SQL Query Cheatsheet : Panduan Lengkap untuk Pemula

Disusun oleh Tim Datasans
https://datasans.medium.com/

Peringatan

Materi ini telah divalidasi, namun bagaimanapun juga, ebook ini tidak luput dari kesalahan baik definisi, konten secara umum, maupun syntax. Segala masukkan dari pengguna sangat terbuka. DM kami di instagram @datasans.book

Himbauan

1. Tidak menjadikan ebook ini satu-satunya sumber pegangan, *cross check* dan validasi segala informasi dari sumber lain.
2. Tidak membagikan atau mencetaknya untuk diperbanyak dan dikomersialkan (cetak untuk pribadi dipersilahkan).
3. Disarankan untuk merekomendasikan langsung ke instagram @datasans.book jika temanmu berminat agar ilmu yang bermanfaat bisa tersebar semakin luas.

Daftar Isi

Bab 1: Pendahuluan

1.1 Apa Itu SQL?

SQL atau Structured Query Language adalah bahasa pemrograman yang dirancang khusus untuk mengatur dan memanipulasi data dalam sistem manajemen basis data relasional (RDBMS), atau untuk aliran pemrosesan data dalam sistem manajemen basis data relasional aliran data (RDBMS). SQL merupakan standar internasional dalam melakukan operasi database, seperti meminta, memperbarui, atau menghapus data.

Pertama kali kamu mendengar istilah "SQL", mungkin yang terlintas dalam benakmu adalah kode-kode rumit yang hanya bisa dimengerti oleh programmer. Padahal, SQL sendiri sebenarnya cukup mudah dipelajari dan sangat berguna dalam berbagai bidang pekerjaan, terutama yang berhubungan dengan pengolahan data.

SQL dikembangkan pada akhir 1970-an dan awal 1980-an oleh IBM untuk dipakai di sistem basis data relasional mereka. Sejak saat itu, SQL telah menjadi standar dalam pengelolaan data relasional, dan telah digunakan oleh hampir semua RDBMS yang ada saat ini, seperti MySQL, Oracle, dan Microsoft SQL Server.

SQL memungkinkan kita untuk mengakses data dalam basis data relasional. SQL bisa digunakan untuk mengambil data, memasukkan baris baru, mengubah baris yang ada, atau menghapus baris dari tabel. Selain itu, SQL juga digunakan untuk membuat, mengubah, dan menghapus tabel atau seluruh database.

Sebagai bahasa, SQL memiliki sintaks dan struktur yang harus diikuti. SQL menggunakan perintah, seperti SELECT, INSERT, UPDATE, DELETE, dan lainnya untuk melakukan operasi pada data. Perintah-perintah ini digabungkan dalam query SQL yang kemudian dieksekusi oleh sistem basis data untuk melakukan tugas yang diminta.

SQL merupakan bahasa deklaratif, artinya kita memberi tahu sistem apa yang kita inginkan (yaitu, data apa yang kita inginkan), dan sistem basis data mencari cara terbaik untuk mendapatkannya. Ini berbeda dengan bahasa pemrograman prosedural, di mana kita harus memberi tahu sistem langkah demi langkah bagaimana mendapatkan apa yang kita inginkan.

Meski SQL merupakan standar, namun setiap RDBMS memiliki variasi dan ekstensi sendiri yang memperluas fungsi SQL. Misalnya, Oracle memiliki PL/SQL (Procedural Language/SQL), sementara PostgreSQL memiliki PL/pgSQL. Ekstensi-ekstensi ini memperluas SQL dengan menambahkan fitur seperti kontrol alur dan struktur data kompleks, yang memungkinkan kita untuk membuat aplikasi basis data yang lebih kompleks dan kuat.

Meski demikian, inti dari SQL tetap sama di semua sistem. Jadi, jika kamu sudah memahami dasar-dasar SQL, kamu akan bisa menerapkannya di hampir semua sistem basis data relasional.

Dalam pelajaran ini, kita akan membahas tentang SQL standar, yang akan bisa kamu gunakan di hampir semua sistem basis data relasional.

Semoga penjelasan ini membantu kamu memahami apa itu SQL dan mengapa penting untuk mempelajarinya. Selanjutnya, kita akan membahas mengapa SQL penting dan sistem manajemen basis data SQL yang populer.

1.2 Mengapa SQL Penting?

Sekarang kamu sudah memahami apa itu SQL, lalu mengapa SQL sangat penting untuk dipelajari dan dikuasai? Ada beberapa alasan mengapa SQL begitu penting dan relevan, bahkan setelah puluhan tahun sejak pertama kali diperkenalkan.

Data adalah Mata Uang Baru

Kita hidup di era informasi, di mana data telah menjadi mata uang baru. Data membantu perusahaan mengambil keputusan bisnis yang tepat berdasarkan analisis dan fakta, bukan asumsi. Dengan peningkatan teknologi dan prevalensi internet, jumlah data yang dihasilkan setiap harinya sangat besar. Perusahaan dan organisasi dari semua ukuran dan sektor sedang mencari cara untuk memanfaatkan data ini, dan disini lah peran SQL menjadi sangat penting.

SQL, sebagai bahasa yang digunakan untuk berinteraksi dengan data, memungkinkan kamu untuk mengekstraksi, memanipulasi, dan menganalisis data. Dengan kemampuan ini, kamu bisa membantu perusahaan menemukan insight dan tren yang mungkin tidak terlihat jika hanya melihat data mentah.

