NAMA : MUH GUFRAN HAKIM

NIM : 200250502098

**SQL**

SQL atau Standard Query Language adalah bahasa pemrograman yang digunakan dalam mengakses, mengubah, dan memanipulasi data yang berbasis relasional. Query merupakan kata benda (noun) yang berarti pertanyaan. Dalam pengolahan database, query merupakan suatu istilah yang merujuk pada permintaan pengguna untuk memperoleh informasi dari database.SQL banyak diterapkan pada pemrograman berbasis client-server seperti PHP, ASP, Java J2EE, dan sebagainya. Contoh software SQL yang terkenal adalah MySQL, MsSQL (Microsoft), Oracle SQL, Postgre SQL (open source). Masing-masing software mempunyai keunggulan sendiri- sendiri, sejauh yang saya tahu Oracle SQL handal dalam hal keamanan dan ukuran database yang bisa mencapai tera byte, sedang MsSQL lebih banyak bermain di Memory untuk processing. Dari ketiga software ini, Oracle SQl bisa dikatakan sebagai pemegang pertama. Perintah dasar SQL pada dasarnya hampir sama baik MySQL, Postgre SQL, MsSQL atau Oracle SQL.

* Apa itu SQL ?
* SQL adalah singkatan dari Structured Query Language
* SQL memungkinkan Anda mengakses dan memanipulasi database
* SQL adalah standar ANSI (American National Standards Institute)
* Apa kegunaan SQL ?
* SQL dapat mengeksekusi query terhadap database
* SQL bisa mengambil data dari database
* SQL bisa memasukkan record dalam database
* SQL bisa mengupdate record dalam database
* SQL dapat menghapus record dari database
* SQL bisa membuat database baru
* SQL dapat membuat tabel baru dalam database
* SQL dapat membuat prosedur tersimpan dalam database
* SQL dapat membuat tampilan dalam database
* SQL dapat mengatur hak akses pada tabel, prosedur, dan tampilan
* SQL adalah sebuah standar

Meskipun SQL adalah standar ANSI (American National Standards Institute), ada beberapa versi bahasa SQL yang berbeda. Namun, agar sesuai dengan standar ANSI, semuanya mendukung setidaknya perintah utama (seperti SELECT, UPDATE, DELETE, INSERT, WHERE) dengan cara yang sama.

* SQL di situs web

Untuk membangun situs web yang menampilkan data dari database, Anda memerlukan :

* Program database RDBMS (yaitu MS Access, SQL Server, MySQL)
* Untuk menggunakan bahasa script sisi server, seperti PHP atau ASP
* Untuk menggunakan SQL dengan mendapatkan data yang anda inginkan
* Untuk menggunakan HTML / CSS ke style halaman
* RDBMS
* RDBMS adalah singkatan dari Relational Database Management System.
* RDBMS adalah basis untuk SQL, dan untuk semua sistem database modern seperti MS SQL Server, IBM DB2, Oracle, MySQL, dan Microsoft Access.
* Data dalam RDBMS disimpan dalam objek database yang disebut tabel. Tabel adalah kumpulan entri data terkait dan terdiri dari kolom dan baris.

1. **Syntax SQL**

Dilihat dari segi penggunaan bahasa syntax biasa juga disebut sebagai sintaks, dapat diartikan sebagai aturan yang harus dipenuhi demi tercapainya tujuan seorang programmer. Maksud adanya sintaks atau aturan ini adalah agar kode atau script yang ditulis dapat dibenarkan dan dimengerti oleh komputer. Berikut adalah syntax dalam bahasa SQL :

* 1. Perintah melihat database : *show databases*;
  2. Perintah membuat database : Contoh *: create kuliah*;
  3. Perintah menggunakan database yang telah dibuat : Use namadatabase; *Contoh : use kuliah;*
  4. Perintah membuat table : Create table namatabel (namafield1 Type(length), namafield2 type(length),primary key(namafield1)); Contoh :

*create table mahasiswa(npm char(9), nama\_mahasiswa varchar(20),umur int(2),alamat text, primary key(npm));screate table mata\_kuliah(kode\_matkul char(5), nama\_matkul varchar(30), primary key(kode\_matkul));*

*create table krs(npm char(9), kode\_matkul char(5), semester int(2), tahun\_ajaran char(10), primary key (npm,kode\_matkul,semester));*

* 1. Perintah melihat table apa saja yang ada pada database : *show tables*;
  2. Perintah melihat deskripsi table/melihat kolom apa saja beserta type, length yang ada pada table : Desc namatabel; Contoh : *desc mahasiswa*;
  3. Perintah melihat deskripsi table/melihat kolom apa saja beserta type, length yang ada pada table : Desc namatabel; Contoh : desc mahasiswa;
  4. Perintah menambah field pada table : Alter table namatabel add namafield type(length); kemudian enter : Contoh :

*alter table mahasiswa add jenis\_kelamin int(1);*

* 1. Perintah menghapus table :  
     drop table namatabel; contoh : *drop table mahasiswa;onamslect*
  2. Perintah mengganti nama field pada table : Alter table namatabel change namafield\_lama namafield\_baru type(length); Contoh :

*alter table mahasiswa change nama\_mahasiswa name\_student varchar(20);*

* 1. Perintah mengubah ukuran (length) dari suatu field :  
     Alter table namatabel modify namafield type(length) not null; Contoh :

*alter table mahasiswa modify name\_student varchar(30);*

* 1. Perintah menghapus field pada table : Alter table namatabel drop namafield; Contoh: *alter table mahasiswa drop jenis\_kelamin;*
  2. Perintah menginput satu data pada table langsung untuk semua field :  
     Insert into namatabel values(‘data1’,’dataA’); Contoh : *insert into mahasiswa values(‘G1A008023’,’Wiwid KZ’,’20’,’Bengkulu’);*
  3. Perintah menginput beberapa data pada table langsung untuk semua field :  
     Insert into namatabel values (‘data1’,’dataA’),(‘data2’,’dataB’); Contoh :

insert into mahasiswa values(‘A1F006010’,’Sanity KZ’,’23’,’Bengkulu’),(‘G1A008023’,’Wiwid KZ’,’20’,’Bengkulu’);

* 1. Perintah menginput data pada table hanya untuk field tertentu : Insert into namatabel (namafield) values(‘data1),(‘data2’); Contoh : *insert into mahasiswa (npm) values(‘F1A006010’),(‘GiA008023’);*
  2. Perintah melihat isi table keseluruhan :  
     Select \* from namatabel; Contoh : *select \* from mahasiswa;*
  3. Perintah menampilkan isi table hanya untuk field tertentu saja :  
     Select namafield from namatabel; Contoh : *select npm from mahasiswa;*
  4. Perintah isi/data lengkap table pada baris tertentu saja : Select \* from namatabel where namafield\_key=’data’; Contoh *: select \* from mahasiswa where npm=’G1A008023’;*
  5. Perintah mengubah isi dari field yang diinginkan : Update namatabel set namafield=’data\_baru’ where namafield\_key=’data\_baru’; Contoh : *update mahasiswa set name\_student=’Sanity Afsyawida KZ’ where npm=’A1F006010’;*
  6. Perintah menghapus baris yang diinginkan: Delete from namatabel where namafield\_key=’data’; Contoh : *delete from mahasiswa where npm=’G1A008023’;*
  7. Perintah menggabungkan beberapa tabel:  
     Alter table namatabel1 add foreign key(namafield\_key) references namatabel2(namafield\_key); Contoh : *alter table mahasiswa add foreign key(npm) references krs(npm);alter table mata\_kuliah add foreign key(kode\_matkul) references krs(kode\_matkul);*
  8. Perintah memanggil beberapa data dari beberapa tabel :  
     Select namafield1,namafield2,namafield3 from namatabel1 inner join namatabel2 where namatabel1.namafield\_key=namatabel2.namafield\_key; Contoh : *select a.npm,a.nama,b.nama\_matkul,c.semester from mahasiswa a inner join mata\_kuliah b inner join krs c where a.npm=c.npm and b.kode\_matkul=c.kode\_matkul;*
  9. Perintah memanggil isi tabel yang dikelompokkan berdasarkan field tertentu:  
     Select \* from namatabel group by namafield; Contoh : *select \* from mahasiswa group by npm;*
  10. Perintah memanggil isi tabel yang diurutkan secara ascending descending  
      Select \* from namatabel order by namafield;  
      Select \* from namatabel order by namafield desc; Contoh : *select \* from mahasiswa order by npm desc;*
  11. Perintah mencari nilai maksimum pada suatu field pada tabel tertentu :  
      Select max(namafield) from namatabel; Contoh : *select max(umur) from mahasiswa;*
  12. Perintah mencari nilai minimum pada suatu field pada tabel tertentu :  
      Select min(namafield) from namatabel; Contoh : *select min(umur) from mahasiswa;*
  13. Perintah mencari banyaknya record/data dalam sebuah field pada sebuah tabel :  
      Select count(namafield) from namatabel; Contoh : *select count(npm) from mahasiswa;*
  14. Perintah menjumlahkan isi dari field:  
      Select sum(namafield) from namatabel; Contoh : *select sum(umur) from mahasiswa;*
  15. Perintah mencari rata2 nilai isi dari suatu field :  
      Select avg(namafield) from namatabel; Contoh : *select avg (umur) from mahasiswa;*
  16. Perintah menentukan standard deviasi isi dari suatu field:  
      Select std(namfield) from namatabel; Contoh : *select std(umur) from mahasiswa;*
  17. perintah untuk mengurutkan data dari atas: *select nim,nama,kelas,jurusan from Mahasiswa order by NIM;*
  18. perintah untuk mengurutkan data dari bawah*: select nim,nama,kelas,jurusan from Mahasiswa order by NIM desc;*

1. **SQL Select**

SELECT digunakan untuk memilih data dari database. Data yang dikembalikan disimpan dalam tabel hasil, yang disebut kumpulan hasil. Syntaxnya yaitu :

SELECT kolom1, kolom2, ...

FROM nama\_tabel;

Di sini, kolom1, kolom2, ... adalah nama bidang dari tabel yang ingin Anda pilih datanya. Jika Anda ingin memilih semua bidang yang tersedia dalam tabel, gunakan sintaks berikut:

SELECT \* FROM nama\_tabel;

Contoh :

Di bawah tabel "Pelanggan" yang sudah kamu buat dengan nama database\_Penjualan di SQL :

|  |  |  |  |
| --- | --- | --- | --- |
| id | Namapelanggan | Alamat | Barang\_yang\_dibeli |
| 001 | Aco | Jl. Sultan Hasanuddin, Binanga, Kabupaten Mamuju, Sulawesi Barat | Kertas HVS |
| 002 | Mina | Tapalang, Mamuju, Sulawei barat | Pulpen |
| 003 | Nana | Jl. Pattalundru, Binanga, Kabupaten Mamuju, Sulawesi Barat | Pulpen |
| 004 | Nina | Jl. WR. Monginsidi, Binanga, Kabupaten Mamuju, Sulawesi Barat | Buku |

contoh Select Kolom

berikut adalah Select untuk memilih kolom "NamaPelanggan" dan "Barang\_yang\_dibeli" dari tabel "Pelanggan":

SELECT Namapelanggan,Barang\_yang\_dibeli FROM Pelanggan;

Maka akan muncul tabel berikut :

|  |  |
| --- | --- |
| Namapelanggan | Barang\_yang\_dibeli |
| Aco | Kertas HVS |
| Mina | Pulpen |
| Nana | Pulpen |
| Nina | Buku |

Contoh penggunaan Penggunaan SELECT \*

SQL berikut memilih semua kolom dari tabel "Pelanggan":

SELECT \* FROM Pelanggan;

Maka akan muncul tabel seperti berikut :

|  |  |  |  |
| --- | --- | --- | --- |
| id | Namapelanggan | Alamat | Barang\_yang\_dibeli |
| 001 | Aco | Jl. Sultan Hasanuddin, Binanga, Kabupaten Mamuju, Sulawesi Barat | Kertas HVS |
| 002 | Mina | Tapalang, Mamuju, Sulawei barat | Pulpen |
| 003 | Nana | Jl. Pattalundru, Binanga, Kabupaten Mamuju, Sulawesi Barat | Pulpen |
| 004 | Nina | Jl. WR. Monginsidi, Binanga, Kabupaten Mamuju, Sulawesi Barat | Buku |

## SQL SELECT DISTINCT

Perintah SELECT DISTINCT digunakan untuk menampilkan nilai yang berbeda. Di dalam tabel, kolom sering kali berisi banyak nilai ysng sama atau duplikat; dan terkadang Anda hanya ingin membuat daftar nilai yang berbeda. Syntax Select distinct yaitu :

SELECT DISTINCT kolom1, kolom2,...  
FROM nama\_tabel;

Contoh:

Di bawah tabel "Pelanggan" yang sudah kamu buat dengan nama database\_Penjualan di SQL:

|  |  |  |  |
| --- | --- | --- | --- |
| id | Namapelanggan | Alamat | Barang\_yang\_dibeli |
| 001 | Aco | Jl. Sultan Hasanuddin, Binanga, Kabupaten Mamuju, Sulawesi Barat | Kertas HVS |
| 002 | Mina | Tapalang, Mamuju, Sulawei barat | Pulpen |
| 003 | Nana | Jl. Pattalundru, Binanga, Kabupaten Mamuju, Sulawesi Barat | Pulpen |
| 004 | Nina | Jl. WR. Monginsidi, Binanga, Kabupaten Mamuju, Sulawesi Barat | Buku |

SELECT DISTINCT Barang\_yang\_dibeli FROM Pelanggan;

|  |
| --- |
| Barang\_yang\_dibeli |
| Kertas HVS |
| Pulpen |
| Buku |

# SQL WHERE

Perintah WHERE digunakan untuk menyaring atau memfilter record (baris). Perintah WHERE digunakan untuk mengekstrak hanya record yang memenuhi persyaratan tertentu yang akan tampil. Syntax WHERE yaitu :

SELECT column1, column2, ...  
FROM table\_name  
WHERE condition;

Contoh :

Di bawah tabel "Pelanggan" yang sudah kamu buat dengan nama database\_Penjualan di SQL:

|  |  |  |  |
| --- | --- | --- | --- |
| id | Namapelanggan | Alamat | Barang\_yang\_dibeli |
| 001 | Aco | Jl. Sultan Hasanuddin, Binanga, Kabupaten Mamuju, Sulawesi Barat | Kertas HVS |
| 002 | Mina | Tapalang, Mamuju, Sulawei barat | Pulpen |
| 003 | Nana | Jl. Pattalundru, Binanga, Kabupaten Mamuju, Sulawesi Barat | Pulpen |
| 004 | Nina | Jl. WR. Monginsidi, Binanga, Kabupaten Mamuju, Sulawesi Barat | Buku |

Berikut adalah syntax untuk memilih semua pelanggan yang membeli pulpen, di tabel "Pelanggan":

SELECT \* FROM Pelanggan  
WHERE Barang\_yang\_dibeli ='Pulpen';

Maka akan muncul tabel berikut :

|  |  |  |  |
| --- | --- | --- | --- |
| id | Namapelanggan | Alamat | Barang\_yang\_dibeli |
| 002 | Mina | Tapalang, Mamuju, Sulawei barat | Pulpen |
| 003 | Nana | Jl. Pattalundru, Binanga, Kabupaten Mamuju, Sulawesi Barat | Pulpen |

# Operator SQL AND, OR and NOT

Klausa WHERE dapat digabungkan dengan operator AND, OR, dan NOT. Operator AND dan OR digunakan untuk memfilter record berdasarkan lebih dari satu kondisi:

* Operator AND menampilkan record jika semua kondisi yang dipisahkan oleh AND adalah TRUE/benar.
* Operator OR menampilkan record jika salah satu kondisi yang dipisahkan oleh OR adalah TRUE/benar.
* Operator NOT menampilkan record jika kondisi NOT TRUE/tidak benar.

