**ATURAN SQL**

SQL (Structured Query Languange) merupakan bahasa pemrograman yang digunakan untuk mengakses tabel dalam database, baik database client tunggal maupun client-server. Nama tabel, nama kolom (fields), alias tabel/fields, dalam pernyataan SQL mempunyai aturannya sendiri.

1. **Nama Tabel**  
Nama tabel hanya boleh terdiri atas karakter alphanumeric dan lambang garis bawah “\_”. Untuk database client tunggal seperti dBASE, Paradox, Foxpro dinyataan berada dalam tanda kutip (tunggal/ganda), sedang database client-server biasanya tidak perlu ada tanda kutip. Nama tabel juga dapat dideklarasikan secara lengkap bersama file path-nya, contoh :

SELECT \*

FROM 'Parts.dbf'

atau

SELECT \*

FROM "c:sampleparts.dbf"

Nama alias dapat disertakan dalam deklarasi nama tabel, caranya deklarasikan nama alias yang berada di antara tanda titik dua sebelum nama tabel kemudian tutup dengan tanda kutip seperti di bawah ini.

SELECT \*

FROM ":DBDEMOS:Customer.db"

2. **Nama Kolom**  
Aturannya sama dengan penulisan nama tabel.

3. **Format Data bertipe Date**  
Format data tanggal yang diterima adalah format tanggal Amerika dan Eropa. Jika tanda pemisah data adalah garis miring (“/”) dan mode shortdate-nya menempatkan bulan sebelum hari maka format tanggal yang digunakan adalah format Amerika yaitu MM/DD/YY atau MM/DD/YYYY. Format Eropa menggunakan tanda pemisah titik (“.”) dan mode shortdate-nya menempatkan hari sebelum bulan, yaitu DD.MM.YY atau DD.MM.YYYY. Kombinasi dengan mode sortdate, penempatan bulan sebelum hari dan tanda pemisah titik tidak diterima oleh Local SQL. Contoh format Amerika

SELECT \*

FROM Orders

WHERE (SaleDate <= "1/23/1998")

Contoh format Eropa

SELECT \*

FROM Orders

WHERE (SaleDate <= "23.1.1998")

4. **Format Data bertipe Time**  
SQL menyarankan literal Time berformat hh:mm:ss AM/PM, dimana hh adalah jam, mm adalah menit dan ss adalah detik. Penanda AM/PM adalah suatu opsi dan bersifat case-insensitive (“AM” sama dengan “am”). Literal Time harus berada dalam tanda kutip.

INSERT INTO WorkOrder

(ID, StartTime)

VALUES ("B00120","10:30:00 PM")

AM atau PM mengindikasikan separuh hari, jika tidak dinyatakan maka data waktu dibandingkan dengan jam 12. Jika kurang dari jam 12 waktu dalam AM, jika lebih besar dari jam 12 adalah petang (PM). Sebagai contoh, pernyataan berikut hanya akan menampilkan data dimana kolom **Event\_time** berisi nilai “**1:00:00 PM**” dan mengabaikan nilai “**1:00:00 AM**“

SELECT \*

FROM "Events.db"

WHERE (Event\_Time = "13:00:00")

5. **Nilai Boolean**  
Literal Boolean bernilai True dan False dapat dinyatakan dalam pernyataan SQL dengan atau tanpa kutip dan bersifat case-insensitive (huruf besar/kecil bernilai sama).

SELECT \*

FROM Transfers

WHERE (Paid = TRUE)

AND NOT (Incomplete = "False")

Nilai tipe Boolean dapat dikonversi menjadi bertipe lain dengan menggunakan fungsi CAST. Hasil konversi nilai True ke tipe data Integer adalah 1 (satu), False menjadi 0 (nol), dan kosong (NULL) menjadi kosong (NULL). Dalam tipe Char, nilai True adalah T, False menjadi F, dan kosong (NULL) menjadi kosong (NULL). Nilai Boolean juga dapat diekpresikan secara langsung sebagai “T” untuk nilai True dan “F” untuk nilai False, pendeklarasiannya harus menyertakan tanda kutip.

SELECT \*

FROM Transfers

WHERE (Paid = "T")

AND NOT (Incomplete = "F")

6. **Nama Alias Tabel**  
Nama alias tabel dapat digunakan untuk menghubungkan suatu kolom dengan tabel dimana kolom tersebut berasal. Hal ini berguna ketika berbagai kolom dengan nama yang sama dinyatakan dalam query yang sama, khususnya query multi-tabel. Nama alias tabel didefinisikan dalam klausa FROM pada pernyataan SELECT dengan identitas yang khas. Identitas ini, nama alias tabel, dapat digunakan untuk mengawali nama kolom. Jika nama tabel bukan string dalam tanda kutip, secara ekplisit nama alias tabel adalah nama tabel tersebut dan dapat digunakan untuk mengawali nama kolom.

SELECT \*

FROM Customer

LEFT OUTER JOIN Orders ON (Customer.CustNo = Orders.CustNo)

Untuk database client tunggal, jika nama tabel berada dalam tanda kutip maka awali nama kolom dengan string yang berada dalam tanda kutip tersebut,

SELECT \*

FROM "Customer.db"

LEFT OUTER JOIN "Orders.db" ON ("Customer.db".CustNo = "Orders.db".CustNo)

atau gunakan nama lengkap tabel sebagai nama alias tabel yang didefinisikan dalam klausa FROM (dan awali semua kolom dengan nama tersebut),

SELECT \*

FROM "Customer.db" CUSTOMER

LEFT OUTER JOIN "Orders.db" ORDERS ON (CUSTOMER.CustNo = ORDERS.CustNo)

atau gunakan suatu karakter pembeda sebagai nama alias tabel yang didefinisikan dalam klausa FROM (dan awali semua kolom dengan karakter tersebut).

SELECT \*

FROM "Customer.db" C

LEFT OUTER JOIN "Orders.db" O ON (C.CustNo = O.CustNo)

7. **Nama Alias Kolom**  
Gunakan kata kunci **AS** untuk menetapkan nama alias suatu kolom, nilai aggregate, atau nilai literal. Nama alias kolom tidak diperbolehkan berada dalam tanda kutip dan juga tidak boleh mengandung spasi. Pernyataan berikut menetapkan Sub dan Word sebagai nama alias kolom.

SELECT

SUBSTRING(Company FROM 1 FOR 1) AS Sub,

"Text" AS Word

FROM Customer

**DML**

Data Manipulation Language (DML) berisi pernyataan SQL yang digunakan untuk mencari/menampilkan, menyisipkan/memasukkan, memperbaharui dan menghapus data tabel. Saya membagi pernyataan DML menjadi dua bagian, yaitu Pernyataan dan Klausa.