SQL adalah Standar Industri

Sebagaimana telah dijelaskan sebelumnya, SQL adalah standar dalam pengelolaan data relasional. Hampir semua sistem manajemen basis data relasional (RDBMS) menggunakan SQL atau variasinya. Jadi, jika kamu tahu SQL, kamu akan bisa bekerja dengan hampir semua basis data. Ini membuat skill SQL sangat berharga di pasar kerja.

SQL Bisa Digunakan di Berbagai Bidang

SQL bukan hanya untuk pengembang basis data. Berbagai profesi lain juga bisa memanfaatkan SQL untuk membantu pekerjaan mereka. Misalnya, analis data dan ilmuwan

data menggunakan SQL untuk mengekstraksi dan menganalisis data. Manajer produk dan manajer proyek bisa menggunakan SQL untuk melacak perkembangan proyek dan memastikan semua data akurat. Bahkan, profesional pemasaran bisa menggunakan SQL untuk menganalisis hasil kampanye pemasaran dan mencari cara untuk meningkatkan kinerja kampanye selanjutnya.

SQL Dapat Menggabungkan Data dari Sumber yang Berbeda

Salah satu kekuatan SQL adalah kemampuannya untuk menggabungkan data dari sumber yang berbeda. Dengan menggunakan perintah JOIN, kamu bisa menggabungkan data dari beberapa tabel berdasarkan kolom yang mereka miliki. Ini memungkinkan kamu untuk mendapatkan gambaran lengkap dan menyeluruh dari data.

SQL Membantu dalam Pembuatan dan Pengelolaan Basis Data

SQL tidak hanya digunakan untuk mengakses dan memanipulasi data. SQL juga digunakan untuk membuat dan mengelola basis data. Dengan SQL, kamu bisa membuat tabel baru, mengubah struktur tabel yang ada, dan bahkan membuat database baru. Kamu juga bisa mengatur siapa saja yang bisa mengakses data dan apa yang mereka bisa lakukan dengan data tersebut.

SQL Membantu dalam Automasi dan Optimasi Proses

SQL juga digunakan dalam pembuatan stored procedures dan triggers, yang bisa membantu dalam automasi dan optimasi proses. Stored procedures adalah sekelompok perintah SQL yang disimpan di server dan bisa dijalankan kapan saja dengan memanggil nama procedure tersebut. Triggers adalah perintah SQL yang otomatis dijalankan oleh sistem basis data ketika kondisi tertentu terpenuhi.

SQL adalah Dasar untuk Teknologi Data yang Lebih Lanjut

Jika kamu berminat dalam bidang data science atau big data, menguasai SQL adalah langkah pertama yang penting. Meskipun ada alat dan teknologi lain yang bisa digunakan untuk mengolah data besar, banyak dari mereka yang masih memerlukan pengetahuan dasar SQL. Bahkan, beberapa dari teknologi ini, seperti Apache Hive, menggunakan dialek SQL mereka sendiri.

Dengan semua alasan ini, mudah untuk melihat mengapa SQL begitu penting dan mengapa penting untuk mempelajarinya. Dengan memahami SQL, kamu akan memiliki kemampuan yang berharga yang akan membantu kamu di banyak bidang pekerjaan. Tidak hanya itu, kamu juga akan memiliki dasar yang kuat untuk belajar teknologi data yang lebih lanjut.

1.3 Sistem Manajemen Database SQL Populer

Setelah memahami apa itu SQL dan mengapa SQL penting, saatnya kita membahas beberapa sistem manajemen database SQL populer yang banyak digunakan dalam industri. Pengetahuan tentang sistem-sistem ini penting, karena meski inti dari SQL tetap sama, namun setiap sistem memiliki fitur dan sintaks spesifik yang mungkin berbeda. Berikut ini beberapa sistem manajemen database SQL yang populer:

1.3.1 MySQL

MySQL adalah sistem manajemen basis data relasional (RDBMS) open source yang paling populer. MySQL digunakan oleh banyak aplikasi web populer, termasuk WordPress dan Drupal. MySQL juga menjadi bagian dari stack LAMP (Linux, Apache, MySQL, PHP), yang merupakan teknologi backend yang populer untuk aplikasi web.

Keuntungan menggunakan MySQL adalah fleksibilitas dan keandalannya. MySQL mendukung berbagai platform, termasuk Linux, Windows, dan Mac, dan menyediakan fitur-fitur yang memungkinkan kamu untuk mengoptimalkan performa database. MySQL juga mendukung stored procedures, triggers, dan tipe data lainnya yang memungkinkan kamu untuk membuat aplikasi basis data yang kompleks.

1.3.2 PostgreSQL

PostgreSQL adalah sistem manajemen basis data relasional (RDBMS) open source yang kuat dan fleksibel. PostgreSQL menawarkan banyak fitur yang tidak dimiliki oleh RDBMS lainnya, seperti kelas data geometri untuk mendukung informasi spasial (GIS), dan "commitment control" yang memungkinkan kamu untuk mengatur transaksi lebih detail.

Keuntungan lain dari PostgreSQL adalah komunitasnya yang aktif dan berdedikasi. Komunitas ini secara rutin merilis update dan peningkatan untuk PostgreSQL, dan memberikan dukungan luas bagi penggunanya.

1.3.3 Oracle Database

Oracle Database adalah sistem manajemen basis data relasional (RDBMS) yang banyak digunakan oleh perusahaan besar dan korporasi. Oracle Database terkenal dengan performa, keandalan, dan keamanannya yang tinggi, yang menjadikannya pilihan yang populer untuk aplikasi bisnis kritis.