Syntax dari operator AND, OR, dan NOT :

AND

SELECT kolom1, kolom2, ...  
FROM nama\_tabel  
WHERE kondisi1 AND kondisi2 AND kondisi3 ...;

OR

SELECT kolom1, kolom2, ...  
FROM nama\_tabel  
WHERE kondisi1 OR kondisi2 OR kondisi3 ...;

NOT

SELECT kolom1, kolom2, ...  
FROM nama\_tabel  
WHERE NOT kondisi;

Contoh :

Di bawah tabel "Pelanggan" yang sudah kamu buat dengan nama database\_Penjualan di SQL:

|  |  |  |  |
| --- | --- | --- | --- |
| id | Namapelanggan | Alamat | Barang\_yang\_dibeli |
| 001 | Aco | Mamuju | Kertas HVS |
| 002 | Mina | Majene | Pulpen |
| 003 | Nana | Makassar | Pulpen |
| 004 | Nina | Makassar | Buku |
| 005 | Anto | Mamuju | Buku |
| 006 | Riri | Palu | Kertas HVS |

Contoh Penggunaan AND :

SELECT \* FROM Pelanggan  
WHERE Alamat='Mamuju' AND Barang\_yang\_dibeli='Buku';

Maka akan muncul tabel berikut :

|  |  |  |  |
| --- | --- | --- | --- |
| id | Namapelanggan | Alamat | Barang\_yang\_dibeli |
| 005 | Anto | Mamuju | Buku |

Contoh Penggunaan OR :

SELECT \* FROM Pelanggan  
WHERE Alamat='Mamuju' OR Alamat='Makassar';

Maka akan muncul tabel berikut :

|  |  |  |  |
| --- | --- | --- | --- |
| id | Namapelanggan | Alamat | Barang\_yang\_dibeli |
| 001 | Aco | Mamuju | Kertas HVS |
| 003 | Nana | Makassar | Pulpen |
| 004 | Nina | Makassar | Buku |
| 005 | Anto | Mamuju | Buku |

Contoh lain dari penggunaan OR :

SELECT \* FROM Pelanggan  
WHERE Barang\_yang\_dibeli='Pulpen' OR Barang\_yang\_dibeli ='buku';

Maka akan muncul tabel berikut :

|  |  |  |  |
| --- | --- | --- | --- |
| id | Namapelanggan | Alamat | Barang\_yang\_dibeli |
| 002 | Mina | Majene | Pulpen |
| 003 | Nana | Makassar | Pulpen |
| 004 | Nina | Makassar | Buku |
| 005 | Anto | Mamuju | Buku |

Contoh Penggunaan NOT :

SELECT \* FROM Pelanggan  
WHERE NOT Alamat='Makassar';

Maka akan muncul tabel berikut :

|  |  |  |  |
| --- | --- | --- | --- |
| id | Namapelanggan | Alamat | Barang\_yang\_dibeli |
| 001 | Aco | Mamuju | Kertas HVS |
| 002 | Mina | Majene | Pulpen |
| 005 | Anto | Mamuju | Buku |
| 006 | Riri | Palu | Kertas HVS |

Kombinasi AND, OR, NOT

Contoh tabel pelanggan yang telah dibuat :

|  |  |  |  |
| --- | --- | --- | --- |
| id | Namapelanggan | Alamat | Barang\_yang\_dibeli |
| 001 | Aco | Mamuju | Kertas HVS |
| 002 | Mina | Majene | Pulpen |
| 003 | Nana | Makassar | Pulpen |
| 004 | Nina | Makassar | Buku |
| 005 | Anto | Mamuju | Buku |
| 006 | Riri | Makassar | Kertas HVS |

Menggunakan AND dan OR

SELECT \* FROM Pelanggan  
WHERE Alamat='Makassar' AND (Barang\_yang\_debeli='Pelpen' OR Barang\_yang\_debeli ='Buku');

Maka akan muncul tabel berikut :

|  |  |  |  |
| --- | --- | --- | --- |
| id | Namapelanggan | Alamat | Barang\_yang\_dibeli |
| 003 | Nana | Makassar | Pulpen |
| 004 | Nina | Makassar | Buku |

Menggunakan AND dan NOT

SELECT \* FROM Pelanggan  
WHERE NOT Alamat='Makassar' AND NOT Alamat='Mejene'  ;

Maka akan muncul table berikut :

|  |  |  |  |
| --- | --- | --- | --- |
| id | Namapelanggan | Alamat | Barang\_yang\_dibeli |
| 001 | Aco | Mamuju | Kertas HVS |
| 005 | Anto | Mamuju | Buku |

# SQL ORDER BY

ORDER BY digunakan untuk mengurutkan hasil-set dalam urutan ascending/menaik atau descending/menurun. Kata kunci ORDER BY mengurutkan catatan dalam urutan menaik secara default atau bawaan. Untuk mengurutkan catatan dalam urutan descending/menurun, gunakan kata kunci DESC. Syntax dari perintah ORDER BY :

SELECT column1, column2, ...  
FROM table\_name  
ORDER BY column1, column2, ... ASC|DESC;

Contoh :

Contoh tabel “pelanggan” yang telah dibuat :

|  |  |  |  |
| --- | --- | --- | --- |
| id | Namapelanggan | Alamat | Barang\_yang\_dibeli |
| 001 | Aco | Mamuju | Kertas HVS |
| 002 | Mina | Majene | Pulpen |
| 003 | Nana | Makassar | Pulpen |
| 004 | Nina | Makassar | Buku |
| 005 | Anto | Mamuju | Buku |
| 006 | Riri | Makassar | Kertas HVS |

Menggunakan ORDER BY

berikut synta yang memilih semua pelanggan dari tabel "Pelanggan", diurutkan berdasarkan kolom "Alamat":

SELECT \* FROM Pelanggan  
ORDER BY Alamat;

Maka akan muncul tabel berikut :

|  |  |  |  |
| --- | --- | --- | --- |
| id | Namapelanggan | Alamat | Barang\_yang\_dibeli |
| 002 | Mina | Majene | Pulpen |
| 003 | Nana | Makassar | Pulpen |
| 004 | Nina | Makassar | Buku |
| 006 | Riri | Makassar | Kertas HVS |
| 001 | Aco | Mamuju | Kertas HVS |
| 005 | Anto | Mamuju | Buku |

Menggunakan ORDER BY DESC

berikut syntax untuk memilih semua pelanggan dari tabel "Pelanggan", diurutkan descending atau menurun menurut kolom "Alamat" :

SELECT \* FROM Pelanggan  
ORDER BY Alamat DESC;

Maka akan muncul tabel berikut :

|  |  |  |  |
| --- | --- | --- | --- |
| id | Namapelanggan | Alamat | Barang\_yang\_dibeli |
| 001 | Aco | Mamuju | Kertas HVS |
| 005 | Anto | Mamuju | Buku |
| 003 | Nana | Makassar | Pulpen |
| 004 | Nina | Makassar | Buku |
| 006 | Riri | Makassar | Kertas HVS |
| 002 | Mina | Majene | Pulpen |

# SQL INSERT INTO

INSERT INTO digunakan untuk menyisipkan record atau kumpulan data baru ke dalam tabel. Syntax untuk INSERT INTO ada dua cara, antara lain :

* + 1. Tentukan nama kolom dan nilai yang akan disisipkan:

INSERT INTO table\_name (column1, column2, column3, ...)  
VALUES (value1, value2, value3, ...);

* + 1. Jika Anda menambahkan nilai untuk semua kolom tabel, Anda tidak perlu menentukan nama kolom dalam kueri SQL. Namun, pastikan urutan nilai dalam urutan yang sama dengan kolom dalam tabel. Di sini, sintaks INSERT INTO adalah sebagai berikut:

INSERT INTO table\_name  
VALUES (value1, value2, value3, ...);

Contoh :

|  |  |  |  |
| --- | --- | --- | --- |
| id | Namapelanggan | Alamat | Barang\_yang\_dibeli |
| 001 | Aco | Mamuju | Kertas HVS |
| 002 | Mina | Majene | Pulpen |
| 003 | Nana | Makassar | Pulpen |

berikut syntaxnya untuk menyisipkan catatan baru di tabel "Pelanggan":

INSERT INTO Pelanggan (id, Namapelanggan, Alamat, Barang\_yang\_dibeli)  
VALUES ('004', 'Nina', 'Makassar', 'Buku');

Maka akan muncul tabel seperti berikut :

|  |  |  |  |
| --- | --- | --- | --- |
| id | Namapelanggan | Alamat | Barang\_yang\_dibeli |
| 001 | Aco | Mamuju | Kertas HVS |
| 002 | Mina | Majene | Pulpen |
| 003 | Nana | Makassar | Pulpen |
| 004 | Nina | Makassar | Buku |

Syntax untuk Menyisipkan Data Hanya di Kolom Tertentu :

INSERT INTO Pelanggan (id, Namapelanggan, Alamat)  
VALUES ('004', 'Nina', 'Makassar');

Tabel akan muncul seperti berikut :

|  |  |  |  |
| --- | --- | --- | --- |
| id | Namapelanggan | Alamat | Barang\_yang\_dibeli |
| 001 | Aco | Mamuju | Kertas HVS |
| 002 | Mina | Majene | Pulpen |
| 003 | Nana | Makassar | Pulpen |
| 004 | Nina | Makassar | null |

# SQL NULL Values

Bidang dengan nilai NULL adalah bidang tanpa nilai. Jika bidang dalam tabel adalah opsional, dimungkinkan untuk menyisipkan catatan baru atau memperbarui catatan tanpa menambahkan nilai ke bidang ini. Kemudian, bidang tersebut akan disimpan dengan nilai NULL. Nilai NULL berbeda dari nilai nol atau bidang yang berisi spasi. Bidang dengan nilai NULL adalah bidang yang dibiarkan kosong selama pembuatan Record.

Kita tidak bisa menguji nilai NULL dengan operator perbandingan, seperti =, <, atau <>. Kita harus menggunakan operator IS NULL dan IS NOT NULL seperti syntax berikut :

SYNTAX IS NULL

SELECT column\_namesFROM table\_name  
WHERE column\_name IS NULL;

SYNTAX NOT NULL

SELECT column\_namesFROM table\_name  
WHERE column\_name IS NOT NULL;

Contoh tabel “pelanggan” yang dibuat :

|  |  |  |  |
| --- | --- | --- | --- |
| id | Namapelanggan | Alamat | Barang\_yang\_dibeli |
| 001 | Aco | Mamuju | Kertas HVS |
| 002 | Mina | Majene | Pulpen |
| 003 | Nana | Makassar | Pulpen |
| 004 | Nina | Makassar | Buku |
| 005 | Anto | Mamuju | Buku |
| 006 | Riri | Makassar | Kertas HVS |

Menggunakan IS NULL Operator

Operator IS NULL digunakan untuk menguji nilai kosong (nilai NULL). Berikut cara mencantumkan semua pelanggan dengan nilai NULL di bidang "Alamat":

SELECT id, Namapelanggan, Alamat,   
FROM Pelanggan  
WHERE Alamat IS NULL;

Maka akan muncul tabel sebagai berikut :

|  |  |  |
| --- | --- | --- |
| id | Namapelanggan | Alamat |
| 001 | Aco | NULL |
| 002 | Mina | NULL |
| 003 | Nana | NULL |
| 004 | Nina | NULL |
| 005 | Anto | NULL |
| 006 | Riri | NULL |

Operator IS NOT NULL

Operator IS NOT NULL digunakan untuk menguji nilai yang tidak kosong (nilai NOT NULL). berikut bagaimana mencantumkan semua pelanggan dengan nilai di bidang "Alamat":

SELECT id, Namapelanggan, Alamat,   
FROM Pelanggan  
WHERE Alamat IS NOTNULL;

|  |  |  |
| --- | --- | --- |
| id | Namapelanggan | Alamat |
| 001 | Aco | Mamuju |
| 002 | Mina | Majene |
| 003 | Nana | Makassar |
| 004 | Nina | Makassar |
| 005 | Anto | Mamuju |
| 006 | Riri | Makassar |

# SQL UPDATE

UPDATE digunakan untuk mengubah catatan yang ada dalam tabel. Syntaxnya yaitu :

UPDATE table\_name  
SET column1 = value1, column2 = value2, ...  
WHERE condition;

Hati-hati saat memperbarui catatan dalam tabel! Perhatikan klausa WHERE dalam pernyataan UPDATE. Klausa WHERE menentukan record mana yang harus diperbarui. Jika Anda menghilangkan klausa WHERE, semua catatan dalam tabel akan diperbarui!

Contoh :

Contoh tabel “pelannggan” yang telah dibuat

|  |  |  |  |
| --- | --- | --- | --- |
| id | Namapelanggan | Alamat | Barang\_yang\_dibeli |
| 001 | Aco | Mamuju | Kertas HVS |
| 002 | Mina | Majene | Pulpen |
| 003 | Nana | Makassar | Pulpen |
| 004 | Nina | Makassar | Buku |
| 005 | Anto | Mamuju | Buku |
| 006 | Riri | Makassar | Kertas HVS |

Menggunakan UPDATE dengan Beberapa record :

Ini adalah klausa WHERE yang menentukan berapa banyak catatan yang akan diperbarui. Pernyataan SQL berikut akan memperbarui barang menjadi "kertas HVS" untuk semua catatan di mana Alamt adalah "Mamuju":

UPDATE Pelanggan  
SET Barang\_yang\_dibeli =Kertas HVS'  
WHERE Alamat ='Makassar';

Maka akan muncul tabel seperti beriku :

|  |  |  |  |
| --- | --- | --- | --- |
| id | Namapelanggan | Alamat | Barang\_yang\_dibeli |
| 001 | Aco | Mamuju | Kertas HVS |
| 002 | Mina | Majene | Pulpen |
| 003 | Nana | Makassar | Kertas HVS |
| 004 | Nina | Makassar | Kertas HVS |
| 005 | Anto | Mamuju | Buku |
| 006 | Riri | Makassar | Kertas HVS |

dalam menggunakan UPDATE Berhati-hatilah saat memperbarui Record. Jika Anda menghilangkan klausa WHERE, SEMUA Record akan diperbarui. Contoh :

UPDATE Pelanggan  
SET Barang\_yang\_dibeli='Kertas HVS';

Maka akan jadi seperti berikut :

|  |  |  |  |
| --- | --- | --- | --- |
| id | Namapelanggan | Alamat | Barang\_yang\_dibeli |
| 001 | Aco | Mamuju | Kertas HVS |
| 002 | Mina | Majene | Kertas HVS |
| 003 | Nana | Makassar | Kertas HVS |
| 004 | Nina | Makassar | Kertas HVS |
| 005 | Anto | Mamuju | Kertas HVS |
| 006 | Riri | Makassar | Kertas HVS |

# SQL DELETE

DELETE digunakan untuk menghapus record yang ada dalam tabel dengan Syntax :

DELETE FROM table\_name WHERE condition;

Hati-hati saat menghapus record dalam tabel! Perhatikan klausa WHERE dalam pernyataan DELETE. Klausa WHERE menentukan record mana yang harus dihapus. Jika Anda menghilangkan klausa WHERE, semua record dalam tabel akan dihapus!

Contoh :

Contoh tabel “pelannggan” yang telah dibuat

|  |  |  |  |
| --- | --- | --- | --- |
| id | Namapelanggan | Alamat | Barang\_yang\_dibeli |
| 001 | Aco | Mamuju | Kertas HVS |
| 002 | Mina | Majene | Pulpen |
| 003 | Nana | Makassar | Pulpen |
| 004 | Nina | Makassar | Buku |
| 005 | Anto | Mamuju | Buku |
| 006 | Riri | Makassar | Kertas HVS |

Menggunakan DELETE

Menghapus Namapelanggan “Mina” dari tabel "Pelanggan":

DELETE FROM Pelanggan WHERE Namapelanggan='Mina';

Maka tabelnya akan seperti berikut :

|  |  |  |  |
| --- | --- | --- | --- |
| id | Namapelanggan | Alamat | Barang\_yang\_dibeli |
| 001 | Aco | Mamuju | Kertas HVS |
| 003 | Nana | Makassar | Pulpen |
| 004 | Nina | Makassar | Buku |
| 005 | Anto | Mamuju | Buku |
| 006 | Riri | Makassar | Kertas HVS |

## Delete Semua Record

Dimungkinkan untuk menghapus semua baris dalam tabel tanpa menghapus tabel. Ini berarti bahwa struktur tabel, atribut, dan indeks akan tetap utuh:

## DELETE FROM table\_name;

Berikut syntax untuk menghapus semua baris dalam tabel "Pelanggan", tanpa menghapus tabel:

DELETE FROM Pelanggan;

# SQL TOP, LIMIT, FETCH FIRST Atau ROWNUM

SELECT TOP digunakan untuk menentukan jumlah record yang akan dikembalikan. Klausa SELECT TOP berguna pada tabel besar dengan ribuan record. Mengembalikan sejumlah besar Record dapat memengaruhi kinerja. Tidak semua sistem database mendukung klausa SELECT TOP. MySQL mendukung klausa LIMIT untuk memilih sejumlah record yang terbatas, sedangkan Oracle menggunakan FETCH FIRST n ROWS ONLY dan ROWNUM. Syntax antara lain :

SQL Server / MS Access Syntax:

SELECT TOP *number*|*percent* *column\_name(s)*  
FROM *table\_name*WHERE *condition*;

MySQL Syntax:

SELECT *column\_name(s)*  
FROM *table\_name*WHERE *condition*  
LIMIT *number*;

Oracle 12 Syntax:

SELECT *column\_name(s)*  
FROM *table\_name*ORDER BY *column\_name(s)*  
FETCH FIRST *number* ROWS ONLY;

Older Oracle Syntax:

SELECT *column\_name(s)*  
FROM *table\_name*  
WHERE ROWNUM <= *number*;

**Older Oracle Syntax (with ORDER BY):**

SELECT \*  
FROM (SELECT *column\_name(s)*FROM *table\_name* ORDER BY *column\_name(s)*)  
WHERE ROWNUM <= *number*;

Contoh untuk MySQL:

Contoh tabel “pelanggan” yang dibuat :

|  |  |  |  |
| --- | --- | --- | --- |
| id | Namapelanggan | Alamat | Barang\_yang\_dibeli |
| 001 | Aco | Mamuju | Kertas HVS |
| 002 | Mina | Majene | Pulpen |
| 003 | Nana | Makassar | Pulpen |
| 004 | Nina | Makassar | Buku |
| 005 | Anto | Mamuju | Buku |
| 006 | Riri | Makassar | Kertas HVS |

Menggunakan LIMIT untuk MySQL

Berikut memilih tiga Record pertama dari tabel "Pelanggan"

SELECT \* FROM Pelanggan  
LIMIT 3;

Makan akan muncul tabel seperti berikut :

|  |  |  |  |
| --- | --- | --- | --- |
| id | Namapelanggan | Alamat | Barang\_yang\_dibeli |
| 001 | Aco | Mamuju | Kertas HVS |
| 002 | Mina | Majene | Pulpen |
| 003 | Nana | Makassar | Pulpen |