1. **Pernyataan**  
1.1. **SELECT**

Syntax:

**SELECT** [**DISTINCT**] \* | column\_list

**FROM** table\_reference

[**WHERE** predicates]

[**GROUP BY** group\_list]

[**HAVING** having\_condition]

[**ORDER BY** order\_list]

Digunakan untuk mencari/menampilkan record dari tabel. Pernyataan ini bisa menampilkan semua atau sebagian data tabel. Gunakan tanda bintang (“\*”) untuk menampilkan nilai dari semua kolom. Kolom yang dideklarasikan dalam klausa SELECT dapat berasal dari satu tabel atau banyak tabel, juga dapat berupa kolom tabel, nilai lateral, dan nilai hasil perhitungan yang dipisahkan oleh tanda koma. Klausa FROM mengidentifikasikan tabel dimana datanya akan ditampilkan. Pernyataan berikut akan menampilkan data dari dua kolom (CustNo dan Company), plus sebuah nilai lateral (“Current”) dan nilai hasil perhitungan untuk semua record dari tabel Customer

SELECT

CustNo,

Company,

"Current" AS CURRENT,

CAST(LastInvoiceDate AS DATE)

FROM Customer

Gunakan kata kunci DISTINCT untuk membatasi data yang ditampilkan hanya sekali pada record yang berbeda, artinya data yang sama pada kolom tersebut hanya ditampilkan sekali saja. Kata kunci ini hanya dapat dipakai dengan kolom sederhana bertipe Char dan Integer, tidak dapat dipakai untuk kolom yang bertipe Blob dan Memo. Untuk mendapatkan record dengan suatu kriteria logika, tambahkan klausa WHERE dalam pernyataan tersebut dan untuk mengelompokkan data gunakan klausa GROUP BY. Klausa HAVING digunakan untuk membatasi record yang ditampilkan berdasarkan hasil dari fungsi aggregate, sedangkan untuk mengurutkan data gunakan klausa ORDER BY. Pernyataan SELECT juga dapat digunakan sebagai subquery dalam pernyataan INSERT, DELETE dan UPDATE.

1.2. **DELETE**

Syntax:

**DELETE**

**FROM** table\_reference

[**WHERE** predicates]

Digunakan untuk menghapus satu record atau lebih dari suatu tabel. Klausa WHERE akan membatasi penghapusan tersebut berdasarkan kriteria logika yang dideklarasikan. Jika tidak mendeklarasikan klausa WHERE maka semua record dalam tabel tersebut akan terhapus. Pernyataan berikut akan menghapus semua data tabel Employee.

DELETE

FROM Employee

Dan pernyataan di bawah ini akan menghapus semua baris dalam tabel Employee jika nilai dari kolom Empno ditemukan dalam subquery pencarian data kolom Empno dari tabel Old\_Employee.

DELETE

FROM Employee

WHERE (Empno IN (SELECT Empno FROM Old\_Employee))

Pernyataan DELETE hanya mendukung subquery SELECT dalam klausa WHERE.

1.3. **INSERT**

Syntax:

**INSERT INTO** table\_reference

[(columns\_list)]

**VALUES** (update\_atoms)

Pernyataan INSERT berfungsi untuk menambahkan record data baru pada suatu tabel. Nama tabel yang dinyatakan dalam klausa INTO merupakan tabel yang akan menerima data. Daftar kolom yang dipisahkan oleh tanda koma dan berada dalam tanda kurung merupakan kolom-kolom dari tabel, sedang klausa VALUES merupakan nilai-nilai yang disisipkan yang dipisahkan oleh tanda koma dan berada dalam tanda kurung. Jika tidak ada kolom yang dideklarasikan, maka nilai yang diberikan disimpan dalam kolom secara berurutan dalam struktur tabel, sehingga nilai pertama dalam klausa VALUES akan dimasukkan dalam kolom pertama tabel, demikian seterusnya.

Jika daftar kolom dinyatakan secara jelas maka nilai yang diberikan akan disimpan sesuai dengan urutan daftar kolom tersebut. Kolom-kolom yang tidak dinyatakan dalam daftar kolom tidak mempunyai nilai (kosong bukan nol). Jumlah kolom yang dinyatakan harus sama dengan jumlah nilai dalam klausa VALUES. Pernyataan berikut hanya memasukkan suatu nilai dalam kolom CustNo dan Company dari tabel Employee, dan nilai kolom-kolom lain adalah kosong.

INSERT INTO Customer

(CustNo, Company)

VALUES (9842, "Inprise Corporation")

Untuk menambahkan record pada satu tabel yang berasal dari tabel lain, hilangkan klausa VALUES dan gunakan subquery sebagai sumber baris baru tersebut.

INSERT INTO Customer

(CustNo, Company)

SELECT CustNo, Company

FROM OldCustomer

Pernyataan INSERT juga hanya mendukung subquery SELECT dalam klausa VALUES.

1.4. **UPDATE**

Syntax:

**UPDATE** table\_reference

**SET** column\_ref = update\_atom

[, column\_ref = update\_atom...]

[**WHERE** predicates]

Untuk memodifikasi satu record atau lebih yang telah ada dalam suatu tabel. Nama tabel dalam pernyataan UPDATE merupakan tabel yang akan menerima perubahan data. Setiap ungkapan dalam klausa SET terdiri atas nama kolom, operator penugasan (=), nilai update dari kolom tersebut. Nilai kondisi **update\_atom** dalam pernyataan UPDATE bisa berupa nilai lateral, nilai tunggal (satu record) dari hasil subquery SELECT, atau nilai hasil perhitungan.

UPDATE SalesInfo

SET TaxRate = 0.0825

Jika nilai kondisi **update\_atom** berasal dari suatu subquery SELECT, sebuah pernyataan SELECT harus dibuat untuk setiap kolom yang akan di-update dalam tabel yang dimaksud. Subquery SELECT tersebut harus berada dalam tanda kurung. Pernyataan berikut meng-update dua kolom OnHand dan InventoryDate dalam tabel Inventory, masing-masing dengan subquery SELECT terpisah.

UPDATE Inventory

SET OnHand = OnHand – (SELECT SUM(Orders.QtySold)

FROM Orders

WHERE (Orders.PartNo = Inventory.PartNo)

AND (Orders.OrderDate BETWEEN "10/01/1999"

AND "10/31/1999")),

InventoryDate = OnHand – (SELECT MAX(Orders.OrderDate)

FROM Orders

WHERE (Orders.PartNo = Inventory.PartNo)

AND (Orders.OrderDate BETWEEN "10/01/1999"

AND "10/31/1999"))

Klausa WHERE membatasi update record dalam tabel, jika tidak ada klausa WHERE yang dinyatakan maka semua baris tabel akan di-update dengan menggunakan ungkapan dalam klausa SET.

UPDATE SalesInfo

SET TaxRate = 0.0825

WHERE (State = "CA")

Pernyataan UPDATE hanya mendukung subquery SELECT dalam klausa WHERE.

2. **Klausa**  
2.1. **FROM**

Syntax:

**FROM** table\_reference [, table\_reference...]

Digunakan untuk menetapkan tabel yang datanya akan ditampilkan melalui suatu pernyataan SELECT. Nilai klausa FROM adalah daftar nama tabel yang dipisahkan oleh tanda koma. Nama alias suatu tabel yang didefinisikan dalam klausa FROM dapat digunakan dalam suatu pernyataan. Pada pernyataan SELECT berikut, nama alias Res didefinisikan untuk tabel RERSERVAT

SELECT

Res.ResNo,

Res.Amt\_Paid

FROM Reservat Res

WHERE (Res.Pay\_Method = "VISA")

Klausa ini diterapkan dalam pernyataan SELECT.

2.2. **WHERE**

Syntax:

**WHERE** predicates

Menyatakan kondisi-kondisi penyaringan data dalam pernyataan SELECT dan UPDATE. Nilai klausa WHERE adalah satu ungkapan logika atau lebih, atau predikat, atau penilaian True atau False setiap record tabel. Multi predikat dalam klausa WHERE harus dipisahkan oleh salah satu operator logika OR atau AND dan setiap nilai predikat dapat dinegasikan dengan operator NOT.

Tanda kurung dapat digunakan untuk mengisolasi perbandingan logika dan kelompok pembanding yang menghasilkan kriteria penilaian baris yang berbeda. Sebagai contoh, pernyataan SELECT berikut menampilkan semua record dimana kolom Shape berisi nilai “round” atau “square” tetapi hanya jika kolom Color berisi nilai “red”.