Oracle Database mendukung PL/SQL, yang merupakan dialek SQL Oracle yang memperluas SQL standar dengan menambahkan fitur bahasa pemrograman prosedural. Ini

memungkinkan kamu untuk menulis kode yang lebih kompleks dan efisien, yang bisa meningkatkan performa aplikasi basis data kamu.

1.3.4 SQL Server

SQL Server adalah sistem manajemen basis data relasional (RDBMS) dari Microsoft. SQL Server populer di lingkungan Windows, dan banyak digunakan oleh perusahaan untuk berbagai aplikasi, mulai dari aplikasi bisnis kecil hingga solusi enterprise.

SQL Server mendukung Transact-SQL (T-SQL), yang merupakan dialek SQL Microsoft yang memperluas SQL standar dengan menambahkan fitur bahasa pemrograman prosedural, seperti kontrol aliran dan penanganan kesalahan.

1.3.5 SQLite

SQLite adalah sistem manajemen basis data relasional (RDBMS) yang unik karena tidak menggunakan server tradisional seperti RDBMS lainnya. Sebaliknya, SQLite menyimpan data dalam file tunggal yang bisa langsung diakses oleh aplikasi. Ini membuat SQLite sangat ringan dan ideal untuk aplikasi mobile, aplikasi desktop, dan aplikasi lainnya di mana kecil dan efisien adalah prioritas.

Setiap sistem manajemen database SQL ini memiliki keunggulan dan kekurangannya masing-masing. Pemilihan sistem manajemen basis data SQL terbaik tergantung pada kebutuhan spesifik aplikasi dan lingkungan kamu. Namun, dengan memahami perbedaan antara sistem-sistem ini, kamu akan lebih siap untuk membuat keputusan tersebut. Selanjutnya, kita akan masuk ke bab berikutnya di mana kita akan mulai membahas dasar-dasar SQL.

Bab 2. Dasar-dasar SQL

2.1 Tabel dan Relasi

Memahami tabel dan relasi adalah fondasi penting dalam mempelajari SQL, karena sistem manajemen basis data relasional (RDBMS) - sebagaimana namanya - bekerja dengan konsep relasi antara tabel-tabel data. Maka dari itu, bab ini akan membahas secara mendalam tentang tabel dan relasi dalam konteks SQL.

2.1.1 Pengertian Tabel

Dalam konteks basis data, tabel adalah struktur yang digunakan untuk menyimpan data. Tabel dalam database relasional mirip dengan tabel dalam konteks umum, seperti dalam spreadsheet Excel, di mana data disimpan dalam baris dan kolom. Dalam konteks SQL, baris dalam tabel biasa disebut 'record', sedangkan kolom disebut 'field'.

Sebagai contoh, kita bisa membayangkan sebuah tabel yang menyimpan data pegawai dalam sebuah perusahaan. Tabel tersebut bisa memiliki kolom seperti 'id_pegawai', 'nama', 'jabatan', 'email', dan 'gaji'. Setiap baris dalam tabel tersebut akan mewakili satu pegawai, dengan data pegawai tersebut disimpan dalam kolom yang relevan.

Berikut adalah contoh tabel 'pegawai':

id_pegawai	nama	jabatan	email	gaji
1	Budi	Manager	budi@contoh.com	15000000
2	Sari	Staff	sari@contoh.com	8000000
3	Agus	Supervisor	agus@contoh.com	10000000
4	Dian	Staff	dian@contoh.com	8000000

Tabel di atas memiliki empat baris (atau record), dan masing-masing baris memiliki lima kolom (atau field). Setiap field memiliki nama dan tipe data tertentu.

2.1.2 Pengertian Relasi

Dalam konteks RDBMS, relasi merujuk pada hubungan antara dua atau lebih tabel. Relasi antara tabel biasanya dibuat melalui kolom yang sama atau kolom yang memiliki hubungan logis satu sama lain.

Misalkan kita memiliki tabel kedua yaitu 'departemen' yang menyimpan data tentang departemen dalam perusahaan tersebut. Tabel tersebut bisa memiliki kolom seperti 'id_departemen' dan 'nama_departemen'.

id_departemen	nama_departemen
1	HR
2	IT
3	Marketing

Kita bisa membuat relasi antara tabel 'pegawai' dan 'departemen' dengan menambahkan kolom 'id_departemen' ke tabel 'pegawai'. Setiap pegawai akan memiliki id_departemen yang mengacu pada departemen di mana mereka bekerja.

id_pegawai	nama	jabatan	email	gaji	id_departemen
1	Budi	Manager	budi@contoh.com	15000000	1
2	Sari	Staff	sari@contoh.com	8000000	2
3	Agus	Supervisor	agus@contoh.com	10000000	2
4	Dian	Staff	dian@contoh.com	8000000	3

Dengan tabel dan relasi ini, kita bisa melihat bahwa Budi bekerja di departemen HR, Sari dan Agus bekerja di departemen IT, dan Dian bekerja di departemen Marketing. Dengan demikian, kita bisa melakukan query SQL yang kompleks untuk mendapatkan insight dari data kita, seperti "Berapa total gaji untuk setiap departemen?" atau "Siapa saja yang bekerja di departemen IT?".

Jadi, memahami tabel dan relasi sangat penting dalam mempelajari SQL. Dengan memahami dua konsep ini, kamu akan dapat merancang dan bekerja dengan database relasional dengan lebih efisien dan efektif. Di bab berikutnya, kita akan membahas tentang tipe data dalam SQL.

