**SQL Tutorial**

1. **Pengertian SQL**

SQL (Structured Query Language) adalah sebuah bahasa yang dipergunakan untuk mengakses data dalam basis data relasional. Bahasa ini secara de facto merupakan bahasa standar yang digunakan dalam manajemen basis data relasional. Saat ini hampir semua server basis data yang ada mendukung bahasa ini untuk melakukan manajemen datanya. Sejarah SQL dimulai dari artikel seorang peneliti dari IBM bernama EF Codd yang membahas tentang ide pembuatan basis data relasional pada bulan Juni 1970. Artikel ini juga membahas kemungkinan pembuatan bahasa standar untuk mengakses data dalam basis data tersebut. Bahasa tersebut kemudian diberi nama SEQUEL (Structured English Query Language). Setelah terbitnya artikel tersebut, IBM mengadakan proyek pembuatan basis data relasional berbasis bahasa SEQUEL. Akan tetapi, karena permasalahan hukum mengenai penamaan SEQUEL, IBM pun mengubahnya menjadi SQL. Implementasi basis data relasional dikenal dengan System/R. Di akhir tahun 1970-an, muncul perusahaan bernama Oracle yang membuat server basis data populer yang bernama sama dengan nama perusahaannya. Dengan naiknya kepopuleran Oracle, maka SQL juga ikut populer sehingga saat ini menjadi standar de facto bahasa dalam manajemen basis data.

1. **Pemakaian dasar**

Secara umum, SQL terdiri dari dua bahasa, yaitu Data Definition Language (DDL ) dan Data Manipulation Language (DML ). Implementasi DDL dan DML berbeda untuk tiap sistem manajemen basis data (SMBD)[1], namun secara umum implementasi tiap bahasa ini memiliki bentuk standar yang ditetapkan ANSI. Artikel ini akan menggunakan bentuk paling umum yang dapat digunakan pada kebanyakan SMBD.

Tujuan dari pemakaian SQL dalam menyelesaikan tugas-tugas yang berkaitan dengan database, diantaranya :

* Memanggil data dari satu table atau lebih
* Memanipulasi data
* Mendapatkan ringkasan informasi
* Membuat, memodifikasi atau menghapus table
* Membuat atau menghapus index

1. **Fungsi (Function ) SQL**

MS SQL Server memiliki beberapa fungsi untuk manipulasi data antara lain :

* Fungsi string atau karakter
* Fungsi tanggal
* Fungsi matematika
* Fungsi system atau fungsi scalar
* Fungsi User Defined
* Fungsi tambahan yang dapat digunakan untuk mengubah (konversi ) dari data yang satu ke data yang lain.

1. **Komponen SQL**
2. Data Definition Language (DDL ) Digunakan untuk membuat dan memodifikasi obyek database. Pernyataan DDL terdiri atas : CREATE, DROP, ALTER, RENAME dan TRUNCATE.
3. Data Manipulation Language (DML ) Digunakan untuk memanipulasi data dalam database. Pernyataan DML meliputi :

* SELECT
* INSERT
* UPDATE dan,
* DELETE

1. Data Control Language (DCL )

Digunakan untuk memberikan hak dalam melakukan pekerjaan yang terkait dengan database dan hak untuk memanipulasi obyek dalam database. Pernyataan DCL meliputi : GRANT dan REVOKE.

**SQL HOME**

**SQL Intro**

**SQL Syntax**

Ketika Anda ingin melakukan beberapa operasi pada data dalam database, maka Anda harus menulis kueri dalam sintaks SQL yang telah ditentukan.

Sintaks bahasa kueri terstruktur adalah seperangkat aturan dan pedoman unik, yang tidak peka huruf besar-kecil. Sintaksnya ditentukan dan dipelihara oleh standar ISO dan ANSI.

Berikut adalah beberapa poin terpenting tentang sintaks SQL yang harus diingat:

* Kita dapat menulis kata kunci SQL dalam huruf besar dan kecil, tetapi menulis kata kunci SQL dalam huruf besar meningkatkan keterbacaan kueri SQL.
* Pernyataan atau sintaks SQL bergantung pada baris teks. Kita dapat menempatkan satu pernyataan SQL pada satu atau beberapa baris teks.
* Kita dapat melakukan sebagian besar tindakan dalam database dengan pernyataan SQL.
* Sintaks SQL bergantung pada aljabar relasional dan kalkulus relasional tuple.

