT NULL Value*

pada tabel mahasiswa diatas, NOT NULL mengindikasikan bahwa *field* tersebut tidak boleh kosong/ tidak berisi. Berbeda apabila tidak ada NOT NULL misalnya nama, bila tidak diisi nilainya secara eksplisit bernilai NULL.

Table dengan *Constraint*

Constraint pada suatu table mendefinisikan aturan-aturan yang membatasi suatu field. Ada beberapa constraint antara lain:

a. UNIQUE

Fungsi UNIQUE untuk menjaga suatu field pada suatu tabel tidak boleh berisi nilai yang sama satu sama lain (duplikat). Namun NULL diperbolehkan menjadi nilai suatu *field*. berikut beberapa syntax untuk memberikan constraint UNIQUE pada tabel.

```
CREATE TABLE mahasiswa (  
  id INT NOT NULL UNIQUE  
);
```

```
CREATE TABLE mahasiswa (  
  id INT NOT NULL,  
  UNIQUE (id)  
);
```

b. PRIMARY KEY

Fungsinya menjaga suatu *field* tidak boleh berisi sama dan tidak boleh berisi NULL. Fungsi dari *primary key* adalah pembeda antara satu record dengan record yang lainnya atau bisa dikatakan suatu identifier.

```
CREATE TABLE mahasiswa (  
  id int not null,  
  nim varchar(10),  
  nilai int,  
  primary key (id)  
)
```

c. REFERENTIAL

Dapat dilakukan pembuatan relasi antar tabel. Tabel yang dirujuk memiliki *primary key*, sedangkan tabel yang merujuk memiliki *foreign key*, dimana *primary key* dan *foreign key* memiliki domain nilai yang sama.

```
CREATE TABLE prodi (  
  id_prodi int primary key,  
  nama varchar(100)  
);  
  
CREATE TABLE mahasiswa (  
  id_mahasiswa int not null auto_increment primary key,  
  nama varchar(100),  
  nim varchar(10) not null,
```

```
id_prodi int,  
foreign key (id_prodi) references prodi(id_prodi)  
);
```

Pada contoh diatas, kolom id_prodi tabel mahasiswa akan merujuk ke primary key id_prodi pada tabel prodi. Nilai kolom pada foreign key id_prodi tabel mahasiswa harus ada nilainya di primary key id_prodi tabel prodi.

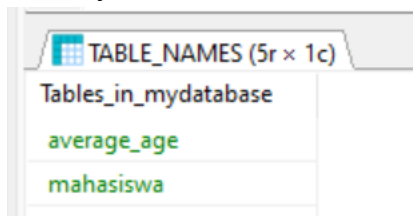
Apabila ingin menambahkan foreign key pastikan tabel yang dirujuk sudah tersedia sebelum membuat references. Seperti contoh diatas pastikan tabel prodi sudah dibuat sebelum tabel mahasiswa merujuknya di foreign key.

2.2 SHOW

Setelah membuat database dan table selanjutnya kita akan menampilkan isi dari tabel yang telah dibuat sebelumnya dengan perintah show atau bisa juga dengan perintah describe. Syntax **SHOW TABLES;**

```
SHOW TABLES;
```

Hasilnya :



Tables_in_mydatabase
average_age
mahasiswa

Untuk melihat deskripsi tabel bisa dengan query, menggunakan syntax **DESCRIBE mahasiswa;**

Contoh:

```
DESCRIBE mahasiswa;
```

Hasil:



Field	Type	Null	Key	Default	Extra
nim	int(10)	NO	PRI	(NULL)	
nama	varchar(100)	YES		(NULL)	
prodi	varchar(100)	YES		(NULL)	
alamat	varchar(200)	YES		(NULL)	

2.3 ALTER

Perintah ALTER Untuk merubah struktur tabel seperti menambah, merubah, menghapus kolom

Menambah Kolom Tabel

Syntax:

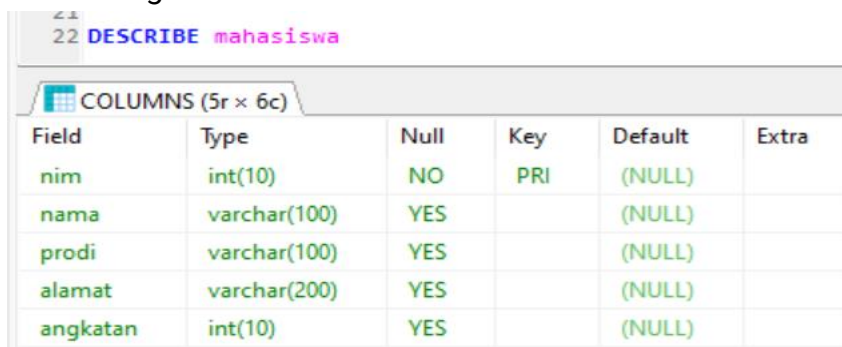
```
ALTER TABLE nama_table  
ADD nama_field tipe_data
```

Contoh :

```
ALTER TABLE mahasiswa  
ADD angkatan INT(10)
```

Hasil :

Atribut angkatan telah ditambahkan



22 DESCRIBE mahasiswa

Field	Type	Null	Key	Default	Extra
nim	int(10)	NO	PRI	(NULL)	
nama	varchar(100)	YES		(NULL)	
prodi	varchar(100)	YES		(NULL)	
alamat	varchar(200)	YES		(NULL)	
angkatan	int(10)	YES		(NULL)	

Modifikasi Kolom Tabel

Format :

```
ALTER TABLE nama_table  
MODIFY nama_field tipe_data
```

Contoh:

Mengubah tipe data dari field nama dari varchar (100) menjadi varchar (200)

```
ALTER TABLE mahasiswa  
MODIFY nama VARCHAR(200)
```

Hasil:

Type data nama telah berubah menjadi varchar (200)

25 `DESCRIBE mahasiswa`

COLUMNS (5r × 6c)					
Field	Type	Null	Key	Default	Extra
nim	int(10)	NO	PRI	(NULL)	
nama	varchar(200)	YES		(NULL)	
prodi	varchar(100)	YES		(NULL)	
alamat	varchar(200)	YES		(NULL)	
angkatan	int(10)	YES		(NULL)	

Menghapus Kolom Tabel

Format:

`ALTER TABLE nama_table
DROP nama_field`

Contoh:

Menghapus atribut angkatan

`ALTER TABLE mahasiswa DROP angkatan;`

Hasil :

28 `DESCRIBE mahasiswa`

COLUMNS (4r × 6c)					
Field	Type	Null	Key	Default	Extra
nim	int(10)	NO	PRI	(NULL)	
nama	varchar(200)	YES		(NULL)	
prodi	varchar(100)	YES		(NULL)	
alamat	varchar(200)	YES		(NULL)	

2.4 DROP

Perintah DROP digunakan untuk menghapus tabel atau database

Menghapus Table

Format:

`DROP TABLE nama_table`

Contoh:

`DROP TABLE mahasiswa`

Menghapus Database

Format:

```
DROP DATABASE nama_database
```

Contoh:

```
DROP DATABASE mydatabase;
```

TUGAS PRAKTIKUM

1. Dengan menggunakan database pada bab ini (database yang dibuat saat praktikum). Buatlah suatu tabel dengan nama “offices” dengan atribut sebagai berikut:
 - officeCode , tipe data varchar dengan panjang 10 karakter dan tidak boleh kosong
 - city, type data varchar dengan panjang 50 karakter dan tidak boleh kosong
 - phone, type data varchar dengan panjang 50 karakter dan tidak boleh kosong
 - addresline1, type data varchar dengan panjang 50 karakter dan tidak boleh kosong
 - addresline2, type data varchar dengan panjang 50 karakter
 - state, type data varchar dengan panjang 50 karakter
 - country, type data varchar dengan panjang 50 karakter dan tidak boleh kosong
 - Primary key nya adalah office code
2. Tampilkan isi tabel pada soal sebelumnya dengan menggunakan perintah describe
3. Modifikasi field phone menjadi int (20) dengan perintah alter
4. Hapus field addressline2 dengan perintah alter
5. Buatlah sebuah database baru dengan nama db_praktikum, kemudian Tuliskan query untuk membuat ERD seperti dibawah ini



Dengan Ketentuan :

- Nama Kolom dan Tipe Data harus sama persis dengan yang di ERD, kecuali jika versi mysql yang belum mendukung atau sintaks berbeda.
- Harus ada relasi antar kolom seperti pada ERD
- Setiap id pada tabel harus bertambah sendiri apabila kita menambah data baru,
- (tabel mahasiswa) nilai dari kolom nim harus unique (tidak boleh sama satu sama lain), dan tidak boleh kosong
- (tabel pinjam) apabila kolom status_pengembalian kosong maka secara otomatis valuenya akan 0

BAB 3 | QUERY DASAR MYSQL

Pada Bab 3 ini akan dipelajari tentang *query-query* MYSQL dasar, yang mencakup cara mendapatkan data dari database, bagaimana menyeleksi data yang didapatkan, bagaimana cara mengurutkan data, dll.

Database yang akan digunakan untuk melakukan *query* adalah database *classicmodels* yang telah digunakan pada pertemuan sebelumnya.

3.1 SELECT

Perintah SELECT merupakan perintah untuk mengambil atau mendapatkan data dari database yang dinamakan *record*. Perintah SELECT wajib ada pada setiap *query* yang bertujuan untuk mendapatkan data. Berikut *syntax* dari SELECT:

```
SELECT nama_kolom FROM nama_tabel
```

Misalnya pada tabel *offices*.

officeCode	city	phone	addressLine1	addressLi...	state	country	postalCode	territory
1	San Francisco	+1 650 219 4782	100 Market Street	Suite 300	CA	USA	94080	NA
2	Boston	+1 215 837 0825	1550 Court Place	Suite 102	MA	USA	02107	NA
3	NYC	+1 212 555 3000	523 East 53rd Street	apt. 5A	NY	USA	10022	NA
4	Paris	+33 14 723 4404	43 Rue Jouffroy D'...	(NULL)	(NULL)	France	75017	EMEA
5	Tokyo	+81 33 224 5000	4-1 Kioicho	(NULL)	Chiyoda-Ku	Japan	102-8578	Japan
6	Sydney	+61 2 9264 2451	5-11 Wentworth A...	Floor #2	(NULL)	Australia	NSW 2010	APAC
7	London	+44 20 7877 2041	25 Old Broad Street	Level 7	(NULL)	UK	EC2N 1HN	EMEA

Kita ingin melakukan *query* untuk mendapatkan kolom *city* saja pada tabel *customers* tersebut. Berikut *query* dan *output*-nya:

```
SELECT city FROM offices
```

Output:

city
San Francisco
Boston
NYC
Paris
Tokyo
Sydney
London

Untuk menyeleksi lebih dari satu kolom, maka kita bisa memisahkan nama-nama kolom dengan koma. Kita juga bisa memberikan alias pada kolom untuk mengubah nama kolom tersebut. Contoh *query* untuk menampilkan kolom *city* dan *officeCode*:

```
SELECT city, officeCode AS office_id FROM offices
```

Output:

city	📌 office_id
San Francisco	1
Boston	2
NYC	3
Paris	4
Tokyo	5
Sydney	6
London	7

Perhatikan bahwa walaupun pada tabel *customers* urutan kolom *officeCode* berada sebelum kolom *city*, namun pada *output* di atas, kolom yang berada pada urutan pertama adalah kolom *city*, karena pada *query* kolom yang pertama dituliskan merupakan kolom yang pertama atau paling kiri ditampilkan pada output.

Untuk mendapatkan semua kolom dari satu tabel, kita tinggal menuliskan '*' pada statement *SELECT* sebagai pengganti nama kolom.

```
SELECT * FROM offices
```

3.2 WHERE

WHERE digunakan untuk mendapatkan data yang memenuhi kriteria/kondisi tertentu. Klausa WHERE bekerja seperti kondisi *if* dalam bahasa pemrograman apa pun. Klausa ini akan mempengaruhi *output* / hasil keluaran dari perintah *SELECT* di mana hanya baris yang kondisi/kriteria nya yang benar saja yang dikembalikan.

Berikut *syntax* dari perintah *SELECT-WHERE* :

```
SELECT nama_kolom FROM nama_tabel WHERE kondisi
```

Contoh *query* SELECT-WHERE untuk mendapatkan baris apa saja pada tabel *offices* yang berasal dari Paris (kolom *city* bernilai Paris):

```
SELECT * FROM offices WHERE city="Paris"
```

Output:

officeCode	city	phone	addressLine1	addressLine2	state	country	postalCode	territory
4	Paris	+33 14 723 4404	43 Rue Jouffroy	(NULL)	(NULL)	France	75017	EMEA

Berikut daftar operator perbandingan yang dapat digunakan pada klausa WHERE, misalkan A=5 dan B=10:

Operator	Deskripsi	Tipe Data	Output dengan operand A dan B
=	Mengecek jika nilai dari dua <i>operand</i> sama atau tidak, jika sama, maka kondisi akan menjadi true	Numerik, String	(A = B) bernilai false
!= atau <>	Mengecek jika nilai dari dua <i>operand</i> tidak sama, jika tidak sama, maka kondisi akan menjadi true begitupun sebaliknya	Numerik, String	(A != B) bernilai true
>	Mengecek jika nilai dari <i>operand</i> di sebelah kiri lebih besar dari nilai dari <i>operand</i> sebelah kanan, jika iya, maka kondisi akan menjadi true	Numerik	(A > B) bernilai false
<	Mengecek jika nilai dari <i>operand</i> di sebelah kiri lebih kecil dari nilai dari <i>operand</i> sebelah kanan, jika iya, maka kondisi akan menjadi true	Numerik	(A < B) bernilai true
>=	Mengecek jika nilai dari <i>operand</i> di sebelah kiri lebih besar atau sama dengan nilai dari <i>operand</i> sebelah kanan, jika iya, maka kondisi akan menjadi true	Numerik	(A >= B) bernilai true
<=	Mengecek jika nilai dari <i>operand</i> di sebelah kiri lebih kecil atau sama dengan nilai dari <i>operand</i>	Numerik	(A <= B) bernilai

	sebelah kanan, jika iya, maka kondisi akan menjadi true		true
--	---	--	------

Terdapat jenis operator lain untuk kondisi klausa WHERE selain jenis operator perbandingan, yaitu operator logika.

3.3 ORDER BY

Klausa ORDER BY digunakan untuk mengurutkan data berdasarkan satu atau lebih kolom, jika kolom tersebut bertipe data *String* maka akan diurutkan berdasarkan alphabet (A-Z atau Z-A). Secara *default* data diurutkan secara menaik (*ascending*) , namun kita juga bisa mengurutkan data secara menurun (*descending*).