Menggunakan LIMIT dan WHERE untuk MySQL

Berikut memilih dua Record pertama dari tabel "Pelanggan", di mana Alamatnya adalah "Makassar"

SELECT \* FROM Pelanggan  
WHERE Alamat='Makassar'  
LIMIT 2;

Akan muncul Tabel akan seperti berikut :

|  |  |  |  |
| --- | --- | --- | --- |
| id | Namapelanggan | Alamat | Barang\_yang\_dibeli |
| 003 | Nana | Makassar | Pulpen |
| 004 | Nina | Makassar | Buku |

# SQL fungsi MIN() and MAX()

Fungsi MIN() mengembalikan nilai terkecil dari kolom yang dipilih.Fungsi MAX() mengembalikan nilai terbesar dari kolom yang dipilih. Syntax sebagai berikut :

### MIN() Syntax

SELECT MIN(column\_name)  
FROM table\_name;

### MAX() Syntax

SELECT MAX(column\_name)  
FROM table\_name;  
Contoh :

Contoh tabel “pelanggan” yang dibuat :

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| id | Namapelanggan | Alamat | Barang\_yang\_dibeli | Dibayar |
| 001 | Aco | Mamuju | Kertas HVS | 120000 |
| 002 | Mina | Majene | Pulpen | 100000 |
| 003 | Nana | Makassar | Pulpen | 20000 |
| 004 | Nina | Makassar | Buku | 30000 |
| 005 | Anto | Mamuju | Buku | 40000 |
| 006 | Riri | Makassar | Kertas HVS | 60000 |

Menggunakan MIN

SELECT MIN(Dibayar) AS BayaranTerendah  
FROM pelanggan;  
Maka akan muncul tabel seperti berikut

|  |
| --- |
| BayaranTerendah |
| 20000 |

Menggunakan MIN

SELECT MAX(Dibayar) AS BayaranTertinggi  
FROM pelanggan;  
Maka akan muncul tabel seperti berikut

|  |
| --- |
| BayaranTertinggi |
| 120000 |

# SQL fungsi COUNT(), AVG() and SUM()

COUNT() fungsi untuk menghitung jumlah baris atau record pada tabel. . syntax :

SELECT COUNT(column\_name)  
FROM table\_name  
WHERE condition;

Fungsi AVG() menghitung nilai rata-rata kolom numerik. syntx :

SELECT AVG(column\_name)  
FROM table\_name  
WHERE condition;

SUM() fungsi mengembalikan jumlah total kolom numerik. syntax :

SELECT SUM(column\_name)  
FROM table\_name  
WHERE condition;

Contoh :

Contoh tabel “pelanggan” yang dibuat :

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| id | Namapelanggan | Alamat | Barang\_yang\_dibeli | Dibayar |
| 001 | Aco | Mamuju | Kertas HVS | 120000 |
| 002 | Mina | Majene | Pulpen | 100000 |
| 003 | Nana | Makassar | Pulpen | 20000 |
| 004 | Nina | Makassar | Buku | 30000 |
| 005 | Anto | Mamuju | Buku | 40000 |
| 006 | Riri | Makassar | Kertas HVS | 60000 |

Menggunakan fungs COUNT()

SELECT COUNT(id)  
FROM pelanggan;  
akan muncul tabel seperti berikut :

|  |
| --- |
| COUNT(id) |
| 6 |

Menggunakan fungs AVG()

SELECT AVG(Dibayar)  
FROM pelanggan;  
Akan muncul tabel seperti berikut :

|  |
| --- |
| AVG(Dibayar) |
| 61666.67 |

Menggunakan fungs SUM()

SELECT SUM(Dibayar)  
FROM pelanggan;  
Akan muncul tabel seperti berikut :

|  |
| --- |
| SUM(Dibayar) |
| 370000 |

# SQL LIKE

Operator LIKE digunakan dalam klausa WHERE untuk mencari pola tertentu dalam kolom. Ada dua wildcard yang sering digunakan bersama dengan operator LIKE:

* Tanda persen (%) mewakili nol, satu, atau beberapa karakter
* Tanda garis bawah (\_) mewakili satu karakter tunggal

MS Access menggunakan tanda bintang (\*) sebagai ganti tanda persen (%), dan tanda tanya (?) sebagai ganti garis bawah (\_). Tanda persen dan garis bawah juga dapat digunakan dalam kombinasi!

Syntax LIKE yaitu :

SELECT column1, column2, ...  
FROM table\_name  
WHERE columnN LIKE pattern;

Bisa juga menggabungkan sejumlah kondisi menggunakan operator AND atau OR.

Contoh tabel “pelanggan” yang dibuat :

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| id | Namapelanggan | Alamat | Barang\_yang\_dibeli | Dibayar |
| 001 | Aco | Mamuju | Kertas HVS | 120000 |
| 002 | Mina | Majene | Pulpen | 100000 |
| 003 | Nana | Makassar | Pulpen | 20000 |
| 004 | Nina | Makassar | Buku | 30000 |
| 005 | Anto | Mamuju | Buku | 40000 |
| 006 | Riri | Makassar | Kertas HVS | 60000 |

Menggunakan LIKE

Berikut memilih semua pelanggan dengan Namapelanggan dimulai dengan "A":

SELECT \* FROM pelanggan  
WHERE Namapelanggan LIKE 'A%';

Maka tabel yang muncul seperti berikut :

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| id | Namapelanggan | Alamat | Barang\_yang\_dibeli | Dibayar |
| 001 | Aco | Mamuju | Kertas HVS | 120000 |
| 005 | Anto | Mamuju | Buku | 40000 |

Berikut memilih semua pelanggan dengan Namapelanggan diakhiri dengan "A":

SELECT \* FROM pelanggan  
WHERE Namapelanggan LIKE '%A';

Maka tabel yang muncul seperti berikut :

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| id | Namapelanggan | Alamat | Barang\_yang\_dibeli | Dibayar |
| 002 | Mina | Majene | Pulpen | 100000 |
| 003 | Nana | Makassar | Pulpen | 20000 |

Berikut memilih semua pelanggan dengan Namapelanggan ditenganya ada "in":

SELECT \* FROM pelanggan  
WHERE Namapelanggan LIKE '%in%';

Maka tabel yang muncul seperti berikut :

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| id | Namapelanggan | Alamat | Barang\_yang\_dibeli | Dibayar |
| 002 | Mina | Majene | Pulpen | 100000 |
| 004 | Nina | Makassar | Buku | 30000 |

Berikut memilih semua pelanggan dengan Namapelanggan mempunyai "i" di posisi ke dua:

SELECT \* FROM pelanggan  
WHERE Namapelanggan LIKE '\_i%';

Maka tabel yang muncul seoerti berikut :

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| id | Namapelanggan | Alamat | Barang\_yang\_dibeli | Dibayar |
| 002 | Mina | Majene | Pulpen | 100000 |
| 004 | Nina | Makassar | Buku | 30000 |
| 006 | Riri | Makassar | Kertas HVS | 60000 |

Berikut memilih semua pelanggan dengan Namapelanggan yang dimulai dengan "a" dan memiliki 3 karakter:

SELECT \* FROM pelanggan  
WHERE Namapelanggan LIKE 'a\_%';

Maka tabel yang muncul seoerti berikut :

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| id | Namapelanggan | Alamat | Barang\_yang\_dibeli | Dibayar |
| 001 | Aco | Mamuju | Kertas HVS | 120000 |

Berikut memilih semua pelanggan dengan Namapelanggan yang dimulai dengan "a" dan dikahiri dengan ‘o’:

SELECT \* FROM pelanggan  
WHERE Namapelanggan LIKE 'a%o';

Maka tabel yang muncul seoerti berikut :

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| id | Namapelanggan | Alamat | Barang\_yang\_dibeli | Dibayar |
| 001 | Aco | Mamuju | Kertas HVS | 120000 |

Berikut memilih semua pelanggan dengan Namapelanggan yang tidak dimulai dengan "a":

SELECT \* FROM pelanggan  
WHERE Namapelanggan NOT LIKE 'a';

Maka tabel yang muncul seoerti berikut :

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| id | Namapelanggan | Alamat | Barang\_yang\_dibeli | Dibayar |
| 002 | Mina | Majene | Pulpen | 100000 |
| 003 | Nana | Makassar | Pulpen | 20000 |
| 004 | Nina | Makassar | Buku | 30000 |
| 006 | Riri | Makassar | Kertas HVS | 60000 |

# SQL Wildcards

Wildcard adalah operator yang digunakan untuk mensubstitusikan berbagai karakter lain dalam suatu string. Wildcard SQL digunakan juga untuk mencari data dengan menggunakan tabel pada SQL. Karakter wildcard digunakan dengan operator LIKE. Operator LIKE digunakan dalam klausa WHERE untuk mencari pola tertentu dalam kolom.

Wildcard Characters in SQL Server

|  |  |  |
| --- | --- | --- |
| simbol | deskripsi | contoh |
| % | Mewakili nol atau lebih karakter | Bi% menemukan bintang, biru, bimo dan bina |
| \_ | Mewakili satu karakter | h\_t Menemukan hot, hat, dan hit |
| [] | Mewakili karakter tunggal apa pun dalam tanda kurung | h[oa]t Menemukan hot and hat, but not hit |
| ^ | Mewakili karakter apa pun yang tidak ada dalam tanda kurung | h[^oa]t menemukan hit, but not hot dan hat |
| - | Mewakili karakter tunggal apa pun dalam rentang yang ditentukan | c[a-b]t Menemukan cat bina cbt |

Contoh :

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| id | Namapelanggan | Alamat | Barang\_yang\_dibeli | Dibayar |
| 001 | Aco | Mamuju | Kertas HVS | 120000 |
| 002 | Mina | Majene | Pulpen | 100000 |
| 003 | Nana | Makassar | Pulpen | 20000 |
| 004 | Rina | Makassar | Buku | 30000 |
| 005 | Antonion | Mamuju | Buku | 40000 |
| 006 | Riri | Makassar | Kertas HVS | 60000 |

Menggunakan % wildcard

Memilih semua pelanggan dengan Namapelanggan yang dimulai dengan "Ri":

SELECT \* FROM pelanggan  
WHERE Namapelanggan LIKE 'Ri%';

Maka tabel akan seperti ini :

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| id | Namapelanggan | Alamat | Barang\_yang\_dibeli | Dibayar |
| 004 | Rina | Makassar | Buku | 30000 |
| 006 | Riri | Makassar | Kertas HVS | 60000 |

Memilih semua pelanggan dengan Namapelanggan yang berisi pola "in":

SELECT \* FROM pelanggan  
WHERE Namapelanggan LIKE '%in%';

Maka tabel akan seperti ini :

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| id | Namapelanggan | Alamat | Barang\_yang\_dibeli | Dibayar |
| 002 | Mina | Majene | Pulpen | 100000 |
| 004 | Rina | Makassar | Buku | 30000 |

Menggunakan \_ WILDCARD

Memilih semua pelanggan dengan Namapelanggan yang dimulai dengan karakter apa pun, diikuti oleh "na":

SELECT \* FROM pelanggan  
WHERE Namapelanggan LIKE '\_na';

Maka tabel akan seperti ini :

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| id | Namapelanggan | Alamat | Barang\_yang\_dibeli | Dibayar |
| 002 | Mina | Majene | Pulpen | 100000 |
| 003 | Nana | Makassar | Pulpen | 20000 |
| 004 | Rina | Makassar | Buku | 30000 |

memilih semua pelanggan dengan Namapelanggan yang dimulai dengan "A", diikuti oleh karakter apa pun, diikuti oleh "t", diikuti oleh karakter apa pun, diikuti oleh "on":

SELECT \* FROM pelanggan  
WHERE Namapelanggan LIKE 'A\_t\_on';

Maka tabel akan seperti ini :

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| id | Namapelanggan | Alamat | Barang\_yang\_dibeli | Dibayar |
| 005 | Antonion | Mamuju | Buku | 40000 |

Menggunakan Wildcard [charlist]

Memilih semua pelanggan dengan Namapelanggan yang dimulai dengan "A", "M", atau "R":

SELECT \* FROM pelanggan  
WHERE Namapelanggan LIKE '[AMR]%';

Maka tabel akan seperti ini :

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| id | Namapelanggan | Alamat | Barang\_yang\_dibeli | Dibayar |
| 001 | Aco | Mamuju | Kertas HVS | 120000 |
| 002 | Mina | Majene | Pulpen | 100000 |
| 004 | Rina | Makassar | Buku | 30000 |
| 005 | Antonion | Mamuju | Buku | 40000 |
| 006 | Riri | Makassar | Kertas HVS | 60000 |

Memilih semua pelanggan dengan Namapelanggan yang dimulai dengan "a", "b", atau "c":

SELECT \* FROM pelanggan  
WHERE Namapelanggan LIKE '[a-c]%';

Maka tabel akan seperti ini :

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| id | Namapelanggan | Alamat | Barang\_yang\_dibeli | Dibayar |
| 001 | Aco | Mamuju | Kertas HVS | 120000 |
| 005 | Antonion | Mamuju | Buku | 40000 |

Menggunakan [!charlist] WILDCARD

Memilih semua pelanggan dengan Namapelanggan yang tidak dimulai dengan "A", "M", atau "R":

SELECT \* FROM pelanggan  
WHERE Namapelanggan LIKE '[!AMR]%';

Atau

SELECT \* FROM pelanggan  
WHERE Namapelanggan NOT LIKE '[AMR]%';

Maka tabel akan seperti ini :

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| id | Namapelanggan | Alamat | Barang\_yang\_dibeli | Dibayar |
| 003 | Nana | Makassar | Pulpen | 20000 |
| 005 | Antonion | Mamuju | Buku | 40000 |

# SQL IN

Operator IN memungkinkan Anda untuk menentukan beberapa nilai dalam klausa WHERE. Operator IN adalah singkatan untuk beberapa kondisi OR. Syntax :

SELECT column\_name(s)  
FROM table\_name  
WHERE column\_name IN (value1, value2, ...);

Atau

SELECT column\_name(s)  
FROM table\_name  
WHERE column\_name IN (*SELECT* STATEMENT);

Contoh :

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| id | Namapelanggan | Alamat | Barang\_yang\_dibeli | Dibayar |
| 001 | Aco | Mamuju | Kertas HVS | 120000 |
| 002 | Mina | Majene | Pulpen | 100000 |
| 003 | Nana | Makassar | Pulpen | 20000 |
| 004 | Rina | Makassar | Buku | 30000 |
| 005 | Antonion | Palu | Buku | 40000 |
| 006 | Riri | Makassar | Kertas HVS | 60000 |

Menggunakan operator IN

Memilih semua pelanggan yang beralamat di "Mamuju" “palu” atau "Majene":

SELECT \* FROM pelanggan  
WHERE Alamat IN ('Mamuju', 'Palu', 'Majene');

Maka tabel yang muncul akan seperti ini :

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| id | Namapelanggan | Alamat | Barang\_yang\_dibeli | Dibayar |
| 001 | Aco | Mamuju | Kertas HVS | 120000 |
| 002 | Mina | Majene | Pulpen | 100000 |
| 005 | Antonion | Palu | Buku | 40000 |

Menggunakan operator NOT IN

Memilih semua pelanggan yang Tidak beralamat di "Mamuju" “palu” atau "Majene":

SELECT \* FROM pelanggan  
WHERE Alamat NOT IN ('Mamuju', 'Palu', 'Majene');

Maka tabel yang muncul akan seperti ini :

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| id | Namapelanggan | Alamat | Barang\_yang\_dibeli | Dibayar |
| 003 | Nana | Makassar | Pulpen | 20000 |
| 004 | Rina | Makassar | Buku | 30000 |
| 006 | Riri | Makassar | Kertas HVS | 60000 |

# SQL BETWEEN

Operator BETWEEN memilih nilai dalam rentang tertentu. Nilai dapat berupa angka, teks, atau tanggal. Operator BETWEEN bersifat inklusif: nilai awal dan akhir disertakan. Syntax :

SELECT column\_name(s)  
FROM table\_name  
WHERE column\_name BETWEEN value1 AND value2;

Contoh :

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| id | Namapelanggan | Alamat | Barang\_yang\_dibeli | Dibayar |
| 001 | Aco | Mamuju | Kertas HVS | 120000 |
| 002 | Mina | Majene | Pulpen | 100000 |
| 003 | Nana | Makassar | Pulpen | 20000 |
| 004 | Rina | Makassar | Buku | 30000 |
| 005 | Antonion | Palu | Buku | 40000 |
| 006 | Riri | Makassar | Kertas HVS | 60000 |

Menggunakan BETWEEN

SELECT \* FROM Pelanggan  
WHERE Dibayar BETWEEN 10000 AND 50000;

Maka tabel yang muncul seperti berikut :

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| id | Namapelanggan | Alamat | Barang\_yang\_dibeli | Dibayar |
| 003 | Nana | Makassar | Pulpen | 20000 |
| 004 | Rina | Makassar | Buku | 30000 |
| 005 | Antonion | Palu | Buku | 40000 |

Menggunakan NOT BETWEEN

Untuk menampilkan produk di luar jangkauan contoh sebelumnya, gunakan NOT BETWEEN:

SELECT \* FROM Pelanggan  
WHERE Dibayar NOT BETWEEN 10000 AND 50000;

Tabel yang muncul :

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| id | Namapelanggan | Alamat | Barang\_yang\_dibeli | Dibayar |
| 001 | Aco | Mamuju | Kertas HVS | 120000 |
| 002 | Mina | Majene | Pulpen | 100000 |
| 006 | Riri | Makassar | Kertas HVS | 60000 |

Menggunakan BETWEEN dengan IN

SELECT \* FROM Pelanggan  
WHERE Dibayar BETWEEN 10000 AND 50000  
AND id NOT IN (001,002,003);

Tabel yang muncul :

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| id | Namapelanggan | Alamat | Barang\_yang\_dibeli | Dibayar |
| 004 | Rina | Makassar | Buku | 30000 |
| 005 | Antonion | Palu | Buku | 40000 |

# SQL Aliases

Alias ​​​​SQL digunakan untuk memberi tabel, atau kolom dalam tabel, nama sementara. Alias ​​​​sering digunakan untuk membuat nama kolom lebih mudah dibaca. Alias ​​​​hanya ada selama kueri itu. Alias ​​dibuat dengan kata kunci AS.