2.2 Data Types dalam SQL

Saat mendefinisikan struktur tabel dalam SQL, kamu harus menentukan jenis data untuk setiap kolom. SQL mendukung berbagai jenis data, yang memungkinkan kamu untuk menyimpan data seperti teks, nomor, tanggal dan waktu, dan bahkan file biner. Menggunakan jenis data yang tepat sangat penting, karena itu menentukan bagaimana data disimpan, berapa banyak ruang yang digunakan, dan bagaimana data dapat dimanipulasi.

Berikut adalah penjelasan tentang jenis data umum dalam SQL dan bagaimana menggunakannya.

2.2.1 Data Types Teks

Jenis data teks digunakan untuk menyimpan data karakter. Jenis data teks umum yang digunakan dalam SQL antara lain:

- CHAR: Untuk string dengan panjang tetap. Misalnya, CHAR(5) akan menyimpan string dengan panjang tepat 5 karakter. Jika kamu memasukkan string dengan panjang kurang dari 5, sistem akan secara otomatis menambahkan spasi di akhir string hingga mencapai 5 karakter.
- VARCHAR: Untuk string dengan panjang variabel. Misalnya, VARCHAR(5) akan menyimpan string dengan panjang hingga 5 karakter, tapi tidak memerlukan spasi tambahan jika stringnya kurang dari 5 karakter.
- TEXT: Untuk string dengan panjang yang sangat besar. Biasanya digunakan untuk menyimpan teks panjang seperti artikel atau deskripsi produk.

Contoh penggunaan data types teks: kolom 'nama' dan 'email' dalam tabel 'pegawai' yang kita bahas di subbab 2.1.

2.2.2 Data Types Numerik

Jenis data numerik digunakan untuk menyimpan angka. Beberapa jenis data numerik umum dalam SQL antara lain:

- INTEGER: Untuk menyimpan bilangan bulat. Misalnya, 1, 2, 3, dan seterusnya.
- DECIMAL atau NUMERIC: Untuk menyimpan bilangan dengan desimal, dengan jumlah digit total dan jumlah digit setelah desimal yang ditentukan. Misalnya, DECIMAL(5,2) bisa menyimpan angka hingga 5 digit, dengan 2 digit di belakang koma.
- FLOAT atau REAL: Untuk menyimpan angka dengan desimal, dengan presisi yang bisa berubah-ubah.

Contoh penggunaan data types numerik: kolom 'gaji' dalam tabel 'pegawai', dimana kita bisa menggunakan DECIMAL(10,2) untuk menyimpan gaji pegawai.

2.2.3 Data Types Tanggal dan Waktu

Jenis data tanggal dan waktu digunakan untuk menyimpan nilai tanggal, waktu, atau keduanya. Beberapa jenis data tanggal dan waktu umum dalam SQL antara lain:

- DATE: Untuk menyimpan tanggal, dengan format YYYY-MM-DD.
- TIME: Untuk menyimpan waktu, dengan format HH:MI:SS.
- DATETIME atau TIMESTAMP: Untuk menyimpan tanggal dan waktu.

Contoh penggunaan data types tanggal dan waktu: kita bisa menambahkan kolom 'tanggal_masuk' dengan jenis data DATE ke tabel 'pegawai' untuk mencatat tanggal masuk pegawai.

2.2.4 Data Types Lainnya

Selain jenis data di atas, SQL juga mendukung jenis data lain seperti:

- BINARY dan VARBINARY: Untuk menyimpan data biner, seperti file gambar atau dokumen.
- BOOLEAN: Untuk menyimpan nilai boolean, yaitu TRUE atau FALSE.
- ENUM: Untuk menyimpan nilai dari daftar nilai yang ditentukan sebelumnya.

Memahami jenis data dalam SQL sangat penting untuk merancang dan bekerja dengan database SQL. Dengan menggunakan jenis data yang tepat, kamu bisa memastikan bahwa data kamu disimpan dengan efisien dan aman, dan kamu bisa memanipulasi data tersebut dengan mudah dan akurat. Di bab berikutnya, kita akan membahas tentang primary key dan foreign key dalam SQL.

2.3 Primary Key dan Foreign Key

Dalam konteks basis data relasional, Primary Key dan Foreign Key adalah dua konsep kunci yang digunakan untuk menjaga integritas data dan membangun relasi antara tabel. Mereka membantu dalam memastikan bahwa data dalam tabel selalu akurat dan dapat diandalkan.

2.3.1 Primary Key

Primary Key adalah kolom atau set kolom dalam tabel yang digunakan untuk mengidentifikasi secara unik setiap baris dalam tabel tersebut. Tidak ada dua baris dalam tabel yang dapat memiliki nilai Primary Key yang sama. Selain itu, setiap tabel biasanya memiliki satu Primary Key.

Ada beberapa aturan yang harus dipenuhi oleh Primary Key:

- Nilai Primary Key harus unik untuk setiap baris dalam tabel.
- Primary Key tidak boleh bernilai NULL.
- Sebuah tabel hanya bisa memiliki satu Primary Key.

Sebagai contoh, dalam tabel 'pegawai' yang kita bahas sebelumnya, kolom 'id_pegawai' bisa menjadi Primary Key. Setiap pegawai memiliki id yang unik, dan tidak ada dua pegawai yang bisa memiliki id yang sama.

2.3.2 Foreign Key

Foreign Key adalah kolom atau set kolom dalam satu tabel yang digunakan untuk mengacu ke Primary Key dalam tabel lain. Penggunaan Foreign Key memungkinkan kita untuk membuat relasi antara tabel.