SELECT

Shape,

Color,

Cost

FROM Objects

WHERE ((Shape = "round") OR (Shape = "square"))

AND (Color = "red")

Tetapi jika semua tanda kurung dalam klausa WHERE tersebut dibuang maka hasilnya akan sangat berbeda. Pernyataan tersebut (semua tanda kurung dibuang) akan menampilkan baris dimana kolom Shape berisi nilai “round” tanpa memperhatikan nilai kolom Color, juga akan menampilkan baris dimana kolom Shape berisi nilai “square” tetapi hanya jika kolom Color berisi nilai “red”. Klausa WHERE menyaring data sebelum pengelompokan data oleh klausa GROUP BY.

2.3. **ORDER BY**

Syntax:

**ORDER BY** column\_reference [, column\_reference...] [**ASC**|**DESC**]

Mengurutkan record yang ditampilkan oleh pernyataan SELECT berdasarkan nilai satu kolom atau lebih. Nilai dari klausa ini adalah daftar kolom yang dipisahkan oleh tanda koma. Kolom-kolom tersebut dapat berasal dari satu atau banyak tabel. Jika kolom-kolom tersebut berasal dari banyak tabel, tabel-tabel tersebut harus merupakan bagian dari gabungan tabel. Gunakan kata kunci ASC (atau ASCENDING) untuk mengurutkan data dari yang paling kecil ke yang paling besar, dan jika sebaliknya gunakan kata kunci DESC (atau DESCENDING). Jika tidak dinyatakan maka pengurutan data berdasarkan metode ASC. Pernyataan berikut mengurutkan data secara ascending oleh Year yang dikutip dari kolom LastInvoiceDate, selanjutnya secara descending oleh kolom State, dan kemudian secara ascending oleh hasil konversi huruf besar kolom Company.

SELECT

EXTRACT(YEAR FROM LastInvoiceDate) AS YY,

State,

UPPER(Company)

FROM Customer

ORDER BY

YY DESC,

State ASC, 3

2.4. **GROUP BY**

Syntax:

**GROUP BY** column\_reference [, column reference...]

Mengelompokan data berdasarkan suatu kolom. Sebagai contoh, suatu query dengan fungsi SUM menghasilkan record tunggal yang merupakan total seluruh nilai dari kolom dalam fungsi SUM tersebut. Tetapi jika klausa GROUP BY ditambahkan, maka fungsi SUM melakukan penjumlahan sekali untuk masing-masing kelompok record. Klausa GROUP BY diperlukan jika kolom aggregate dan non-aggregate dimasukkan dalam pernyataan SELECT yang sama.

Setiap kolom yang dinyatakan dalam klausa ini merupakan bagian dari tabel yang ditetapkan dalam klausa FROM query, juga terdaftar dalam klausa SELECT query dan tidak mempunya fungsi aggregate yang diberlakukan dalam klausa SELECT. Dalam pernyataan SELECT berikut, nilai kolom Sales dikelompokan berdasarkan nilai-nilai dalam kolom Company. Hasilnya adalah total penjualan untuk setiap perusahaan.

SELECT

C.Company,

SUM(O.ItemsTotal) AS TotalSales

FROM Customer C, Orders O

WHERE C.CustNo = O.CustNo

GROUP BY C.Company

ORDER BY C.Company

Nama kolom yang dinyatakan dalam klausa GROUP BY dapat berupa nama korelasi kolom, seperti contoh berikut ini :

SELECT

C.Company Co,

SUM(O.ItemsTotal) AS TotalSales

FROM Customer C, Orders O

WHERE C.CustNo = O.CustNo

GROUP BY Co

ORDER BY 1

Nilai hasil perhitungan tidak dapat dinyatakan dalam klausa GROUP BY. Dan klausa ini hanya dapat diterapkan dalam pernyataan SELECT.

2.5. **HAVING**

Syntax:

**HAVING** predicates

Untuk membatasi record yang ditampilkan oleh pernyataan SELECT dimana nilai-nilai kolom yang telah dikumpulkan memenuhi kriteria yang ditetapkan. Klausa HAVING hanya dapat digunakan dalam pernyataan SELECT jika pernyataan tersebut mempunyai klausa GROUP BY dan satu kolom atau lebih menjadi subjek fungsi aggregate. Nilai klausa HAVING adalah satu ekpresi logika atau lebih, atau predikat yang menilai True atau False tiap record aggregate dari tabel. Sebagai contoh, pernyataan SELECT di bawah menampilkan semua baris dimana total penjualan untuk perusahaan melebihi 1,000.

SELECT

Company,

SUM(sales) AS TOTALSALES

FROM Sales1998

GROUP BY Company

HAVING (SUM(sales) >= 1000)

ORDER BY Company

Pernyataan multi predikat harus dipisahkan oleh salah satu operator logika OR atau AND dan setiap predikat dapat dinegasikan dengan operator NOT. Tanda kurung dapat digunakan untuk mengisolasi perbandingan logika dan kelompok pembanding yang menghasilkan kriteria penilaian record yang berbeda.

Pernyataan SELECT dapat terdiri atas klausa WHERE dan HAVING. Klausa WHERE menyaring data yang dikumpulkan dan menggunakan kolom yang bukan subjek fungsi aggregate. Sedang klausa HAVING melanjutkan penyaring data setelah dikumpulkan dan kolom yang digunakan adalah subjek dari fungsi aggregate. Perhatikan contoh berikut ini, sama dengan contoh terdahulu tetapi data yang ditampilkan jika nilai kolom State adalah “CA”.

SELECT

Company,

SUM(sales) AS TOTALSALES

FROM Sales1998

WHERE (State = "CA")

GROUP BY Company

HAVING (SUM(sales) >= 1000)

ORDER BY Company

Pernyataan berikut menggunakan subquery yang menjumlah semua penjualan untuk California (“CA”) dan menghitung nilai total yang dibagi dua. Nilai perhitungan ini kemudian digunakan untuk membatasi data hasil fungsi aggregate dimana subtotal (didasarkan pada nomor pelanggan) dikelompokkan pada lebih dari atau sama dengan nilai hasil perhitungan tersebut.

SELECT

O.CustNo,

SUM(O.ItemsTotal)

FROM Orders O

GROUP BY O.CustNo

HAVING SUM(O.ItemsTotal) >= (SELECT SUM(O2.ItemsTotal) / 2

FROM Customer C,Orders O2

WHERE (C.CustNo = O2.CustNo) AND (C.State = "CA"))

Jadi klausa HAVING menyaring data setelah pengumpulan oleh klausa GROUP BY, dan untuk menyaring data sebelum pengumpulan data gunakan klausa WHERE.

**DDL**

Data Definition Language (DDL) berisi pernyataan SQL yang digunakan untuk mendefinisikan, mengatur semua atribut dan properti sebuah database. Dengan pernyataan DLL kamu bisa membuat, mengubah dan menghapus tabel dan juga untuk membuat dan menghapus indeks atau view.

1. **CREATE TABLE**

Syntax:

**CREATE TABLE** table\_name

(column\_definition [,column\_definition, ...] [, primary\_key\_constraint])

Digunakan untuk membuat suatu tabel , mendefinisikan kolom-kolomnya, dan mendefinisikan primary key. Untuk database client tunggal, nama tabel yang digunakan harus berada dalam tanda kutip dan memenuhi aturan yang berlaku. Definisi kolom/fields berisi kombinasi nama kolom, tipe data beserta ukurannya (jika ada). Daftar definisi kolom dipisahkan oleh tanda koma dan berada dalam tanda kurung. Jumlah dan jenis ukuran harus disesuaikan dengan tipe kolom. Sedangkan Primary key digunakan untuk membuat indeks utama dari tabel. Pernyataan berikut membuat tabel Paradox bernama employee.db dengan primary key kolom Last\_Name dan First\_Name :

CREATE TABLE "employee.db"

(Last\_Name CHAR(20), First\_Name CHAR(15), Salary NUMERIC(10,2), Dept\_No SMALLINT,

PRIMARY KEY (Last\_Name, First\_Name))

Jika kamu ingin membuat tabel dBASE berikan ekstensi .dbf pada waktu pemberian nama, menjadi employee.dbf. Dalam penulisan nama tabel, diperbolehkan untuk tidak menyertakan ekstensinya tetapi tanda kutip harus dihilangkan jika tidak maka nama tabel adalah db/dbf.