Syntax Alias kolom

SELECT column\_name AS alias\_name  
FROM table\_name;

Syntax Alias Tabel

SELECT column\_name(s)  
FROM table\_name AS alias\_name;

Contoh :

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| id | Namapelanggan | Alamat | Barang\_yang\_dibeli | Dibayar |
| 001 | Aco | Mamuju | Kertas HVS | 120000 |
| 002 | Mina | Majene | Pulpen | 100000 |
| 003 | Nana | Makassar | Pulpen | 20000 |
| 004 | Rina | Makassar | Buku | 30000 |
| 005 | Antonion | Palu | Buku | 40000 |
| 006 | Riri | Makassar | Kertas HVS | 60000 |

Penggunaan Alias untuk Kolom

Pernyataan SQL berikut membuat dua alias, satu untuk kolom Namapelanggan dan satu lagi untuk kolom Barang\_yang\_dibeli :

SELECT Namapelanggan AS Nama, Barang\_yang\_dibeli AS Barang  
FROM Customers;

Maka tabelnya menjadi :

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| id | Nama | Alamat | Barang | Dibayar |
| 001 | Aco | Mamuju | Kertas HVS | 120000 |
| 002 | Mina | Majene | Pulpen | 100000 |
| 003 | Nana | Makassar | Pulpen | 20000 |
| 004 | Rina | Makassar | Buku | 30000 |
| 005 | Antonion | Palu | Buku | 40000 |
| 006 | Riri | Makassar | Kertas HVS | 60000 |

membuat dua alias, satu untuk kolom NamaPelanggan dan satu lagi untuk kolom NamaKontak. Ini membutuhkan tanda kutip ganda atau tanda kurung siku jika nama alias berisi spasi:

SELECT Namapelanggan AS Nama, Barang\_yang\_dibeli AS [Barang yang dibeli]  
FROM Customers;

Maka tabelnya menjadi :

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| id | Nama | Alamat | Barang yang dibeli | Dibayar |
| 001 | Aco | Mamuju | Kertas HVS | 120000 |
| 002 | Mina | Majene | Pulpen | 100000 |
| 003 | Nana | Makassar | Pulpen | 20000 |
| 004 | Rina | Makassar | Buku | 30000 |
| 005 | Antonion | Palu | Buku | 40000 |
| 006 | Riri | Makassar | Kertas HVS | 60000 |

## SQL JOIN

JOIN digunakan untuk menggabungkan baris dari dua atau lebih tabel, berdasarkan kolom terkait di antara mereka. Ada beberapa SQL JOIN antara lain

1. **SQL INNER JOIN**

INNER JOIN memilih record yang memiliki nilai yang sama atau cocok di kedua tabel. Mari kita lihat contoh pilihan dari tabel "Pesanan":

|  |  |  |
| --- | --- | --- |
| id | Barang\_yang\_dibeli | Dibayar |
| 001 | Kertas HVS | 120000 |
| 002 | Pulpen | 100000 |
| 003 | Pulpen | 20000 |
| 004 | Buku | 30000 |
| 005 | Buku | 40000 |
| 006 | Kertas HVS | 60000 |

Kemudian, lihat tabel “pelanggan”:

|  |  |  |
| --- | --- | --- |
| id | Namapelanggan | Alamat |
| 001 | Aco | Mamuju |
| 002 | Mina | Majene |
| 003 | Nana | Makassar |
| 004 | Rina | Makassar |
| 005 | Antonion | Palu |
| 006 | Riri | Makassar |

Perhatikan bahwa kolom "id" di tabel "Pesanan" merujuk ke "id" di tabel "Pelanggan". Hubungan antara dua tabel di atas adalah kolom "id". Kemudian, kita dapat membuat pernyataan SQL berikut (yang berisi INNER JOIN), yang memilih record yang memiliki nilai yang cocok di kedua tabel:

SELECT pelanggan.Namapelanggan,pesanan.Barang\_yang\_dibeli, Pesanan.Dibayar   
FROM pelanggan  
INNER JOIN Pesanan ON pelanggan.id=Pesanan.id;

Maka tabel akan seperti berikut :

|  |  |  |
| --- | --- | --- |
| Namapelanggan | Barang\_yang\_dibeli | Dibayar |
| Aco | Kertas HVS | 120000 |
| Mina | Pulpen | 100000 |
| Nana | Pulpen | 20000 |
| Rina | Buku | 30000 |
| Antonion | Buku | 40000 |
| Riri | Kertas HVS | 60000 |

Ada juga syntax untuk JOIN dengan 3 table yaitu :

tabel "Pesanan":

|  |  |  |  |
| --- | --- | --- | --- |
| id | Barang\_yang\_dibeli | Dibayar | Kd |
| 001 | Kertas HVS | 120000 | 101 |
| 002 | Pulpen | 100000 | 102 |
| 003 | Pulpen | 20000 | 101 |
| 004 | Buku | 30000 | 101 |
| 005 | Buku | 40000 | 103 |
| 006 | Kertas HVS | 60000 | 101 |

tabel “pelanggan”:

|  |  |  |  |
| --- | --- | --- | --- |
| id | Namapelanggan | Alamat | Kd |
| 001 | Aco | Mamuju | 101 |
| 002 | Mina | Majene | 102 |
| 003 | Nana | Makassar | 101 |
| 004 | Rina | Makassar | 101 |
| 005 | Antonion | Palu | 103 |
| 006 | Riri | Makassar | 101 |

tabel “Pengantar”:

|  |  |  |  |
| --- | --- | --- | --- |
| Id | Kd | Namapengantar | Jalur |
| 1 | 101 | Agi | Mamuju-Makassar |
| 2 | 102 | Nano | Mamujuj-Majene |
| 3 | 103 | Andi | Mamuju-Palu |

SELECT Pesanan.Barang\_yang\_dibeli, pelanggan.Namapelanggan, pengantar.Namapengantar   
FROM ((pesanan  
INNER JOIN Pelanggan ON pesanan.id = Pelanggan.id)  
INNER JOIN Pengantar ON Pesanan.kd = Pengan.Kd);

Maka tabelnya seperti berikut:

|  |  |  |
| --- | --- | --- |
| Barang\_yang\_dibeli | Namapelanggan | Namapengantar |
| Kertas HVS | Aco | Agi |
| Pulpen | Mina | nano |
| Pulpen | Nana | Agi |
| Buku | Rina | Agi |
| Buku | Antonion | Andi |
| Kertas HVS | Riri | Agi |

1. **SQL LEFT JOIN**

LEFT JOIN mengembalikan semua record dari tabel kiri (table1), dan record yang cocok dari tabel kanan (table2). Hasilnya adalah Null atau 0 catatan dari sisi kanan, jika tidak ada kecocokan. Syntaxnya yaitu :

SELECT column\_name(s)  
FROM table1  
LEFT JOIN table2ON table1.column\_name = table2.column\_name;

Dalam beberapa database LEFT JOIN disebut LEFT OUTER JOIN.

LEFT JOIN

Tabel 2

Tabel 1

Contoh :

Tabel "Pesanan":

|  |  |  |
| --- | --- | --- |
| id | Barang\_yang\_dibeli | Dibayar |
| 001 | Kertas HVS | 120000 |
| 002 | Pulpen | 100000 |
| 003 | Pulpen | 20000 |
| 004 | Buku | 30000 |

Tabel “pelanggan”:

|  |  |  |
| --- | --- | --- |
| id | Namapelanggan | Alamat |
| 001 | Aco | Mamuju |
| 002 | Mina | Majene |
| 003 | Nana | Makassar |
| 004 | Rina | Makassar |
| 005 | Antonion | Palu |
| 006 | Riri | Makassar |

Menggunkan LEFT JOIN

SELECT pelanggan.Namapelanggan, pesanan.Barang\_yang\_dibeli  
FROM Pelanggan  
LEFT JOIN Pesanan ON Pelanggan.id = Pesanan.id;

Maka tabel akan seperti berikut :

|  |  |
| --- | --- |
| Namapelanggan | Barang\_yang\_dibeli |
| Aco | Kertas HVS |
| Mina | Pulpen |
| Nana | Pulpen |
| Rina | Buku |
| Antonion | Null |
| Riri | Null |

1. **SQL RIGHT JOIN**

RIGHT JOIN mengembalikan semua record dari tabel kanan (table2), dan record yang cocok dari tabel kiri (table1). Hasilnya adalah Null atau 0 catatan dari sisi kiri, jika tidak ada kecocokan. Syntaxnya :

SELECT column\_name(s)  
FROM table1  
RIGHT JOIN table2ON table1.column\_name = table2.column\_name;

Dalam beberapa database RIGHT JOIN disebut RIGHT OUTER JOIN.

RIGHT JOIN

Tabel 2

Tabel 1

Contoh :

Tabel "Pesanan":

|  |  |  |
| --- | --- | --- |
| id | Barang\_yang\_dibeli | Dibayar |
|  | Kertas HVS | 120000 |
|  | Pulpen | 100000 |
|  | Pulpen | 20000 |
| 004 | Buku | 30000 |
| 005 | Buku | 40000 |
| 006 | KertasHVS | 60000 |

Tabel “pelanggan”:

|  |  |  |
| --- | --- | --- |
| id | Namapelanggan | Alamat |
| 001 | Aco | Mamuju |
| 002 | Mina | Majene |
| 003 | Nana | Makassar |
| 004 | Rina | Makassar |
| 005 | Antonion | Palu |
| 006 | Riri | Makassar |

Menggunakan RIGHT JOIN

SELECT pelanggan.Namapelanggan, pesanan.Barang\_yang\_dibeli  
FROM Pelanggan  
RIGHT JOIN PesananON Pesanan ON Pelanggan.id = Pesanan.id;

Tabelnya akan jadi :

|  |  |
| --- | --- |
| Namapelanggan | Barang\_yang\_dibeli |
| Null | Kertas HVS |
| Null | Pulpen |
| Null | Pulpen |
| Rina | Buku |
| Antonion | Buku |
| Riri | Kertas HVS |

1. **SQL FULL JOIN**

FULL OUTER JOIN mengembalikan semua record ketika ada kecocokan pada record tabel kiri (table1) atau kanan (table2). FULL OUTER JOIN dan FULL JOIN adalah sama. Syn taxnya

SELECT column\_name(s)  
FROM table1  
FULL OUTER JOIN table2ON table1.column\_name = table2.column\_nameWHERE condition;

Contoh :

Tabel "Pesanan":

|  |  |  |
| --- | --- | --- |
| id | Barang\_yang\_dibeli | Dibayar |
| 001 |  | 120000 |
| 002 |  | 100000 |
| 003 | Pulpen | 20000 |
| 004 | Buku | 30000 |
| 005 | Buku | 40000 |
| 006 | KertasHVS | 60000 |

Tabel “pelanggan”:

|  |  |  |
| --- | --- | --- |
| id | Namapelanggan | Alamat |
| 001 | Aco | Mamuju |
| 002 | Mina | Majene |
| 003 | Nana | Makassar |
| 004 | Rina | Makassar |
| 005 |  | Palu |
| 006 |  | Makassar |

Tabel yang akan muncul

|  |  |
| --- | --- |
| Namapelanggan | Barang\_yang\_dibeli |
| Aco | Null |
| Mina | Null |
| Nana | Pulpen |
| Rina | Buku |
| Null | Buku |
| Null | Kertas HVS |

FULL OUTER JOIN memperliatkan semua record yang cocok dari kedua tabel apakah tabel lainnya cocok atau tidak. Jadi, jika ada baris di "Pelanggan" yang tidak cocok di "Pesanan", atau jika ada baris di "Pesanan" yang tidak cocok di "Pelanggan", baris itu juga akan dicantumkan

1. **SQL SELF JOIN**

SELF JOIN adalah gabungan biasa, tetapi tabel bergabung dengan dirinya sendiri. Syntaxnya :

SELECT column\_name(s)  
FROM table1 T1, table1 T2  
WHERE condition;

T1 dan T2 adalah alias tabel yang berbeda untuk tabel yang sama.

1. **SQL UNION**

UNION digunakan untuk menggabungkan kumpulan hasil dari dua atau lebih pernyataan SELECT. Setiap pernyataan SELECT dalam UNION harus memiliki jumlah kolom yang sama Kolom juga harus memiliki tipe data yang serupa Kolom dalam setiap pernyataan SELECT juga harus dalam urutan yang sama. Syntaxnya :

SELECT column\_name(s) FROM table1  
UNION  
SELECT column\_name(s) FROM table2;

UNION hanya memilih nilai yang berbeda secara default. Untuk mengizinkan nilai duplikat, gunakan UNION ALL . syntaxnya yaitu :

SELECT column\_name(s) FROM table1  
UNION ALL  
SELECT column\_name(s) FROM table2;

Contoh :

Tabel “pelanggan1” :

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| id | Namapelanggan | Alamat | Barang\_yang\_dibeli | Dibayar |
| 001 | Aco | Mamuju | Kertas HVS | 120000 |
| 002 | Mina | Majene | Pulpen | 100000 |
| 003 | Nana | Makassar | Pulpen | 20000 |
| 004 | Rina | Makassar | Buku | 30000 |
| 005 | Antonion | Palu | Buku | 40000 |
| 006 | Riri | Makassar | Kertas HVS | 60000 |

Tabel “pelanggan2” :

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| id | Namapelanggan | Alamat | Barang\_yang\_dibeli | Dibayar |
| 001 | Angga | Mamuju | Kertas HVS | 120000 |
| 002 | Ari | Majene | Pulpen | 100000 |
| 003 | Nara | Makassar | Pulpen | 20000 |
| 004 | Romi | Makassar | Buku | 30000 |
| 005 | Ajeng | Palu | Buku | 40000 |
| 006 | Mariadi | Makassar | Kertas HVS | 60000 |

Menggunakan union :

SELECT Alamat FROM pelanggan1  
UNION  
SELECT Alamat FROM pelanggan2;

|  |
| --- |
| Alamat |
| Mamuju |
| Majene |
| Palu |
| Makassar |

Menggunakan UNION ALL

SELECT Alamat FROM pelanggan1  
UNION ALL  
SELECT Alamat FROM pelanggan2;

|  |
| --- |
| Alamat |
| Mamuju |
| Majene |
| Makassar |
| Makassar |
| Palu |
| Makassar |
| Mamuju |
| Majene |
| Makassar |
| Makassar |
| Palu |
| Makassar |

1. **SQL GROUP**

GROUP BY mengelompokkan baris yang memiliki nilai yang sama ke dalam baris ringkasan, seperti "temukan jumlah pelanggan di setiap negara". Pernyataan GROUP BY sering digunakan dengan fungsi agregat (COUNT(), MAX(), MIN(), SUM(), AVG()) untuk mengelompokkan kumpulan hasil menurut satu atau beberapa kolom.

Syntax:

SELECT column\_name(s)  
FROM table\_name  
WHERE condition  
GROUP BY column\_name(s)

Contoh :

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| id | Namapelanggan | Alamat | Barang\_yang\_dibeli | Dibayar |
| 001 | Angga | Mamuju | Kertas HVS | 120000 |
| 002 | Ari | Majene | Pulpen | 100000 |
| 003 | Nara | Makassar | Pulpen | 20000 |
| 004 | Romi | Makassar | Buku | 30000 |
| 005 | Ajeng | Palu | Buku | 40000 |
| 006 | Mariadi | Makassar | Kertas HVS | 60000 |

Menggunakan syntak berikut :

SELECT COUNT(id), Alamat  
FROM Customers  
GROUP BY Alamat;

Maka akan muncul tabel berikut :

|  |  |
| --- | --- |
| Count(id) | Alamat |
| 1 | Mamuju |
| 1 | Majene |
| 1 | Makassar |
| 1 | Makassar |
| 1 | Palu |
| 1 | Makassar |

1. **SQL HAVING**

HAVING digunakan untuk menggantikan WHERE ketika menggunakan Group BY yang datanya di aggregasi atau . Syntaxnya :

SELECT column\_name(s)  
FROM table\_name  
WHERE condition  
GROUP BY column\_name(s)HAVING conditionORDER BY column\_name(s);

1. **SQL EXISTS**

EXISTS digunakan untuk menguji keberadaan record apapun dalam subquery. Operator EXISTS mengembalikan TRUE jika subquery mengembalikan satu atau lebih record. EXISTS Syntax :

SELECT column\_name(s)  
FROM table\_name  
WHERE EXISTS  
(SELECT column\_name FROM table\_name WHERE condition);

1. **SQL ANY and ALL**

ANY dan ALL memungkinkan Anda melakukan perbandingan antara nilai kolom tunggal dan rentang nilai lainnya.