Berikut *syntax* SELECT-ORDER BY :

```
SELECT * FROM nama_tabel ORDER BY nama_kolom_urut
```

Untuk menggunakan jenis urutan tertentu, kita bisa menambahkan keyword ASC (untuk menurun) atau DESC (untuk menaik) di belakang nama_kolom_urut. Misalkan kita ingin mendapatkan semua data pada tabel *offices* tapi datanya diurutkan secara menurun berdasarkan kolom *officeCode* :

```
SELECT * FROM offices ORDER BY officeCode DESC
```

Output:

officeCode	city	phone	addressLine1	addressLine2	state	country	postalCode	territory
7	London	+44 20 7877 2041	25 Old Broad Street	Level 7	(NULL)	UK	EC2N 1HN	EMEA
6	Sydney	+61 2 9264 2451	5-11 Wentworth Avenue	Floor #2	(NULL)	Australia	NSW 2010	APAC
5	Tokyo	+81 33 224 5000	4-1 Kioicho	(NULL)	Chiyoda-Ku	Japan	102-8578	Japan
4	Paris	+33 14 723 4404	43 Rue Jouffroy D'abbans	(NULL)	(NULL)	France	75017	EMEA
3	NYC	+1 212 555 3000	523 East 53rd Street	apt. 5A	NY	USA	10022	NA
2	Boston	+1 215 837 0825	1550 Court Place	Suite 102	MA	USA	02107	NA
1	San Franci...	+1 650 219 4782	100 Market Street	Suite 300	CA	USA	94080	NA

3.4 LIMIT

Klausa LIMIT digunakan untuk membatasi jumlah baris dari data yang didapatkan. *syntax* SELECT-LIMIT :

```
SELECT nama_kolom FROM nama_tabel LIMIT jumlah_baris
```

Contoh *query* untuk mendapatkan 5 data pertama dari tabel *offices*:

```
SELECT * FROM offices LIMIT 5
```

Output:

officeCode	city	phone	addressLine1	addressLine2	state	country	postalCode	territory
1	San Francisco	+1 650 219 4782	100 Market Street	Suite 300	CA	USA	94080	NA
2	Boston	+1 215 837 0825	1550 Court Place	Suite 102	MA	USA	02107	NA
3	NYC	+1 212 555 3000	523 East 53rd Street	apt. 5A	NY	USA	10022	NA
4	Paris	+33 14 723 4404	43 Rue Joffroy D'abbans	(NULL)	(NULL)	France	75017	EMEA
5	Tokyo	+81 33 224 5000	4-1 Kioicho	(NULL)	Chiyoda-Ku	Japan	102-8578	Japan

Pada klausa LIMIT kita juga bisa mendefinisikan *offset* atau dari baris mana klausa LIMIT akan dijalankan. *syntax* dari SELECT-LIMIT dengan *offset*:

```
SELECT nama_kolom FROM nama_tabel LIMIT offset, jumlah_baris
```

Contoh *query* untuk mendapatkan 5 data dari tabel *offices* dari baris 3 (*offset* = 3-1= 2):

```
SELECT * FROM offices LIMIT 2,5
```

Output:

officeCode	city	phone	addressLine1	addressLine2	state	country	postalCode	territory
3	NYC	+1 212 555 3000	523 East 53rd Street	apt. 5A	NY	USA	10022	NA
4	Paris	+33 14 723 4404	43 Rue Joffroy D'abbans	(NULL)	(NULL)	France	75017	EMEA
5	Tokyo	+81 33 224 5000	4-1 Kioicho	(NULL)	Chiyoda-Ku	Japan	102-8578	Japan
6	Sydney	+61 2 9264 2451	5-11 Wentworth Avenue	Floor #2	(NULL)	Australia	NSW 2010	APAC
7	London	+44 20 7877 2041	25 Old Broad Street	Level 7	(NULL)	UK	EC2N 1HN	EMEA

Output dari *query* tersebut tetap membatasi lima baris namun dimulai dari baris ke-3.

3.5 DISTINCT

Klausa DISTINCT digunakan untuk mendapatkan data yang unik dari suatu kolom, maksudnya jika pada satu kolom terdapat nilai yang identik atau sama, maka hanya akan diambil satu nilai dan sisanya diabaikan. *syntax* dari DISTINCT:

```
SELECT DISTINCT nama_kolom FROM nama_tabel
```


Misalnya kita ingin mendapatkan *jobTitle* apa saja yang tersedia pada tabel *employees* berikut:

🔑 employeeNumber	lastName	firstName	extension	email	📍 officeCode	📍 reportsTo	jobTitle
1,002	Murphy	Diane	x5800	dmurphy@classicmodelcars.com	1	(NULL)	President
1,056	Patterson	Mary	x4611	mpatterso@classicmodelcars.com	1	1,002	VP Sales
1,076	Firrelli	Jeff	x9273	jfirrelli@classicmodelcars.com	1	1,002	VP Marketing
1,088	Patterson	William	x4871	wpatterson@classicmodelcars.com	6	1,056	Sales Manager (APAC)
1,102	Bondur	Gerard	x5408	gbondur@classicmodelcars.com	4	1,056	Sale Manager (EMEA)
1,143	Bow	Anthony	x5428	abow@classicmodelcars.com	1	1,056	Sales Manager (NA)
1,165	Jennings	Leslie	x3291	ljennings@classicmodelcars.com	1	1,143	Sales Rep
1,166	Thompson	Leslie	x4065	lthompson@classicmodelcars.com	1	1,143	Sales Rep
1,188	Firrelli	Julie	x2173	jfirrelli@classicmodelcars.com	2	1,143	Sales Rep
1,216	Patterson	Steve	x4334	spatterson@classicmodelcars.com	2	1,143	Sales Rep

Query:

```
SELECT DISTINCT jobTitle FROM employees
```

Output:

jobTitle
President
VP Sales
VP Marketing
Sales Manager (APAC)
Sale Manager (EMEA)
Sales Manager (NA)
Sales Rep

TUGAS PRAKTIKUM

1. Tampilkan semua kolom dari tabel *offices* yang berasal dari kota 'San Francisco'.
2. Tampilkan semua kolom dari tabel *orderdetails*, di mana *quantityOrdered* lebih besar dari 70, diurutkan secara menaik berdasarkan *orderLineNumber*.
3. Tampilkan *productLine* apa saja yang tersedia pada tabel *products*.
4. Tampilkan *customerNumber* dan *customerName* pada tabel *customers* dengan *customerNumber* di kisaran 100-150. Jangan lupa berikan alias pada kolom keluaran.
5. Berikan Data *customers* yang bukan dari *USA* yang tidak bisa lagi menarik uang dari rekeningnya. Urutkan berdasarkan *customerName* secara menaik lalu berikan data dari index (inklusif) 10 sampai 19.

BAB 4 | DATA MANIPULATION LANGUAGE (DML)

Data Manipulation Language (DML) merupakan perintah-perintah yang digunakan dalam *query* untuk melakukan perubahan pada data di database. Secara umum, DML terdiri atas tiga buah perintah, yaitu INSERT, UPDATE, DELETE.

4.1.INSERT

INSERT merupakan perintah yang digunakan dalam *query* untuk memasukan data ke dalam suatu tabel dalam database. Secara umum, *syntax* untuk INSERT adalah seperti berikut.

```
INSERT INTO nama_tabel (kolom1, kolom2, kolom3, ...)
VALUES (nilai1, nilai2, nilai3, ...);
```

Sebagai contoh, kita akan mencoba memasukkan data baru ke dalam tabel *employees* pada database *Classic Models*.

employeeNumber	lastName	firstName	extension	email	officeCode	reportsTo	jobTitle
1.188	Firrelli	Julie	x2173	jfirrelli@classicmodelcars.com	2	1.143	Sales Rep
1.216	Patterson	Steve	x4334	spatterson@classicmodelcars.com	2	1.143	Sales Rep
1.286	Tseng	Foon Yue	x2248	ftseng@classicmodelcars.com	3	1.143	Sales Rep
1.323	Vanauf	George	x4102	gvanauf@classicmodelcars.com	3	1.143	Sales Rep
1.337	Bondur	Loui	x6493	lbondur@classicmodelcars.com	4	1.102	Sales Rep
1.370	Hernandez	Gerard	x2028	ghernande@classicmodelcars.com	4	1.102	Sales Rep
1.401	Castillo	Pamela	x2759	pcastillo@classicmodelcars.com	4	1.102	Sales Rep
1.501	Bott	Larry	x2311	lbott@classicmodelcars.com	7	1.102	Sales Rep
1.504	Jones	Barry	x102	bjones@classicmodelcars.com	7	1.102	Sales Rep
1.611	Fixter	Andy	x101	afixter@classicmodelcars.com	6	1.088	Sales Rep
1.612	Marsh	Peter	x102	pmarsh@classicmodelcars.com	6	1.088	Sales Rep
1.619	King	Tom	x103	tking@classicmodelcars.com	6	1.088	Sales Rep
1.621	Nishi	Mami	x101	mnishi@classicmodelcars.com	5	1.056	Sales Rep
1.625	Kato	Yoshimi	x102	ykato@classicmodelcars.com	5	1.621	Sales Rep
1.702	Gerard	Martin	x2312	mgerard@classicmodelcars.com	4	1.102	Sales Rep

Kemudian, untuk memasukkan datanya ke dalam tabel, maka dapat digunakan *query* seperti berikut :

```
INSERT INTO employees (employeeNumber, lastName, firstName, extension, email,
officeCode, reportsTo, jobTitle )
VALUE (2705, 'Jake', 'Finch', 'x3227', 'jakefinch@gmail.com', 1, 1102, 'Sales Rep');
```

Setelah *query* dijalankan, maka data akan ditambahkan ke dalam tabel *employees*. Hasilnya seperti berikut :

employeeNumber	lastName	firstName	extension	email	officeCode	reportsTo	jobTitle
1.337	Bondur	Loui	x6493	lbondur@classicmodelcars.com	4	1.102	Sales Rep
1.370	Hernandez	Gerard	x2028	ghernande@classicmodelcars.com	4	1.102	Sales Rep
1.401	Castillo	Pamela	x2759	pcastillo@classicmodelcars.com	4	1.102	Sales Rep
1.501	Bott	Larry	x2311	lbott@classicmodelcars.com	7	1.102	Sales Rep
1.504	Jones	Barry	x102	bjones@classicmodelcars.com	7	1.102	Sales Rep
1.611	Fixter	Andy	x101	afixter@classicmodelcars.com	6	1.088	Sales Rep
1.612	Marsh	Peter	x102	pmarsh@classicmodelcars.com	6	1.088	Sales Rep
1.619	King	Tom	x103	tking@classicmodelcars.com	6	1.088	Sales Rep
1.621	Nishi	Mami	x101	mnishi@classicmodelcars.com	5	1.056	Sales Rep
1.625	Kato	Yoshimi	x102	ykato@classicmodelcars.com	5	1.621	Sales Rep
1.702	Gerard	Martin	x2312	mgerard@classicmodelcars.com	4	1.102	Sales Rep
2.705	Jake	Finch	x3227	jakefinch@gmail.com	1	1.102	Sales Rep

4.1.1 INSERT Tanpa Memasukkan Nama Kolom

Cara ini hanya digunakan apabila data yang akan dimasukkan memiliki nilai untuk semua kolom pada tabel, sehingga kolom-kolom tabel tidak perlu didefinisikan dalam *query*. Berikut contohnya :

```
INSERT INTO employees  
VALUE (2706, 'Jack', 'Jill', 'x3322', 'jjill@gmail.com', '1', 2705, 'Sales Rep');
```

Setelah dijalankan, maka data akan masuk ke tabel *employees* dan hasilnya seperti berikut:

employeeNumber	lastName	firstName	extension	email	officeCode	reportsTo	jobTitle
1.370	Hernandez	Gerard	x2028	ghernande@classicmodelcars.com	4	1.102	Sales Rep
1.401	Castillo	Pamela	x2759	pcastillo@classicmodelcars.com	4	1.102	Sales Rep
1.501	Bott	Larry	x2311	lbott@classicmodelcars.com	7	1.102	Sales Rep
1.504	Jones	Barry	x102	bjones@classicmodelcars.com	7	1.102	Sales Rep
1.611	Fixter	Andy	x101	afixter@classicmodelcars.com	6	1.088	Sales Rep
1.612	Marsh	Peter	x102	pmarsh@classicmodelcars.com	6	1.088	Sales Rep
1.619	King	Tom	x103	tking@classicmodelcars.com	6	1.088	Sales Rep
1.621	Nishi	Mami	x101	mnishi@classicmodelcars.com	5	1.056	Sales Rep
1.625	Kato	Yoshimi	x102	ykato@classicmodelcars.com	5	1.621	Sales Rep
1.702	Gerard	Martin	x2312	mgerard@classicmodelcars.com	4	1.102	Sales Rep
2.705	Jake	Finch	x3227	jakefinch@gmail.com	1	1.102	Sales Rep
2.706	Jack	Jill	x3322	jjill@gmail.com	1	2.705	Sales Rep

4.1.2 INSERT Data Dengan Nilai NULL atau Tanpa Nilai

Apabila dalam data yang akan dimasukkan terdapat beberapa nilai *NULL*, maka ada dua cara yang dapat digunakan untuk memasukkan datanya. Cara tersebut antara lain secara implisit dan secara eksplisit.

Jika dilakukan secara implisit, maka di dalam *query*, cukup kolom yang memiliki nilai saja yang dimasukkan. Berikut contoh penggunaannya :

```
INSERT INTO employees (employeeNumber, firstName, extension, officeCode,  
reportsTo, jobTitle)  
VALUE (2707, 'Derek', 'x3219', '7', '1002', 'Janitor');
```

Setelah dijalankan, maka data pada tabel *employees* akan bertambah dan hasilnya seperti berikut :

employeeNumber	lastName	firstName	extension	email	officeCode	reportsTo	jobTitle
1.401	Castillo	Pamela	x2759	pcastillo@classicmodelcars.com	4	1.102	Sales Rep
1.501	Bott	Larry	x2311	lbott@classicmodelcars.com	7	1.102	Sales Rep
1.504	Jones	Barry	x102	bjones@classicmodelcars.com	7	1.102	Sales Rep
1.611	Fixter	Andy	x101	afixter@classicmodelcars.com	6	1.088	Sales Rep
1.612	Marsh	Peter	x102	pmarsh@classicmodelcars.com	6	1.088	Sales Rep
1.619	King	Tom	x103	tking@classicmodelcars.com	6	1.088	Sales Rep
1.621	Nishi	Mami	x101	mnishi@classicmodelcars.com	5	1.056	Sales Rep
1.625	Kato	Yoshimi	x102	ykato@classicmodelcars.com	5	1.621	Sales Rep
1.702	Gerard	Martin	x2312	mgerard@classicmodelcars.com	4	1.102	Sales Rep
2.705	Jake	Finch	x3227	jakefinch@gmail.com	1	1.102	Sales Rep
2.706	Jack	Jill	x3322	jjill@gmail.com	1	2.705	Sales Rep
2.707	(NULL)	Derek	x3219	(NULL)	7	1.002	Janitor

Note : Jika kolom pada tabel tidak memperbolehkan nilai *NULL*, maka cara diatas hanya akan menghasilkan blank. Nilai *NULL* dan blank merupakan dua nilai yang berbeda

Selain cara diatas, kita juga dapat secara eksplisit memberikan nilai *NULL* untuk kolom tertentu pada tabel. Caranya adalah dengan mendefinisikan kembali seluruh

tabel, lalu memberikan nilai NULL pada kolom yang diinginkan. Berikut contoh penggunaannya :

```
INSERT INTO employees (employeeNumber, lastName, firstName, extension, email, officeCode, reportsTo, jobTitle) VALUE (2708, NULL, 'Dan', 'x3219', 'dan@gmail.com', '7', '1002', 'Janitor');
```