Pernyataan SQL

Pernyataan SQL memberi tahu database operasi apa yang ingin Anda lakukan pada data terstruktur dan informasi apa yang ingin Anda akses dari database.

Pernyataan SQL sangat sederhana dan mudah digunakan dan dipahami. Mereka seperti bahasa Inggris biasa tetapi dengan sintaks tertentu.

Contoh sederhana dari pernyataan SQL:

PILIH "column\_name" DARI "table\_name" ;

Setiap pernyataan SQL dimulai dengan salah satu kata kunci SQL dan diakhiri dengan titik koma (;). Titik koma digunakan dalam SQL untuk memisahkan beberapa pernyataan Sql yang akan dieksekusi dalam panggilan yang sama. Dalam tutorial SQL ini, kita akan menggunakan titik koma (;) di akhir setiap query atau pernyataan SQL.

**SQL Select**

Perintah SQL yang paling umum digunakan adalah pernyataan SELECT . Hal ini digunakan untuk query database dan mengambil data yang dipilih yang mengikuti kondisi yang kita inginkan.

Dengan kata sederhana, kita dapat mengatakan bahwa pernyataan pilih digunakan untuk meminta atau mengambil data dari tabel dalam database.

**SELECT** expressions

**FROM** tables

**WHERE** conditions;

**SQL Select Distinct**

Perintah SQL DISTINCT digunakan dengan kata kunci SELECT untuk mengambil hanya data yang berbeda atau unik.

Dalam sebuah tabel, mungkin ada kemungkinan adanya nilai duplikat dan terkadang kita hanya ingin mengambil nilai unik. Dalam skenario seperti itu, pernyataan SQL SELECT DISTINCT digunakan.

**PILIH BERBEDA** column\_name ,column\_name

**DARI**  nama\_tabel;

Mari kita coba memahaminya dengan tabel yang diberikan di bawah ini:

|  |  |  |  |
| --- | --- | --- | --- |
| **Nama siswa** | **Jenis kelamin** | **Nomor handphone** | **HOME\_TOWN** |
| Rahul Ojha | Pria | 7503896532 | Beruntung |
| Disha Rai | Perempuan | 9270568893 | Varanasi |
| Sonoo Jaiswal | Pria | 9990449935 | Beruntung |

Berikut adalah tabel siswa dari mana kami ingin mengambil informasi yang berbeda Misalnya: kota asal yang berbeda.

**SELECT DISTINCT** home\_town

**DARI** siswa

Sekarang, itu akan mengembalikan dua baris.

|  |
| --- |
| **HOME\_TOWN** |
| Beruntung |
| Varanasi |

**SQL Where**

SQL Klausa WHERE digunakan untuk menentukan kondisi saat mengambil data dari satu tabel atau dengan bergabung dengan beberapa tabel. Jika kondisi yang diberikan terpenuhi, maka hanya itu yang mengembalikan nilai tertentu dari tabel. Anda harus menggunakan klausa WHERE untuk memfilter catatan dan hanya mengambil catatan yang diperlukan.

Klausa WHERE tidak hanya digunakan dalam pernyataan SELECT, tetapi juga digunakan dalam pernyataan UPDATE, DELETE, dll.

SELECT column1, column2, columnN

FROM table\_name

WHERE [condition]

Anda dapat menentukan kondisi menggunakan perbandingan atau operator logika seperti >, <, =, LIKE, NOT , dll. Contoh berikut akan memperjelas konsep ini.

Contoh :

Pertimbangkan tabel PELANGGAN yang memiliki catatan berikut:

+----+----------+-----+-----------+----------+

| ID | NAME | AGE | ADDRESS | SALARY |

+----+----------+-----+-----------+----------+

| 1 | Ramesh | 32 | Ahmedabad | 2000.00 |

| 2 | Khilan | 25 | Delhi | 1500.00 |

| 3 | kaushik | 23 | Kota | 2000.00 |

| 4 | Chaitali | 25 | Mumbai | 6500.00 |

| 5 | Hardik | 27 | Bhopal | 8500.00 |

| 6 | Komal | 22 | MP | 4500.00 |

| 7 | Muffy | 24 | Indore | 10000.00 |

+----+----------+-----+-----------+----------+

Kode berikut adalah contoh yang akan mengambil bidang ID, Nama dan Gaji dari tabel PELANGGAN, di mana gaji lebih besar dari 2000

SQL> SELECT ID, NAME, SALARY

FROM CUSTOMERS

WHERE SALARY > 2000;

Ini akan menghasilkan hasil berikut

+----+----------+----------+

| ID | NAME | SALARY |

+----+----------+----------+

| 1 | Chaitali | 6500.00 |

| 2 | Hardik | 8500.00 |

| 3 | Komal | 4500.00 |

| 4 | Muffy | 10000.00 |

+----+----------+----------+

Kueri berikut adalah contoh, yang akan mengambil bidang ID, Nama, dan Gaji dari tabel PELANGGAN untuk pelanggan dengan nama Hardik .