* Operator ANY SQL

Mengembalikan nilai boolean sebagai hasilnya mengembalikan TRUE jika APAPUN dari nilai subquery memenuhi kondisi APAPUN berarti kondisi akan benar jika operasi benar untuk salah satu nilai dalam rentang. ANY Syntax :

SELECT column\_name(s)  
FROM table\_name  
WHERE column\_name operator ANY  
(SELECT column\_nameFROM table\_nameWHERE condition);

* Operator ALL SQL

mengembalikan nilai boolean sebagai hasilnya mengembalikan TRUE jika SEMUA nilai subquery memenuhi kondisi digunakan dengan pernyataan SELECT, WHERE dan HAVING ALL berarti bahwa kondisi akan benar hanya jika operasi benar untuk semua nilai dalam rentang.

### ALL Syntax dengan SELECT

SELECT ALL column\_name(s)  
FROM table\_name  
WHERE condition;

### ALL Syntax dengan WHERE atau HAVING

SELECT column\_name(s)  
FROM table\_name  
WHERE column\_name operator ALL  
(SELECT column\_nameFROM table\_nameWHERE condition);

1. **SQL SELECT INTO**

SELECT INTO menyalin data dari satu tabel ke tabel baru.

### Syntax SELECT INTO

mengnyalin semua kolom ke dalam tabel baru :

SELECT \*  
INTO newtable [IN externaldb]  
FROM oldtableWHERE condition;

Mengnyalin hanya beberapa kolom kedalm kolom baru :

SELECT column1, column2, column3, ...  
INTO newtable [IN externaldb]  
FROM oldtableWHERE condition;

1. **SQL INSERT INTO SELECT**

INSERT INTO SELECT menyalin data dari satu tabel dan menyisipkannya ke tabel lain. Pernyataan INSERT INTO SELECT mengharuskan tipe data dalam tabel sumber dan target cocok. Catatan yang ada di tabel target tidak terpengaruh. Syntax INSERT INTO SELECT :

Menyalin Semua kolom dari satu kolom kekolom lainnya :

INSERT INTO table2  
SELECT \* FROM table1WHERE condition;

Menyalin hanya beberapa kolom dari satu kolom kekolom lainnya:

INSERT INTO table2 (column1, column2, column3, ...)  
SELECT column1, column2, column3, ...  
FROM table1  
WHERE condition;

1. **SQL CASE**

CASE melewati kondisi dan mengembalikan nilai ketika kondisi pertama terpenuhi (seperti pernyataan if-then-else). Jadi, begitu suatu kondisi benar, itu akan berhenti membaca dan mengembalikan hasilnya. Jika tidak ada kondisi yang benar, ini mengembalikan nilai dalam klausa ELSE. Jika tidak ada bagian ELSE dan tidak ada kondisi yang benar, ia mengembalikan NULL. Syntax CASE :

CASE  
    WHEN condition1 THEN result1  
    WHEN condition2 THEN result2  
    WHEN conditionN THEN resultN  
    ELSE result  
END;

1. **SQL NULL Functions**

Lihat tabel "Barang" berikut:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| P\_Id | ProductName | UnitPrice | UnitsInStock | UnitsOnOrder |
| 1 | Jarlsberg | 10.45 | 16 | 15 |
| 2 | Mascarpone | 32.56 | 23 |  |
| 3 | Gorgonzola | 15.67 | 9 | 20 |

Misalkan kolom "UnitsOnOrder" adalah opsional, dan mungkin berisi nilai NULL.

Perhatikan pernyataan SELECT berikut:

SELECT ProductName, UnitPrice \* (UnitsInStock + UnitsOnOrder)  
FROM Products;

Dalam contoh di atas, jika salah satu nilai "UnitsOnOrder" adalah NULL, hasilnya akan NULL.

Perhatikan pernyataan SELECT berikut:

SELECT ProductName, UnitPrice \* (UnitsInStock + UnitsOnOrder)  
FROM Products;

Dalam contoh di atas, jika salah satu nilai "UnitsOnOrder" adalah NULL, hasilnya akan NULL. Solusinya Fungsi MySQL IFNULL() memungkinkan Anda mengembalikan nilai alternatif jika ekspresi NULL:

SELECT ProductName, UnitPrice \* (UnitsInStock + IFNULL(UnitsOnOrder, 0))  
FROM Products;

atau kita bisa menggunakan fungsi COALESCE(), seperti ini:

SELECT ProductName, UnitPrice \* (UnitsInStock + COALESCE(UnitsOnOrder, 0))  
FROM Products;

1. **SQL Stored Procedures**

Apa itu Stored Procedures ? Stored Procedures adalah kode SQL yang disiapkan yang dapat Anda simpan, sehingga kode tersebut dapat digunakan kembali berulang kali. Jadi, jika Anda memiliki kueri SQL yang Anda tulis berulang kali, simpan sebagai prosedur tersimpan, lalu panggil saja untuk menjalankannya. Anda juga dapat meneruskan parameter ke prosedur tersimpan, sehingga prosedur tersimpan dapat bertindak berdasarkan nilai parameter yang diteruskan.

Syntax Stored Procedure:

CREATE PROCEDURE procedure\_name  
AS  
sql\_statement  
GO;

Execute a Stored Procedure

EXEC procedure\_name;

1. **SQL Comments**

Commnet atau komentar digunakan untuk menjelaskan bagian dari pernyataan SQL, atau untuk mencegah eksekusi pernyataan SQL.

* Single Line Comments

Komentar satu baris dimulai dengan --. Teks apa pun antara -- dan akhir baris akan diabaikan (tidak akan dieksekusi). Contoh berikut menggunakan komentar satu baris sebagai penjelasan:

--Select all:  
SELECT \* FROM Customers;

Contoh berikut menggunakan komentar satu baris untuk mengabaikan akhir baris:

SELECT \* FROM Customers -- WHERE City='Berlin';

Contoh berikut menggunakan komentar satu baris untuk mengabaikan pernyataan:

--SELECT \* FROM Customers;  
SELECT \* FROM Products;

* Multi-line Comments

Multi-line Comments dimulai dengan /\* dan diakhiri dengan \*/. Setiap teks antara /\* dan \*/ akan diabaikan. Contoh berikut menggunakan komentar multi-baris sebagai penjelasan:

/\*Select all the columns  
of all the records  
in the Customers table:\*/  
SELECT \* FROM Customers;

1. **SQL Operators**

* SQL operaotor Aritmatika

|  |  |
| --- | --- |
| Operator | Penjelasan |
| + | Menjumlah |
| - | Mengurangi |
| \* | Mengkalikan |
| / | Membagi |
| % | Modul |

* SQL Bitwise Operators

|  |  |
| --- | --- |
| Operator | Penjelasan |
| & | Bitwise AND |
| | | Bitwise OR |
| ^ | Bitwise exclusive OR |

* SQL Comparison Operators

|  |  |
| --- | --- |
| Operator | Penjelasan |
| = | Sama dengan |
| > | Lebih besar |
| < | Lebih kecil |
| >= | Lebih besar atau sama dengan |
| <= | Lebih kecil atau sama dengan |
| <> | Tidak dama dengan |

* SQL Compound Operators

|  |  |
| --- | --- |
| Operator | Penjelasan |
| += | Tambah dengan |
| -= | Kurangi dengan |
| \*= | Kali dengan |
| /= | Bagi dengan |
| %= | Modulo dengan |
| &= | Bitwise AND dengan |
| ^-= | Bitwise exclusive dengan |
| |\*= | Bitwise OR dengan |

* SQL Logical Operators

|  |  |
| --- | --- |
| Operator | Penjelasan |
| ALL | TRUE jika semua nilai subquery memenuhi kondisi |
| AND | TRUE jika semua kondisi yang dipisahkan oleh AND bernilai TRUE |
| ANY | TRUE jika salah satu nilai subquery memenuhi kondisi |
| BETWEEN | TRUE if the operand is within the range of comparisons |
| EXISTS | TRUE if the subquery returns one or more records |
| IN | TRUE if the operand is equal to one of a list of expressions |
| LIKE | TRUE if the operand matches a pattern |
| NOT | Displays a record if the condition(s) is NOT TRUE |
| OR | TRUE if any of the conditions separated by OR is TRUE |
| SOME | TRUE if any of the subquery values meet the condition |

**SQL DATABASE**

1. **SQL CREATE DATABASE**

CREATE DATABASE digunakan untuk membuat database SQL baru. Syntax :

CREATE DATABASE databasename;

Contoh : CREATE DATABASE pelanggan;

1. **SQL DROP DATABASE**

DROP DATABASE digunakan untuk mengahapus database SQL yang ada. Syntax :

DROP DATABASE databasename;

Contoh : DROP DATABASE pelanggan;

1. **SQL BACKUP DATABASE**

CADANGAN DATABASE digunakan di SQL Server untuk membuat cadangan penuh dari database SQL yang ada. Syntax :

BACKUP DATABASE databasename  
TO DISK = 'filepath';

SQL BACKUP dengan DIFFERENTIAL :

BACKUP DATABASE databasename  
TO DISK = 'filepath'  
WITH DIFFERENTIAL;

Contoh :

BACKUP DATABASE pelanggan  
TO DISK = 'D:\backups\testDB.bak';

Contoh BACKUP dengan DIFFERENTIAL :

BACKUP DATABASE pelanggan  
TO DISK = 'D:\backups\testDB.bak'  
WITH DIFFERENTIAL;

1. **SQL CREATE TABLE**

Pernyataan CREATE TABLE digunakan untuk membuat tabel baru dalam database. Syntax :

CREATE TABLE table\_name (column1 datatype, column2 datatype, column3 datatype, ....);

Parameter kolom menentukan nama kolom tabel.Parameter tipe data menentukan tipe data yang dapat disimpan oleh kolom (misalnya varchar, integer, tanggal, dll.). contoh :

CREATE TABLE Pelanggan ( id int, nama varchar(255), Namabelakang varchar(255), Alamat varchar(255), Kota varchar(255));

1. **SQL DROP TABLE**

DROP TABLE digunakan untuk menghapus tabel yang ada dalam database. Syntax:

DROP TABLE table\_name;

Contoh : DROP TABLE Pelanggan;

Ada juga istilah TRUNCATE TABLE yang mana digunakan untuk menghapus data di dalam tabel, tetapi bukan tabel itu sendiri. Syntax :

TRUNCATE TABLE table\_name;

1. **SQL ALTER TABLE**

ALTER TABLE digunakan untuk menambah, menghapus, atau memodifikasi kolom dalam tabel yang sudah ada. ALTER TABLE juga digunakan untuk menambah dan menghapus berbagai batasan pada tabel yang ada.

* ALTER TABLE – ADD kolom

Untuk menambahkan kolom dalam tabel menggunakan syntax :

ALTER TABLE table\_name  
ADD column\_name datatype;

Contoh :

ALTER TABLE Pelanggan  
ADD NoTelp(15);

* ALTER TABLE – DROP kolom

Untuk menghapus kolom dalam tabel, gunakan sintaks berikut (perhatikan bahwa beberapa sistem database tidak mengizinkan penghapusan kolom):

ALTER TABLE table\_name  
DROP COLUMN column\_name;

Contoh :

ALTER TABLE Pelanggan  
DROP NoTelp(15);

* ALTER TABLE - ALTER/MODIFY

Untuk mengubah tipe data kolom dalam tabel, sintaks berikut:

ALTER TABLE table\_name  
MODIFY COLUMN column\_name datatype;

Contoh :

ALTER TABLE Pelanggan   
MODIFY COLUMN NoTlp;

1. **SQL Constraints**

Constraints dapat ditentukan ketika tabel dibuat dengan pernyataan CREATE TABLE, atau setelah tabel dibuat dengan pernyataan ALTER TABLE. Syntax :

CREATE TABLE table\_name ( column1 datatype *constraint*, column2 datatype *constraint*, column3 datatype *constraint*, ....);

Constraints digunakan untuk menentukan aturan untuk data dalam tabel. Constraint digunakan untuk membatasi jenis data yang dapat masuk ke dalam tabel. Ini memastikan keakuratan dan keandalan data dalam tabel. Jika ada pelanggaran antara batasan dan tindakan data, tindakan dibatalkan. Batasan dapat berupa level kolom atau level tabel. Batasan tingkat kolom berlaku untuk kolom, dan batasan tingkat tabel berlaku untuk seluruh tabel. Batasan berikut biasanya digunakan dalam SQL:

* NOT NULL - Memastikan bahwa kolom tidak boleh memiliki nilai NULL
* UNIK - Memastikan bahwa semua nilai dalam kolom berbeda
* KUNCI UTAMA - Kombinasi NOT NULL dan UNIK. Secara unik mengidentifikasi setiap baris dalam sebuah tabel
* KUNCI ASING - Mencegah tindakan yang akan menghancurkan tautan antar tabel
* CHECK - Memastikan bahwa nilai dalam kolom memenuhi kondisi tertentu
* DEFAULT - Menetapkan nilai default untuk kolom jika tidak ada nilai yang ditentukan
* CREATE INDEX - Digunakan untuk membuat dan mengambil data dari database dengan sangat cepat

1. **SQL NOT NULL**

Secara default, kolom dapat menampung nilai NULL. Batasan NOT NULL memaksa kolom untuk NOT menerima nilai NULL. Ini memaksa bidang untuk selalu berisi nilai, yang berarti Anda tidak dapat menyisipkan catatan baru, atau memperbarui catatan tanpa menambahkan nilai ke bidang ini.

* SQL NOT NULL di CREATE TABLE

Contoh :

CREATE TABLE Pelanggan (id int NOT NULL, Nama(255) NOT NULL, Alamat varchar(255) NOT NULL, Umur int );

* SQL NOT NULL on ALTER TABLE

Contoh :

ALTER TABLE Persons  
MODIFY Age int NOT NULL;

1. **SQL UNIQUE**

UNIQUE memastikan bahwa semua nilai dalam kolom berbeda. Kendala UNIQUE dan PRIMARY KEY memberikan jaminan keunikan untuk kolom atau kumpulan kolom. Kendala PRIMARY KEY secara otomatis memiliki kendala UNIQUE. Namun, Anda dapat memiliki banyak batasan UNIQUE per tabel, tetapi hanya satu batasan PRIMARY KEY per tabel.

* SQL UNIQUE di CREATE TABLE

Contoh :

CREATE TABLE pelanggan (  
    id int NOT NULL,  
    nama varchar(255) NOT NULL,  
    Alamat varchar(255),  
    Umur int

UNIQUE (ID)  
);

* SQL UNIQUE di ALTER TABLE

ALTER TABLE pelanggan  
ADD UNIQUE (id);

Dan juga

ALTER TABLE pelanggan  
ADD CONSTRAINT Email UNIQUE (id,nama);

* DROP sebuah UNIQUE

ALTER TABLE Pelanggan  
DROP INDEX Email;

1. **SQL PRIMARY KEY**

PRIMARY KEY secara unik mengidentifikasi setiap record dalam sebuah tabel. Kunci utama harus berisi nilai UNIK, dan tidak boleh berisi nilai NULL. Sebuah tabel hanya dapat memiliki SATU kunci utama; dan dalam tabel, kunci utama ini dapat terdiri dari satu atau beberapa kolom (bidang).

* SQL PRIMARY KEY on CREATE TABLE

Contoh :

CREATE TABLE pelanggan (  
    id int NOT NULL,  
    nama varchar(255) NOT NULL,  
    Alamat varchar(255),  
    Tahun int,  
    PRIMARY KEY (ID)  
);

* SQL PRIMARY KEY on ALTER TABLE

ALTER TABLE Persons  
ADD PRIMARY KEY (ID);

* DROP a PRIMARY KEY Constraint

ALTER TABLE Persons  
DROP PRIMARY KEY;

1. **SQL FOREIGN KEY**

FOREIGN KEY digunakan untuk mencegah tindakan yang akan menghancurkan link antar tabel. FOREIGN KEY adalah bidang (atau kumpulan bidang) dalam satu tabel, yang merujuk ke PRYMARY KEY di tabel lain. Tabel dengan kunci asing disebut tabel anak, dan tabel dengan kunci utama disebut tabel referensi atau tabel induk.

* OREIGN KEY di CREATE TABLE

CREATE TABLE Pelanggan (  
    id int NOT NULL,  
    nomorpelanggan int NOT NULL,  
    idKasir int,  
    PRIMARY KEY (id),  
    FOREIGN KEY (idasir) REFERENCES kasir(idKasir));

* FOREIGN KEY on ALTER TABLE

ALTER TABLE Pelanggan  
ADD FOREIGN KEY (idKasir) REFERENCES Kasir(idKasri);

* DROP a FOREIGN KEY

ALTER TABLE pelanggan  
DROP FOREIGN KEY Pesanan\_pelanggan;

1. **SQL CHECK**

CHECK digunakan untuk membatasi rentang nilai yang dapat ditempatkan dalam kolom. Jika Anda menentukan batasan CHECK pada kolom, itu hanya akan mengizinkan nilai tertentu untuk kolom ini. Jika Anda mendefinisikan batasan CHECK pada tabel, itu dapat membatasi nilai di kolom tertentu berdasarkan nilai di kolom lain di baris.

* SQL CHECK di CREATE TABLE

CREATE TABLE pelanggan (  
    id int NOT NULL,  
    nama varchar(255) NOT NULL,  
    alamat varchar(255),  
    Umur int,  
    CHECK (Age>=18)  
);

* SQL CHECK di ALTER TABLE

ALTER TABLE Pelanggan  
ADD CHECK (umur>=18);

* DROP a CHECK Constraint

ALTER TABLE Pelanggan  
DROP CONSTRAINT CHK\_Pelanggantahun;

1. **SQL DEFAULT**

DEFAULT digunakan untuk menetapkan nilai default untuk kolom. Nilai default akan ditambahkan ke semua catatan baru, jika tidak ada nilai lain yang ditentukan.