Setelah dijalankan, maka data pada tabel akan bertambah dan berikut hasilnya :

employeeNumber	lastName	firstName	extension	email	officeCode	reportsTo	jobTitle
1.501	Bott	Larry	x2311	lbott@classicmodelcars.com	7	1.102	Sales Rep
1.504	Jones	Barry	x102	bjones@classicmodelcars.com	7	1.102	Sales Rep
1.611	Fixter	Andy	x101	afixter@classicmodelcars.com	6	1.088	Sales Rep
1.612	Marsh	Peter	x102	pmarsh@classicmodelcars.com	6	1.088	Sales Rep
1.619	King	Tom	x103	tking@classicmodelcars.com	6	1.088	Sales Rep
1.621	Nishi	Mami	x101	mnishi@classicmodelcars.com	5	1.056	Sales Rep
1.625	Kato	Yoshimi	x102	ykato@classicmodelcars.com	5	1.621	Sales Rep
1.702	Gerard	Martin	x2312	mgerard@classicmodelcars.com	4	1.102	Sales Rep
2.705	Jake	Finch	x3227	jakefinch@gmail.com	1	1.102	Sales Rep
2.706	Jack	Jill	x3322	jjill@gmail.com	1	2.705	Sales Rep
2.707	(NULL)	Derek	x3219	(NULL)	7	1.002	Janitor
2.708	(NULL)	Dan	x3219	dan@gmail.com	7	1.002	Janitor

Note : Jika kolom tidak memperbolehkan nilai NULL, maka cara diatas akan menghasilkan error

4.2. UPDATE

UPDATE merupakan perintah dalam *query* yang digunakan untuk memperbaharui/mengubah data yang sudah ada dalam database. Secara umum, *syntax* untuk UPDATE adalah seperti berikut :

```
UPDATE nama_tabel
SET kolom1 = nilai, kolom2 = nilai2, ...
WHERE kondisi;
```

4.2.1 UPDATE Satu Data pada Tabel

Jika ingin melakukan UPDATE atau perubahan pada salah satu data di tabel, maka di dalam *query* kondisi yang perlu digunakan adalah *primary* key atau kolom yang sifatnya unik. Sebagai contoh, kita akan mencoba memperbaharui data yang sudah ada pada tabel *employees* sebelumnya dengan *employeeNumber* sebagai *primary* key atau kolom unik. Berikut beberapa data yang sudah ada di tabel tersebut.

employeeNumber	lastName	firstName	extension	email	officeCode	reportsTo	jobTitle
1.501	Bott	Larry	x2311	lbott@classicmodelcars.com	7	1.102	Sales Rep
1.504	Jones	Barry	x102	bjones@classicmodelcars.com	7	1.102	Sales Rep
1.611	Fixter	Andy	x101	afixter@classicmodelcars.com	6	1.088	Sales Rep
1.612	Marsh	Peter	x102	pmarsh@classicmodelcars.com	6	1.088	Sales Rep
1.619	King	Tom	x103	tking@classicmodelcars.com	6	1.088	Sales Rep
1.621	Nishi	Mami	x101	mnishi@classicmodelcars.com	5	1.056	Sales Rep
1.625	Kato	Yoshimi	x102	ykato@classicmodelcars.com	5	1.621	Sales Rep
1.702	Gerard	Martin	x2312	mgerard@classicmodelcars.com	4	1.102	Sales Rep
2.705	Jake	Finch	x3227	jakefinch@gmail.com	1	1.102	Sales Rep
2.706	Jack	Jill	x3322	jjill@gmail.com	1	2.705	Sales Rep
2.707	(NULL)	Derek	x3219	(NULL)	7	1.002	Janitor
2.708	(NULL)	Dan	x3219	dan@gmail.com	7	1.002	Janitor

Kemudian, kita akan mencoba memperbaharui data yang memiliki nilai *employeeNumber* berupa 2706. Maka *query* dapat dituliskan seperti berikut.

```
UPDATE employees
SET officeCode = '2', reportsTo = 1002
WHERE employeeNumber = 2706;
```

Setelah dijalankan, maka data dengan *employeeNumber* berupa 2706 akan diperbaharui dan hasilnya seperti berikut.

employeeNumber	lastName	firstName	extension	email	officeCode	reportsTo	jobTitle
1.501	Bott	Larry	x2311	lbott@classicmodelcars.com	7	1.102	Sales Rep
1.504	Jones	Barry	x102	bjones@classicmodelcars.com	7	1.102	Sales Rep
1.611	Fixter	Andy	x101	afixter@classicmodelcars.com	6	1.088	Sales Rep
1.612	Marsh	Peter	x102	pmarsh@classicmodelcars.com	6	1.088	Sales Rep
1.619	King	Tom	x103	tking@classicmodelcars.com	6	1.088	Sales Rep
1.621	Nishi	Mami	x101	mnishi@classicmodelcars.com	5	1.056	Sales Rep
1.625	Kato	Yoshimi	x102	ykato@classicmodelcars.com	5	1.621	Sales Rep
1.702	Gerard	Martin	x2312	mgerard@classicmodelcars.com	4	1.102	Sales Rep
2.705	Jake	Finch	x3227	jakefinch@gmail.com	1	1.102	Sales Rep
2.706	Jack	Jill	x3322	jjill@gmail.com	2	1.002	Sales Rep
2.707	(NULL)	Derek	x3219	(NULL)	7	1.002	Janitor
2.708	(NULL)	Dan	x3219	dan@gmail.com	7	1.002	Janitor

4.2.2 UPDATE Lebih Dari Satu Data Secara Bersamaan

Selain cara sebelumnya, kita juga dapat mengubah lebih dari satu data secara bersamaan. Untuk cara ini, kondisi yang digunakan adalah kolom yang tidak bersifat unik atau memiliki beberapa nilai yang sama dalam satu kolom. Misalnya, kita ingin mengubah data *officeCode* semua data yang memiliki *jobTitle* berupa 'Janitor'. Maka, untuk melakukan hal tersebut dapat digunakan *query* seperti berikut.

```
UPDATE employees
SET officeCode = '6'
WHERE jobTitle = 'Janitor';
```

Setelah dijalankan, maka setiap data yang memiliki *jobTitle* berupa 'Janitor' akan mengalami perubahan pada kolom *officeCode*. Berikut hasilnya.

employeeNumber	lastName	firstName	extension	email	officeCode	reportsTo	jobTitle
1.501	Bott	Larry	x2311	lbott@classicmodelcars.com	7	1.102	Sales Rep
1.504	Jones	Barry	x102	bjones@classicmodelcars.com	7	1.102	Sales Rep
1.611	Fixter	Andy	x101	afixter@classicmodelcars.com	6	1.088	Sales Rep
1.612	Marsh	Peter	x102	pmarsh@classicmodelcars.com	6	1.088	Sales Rep
1.619	King	Tom	x103	tking@classicmodelcars.com	6	1.088	Sales Rep
1.621	Nishi	Mami	x101	mnishi@classicmodelcars.com	5	1.056	Sales Rep
1.625	Kato	Yoshimi	x102	ykato@classicmodelcars.com	5	1.621	Sales Rep
1.702	Gerard	Martin	x2312	mgerard@classicmodelcars.com	4	1.102	Sales Rep
2.705	Jake	Finch	x3227	jakefinch@gmail.com	1	1.102	Sales Rep
2.706	Jack	Jill	x3322	jjill@gmail.com	2	1.002	Sales Rep
2.707	(NULL)	Derek	x3219	(NULL)	6	1.002	Janitor
2.708	(NULL)	Dan	x3219	dan@gmail.com	6	1.002	Janitor

4.2.3 UPDATE Seluruh Data Secara Bersamaan

UPDATE juga dapat dilakukan pada seluruh data dalam suatu tabel. Untuk melakukannya, cukup menghilangkan kondisi atau WHERE pada *query* UPDATE. Misalkan kita ingin mengubah *officeCode* pada tabel *employees* agar memiliki nilai yang sama. Maka *query* yang dapat digunakan seperti berikut.

```
UPDATE employees
SET officeCode = '1';
```

Setelah dijalankan, maka seluruh data pada tabel *employees* akan memiliki nilai '1' pada kolom *officeCode*. Hasilnya seperti berikut.

employeeNumber	lastName	firstName	extension	email	officeCode	reportsTo	jobTitle
1.501	Bott	Larry	x2311	lbott@classicmodelcars.com	1	1.102	Sales Rep
1.504	Jones	Barry	x102	bjones@classicmodelcars.com	1	1.102	Sales Rep
1.611	Fixter	Andy	x101	afixter@classicmodelcars.com	1	1.088	Sales Rep
1.612	Marsh	Peter	x102	pmarsh@classicmodelcars.com	1	1.088	Sales Rep
1.619	King	Tom	x103	tking@classicmodelcars.com	1	1.088	Sales Rep
1.621	Nishi	Mami	x101	mnishi@classicmodelcars.com	1	1.056	Sales Rep
1.625	Kato	Yoshimi	x102	ykato@classicmodelcars.com	1	1.621	Sales Rep
1.702	Gerard	Martin	x2312	mgerard@classicmodelcars.com	1	1.102	Sales Rep
2.705	Jake	Finch	x3227	jakefinch@gmail.com	1	1.102	Sales Rep
2.706	Jack	Jill	x3322	jjill@gmail.com	1	1.002	Sales Rep
2.707	(NULL)	Derek	x3219	(NULL)	1	1.002	Janitor
2.708	(NULL)	Dan	x3219	dan@gmail.com	1	1.002	Janitor

4.3. DELETE

DELETE merupakan perintah dalam *query* yang digunakan untuk menghapus data dari suatu tabel. Secara umum, *syntax* untuk perintah DELETE adalah sebagai berikut :

```
DELETE FROM nama_tabel WHERE kolom = nilai;
```

4.3.1 DELETE Satu Data

Sama seperti perintah UPDATE, apabila ingin menghapus salah satu data pada tabel, maka kolom yang digunakan pada kondisinya adalah kolom yang merupakan *primary key* atau yang bersifat unik. Misalnya, dalam tabel *employees*, kita ingin menghapus data dengan *employeeNumber* berupa 2706. Maka *query* yang dapat digunakan adalah seperti berikut.

```
DELETE FROM employees WHERE employeeNumber = '2706';
```

Setelah dijalankan, maka data dengan *employeeNumber* '2706' akan terhapus dari tabel. Berikut hasilnya.

employeeNumber	lastName	firstName	extension	email	officeCode	reportsTo	jobTitle
1.401	Castillo	Pamela	x2759	pcastillo@classicmodelcars.com	1	1.102	Sales Rep
1.501	Bott	Larry	x2311	lbott@classicmodelcars.com	1	1.102	Sales Rep
1.504	Jones	Barry	x102	bjones@classicmodelcars.com	1	1.102	Sales Rep
1.611	Fixter	Andy	x101	afixter@classicmodelcars.com	1	1.088	Sales Rep
1.612	Marsh	Peter	x102	pmarsh@classicmodelcars.com	1	1.088	Sales Rep
1.619	King	Tom	x103	tking@classicmodelcars.com	1	1.088	Sales Rep
1.621	Nishi	Mami	x101	mnishi@classicmodelcars.com	1	1.056	Sales Rep
1.625	Kato	Yoshimi	x102	ykato@classicmodelcars.com	1	1.621	Sales Rep
1.702	Gerard	Martin	x2312	mgerard@classicmodelcars.com	1	1.102	Sales Rep
2.705	Jake	Finch	x3227	jakefinch@gmail.com	1	1.102	Sales Rep
2.707	(NULL)	Derek	x3219	(NULL)	1	1.002	Janitor
2.708	(NULL)	Dan	x3219	dan@gmail.com	1	1.002	Janitor

4.3.2 DELETE Lebih Dari Satu Data

Jika ingin menghapus lebih dari satu data secara bersamaan, maka dalam kondisi, kolom yang digunakan adalah kolom yang tidak bersifat unik. Misalnya, kita ingin menghapus seluruh data dengan *jobTitle* berupa 'Janitor'. Maka, *query* yang dapat digunakan seperti berikut.

```
DELETE FROM employees WHERE jobTitle = 'Janitor';
```

Setelah dijalankan, maka data dengan *jobTitle* = 'Janitor' akan terhapus dari tabel. Berikut hasilnya.

employeeNumber	lastName	firstName	extension	email	officeCode	reportsTo	jobTitle
1.337	Bondur	Loui	x6493	lbondur@classicmodelcars.com	1	1.102	Sales Rep
1.370	Hernandez	Gerard	x2028	ghernande@classicmodelcars.com	1	1.102	Sales Rep
1.401	Castillo	Pamela	x2759	pcastillo@classicmodelcars.com	1	1.102	Sales Rep
1.501	Bott	Larry	x2311	lbott@classicmodelcars.com	1	1.102	Sales Rep
1.504	Jones	Barry	x102	bjones@classicmodelcars.com	1	1.102	Sales Rep
1.611	Fixter	Andy	x101	afixter@classicmodelcars.com	1	1.088	Sales Rep
1.612	Marsh	Peter	x102	pmarsh@classicmodelcars.com	1	1.088	Sales Rep
1.619	King	Tom	x103	tking@classicmodelcars.com	1	1.088	Sales Rep
1.621	Nishi	Mami	x101	mnishi@classicmodelcars.com	1	1.056	Sales Rep
1.625	Kato	Yoshimi	x102	ykato@classicmodelcars.com	1	1.621	Sales Rep
1.702	Gerard	Martin	x2312	mgerard@classicmodelcars.com	1	1.102	Sales Rep
2.705	Jake	Finch	x3227	jakefinch@gmail.com	1	1.102	Sales Rep

4.3.3 DELETE Seluruh Data

Jika ingin menghapus seluruh data, maka cukup menghilangkan WHERE pada *query* DELETE. Misalnya, jika ingin menghapus seluruh data pada tabel *employees*, maka *syntax* yang dapat digunakan adalah seperti berikut.

```
DELETE FROM employees;
```

Jika dijalankan, maka seluruh data pada tabel *employees* akan terhapus. Meskipun terhapus, namun struktur tabelnya tetap masih ada.

4.4. Hal Penting dalam DML

Dalam melakukan perintah-perintah DML, perlu diperhatikan bagaimana kolom-kolom dalam tabel tersebut didefinisikan.

- Beberapa kolom mungkin saja tidak memperbolehkan adanya nilai yang sama, sehingga jika melakukan INSERT atau UPDATE dengan nilai yang sudah ada dapat menyebabkan error.
- Kolom dalam tabel juga telah menetapkan tipe data dan jumlah maksimal karakter yang dapat dimasukkan. Jika tipe data suatu nilai tidak sesuai atau panjang karakter nilai tersebut melebihi batas, maka bisa saja terjadi error.
- Terkadang ada beberapa kolom yang tidak memperbolehkan nilai NULL. Sehingga, jika memasukkan data yang memiliki nilai NULL, maka akan terjadi error.
- Ada beberapa kolom yang mungkin merupakan *foreign key* dari kolom lain. *Foreign key* artinya nilai kolom tersebut bergantung kepada nilai yang ada pada kolom di salah satu tabel pada database. Jika nilai yang dimasukkan tidak terdapat dalam kolom asal dari *foreign key*, maka akan terjadi error.
- Jika data memuat nilai yang merupakan *foreign key* di kolom atau tabel yang lain, maka perintah DELETE tidak dapat dieksekusi. Jika ingin menghapus data, pastikan tidak ada *foreign key* yang menggunakan nilai pada data tersebut.