Dan jika ingin membuat tabel dengan kolom berkarakter non-alpanumerik atau nama kolom berspasi, kamu harus memasukkan nama kolom tersebut dalam tanda kutip dan sertakan di depan nama kolom tersebut dengan nama tabel (dengan atau tanpa ekstensi). Tetapi sebaiknya hindari nama kolom berspasi, gunakan tanda garis bawah “\_”. Berikut ini dua macam cara pembuatan tabel dengan nama kolom berspasi.

CREATE TABLE "abc.db"

(ID CHAR(3), "ABC.db"."Funny Name" CHAR(10))

CREATE TABLE abc

(ID CHAR(3), ABC."Funny Name" CHAR(10))

2. **ALTER TABLE**

Syntax:

**ALTER TABLE** table\_reference **DROP** [COLUMN] column\_reference

| **ADD** [COLUMN] column\_reference [,reference **DROP** [COLUMN] column\_reference

| **ADD** [COLUMN] column\_reference...]

Pernyataan ini berfungsi untuk menambahkan atau menghapus suatu kolom dari tabel yang telah ada. Dalam pernyataan ALTER TABLE yang sama, diperbolehkan menghapus satu kolom dan menambahkan kolom yang lain. Kata kunci DROP hanya memerlukan nama kolom yang akan dihapus, sedangkan ADD memerlukan kombinasi lengkap dari nama kolom, tipe data dan mungkin ukurannya yang sesuai. Pernyataan berikut menghapus kolom FullName dan menambahkan kolom LastName dalam tabel Names.

ALTER TABLE Names

DROP FullName, ADD LastName CHAR(25)

Pernyataan ini juga memungkinkan penghapusan dan penambahan kolom bernama sama dalam satu pernyataan ALTER TABLE, tetapi data dalam kolom lama tersebut akan hilang.

ALTER TABLE Names

DROP LastName, ADD LastName CHAR(30)

Jika kolom yang dihapus merupakan bagian dari primary key maka indeks utama akan dihapus. Dan jika kolom tersebut adalah kolom berkarakter non-alpanumerik atau nama kolom berspasi, kamu harus memasukkan nama kolom tersebut dalam tanda kutip dan sertakan di depan nama kolom tersebut dengan nama tabel (dengan atau tanpa ekstensi). Perhatikan kode pernyataan berikut ini:

ALTER TABLE "Customer.db"

ADD "Customer.db"."#ID" CHAR(3)

3. **DROP TABLE**

Syntax:

**DROP TABLE** table\_reference

Digunakan untuk menghapus suatu tabel yang telah ada. Pernyataan berikut ini menghapus suatu tabel

DROP TABLE Employee

4. **CREATE INDEX**

Syntax:

**CREATE** [**UNIQUE**] [**ASC** | **DESC**] **INDEX** index\_reference **ON**

table\_reference (column\_reference [,column\_reference...])

Pernyataan CREATE INDEX digunakan untuk membuat suatu indeks sekunder dari tabel yang telah ada. Nama indeks tidak boleh mengandung spasi. Kata kunci UNIQUE digunakan untuk membuat suatu indeks yang akan membangkitkan pesan kesalahan jika record bernilai sama dimasukkan dalam kolom yang sama. Secara default, indeks tidak unik. Pernyataan ASC (atau ASCENDING) disertakan jika kamu ingin membuat suatu indeks yang mengurutkan data dari yang paling kecil ke yang paling besar, dan jika sebaliknya gunakan kata kunci DESC. Secara default pengurutan data bersifat ascending jika kedua kata kunci tersebut tidak dinyatakan. Indeks sekunder dari suatu tabel Paradox dapat dibuat multi kolom melalui pernyataan berikut :

CREATE INDEX CustDate ON "Orders.db" (CustNo, SaleDate)

Sedangkan untuk tabel dBASE hanya boleh membuat indeks tunggal.

CREATE UNIQUE INDEX Namex ON "Employee.dbf" (Last\_Name)

5. **DROP INDEX**

Syntax:

**DROP INDEX** table\_reference.index\_reference | PRIMARY

Pernyataan ini digunakan untuk menghapus suatu indeks utama atau sekunder. Untuk menghapus indeks utama atau sekunder tabel dBASE atau suatu indeks sekunder tabel Paradox, penulisan nama tabel dan nama indek dipisahkan oleh simbol titik (“.”).

DROP INDEX "Employee.dbf".Namex

Untuk menghapus indeks utama tabel Paradox sertakan kata kunci PRIMARY di belakang nama tabel.

DROP INDEX Orders.PRIMARY

FUNGGSI SQL

Setiap bahasa pemrograman biasanya memiliki fungsi-fungsi yang memudahkan pengguna-nya, tidak terkecuali SQL. Beberapa fungsi yang tertulis dalam artikel ini merupakan fungsi standar dari bahasa pemrograman SQL.

1. **Concatenation**

*Syntax*:

value1 || value2

Digunakan untuk menggabungkan dua nilai karakter (nilai kolom, literal, parameter, atau nilai hasil perhitungan) menjadi data bertipe string. Pernyataan berikut ini mengkombinasikan nilai-nilai kolom dengan karakter literal.

SELECT

LastName || ", " || FirstName

FROM Names

Fungsi ini tidak dapat digunakan dengan kolom bertipe Memo atau Blob.

2. **LOWER**

*Syntax*:

**LOWER**(column\_reference)

Mengkonversi semua nilai karakter (nilai kolom, literal, parameter, atau nilai hasil perhitungan) menjadi huruf kecil. Database seperti SQL Server memakai pernyataan LOWERCASE. Pernyataan SELECT berikut mengubah nilai-nilai dalam kolom Name menjadi huruf kecil.

SELECT

LOWER(Name)

FROM Country

Perubahan nilai akibat fungsi ini tidak akan disimpan jika diberlakukan dalam pernyataan SELECT. Jika dinyatakan sebagai nilai kondisi (update\_atom) dalam pernyataan UPDATE, perubahan nilai tersebut akan disimpan. Fungsi LOWER hanya dapat digunakan dengan kolom bertipe karakter atau literal dan tidak dapat digunakan untuk kolom bertipe Memo dan Blob.

3. **UPPER**

*Syntax*:

**UPPER**(column\_reference)

Merupakan kebalikan dari fungsi LOWER, yaitu mengkonversi semua nilai karakter (nilai kolom, literal, parameter, atau nilai hasil perhitungan) menjadi huruf kapital. Sebagai contoh, pernyataan berikut akan menyaring data berdasarkan nilai kolom Name yang dikonversi ke dalam huruf kapital yang diawali karakter PE (nilai lateral).

SELECT

Name,

Capital,

Continent

FROM Country

WHERE UPPER(Name) LIKE UPPER("Pe%")

Sama seperti fungsi LOWER, perubahan nilai akibat fungsi ini tidak akan disimpan jika diakukan dalam pernyataan SELECT. Jika dinyatakan sebagai nilai kondisi dalam pernyataan UPDATE, perubahan nilai tersebut akan disimpan.