* SQL DEFAULT di CREATE TABLE

CREATE TABLE Pelanggan (  
    id int NOT NULL,  
    nama varchar(255) NOT NULL,  
    Alamat varchar(255),  
    Umur int,  
    Kota varchar(255) DEFAULT 'Mamuju'  
);

* SQL DEFAULT di ALTER TABLE

ALTER TABLE Pelanggan  
ALTER Kota SET DEFAULT 'Mamuju';

* DROP Sebuah DEFAULT

ALTER TABLE Pelanggan  
ALTER Kota DROP DEFAULT;

1. **SQL INDEX**

CREATE INDEX digunakan untuk membuat indeks dalam tabel. Indeks digunakan untuk mengambil data dari database lebih cepat daripada sebaliknya. Pengguna tidak dapat melihat indeks, mereka hanya digunakan untuk mempercepat pencarian/kueri. Syntax :

CREATE INDEX index\_name  
ON table\_name (column1, column2, ...);

* CREATE UNIQUE INDEX

CREATE UNIQUE INDEX index\_name  
ON table\_name (column1, column2, ...);

* Contoh CREATE INDEX

CREATE INDEX idx\_nama  
ON Pelanggan (nama);

* DROP INDEX

ALTER TABLE table\_nameDROP INDEX index\_name;

1. **SQL AUTO INCREMENT**

Auto-increment memungkinkan nomor unik dihasilkan secara otomatis ketika catatan baru dimasukkan ke dalam tabel. Seringkali ini adalah bidang kunci utama yang ingin kita buat secara otomatis setiap kali catatan baru dimasukkan. Syntax untuk MySQL :

CREATE TABLE pelanggan (  
    id int NOT NULL AUTO\_INCREMENT,  
    nama varchar(255) NOT NULL,  
    Alamat varchar(255),  
    Umur int,  
    PRIMARY KEY (id)  
);

1. **SQL Dates**

Selama data Anda hanya berisi bagian tanggal, kueri Anda akan berfungsi seperti yang diharapkan. Namun, jika ada porsi waktu yang terlibat, itu menjadi lebih rumit.

* Tipe data SQL Date

MySQL hadir dengan tipe data berikut untuk menyimpan tanggal atau nilai tanggal/waktu dalam database:

* TANGGAL - format YYYY-MM-DD
* DATETIME - format: YYYY-MM-DD HH:MI:SS
* TIMESTAMP - format: YYYY-MM-DD HH:MI:SS
* TAHUN - format YYYY atau YY

Contoh : SELECT \* FROM Orders WHERE OrderDate='2008-11-11'

1. **SQL Views**

Dalam SQL, VIEW adalah tabel virtual berdasarkan kumpulan hasil dari pernyataan SQL. VIEW berisi baris dan kolom, seperti tabel nyata. Bidang dalam tampilan adalah bidang dari satu atau beberapa tabel nyata dalam database. Anda dapat menambahkan pernyataan dan fungsi SQL ke tampilan dan menyajikan data seolah-olah data berasal dari satu tabel. Tampilan dibuat dengan statement CREATE VIEW. Syntax :

CREATE VIEW view\_name AS  
SELECT column1, column2, ...  
FROM table\_name  
WHERE condition;

Contoh :

CREATE VIEW [Pelanggan Mamuju] AS  
SELECT Nama, Alamat  
FROM Pelanggan   
WHERE Alamat = 'Mamuju';

1. **SQL Injection**

SQL Injection adalah teknik injeksi kode yang dapat menghancurkan database Anda.

SQL Injection adalah salah satu teknik peretasan web yang paling umum. SQL Injection adalah penempatan kode berbahaya dalam pernyataan SQL, melalui input halaman web.

* SQL Injection dasar di 1=1 adalah selalu True/benar

Contoh : SELECT \* FROM Users WHERE UserId = 105 OR 1=1;

* SQL Injection dasar di ""="" adalah selalu True/benar

Contoh :

uName = getRequestString("username");  
uPass = getRequestString("userpassword");  
  
sql = 'SELECT \* FROM Users WHERE Name ="' + uName + '" AND Pass ="' + uPass + '"'

* SQL Injection dasar di Batched SQL Statements

Contoh :

SELECT \* FROM Users; DROP TABLE Suppliers

* Use SQL Parameters untuk Protection

txtUserId = getRequestString("UserId");  
txtSQL = "SELECT \* FROM Users WHERE UserId = @0";  
db.Execute(txtSQL,txtUserId);

1. **SQL Hosting**

Jika Anda ingin situs web Anda dapat menyimpan dan mengambil data dari database, server web Anda harus memiliki akses ke sistem database yang menggunakan bahasa SQL. Jika server web Anda di-host oleh Penyedia Layanan Internet (ISP), Anda harus mencari paket hosting SQL. Database hosting SQL yang paling umum adalah MS SQL Server, Oracle, MySQL, dan MS Access.

1. **SQL Data Types**

Di MySQL Data Types (Version 8.0) ada tipe data utama yaitu : string, numeric, and date and time.

**Tipe Data String**

|  |  |
| --- | --- |
| Tipe data | Penjelasan |
| CHAR(size) | Sebuah string panjang TETAP (dapat berisi huruf, angka, dan karakter khusus). Parameter ukuran menentukan panjang kolom dalam karakter - bisa dari 0 hingga 255. Standarnya adalah 1 |
| VARCHAR(size) | Sebuah string panjang VARIABEL (dapat berisi huruf, angka, dan karakter khusus). Parameter ukuran menentukan panjang kolom maksimum dalam karakter - bisa dari 0 hingga 65535 |
| BINARY(size) | Sama dengan CHAR(), tetapi menyimpan string byte biner. Parameter ukuran menentukan panjang kolom dalam byte. Standarnya adalah 1 |
| VARBINARY(size) | Sama dengan VARCHAR(), tetapi menyimpan string byte biner. Parameter ukuran menentukan panjang kolom maksimum dalam byte. |
| TINYBLOB | Untuk BLOB (Objek Besar Biner). Panjang maksimum: 255 byte |
| TINYTEXT | Memegang string dengan panjang maksimum 255 karakter |
| TEXT(size) | Memegang string dengan panjang maksimum 65.535 byte |
| BLOB(size) | For BLOBs (Binary Large Objects). Holds up to 65,535 bytes of data |
| MEDIUMTEXT | Untuk BLOB (Objek Besar Biner). Menampung hingga 65.535 byte data |
| MEDIUMBLOB | Untuk BLOB (Objek Besar Biner). Menampung hingga 16.777.215 byte data |
| LONGTEXT | Memegang string dengan panjang maksimum 4.294.967.295 karakter |
| LONGBLOB | Untuk BLOB (Objek Besar Biner). Menampung hingga 4.294.967.295 byte data |
| ENUM(val1, val2, val3, ...) | Objek string yang hanya dapat memiliki satu nilai, dipilih dari daftar kemungkinan nilai. Anda dapat membuat daftar hingga 65535 nilai dalam daftar ENUM. Jika nilai yang dimasukkan tidak ada dalam daftar, nilai kosong akan dimasukkan. Nilai diurutkan sesuai urutan yang Anda masukkan |
| SET(val1, val2, val3, ...) | Objek string yang dapat memiliki 0 atau lebih nilai, dipilih dari daftar kemungkinan nilai. Anda dapat membuat daftar hingga 64 nilai dalam daftar SET |

**Type data Numeric**

|  |  |
| --- | --- |
| Tipe data | Penjelasan |
| BIT(size) | Jenis nilai bit. Jumlah bit per nilai ditentukan dalam ukuran. Parameter ukuran dapat menyimpan nilai dari 1 hingga 64. Nilai default untuk ukuran adalah 1. |
| TINYINT(size) | Sebuah bilangan bulat yang sangat kecil. Rentang yang ditandatangani adalah dari -128 hingga 127. Rentang yang tidak ditandatangani adalah dari 0 hingga 255. Parameter ukuran menentukan lebar tampilan maksimum (yaitu 255) |
| BOOL | Nol dianggap salah, nilai bukan nol dianggap benar. |
| BOOLEAN | Sama dengan BOOL |
| SMALLINT(size) | Sebuah bilangan bulat kecil. Rentang yang ditandatangani adalah dari -32768 hingga 32767. Rentang yang tidak ditandatangani adalah dari 0 hingga 65535. Parameter ukuran menentukan lebar tampilan maksimum (yaitu 255) |
| MEDIUMINT(size) | Sebuah bilangan bulat sedang. Rentang yang ditandatangani adalah dari -8388608 hingga 8388607. Rentang yang tidak ditandatangani adalah dari 0 hingga 16777215. Parameter ukuran menentukan lebar tampilan maksimum (yaitu 255) |
| INT(size) | Sebuah bilangan bulat sedang. Rentang yang ditandatangani adalah dari -2147483648 hingga 2147483647. Rentang yang tidak ditandatangani adalah dari 0 hingga 4294967295. Parameter ukuran menentukan lebar tampilan maksimum (yaitu 255) |
| INTEGER(size) | Sama dengan INT(ukuran) |
| BIGINT(size) | Sebuah bilangan bulat besar. Rentang yang ditandatangani adalah dari -9223372036854775808 hingga 9223372036854775807. Rentang yang tidak ditandatangani adalah dari 0 hingga 18446744073709551615. Parameter ukuran menentukan lebar tampilan maksimum (yaitu 255) |
| FLOAT(size, d) | Angka titik mengambang. Jumlah total digit ditentukan dalam ukuran. Jumlah digit setelah titik desimal ditentukan dalam parameter d. Sintaks ini tidak digunakan lagi di MySQL 8.0.17, dan akan dihapus di versi MySQL mendatang |
| FLOAT(p) | Angka titik mengambang. MySQL menggunakan nilai p untuk menentukan apakah akan menggunakan FLOAT atau DOUBLE untuk tipe data yang dihasilkan. Jika p adalah dari 0 hingga 24, tipe datanya menjadi FLOAT(). Jika p dari 25 hingga 53, tipe datanya menjadi DOUBLE() |
| DOUBLE(size, d) | Angka floating point ukuran normal. Jumlah total digit ditentukan dalam ukuran. Jumlah digit setelah titik desimal ditentukan dalam parameter d |
| DOUBLE PRECISION(size, d) |  |
| DECIMAL(size, d) | Angka titik tetap yang tepat. Jumlah total digit ditentukan dalam ukuran. Jumlah digit setelah titik desimal ditentukan dalam parameter d. Jumlah maksimum untuk ukuran adalah 65. Jumlah maksimum untuk d adalah 30. Nilai default untuk ukuran adalah 10. Nilai default untuk d adalah 0. |
| DEC(size, d) | Sama dengan DECIMAL(ukuran,d) |

Tipe data Date and Time

|  |  |
| --- | --- |
| Tipe data | Penjelasan |
| DATE | Kencan. Format: YYYY-MM-DD. Rentang yang didukung adalah dari '1000-01-01' hingga '9999-12-31' |
| DATETIME(fsp) | Kombinasi tanggal dan waktu. Format: YYYY-MM-DD jj:mm:dd. Rentang yang didukung adalah dari '1000-01-01 00:00:00' hingga '9999-12-31 23:59:59'. Menambahkan DEFAULT dan ON UPDATE dalam definisi kolom untuk mendapatkan inisialisasi dan pembaruan otomatis ke tanggal dan waktu saat ini |
| TIMESTAMP(fsp) | Sebuah stempel waktu. Nilai TIMESTAMP disimpan sebagai jumlah detik sejak zaman Unix ('1970-01-01 00:00:00' UTC). Format: YYYY-MM-DD jj:mm:dd. Rentang yang didukung adalah dari '1970-01-01 00:00:01' UTC hingga '2038-01-09 03:14:07' UTC. Inisialisasi dan pembaruan otomatis ke tanggal dan waktu saat ini dapat ditentukan menggunakan DEFAULT CURRENT\_TIMESTAMP dan ON UPDATE CURRENT\_TIMESTAMP dalam definisi kolom |
| TIME(fsp) | Sebuah waktu. Format: jj:mm:dd. Rentang yang didukung adalah dari '-838:59:59' hingga '838:59:59' |
| YEAR | Setahun dalam format empat digit. Nilai yang diizinkan dalam format empat digit: 1901 hingga 2155, dan 0000.  MySQL 8.0 tidak mendukung tahun dalam format dua digit.MySQL 8.0 does not support year in two-digit format. |

**SQL REFERENCE**

1. **SQL Keywords**

|  |  |
| --- | --- |
| Keyword | Penjelasan |
| ADD | Menambahkan kolom di tabel yang ada |
| ADD CONSTRAINT | Menambahkan batasan setelah tabel dibuat |
| ALTER | Menambah, menghapus, atau memodifikasi kolom dalam tabel, atau mengubah tipe data kolom dalam tabel |
| ALTER COLUMN | Mengubah tipe data kolom dalam tabel |
| ALTER TABLE | Menambah, menghapus, atau memodifikasi kolom dalam tabel |
| ALL | Mengembalikan nilai true jika semua nilai subquery memenuhi kondisi |
| AND | Hanya menyertakan baris yang kedua kondisinya benar |
| ANY | Mengembalikan nilai true jika salah satu nilai subquery memenuhi kondisi |
| AS | Mengganti nama kolom atau tabel dengan alias |
| ASC | Mengurutkan hasil yang diatur dalam urutan menaik |
| BACKUP DATABASE | Membuat cadangan dari database yang ada |
| BETWEEN | Memilih nilai dalam rentang tertentu |
| CASE | Membuat output yang berbeda berdasarkan kondisi |
| CHECK | Batasan yang membatasi nilai yang dapat ditempatkan dalam kolom |
| COLUMN | Mengubah tipe data kolom atau menghapus kolom dalam tabel |
| CONSTRAINT | Menambah atau menghapus batasan |
| CREATE | Membuat database, indeks, tampilan, tabel, atau prosedur |
| CREATE DATABASE | Membuat database SQL baru |
| CREATE INDEX | Membuat indeks pada tabel (memungkinkan nilai duplikat) |
| CREATE OR REPLACE VIEW | Memperbarui tampilan |
| CREATE TABLE | Membuat tabel baru di database |
| CREATE PROCEDURE | Membuat prosedur tersimpan |
| CREATE UNIQUE INDEX | Membuat indeks unik pada tabel (tidak ada nilai duplikat) |
| CREATE VIEW | Membuat tampilan berdasarkan kumpulan hasil dari pernyataan SELECT |
| DATABASE | Membuat atau menghapus database SQL |
| DEFAULT | Batasan yang memberikan nilai default untuk kolom |
| DELETE | Menghapus baris dari tabel |
| DESC | Mengurutkan hasil yang diatur dalam urutan menurun |
| DISTINCT | Hanya memilih nilai yang berbeda (berbeda) |
| DROP | Menghapus kolom, batasan, database, indeks, tabel, atau tampilan |
| DROP COLUMN | Menghapus kolom dalam tabel |
| DROP CONSTRAINT | Menghapus UNIK, KUNCI UTAMA, KUNCI ASING, atau PERIKSA kendala |
| DROP DATABASE | Menghapus database SQL yang ada |
| DROP DEFAULT | Menghapus batasan DEFAULT |
| DROP INDEX | Menghapus indeks dalam tabel |
| DROP TABLE | Menghapus tabel yang ada di database |
| DROP VIEW | Menghapus tampilan |
| EXEC | Menjalankan prosedur tersimpan |
| EXISTS | Menguji keberadaan record apa pun dalam subquery |
| FOREIGN KEY | Batasan yang merupakan kunci yang digunakan untuk menghubungkan dua tabel bersama-sama |
| FROM | Menentukan tabel mana yang akan dipilih atau dihapus datanya |
| FULL OUTER JOIN | Mengembalikan semua baris ketika ada kecocokan di tabel kiri atau tabel kanan |
| GROUP BY | Mengelompokkan kumpulan hasil (digunakan dengan fungsi agregat: COUNT, MAX, MIN, SUM, AVG) |
| HAVING | Digunakan sebagai pengganti WHERE dengan fungsi agregat |
| IN | Memungkinkan Anda untuk menentukan beberapa nilai dalam klausa WHERE |
| INDEX | Membuat atau menghapus indeks dalam tabel |
| INNER JOIN | Mengembalikan baris yang memiliki nilai yang cocok di kedua tabel |
| INSERT INTO | Menyisipkan baris baru dalam tabel |
| INSERT INTO SELECT | Menyalin data dari satu tabel ke tabel lain |
| IS NULL | Tes untuk nilai kosong |
| IS NOT NULL | Tes untuk nilai yang tidak kosong |
| JOIN | Bergabung dengan tabel |
| LEFT JOIN | Mengembalikan semua baris dari tabel kiri, dan baris yang cocok dari tabel kanan |
| LIKE | Mencari pola tertentu dalam kolom |
| LIMIT | Menentukan jumlah catatan yang akan dikembalikan dalam kumpulan hasil |
| NOT | Hanya menyertakan baris yang kondisinya tidak benar |
| NOT NULL | Batasan yang memaksa kolom untuk tidak menerima nilai NULL |
| OR | Termasuk baris yang salah satu kondisinya benar |
| ORDER BY | Mengurutkan hasil yang diatur dalam urutan menaik atau menurun |
| OUTER JOIN | Mengembalikan semua baris ketika ada kecocokan di tabel kiri atau tabel kanan |
| PRIMARY KEY | Batasan yang secara unik mengidentifikasi setiap record dalam tabel database |
| PROCEDURE | Prosedur tersimpan |
| RIGHT JOIN | Mengembalikan semua baris dari tabel kanan, dan baris yang cocok dari tabel kiri |
| ROWNUM | Menentukan jumlah catatan yang akan dikembalikan dalam kumpulan hasil |
| SELECT | Memilih data dari database |
| SELECT DISTINCT | Hanya memilih nilai yang berbeda (berbeda) |
| SELECT INTO | Menyalin data dari satu tabel ke tabel baru |
| SELECT TOP | Menentukan jumlah catatan yang akan dikembalikan dalam kumpulan hasil |
| SET | Menentukan kolom dan nilai mana yang harus diperbarui dalam tabel |
| TABLE | Membuat tabel, atau menambahkan, menghapus, atau memodifikasi kolom dalam tabel, atau menghapus tabel atau data di dalam tabel |
| TOP | Menentukan jumlah catatan yang akan dikembalikan dalam kumpulan hasil |
| TRUNCATE TABLE | Menghapus data di dalam tabel, tetapi bukan tabel itu sendiri |
| UNION | Menggabungkan kumpulan hasil dari dua atau lebih pernyataan SELECT (hanya nilai yang berbeda) |
| UNION ALL | Menggabungkan kumpulan hasil dari dua atau lebih pernyataan SELECT (memungkinkan nilai duplikat) |
| UNIQUE | Batasan yang memastikan bahwa semua nilai dalam kolom adalah unik |
| UPDATE | Memperbarui baris yang ada dalam tabel |
| VALUES | Menentukan nilai dari pernyataan INSERT INTO |
| VIEW | Membuat, memperbarui, atau menghapus tampilan |
| WHERE | Memfilter kumpulan hasil untuk menyertakan hanya catatan yang memenuhi kondisi tertentu |