Di sini, penting untuk dicatat bahwa semua string harus diberikan di dalam tanda kutip tunggal (''). Padahal, nilai numerik harus diberikan tanpa tanda kutip seperti pada contoh di atas.

SQL> SELECT ID, NAME, SALARY

FROM CUSTOMERS

WHERE NAME = 'Hardik';

Ini akan menghasilkan hasil berikut

+----+----------+----------+

| ID | NAME | SALARY |

+----+----------+----------+

| 1 | Hardik | 8500.00 |

+----+----------+----------+

**SQL And, Or, Not**

The WHERE klausa dapat dikombinasikan dengan AND, OR, dan NOT operator.

The ANDdan ORoperator yang digunakan untuk catatan filter berdasarkan lebih dari satu syarat:

The AND menampilkan Operator catatan jika semua kondisi dipisahkan oleh AND BENAR.

The OR menampilkan Operator catatan jika salah satu kondisi yang dipisahkan oleh OR BENAR.

The NOT menampilkan Operator catatan jika kondisi (s) adalah TIDAK BENAR.

AND Syntax

SELECT *column1*, *column2, ...*  
FROM *table\_name*  
WHERE *condition1* AND *condition2* AND *condition3 ...*;

OR Syntax

SELECT *column1*, *column2, ...*  
FROM *table\_name*  
WHERE *condition1* OR *condition2* OR *condition3 ...*;

NOT Syntax

SELECT *column1*, *column2, ...*  
FROM *table\_name*  
WHERE NOT *condition*;

**SQL Order By**

Pernyataan ini digunakan untuk mengurutkan data dalam urutan menaik. Jika Anda melewatkan atribut ASC, kueri SQL ORDER BY mengambil urutan menaik secara default.

Contoh Supplier

**SELECT** supplier\_city

**FROM** suppliers

**WHERE** supplier\_name = 'IBM'

**ORDER** **BY** supplier\_city;

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Indo** | **NAMA** | **USIA** | **ALAMAT** | **GAJI** |
| 1 | himani gupta | 21 | Modinagar | 22000 |
| 2 | Shiva tiwari | 22 | Bhopal | 21000 |
| 3 | Ajeet bhargav | 45 | meerut | 65000 |
| 4 | Ritesh yadav | 36 | Azamgarh | 26000 |
| 5 | Balwant singh | 45 | Varanasi | 36000 |
| 6 | Mahesh sharma | 26 | Mathura | 22000 |

Mari kita ambil tabel PELANGGAN yang memiliki catatan berikut:

Ini adalah contoh untuk mengurutkan hasil dalam urutan menaik berdasarkan NAME dan SALARY.

**SELECT** \* **FROM** CUSTOMERS

**ORDER** **BY** **NAME**, SALARY;

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Indo** | **NAMA** | **USIA** | **ALAMAT** | **GAJI** |
| 1 | Ajeet bhargav | 45 | meerut | 65000 |
| 2 | Balwant singh | 45 | Varanasi | 36000 |
| 3 | himani gupta | 21 | Modinagar | 22000 |
| 4 | Mahesh sharma | 26 | Mathura | 22000 |
| 5 | Ritesh yadav | 36 | Azamgarh | 26000 |
| 6 | Shiva tiwari | 22 | Bhopal | 21000 |

Ini akan menghasilkan hasil berikut.

**SQL Insert Into**

SQL INSERT INTO SELECT Syntax

**INSERT** **INTO** table\_name

[(column1, column2, .... **column**)]

**SELECT** column1, column2, .... **Column**

**FROM** table\_name [**WHERE** condition];

**SQL Null Values**

Pertama-tama kita harus tahu bahwa apa itu nilai nol? Nilai null digunakan untuk mewakili data yang tidak diketahui yang hilang.