4. **SUBSTRING**

*Syntax*:

**SUBSTRING**(column\_reference **FROM** start\_index [**FOR** length]

Mengutip substring dari nilai karakter (nilai kolom, literal, parameter, atau nilai hasil perhitungan) yang dinyatakan dalam kolom yang dimaksud (**colom\_reference** dalam *syntax* di atas). Klausa FROM adalah posisi karakter untuk memulai pengutipan substring. Karakter pertama nilai sumber diberi indeks 1 (satu). Dan kata kunci FOR menyatakan banyaknya substring yang ingin dikutip. Jika kata kunci FOR tidak dideklarasikan maka substring dimulai dari posisi yang dinyatakan oleh kata kunci FROM sampai akhir string. Contoh berikut mengutip tiga huruf dimulai dari huruf kedua, hasilnya adalah “BCD” jika nilai asalnya adalah “ABCDE” dan dimasukkan dalam kolom lietral bernama Sub

SELECT

SUBSTRING("ABCDE" FROM 2 FOR 3) AS Sub

FROM Country

Dan pernyataan SELECT berikut hanya akan menampilkan huruf kedua dan seterusnya dari kolom Name, artinya nilai “Belgium” dalam kolom tersebut akan ditampilkan sebagai “elgium”.

SELECT

SUBSTRING(Name FROM 2)

FROM Country

Perubahan nilai akibat fungsi ini tidak akan disimpan jika diberlakukan dalam pernyataan SELECT. Jika dinyatakan sebagai nilai kondisi (update\_atom) dalam pernyataan UPDATE, perubahan nilai tersebut akan disimpan. Fungsi SUBSTRING hanya dapat digunakan dengan kolom bertipe karakter atau literal dan tidak dapat digunakan untuk kolom bertipe Memo dan Blob.

5. **TRIM**

*Syntax* :

**TRIM**([**LEADING**|**TRAILING**|**BOTH**] [trimmed\_char] **FROM** column\_reference)

Untuk menghapus karakter pertama, terakhir atau kedua-duanya (pertama dan terkahir) dari nilai karakter (nilai kolom, literal, parameter, atau nilai hasil perhitungan). Berkut ini deskripsi parameter yang ada dalam fungsi TRIM :

|  |  |
| --- | --- |
| **Parameter** | **Deskripsi** |
| LEADING | Menghapus karakter terakhir sisi kiri *string* |
| TRAILING | Menghapus karakter terakhir sisi kanan *string* |
| BOTH | Menghapus karakter terakhir kedua sisi *string* |

*Case-sensitivity* (huruf besar/kecil mempunyai nilai berbeda) berlaku dalam pernyataan parameter tersebut. FROM merupakan nilai karakter sumber yang akan dihapus. Tabel berikut menunjukkan hasil dari variasi deklarasi fungsi TRIM.

|  |  |
| --- | --- |
| **TRIM syntax** | **Hasil** |
| TRIM(LEADING “\_” FROM “\_ABC\_”) | “ABC\_” |
| TRIM(TRAILING “\_” FROM “\_ABC\_”) | “\_ABC” |
| TRIM(BOTH “\_” FROM “\_ABC\_”) | “ABC” |
| TRIM(BOTH “A” FROM “ABC”) | “BC” |

Perubahan nilai akibat fungsi ini tidak akan disimpan jika diberlakukan dalam pernyataan SELECT. Jika dinyatakan sebagai nilai kondisi (update\_atom) dalam pernyataan UPDATE, perubahan nilai tersebut akan disimpan. Fungsi TRIM hanya dapat digunakan dengan kolom bertipe karakter atau literal dan tidak dapat digunakan untuk kolom bertipe Memo dan Blob.

6. **AVG**

*Syntax*:

**AVG**([**ALL**] column\_reference | **DISTINCT** column\_reference)

Menghitung rerata dari nilai numerik (nilai kolom, literal, parameter, atau nilai hasil perhitungan). Nilai 0 (nol) dalam suatu kolom tetap diperhitungkan, sedangkan jika tidak ada nilai (kosong bukan nol) maka fungsi AVG akan mengabaikannya. Kombinasi nilai kolom dan suatu konstanta dapat digunakan dalam fungsi AVG.

SELECT

AVG(ItemsTotal),

AVG(ItemsTotal \* 0.0825) AS AverageTax

FROM Orders

ALL untuk mendapatkan nilai rerata semua *record* sedang DISTINCT akan mengabaikan nilai yang sama suatu kolom pada saat perhitungan nilai rerata. Jika kata kunci DISTINCT tidak dinyatakan, ALL adalah kata kunci default. Jika memakai klausa GROUP BY, fungsi AVG akan menghitung satu nilai untuk tiap kelompok. Nilai tersebut merupakan pengumpulan kolom untuk semua baris pada tiap kelompok. Pernyataan berikut mengelompokkan nilai rerata kolom ItemsTotal dari tabel Orders, menghasilkan subtotal untuk tiap perusahaan dalam tabel Company.

SELECT

C."Company",

AVG(O."ItemsTotal") AS Average

FROM "Customer.db" C, "Orders.db" O

WHERE (C."CustNo" = O."CustNo")

GROUP BY C."Company"

ORDER BY C."Company"

Fungsi AVG hanya beroperasi pad nilai numerik. Untuk nilai non-numerik, sebelum menggunakan fungsi ini, gunakan fungsi CAST untuk mengkonversi kolom menjadi bertipe numerik. Dan fungsi ini tidak dapat digunakan dengan kolom betipe Memo dan Blob.

7. **COUNT**

*Syntax*:

**COUNT**(\* | [**ALL**] column\_reference | **DISTINCT** column\_reference)

Untuk menghitung banyaknya *record* yang ditampilkan oleh pernyataan SELECT (tunggal atau multi query). Jika menambahkan klausa WHERE, fungsi COUNT menampilkan banyaknya *record* yang tersaring.

SELECT COUNT(\*)

FROM "Clients.dbf"

WHERE State = "CA"

8. **MAX**

*Syntax*:

**MAX**([**ALL**] column\_reference | **DISTINCT** column\_reference)

Menghitung nilai terbesar dari nilai (nilai kolom, literal, parameter, atau nilai hasil perhitungan) yang menggunakan dataset. Dataset dapat berupa tabel, suatu dataset yang disaring, atau kelompok logika yang dihasilkan oleh klausa GROUP BY. Fungsi MAX memberikan nilai kosong untuk kolom yang bernilai kosong (bukan nol). ALL untuk mendapatkan nilai tertinggi semua baris sedang DISTINCT akan mengabaikan nilai yang sama suatu kolom pada saat menghitung nilai tertinggi. Jika kata kunci DISTINCT tidak dinyatakan, ALL adalah kata kunci default. Kombinasi nilai kolom dan suatu konstanta dapat digunakan dalam fungsi MAX.

SELECT

MAX(ItemsTotal),

MAX(ItemsTotal \* 0.0825) AS HighestTax

FROM Orders

Jika menggunakan kolom bertipe numerik, nilai yang dihasilkan bertipe sama dengan tipe kolom tersebut (seperti Integer atau Float) . Fungsi MAX tidak dapat digunakan dengan kolom bertipe Memo atau Blob.

SELECT

C."Company",

MAX(O."ItemsTotal") AS Biggest

FROM "Customer.db" C, "Orders.db" O

WHERE (C."CustNo" = O."CustNo")

GROUP BY C."Company"

ORDER BY C."Company"

9. **MIN**

*Syntax*:

**MIN**([**ALL**] column\_reference | **DISTINCT** column\_reference)

Fungsi MIN merupakan kebalikan dari fungsi MAX yaitu menghitung nilai terkecil dari nilai (nilai kolom, literal, parameter, atau nilai hasil perhitungan) yang menggunakan dataset. Deskripsi lainnya hampir sama dengan fungsi MAX.

