



Back End Development 1

sesi 6



Database ⁺ Operations

INTRODUCTIONS Microsoft SQL Server

Microsoft SQL Server adalah sebuah program sistem manajemen basis data relasional (RDBMS) yang diciptakan oleh Microsoft.

Bahasa kueri utamanya adalah Transact-SQL yang merupakan implementasi dari SQL standar ANSI/ISO yang digunakan oleh Microsoft dan Sybase, saat ini SQL Server banyak digunakan di dunia bisnis berskala kecil hingga skala besar.

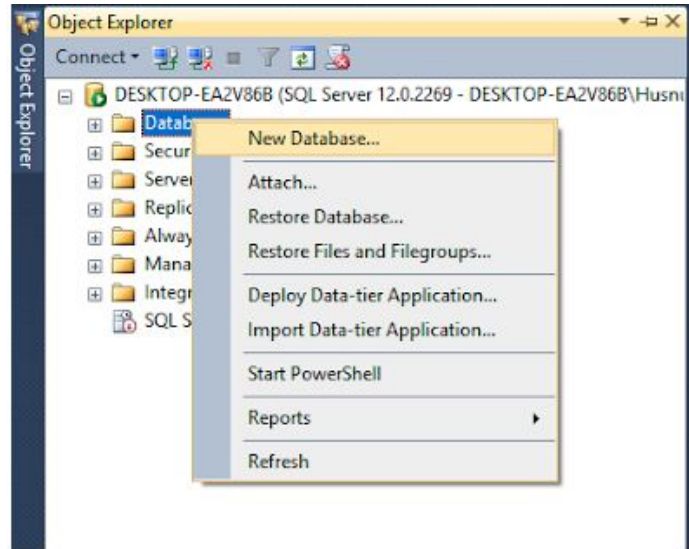
Di akhir sesi ke 5 teman-teman sudah dipastikan instalasi Microsoft SQL Server, Pada sesi 6 ini kita akan membahas SQL Dasar dengan Microsoft SQL Server, seperti : membuat database, CRUD Table dan membuat stored procedure.

Untuk itu buka Microsoft SQL Server kalian.

C# Database Operations - Sesi 06

Membuat Database

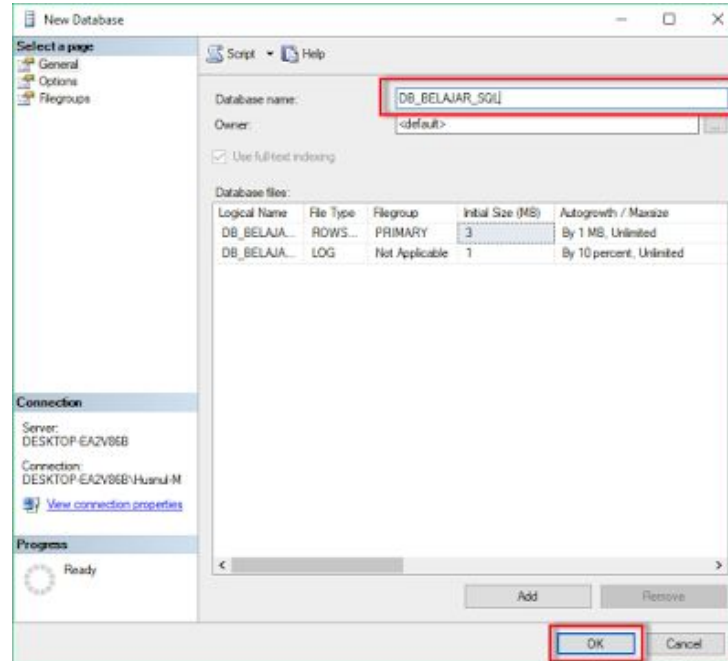
Untuk membuat database di SQL Server bisa dengan query atau dengan GUI, membuat database dengan GUI dengan cara seperti berikut :