Mungkin ada dua kondisi:

1. Di mana SQL adalah NULL
2. Di mana SQL BUKAN NULL

Jika dalam tabel, kolom adalah opsional, sangat mudah untuk menyisipkan data di kolom atau memperbarui catatan yang ada tanpa menambahkan nilai di kolom ini. Ini berarti bahwa bidang tersebut memiliki nilai nol.

Di mana SQL adalah NULL:

cara memilih catatan dengan nilai nol saja? (di kolom tanda)

|  |  |  |
| --- | --- | --- |
| **NAMA BAPAK** | **NAMA** | **TANDA** |
| TYAGI | SEEMA |  |
| SINGH | RAMAN | 5.5 |
| SHARMA | AMAR |  |
| JAISWAL | VICKY | 6.2 |

Berikut adalah contoh tabel siswa:

kueri untuk mendapatkan semua catatan dengan tanda NULL:

**SELECT** SIR\_NAME, **NAME**, MARKS **FROM** STUDENTS

**WHERE** MARKS **IS** NULL

It will return the following records:

|  |  |  |
| --- | --- | --- |
| **SIR\_NAME** | **NAME** | **MARKS** |
| SHARMA | AMAR |  |
| TYAGI | SEEMA |  |

**SQL Update**

cara memperbarui bidang tanggal dan waktu dalam SQL?

Jika ingin memperbarui bidang tanggal & waktu di SQL, kita harus menggunakan kueri berikut.

mari kita lihat sintaks tanggal pembaruan sql.

**UPDATE** **table**

**SET** Column\_Name = 'YYYY-MM-DD HH:MM:SS'

**WHERE** Id = value

Pertama kami mengambil tabel di mana kami ingin memperbarui bidang tanggal dan waktu.

Jika Anda ingin mengubah baris pertama yang idnya 1 maka Anda harus menulis sintaks berikut:

**UPDATE** **table**

**SET** EndDate = '2014-03-16 00:00:00.000'

**WHERE** Id = 1

**SQL Delete**

Contoh SQL DELETE

Mari kita ambil sebuah tabel, bernama tabel "EMPLOYEE".

|  |  |  |  |
| --- | --- | --- | --- |
| **ID** | **EMP\_NAME** | **CITY** | **SALARY** |
| 101 | Adarsh Singh | Obra | **20000** |
| 102 | Sanjay Singh | Meerut | **21000** |
| 103 | Priyanka Sharma | Raipur | **25000** |
| 104 | Esha Singhal | Delhi | **26000** |

**SQL Select Top**

Pernyataan SELECT TOP dalam SQL menunjukkan jumlah record atau baris yang terbatas dari tabel database. Klausa TOP dalam pernyataan menentukan berapa banyak baris yang dikembalikan.

Ini menunjukkan jumlah N baris teratas dari tabel di output. Klausa ini digunakan ketika ada ribuan record yang disimpan dalam tabel database.

Mari kita ambil contoh sederhana: Jika tabel Siswa memiliki sejumlah besar data tentang siswa, pernyataan pilih TOP menentukan berapa banyak data siswa yang akan diambil dari tabel yang diberikan.

Sintaks TOP Clause dalam SQL

PILIH nomor TOP | persen column\_Name1, column\_Name2, ....., column\_NameN FROM table\_name WHERE [Kondisi] ;

Dalam sintaks, **nomor** menunjukkan jumlah baris yang ditampilkan dari atas dalam output. column\_Name menunjukkan kolom yang recordnya ingin kita tampilkan di output. Kita juga dapat menentukan kondisi menggunakan klausa WHERE.

**SQL Min and Max**

Fungsi SQL MIN() dan MAX()

The MIN() fungsi mengembalikan nilai terkecil dari kolom yang dipilih.

The MAX() fungsi mengembalikan nilai terbesar dari kolom yang dipilih.

MIN() Contoh

Pernyataan SQL berikut menemukan harga produk termurah:

Contoh

SELECT MIN(Price) AS SmallestPrice  
FROM Products;

MAX() Contoh

Pernyataan SQL berikut menemukan harga produk paling mahal:

Contoh

SELECT MAX(Price) AS LargestPrice  
FROM Products;

**SQL Count, Avg, Sum**

Fungsi SQL COUNT, SUM, AVG

1. COUNT, SUM, dan AVG adalah fungsi agregat .
2. COUNT mengembalikan hitungan jumlah nilai data.
3. SUM mengembalikan jumlah nilai data.
4. AVG mengembalikan rata-rata nilai data.