Praktikum

Untuk mengerjakan soal latihan, perhatikan kembali database *classic models* yang telah digunakan sebelumnya.

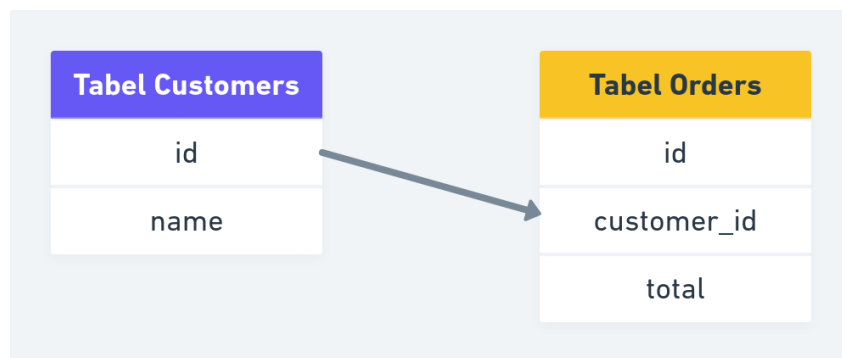
TUGAS PRAKTIKUM

1. Tambahkan 3 orang *employee* baru pada tabel *employees* dalam database *Classic Models* dengan salah satu nama *employee* merupakan nama anda.
2. Tambahkan data baru pada tabel *offices* dan ubahlah nilai *officeCode* pada tabel *employees* dengan data *office* yang baru untuk data yang memiliki nilai *officeCode* = '4' dan *jobTitle* = 'Sales Rep'.
3. Hapuslah data pada tabel *payments* yang memiliki nilai *amount* yang lebih kecil dari 10.000.

BAB 5 | JOIN

Perintah JOIN digunakan untuk menggabungkan dua tabel atau lebih berdasarkan kolom tertentu yang memiliki nilai atau key yang terkait satu sama lain.

Contoh penggunaan JOIN dapat dilihat pada penggabungan tabel Customers dan tabel Orders. Perintah JOIN pada kedua tabel ini dapat dilakukan karena pada tabel Customers terdapat kolom id yang berhubungan dengan kolom id_customer pada tabel Orders.



Terdapat dua tipe JOIN atau penggabungan tabel pada perintah SQL yaitu *Inner Join* dan *Outer Join*.

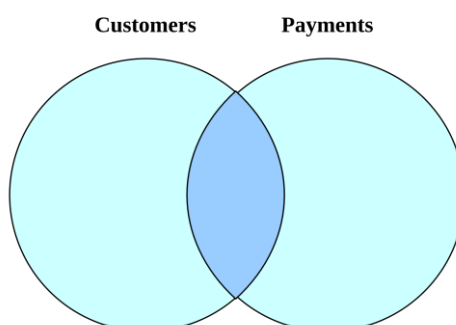
5.1 INNER JOIN

INNER JOIN membandingkan *record* di setiap tabel untuk di cek apakah nilai sama atau tidak. Jika nilai kedua tabel sama, maka akan terbentuk tabel baru yang hanya menampilkan record yang sama dari kedua tabel.

Syntax dari INNER JOIN :

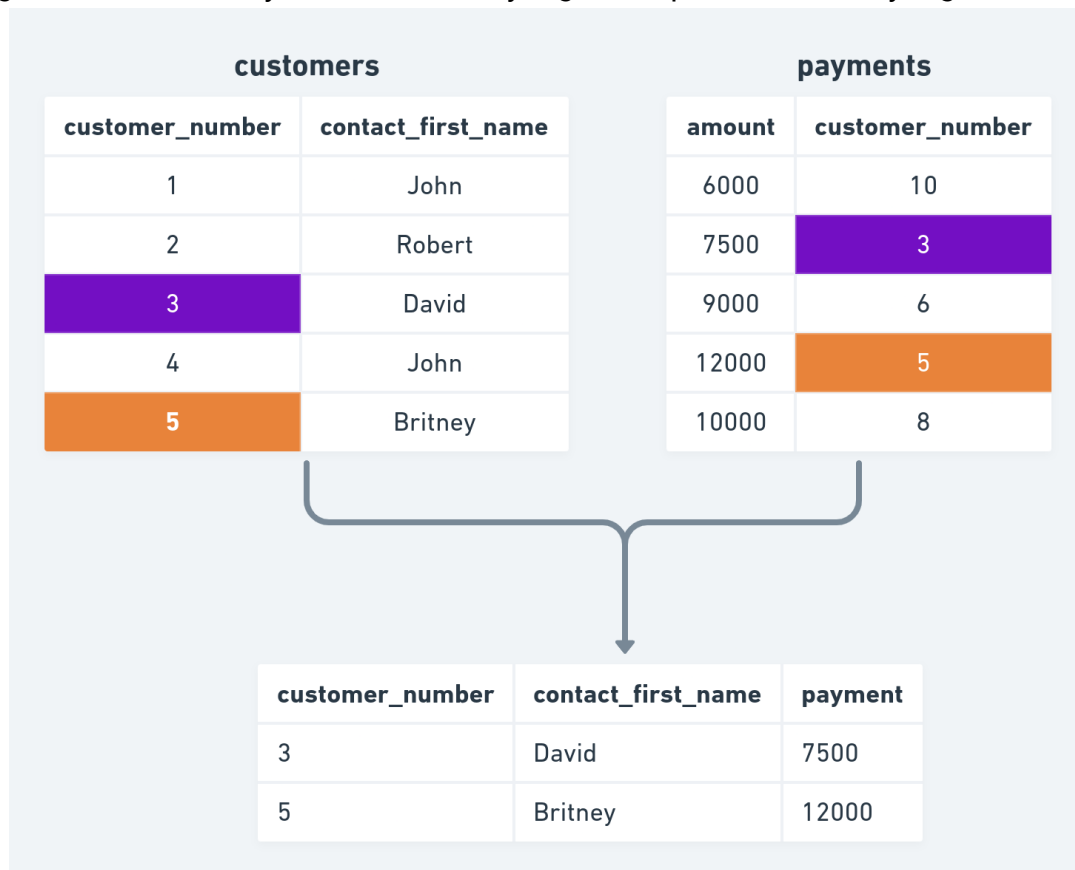
```
SELECT columns
FROM table1
INNER JOIN table2
ON table1.column_name = table2.column_name;
```

Berikut contoh penggunaan INNER JOIN menggunakan database *classicmodels*.



```
SELECT customers.customerNumber, customers.contactFirstName,
payments.amount
FROM customers
INNER JOIN payments
ON customers.customerNumber = payments.customerNumber;
```

Perhatikan kode diatas dapat dilihat bahwa pada perintah SELECT sebelum nama column dituliskan terlebih dahulu nama dari tabel yang digunakan seperti **customers.customerNumber**. Hal ini dilakukan untuk menghindari terjadinya error yang disebabkan adanya nama column yang sama pada dua tabel yang berbeda.



Gambaran Skema yang dihasilkan dari eksekusi kode di sebelumnya

Pada skema diatas dapat dilihat bahwa perintah SQL memilih column customer_number dan contact_first_name (dari tabel customers) dan kolom amount (dari tabel payments). Kemudian hasilnya akan berupa baris-baris dimana terdapat kecocokan antara customer_number dari tabel customers dan customer_number dari tabel orders.

5.1.1 INNER JOIN dengan klausa WHERE

berikut contoh penggunaan klausa where pada INNER JOIN

```
SELECT customers.customerNumber, customers.contactFirstName,  
payments.amount  
FROM customers  
INNER JOIN payments  
ON customers.customerNumber = payments.customerNumber  
WHERE payments.amount BETWEEN 6000 AND 14000;
```

Perintah diatas seperti sebelumnya hanya saja ditambahkan klausa where untuk memfilter data yang ada pada kolom amount di tabel payments.

5.1.2 INNER JOIN dengan AS Alias

```
SELECT C.customerNumber, C.contactFirstName, P.amount  
FROM customers AS C  
INNER JOIN payments AS P  
ON C.customerNumber = P.customerNumber;
```

Untuk mempermudah penulisan nama tabel maka dapat diterapkan penggunaan AS Alias seperti diatas, sehingga misalnya yang awalnya ditulis customers dapat diganti menjadi c.

5.1.3 INNER JOIN dengan tiga tabel

```
SELECT C.customerNumber, C.contactFirstName, P.amount, O.status  
FROM customers AS C  
INNER JOIN payments AS P  
ON C.customerNumber = P.customerNumber  
INNER JOIN orders AS O  
ON C.customerNumber = O.customerNumber;
```

Penggabungan tabel customers dan payments berdasarkan customerNumber begitupun dengan penggabungan tabel customers dan orders juga berdasarkan customerNumber. Untuk INNER JOIN lagi tabel yang lain caranya tetap sama.

Kita juga bisa menggunakan perintah JOIN sebagai ganti INNER JOIN. Pada dasarnya, kedua klausa ini sama.

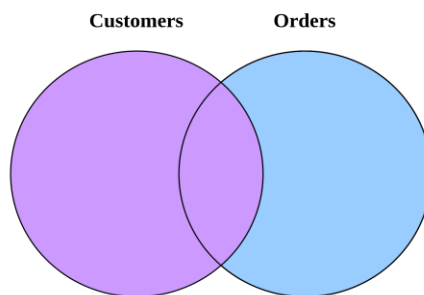
```
SELECT customers.customerNumber, customers.contactFirstName,  
payments.amount  
FROM customers  
JOIN payments  
ON customers.customerNumber = payments.customerNumber;
```

5.2 OUTER JOIN

Pada OUTER JOIN, data pada salah satu tabel akan ditampilkan semua, sedangkan data pada tabel yang lain hanya akan ditampilkan jika data tersebut ada pada tabel pertama. Pada bahasa SQL, OUTER JOIN dibagi menjadi 3, yaitu RIGHT JOIN, LEFT JOIN dan FULL JOIN.

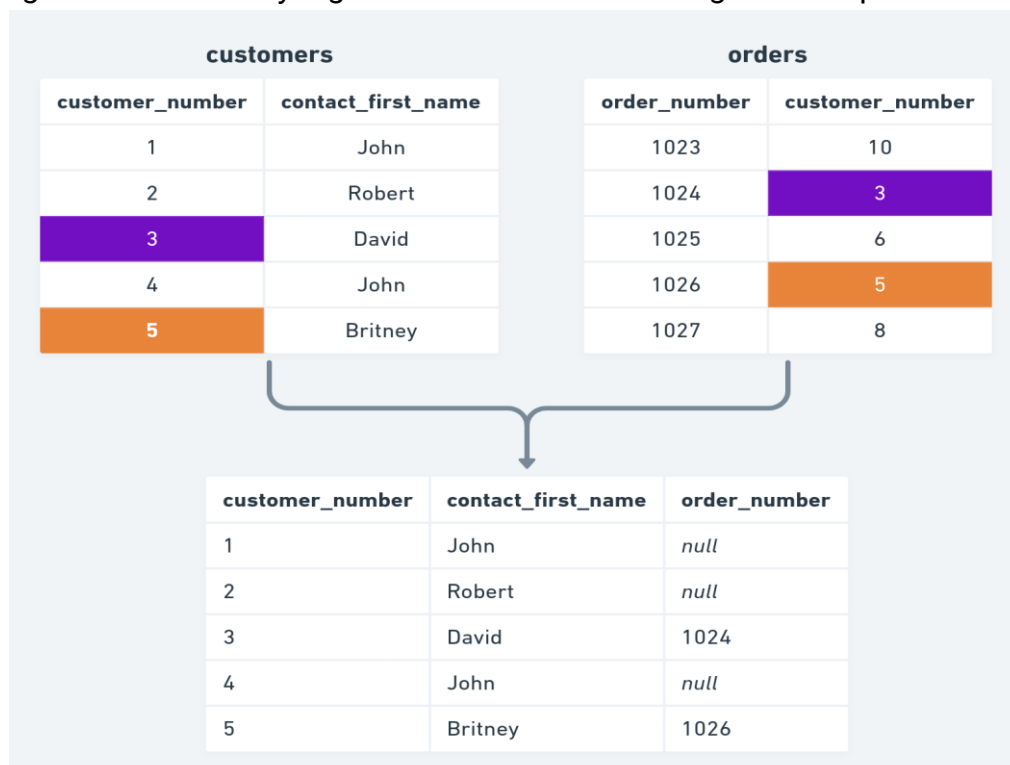
5.2.1 LEFT JOIN

Pada perintah LEFT JOIN semua data pada tabel sebelah kiri akan ditampilkan, sedangkan data tabel sebelah kanan hanya akan ditampilkan jika data terkait pada tabel tersebut muncul di tabel sebelah kiri.



```
SELECT c.customerNumber, c.contactFirstName, o.orderNumber
FROM customers AS c
LEFT JOIN orders AS o
ON c.customerNumber = o.customerNumber
```

Berikut gambaran skema yang bisa muncul setelah mengeksekusi perintah diatas



Dapat dilihat bahwa semua value dari dari tabel customers diambil sehingga apabila terdapat nilai *customer_number* yang hanya ada pada tabel customers namun tidak ada di orders maka tentu saja kolom-kolom dari orders pada baris tersebut akan menghasilkan null atau kosong.

Untuk mengecek nilai kosong pada *order_number* kita dapat menambahkan WHERE clause ke perintah seperti berikut

```
SELECT c.customerNumber, c.contactFirstName, o.orderNumber
FROM customers AS c
LEFT JOIN orders AS o
ON c.customerNumber = o.customerNumber
WHERE o.orderNumber IS NULL;
```

Left Join untuk tiga tabel atau lebih caranya tetap sama. Namun perlu diperhatikan apabila records yang sebagai key kosong maka tentu saja nilai kolom baru yang bergabung juga kosong. Misalnya jika pada perintah sebelumnya ditambahkan lagi tabel order detail seperti berikut

```
SELECT c.customerNumber, c.contactFirstName, o.orderNumber,
od.quantityOrdered
FROM customers AS c
LEFT JOIN orders AS o
ON c.customerNumber = o.customerNumber
LEFT JOIN orderdetails AS od
ON o.orderNumber = od.orderNumber
```

Jadi apabila records *orderNumber* pada tabel orders kosong maka otomatis records *orderdetails* yang terhubung dengannya juga kosong.

customerNumber	contactFirstName	orderNumber	quantityOrder
1	John	null	null
2	Robert	null	null
3	David	1024	10
4	Britney	null	null

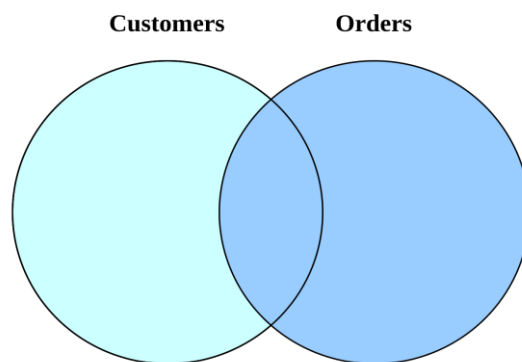
↑
Dari Orders

↑
Dari Order Details

Kita juga dapat menggunakan LEFT OUTER JOIN untuk mengganti LEFT JOIN karena fungsinya tetap sama.