SELECT

MIN(ItemsTotal)

FROM Orders

10. **SUM**

*Syntax*:

**SUM**([**ALL**] column\_reference | **DISTINCT** column\_reference)

Menghitung total dari nilai (nilai kolom, literal, parameter, atau nilai hasil perhitungan) yang menggunakan dataset. Deskripsi lainnya hampir sama dengan fungsi MAX. Contoh penulisan syntax-nya adalah :

SELECT

SUM(Salary + (Sarary \* 0.1))

FROM "Employee.db"

11. **CAST**

*Syntax*:

**CAST**(column\_reference **AS** data\_type)

Mengkonversi kolom tertentu dalam tipe data tertentu. CAST dapat diterapkan juga untuk nilai literal dan nilai hasil perhitungan. Fungsi ini dapat digunakan dalam daftar kolom pernyataan SELECT, predikat untuk klausa WHERE dan modifikasi nilai dalam pernyataan UPDATE. Parameter **data\_tipe** yang dapat digunakan adalah Char, Integer, Numeric dan seterusnya, sedangkan yang tidak dapat dipakai adalah Blob, Memo dan Byte. Pernyataan berikut mengkonversi nilai kolom bertipe TimeStamp ke tipe Date

SELECT

CAST(SaleDate AS DATE)

FROM ORDERS

Kombinasi fungsi CAST dengan fungsi yang lain dapat digunakan selama tipe datanya sesuai, seperti contoh berikut.

SELECT

SaleDate,

SUBSTRING(CAST(CAST(SaleDate AS DATE) AS CHAR(10)) FROM 1 FOR 1)

FROM Orders

Perubahan nilai akibat fungsi ini tidak akan disimpan jika diberlakukan dalam pernyataan SELECT. Jika dinyatakan sebagai nilai kondisi (update\_atom) dalam pernyataan UPDATE, perubahan nilai tersebut akan disimpan.

12. **EXTRACT**

*Syntax*:

**EXTRACT**(extract\_field **FROM** column\_reference)

Untuk mendapatkan kolom tahun, bulan dan hari dari kolom bertipe Date atau TimeStamp. Jika kolom yang dideklarasikan tidak mempunyai nilai maka hasilnya adalah kosong (bukan nol). Jika nilainya tidak kosong, fungsi EXTRACT akan menyatakan nilai elemen tanggal tertentu sebagai data bertipe SmallInt. Parameter **extract\_fields** yang diperbolehkan adalah : YEAR, MONTH, DAY, HOUR, MINUTE, atau SECOND. Parameter YEAR, MONTH, dan DAY hanya digunakan dengan kolom bertipe Date dan TimeStamp. Sedangkan parameter HOUR, MINUTE, dan SECOND hanya digunakan dengan kolom bertipe TimeStamp dan Time.

SELECT

SaleDate,

EXTRACT(YEAR FROM saledate) AS YY,

EXTRACT(MONTH FROM saledate) AS MM,

EXTRACT(DAY FROM saledate) AS DD

FROM Orders

Pernyataan berikut menggunakan kolom DOB (berisi tanggal lahir) yang menyaring record dimana tanggal lahirnya adalah bulan MEI (dinyatakan dengan angka 5).

SELECT

DOB,

LastName,

FirstName

FROM People

WHERE (EXTRACT(MONTH FROM DOB) = 5)

Fungsi EXTRACT hanya beroperasi pada data bertipe Date, Time dan TimeStamp. Gunakan fungsi CAST untuk mengkonversi tipe data menjadi data bertipe waktu.

[**Relasi Tabel dalam Pernyataan SQL**](http://www.havetogether.com/relasi-tabel-dalam-pernyataan-sql.html)

By

[*rumono*](http://www.havetogether.com/author/rumono/)

– January 13, 2010**Posted in:** [Database](http://www.havetogether.com/category/database)

Data dalam suatu tabel dapat dihubungkan dengan tabel lain dengan operator relasi.

1. **Equi-Join**

*Syntax*:

**SELECT** column\_list

**FROM** table\_reference, table\_reference [, table\_reference...]

**WHERE** predicate [**AND** predicate...]

Operator Equi-Join digunakan untuk menggabungkan dua tabel, suatu tabel sumber dan anggota, yang mempunyai nilai dari satu kolom atau lebih. Untuk *record* tabel sumber yang sesuai dengan tabel anggota, akan dimasukkan dalam hasil. Jika tidak ada kecocokan, data tersebut tidak dimunculkan dalam hasil. Contoh pernyataan berikut ini menggabungkan tabel Customer dan Orders berdasarkan nilai-nilai dalam kolom CustNo, dimana keduanya mempunyai kolom tersebut.

SELECT \*

FROM Customer C, Orders O

WHERE (C.CustNo = O.CustNo)

Tabel juga dapat digabungkan dengan menggunakan hasil penggabungan beberapa nilai kolom. Di sini, kolom ID1 dan ID2 dari tabel Joining digabungkan dan dibandingkan dengan nilai-nilai kolom tunggal ID tabel Source.

SELECT \*

FROM Source S, Joining J

WHERE (S.ID = J.ID1 || J.ID2)

2. **INNER JOIN**

*Syntax*:

**SELECT** column\_list

**FROM** table\_reference

[**INNER**] **JOIN** table\_reference **ON** predicate

[[**INNER**] **JOIN** table\_reference **ON** predicate...]

Fungsinya sama dengan operator Equi-Join hanya saja kolom untuk setiap tabel dibandingkan dalam klausa ON untuk nilai-nilai yang sama. Contoh pernyataan pertama dan kedua dalam operator Equi-Join, jika dideklarasikan dengan operator INNER JOIN menjadi :

SELECT \*

FROM Customer C

INNER JOIN Orders O ON (C.CustNo = O.CustNo)

SELECT \*

FROM Source S

INNER JOIN Joining J ON (S.ID = J.ID1 || J.ID2)

Untuk database Ms Access, pernyataan INNER JOIN harus menyertakan tanda kurung jika menggabungkan lebih dari dua tabel, seperti berikut ini

SELECT \*

FROM (Customer C

INNER JOIN Orders O ON (C.CustNo = O.CustNo))

INNER JOIN AddCusPrice A ON (C.CustNo = A.CustNo)

3. **OUTER JOIN**

*Syntax*:

**SELECT** column\_list

**FROM** table\_reference

**LEFT** | **RIGHT** | **FULL** [**OUTER**] **JOIN** table\_reference **ON** predicate

[**LEFT** | **RIGHT** | **FULL** [**OUTER**] **JOIN** table\_reference **ON** predicate...]

Fungsinya sama dengan operator Equi-Join dan INNER JOIN yaitu menggabungkan dua macam tabel. Perbedaannya, pada operator OUTER JOIN, *record* dari tabel sumber yang tidak cocok dengan tabel anggota akan dimunculkan dalam hasil. Kolom tabel anggota untuk *record* dalam tabel sumber yang tidak cocok mempunyai nilai kosong (bukan nol). Pernyataan berikut ini menggabungkan tabel Customer dan Orders berdasarkan nilai-nilai dalam kolom CustNo. Untuk *record* tabel Customer yang tidak mempunyai kecocokan nilai di antara Customer.CustNo dan Orders.CustNo, kolom dari tabel Orders berisi nilai kosong (bukan nol).