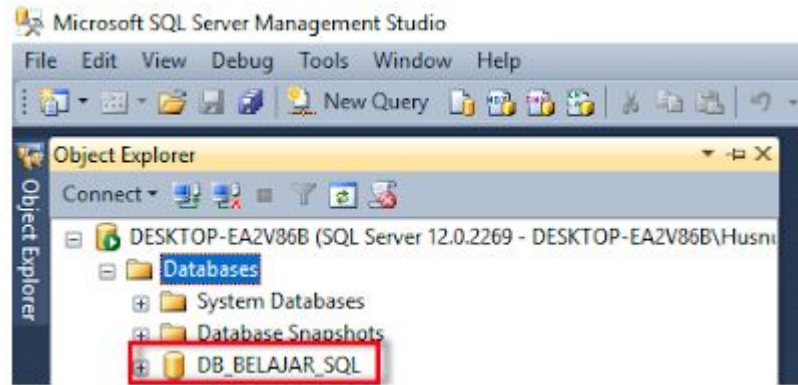
C# Database Operations - Sesi 06

Membuat Database

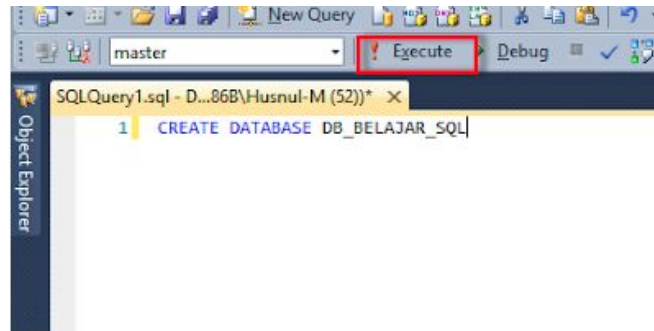
klik kanan pada databases dan pilih new database lalu isikan nama databasenya seperti dibawah ini dan klik OK



jika sudah maka akan muncul database baru di object explorer seperti gambar berikut :



FYI kita sudah berhasil membuat database dengan GUI, berikut cara membuat database dengan query SQL :



tinggal kita menuliskan source code seperti diatas dan klik execute.

C# Database Operations - Sesi 06

Create Table

Selanjutnya kita akan membuat table, untuk membuat table kita bisa menggunakan source code berikut :

```
CREATE TABLE TB_SISWA(  
NIS VARCHAR(10) PRIMARY KEY,  
NAMA VARCHAR(50) NULL,  
ALAMAT VARCHAR(100) NULL,  
TG_LAHIR DATE NULL,  
J_KELAMIN VARCHAR(1) NULL  
)
```

TB_SISWA merupakan nama table bisa diganti sesuai keinginan lalu klik execute, di dalam table TB_SISWA field NIS menjadi primary key, primary key ini bisa menjadi attribut field kunci dari table TB_SISWA. Mengupdate type data column / field pada table, disini kita akan coba mengganti type data field J_KELAMIN dari VARCHAR(1) menjadi VARCHAR(10), query-nya seperti berikut :

```
ALTER TABLE TB_SISWA ALTER COLUMN J_KELAMIN VARCHAR(10)
```

setelah menjalankan query diatas coba refresh table TB_SISWA pasti field j_kelamin akan berubah menjadi varchar(10).

C# Database Operations - Sesi 06

Insert, Update, Delete, Select

Setelah membuat table dan merubah type data field pada table kita akan lanjutkan dengan mengisikan, mengupdate, menghapus dan menampilkan data table TB_SISWA

untuk insert data bisa menggunakan query berikut :

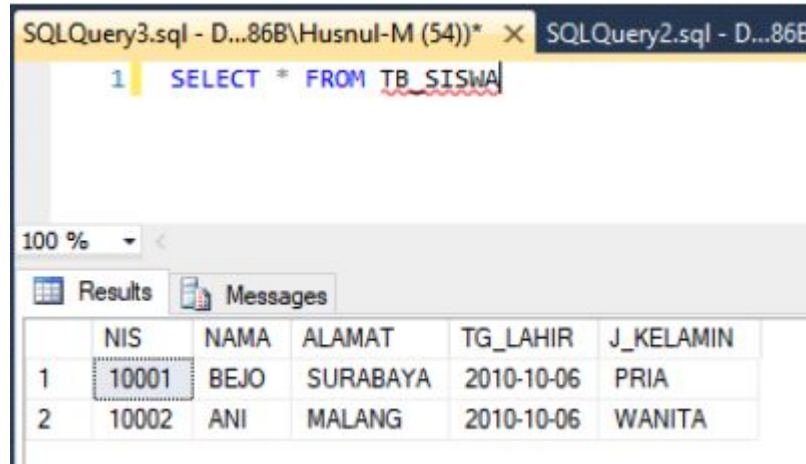
```
INSERT INTO TB_SISWA (NIS, NAMA, ALAMAT, TG_LAHIR, J_KELAMIN)
VALUES ('10001', 'BEJO', 'SURABAYA', '2010-10-06', 'PRIA');
```

```
INSERT INTO TB_SISWA (NIS, NAMA, ALAMAT, TG_LAHIR, J_KELAMIN)
VALUES ('10002', 'ANI', 'MALANG', '2010-10-06', 'WANITA');
```