1. **MySQL Functions**

* Fungsi MySQL String

|  |  |
| --- | --- |
| Fungsi | Penjelasan |
| ASCII | Mengembalikan nilai ASCII untuk karakter tertentu |
| CHAR\_LENGTH | Mengembalikan panjang string (dalam karakter) |
| CHARACTER\_LENGTH | Mengembalikan panjang string (dalam karakter) |
| CONCAT | Menambahkan dua atau lebih ekspresi bersama |
| CONCAT\_WS | Menambahkan dua atau lebih ekspresi bersama dengan pemisah |
| FIELD | Mengembalikan posisi indeks nilai dalam daftar nilai |
| FIND\_IN\_SET | Mengembalikan posisi string dalam daftar string |
| FORMAT | Memformat angka ke format seperti "#,###,###.##", dibulatkan ke sejumlah tempat desimal tertentu |
| INSERT | Menyisipkan string di dalam string pada posisi yang ditentukan dan untuk sejumlah karakter tertentu |
| INSTR | Mengembalikan posisi kemunculan pertama string di string lain |
| LCASE | Mengonversi string menjadi huruf kecil |
| LEFT | Ekstrak sejumlah karakter dari string (mulai dari kiri) |
| LENGTH | Mengembalikan panjang string (dalam byte) |
| LOCATE | Mengembalikan posisi kemunculan pertama substring dalam string |
| LOWER | Mengonversi string menjadi huruf kecil |
| LPAD | Kiri-bantalan string dengan string lain, dengan panjang tertentu |
| LTRIM | Menghapus spasi awal dari string |
| MID | Mengekstrak substring dari string (dimulai dari posisi apa pun) |
| POSITION | Mengembalikan posisi kemunculan pertama substring dalam string |
| REPEAT | Mengulangi string sebanyak yang ditentukan |
| REPLACE | Mengganti semua kemunculan substring dalam string, dengan substring baru |
| REVERSE | Membalikkan string dan mengembalikan hasilnya |
| RIGHT | Mengekstrak sejumlah karakter dari sebuah string (dimulai dari kanan) |
| RPAD | Menyambungkan senar dengan senar lain, dengan panjang tertentu |
| RTRIM | Menghapus spasi tambahan dari string |
| SPACE | Mengembalikan string dengan jumlah karakter spasi yang ditentukan |
| STRCMP | Membandingkan dua string |
| SUBSTR | Mengekstrak substring dari string (dimulai dari posisi apa pun) |
| SUBSTRING | Mengekstrak substring dari string (dimulai dari posisi apa pun) |
| SUBSTRING\_INDEX | Mengembalikan substring dari string sebelum jumlah pembatas yang ditentukan terjadi |
| TRIM | Menghapus spasi awal dan akhir dari string |
| UCASE | Mengonversi string menjadi huruf besar |
| UPPER | Mengonversi string menjadi huruf besar |

* **MySQL Numeric Functions**

|  |  |
| --- | --- |
| Fungsi | Penjelasan |
| ABS | Mengembalikan nilai absolut dari suatu angka |
| ACOS | Mengembalikan arc cosinus suatu angka |
| ASIN | Mengembalikan sinus busur angka |
| ATAN | Mengembalikan tangen busur dari satu atau dua angka |
| ATAN2 | Mengembalikan tangen busur dua angka |
| AVG | Mengembalikan nilai rata-rata ekspresi |
| CEIL | Mengembalikan nilai bilangan bulat terkecil yaitu >= ke angka |
| CEILING | Mengembalikan nilai bilangan bulat terkecil yaitu >= ke angka |
| COS | Mengembalikan kosinus angka |
| COT | Mengembalikan kotangen dari suatu angka |
| COUNT | Mengembalikan jumlah rekaman yang dikembalikan oleh kueri pemilihan |
| DEGREES | Mengonversi nilai dalam radian ke derajat |
| DIV | Digunakan untuk pembagian bilangan bulat |
| EXP | Mengembalikan e yang dipangkatkan ke angka tertentu |
| FLOOR | Mengembalikan nilai bilangan bulat terbesar yaitu <= ke angka |
| GREATEST | Mengembalikan nilai terbesar dari daftar argumen |
| LEAST | Mengembalikan nilai terkecil dari daftar argumen |
| LN | Mengembalikan logaritma natural dari suatu bilangan |
| LOG | Mengembalikan logaritma natural dari suatu bilangan, atau logaritma suatu bilangan ke basis yang ditentukan |
| LOG10 | Mengembalikan logaritma natural suatu bilangan ke basis 10 |
| LOG2 | Mengembalikan logaritma natural suatu bilangan ke basis 2 |
| MAX | Mengembalikan nilai maksimum dalam sekumpulan nilai |
| MIN | Mengembalikan nilai minimum dalam kumpulan nilai |
| MOD | Mengembalikan sisa angka dibagi dengan angka lain |
| PI | Mengembalikan nilai PI |
| POW | Mengembalikan nilai angka yang dipangkatkan dengan angka lain |
| POWER | Mengembalikan nilai angka yang dipangkatkan dengan angka lain |
| RADIANS | Mengubah nilai derajat menjadi radian |
| RAND | Mengembalikan nomor acak |
| ROUND | Membulatkan angka ke jumlah tempat desimal yang ditentukan |
| SIGN | Mengembalikan tanda angka |
| SIN | Mengembalikan sinus angka |
| SQRT | Mengembalikan akar kuadrat dari suatu angka |
| SUM | Menghitung jumlah dari sekumpulan nilai |
| TAN | Mengembalikan tangen angka |
| TRUNCATE | Memotong angka ke jumlah tempat desimal yang ditentukan |

* **Fungsi MySQL Date**

|  |  |
| --- | --- |
| Fungsi | Penjelasan |
| ADDDATE | Menambahkan interval waktu/tanggal ke tanggal dan kemudian mengembalikan tanggal |
| ADDTIME | Menambahkan interval waktu ke waktu/datetime dan kemudian mengembalikan waktu/datetime |
| CURDATE | Mengembalikan tanggal saat ini |
| CURRENT\_DATE | Mengembalikan tanggal saat ini |
| CURRENT\_TIME | Mengembalikan waktu saat ini |
| CURRENT\_TIMESTAMP | Mengembalikan tanggal dan waktu saat ini |
| CURTIME | Mengembalikan waktu saat ini |
| DATE | Mengekstrak bagian tanggal dari ekspresi datetime |
| DATEDIFF | Mengembalikan jumlah hari antara dua nilai tanggal |
| DATE\_ADD | Menambahkan interval waktu/tanggal ke tanggal dan kemudian mengembalikan tanggal |
| DATE\_FORMAT | Memformat tanggal |
| DATE\_SUB | Kurangi interval waktu/tanggal dari tanggal dan kemudian kembalikan tanggalnya |
| DAY | Mengembalikan hari dalam sebulan untuk tanggal tertentu |
| DAYNAME | Mengembalikan nama hari kerja untuk tanggal tertentu |
| DAYOFMONTH | Mengembalikan hari dalam sebulan untuk tanggal tertentu |
| DAYOFWEEK | Mengembalikan indeks hari kerja untuk tanggal tertentu |
| DAYOFYEAR | Mengembalikan hari dalam setahun untuk tanggal tertentu |
| EXTRACT | Mengekstrak bagian dari tanggal tertentu |
| FROM\_DAYS | Mengembalikan tanggal dari nilai tanggal numerik |
| HOUR | Mengembalikan bagian jam untuk tanggal tertentu |
| LAST\_DAY | Ekstrak hari terakhir bulan itu untuk tanggal tertentu |
| LOCALTIME | Mengembalikan tanggal dan waktu saat ini |
| LOCALTIMESTAMP | Mengembalikan tanggal dan waktu saat ini |
| MAKEDATE | Membuat dan mengembalikan tanggal berdasarkan nilai tahun dan jumlah hari |
| MAKETIME | Membuat dan mengembalikan waktu berdasarkan nilai jam, menit, dan detik |
| MICROSECOND | Mengembalikan bagian mikrodetik dari waktu/datetime |
| MINUTE | Mengembalikan bagian menit dari waktu/datetime |
| MONTH | Mengembalikan bagian bulan untuk tanggal tertentu |
| MONTHNAME | Mengembalikan nama bulan untuk tanggal tertentu |
| NOW | Mengembalikan tanggal dan waktu saat ini |
| PERIOD\_ADD | Menambahkan jumlah bulan tertentu ke suatu periode |
| PERIOD\_DIFF | Mengembalikan selisih antara dua periode |
| QUARTER | Mengembalikan kuartal tahun untuk nilai tanggal tertentu |
| SECOND | Mengembalikan detik bagian dari waktu/datetime |
| SEC\_TO\_TIME | Mengembalikan nilai waktu berdasarkan detik yang ditentukan |
| STR\_TO\_DATE | Mengembalikan tanggal berdasarkan string dan format |
| SUBDATE | Kurangi interval waktu/tanggal dari tanggal dan kemudian kembalikan tanggalnya |
| SUBTIME | Kurangi interval waktu dari datetime dan kemudian kembalikan waktu/datetime |
| SYSDATE | Mengembalikan tanggal dan waktu saat ini |
| TIME | Mengekstrak bagian waktu dari waktu/waktu tertentu |
| TIME\_FORMAT | Memformat waktu dengan format tertentu |
| TIME\_TO\_SEC | Mengubah nilai waktu menjadi detik |
| TIMEDIFF | Mengembalikan perbedaan antara dua ekspresi waktu/datetime |
| TIMESTAMP | Mengembalikan nilai datetime berdasarkan tanggal atau nilai datetime |
| TO\_DAYS | Mengembalikan jumlah hari antara tanggal dan tanggal "0000-00-00" |
| WEEK | Mengembalikan nomor minggu untuk tanggal tertentu |
| WEEKDAY | Mengembalikan nomor hari kerja untuk tanggal tertentu |
| WEEKOFYEAR | Mengembalikan nomor minggu untuk tanggal tertentu |
| YEAR | Mengembalikan bagian tahun untuk tanggal tertentu |
| YEARWEEK | Mengembalikan nomor tahun dan minggu untuk tanggal tertentu |

* **Fungsi MySQL Advanced**

|  |  |
| --- | --- |
| Fungsi | Penjelasan |
| BIN | Mengembalikan representasi biner dari suatu angka |
| BINARY | Mengonversi nilai menjadi string biner |
| CASE | Melewati kondisi dan mengembalikan nilai saat kondisi pertama terpenuhi |
| CAST | Mengonversi nilai (dari tipe apa pun) menjadi tipe data yang ditentukan |
| COALESCE | Mengembalikan nilai non-null pertama dalam daftar |
| CONNECTION\_ID | Mengembalikan ID koneksi unik untuk koneksi saat ini |
| CONV | Mengonversi angka dari satu sistem dasar numerik ke yang lain |
| CONVERT | Mengonversi nilai menjadi tipe data atau set karakter yang ditentukan |
| CURRENT\_USER | Mengembalikan nama pengguna dan nama host untuk akun MySQL yang digunakan server untuk mengotentikasi klien saat ini |
| DATABASE | Mengembalikan nama database saat ini |
| IF | Mengembalikan nilai jika kondisi TRUE, atau nilai lain jika kondisi FALSE |
| IFNULL | Kembalikan nilai yang ditentukan jika ekspresinya NULL, jika tidak, kembalikan ekspresi |
| ISNULL | Mengembalikan 1 atau 0 tergantung pada apakah suatu ekspresi NULL |
| LAST\_INSERT\_ID | Mengembalikan id AUTO\_INCREMENT dari baris terakhir yang telah dimasukkan atau diperbarui dalam tabel |
| NULLIF | Membandingkan dua ekspresi dan mengembalikan NULL jika keduanya sama. Jika tidak, ekspresi pertama dikembalikan |
| SESSION\_USER | Mengembalikan nama pengguna dan nama host MySQL saat ini |
| SYSTEM\_USER | Mengembalikan nama pengguna dan nama host MySQL saat ini |
| USER | Mengembalikan nama pengguna dan nama host MySQL saat ini |
| VERSION | Mengembalikan versi database MySQL saat ini |
| BIN | Mengembalikan representasi biner dari suatu angka |
| BINARY | Mengonversi nilai menjadi string biner |
| CASE | Melewati kondisi dan mengembalikan nilai saat kondisi pertama terpenuhi |
| CAST | Mengonversi nilai (dari tipe apa pun) menjadi tipe data yang ditentukan |
| COALESCE | Mengembalikan nilai non-null pertama dalam daftar |
| CONNECTION\_ID | Mengembalikan ID koneksi unik untuk koneksi saat ini |
| CONV | Mengonversi angka dari satu sistem dasar numerik ke yang lain |
| CONVERT | Mengonversi nilai menjadi tipe data atau set karakter yang ditentukan |
| CURRENT\_USER | Mengembalikan nama pengguna dan nama host untuk akun MySQL yang digunakan server untuk mengotentikasi klien saat ini |
| DATABASE | Mengembalikan nama database saat ini |
| IF | Mengembalikan nilai jika kondisi TRUE, atau nilai lain jika kondisi FALSE |
| IFNULL | Kembalikan nilai yang ditentukan jika ekspresinya NULL, jika tidak, kembalikan ekspresi |
| ISNULL | Mengembalikan 1 atau 0 tergantung pada apakah suatu ekspresi NULL |
| LAST\_INSERT\_ID | Mengembalikan id AUTO\_INCREMENT dari baris terakhir yang telah dimasukkan atau diperbarui dalam tabel |
| NULLIF | Membandingkan dua ekspresi dan mengembalikan NULL jika keduanya sama. Jika tidak, ekspresi pertama dikembalikan |
| SESSION\_USER | Mengembalikan nama pengguna dan nama host MySQL saat ini |
| SYSTEM\_USER | Mengembalikan nama pengguna dan nama host MySQL saat ini |
| USER | Mengembalikan nama pengguna dan nama host MySQL saat ini |
| VERSION | Mengembalikan versi database MySQL saat ini |
| BIN | Mengembalikan representasi biner dari suatu angka |
| BINARY | Mengonversi nilai menjadi string biner |
| CASE | Melewati kondisi dan mengembalikan nilai saat kondisi pertama terpenuhi |
| CAST | Mengonversi nilai (dari tipe apa pun) menjadi tipe data yang ditentukan |
| COALESCE | Mengembalikan nilai non-null pertama dalam daftar |
| CONNECTION\_ID | Mengembalikan ID koneksi unik untuk koneksi saat ini |
| CONV | Mengonversi angka dari satu sistem dasar numerik ke yang lain |
| CONVERT | Mengonversi nilai menjadi tipe data atau set karakter yang ditentukan |
| CURRENT\_USER | Mengembalikan nama pengguna dan nama host untuk akun MySQL yang digunakan server untuk mengotentikasi klien saat ini |
| DATABASE | Mengembalikan nama database saat ini |
| IF | Mengembalikan nilai jika kondisi TRUE, atau nilai lain jika kondisi FALSE |
| IFNULL | Kembalikan nilai yang ditentukan jika ekspresinya NULL, jika tidak, kembalikan ekspresi |
| ISNULL | Mengembalikan 1 atau 0 tergantung pada apakah suatu ekspresi NULL |
| LAST\_INSERT\_ID | Mengembalikan id AUTO\_INCREMENT dari baris terakhir yang telah dimasukkan atau diperbarui dalam tabel |
| NULLIF | Membandingkan dua ekspresi dan mengembalikan NULL jika keduanya sama. Jika tidak, ekspresi pertama dikembalikan |
| SESSION\_USER | Mengembalikan nama pengguna dan nama host MySQL saat ini |
| SYSTEM\_USER | Mengembalikan nama pengguna dan nama host MySQL saat ini |
| USER | Mengembalikan nama pengguna dan nama host MySQL saat ini |
| VERSION | Mengembalikan versi database MySQL saat ini |