Contoh

Masalah : Dapatkan jumlah total produk.

**SELECT COUNT(Id) AS 'Product Count'**

**FROM Product**

**SQL Like**

SQL LIKE SEPERTI Wildcard

1. WHERE LIKE menentukan apakah string karakter cocok dengan suatu pola .
2. WHERE LIKE mendukung dua opsi kecocokan wildcard: % dan \_.
3. Gunakan WHERE LIKE ketika hanya sebagian dari nilai teks yang diketahui.

Contoh :

**SELECT \***

**FROM Product**

**WHERE Package LIKE '%jars%'**

**SQL Wildcards**

Karakter wildcard digunakan untuk menggantikan satu atau lebih karakter dalam sebuah string.

Karakter wildcard digunakan dengan operator. The operator yang digunakan dalam klausul untuk mencari pola yang ditentukan dalam kolom. LIKELIKE WHERE

Karakter Wildcard di MS Access

|  |  |  |
| --- | --- | --- |
| **Symbol** | **Description** | **Example** |
| \* | Represents zero or more characters | bl\* finds bl, black, blue, and blob |
| ? | Represents a single character | h?t finds hot, hat, and hit |
| [] | Represents any single character within the brackets | h[oa]t finds hot and hat, but not hit |
| ! | Represents any character not in the brackets | h[!oa]t finds hit, but not hot and hat |
| - | Represents any single character within the specified range | c[a-b]t finds cat and cbt |
| # | Represents any single numeric character | 2#5 finds 205, 215, 225, 235, 245, 255, 265, 275, 285, and 295 |

Karakter Wildcard di SQL Server

|  |  |  |
| --- | --- | --- |
| **Symbol** | **Description** | **Example** |
| % | Represents zero or more characters | bl% finds bl, black, blue, and blob |
| \_ | Represents a single character | h\_t finds hot, hat, and hit |
| [] | Represents any single character within the brackets | h[oa]t finds hot and hat, but not hit |
| ^ | Represents any character not in the brackets | h[^oa]t finds hit, but not hot and hat |
| - | Represents any single character within the specified range | c[a-b]t finds cat and cbt |

**SQL In**

SQL IN DI Subquery

1. WHERE IN mengembalikan nilai yang cocok dengan nilai dalam daftar .
2. Daftar ini di-hardcode atau dibuat oleh subquery.
3. WHERE IN adalah singkatan untuk beberapa kondisi OR.

Contoh

Masalah: Daftar semua pelanggan dari London atau Paris.

**SELECT \***

**FROM Customer**

**WHERE City IN ('Paris','London')**

**SQL Between**

SQL where Betwen Dijelaskan

1. WHERE BETWEEN mengembalikan nilai dalam rentang tertentu .
2. WHERE BETWEEN adalah singkatan dari >= AND <=.
3. BETWEEN bersifat inklusif, yaitu nilai awal dan akhir disertakan.

Contoh

Masalah: Buat daftar semua pesanan antara $1000 dan $2000.

**SELECT \***

**FROM [Order]**

**WHERE TotalAmount BETWEEN 1000 AND 2000**

**SQL Aliases**

SQL Alias ​​Dijelaskan

1. Alias ​​​​adalah “**shorthand**” untuk nama tabel atau kolom.
2. Alias ​​​​mengurangi jumlah pengetikan yang diperlukan untuk memasukkan kueri.
3. Kueri kompleks dengan alias umumnya lebih mudah dibaca.
4. Alias ​​​​berguna dengan GABUNG dan agregat: SUM, COUNT, dll.
5. Alias ​​hanya ada selama kueri.

Contoh :

Masalah: Buat daftar semua nama pelanggan dengan judul Nama dan nama kolom.

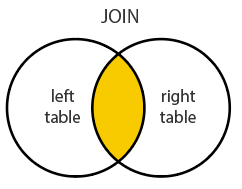
**SELECT FirstName + ' ' + LastName AS Name**

**FROM Customer**

**SQL Joins**

SQL JOINS Dijelaskan

1. Kueri SQL JOIN menggabungkan catatan dari dua tabel.
2. GABUNG menempatkan nilai kolom terkait dalam dua tabel.
3. Kueri dapat berisi nol, satu, atau beberapa operasi GABUNG.



Contoh

Masalah: Daftar semua pemasok dengan produk mereka.