SELECT \*

FROM Customer C

LEFT OUTER JOIN Orders O ON (C.CustNo = O.CustNo)

Peubah LEFT menyebabkan semua *record* tabel pada sisi kiri operator OUTER JOIN dimasukkan dalam hasil, dengan atau tanpa kecocokan dalam tabel sebelah kanan. Jika tidak ada kecocokan *record* dari tabel sebelah kanan, maka kolom tersebut berisi nilai kosong. Gunakan peubah RIGHT jika menginginkan kondisi sebaliknya. Dan peubah FULL menyebabkan semua *record* dari seluruh tabel yang ditetapkan dalam klausa FROM dimasukkan dalam hasil, dengan atau tanpa kecocokan nilai. Tabel juga dapat digabung dengan mengunakan ekpresi yang menghasilkan suatu nilai tunggal sebagai predikat perbandingan. Di sini, kolom ID1 dan ID2 dari tabel Joining dibandingan secara terpisah dengan dua nilai yang dihasilkan oleh fungsi SUBSTRING dengan menggunakan kolom tunggal ID tabel Source.

SELECT \*

FROM Source S

RIGHT OUTER JOIN Joining J ON (SUBSTRING(S.ID FROM 1 FOR 2) = J.ID1)

AND (SUBSTRING(S.ID FROM 3 FOR 1) = J.ID2)

4. **Cartesian**

*Syntax*:

**SELECT** \*

**FROM** table\_reference, table\_reference [,table\_reference...]

Menggabungkan kolom dua buah tabel dalam satu hasil, tetapi tidak ada korelasi di antara *record* tabel. Operator Cartesian Join mencocokkan setiap *record* tabel sumber dengan setiap *record* tabel anggota. Jika tabel sumber mempunyai 10 *record* dan tabel anggota juga mempunyai 10 *record* maka hasilnya berisi 100 *record*.

SELECT \*

FROM "Employee.dbf", "Items.db"

5. **UNION**

*Syntax*:

**SELECT** col\_1 [, col\_2, ... col\_n]

**FROM** table\_reference

**UNION** [**ALL**]

**SELECT** col\_1 [, col\_2, ... col\_n]

**FROM** table\_reference

Menyusun *record* satu tabel pada akhir tabel yang lain. Pernyataan SELECT untuk tabel sumber dan anggota harus berisi kolom dengan jumlah yang sama.

SELECT

CustNo,

Company

FROM Customers

UNION

SELECT

CustNo,

Company

FROM Old\_Customers

Jika ada perbedaan tipe data kolom tertentu di antara dua tabel, suatu pesan kesalahan akan terjadi jika kolom yang sama dari tabel kedua (atau selanjutnya) kehilangan data. Sebagai contoh, kolom tabel pertama bertipe Date dan kolom tabel kedua bertipe TimeStamp, sebagian nilai bertipe TimeStamp akan hilang jika memasukkan kolom bertipe Date. Pesan kesalahan akan dibangkitkan untuk situasi ini. Gunakan fungsi CAST untuk mengubah tipe kolom tersebut.

SELECT

S.ID,

CAST(S.Date\_Field AS TIMESTAMP)

FROM Source S

UNION ALL

SELECT

J.ID,

J.Timestamp\_Field

FROM Joiner J

Secara default, operator UNION akan mengumpulkan *record* non-distinct dalam *record* tunggal. Gunakan ALL untuk mempertahankan *record* non-distinct. Untuk menggabungkan dua buah tabel dimana satu tabel tidak mempunyai suatu kolom tabel lain, gunakan suatu ekpresi atau literal yang sesuai. Sebagai contoh, tabel Joining tidak mempunyai kolom yang sesuai dengan kolom Name dari tabel Source, buat ekpresi untuk menampung nilai kolom bayangan Joining.Name. Kolom Source.Name diasumsikan bertipe CHAR(10).

SELECT

S.ID,

S.Name

FROM Source S

UNION ALL

SELECT

J.ID,

CAST("" AS CHAR(10))

FROM Joiner J

6. **Heterogeneous**

*Syntax*:

**SELECT** column\_list

**FROM** ":database\_reference:table\_reference", ":database\_reference:table\_reference"

[,":database\_reference:table\_reference"...]

**WHERE** predicate [**AND** predicate...]

Menggabungkan dua buah tabel dari database yang berbeda. Tabel yang digabung dapat bertipe dBASE ke Paradox atau Paradox ke InterBase, selama dapat diakses oleh BDE (local, ODBC, or SQL Links).

SELECT \*

FROM ":DBDEMOS:Customer.db" C, ":BCDEMOS:Orders.db" O

WHERE (C.CustNo = O.CustNo)

[**Operator dan Predikat Pernyataan SQL**](http://www.havetogether.com/operator-dan-predikat-pernyataan-sql.html)

By

[*rumono*](http://www.havetogether.com/author/rumono/)

– January 13, 2010**Posted in:** [Database](http://www.havetogether.com/category/database)

Pernyataan SQL, terutama Data Manipulation Language (DML), kadang-kadang mengunakan beberapa operator dan predikat untuk mendeklarasikan pernyataannya.

1. **Aritmatika**

*Syntax*:

numeric\_value1 + numeric\_value2

numeric\_value1 - numeric\_value2

numeric\_value1 \* numeric\_value2

numeric\_value1 / numeric\_value2

Digunakan untuk melakukan perhitungan aritmatika dalam query SELECT. Secara normal perhitungan aritmatika dilakukan mulai dari operasi perkalian, pembagian, penjumlahan, dan kemudian pengurangan. Gunakan tanda kurung untuk melalukan perhitungan di luar keadaan normal. Pernyataan berikut akan melakukan operasi penjumlahan sebelum melakukan operasi perkalian.

SELECT

(N.Numbers \* (N.Multiple + 1)) AS Result

FROM Numbertable N

Operator aritmatika hanya beroperasi pada data bertipe numerik, gunakan fungsi CAST untuk mengkonversi tipe data menjadi data bertipe numerik.

2. **Logika**

*Syntax*:

[**NOT**] predicate **OR** [**NOT**] predicate

[**NOT**] predicate **AND** [**NOT**] predicate

Operator OR dan NOT digunakan untuk menghubungkan dua predikat dalam klausa tunggal WHERE. Jika menggunakan operator OR, akan menghasilkan nilai True jika salah satu atau kedua prdikat bernilai True. Jika operator AND digunakan, untuk menghasilkan nilai True, kedua perdikat harus bernilai True. Pernyataan berikut akan menampilkan data (bernilai True) jika salah satu predikat dalam klausa WHERE bernilai True.

SELECT \*

FROM Reservations

WHERE ((ReserveDate < "1/31/1998") OR (Paid = TRUE))

Dan operator NOT berfungsi untuk menegasi nilai predikat. Pernyataan berikut hanya menampilkan baris dimana kolom Paid yang bernilai False.

SELECT \*

FROM Reservations

WHERE (NOT (Paid = "TRUE"))

3. **Predikat**  
3.1. **Comparison**

*Syntax*:

value1 < value2 kurang dari

value1 > value2 lebih dari

value1 = value2 sama dengan

value1 <> value2 tidak sama dengan

value1 != value2 tidak sama dengan (alternatif)

value1 >= value2 lebih besar dari atau sama dengan

value1 <= value2 kurang dari atau sama dengan

Membandingkan dua nilai (nilai kolom, literal, atau hasil perhitungan) yang sejenis. Hasil perbandingan dapat dimodifikasi dengan menggunakan operator logika, seperti NOT.

SELECT \*

FROM Orders

WHERE NOT (ItemsTotal >= 1000)

Perbandingan predikat hanya dapat digunakan dalam klausa WHERE atau HAVING, atau dalam klausa ON pada penggabungan tabel dan tidak dapat digunakan dalam pernyataan SELECT.