Aturan tentang Foreign Key meliputi:

- Nilai dalam kolom Foreign Key harus sesuai dengan nilai dalam kolom Primary Key dari tabel yang direferensikannya, atau bernilai NULL.
- Jika nilai Primary Key yang direferensikan oleh Foreign Key diubah atau dihapus (tergantung pada aturan yang ditetapkan), nilai Foreign Key juga harus diubah atau dihapus.

Dalam contoh tabel 'pegawai' dan 'departemen' kita, kolom 'id_departemen' dalam tabel 'pegawai' adalah Foreign Key yang merujuk ke 'id_departemen' dalam tabel 'departemen'. Ini memungkinkan kita untuk mengetahui departemen mana yang menjadi tempat bekerja setiap pegawai.

Mengapa Primary Key dan Foreign Key Penting?

Primary Key dan Foreign Key sangat penting dalam mendesain database relasional. Mereka memungkinkan kita untuk membuat relasi antara tabel, yang kemudian memungkinkan kita untuk melakukan query SQL yang kompleks dan mendapatkan insight dari data kita.

Selain itu, Primary Key dan Foreign Key juga berfungsi untuk menjaga integritas data. Mereka memastikan bahwa tidak ada data yang berulang atau kontradiktif dalam database kita. Misalnya, dengan menggunakan Foreign Key, kita bisa memastikan bahwa setiap pegawai harus terdaftar ke departemen yang valid, dan kita tidak bisa secara tidak sengaja menghapus departemen jika masih ada pegawai yang terdaftar di departemen tersebut.

Berikut adalah contoh penggunaan Primary Key dan Foreign Key dalam bentuk tabel.

Tabel: Pegawai

id_pegawai (PK)	nama	jabatan	email	gaji	id_departemen (FK)
1	Budi	Manager	budi@contoh.com	15000000	1
2	Sari	Staff	sari@contoh.com	8000000	2
3	Agus	Supervisor	agus@contoh.com	10000000	2
4	Dian	Staff	dian@contoh.com	8000000	3

PK: Primary Key
FK: Foreign Key

Dalam tabel 'Pegawai' di atas, 'id_pegawai' adalah Primary Key. Setiap pegawai memiliki ID yang unik, dan tidak ada dua pegawai yang bisa memiliki ID yang sama.

Tabel: Departemen

id_departemen (PK)	nama_departemen
1	HR
2	IT
3	Marketing

Dalam tabel 'Departemen', 'id_departemen' adalah Primary Key. Setiap departemen memiliki ID yang unik.

Kembali ke tabel 'Pegawai', 'id_departemen' adalah Foreign Key yang mengacu pada 'id_departemen' di tabel 'Departemen'. Ini memungkinkan kita untuk mengetahui departemen mana yang menjadi tempat bekerja setiap pegawai.

Dengan cara ini, kita dapat menjaga integritas data dan membangun relasi antar tabel. Misalnya, kita tidak bisa menambahkan pegawai dengan 'id_departemen' yang tidak ada di tabel 'Departemen', dan kita tidak bisa menghapus departemen dari tabel 'Departemen' jika masih ada pegawai yang bekerja di departemen tersebut di tabel 'Pegawai'.

2.4 Null Values

Dalam database SQL, NULL merupakan sebuah nilai khusus yang digunakan untuk merepresentasikan data yang tidak diketahui atau tidak ada. Kamu mungkin bertanya, "bukankah bisa saja kita mengisi kolom tersebut dengan 'N/A' atau 0 atau string kosong?" Ya, memang bisa, tetapi penggunaan NULL memiliki arti yang lebih spesifik dan lebih akurat dalam konteks basis data.

Perlu diingat bahwa NULL berbeda dari 0 atau string kosong. Jika sebuah kolom diisi dengan 0, itu berarti kita tahu bahwa nilai tersebut adalah 0. Jika sebuah kolom diisi dengan string kosong, itu berarti kita tahu bahwa kolom tersebut tidak memiliki isi. Namun, jika sebuah kolom diisi dengan NULL, itu berarti kita tidak tahu apa nilai sebenarnya dari kolom tersebut.

Berikut beberapa poin penting terkait NULL dalam SQL:

- NULL dalam Konteks Kolom: Dalam mendefinisikan sebuah tabel, kamu bisa menetapkan apakah sebuah kolom bisa berisi NULL atau tidak. Jika kamu

menetapkan bahwa sebuah kolom tidak boleh NULL (dengan menggunakan kata kunci NOT NULL saat mendefinisikan kolom), maka kamu harus menyertakan nilai untuk kolom tersebut setiap kali memasukkan baris baru ke dalam tabel.

- Penggunaan NULL dalam Query: Ketika menulis query SQL, kamu perlu berhati-hati saat menangani NULL. Sebagian besar operasi yang melibatkan NULL akan menghasilkan NULL. Misalnya, jika kamu mencoba untuk menambahkan NULL dengan angka, hasilnya akan NULL. Jika kamu mencoba untuk membandingkan nilai dengan NULL (seperti nilai = NULL), hasilnya akan NULL, bukan TRUE atau FALSE. Untuk memeriksa apakah nilai NULL atau tidak, kamu harus menggunakan kata kunci IS NULL atau IS NOT NULL.
- NULL dalam Konteks Primary Key dan Foreign Key: Kolom yang digunakan sebagai Primary Key tidak boleh berisi NULL, karena nilai Primary Key harus unik dan diketahui untuk setiap baris. Sementara itu, kolom yang digunakan sebagai Foreign Key bisa berisi NULL, kecuali jika kamu menetapkan bahwa kolom tersebut harus berisi referensi yang valid (dengan menggunakan kata kunci FOREIGN KEY REFERENCES ... ON DELETE NO ACTION atau ON UPDATE NO ACTION).