**SELECT CompanyName, ProductName**

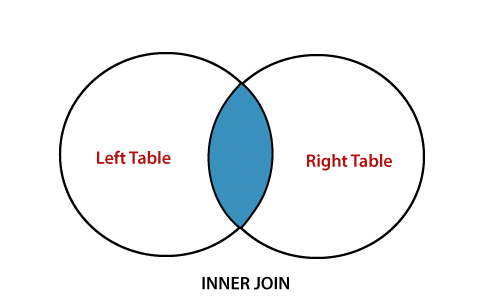
**FROM Supplier**

**JOIN Product ON Supplier.Id = Product.SupplierId**

**SQL Inner Join**

INNER JOIN dalam SQL adalah jenis join yang paling umum dan penting yang memungkinkan pengguna untuk mengakses data yang cocok dari dua atau lebih tabel database.

Ketika kondisi gabungan terpenuhi di antara tabel, maka ia mengembalikan semua baris umum dari mereka.

Diagram Venn INNER JOIN ditunjukkan pada gambar berikut. Daerah yang diarsir dari diagram Venn menunjukkan nilai perpotongan dari dua tabel:

Syntax of INNER JOIN in SQL

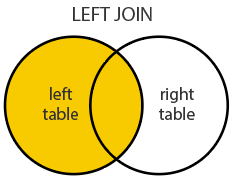
**SELECT** Column\_Name1, Column\_Name2, ….., Column\_NameN

**FROM** Table\_Name1 **INNER** JOIN Table\_Name2

**ON** Table\_Name1.Column\_Name = Table\_Name2.Column\_Name;

**SQL Left Join**

SQL left join Dijelaskan

1. LEFT JOIN melakukan penggabungan yang dimulai dengan tabel kiri .
2. Kemudian, catatan yang cocok dari tabel kanan akan disertakan.
3. Baris tanpa kecocokan mengembalikan kolom dengan nilai NULL.

Contoh

Masalah: Daftar semua pemasok dan produk mereka, termasuk pemasok tanpa produk.

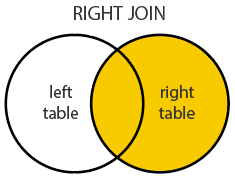
**SELECT CompanyName, ProductName**

**FROM Supplier**

**LEFT JOIN Product ON Supplier.Id = Product.SupplierId**

**SQL Right Join**

SQL RIGHT JOIN Dijelaskan

1. RIGHT JOIN melakukan join yang dimulai dengan tabel yang benar .
2. Kemudian, catatan yang cocok dari tabel kiri akan disertakan.
3. Baris tanpa kecocokan mengembalikan kolom dengan nilai NULL.

Contoh

Masalah: Daftar semua produk yang tidak memiliki pesanan.

**SELECT ProductName**

**FROM OrderItem I**

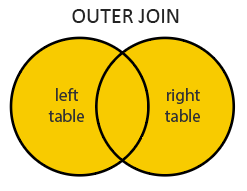
**RIGHT JOIN Product P ON P.Id = I.ProductId**

**WHERE I.Id IS NULL**

**SQL Full Join**

SQL FULL JOIN Dijelaskan

1. A FULL JOIN mengembalikan semua record dari kedua tabel .
2. Ini termasuk catatan yang tidak cocok.
3. Baris yang tidak cocok mengembalikan kolom dengan nilai NULL.
4. A FULL JOIN berpotensi mengembalikan kumpulan data besar .



Contoh

Masalah: Cantumkan semua pelanggan dan tanggal pesanan mereka, termasuk mereka yang tidak memesan.

**SELECT C.FirstName, C.LastName, O.OrderDate**

**FROM Customer C**

**FULL JOIN [Order] O ON C.Id = O.CustomerId**

**ORDER BY O.OrderDate**

**SQL Self Join**

SQL Self JOIN Dijelaskan

1. Self JOIN terjadi ketika sebuah tabel bergabung dengan dirinya sendiri .
2. Ini dapat berguna saat membuat model hierarki.
3. Self JOIN juga berguna untuk perbandingan dalam sebuah tabel.

Contoh

Masalah: Cocokkan pemasok dari negara yang sama.