5.2.2 RIGHT JOIN

Berkebalikan dengan LEFT JOIN, pada perintah RIGHT JOIN semua data pada tabel sebelah kanan akan ditampilkan, sedangkan data tabel sebelah kiri hanya akan ditampilkan jika data yang terkait pada tabel tersebut muncul di tabel sebelah kanan.



Berikut contoh kode penggunaan RIGHT JOIN

```
SELECT c.customerNumber, c.contactFirstName, o.orderNumber
FROM customers AS c
RIGHT JOIN orders AS o
ON c.customerNumber = o.customerNumber;
```



Dapat dilihat bahwa semua value dari dari tabel orders diambil sehingga apabila terdapat nilai *order_number* yang hanya ada pada tabel orders namun tidak ada di customers maka tentu saja kolom-kolom dari customers pada baris tersebut akan menghasilkan null atau kosong.

Selain menggunakan perintah RIGHT JOIN kita juga bisa menggunakan RIGHT OUTER JOIN karena fungsinya akan sama.

5.2.3 FULL OUTER JOIN

FULL OUTER JOIN atau FULL JOIN digunakan untuk menggabungkan dua tabel berdasarkan kolom tertentu. Perintah ini akan menampilkan data-data yang tidak berelasi. Namun pada tabel kanan dan tabel kiri data yang tidak berelasi akan bernilai NULL.

Namun MYSQL DBMS tidak mendukung sintaks FULL OUTER JOIN maka untuk memanipulasi hal ini dapat menggunakan LEFT JOIN dan RIGHT JOIN kemudian menggabungkannya menggunakan UNION.

TUGAS PRAKTIKUM

1. Tampilkan kolom tanggal pemesanan product dalam urutan menurun untuk pesanan *Ford Pickup Truck 1940*. (database *classicmodels*)
2. Tampilkan daftar nama produk yang dijual dengan harga kurang dari 80% dari MSRP (Harga Eceran Yang Disarankan). (database *classicmodels*)
3. Tampilkan Pembimbing Utama dari Mahasiswa bernama Sulaeman! (database *appseminar*)

BAB 6 | OPERATOR

Pada BAB 6 ini akan dipelajari berbagai macam operator yang dapat digunakan dalam statement SQL. Beberapa Operator di MySQL antara lain:

- Operator Aritmatika
- Operator Penugasan
- Operator Logika
- Operator Pembandingan
- Operator String
- Operator Bitwise
- Operator Majemuk (Compound)
- Operator Set

6.1 Operator Aritmatika

Operator Aritmatika adalah simbol yang digunakan untuk melakukan operasi terhadap atribut bertipe numerik, antara lain: $+$ (*penjumlahan*), $-$ (*pengurangan*), $*$ (*perkalian*), dan $/$ (*pembagian*). Ingat, SQL juga memberlakukan *precedence*, artinya perkalian dan pembagian diprioritaskan (dilakukan terlebih dahulu), kemudian penjumlahan dan pengurangan. Berikut contoh operasi perkalian:

```
SELECT *, quantityOrdered * priceEach + 100 AS priceTotal FROM orderdetails;
```

orderNumber	productCode	quantityOrdered	priceEach	orderLineNumber	priceTotal
10,100	S18_1749	30	136.0	3	4,180.0
10,100	S18_2248	50	55.09	2	2,854.5
10,100	S18_4409	22	75.46	4	1,760.12
10,100	S24_3969	49	35.29	1	1,829.21
10,101	S18_2325	25	108.06	4	2,801.5
10,101	S18_2795	26	167.06	1	4,443.56
10,101	S24_1937	45	32.53	3	1,563.85
10,101	S24_2022	46	44.35	2	2,140.1
10,102	S18_1342	39	95.55	2	3,826.45
10,102	S18_1367	41	43.13	1	1,868.33
10,103	S10_1949	26	214.3	11	5,671.8
10,103	S10_4962	42	119.67	4	5,126.14
10,103	S12_1666	27	121.64	8	3,384.28

Tanda kurung memiliki *precedence* tertinggi. Apabila operator dibungkus dalam tanda kurung, operator tersebut akan dieksekusi terlebih dahulu.

```
SELECT *, quantityOrdered * (priceEach + 100) AS priceTotal FROM orderdetails;
```

orderNumber	productCode	quantityOrdered	priceEach	orderLineNumber	priceTotal
10,100	S18_1749	30	136.0	3	7,080.0
10,100	S18_2248	50	55.09	2	7,754.5
10,100	S18_4409	22	75.46	4	3,860.12
10,100	S24_3969	49	35.29	1	6,629.21
10,101	S18_2325	25	108.06	4	5,201.5
10,101	S18_2795	26	167.06	1	6,943.56
10,101	S24_1937	45	32.53	3	5,963.85
10,101	S24_2022	46	44.35	2	6,640.1
10,102	S18_1342	39	95.55	2	7,626.45

6.2 Operator Bitwise

Operator bitwise mengeksekusi operasi dalam unit terkecil komputer (biner). Untuk penjelasan lebih lengkap, kunjungi [situs ini](#)

Operator	Deskripsi
&	Bitwise AND
	Bitwise OR
^	Bitwise Exclusive OR
~	Bitwise NOT

```
SELECT 25&20 AS result;
```

result

16

6.3 Operator Pembandingan

Operator pembandingan berguna untuk membandingkan dua ekspresi dan akan mengembalikan nilai boolean: *TRUE*, *FALSE*, atau *UNKNOWN*. Nilai *UNKNOWN* akan diberikan apabila suatu ekspresi memiliki nilai *null* atau tipe data tak sesuai.

Operator	Deskripsi	Tipe Data
=	Sama dengan	Numerik, String
>	Lebih dari	Numerik

<	Kurang dari	Numerik
>=	Lebih dari sama dengan	Numerik
<=	Lebih kecil sama dengan	Numerik
<>	Tidak sama dengan	Numerik, String

Untuk menampilkan harga (priceEach) yang lebih dari sama dengan 100:

```
SELECT * FROM orderdetails WHERE priceEach > 100;
```

orderNumber	productCode	quantityOrdered	priceEach	orderLineNumber
10,100	S18_1749	30	136.0	3
10,101	S18_2325	25	108.06	4
10,101	S18_2795	26	167.06	1
10,103	S10_1949	26	214.3	11

Untuk menampilkan order dengan productCode 'S18_1749':

```
SELECT * from orderdetails WHERE productCode = 'S18_1749';
```

orderNumber	productCode	quantityOrdered	priceEach	orderLineNumber
10,100	S18_1749	30	136.0	3
10,110	S18_1749	42	153.0	7
10,124	S18_1749	21	153.0	6

6.4 Operator Logika

Operator logika digunakan untuk menguji kebenaran dari suatu ekspresi, dan mengembalikan nilai boolean: *TRUE*, *FALSE*, atau *UNKNOWN*. Nilai *UNKNOWN* akan diberikan apabila suatu ekspresi memiliki nilai *null* atau tipe data tak sesuai.

Operator	Deskripsi	Tipe Data
ALL	TRUE jika semua himpunan perbandingan bernilai TRUE	
AND	TRUE jika kedua ekspresi <i>boolean</i> bernilai TRUE.	
ANY	TRUE jika salah satu dari himpunan perbandingan bernilai TRUE	
BETWEEN	TRUE jika nilai berada di antara suatu	Numerik

	rentang	
EXISTS	TRUE jika <i>subquery</i> menghasilkan baris <i>record</i>	Array (Numerik, String)
IN	TRUE jika operand sama dengan salah satu dari daftar ekspresi/ <i>tuple</i>	Array (Numerik, String)
LIKE	TRUE jika sesuai dengan pola tertentu	String
NOT	Kebalikan nilai dari operator <i>boolean</i> apapun	
OR	TRUE jika salah satu ekspresi <i>boolean</i> bernilai TRUE	
SOME	TRUE jika beberapa nilai dalam himpunan bernilai TRUE	

```
SELECT * FROM orderdetails WHERE productCode = 'S18_1749' OR productCode = 'S18_2248';
```

orderNumber	productCode	quantityOrdered	priceEach	orderLineNumber
10,407	S18_1749	76	141.1	2
10,420	S18_1749	37	153.0	5
10,100	S18_2248	50	55.09	2
10,110	S18_2248	32	51.46	6

Operator LIKE merupakan operator yang digunakan untuk tipe data *string*. Bedanya jika '=' itu berarti sama persis, LIKE dapat digunakan untuk mendeteksi pola tertentu atau sepotong *string* saja. Berikut contoh penggunaan operator LIKE:

Wildcard Character	Deskripsi	Contoh
%	Mewakili nol atau lebih karakter	bl% menemukan bl, black, blue, dan blob
_	Mewakili satu karakter	h_t menemukan hot, hat, dan hit
[]	Merepresentasikan karakter tunggal apa pun dalam tanda kurung	h[oa]t menemukan hot dan hat, tapi bukan hit

^	Mewakili karakter apa pun yang tidak ada di dalam tanda kurung	h[^oa]t menemukan hit,tapi bukan hot dan hat
-	Mewakili berbagai karakter	c[a-b]t menemukan cat dan cbt

Query	Deskripsi
WHERE customerName LIKE 's%'	Tampilkan <i>string</i> yang <i>dimulai</i> dengan huruf "s" (contoh: schuyler)
WHERE customerName LIKE '%s'	Tampilkan <i>string</i> yang <i>diakhiri</i> dengan huruf "s" (contoh: hors)
WHERE customerName LIKE '%ab%'	Tampilkan <i>string</i> yang memiliki "ab" - <i>di posisi manapun</i> (contoh: collectables)
WHERE customerName LIKE '_r%'	Tampilkan <i>string</i> yang memiliki huruf "r" di posisi kedua atau sebagai huruf kedua (contoh: dragon)
WHERE customerName LIKE 'a_%'	Tampilkan <i>string</i> yang dimulai dengan "a" <i>dan string</i> dengan <i>length</i> minimal 2 (contoh: aerius, length: 6)
WHERE customerName LIKE 'a__%'	Tampilkan <i>string</i> yang dimulai dengan "a" <i>dan string</i> dengan <i>length</i> minimal 3 (contoh: aer, length: 3)
WHERE customerName LIKE 'a%o'	Tampilkan <i>string</i> yang <i>dimulai</i> dengan "a" <i>dan diakhiri</i> dengan "o" (contoh: Ascari Co)

```
SELECT * FROM orderdetails WHERE productCode LIKE 'S18%9' AND (priceEach > 100 OR priceEach < 70) ;
```

orderNumber	productCode	quantityOrdered	priceEach	orderLineNumber
10,100	S18_1749	30	136.0	3
10,104	S18_2319	29	122.73	12
10,108	S18_1889	38	67.76	2
10,109	S18_1130	26	117.40	4

6.5 Predicate

Predicate akan mengevaluasi suatu ekspresi dan mengembalikan nilai *boolean*: TRUE, FALSE, dan UNKNOWN.

Operator	Deskripsi	Tipe Data
<i>CONTAINS</i>	Digunakan untuk pencarian <i>full text</i> atau indeks <i>full text</i>	String
<i>IS NULL</i>	Bernilai <i>NULL</i>	-
<i>IS NOT NULL</i>	Bernilai <i>NON-NULL</i>	-

```
SELECT * FROM customers WHERE state IS NOT NULL AND addressLine2 IS NOT NULL;
```

customerName	contactLastName	contactFirstName	phone	addressLine1	addressLine2	city	state
Australian Collectors, Co.	Ferguson	Peter	03 9520 4555	636 St Kilda Road	Level 3	Melbourne	Victoria
Muscle Machine Inc	Young	Jeff	2125557413	4092 Furth Circle	Suite 400	NYC	NY
American Souvenirs Inc	Franco	Keith	2035557845	149 Spinnaker Dr.	Suite 101	New Haven	CT
Vitachrome Inc.	Frick	Michael	2125551500	2678 Kingston Rd.	Suite 101	NYC	NY

6.6 Fungsi (Function)

Fungsi, atau disebut pula *method* adalah sebuah sintaks yang memanggil suatu perintah tertentu sesuai dengan guna fungsi itu sendiri. Dalam SQL sudah terdapat banyak fungsi *built-in*. Secara umum, sintaks untuk memanggil fungsi seperti berikut: *function_name(column|expression, [arg1, arg2, ...])*

Pemanggilan fungsi dapat dilakukan di dalam klausa SELECT maupun WHERE atau HAVING.

Daftar fungsi lengkap dengan fungsinya dapat dilihat melalui situs [W3Schools](#) atau [dokumentasi SQL](#).

6.6.1.1 Fungsi Karakter/String

Berikut adalah daftar fungsi manipulasi *string* yang terdapat SQL:

ASCII	LTRIM	SOUNDEX
CHAR	NCHAR	SPACE
CHARINDEX	PATINDEX	STR
CONCAT	QUOTENAME	STRING ESCAPE
DIFFERENCE	REPLACE	STRING SPLIT
FORMAT	REPLICATE	STUFF

LEFT	REVERSE	SUBSTRING
LEN	RIGHT	UNICODE
LOWER	RTRIM	UPPER

6.6.1.2 Fungsi Numerik

Berikut ini adalah fungsi numerik yang didukung pada SQL:

ABS	DEGREES	RAND
ACOS	EXP	ROUND
ASIN	FLOOR	SIGN
ATAN	LOG	SIN
ATN2	LOG10	SQRT
CEILING	PI	SQUARE
cos	POWER	TAN

```
SELECT amount, ROUND(amount) AS rounded_amount FROM payments;
```


6.6.1.3 Fungsi Tanggal & Waktu

Berikut ini fungsi tanggal & waktu yang didukung pada SQL:

CURRENT_TIMESTAMP	GETDATE	GETUTCDATE
DATENAME	DATEPART	DAY
MONTH	YEAR	DATEDIFF
DATEDIFF_BIG		

Tugas Praktikum

1. Dalam database *classicmodels*, lakukan join antara table *orders* dan *orderdetails*. Kemudian, tampilkan empat kolom: *orderNumber*, *status*, *shippedDate*, dan *quantityOrdered*. Dimana kolom *status* sama dengan 'cancelled' dan *shippedDate* tidak boleh *null*. Tuliskan query-nya agar format tabel sesuai dengan gambar berikut!

orderNumber		status	shippedDate	quantityOrdered
10,179		Cancelled	2003-11-13	24
10,179		Cancelled	2003-11-13	47
10,179		Cancelled	2003-11-13	27

2. Seperti tercatat pada tabel *orders*, beberapa *customers* berkomentar untuk menggunakan *DHL* sebagai jasa *shipping*. Siapa saja *customers* yang dimaksud? Tuliskan query-nya agar format tabel sesuai dengan gambar berikut!

customerName	status	comments
Euro+ Shopping Channel	Shipped	Customer requested that DHL is used for this shipping
La Corne D'abondance, Co.	Shipped	Customer requested that DHL is used for this shipping
Gift Depot Inc.	Shipped	Customer requested that DHL is used for this shipping
Down Under Souvenirs, Inc	Shipped	Customer requested that DHL is used for this shipping
The Sharp Gifts Warehouse	Shipped	Customer requested that DHL is used for this shipping
Microscale Inc.	Shipped	Customer requested that DHL is used for this shipping

3. Menggunakan database *classicmodels*, lakukan join dan tampilkan semua *order* pembelian mobil bermerek *Ferrari* dan *status* sama dengan 'Shipped'. Tuliskan query-nya agar format tabel sesuai dengan gambar berikut!