Mari kita lihat contoh penggunaan NULL dalam konteks tabel 'Pegawai' dan 'Departemen' yang telah kita buat sebelumnya.

Misalkan kita menambahkan kolom 'tanggal_keluar' ke tabel 'Pegawai' untuk mencatat tanggal seorang pegawai keluar atau resign dari perusahaan. Kolom ini akan berisi NULL untuk pegawai yang masih bekerja di perusahaan.

Tabel: Pegawai (dengan kolom baru 'tanggal_keluar')

id_pegawai (PK)	nama	jabatan	email	gaji	id_departemen (FK)	tanggal_masuk	tanggal_keluar
1	Budi	Manager	budi@contoh.com	15000000	1	2020-01-10	NULL
2	Sari	Staff	sari@contoh.com	8000000	2	2020-03-15	2023-01-15
3	Agus	Supervisor	agus@contoh.com	10000000	2	2020-06-01	NULL
4	Dian	Staff	dian@contoh.com	8000000	3	2020-08-20	NULL

Dalam contoh di atas, Budi, Agus, dan Dian masih bekerja di perusahaan, sehingga 'tanggal_keluar' mereka adalah NULL. Sementara itu, Sari telah keluar dari perusahaan pada tanggal 15 Januari 2023, sehingga 'tanggal_keluar' Sari adalah 2023-01-15.

Dengan pemahaman yang baik tentang NULL, kamu bisa lebih efektif dalam menangani data yang tidak diketahui atau tidak ada dalam database SQL kamu.

Bab 3. Manipulasi Data Dengan SQL

3.1 Menggunakan SELECT

Perintah SELECT adalah perintah fundamental dalam SQL yang digunakan untuk mengambil data dari database. Dengan perintah SELECT, kamu bisa menentukan kolom apa yang ingin ditampilkan dan dari tabel mana data tersebut diambil.

Mari kita lihat contoh konkretnya menggunakan dua tabel yang telah kita buat sebelumnya, yaitu tabel 'Pegawai' dan tabel 'Departemen'.

Tabel Original: Pegawai

id_pegawai	nama	jabatan	email	gaji	id_departemen
1	Budi	Manager	budi@mail.com	10000	1
2	Sari	Supervisor	sari@mail.com	8000	2
3	Agus	Staff	agus@mail.com	6000	2
4	Dian	Staff	dian@mail.com	6000	3

Tabel Original: Departemen

id_departemen	nama_departemen
1	HR
2	IT
3	Marketing

Dalam subbab ini, kita akan fokus pada bagaimana menggunakan perintah SELECT untuk mengambil data dari tabel 'Pegawai'.

Contoh 1: Mengambil Semua Kolom

Misalkan kamu ingin melihat semua informasi dari tabel 'Pegawai'. Kamu bisa menggunakan tanda bintang (*) untuk menunjukkan semua kolom:

```
SELECT *
FROM Pegawai;
```

Tabel Output

id_pegawai	nama	jabatan	email	gaji	id_departemen
1	Budi	Manager	budi@mail.com	10000	1

2	Sari	Supervisor	sari@mail.com	8000	2
3	Agus	Staff	agus@mail.com	6000	2
4	Dian	Staff	dian@mail.com	6000	3

Contoh 2: Mengambil Kolom Tertentu

Jika kamu hanya tertarik pada nama dan jabatan pegawai, kamu bisa langsung menentukan kolom tersebut:

```
SELECT nama, jabatan
FROM Pegawai;
```

Tabel Output

nama	jabatan
Budi	Manager
Sari	Supervisor
Agus	Staff
Dian	Staff

Perintah SELECT adalah dasar dari banyak operasi SQL lainnya. Dengan menguasai perintah ini, kamu telah membuat langkah besar pertama dalam memanfaatkan kekuatan SQL untuk mengelola dan menganalisis data.

3.2 Menggunakan WHERE

Perintah WHERE dalam SQL digunakan untuk menyaring baris yang memenuhi kondisi tertentu. Ini memungkinkan kamu untuk mengambil data yang spesifik dari suatu tabel. Kombinasi antara SELECT dan WHERE adalah salah satu yang paling sering digunakan dalam pengolahan database.

Mari kita lihat contoh penggunaan perintah WHERE dengan tabel 'Pegawai' yang telah kita buat sebelumnya.

Tabel Original: Pegawai

id_pegawai	nama	jabatan	email	gaji	id_departemen
1	Budi	Manager	budi@mail.com	10000	1
2	Sari	Supervisor	sari@mail.com	8000	2
3	Agus	Staff	agus@mail.com	6000	2
4	Dian	Staff	dian@mail.com	6000	3

Contoh 1: Mengambil Data dengan Kondisi Tertentu

Misalkan kamu ingin mendapatkan informasi pegawai yang bekerja di departemen dengan id 2. Kamu bisa menggunakan perintah WHERE seperti ini:

```
SELECT *
FROM Pegawai
WHERE id_departemen = 2;
```

Tabel Output

id_pegawai	nama	jabatan	email	gaji	id_departemen
2	Sari	Supervisor	sari@mail.com	8000	2
3	Agus	Staff	agus@mail.com	6000	2

Contoh 2: Menggunakan WHERE dengan Beberapa Kondisi

Kamu juga bisa menggabungkan beberapa kondisi dalam perintah WHERE. Misalkan kamu ingin mengetahui pegawai dengan gaji di atas 6000 yang bekerja di departemen dengan id 2. Kamu bisa menggunakan operator AND atau OR untuk menggabungkan kondisi.

```
SELECT *
FROM Pegawai
WHERE gaji > 6000 AND id_departemen = 2;
```

Tabel Output

id_pegawai	nama	jabatan	email	gaji	id_departemen
2	Sari	Supervisor	sari@mail.com	8000	2

Perintah WHERE sangat penting dalam SQL, karena ia memungkinkan kamu untuk bekerja dengan subset data yang relevan.