untuk melihat hasilnya kita bisa menggunakan query select seperti berikut :

```
SELECT * FROM TB_SISWA
```


query diatas akan menampilkan semua data yang ada didalam table dan hasilnya seperti berikut :



The screenshot shows a SQL query editor with two tabs: 'SQLQuery3.sql - D...86B\Husnul-M (54))' and 'SQLQuery2.sql - D...86B'. The active tab contains the query: `1 SELECT * FROM TB_SISWA`. Below the query editor, there is a 'Results' tab showing a table with 6 columns: NIS, NAMA, ALAMAT, TG_LAHIR, and J_KELAMIN. The table contains two rows of data.

| | NIS | NAMA | ALAMAT | TG_LAHIR | J_KELAMIN |
|---|-------|------|----------|------------|-----------|
| 1 | 10001 | BEJO | SURABAYA | 2010-10-06 | PRIA |
| 2 | 10002 | ANI | MALANG | 2010-10-06 | WANITA |

jika kita hanya ingin menampilkan data siswa yang NIS nya 10001 maka querynya akan seperti berikut :

```
SELECT * FROM TB_SISWA WHERE NIS = '10001'
```

Dalam select ada banyak parameter yang bisa kita gunakan untuk menampilkan data dan kita bisa menggunakan lebih dari satu parameter adapun parameter yang bisa kita pakai antara lain sebagai berikut :

LIKE : like ini bisa kita gunakan untuk mencari karakter yang mirip dalam sebuah parameter contohnya kita ingin menampilkan data siswa yang alamatnya mengandung karakter 'SU'

```
SELECT * FROM TB_SISWA WHERE ALAMAT LIKE 'SU%'
```

BETWEEN : between ini biasanya digunakan untuk meng-filter berdasarkan tanggal, misalnya kita mau menampilkan data siswa yang tanggal lahirnya antara tanggal 09-10-2010 sampai 10-10-2010

```
SELECT * FROM TB_SISWA WHERE TG_LAHIR BETWEEN '09-10-2010' AND '10-10-2010'
```

selanjutnya kita akan mengupdate data, misalnya kita akan meng-update nama siswa yang NIS nya 10001

```
UPDATE TB_SISWA SET NAMA = 'BEJO SUTEDJO' WHERE NIS = '10001'
```

jika ingin menghapus data siswa, kita bisa langsung menghapus semua atau siswa tertentu, untuk menghapus data tertentu contoh query-nya seperti berikut, misal kita mau menghapus siswa dengan NIS 10002

```
DELETE TB_SISWA WHERE NIS = '10002'
```

kalau kita mau menghapus semua kita tidak perlu menggunakan parameter atau WHERE nya dihapus saja.

C# Database Operations - Sesi 06

Do It Yourself

Pada sesi ini, silahkan explorasi query database untuk database dibawah ini :

- Buat Database baru dengan nama staff
- Buat Table data_staff dengan desain spt dibawah ini :

| | |
|-----------|--------------|
| nik | int |
| nama | varchar(50) |
| alamat | varchar(200) |
| handphone | varchar(15) |

- Masukkan data ke dalam table data_staff sesuai dengan table diatas
- Masukkan lebih dari 1 data ke dalam table data masih dalam table diatas.
- Tambahkan kolom baru pada table data_staff yaitu joindate sekaligus masukan 1 data kedalam table data_staff

C# Database Operations - Sesi 06

Do It Yourself

- F. Tampilkan 2 data SQL kalian sekarang
- G. Tampilkan 3 data SQL kalian sekarang
- H. Buat Table baru staffoutsorce dengan isi yang sama seperti data_staff
- I. Masukkan 10 data baru ke table staffoutsorce
- J. Tampilkan data pada 2 Table yang sudah kalian buat dengan joindate yang sama
- K. Tampilkan seluruh data sebelah kanan yang sama pada table staff_outsource
- L. Tampilkan seluruh data staff sebelah kiri yang punya nilai tidak sama akan bernilai null
- M. Tampilkan seluruh data antara 2 tabel baik itu tidak punya kesamaan dan bernilai null.