customerName	productName	paymentDate	status
Signal Gift Stores	1992 Ferrari 360 Spider red	2004-12-17	Shipped
Signal Gift Stores	1992 Ferrari 360 Spider red	2003-06-06	Shipped
Signal Gift Stores	1992 Ferrari 360 Spider red	2004-08-20	Shipped

BAB 7 | FUNCTION DAN GROUPING

7.1 AGGREGATE FUNCTION

Aggregate function adalah fungsi yang melakukan operasi pada beberapa nilai dan mengembalikan satu nilai.

syntax aggregate function :

```
function_name(DISTINCT | ALL expression)
```

function_name : nama aggregate function yang ingin digunakan

DISTINCT | ALL : DISTINCT digunakan untuk melakukan operasi pada kolom dengan nilai yang unik, sebaliknya ALL melakukan operasi pada semua nilai pada kolom.
default ALL

expression : input nilai, biasanya nama kolom

7.1.1 AVG

fungsi AVG digunakan untuk mendapatkan nilai rata - rata.

students

Name	Score
Yoyo	75
Ucup	75
Ikhsan	80
Richard	85
Fatwa	90

contoh :

```
SELECT AVG(Score) from students;
```

output :

```
81.0
```

penjelasan :

$$\text{rata - rata}(75, 75, 80, 85, 90) = \frac{(75 + 75 + 80 + 85 + 90)}{5} = 81.0$$

contoh distinct :

```
SELECT AVG(DISTINCT Score) from students;
```

output :

66.0

penjelasan :

$$\text{rata - rata}(75, 80, 85, 90) = \frac{(75 + 80 + 85 + 90)}{5} = 66.0$$

7.1.2 SUM

fungsi SUM digunakan untuk mendapatkan jumlahan nilai.

students

Name	Score
Yoyo	75
Ucup	75
Ikhsan	80
Richard	85
Fatwa	90

contoh :

```
SELECT SUM(Score) from students;
```

output :

405

penjelasan :

$$\text{SUM}(75, 75, 80, 85, 90) = 75 + 75 + 80 + 85 + 90 = 450$$

contoh distinct :

```
SELECT SUM(DISTINCT Score) from students;
```

output :

330

penjelasan :

$$\text{SUM}(75, 80, 85, 90) = 75 + 80 + 85 + 90 = 330$$

7.1.3 COUNT

fungsi COUNT digunakan untuk menghitung jumlah data pada tabel.

students

Name	Score
Yoyo	75
Ucup	75
Ikhsan	80
Richard	85
Fatwa	90

contoh :

```
SELECT COUNT(*) from students;
```

output :

```
5
```

penjelasan : fungsi COUNT(*) menghitung berapa banyak jumlah data yang ada pada tabel students. simbol * biasanya digunakan sebagai argumen fungsi COUNT.

contoh distinct :

```
SELECT COUNT(DISTINCT Score) from students;
```

output :

```
4
```

penjelasan :

fungsi COUNT(DISTINCT Score) akan menghitung banyaknya data pada table students dan hanya akan menghitung nilai yang muncul lebih dari sekali pada kolom Score.

nilai pada kolom Score = 75, 75, 80, 85, 90

sehingga nilai

75 hanya akan dihitung sekali.

7.1.4 MAX

fungsi MAX digunakan untuk mendapatkan nilai tertinggi pada kolom.

students

Name	Score
Yoyo	75
Ucup	75
Ikhsan	80
Richard	85
Fatwa	90

contoh :

```
SELECT MAX(Score) from students;
```

output :

```
90
```

penjelasan :

$\text{MAX}(75, 75, 80, 85, 90) = 90$ adalah nilai tertinggi

contoh distinct :

```
SELECT MAX(DISTINCT Score) from students;
```

output :

```
90
```

penjelasan :

$\text{MAX}(75, 80, 85, 90) = 90$

7.1.4 MIN

fungsi MIN digunakan untuk mendapatkan nilai terendah pada kolom.

students

Name	Score
Yoyo	75
Ucup	75
Ikhsan	80
Richard	85
Fatwa	90

contoh :

```
SELECT MIN(Score) from students;
```

output :

```
75
```

penjelasan :

MIN(75, 75, 80, 85, 90) = 75 adalah nilai terendah

contoh distinct :

```
SELECT MIN(DISTINCT Score) from students;
```

output :

```
75
```

penjelasan :

MIN(75, 80, 85, 90) = 75

7.1.5 CONCAT_GROUP

fungsi CONCAT_GROUP digunakan untuk menggabung nilai(string) pada kolom.

syntax CONCAT_GROUP :

```
CONCAT_GROUP([DISTINCT] expr [,expr ...]  
             [ORDER BY {unsigned_integer | col_name | expr}  
               [ASC | DESC] [,col_name ...]]  
             [SEPARATOR str_val])
```

ORDER BY : hasil dari penggabungan akan diurutkan berdasarkan urutan alfabet(*lexicographic*)

SEPARATOR : separator dari penggabungan string, default “,”

students

Name	Score
Yoyo	75
Ucup	75
Ikhsan	80
Richard	85
Fatwa	90

contoh :

```
SELECT CONCAT_GROUP(Name ORDER BY Name ASC SEPARATOR “;”) from  
students;
```

output :

```
Fatwa;Ikhsan;Richard,Ucup;Yoyo
```

penjelasan :

CONCAT_GROUP(Name ORDER BY Name ASC SEPARATOR “;”)

Name -> menggabung nilai yang ada pada kolom Name

ORDER BY Name ASC -> hasil penggabungan akan diurut secara menaik berdasarkan urutan alfabet

SEPARATOR “,” -> pemisah antar nilai yang digabung

7.1.6 GROUP BY CLAUSE

mengelompokkan beberapa data hasil dari aggregate function.

syntax :

```
SELECT
  c1, c2,..., cn, aggregate_function(ci)
FROM
  table
WHERE
  where_conditions
GROUP BY c1 , c2,...,cn;
```

products

Product Name	Category	Price
Sate	Makanan	15000
Nasi Padang	Makanan	13000
Es Jeruk	Minuman	3500
Coto	Makanan	20000
Es Teh	Minuman	2000
Gelas	Lain - Lain	15000

contoh :

```
SELECT Category, COUNT(Category) AS frequencies
FROM products
GROUP BY Category;
```

output :

Category	frequencies
Makanan	3
Minuman	2
Lain - Lain	1

contoh :

```
SELECT Category, SUM(Price) as categories_total_price from products group by
Category;
```

output :

Category	categories_total_price
Makanan	48000
Minuman	5500
Lain - Lain	15000

7.1.6 HAVING CLAUSE

HAVING CLAUSE digunakan untuk memfilter hasil dari group by. WHERE CLAUSE tidak dapat digunakan untuk memfilter hasil dari group by sehingga digunakan HAVING CLAUSE.

syntax :

```
SELECT
  c1, c2,..., cn, aggregate_function(ci) AS aggregate_result
FROM
  table
WHERE
  where_conditions
GROUP BY c1 , c2,...,cn;
HAVING
(FILTER EXPRESSION)
```

FILTER_EXPRESSION : mempunyai konsep yang sama seperti memfilter menggunakan WHERE CLAUSE.

products

Product Name	Category	Price
Sate	Makanan	15000
Nasi Padang	Makanan	13000
Es Jeruk	Minuman	3500
Coto	Makanan	20000
Es Teh	Minuman	2000
Gelas	Lain - Lain	15000

contoh :

```
SELECT Category, COUNT(Category) AS frequencies
FROM products
GROUP BY Category
HAVING
frequencies > 1;
```

output :

Category	frequencies
Makanan	3
Minuman	2

penjelasan :

bisa dilihat Category dengan nilai "Lain - Lain" tidak dimunculkan karena pada HAVING CLAUSE kita hanya mengambil kolom "frequencies" dengan nilai lebih dari 1.

```
SELECT Category, COUNT(Category) AS frequencies
FROM products
GROUP BY Category
WHERE
frequencies > 1;
```

output :

akan menghasilkan error.

contoh :

```
SELECT Category,
COUNT(Category) AS frequencies,
SUM(Price) AS total_price
FROM products
WHERE price > 15000
GROUP BY Category
HAVING frequencies > 1;
```

output :

Category	frequencies	total_price
Makanan	2	35000

7.2 COMPARISON FUNCTION

7.2.1 COALESCE

Mengembalikan nilai NON-NULL pertama pada argumen yang diberikan.

syntax CONCAT_GROUP :

```
COALESCE(value1, value2, value3, value4)
```

jika value1, value2, value3, value4 semua nya bernilai NULL maka COALESCE akan mengembalikan NULL.

contoh :

```
SELECT COALESCE(1, NULL, 2);
```

output :

```
1
```

penjelasan :

karena nilai NON-NULL pertama pada argument COALESCE(1, NULL, 2) adalah 1.

contoh :

```
SELECT COALESCE(NULL, NULL, NULL);
```

output :

```
NULL
```

penjelasan :

karena nilai pada semua argument COALESCE(NULL, NULL, NULL) adalah NULL.

*biasanya COALESCE function digunakan untuk mensubstitusi nilai pada NULL pada kolom

Movies

Movie	Description
The Avengers	NULL
The Batman	NULL

contoh :

```
SELECT name, COALESCE(Description, "No Description") from Movies;
```

output :

Movie	Description
The Avengers	No Description
The Batman	No Description

7.2.1 GREATEST AND LEAST

GREATEST digunakan untuk mencari nilai terbesar.

LEAST digunakan untuk mencari nilai terkecil.

syntax CONCAT_GROUP :

```
GREATEST(value1, value2, value3, value4)
```

```
LEAST(value1, value2, value3, value4)
```

contoh :

```
SELECT GREATEST(1, 2, 1, 2, 5);
```

output :

```
5
```

contoh :

```
SELECT LEAST(1, 2, 1, 2, 5);
```

output :

```
1
```

E_commerce

Product_Name	Tokopedia	Shopee
Earphone	125000	100000
T-Shirt	55000	65000

contoh :

```
SELECT GREATEST(Tokopedia, Shopee) AS more_expensive FROM E_commerce;
```

output :

Product_Name	more_expensive
Earphone	125000
T-Shirt	65000

penjelasan :

pada argumen GREATEST function digunakan nama kolom pada tabel E_commerce

E_commerce

Product_Name	Tokopedia	Shopee
Earphone	125000	100000
T-Shirt	55000	65000

contoh :

```
SELECT LEAST(Tokopedia, Shopee) AS cheaper FROM E_commerce;
```

output :

Product_Name	more_expensive
Earphone	100000
T-Shirt	55000

7.3 DATE FUNCTION

7.3.1 DAY

mengembalikan tanggal pada tipe data "timestamp, datetime, date"

syntax :

```
SELECT DAY(COLUMN | VALUE);
```

contoh :

```
SELECT DAY(CURRENT_TIMESTAMP);
```

7.3.2 MONTH

mengembalikan bulan pada tipe data "timestamp, datetime, date"

syntax :

```
SELECT MONTH(COLUMN | VALUE);
```

contoh :

```
SELECT MONTH(CURRENT_TIMESTAMP);
```

7.3.2 YEAR

mengembalikan tahun pada tipe data "timestamp, datetime, date"

syntax :

```
SELECT YEAR(COLUMN | VALUE);
```

contoh :

```
SELECT YEAR(CURRENT_TIMESTAMP);
```

7.3.3 NOW

mengembalikan waktu sekarang dalam bentuk “timestamp, datetime, atau date”.

contoh :

```
SELECT NOW();
```

7.3.3 DATEDIFF

mengembalikan selisih hari antara 2 “date, datetime, timestamp”.

contoh :

```
SELECT DATEDIFF('2011-08-17','2011-08-17'); -- 0 day
```

output :

```
0
```

contoh :

```
SELECT DATEDIFF('2011 - 08 - 17','2011 - 08 - 08'); -- 9 days
```

output :

```
9
```

7.4 STRING FUNCTION

7.4.1 LOWER

mengubah huruf kapital pada string menjadi kecil.

syntax :

```
SELECT LOWER(COLUMN | VALUE);
```

contoh :

```
SELECT LOWER('KAPITAL');
```

output :

```
kapital
```

7.4.2 UPPER

mengubah huruf kecil pada string menjadi kapital.

syntax :

```
SELECT UPPER(COLUMN | VALUE);
```

contoh :

```
SELECT UPPER('kapital');
```

output :

```
KAPITAL
```

7.4.3 LEFT

mengeksrak bagian paling kiri pada string sebesar panjang yang ditentukan.

syntax :

```
SELECT LEFT(COLUMN | VALUE, length);
```

length -> banyaknya karakter yang ingin di ekstrak pada VALUE

contoh :

```
SELECT LEFT('I am learning MYSQL', 13);
```

output :

```
I am learning
```

7.4.4 RIGHT

mengeksrak bagian paling kanan pada string sebesar panjang yang ditentukan.

syntax :

```
SELECT RIGHT(COLUMN | VALUE, length);
```

length -> banyaknya karakter yang ingin di ekstrak pada VALUE

contoh :

```
SELECT RIGHT('I am learning MYSQL', 5);
```

output :

```
MYSQL
```

7.4.5 CONCAT

menggabungkan beberapa string menjadi satu.