**SELECT A.CompanyName AS Company1, B.CompanyName AS Company2, A.Country**

**FROM Supplier A, Supplier B**

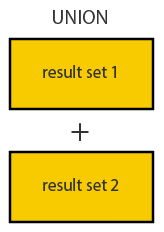
**WHERE A.Id <> B.Id**

**AND A.Country = B.Country**

**ORDER BY A.Country**

**SQL Union**

SQL UNION versus SQL UNION ALL

1. UNION menggabungkan kumpulan hasil dari 2 kueri atau lebih.
2. Tipe data kolom dalam kueri ini harus cocok.
3. SQL UNION hanya mengembalikan baris unik.
4. SQL UNION ALL memungkinkan baris duplikat hadir.

Contoh

Masalah: Daftar semua kontak (pelanggan dan pemasok) dan nomor telepon mereka, diurutkan berdasarkan negara.

**SELECT FirstName + ' ' + LastName AS Contact, Phone, Country**

**FROM Customer**

**UNION**

**SELECT ContactName, Phone, Country**

**FROM Supplier**

**ORDER BY Country**

**SQL Group By**

SQL GROUP BY Dijelaskan

1. Klausa GROUP BY mengelompokkan catatan ke dalam baris ringkasan .
2. GROUP BY mengembalikan satu record untuk setiap grup.
3. GROUP BY dapat mengelompokkan record menurut satu atau lebih kolom.
4. GROUP BY digunakan dengan agregat: COUNT, MAX, dll.

Contoh

Masalah: Buat daftar jumlah produk untuk setiap pemasok, diurutkan dari tinggi ke rendah

**SELECT S.CompanyName, COUNT(P.Id) AS Products**

**FROM Supplier S**

**JOIN Product P ON S.Id = P.SupplierId**

**GROUP BY S.CompanyName**

**ORDER BY COUNT(P.Id) DESC**

**SQL Having**

SQL GROUP BY HAVING Dijelaskan

1. HAVING seperti WHERE tetapi beroperasi pada catatan yang dikelompokkan.
2. HAVING mengharuskan adanya klausa GROUP BY.
3. Grup yang memenuhi kriteria HAVING akan dikembalikan.
4. WHERE dan HAVING dapat digunakan dalam kueri yang sama.
5. HAVING digunakan dengan agregat: COUNT, SUM, dll.

Contoh

Masalah: Daftar semua negara dengan lebih dari 2 pemasok.

**SELECT Country, COUNT(Id) AS Count**

**FROM Supplier**

**GROUP BY Country**

**HAVING COUNT(Id) > 2**

**SQL Exists**

SQL WHERE EXISTS Dijelaskan

1. WHERE EXISTS menguji apakah subquery mengembalikan catatan apa pun .
2. EXISTS mengembalikan nilai true jika subquery mengembalikan satu atau lebih record.
3. EXISTS biasanya digunakan dengan subquery yang berkorelasi .

Contoh

Masalah: Buat daftar pelanggan dengan pesanan lebih dari $5000.

**SELECT FirstName, LastName**

**FROM Customer**

**WHERE EXISTS**

**(SELECT Id**

**FROM [Order]**

**WHERE CustomerId = Customer.Id**

**AND TotalAmount > 5000)**

**SQL Any, All**

SQL ANY and ALL Dijelaskan

1. Kata kunci ANY dan ALL digunakan dengan WHERE atau HAVING.
2. ANY dan ALL beroperasi pada subkueri yang mengembalikan banyak nilai.
3. ANY mengembalikan true jika salah satu nilai subquery memenuhi kondisi.
4. ALL mengembalikan true jika semua nilai subquery memenuhi kondisi.

Contoh

Masalah: Buat daftar produk yang telah terjual lebih dari $45.

**SELECT ProductName AS 'Product'**

**FROM Product**

**WHERE Id = ANY**

**(SELECT ProductId**

**FROM OrderItem**

**WHERE UnitPrice > 45)**

**SQL Select Into**

SQL SELECT INTO Pernyataan

1. SELECT INTO menyalin data dari satu tabel ke tabel baru .
2. SELECT INTO membuat tabel baru yang terletak di grup file default.

Contoh

Masalah: Salin produk yang harganya lebih dari $75 ke tabel LuxeProduct baru.

**SELECT \***

**INTO LuxeProduct**

**FROM Product**

**WHERE UnitPrice > 75**

**SQL Insert Into Select**

Pernyataan SQL INSERT INTO SELECT

1. INSERT INTO SELECT menyalin data dari satu tabel ke tabel lain.
2. INSERT INTO SELECT mengharuskan tipe data dalam tabel sumber dan target cocok.