Lakukan sambil praktek dalam 45 menit.

C# Database Operations - Sesi 06

Relasi Database

Pada saat kita membuat sebuah database, kita akan gunakan tabel-tabel terpisah dengan berbagai jenis entitas.

Contoh :

Customers, orders, product, dll

PENTING : Setiap tabel harus memiliki relasi, misalnya :

Customer membuat Orders dan Orders berisi produk

Pada sesi kali ini kita akan membahas :

- A. Relasi One To One
- B. Relasi One to Many dan Many to One
- C. Relasi Many to Many
- D. Relasi Referensi Mandiri

Dan tidak lupa kita akan menggunakan beberapa Query JOIN untuk mendapatkan apa yang kita butuhkan.

C# Database Operations - Sesi 06

Relasi One to One

Silahkan Perhatikan table customers dibawah ini.

| customer_id | customer_name | customer_address |
|-------------|---------------|--------------------------------|
| 101 | John Doe | Jalan Angkasa Raya 12 , Depok |
| 102 | Bruce Wayne | Jalan MH Thamrin 3 , Tangerang |

Apakah ada relasi dari tabel diatas?

YA, Kita bisa tempatkan customer_address pada tabel yang terpisah

| customer_id | customer_name | address_id |
|-------------|---------------|------------|
| 101 | John Doe | 301 |
| 102 | Bruce Wayne | 302 |

Tabel Customer

| address_id | address |
|------------|--------------------------------|
| 301 | Jalan Angkasa Raya 12 , Depok |
| 302 | Jalan MH Thamrin 3 , Tangerang |

Tabel Address

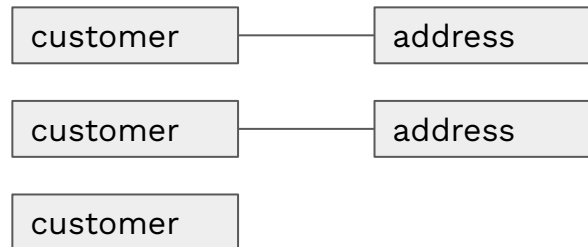
Sekarang kita memiliki relasi antara tabel Customers dan tabel Addresses. Jika address masing-masing hanya dapat menjadi milik satu customer, relasi ini adalah "One to One".

Perlu diingat bahwa relasi semacam ini sangat tidak umum.

Tabel awal kita yang disertakan address bersama dengan customer bisa bekerja baik dalam kebanyakan kasus.

Perhatikan bahwa sekarang ada sebuah field yang bernama "address_id" di tabel Customers, yang mengacu pada record yang cocok dalam tabel Address. Ini disebut "Foreign Key" dan digunakan untuk semua jenis relasi database.

Kita dapat memvisualisasikan relasi antara record customer dan address seperti ini:



Relasi One to Many dan Many to One

Ini adalah jenis yang paling umum digunakan dari relasi. Contoh sebuah situs web e-commerce, dengan berikut:

- Customers dapat membuat banyak orders.
- Orders dapat berisi banyak produk.
- Produk dapat memiliki deskripsi dalam banyak bahasa.