syntax :

```
SELECT CONCAT(COLUMN | VALUE 1, COLUMN | VALUE 2, ... COLUMN | VALUE N);
```

length -> banyaknya karakter yang ingin di ekstrak pada VALUE

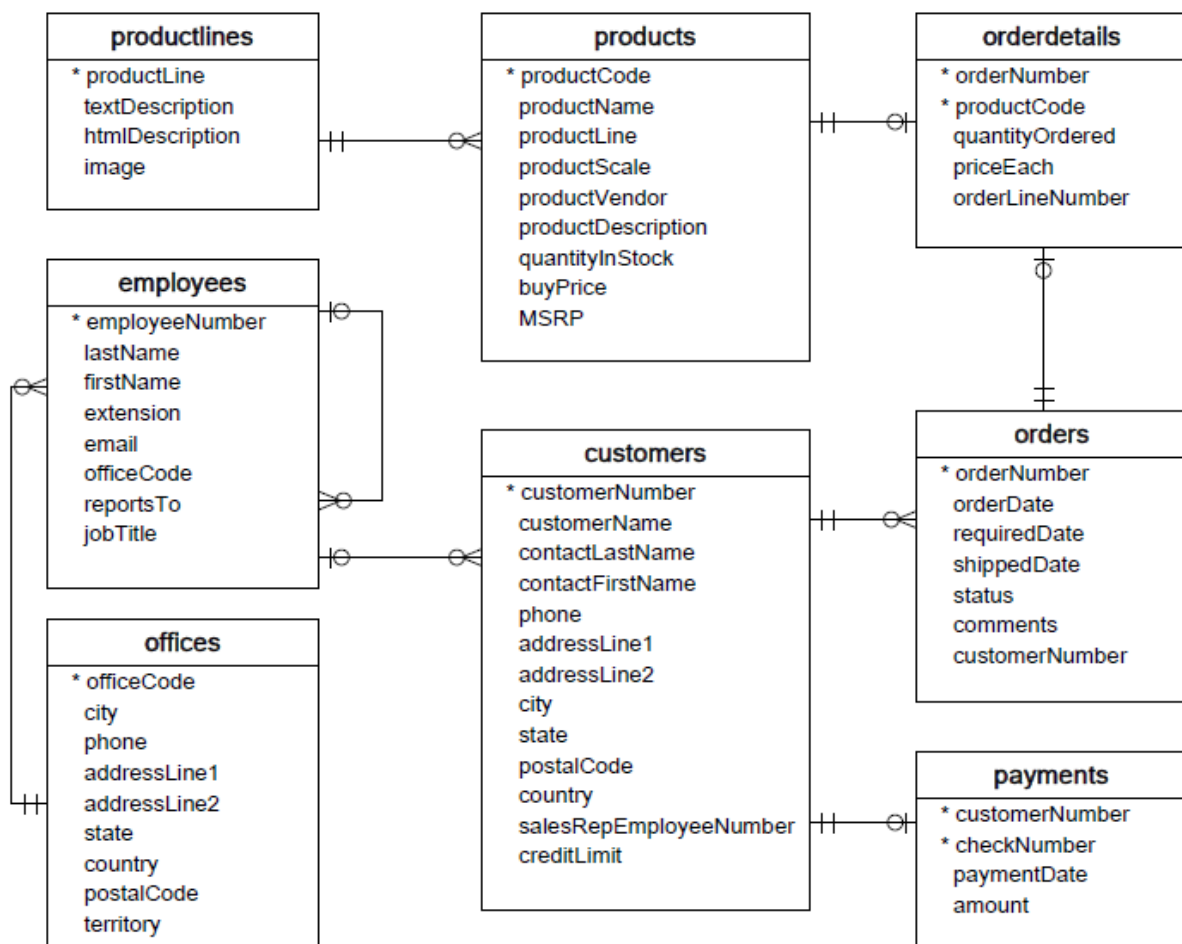
contoh :

```
SELECT CONCAT('A', 'B', 'C', 'D');
```

output :

```
ABCD
```

TUGAS PRAKTIKUM



1. menggunakan database classicmodels
berapa banyak customer yang belanjanya melebihi creditLimit yang dimiliki ?
2. siapa customer yang membeli barang terbanyak (dihitung dari quantity bukan price) ?
format output :
[customerName] : [contactFirstName] [contactLastName]@[addressLine1]
3. Pada database appseminar, tampilkan nama mahasiswa yang seminar pada hari senin atau jumat. (nama mahasiswa ditampilkan dalam huruf kapital)

BAB 8 | SUBQUERY SQL

8.1 Pengenalan Subquery

Subquery merupakan sebuah *query* yang berada di dalam *query* lainnya yang biasanya digunakan dalam *query* SELECT, INSERT, UPDATE, atau DELETE. *Subquery* juga bisa digunakan dalam *subquery* lainnya. *Subquery* dapat juga disebut dengan *inner query* sedangkan *query* yang mengandung *subquery* tersebut disebut dengan *outer query*.

```
SELECT lastname, firstname
FROM employees
WHERE officeCode IN (SELECT officeCode
FROM offices
WHERE country = 'USA')
```

Secara umum, *syntaxnya* berbentuk seperti berikut.

```
SELECT kolom1, kolom2, ... FROM nama_tabel
WHERE nama_kolom OPERATOR
(SELECT nama_kolom FROM nama_tabel WHERE kondisi)
```

8.2 Penggunaan Subquery dalam WHERE

Subquery dapat diletakkan di dalam perintah WHERE dalam suatu *query*. Misalnya dalam database *classic models*, kita ingin menemukan pelanggan dengan jumlah pembayaran (*amount*) paling besar pada tabel *payments*. Maka contoh *query* yang dapat digunakan adalah sebagai berikut.

```
SELECT customerNumber, checkNumber, amount
FROM payments
WHERE amount =
(SELECT MAX(amount) FROM payments);
```

Setelah dijalankan, maka akan diperoleh hasil seperti pada gambar berikut.

customerNumber	checkNumber	amount
141	JE105477	120.166,58

Kita juga dapat langsung mengetahui nama pelanggannya dengan menggunakan *subquery*. Berikut *query* yang dapat digunakan.

```
SELECT payments.customerNumber, customers.customerName,
payments.checkNumber,
payments.amount
FROM payments
```

```
INNER JOIN customers ON payments.customerNumber =
customers.customerNumber
WHERE payments.amount =
(SELECT MAX(payments.amount) FROM payments)
```

Setelah dijalankan, maka hasil yang diperoleh adalah seperti berikut.

customerNumber	customerName	checkNumber	amount
141	Euro+ Shopping Channel	JE105477	120.166,58

8.3 Penggunaan Subquery dalam FROM

Subquery juga dapat digunakan di dalam perintah FROM. Misalnya dalam tabel orderDetails, kita ingin menghitung berapa maksimum dan minimum jumlah barang yang dicapai dalam satu pesanan serta berapa jumlah rata-rata barang yang dipesan dalam setiap pesanan. Maka, query yang dapat digunakan adalah seperti berikut.

```
SELECT
  MAX(items) AS max_item,
  MIN(items) AS min_item,
  FLOOR(AVG(items)) AS avg_item
FROM (SELECT
  orderNumber,
  COUNT(orderNumber) AS items
  FROM orderdetails
  GROUP BY orderNumber) AS lineitems;
```

Setelah dijalankan, maka akan keluar hasil seperti berikut.


max_item	min_item	avg_item
18	1	9

8.4 Penggunaan subquery dalam SELECT

Subquery juga bisa digunakan dalam perintah SELECT. Misalnya kita ingin melihat berapa pesanan yang telah dilakukan oleh para pelanggan selama ini. Maka query yang dapat digunakan adalah seperti berikut.

```
SELECT
  customerNumber,
  customerName,
  (SELECT COUNT(*) FROM payments WHERE customers.customerNumber =
payments.customerNumber) AS orderNumber
FROM customers
```

Ketika dijalankan, maka akan keluar hasil seperti berikut.


customerNumber		customerName	orderNumber
103		Atelier graphique	1
112		Signal Gift Stores	3
114		Australian Collectors, Co.	3
119		La Rochelle Gifts	3
121		Baane Mini Imports	3
124		Mini Gifts Distributors Ltd.	9
125		Havel & Zbyszek Co	0
128		Blauer See Auto, Co.	3
129		Mini Wheels Co.	3
131		Land of Toys Inc.	3
141		Euro+ Shopping Channel	13
144		Volvo Model Replicas, Co	1
145		Danish Wholesale Imports	3
146		Saveley & Henriot, Co.	3

8.5. Penggunaan IN/NOT IN dalam *subquery*

IN/NOT IN digunakan dalam *subquery* untuk menemukan data yang termasuk atau tidak termasuk dalam tabel. Misalnya, kita ingin melihat daftar nama *employee* pada tabel *employees* yang ada di dalam kantor 1 (*officeCode* = '1'). Maka, *query* yang dapat digunakan adalah sebagai berikut.

```
SELECT employeeNumber, firstName, lastName
FROM employees
WHERE employeeNumber IN
(SELECT employeeNumber FROM employees
WHERE officeCode = '1'
GROUP BY employeeNumber);
```

Jika dijalankan, maka akan muncul seluruh nama *employee* yang bekerja di kantor 1. Hasilnya seperti berikut

employeeNumber		firstName	lastName
1.002		Diane	Murphy
1.056		Mary	Patterson
1.076		Jeff	Firrelli
1.143		Anthony	Bow
1.165		Leslie	Jennings
1.166		Leslie	Thompson

Kemudian, jika diganti perintahnya menjadi NOT IN, maka akan ditampilkan nama *employee* yang tidak bekerja di kantor 1. Hasilnya akan menjadi seperti berikut.

employeeNumber	🔑	firstName	lastName
1.088		William	Patterson
1.102		Gerard	Bondur
1.188		Julie	Firrelli
1.216		Steve	Patterson
1.286		Foon Yue	Tseng
1.323		George	Vanauf
1.337		Loui	Bondur
1.370		Gerard	Hernandez
1.401		Pamela	Castillo
1.501		Larry	Bott
1.504		Barry	Jones
1.611		Andy	Fixter
1.612		Peter	Marsh

8.6. Penggunaan EXIST/NOT EXIST dalam *subquery*

EXIST/NOT EXIST digunakan untuk menguji apakah suatu *subquery* mengembalikan data atau tidak. EXIST/NOT EXIST hanya akan mengembalikan nilai boolean, yaitu TRUE/FALSE. Misalnya, kita ingin mengecek apakah ada pelanggan yang melakukan pemesanan lebih dari 6 kali pada database *classic models*. Maka, *query* yang dapat digunakan adalah sebagai berikut.

```
SELECT
  customerNumber,
  customerName
FROM customers cust
WHERE EXISTS (
  SELECT customers.customerNumber, customers.customerName,
  COUNT(payments.checkNumber) AS orderNumber
  FROM customers
  LEFT JOIN payments ON customers.customerNumber =
  payments.customerNumber
  WHERE customers.customerNumber = cust.customerNumber
  GROUP BY customers.customerNumber
  HAVING orderNumber > 6)
```

Setelah dijalankan, maka akan terlihat pelanggan mana saja yang melakukan pemesanan lebih dari 6 kali. Berikut hasilnya.

customerNumber	🔑	customerName
124		Mini Gifts Distributors Ltd.
141		Euro+ Shopping Channel

8.7. Correlated Subquery

Correlated subquery merupakan jenis *query* dimana *inner query* dijalankan terlebih dahulu sebelum melakukan *outer query*. Biasanya, *query* jenis ini digunakan apabila *outer* dan *inner query* merujuk kepada tabel yang sama. Contoh pada bagian EXIST/NOT EXIST merupakan salah satu contoh dari *correlated subquery*. Contoh lainnya misalkan kita ingin mengetahui barang apa saja yang harga penjualannya melebihi rata-rata barang keseluruhan pada tabel *products* dalam database *classic models*. Berikut *query* yang dapat digunakan.

```
SELECT
  productname,
  buyprice
FROM
  products p1
WHERE
  buyprice >
  (SELECT AVG(buyprice)
   FROM products
   WHERE productline = p1.productline)
```

Setelah dijalankan, maka akan muncul seluruh nama barang yang harga jualnya diatas rata-rata. Berikut hasilnya.

productname	buyprice
1952 Alpine Renault 1300	98,58
1996 Moto Guzzi 1100i	68,99
2003 Harley-Davidson Eagle Drag Bike	91,02
1972 Alfa Romeo GTA	85,68
1962 LanciaA Delta 16V	103,42
1968 Ford Mustang	95,34
2001 Ferrari Enzo	95,59
1958 Setra Bus	77,90
2002 Suzuki XREO	66,27
1969 Corvair Monza	89,14
1968 Dodge Charger	75,16
1969 Ford Falcon	83,05
1940 Ford Pickup Truck	58,33

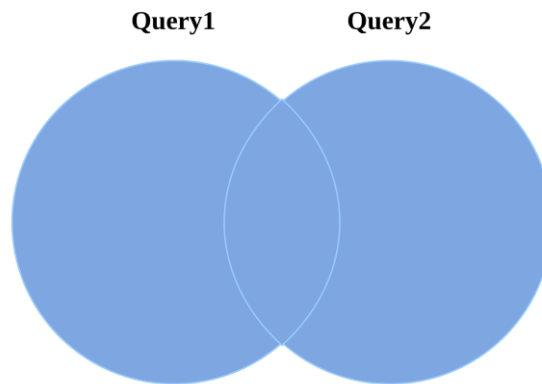
TUGAS PRAKTIKUM

1. Perhatikan database *classic models*. Buatlah *query* untuk menampilkan data pembayaran (*payment*) terkecil dan terbesar untuk masing-masing pengguna dengan menggunakan *subquery*. (Hint : gunakan tabel *payments* dan *customers*)
2. Buatlah *query* untuk menampilkan seluruh *employee* yang bekerja di *office* dengan *employee* terbanyak (misalnya *office A* memiliki paling banyak *employee*, maka buatlah daftar *employee* pada *office A*).
3. Buatlah *query* yang akan menampilkan seluruh *product* yang di namanya mengandung kata 'Ford' beserta *productScale* dari *product* tersebut. Gunakan *subquery* untuk menampilkannya.

BAB 9 | UNION, INTERSECT & EXCEPT

9.1 Union

Perintah UNION merupakan operasi menggabungkan dua subquery menjadi satu hasil yang terdiri dari baris yang dikembalikan oleh kedua query.

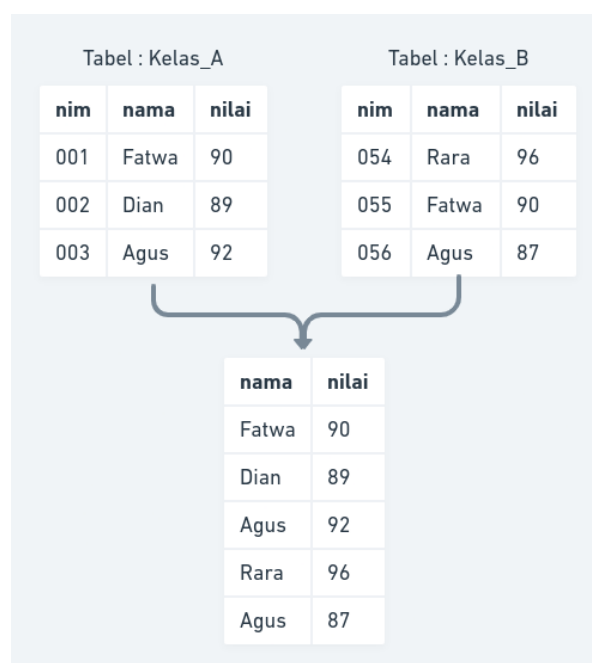


Pada UNION, apabila hasil penggabungan terdapat duplikasi data maka data tersebut hanya dimunculkan sekali.

Berikut contoh penggunaan perintah UNION

```
SELECT nama, nilai
FROM kelas_a
UNION
SELECT nama, nilai
FROM kelas_b;
```

Perintah diatas apabila dieksekusi maka akan menghasilkan output dengan alur seperti berikut



Ketika menggunakan perintah UNION di SQL kita harus selalu mengingat bahwa :

- Jumlah kolom dari setiap tabel harus sama. Seperti Kelas_A dan Kelas_B masing-masing memiliki 3 kolom.
- Tipe data dari setiap kolom harus sama. Seperti tipe data nama, dan nilai yang sama antara tabel Kelas_A dan Kelas_B
- Setiap kolom harus memiliki urutan yang sama pada setiap tabel. Seperti urutan kolom nim-nama-nilai pada tabel Kelas_A juga sama dengan urutan kolom tabel Kelas_B.

Selain operator UNION juga terdapat operator UNION ALL, operator ini mirip seperti UNION yang memilih baris dari dua atau lebih tabel namun UNION ALL mengabaikan adanya baris duplikat.