3.2. **BETWEEN**

*Syntax*:

value1 [**NOT**] **BETWEEN** value2 **AND** value3

Membandingkan suatu nilai dalam suatu range nilai. Jika nilai tersebut lebih besar atau sama dengan nilai terendah range dan kurang dari atau sama dengan nilai terbesar range maka BETWEEN bernilai True. Sebaliknya jika nilai tersebut lebih rendah dari nilai terendah range atau lebih besar dari nilai terbesar range maka BETWEEN bernilai False. BETWEEN dapat digunakan untuk semua data bertipe non-Blob, tetapi semua nilai yang dibandingkan harus bertipe sama atau sesuai. Untuk mengkonversi data gunakan fungsi CAST. Nilai yang digunakan dalam perbandingan BETWEEN dapat berupa nilai kolom, literal, atau nilai hasil perhitungan.

SELECT

Saledate

FROM Orders

WHERE (Saledate BETWEEN "1/1/1988" AND "12/31/1988")

3.3. **EXIST**

*Syntax*:

**EXISTS** subquery

Untuk menyaring tabel yang didasarkan pada keberadaan nilai kolom suatu tabel dalam subquery. Subquery dinyatakan dengan menggunakan klausa WHERE. EXIST bernilai True jika subquery telah menghasilkan setidaknya satu *record* data, dan bernilai False jika tidak ada *record* yang didapatkan. Subquery tersebut dapat disaring lagi dengan menggunakan kondisi-kondisi lain. Gunakan operator NOT untuk mengkonversi predikat pembanding EXIST. Sebagai contoh, pernyataan berikut menghasilkan *record* yang menyinggung semua pelanggan yang telah membukukan total pesanan melebihi $ 1,000.

SELECT

C.Company,

C.CustNo

FROM Customer C

WHERE EXISTS(SELECT O.CustNo

FROM Orders O

WHERE (O.CustNo = C.CustNo) AND (O.ItemsTotal > 1000))

3.4. **IN**

*Syntax*:

value [**NOT**] **IN** (value\_set)

Untuk menyaring tabel yang didasarkan pada keberadaan nilai kolom suatu tabel dalam suatu set nilai pembanding tertentu. Nilai kolom atau nilai literal atau nilai hasil perhitungan (termasuk kombinasinya) dapat digunakan sebagai nilai pembanding predikat IN. Nilai pembanding dapat berupa daftar pernyataan statis yang dipisahkan tanda koma.

SELECT

C.Company,

C.State

FROM Customer C

WHERE (C.State IN ("CA", "HI"))

Atau pernyataan dinamis yang dihasilkan melalui suatu subquery. Subquery dapat menghasilkan banyak baris, tetapi harus hanya menghasilkan kolom tunggal sebagai nilai pembanding.

SELECT

C.Company,

C.State

FROM Customer C

WHERE (C.State IN (SELECT R.State

FROM Regions R

WHERE (R.Region = "Pacific")))

Predikat IN dapat digunakan untuk semua data bertipe non-Blob, tetapi semua nilai yang dibandingkan harus bertipe sama atau sesuai. Untuk mengkonversi data gunakan fungsi CAST. Predikat IN berguna jika ingin menyaring data yang bernilai tidak ‘berdekatan’ (data noncontiguous). Dan gunakan predikat BETWEEN jika ingin menampilkan baris data bernilai mendekati range tertentu.

3.5. **LIKE**

*Syntax*:

value [**NOT**] **LIKE** [substitution\_char] comparison\_value [substitution\_char] [**ESCAPE** escape\_char]

Untuk menyaring tabel berdasarkan persamaan nilai kolom terhadap nilai pembanding. Karakter pengganti *wildcard* (“%”) dapat digunakan dalam perbandingan untuk mewakili jumlah karakter yang tidak diketahui. Predikat perbandingan LIKE bernilai True jika bagian nilai kolom cocok dengan posisi karakter wildcard. Posisi karakter *wildcard* bisa pada permulaan, pertengahan, atau akhir nilai perbandingan (atau berbagai kombinasi posisi tersebut). Sebagai contoh, pernyataan berikut akan menampilkan *record* dimana nilai kolom diawali huruf “A” tanpa memperhitungkan banyaknya karakter sesudah huruf tersebut.

SELECT \*

FROM Customer

WHERE (Company LIKE "A%")

Karakter pengganti garis bawah (“\_”) dapat digunakan dalam perbandingan untuk mewakili suatu karakter tunggal. Sebagai contoh, pernyataan berikut akan menampilkan *record* dimana nilai kolom diawali huruf “b” dan diakhiri huruf “n”, dengan sebuah karakter diantara nilai tersebut. nilai yang cocok misalnya “bin” atau “ban” tetapi bukan “barn”.

SELECT

Words

FROM Dictionary

WHERE (Words LIKE "b\_n")

Gunakan operator NOT untuk menegasi hasil perbandingan. Dan gunakan kata kunci ESCAPE untuk memunculkan karakter *wildcard* “%” atau “\_” sebagai data dalam kolom. Karakter yang mengikuti kata kunci ESCAPE diperlakukan sebagai suatu karakter lepas untuk memunculkan suatu karakter *wildcard* sebagai data. Contoh berikut menunjukkan karakter “^” sebagai karakter lepas dan karakter “%” diperlakukan sebagai data dalam kolom PercentValue. Jadi penyaringan data berdasarkan string “10%”.

SELECT \*

FROM Sales

WHERE (PercentValue LIKE "%10^%%" ESCAPE "^")

*Case-sensitive* (huruf besar/kecil mempunyai nilai berbeda) berlaku dalam predikat perbandingan ini. Dan hanya data bertipe Char yang dapat dinyatakan dalam predikat LIKE.

3.6. **IS NUL**

*Syntax*:

column\_reference **IS** [**NOT**] **NULL**

Predikat perbandingan IS NULL digunakan untuk menyaring tabel berdasarkan kolom tertentu yang berisi suatu nilai kosong (bukan nol). Untuk kolom bertipe numeric, nilai 0 (nol) tidak sama dengan nilai kosong. Gunakan operator NOT untuk menegasi hasil predikat perbandingan ini.

SELECT \*

FROM Customer

WHERE (InvoiceDate IS NULL)

3.7. **SOME/ANY/ALL**

*Syntax*:

column\_reference comparison\_predicate **SOME** | **ANY** | **ALL** (subquery)

Predikat perbandingan SOME, ANY, dan ALL digunakan untuk menyaring suatu tabel dengan membandingkan suatu nilai kolom terhadap beberapa nilai perbandingan. Predikat perbandingan ANY dan SOME bernilai True jika predikat perbandingan yang mengiringinya bernilai True untuk beberapa nilai subquery. Sebagai contoh, untuk menampilkan beberapa *record* dalam tabel Holding, nilai dalam kolom Pur\_Price hanya membutuhkan nilai yang lebih besar dari beberapa nilai yang dihasilkan dalam subquery kolom Price tabel Old\_Sales.

SELECT \*

FROM "Holdings.db" H

WHERE (H."Pur\_Price" > ANY (SELECT O."Price"

FROM "Old\_Sales.db"))

Predikat perbandingan ALL bernilai True jika predikat perbandingan yang mengiringinya bernilai True untuk semua nilai subquery. Sebagai contoh, untuk menampilkan beberapa *record* dalam tabel Holding, nilai dalam kolom Pur\_Price hanya membutuhkan nilai yang lebih besar dari setiap nilai yang dihasilkan dalam subquery kolom Price tabel Old\_Sales.

SELECT \*

FROM "Holdings.db" H

WHERE (H."Pur\_Price" > ALL (SELECT O."Price"

FROM "Old\_Sales.db"))