Dalam kasus ini kita akan perlu untuk membuat relasi "One to Many". Berikut adalah contohnya:

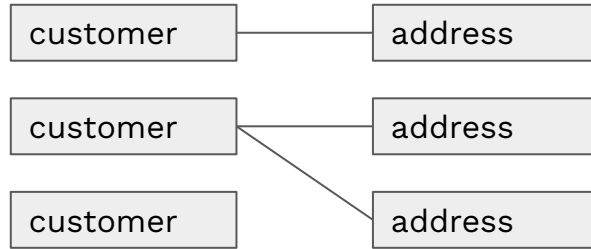
| customer_id | customer_name |
|-------------|---------------|
| 101 | John Doe |
| 102 | Bruce Wayne |

Tabel Customer

| order_id | customer_id | order_date | amount |
|----------|-------------|------------|-------------|
| 555 | 101 | 8/24/21 | Rp. 159.500 |
| 556 | 102 | 9/05/21 | Rp. 346.000 |
| 557 | 102 | 9/14/21 | Rp. 79.000 |

Tabel Orders

Setiap customer dapat memiliki nol, satu atau beberapa orders.
Tapi sebuah order dapat hanya menjadi milik satu customer.



Relasi Many to Many

Dalam beberapa Kasus, kita perlu beberapa contoh relasi many to many, contoh :
Setiap order dapat berisi beberapa produk, dan masing-masing produk juga bisa dalam beberapa orders.

Nah untuk relasi ini kita perlu buat tabel tambahan :

| order_id | customer_id | order_date | amount |
|----------|-------------|------------|-------------|
| 555 | 101 | 8/24/21 | Rp. 159.500 |
| 556 | 102 | 9/05/21 | Rp. 346.000 |
| 557 | 102 | 9/14/21 | Rp. 79.000 |

Tabel Orders

| produk_id | produk_name | produk_desc |
|-----------|-------------|--------------------------|
| 201 | Base Shirt | Base Shirt Anime Design |
| 202 | Jeans | Slimfit Jeans |
| 203 | Neklace | Elegant Silver - Neklace |

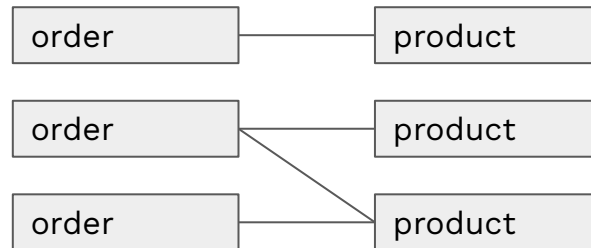
Tabel Products

| order_id | product_id |
|----------|------------|
| 555 | 201 |
| 555 | 202 |
| 556 | 202 |
| 556 | 203 |

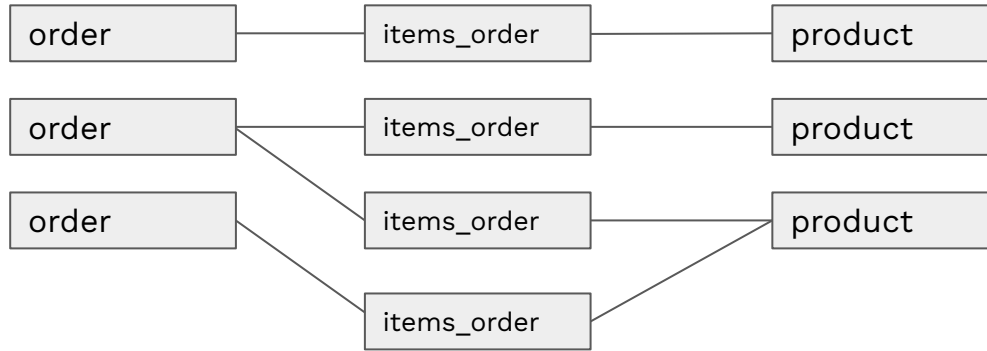
Tabel Items Orders

Tabel Items_Orders hanya memiliki satu tujuan, dan itu adalah untuk menciptakan sebuah relasi "Many to Many" antara products dan orders.