1. **SQL Server Functions**

* **Fungsi SQL Server String**

|  |  |
| --- | --- |
| Fungsi | Penjelasan |
| ASCII | Mengembalikan nilai ASCII untuk karakter tertentu |
| CHAR | Mengembalikan karakter berdasarkan kode ASCII |
| CHARINDEX | Mengembalikan posisi substring dalam string |
| CONCAT | Menambahkan dua atau lebih string bersama-sama |
| Concat with + | Menambahkan dua atau lebih string bersama-sama |
| CONCAT\_WS | Menambahkan dua atau lebih string bersama-sama dengan pemisah |
| DATALENGTH | Mengembalikan jumlah byte yang digunakan untuk mewakili ekspresi |
| DIFFERENCE | Membandingkan dua nilai SOUNDEX, dan mengembalikan nilai integer |
| FORMAT | Memformat nilai dengan format yang ditentukan |
| LEFT | Ekstrak sejumlah karakter dari string (mulai dari kiri) |
| LEN | Mengembalikan panjang string |
| LOWER | Mengonversi string menjadi huruf kecil |
| LTRIM | Menghapus spasi awal dari string |
| NCHAR | Mengembalikan karakter Unicode berdasarkan kode angka |
| PATINDEX | Mengembalikan posisi pola dalam string |
| QUOTENAME | Mengembalikan string Unicode dengan pembatas yang ditambahkan untuk membuat string menjadi pengidentifikasi delimited SQL Server yang valid |
| REPLACE | Mengganti semua kemunculan substring dalam string, dengan substring baru |
| REPLICATE | Mengulangi string beberapa kali |
| REVERSE | Membalikkan string dan mengembalikan hasilnya |
| RIGHT | Mengekstrak sejumlah karakter dari sebuah string (dimulai dari kanan) |
| RTRIM | Menghapus spasi tambahan dari string |
| SOUNDEX | Mengembalikan kode empat karakter untuk mengevaluasi kesamaan dua string |
| SPACE | Mengembalikan string dengan jumlah karakter spasi yang ditentukan |
| STR | Mengembalikan angka sebagai string |
| STUFF | Menghapus bagian dari string dan kemudian memasukkan bagian lain ke dalam string, mulai dari posisi tertentu |
| SUBSTRING | Ekstrak beberapa karakter dari string |
| TRANSLATE | Mengembalikan string dari argumen pertama setelah karakter yang ditentukan dalam argumen kedua diterjemahkan ke dalam karakter yang ditentukan dalam argumen ketiga. |
| TRIM | Menghapus spasi awal dan akhir (atau karakter tertentu lainnya) dari string |
| UNICODE | Mengembalikan nilai Unicode untuk karakter pertama dari ekspresi input |
| UPPER | Mengonversi string menjadi huruf besar |

* **Fungsi SQL Server Math/Numeric**

|  |  |
| --- | --- |
| Fungsi | Penjelasan |
| ABS | Mengembalikan nilai absolut dari suatu angka |
| ACOS | Mengembalikan arc cosinus suatu angka |
| ASIN | Mengembalikan sinus busur angka |
| ATAN | Mengembalikan tangen busur angka |
| ATN2 | Mengembalikan tangen busur dua angka |
| AVG | Mengembalikan nilai rata-rata ekspresi |
| CEILING | Mengembalikan nilai bilangan bulat terkecil yaitu >= angka |
| COUNT | Mengembalikan jumlah rekaman yang dikembalikan oleh kueri pemilihan |
| COS | Mengembalikan kosinus angka |
| COT | Mengembalikan kotangen dari suatu angka |
| DEGREES | Mengonversi nilai dalam radian ke derajat |
| EXP | Mengembalikan e yang dipangkatkan ke angka tertentu |
| FLOOR | Mengembalikan nilai bilangan bulat terbesar yaitu <= ke angka |
| LOG | Mengembalikan logaritma natural dari suatu bilangan, atau logaritma suatu bilangan ke basis yang ditentukan |
| LOG10 | Mengembalikan logaritma natural suatu bilangan ke basis 10 |
| MAX | Mengembalikan nilai maksimum dalam sekumpulan nilai |
| MIN | Mengembalikan nilai minimum dalam kumpulan nilai |
| PI | Mengembalikan nilai PI |
| POWER | Mengembalikan nilai angka yang dipangkatkan dengan angka lain |
| RADIANS | Mengubah nilai derajat menjadi radian |
| RAND | Mengembalikan nomor acak |
| ROUND | Membulatkan angka ke jumlah tempat desimal yang ditentukan |
| SIGN | Mengembalikan tanda angka |
| SIN | Mengembalikan sinus angka |
| SQRT | Mengembalikan akar kuadrat dari suatu angka |
| SQUARE | Mengembalikan kuadrat dari angka |
| SUM | Menghitung jumlah dari sekumpulan nilai |
| TAN | Mengembalikan tangen angka |

* **Fungsi SQL Server Date**

|  |  |
| --- | --- |
| Fungsi | Penjelasan |
| CURRENT\_TIMESTAMP | Mengembalikan tanggal dan waktu saat ini |
| DATEADD | Menambahkan interval waktu/tanggal ke tanggal dan kemudian mengembalikan tanggal |
| DATEDIFF | Mengembalikan selisih antara dua tanggal |
| DATEFROMPARTS | Mengembalikan tanggal dari bagian tertentu (tahun, bulan, dan nilai hari) |
| DATENAME | Mengembalikan bagian tertentu dari tanggal (sebagai string) |
| DATEPART | Mengembalikan bagian tertentu dari tanggal (sebagai bilangan bulat) |
| DAY | Mengembalikan hari dalam sebulan untuk tanggal yang ditentukan |
| GETDATE | Mengembalikan tanggal dan waktu sistem database saat ini |
| GETUTCDATE | Mengembalikan tanggal dan waktu UTC sistem database saat ini |
| ISDATE | Memeriksa ekspresi dan mengembalikan 1 jika itu adalah tanggal yang valid, jika tidak 0 |
| MONTH | Mengembalikan bagian bulan untuk tanggal tertentu (angka dari 1 hingga 12) |
| SYSDATETIME | Mengembalikan tanggal dan waktu SQL Server |
| YEAR | Mengembalikan bagian tahun untuk tanggal tertentu |

* **Fungsi SQL Server Date**

|  |  |
| --- | --- |
| Fungsi | Penjelasan |
| CAST | Mengonversi nilai (dari tipe apa pun) menjadi tipe data yang ditentukan |
| COALESCE | Mengembalikan nilai non-null pertama dalam daftar |
| CONVERT | Mengonversi nilai (dari tipe apa pun) menjadi tipe data yang ditentukan |
| CURRENT\_USER | Mengembalikan nama pengguna saat ini di database SQL Server |
| IIF | Mengembalikan nilai jika kondisi TRUE, atau nilai lain jika kondisi FALSE |
| ISNULL | Kembalikan nilai yang ditentukan jika ekspresinya NULL, jika tidak, kembalikan ekspresi |
| ISNUMERIC | Menguji apakah suatu ekspresi numerik |
| NULLIF | Mengembalikan NULL jika dua ekspresi sama |
| SESSION\_USER | Mengembalikan nama pengguna saat ini di database SQL Server |
| SESSIONPROPERTY | Mengembalikan pengaturan sesi untuk opsi tertentu |
| SYSTEM\_USER | Mengembalikan nama login untuk pengguna saat ini |
| USER\_NAME | Mengembalikan nama pengguna database berdasarkan id yang ditentukan |

1. **MS Access Functions**

* **Fungsi MS Access String**

|  |  |
| --- | --- |
| Fungsi | Penjelasan |
| Asc | Mengembalikan nilai ASCII untuk karakter tertentu |
| Chr | Mengembalikan karakter untuk kode nomor ASCII yang ditentukan |
| Concat with & | Menambahkan dua atau lebih string bersama-sama |
| CurDir | Mengembalikan path lengkap untuk drive tertentu |
| Format | Memformat nilai dengan format yang ditentukan |
| InStr | Mendapat posisi kemunculan pertama string di yang lain |
| InstrRev | Mendapat posisi kemunculan pertama string di string lain, dari akhir string |
| LCase | Mengonversi string menjadi huruf kecil |
| Left | Ekstrak sejumlah karakter dari string (mulai dari kiri) |
| Len | Mengembalikan panjang string |
| LTrim | Menghapus spasi awal dari string |
| Mid | Mengekstrak beberapa karakter dari string (dimulai dari posisi apa pun) |
| Replace | Menggantikan substring dalam string, dengan substring lain, beberapa kali tertentu |
| Right | Mengekstrak sejumlah karakter dari sebuah string (dimulai dari kanan) |
| RTrim | Menghapus spasi tambahan dari string |
| Space | Mengembalikan string dengan jumlah karakter spasi yang ditentukan |
| Split | Membagi string menjadi array substring |
| Str | Mengembalikan angka sebagai string |
| StrComp | Membandingkan dua string |
| StrConv | Mengembalikan string yang dikonversi |
| StrReverse | Membalikkan string dan mengembalikan hasilnya |
| Trim | Menghapus spasi awal dan akhir dari string |
| UCase | Mengonversi string menjadi huruf besa |

* **Fungsi MS Access Numeric**

|  |  |
| --- | --- |
| Fungsi | Penjelasan |
| Abs | Mengembalikan nilai absolut dari suatu angka |
| Atn | Mengembalikan tangen busur angka |
| Avg | Mengembalikan nilai rata-rata ekspresi |
| Cos | Mengembalikan kosinus suatu sudut |
| Count | Mengembalikan jumlah rekaman yang dikembalikan oleh kueri pemilihan |
| Exp | Mengembalikan e yang dipangkatkan ke angka tertentu |
| Fix | Mengembalikan bagian bilangan bulat dari suatu angka |
| Format | Memformat nilai numerik dengan format yang ditentukan |
| Int | Mengembalikan bagian bilangan bulat dari suatu angka |
| Max | Mengembalikan nilai maksimum dalam sekumpulan nilai |
| Min | Mengembalikan nilai minimum dalam kumpulan nilai |
| Randomize | Inisialisasi generator nomor acak (digunakan oleh Rnd()) dengan seed |
| Rnd | Mengembalikan nomor acak |
| Round | Membulatkan angka ke jumlah tempat desimal yang ditentukan |
| Sgn | Mengembalikan tanda angka |
| Sqr | Mengembalikan akar kuadrat dari suatu angka |
| Sum | Menghitung jumlah dari sekumpulan nilai |
| Val | Membaca string dan mengembalikan angka yang ditemukan dalam string |

* **Fungsi MS Access Date**

|  |  |
| --- | --- |
| Fungsi | Penjelasan |
| Date | Mengembalikan tanggal sistem saat ini |
| DateAdd | Menambahkan interval waktu/tanggal ke tanggal dan kemudian mengembalikan tanggal |
| DateDiff | Mengembalikan selisih antara dua tanggal |
| DatePart | Mengembalikan bagian tertentu dari tanggal (sebagai bilangan bulat) |
| DateSerial | Mengembalikan tanggal dari bagian tertentu (tahun, bulan, dan nilai hari) |
| DateValue | Mengembalikan tanggal berdasarkan string |
| Day | Mengembalikan hari dalam sebulan untuk tanggal tertentu |
| Format | Memformat nilai tanggal dengan format yang ditentukan |
| Hour | Mengembalikan jam bagian dari waktu/datetime |
| Minute | Mengembalikan bagian menit dari waktu/datetime |
| Month | Mengembalikan bagian bulan dari tanggal tertentu |
| MonthName | Mengembalikan nama bulan berdasarkan angka |
| Now | Mengembalikan tanggal dan waktu saat ini berdasarkan tanggal dan waktu sistem komputer |
| Second | Mengembalikan detik bagian dari waktu/datetime |
| Time | Mengembalikan waktu sistem saat ini |
| TimeSerial | Mengembalikan waktu dari bagian yang ditentukan (nilai jam, menit, dan detik) |
| TimeValue | Mengembalikan waktu berdasarkan string |
| Weekday | Mengembalikan nomor hari kerja untuk tanggal tertentu |
| WeekdayName | Mengembalikan nama hari kerja berdasarkan angka |
| Year | Mengembalikan bagian tahun dari tanggal tertentu |

* **Beberapa Fungsi lain MS Access**

|  |  |
| --- | --- |
| Fungsi | Penjelasan |
| CurrentUser | Mengembalikan nama pengguna database saat ini |
| Environ | Mengembalikan string yang berisi nilai variabel lingkungan sistem operasi |
| IsDate | Memeriksa apakah ekspresi dapat dikonversi menjadi tanggal |
| IsNull | Memeriksa apakah ekspresi berisi Null (tidak ada data) |
| IsNumeric | Memeriksa apakah ekspresi adalah angka yang valid |

1. **SQL Quick Ref**

|  |  |
| --- | --- |
| SQL Statement | Syntax |
| AND / OR | SELECT column\_name(s) FROM table\_name WHERE condition AND|OR condition |
| ALTER TABLE | ALTER TABLE table\_name ADD column\_name datatype  or  ALTER TABLE table\_name DROP COLUMN column\_name |
| AS (alias) | SELECT column\_name AS column\_alias FROM table\_name  or  SELECT column\_name FROM table\_name  AS table\_alias |
| BETWEEN | ELECT column\_name(s)  FROM table\_name  WHERE column\_name  BETWEEN value1 AND value2 |
| CREATE DATABASE | CREATE DATABASE database\_name |
| CREATE TABLE | CREATE TABLE table\_name  (  column\_name1 data\_type,  column\_name2 data\_type,  column\_name3 data\_type,  ...  ) |
| CREATE INDEX | CREATE INDEX index\_name  ON table\_name (column\_name)  or  CREATE UNIQUE INDEX index\_name  ON table\_name (column\_name) |
| CREATE VIEW | CREATE VIEW view\_name AS  SELECT column\_name(s)  FROM table\_name  WHERE condition |
| DELETE | DELETE FROM table\_name  WHERE some\_column=some\_value  or  DELETE FROM table\_name  (Note: Deletes the entire table!!)  DELETE \* FROM table\_name  (Note: Deletes the entire table!!) |
| DROP DATABASE | DROP DATABASE database\_name |
| DROP INDEX | DROP INDEX table\_name.index\_name (SQL Server)  DROP INDEX index\_name ON table\_name (MS Access)  DROP INDEX index\_name (DB2/Oracle)  ALTER TABLE table\_name  DROP INDEX index\_name (MySQL) |
| DROP TABLE | DROP TABLE table\_name |
| EXISTS | IF EXISTS (SELECT \* FROM table\_name WHERE id = ?)  BEGIN  --do what needs to be done if exists  END  ELSE  BEGIN  --do what needs to be done if not  END |
| GROUP BY | SELECT column\_name, aggregate\_function(column\_name)  FROM table\_name  WHERE column\_name operator value  GROUP BY column\_name |
| HAVING | SELECT column\_name, aggregate\_function(column\_name)  FROM table\_name  WHERE column\_name operator value  GROUP BY column\_name  HAVING aggregate\_function(column\_name) operator value |
| IN | SELECT column\_name(s)  FROM table\_name  WHERE column\_name  IN (value1,value2,..) |
| INSERT INTO | INSERT INTO table\_name  VALUES (value1, value2, value3,....)  or  INSERT INTO table\_name  (column1, column2, column3,...)  VALUES (value1, value2, value3,....) |
| INNER JOIN | SELECT column\_name(s)  FROM table\_name1  INNER JOIN table\_name2  ON table\_name1.column\_name=table\_name2.column\_name |
| LEFT JOIN | SELECT column\_name(s)  FROM table\_name1  LEFT JOIN table\_name2  ON table\_name1.column\_name=table\_name2.column\_name |
| RIGHT JOIN | SELECT column\_name(s)  FROM table\_name1  RIGHT JOIN table\_name2  ON table\_name1.column\_name=table\_name2.column\_name |
| FULL JOIN | SELECT column\_name(s) FROM table\_name1 FULL JOIN table\_name2 ON table\_name1.column\_name=table\_name2.column\_name |
| LIKE | SELECT column\_name(s) FROM table\_name WHERE column\_name LIKE pattern |
| ORDER BY | SELECT column\_name(s)  FROM table\_name  ORDER BY column\_name [ASC|DESC] |
| SELECT | SELECT column\_name(s)  FROM table\_name |
| SELECT \* | SELECT \*  FROM table\_name |
| SELECT DISTINCT | SELECT DISTINCT column\_name(s)  FROM table\_name |
| SELECT INTO | SELECT \* INTO new\_table\_name [IN externaldatabase] FROM old\_table\_name  or  SELECT column\_name(s) INTO new\_table\_name [IN externaldatabase] FROM old\_table\_name |
| SELECT TOP | SELECT TOP number|percent column\_name(s)  FROM table\_name |
| TRUNCATE TABLE | TRUNCATE TABLE table\_name |
| UNION | SELECT column\_name(s) FROM table\_name1  UNION  SELECT column\_name(s) FROM table\_name2 |
| UNION ALL | SELECT column\_name(s) FROM table\_name1  UNION ALL  SELECT column\_name(s) FROM table\_name2 |
| UPDATE | UPDATE table\_name  SET column1=value, column2=value,...  WHERE some\_column=some\_value |
| WHERE | SELECT column\_name(s)  FROM table\_name  WHERE column\_name operator value |