Berikut contoh kode UNION ALL

```
SELECT nama, nilai
FROM kelas_a
UNION ALL
SELECT nama, nilai
FROM kelas_b;
```

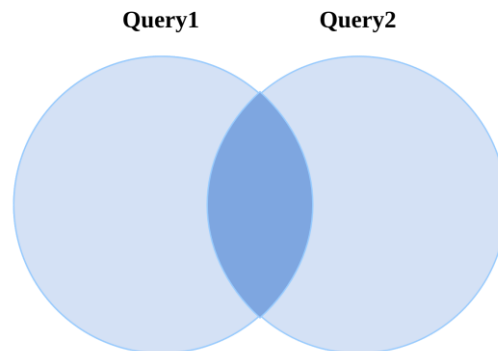
Berikut skema hasil eksekusi kode diatas

Tabel : Kelas_A			Tabel : Kelas_B		
nim	nama	nilai	nim	nama	nilai
001	Fatwa	90	054	Rara	96
002	Dian	89	055	Fatwa	90
003	Agus	92	056	Agus	87

nama	nilai
Fatwa	90
Dian	89
Agus	92
Rara	96
Fatwa	90
Agus	87

9.2 INTERSECT

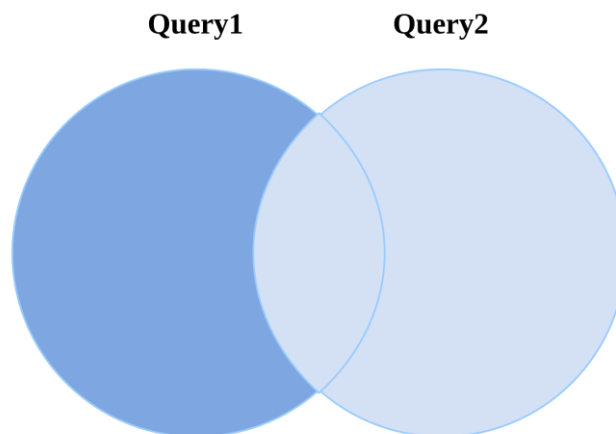
Operasi INTERSECT menggabungkan hasil dua query menjadi satu hasil yang terdiri dari semua baris yang memenuhi syarat untuk kedua query. Jika menggunakan operator logika maka UNION adalah Logika OR sedangkan INTERSECT adalah logika AND.



Namun MYSQL tidak mendukung keyword INTERSECT jadi kita perlu mengakalinya menggunakan INNER JOIN dan SUBQUERY.

9.3 EXCEPT

Operasi EXCEPT digunakan untuk memperoleh perbedaan antara dua query yang hasilnya terdiri dari baris-baris yang hanya dimiliki query pertama namun tidak dimiliki query kedua.



Namun seperti INTERSECT di MYSQL tidak mendukung keyword EXCEPT. Oleh karena itu untuk memperoleh hasil seperti operasi EXCEPT di MYSQL dapat menggunakan NOT IN dan subquery.

Berikut contoh

```
SELECT nilai from class_a
WHERE nilai not in (SELECT nilai from class_b);
```

TUGAS PRAKTIKUM

1. Tampilkan nama, nama program studi, dan nim/nip dari dosen dan mahasiswa Ilmu Komputer!
2. Tampilkan nama, prodi, dan riwayat status yang pernah dijalani (pembimbing utama, pertama, penguji 1, penguji 2) dari setiap dosen, kemudian urutkan berdasarkan nama ASC.

	Nama	Prodi	Riwayat Status`
1	Supri bin Hj Amir, S.S...	Ilmu Komputer	penguji 2,penguji 1,pe...
2	Prof., Dr., Jeffry Kus...	Matematika	penguji 1
3	Prof., Dr. Eng., Mawar...	Matematika	penguji 1
4	Prof. Dr. Syamsuddin T...	Matematika	pembimbing utama
5	Prof. Dr. Hj. Aidawaya...	Matematika	pembimbing utama,pengu...
6	Prof. Dr. Amir Kamal A...	Matematika	pembimbing pertama
7	Nur Hilal A Syahrir, S...	Ilmu Komputer	penguji 2
8	Naimah Aris, S.Si.M.Ma...	Matematika	penguji 1,pembimbing p...
9	Musfira Putri Lukman,...	Ilmu Komputer	pembimbing pertama
10	Jusmawati Massalesse, ...	Matematika	pembimbing pertama,pen...
11	Edy Saputra R, S.Si, M...	Ilmu Aktuaria	penguji 2,penguji 1,pe...
12	Drs. Khaeruddin, MSc	Matematika	pembimbing utama
13	Dra. Nur Erawati, M.Si	Matematika	penguji 2
14	Dr., Hendra ,S.Si., M...	Ilmu Komputer	penguji 2,pembimbing p...
15	Dr., Budi Nurwahyu, MS	Matematika	penguji 1
16	Dr. Nurdin,. S.Si.M.Si	Matematika	pembimbing utama
17	Dr. Muhammad Zakir, M...	Matematika	pembimbing pertama
18	Dr. Muhammad Hasbi, MSc	Ilmu Komputer	penguji 1,penguji 2,pe...
19	Dr. Kasbawati, S.Si, M...	<null>	penguji 1,penguji 2
20	Dr. Firman, S.Si, M.Si	<null>	penguji 1,penguji 2
21	Dr. Eng. Armin Lawi, ...	Ilmu Komputer	penguji 1,pembimbing u...
22	Dr. Diaraya, M.Ak	Ilmu Komputer	penguji 1,pembimbing p...
23	Dr. Amran, S.Si., M.Si.	<null>	pembimbing pertama,pen...
24	Dr. Agustinus Ribal ,M...	Matematika	penguji 2,penguji 1
25	Andi Muhammad Anwar, S...	Ilmu Komputer	penguji 2,penguji 1
26	Andi Galsan Mahie,S.Si...	Matematika	pembimbing utama,pengu...
27	A. Muh Amil Siddik, S...	Ilmu Komputer	pembimbing pertama,pen...

BAB 10 | TRANSACTIONS

10.1 Pengertian Database Transaction

Transaction adalah fitur sistem manajemen basis data (DBMS) yang membantu pemulihan data pada kesalahan internal. Transaction pada basis data adalah kumpulan kueri yang harus dieksekusi sehingga jika salah satu eksekusi kueri gagal, eksekusi kueri Anda akan kembali ke awal. Transaction adalah mekanisme yang memungkinkan untuk menginterpretasikan beberapa perubahan pada database dengan satu operasi.

Pada MySQL, transaction dimulai dengan pernyataan **START TRANSACTION** dan diakhiri dengan pernyataan **COMMIT** atau **ROLLBACK**, serta dapat mengatur **SET autocommit** menjadi ya atau tidak.

Statement Transaction pada MYSQL

MySQL menggunakan statement penting berikut untuk mengontrol transaksi :

- **START TRANSACTION** digunakan untuk memulai sebuah transaction baru. selain itu kita juga dapat menggunakan **BEGIN** atau **BEGIN WORK** sebagai alias dari **START TRANSACTION**.
- **COMMIT** digunakan untuk menyimpan transaksi secara permanen di database.
- **ROLLBACK** merupakan perintah yang digunakan untuk mengembalikan database ke bentuk awal
- **SET autocommit** digunakan untuk menonaktifkan atau mengaktifkan mode **COMMIT** otomatis pada sesi saat ini.

Secara default, MySQL secara otomatis melakukan perubahan secara permanen ke database. Untuk memaksa MySQL agar tidak melakukan perubahan secara otomatis, digunakan pernyataan berikut:

```
SET autocommit = 0; atau SET autocommit = OFF;
```

Untuk mengaktifkan kembali mode auto commit dapat menggunakan statement:

```
SET autocommit = 1; atau SET autocommit = ON;
```

10.2 Cara menggunakan perintah COMMIT

Dengan menggunakan **TRANSACTION** saat melakukan perintah seperti **INSERT**, **UPDATE**, **DELETE** maka transaksi sebenarnya belum dilakukan secara permanen sehingga masih bisa di **rollback** / di batalkan. Jika tidak ada kesalahan maka seluruh rangkaian statement akan di **COMMIT** untuk menyimpan transaction secara permanen.

Untuk lebih jelasnya perhatikan contoh berikut :

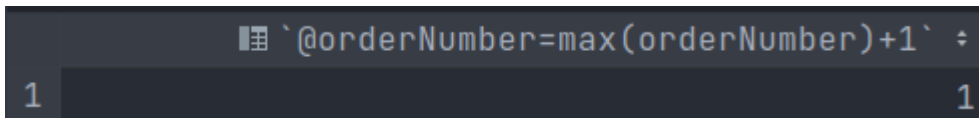
```
# 1. Mulai transaction baru
```

```

START TRANSACTION;
# 2. Mendapatkan order number terbaru
SELECT @orderNumber=max(orderNumber)+1 from
orders;
# 3. Menambahkan order baru untuk customer 145
INSERT INTO orders(orderNumber,
                    orderDate,
                    requiredDate,
                    shippedDate,
                    status,
                    customerNumber)
VALUES(@orderNumber,
       '2005-05-31',
       '2005-06-10',
       '2005-06-11',
       'In Process',
       145);
# 4. Commit Perubahan
COMMIT;

```

Query diatas akan menghasilkan output seperti berikut



1	1

Dapat kita lihat bahwa query tersebut akan melakukan :

Baris 1. Memulai transaction baru,

Baris 2. Mengambil atau menampilkan order number yang terbaru kemudian ditambah satu, kemudian memasukkan nilainya ke variable @orderNumber.

Baris 3. Menambahkan order baru ke tabel orders untuk customer 145, dengan nilai order number yang berasal dari nilai variabel @orderNumber.

Baris 4. Untuk menyimpan transaksi secara permanen di database maka digunakan **COMMIT**.

10.3 Cara menggunakan perintah ROLLBACK

Perintah **COMMIT** dan **ROLLBACK** saling berkaitan. Perintah **ROLLBACK** digunakan untuk mengembalikan database ke bentuk awal.

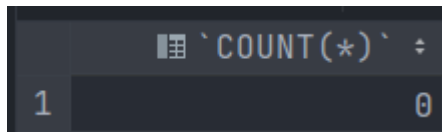
Contoh penggunaan **ROLLBACK**,

```

BEGIN;
DELETE FROM orderdetails;
SELECT COUNT(*) FROM orderdetails;

```

Outputnya akan seperti berikut



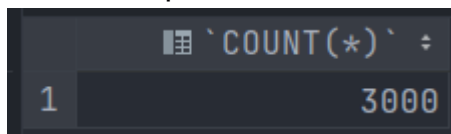
	1	2
1		0

Dapat dilihat bahwa pertama untuk memulai transaction digunakan statement **BEGIN**, kemudian dilakukan penghapusan semua isi tabel orderdetails. dengan menggunakan perintah **ROLLBACK** data orderdetails yang sudah di hapus **dapat dikembalikan**.

Berikut contohnya :

```
ROLLBACK;  
SELECT COUNT(*) FROM orderdetails;
```

Maka data pada tabel orderdetails kembali seperti berikut :



	1	2
1		3000

TUGAS PRAKTIKUM

1. Dengan menggunakan database “classicmodels”, tambahkan 3 data ke dalam tabel orders dan orderdetails setelah itu lakukan perintah commit:
2. Hapus seluruh data dari tabel orders details dan gunakan perintah ROLLBACK untuk mengembalikan data tersebut

BAB 11 | INDEX, CONTROL FLOW

11.1 INDEX

Index digunakan untuk mengoptimasi query pada database mysql. Dengan menggunakan index pada kolom - kolom yang sangat sering dijadikan target untuk memfilter data akan sangat mempersingkat waktu query pada database.

penggunaan index dianjurkan ketika terdapat kolom yang sering menjadi target filter, sebagai contoh misal kita memiliki sebuah database telepon dimana database ini menyimpan semua informasi tentang nama, alamat, dan no telepon seseorang. Andaikan terdapat ratusan ribu bahkan jutaan data di dalam database tersebut maka pada saat kita melakukan pencarian dengan query seperti berikut :

```
SELECT first_name, last_name, address, phone_number FROM phone_book
WHERE
first_name = 'Andi' AND last_name = 'Gunawan';
```

query di atas terlihat simple dan mungkin pada beberapa kasus dimana database nya tidak memiliki banyak data query tersebut akan memberikan nilai dengan cepat tapi tidak untuk kasus dengan banyak data karena secara default database mysql dan DBMS(*Database Management System*) yang lain umumnya akan melakukan *scanning* semua row hingga ditemukan data dengan nilai "first_name = 'Andi'" dan "last_name = 'Gunawan'". Disinilah digunakan *indexing* dimana *index* pada database sql menggunakan struktur data *B-tree* yang lebih optimal dalam melakukan pencarian.

contoh :

```
CREATE TABLE t(
  c1 INT PRIMARY KEY,
  c2 INT NOT NULL,
  c3 INT NOT NULL,
  c4 VARCHAR(10),
  INDEX index1 (c2,c3)
);
```

- index dapat memiliki 1 atau lebih kolom
- kita dapat membuat lebih dari 1 index dalam satu tabel

Menambah dan menghapus index :

```
-- menambah index
ALTER TABLE table
ADD INDEX index_name (column);

-- menghapus index
ALTER TABLE table
DROP INDEX index_name;
```

11.2 CONTROL FLOW

Control flow pada mysql memiliki konsep yang sama seperti pada bahasa pemrograman pada yang lain pada umumnya hanya saja dengan kemampuan yang lebih terbatas.

11.2.1 CASE...WHEN

CASE...WHEN mirip seperti switch case statement pada bahasa pemrograman java
syntax :

```
CASE value
  WHEN value1 THEN result1
  WHEN value2 THEN result2
  ...
  [ELSE else_result]
END
```

products

id	name	category
1	Bakso	Makanan
2	Es Kelapa	Minuman
3	Bensin	Lain - Lain

contoh :

```
SELECT name,
CASE category
  WHEN 'Makanan' THEN 'Ini makanan'
  WHEN 'Minuman' THEN 'Ini minuman'
  ELSE 'Bukan makanan atau minuman'
END
AS `Makanan atau Minuman ?`
FROM products;
```

output :

name	Makanan atau Minuman ?
Bakso	Ini makanan
Es Kelapa	Ini minuman
Bensin	Bukan makanan atau minuman

TUGAS PRAKTIKUM

Menggunakan database classicmodels.

Buatlah query untuk menampilkan output sebagai berikut :

customerName	Are you safe ?	total
Atelier graphique	you are in debt	-1314.36
Signal Gift Stores	you are in debt	-8380.98
Australian Collectors, Co.	you are in debt	-63285.07
La Rochelle Gifts	you are in debt	-40373.12
Baane Mini Imports	you are in debt	-22524.79
Mini Gifts Distributors Ltd.	you are in debt	-381327.34
Blauer See Auto, Co.	you are in debt	-16237.76
Mini Wheels Co.	you are in debt	-2110.56
Land of Toys Inc.	you are in debt	-34185.15
Euro+ Shopping Channel	you are in debt	-593089.54
Volvo Model Replicas, Co	you are in debt	-13594.82
Danish Wholesale Imports	you are in debt	-45685.12
Saveley & Henriot, Co.	you are in debt	-6405.35
Dragon Souvenirs, Ltd.	you are in debt	-52451.03
Muscle Machine Inc	you are in debt	-39413.95
Diecast Classics Inc.	you are in debt	-3758.69

penjelasan :

- kolom total adalah selisih customers.creditLimit dan banyak nya produk yang dibeli oleh customer (jumlah barang * harga per satuan barang)
- kolom "Are you safe ?" didapat dari mengecek nilai kolom total
 - jika kolom total > 0 maka tampilkan "you are safe"
 - jika kolom total < 0 maka tampilkan "you are in debt"
 - jika kolom total = 0 maka tampilkan "you are running out of credits"