Contoh

Masalah: Salin semua pelanggan dari Paris ke tabel Pemasok.

**INSERT INTO Supplier (CompanyName, ContactName, City, Country, Phone)**

**SELECT FirstName + ' Company', FirstName + ' ' + LastName, City, Country, Phone**

**FROM Customer**

**WHERE City = 'Paris'**

**SQL Case**

Pernyataan SQL CASE

The CASE pernyataan melewati kondisi dan mengembalikan nilai saat kondisi pertama bertemu (seperti pernyataan if-then-else). Jadi, begitu suatu kondisi benar, itu akan berhenti membaca dan mengembalikan hasilnya. Jika tidak ada kondisi yang benar, ia mengembalikan nilai dalam ELSE klausa.

Jika tidak ada ELSE bagian dan tidak ada kondisi yang benar, ia mengembalikan NULL.

Contoh KASUS SQL

SQL berikut melewati kondisi dan mengembalikan nilai saat kondisi pertama terpenuhi:

Contoh

SELECT OrderID, Quantity,  
CASE  
    WHEN Quantity > 30 THEN 'The quantity is greater than 30'  
    WHEN Quantity = 30 THEN 'The quantity is 30'  
    ELSE 'The quantity is under 30'  
END AS QuantityText  
FROM OrderDetails;

**SQL Null Functions**

Lihat tabel "Produk" 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.

**SQL Stored Procedures**

SQL 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.

Contoh Prosedur Tersimpan

Pernyataan SQL berikut membuat prosedur tersimpan bernama "SelectAllCustomers" yang memilih semua rekaman dari tabel "Pelanggan":

Contoh

CREATE PROCEDURE SelectAllCustomers  
AS  
SELECT \* FROM Customers  
GO;

Jalankan prosedur tersimpan di atas sebagai berikut:

Contoh

EXEC SelectAllCustomers;

**SQL Comments**

SQL Comments

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

Comments Satu Baris

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:

Contoh

--Select all:  
SELECT \* FROM Customers;

Contoh berikut menggunakan komentar satu baris untuk mengabaikan akhir baris:

Contoh

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

Contoh berikut menggunakan komentar satu baris untuk mengabaikan pernyataan:

Contoh

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

**SQL Operators**

Kata-kata dan karakter yang dicadangkan SQL disebut operator, yang digunakan dengan klausa WHERE dalam kueri SQL. Dalam SQL, operator dapat berupa operator unary atau biner. Operator unary hanya menggunakan satu operan untuk melakukan operasi unary, sedangkan operator biner menggunakan dua operan untuk melakukan operasi biner.

**For Example,**

UPDATE employee

SET salary = 20 - 3 \* 5 WHERE Emp\_Id = 5;

Dalam contoh SQL di atas, gaji ditetapkan 5, bukan 85, karena \* (Perkalian)

Operator memiliki prioritas lebih tinggi daripada operator - (pengurangan), jadi pertama kali dikalikan dengan 3\*5 dan kemudian dikurangi dari 20.

**Jenis Operator**

Operator SQL dikategorikan dalam kategori berikut:

1. Operator Aritmatika SQL
2. Operator Perbandingan SQL
3. Operator Logika SQL
4. Operator Set SQL
5. Operator SQL Bit-bijaksana
6. Operator SQL Unary

**SQL Database**

**SQL Create DB**

**SQL Drop DB**

**SQL Backup DB**

**SQL Create Table**

**SQL Drop Table**

**SQL Alter Table**

**SQL Constraints**

**SQL Not Null**

**SQL Unique**

**SQL Primary Key**

**SQL Foreign Key**

**SQL Check**

**SQL Default**

**SQL Index**

**SQL Auto Increment**

**SQL Dates**

**SQL Views**

**SQL Injection**

**SQL Hosting**

**SQL Data Types**

**SQL References**

**SQL Keywords**

**MySQL Functions**

**SQL Server Functions**

**MS Access Functions**

**SQL Quick Ref**

[**https://www-javatpoint-com.translate.goog/sql-select?\_x\_tr\_sl=en&\_x\_tr\_tl=id&\_x\_tr\_hl=en-US&\_x\_tr\_pto=nui,op,sc**](https://www-javatpoint-com.translate.goog/sql-select?_x_tr_sl=en&_x_tr_tl=id&_x_tr_hl=en-US&_x_tr_pto=nui,op,sc)