3.3 Menggunakan DISTINCT

Perintah DISTINCT dalam SQL digunakan untuk menghilangkan duplikasi dalam hasil set yang dikembalikan oleh query. Ini sangat berguna jika kamu ingin mengetahui nilai unik dalam suatu kolom tanpa harus melihat duplikasi.

Mari kita lihat contoh penggunaannya dengan tabel 'Pegawai' yang telah kita gunakan sebelumnya.

Tabel Original: Pegawai

id_pegawai	nama	jabatan	email	gaji	id_departemen
1	Budi	Manager	budi@mail.com	10000	1
2	Sari	Supervisor	sari@mail.com	8000	2
3	Agus	Staff	agus@mail.com	6000	2
4	Dian	Staff	dian@mail.com	6000	3

Contoh 1: Mengambil Nilai Unik

Misalkan kamu ingin mengetahui jabatan apa saja yang ada dalam perusahaan tanpa duplikasi. Kamu bisa menggunakan perintah DISTINCT untuk itu:

```
SELECT DISTINCT jabatan
FROM Pegawai;
```

Tabel Output

jabatan
Manager
Supervisor
Staff

Contoh 2: Menggunakan DISTINCT pada Beberapa Kolom

DISTINCT juga bisa digunakan pada lebih dari satu kolom. Misalkan kamu ingin mengetahui kombinasi unik antara jabatan dan id_departemen:

```
SELECT DISTINCT jabatan, id_departemen
FROM Pegawai;
```

Tabel Output

jabatan	id_departemen
Manager	1
Supervisor	2
Staff	2
Staff	3

Perhatikan bahwa dalam contoh kedua, masih ada dua baris dengan jabatan 'Staff'. Ini karena meskipun jabatan mereka sama, id_departemen mereka berbeda, sehingga kombinasi jabatan dan id_departemen dianggap unik.

Perintah DISTINCT sangat berguna jika kamu ingin mengetahui nilai unik dalam suatu kolom atau beberapa kolom. Perlu diperhatikan, penggunaan DISTINCT harus dilakukan secara bijaksana karena bisa berdampak pada performa query, terutama pada tabel dengan ukuran yang sangat besar.

3.4 Menggunakan AND, OR, NOT

Dalam SQL, kamu bisa menggunakan operator logika AND, OR, dan NOT untuk menggabungkan atau memodifikasi kondisi dalam perintah WHERE. Mari kita lihat masing-masing operator ini dalam detail.

Mari kita gunakan tabel 'Pegawai' yang sama untuk memahami penggunaan AND, OR, dan NOT dalam SQL.

Tabel Original: Pegawai

id_pegawai	nama	jabatan	email	gaji	id_departemen
1	Budi	Manager	budi@mail.com	10000	1
2	Sari	Supervisor	sari@mail.com	8000	2
3	Agus	Staff	agus@mail.com	6000	2
4	Dian	Staff	dian@mail.com	6000	3

Contoh 1: Menggunakan AND

Operator AND memungkinkan kamu untuk menggabungkan dua atau lebih kondisi, dan baris yang memenuhi semua kondisi tersebut akan dikembalikan. Misalkan kamu ingin mengetahui pegawai yang memiliki jabatan 'Staff' dan bekerja di departemen dengan id 2.

```
SELECT *
FROM Pegawai
WHERE jabatan = 'Staff' AND id_departemen = 2;
```

Tabel Output

id_pegawai	nama	jabatan	email	gaji	id_departemen
3	Agus	Staff	agus@mail.com	6000	2

Contoh 2: Menggunakan OR

Operator OR memungkinkan kamu untuk menggabungkan dua atau lebih kondisi, dan baris yang memenuhi salah satu kondisi akan dikembalikan. Misalkan kamu ingin mengetahui pegawai yang jabatannya 'Manager' atau yang bekerja di departemen dengan id 2.

```
SELECT *
FROM Pegawai
WHERE jabatan = 'Manager' OR id_departemen = 2;
```

Tabel Output

id_pegawai	nama	jabatan	email	gaji	id_departemen
1	Budi	Manager	budi@mail.com	10000	1
2	Sari	Supervisor	sari@mail.com	8000	2
3	Agus	Staff	agus@mail.com	6000	2

Contoh 3: Menggunakan NOT

Operator NOT digunakan untuk membalikkan kondisi yang ditentukan. Misalkan kamu ingin mengetahui pegawai yang bukan merupakan 'Staff'.

```
SELECT *
FROM Pegawai
WHERE NOT jabatan = 'Staff';
```

Tabel Output

id_pegawai	nama	jabatan	email	gaji	id_departemen
1	Budi	Manager	budi@mail.com	10000	1
2	Sari	Supervisor	sari@mail.com	8000	2

Operator AND, OR, dan NOT sangat berguna dalam menentukan kondisi yang lebih kompleks dalam perintah WHERE.