Berikut adalah bagaimana kita dapat memvisualisasikan relasi semacam ini:



Jika Anda ingin menyertakan record items_orders pada grafik, ini dapat terlihat seperti ini:



C# Database Operations - Sesi 06

Relasi Referensi Mandiri

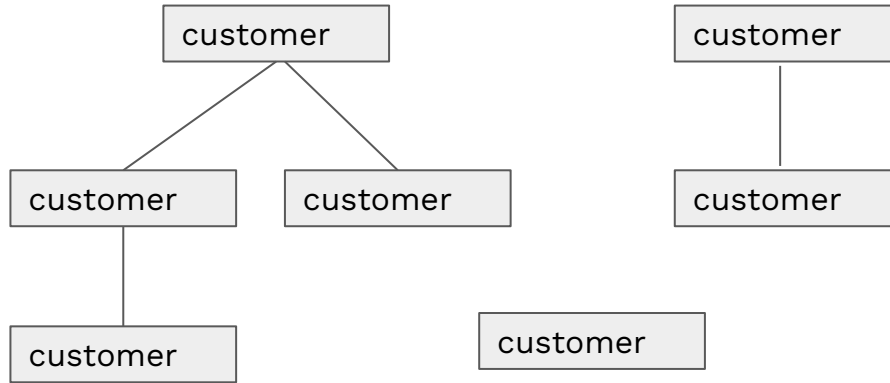
Ini digunakan ketika sebuah tabel membutuhkan untuk memiliki relasi dengan dirinya sendiri. Sebagai contoh, katakanlah Anda memiliki program referral. Customers dapat merujuk customers lain ke situs belanja Anda. Tabel dapat terlihat seperti ini:

| customer_id | customer_name | referrer_customer_id |
|-------------|---------------|----------------------|
| 101 | John Doe | 0 |
| 102 | Bruce Wayne | 101 |
| 103 | Adam Smith | 101 |

Customers 102 dan 103 dirujuk oleh customer 101.

Ini sebenarnya juga dapat mirip dengan relasi "one to many" karena satu pelanggan dapat merujuk ke beberapa pelanggan.

Juga hal ini dapat divisualisasikan seperti struktur pohon:



Satu customer dapat merujuk nol, satu atau beberapa customers. Setiap customer bisa dirujuk juga dengan hanya satu customer, atau tidak sama sekali.

Jika Anda ingin membuat referensi mandiri relasi "many to many", Anda akan membutuhkan sebuah tabel tambahan seperti yang kita bicarakan dalam bagian sebelumnya.

C# Database Operations - Sesi 06

Foreign Key

Sejauh ini kita telah belajar tentang beberapa konsep. Sekarang adalah waktunya menghidupkan mereka dengan menggunakan SQL. Untuk bagian ini, kita perlu memahami apa itu Foreign Key.

Dalam contoh-contoh relasi di atas, kita selalu memiliki field ini "****_id" yang direferensikan sebuah kolom di tabel yang lain.

Dalam contoh ini, kolom customer_id dalam tabel Orders adalah kolom Foreign Key:

| customer_id | customer_name |
|-------------|---------------|
| 101 | John Doe |
| 102 | Bruce Wayne |

Tabel Customer

| order_id | customer_id | order_date | amount |
|----------|-------------|------------|-------------|
| 555 | 101 | 8/24/21 | Rp. 159.500 |
| 556 | 102 | 9/05/21 | Rp. 346.000 |
| 557 | 102 | 9/14/21 | Rp. 79.000 |

Tabel Orders



Dengan database seperti MySQL, ada dua cara untuk membuat kolom foreign key kolom:

```
1 CREATE TABLE customers (  
2     customer_id INT AUTO_INCREMENT PRIMARY KEY,  
3     customer_name VARCHAR(100)  
4 );
```

Sekarang tabel orders, yang akan berisi Foreign Key:

```
1 CREATE TABLE orders (  
2     order_id INT AUTO_INCREMENT PRIMARY KEY,  
3     customer_id INT,  
4     amount DOUBLE,  
5     FOREIGN KEY (customer_id) REFERENCES customers(customer_id)  
6 );
```

Kedua kolom (customers.customer_id dan orders.customer_id) harus dengan struktur data yang sama persis. Jika salah satunya adalah INT, yang lain tidak boleh BIGINT misalnya.

Harap dicatat bahwa di MySQL hanya mesin InnoDB yang memiliki dukungan penuh untuk Foreign Key. Tapi mesin penyimpanan lainnya masih akan memungkinkan Anda untuk menentukan mereka tanpa ada kesalahan.

Juga kolom Foreign Key adalah yang diindeks secara otomatis, kecuali jika Anda menentukan indeks lain untuknya.

C# Database Operations - Sesi 06

Query Join

Untuk mengambil data dari database yang memiliki relasi, kita sering perlu untuk menggunakan query JOIN.

Sebelum kita mulai, mari kita membuat tabel dan beberapa sampel data untuk bekerja dengannya.

```
1 CREATE TABLE customers (  
2     customer_id INT AUTO INCREMENT PRIMARY KEY,  
3     customer_name VARCHAR(100)  
4 );  
5  
6 CREATE TABLE orders (  
7     order_id INT AUTO INCREMENT PRIMARY KEY,  
8     customer_id INT,  
9     amount DOUBLE,  
10    FOREIGN KEY (customer_id) REFERENCES customers(customer_id)  
11 );  
12  
13 INSERT INTO belajar.dbo.customers VALUES  
14 (1, 'Adam'),  
15 (2, 'Andy'),  
16 (3, 'Joe'),  
17 (4, 'Sandy');  
18  
19 INSERT INTO belajar.dbo.orders VALUES  
20 (1, 1, 149.000),  
21 (2, 1, 535.000),  
22 (3, 3, 176.500),  
23 (4, 4, 89.000);
```



Kita memiliki 4 customers. Satu customer memiliki dua orders, dua customers memiliki satu order masing-masing, dan salah satu customer tidak memiliki order.

Sekarang mari kita lihat berbagai jenis query JOIN yang dapat kita jalankan di tabel ini.

Cross Join

Ini adalah default tipe query JOIN ketika tidak ada kondisi yang ditentukan.

```
25  SELECT * FROM customers JOIN orders;
```

Hasilnya adalah apa yang disebut "produk Cartesian" dari tabel. Itu berarti bahwa setiap baris dari tabel pertama dicocokkan dengan setiap baris dari tabel kedua. Karena setiap tabel memiliki 4 baris, kami akhirnya mendapatkan hasil 16 baris.

Kata kunci JOIN secara opsional dapat diganti dengan koma sebagai gantinya.

```
27  -- SELECT * FROM customers, orders;  
--
```



Natural Join

Dengan query JOIN semacam ini, tabel harus memiliki pencocokan nama kolom.

Dalam kasus ini, kedua tabel memiliki kolom `customer_id`.

Jadi, MySQL akan menggabung record hanya ketika nilai dari kolom ini adalah cocok pada dua record.

```
29  SELECT * FROM customers NATURAL JOIN orders;
```

Seperti yang Kita lihat kolom `customer_id` ini hanya ditampilkan sekali saat ini, karena mesin database memperlakukan ini sebagai kolom umum.

Kita dapat melihat dua orders yang dilakukan oleh Adam, dan dua orders lainnya oleh Joe dan Sandy.

Akhirnya kita mendapatkan beberapa informasi yang berguna.



Inner Join

Ketika penggabungan dengan kondisi yang ditentukan, Inner Join dilakukan.

Dalam kasus ini, akan menjadi ide yang baik untuk memiliki field `customer_id` yang cocok pada kedua tabel.

Hasilnya harus sama dengan Natural Join.

```
31 SELECT * From Customers JOIN orders WHERE customers.customer_id = orders.customer_id;
```

Hasil yang sama kecuali ada sedikit perbedaan. Kolom `customer_id` ini diulang dua kali, sekali untuk setiap tabel. Alasannya adalah, kita hanya bertanya database untuk mencocokkan nilai-nilai pada dua kolom.

Tetapi itu adalah benar-benar tidak menyadari bahwa mereka mewakili informasi yang sama. Mari kita tambahkan beberapa kondisi lagi ke query.

```
33 SELECT * From Customers JOIN orders WHERE customers.customer_id = orders.customer_id AND orders.amount > 120000;
```

Kali ini kami hanya menerima orders lebih dari Rp 120.000 .

Klausula ON

Sebelum pindah ke penggabungan jenis lainnya, kita perlu melihat pada klausa ON. Ini berguna untuk menempatkan kondisi JOIN di klausa terpisah.

```
35 SELECT * From Customers JOIN orders ON <customers.customer_id = orders.customer_id> WHERE orders.amount > 120000;
```

Sekarang kita dapat membedakan kondisi JOIN dari klausa kondisi WHERE. Tapi juga ada sedikit perbedaan dalam fungsi. Kita akan melihatnya di contoh-contoh LEFT JOIN.