3.5 Menggunakan ORDER BY

Perintah ORDER BY dalam SQL memungkinkan kamu untuk mengurutkan hasil yang dikembalikan oleh query berdasarkan satu atau lebih kolom. Kamu bisa mengurutkan data dalam urutan naik (ASC) atau turun (DESC).

Mari kita gunakan tabel 'Pegawai' yang sama untuk memahami penggunaan ORDER BY dalam SQL.

Tabel Original: Pegawai

id_pegawai	nama	jabatan	email	gaji	id_departemen
1	Budi	Manager	budi@mail.com	10000	1
2	Sari	Supervisor	sari@mail.com	8000	2
3	Agus	Staff	agus@mail.com	6000	2
4	Dian	Staff	dian@mail.com	6000	3

Contoh 1: Mengurutkan Data

Misalkan kamu ingin melihat daftar pegawai, tapi diurutkan berdasarkan gaji mereka dari yang terbesar ke terkecil.

```
SELECT *
FROM Pegawai
ORDER BY gaji DESC;
```

Tabel Output

id_pegawai	nama	jabatan	email	gaji	id_departemen
1	Budi	Manager	budi@mail.com	10000	1
2	Sari	Supervisor	sari@mail.com	8000	2
3	Agus	Staff	agus@mail.com	6000	2
4	Dian	Staff	dian@mail.com	6000	3

Contoh 2: Mengurutkan Data Berdasarkan Beberapa Kolom

Kamu juga bisa mengurutkan data berdasarkan lebih dari satu kolom. Misalkan kamu ingin melihat daftar pegawai, diurutkan berdasarkan jabatan dan kemudian berdasarkan gaji dalam setiap jabatan.

```
SELECT *
FROM Pegawai
ORDER BY jabatan, gaji DESC;
```

Tabel Output

id_pegawai	nama	jabatan	email	gaji	id_departemen
1	Budi	Manager	budi@mail.com	10000	1
2	Sari	Supervisor	sari@mail.com	8000	2
3	Agus	Staff	agus@mail.com	6000	2
4	Dian	Staff	dian@mail.com	6000	3

Perintah ORDER BY sangat berguna jika kamu ingin melihat data dalam urutan tertentu.

3.6 Menggunakan INSERT INTO

Ketika bekerja dengan database, penting bagi kamu untuk memahami bagaimana cara menambahkan data baru ke dalam tabel. Salah satu cara untuk melakukan ini adalah dengan menggunakan perintah INSERT INTO.

Perintah INSERT INTO digunakan untuk memasukkan data baru ke dalam tabel. Data baru ini disisipkan ke dalam setiap kolom tabel dalam urutan yang ditentukan oleh skema tabel.

Berikut contoh tabel dan bagaimana menggunakan perintah INSERT INTO:

Tabel Original: Pegawai

id_pegawai	nama	jabatan	email	gaji	id_departemen
1	Budi	Manager	budi@mail.com	10000	1
2	Sari	Supervisor	sari@mail.com	8000	2
3	Agus	Staff	agus@mail.com	6000	2
4	Dian	Staff	dian@mail.com	6000	3

Contoh: Menambahkan Data Baru ke Tabel

Misalkan kita ingin menambahkan pegawai baru ke dalam tabel Pegawai. Pegawai baru ini bernama Rina, berjabatan sebagai Manager, emailnya rina@mail.com, mendapatkan gaji sebesar 11000, dan bekerja di departemen 1.

```
INSERT INTO Pegawai (id_pegawai, nama, jabatan, email, gaji,
id_departemen)
VALUES (5, 'Rina', 'Manager', 'rina@mail.com', 11000, 1);
```

Tabel Output

id_pegawai	nama	jabatan	email	gaji	id_departemen
1	Budi	Manager	budi@mail.com	10000	1
2	Sari	Supervisor	sari@mail.com	8000	2
3	Agus	Staff	agus@mail.com	6000	2
4	Dian	Staff	dian@mail.com	6000	3
5	Rina	Manager	rina@mail.com	11000	1

Perhatikan bahwa kolom id_pegawai sudah diisi dengan 5. Biasanya, dalam aplikasi nyata, id_pegawai akan diisi secara otomatis oleh sistem (biasa disebut auto-increment).

3.7 Mengolah NULL Values

Dalam database, NULL adalah istilah yang digunakan untuk menggambarkan nilai yang tidak ada atau tidak diketahui. Dalam SQL, NULL berarti "tidak ada nilai" atau "nilai kosong". Perlu diingat bahwa NULL berbeda dari nol atau kolom yang berisi spasi. Nol adalah nilai, seperti juga spasi atau string kosong, sedangkan NULL berarti "tidak ada nilai sama sekali".

Menangani NULL Values bisa jadi sedikit rumit karena perilakunya yang unik dalam perhitungan dan logika.

Misalnya, kamu memiliki tabel pegawai sebagai berikut:

Tabel Original: Pegawai

id_pegawai	nama	jabatan	email	gaji	id_departemen
1	Budi	Manager	budi@mail.com	10000	1
2	Sari	Supervisor	sari@mail.com	8000	NULL
3	Agus	Staff	agus@mail.com	6000	2
4	Dian	NULL	dian@mail.com	6000	3
5	Rina	Manager	rina@mail.com	NULL	1

Dalam tabel ini, Sari tidak memiliki id_departemen, jabatan Dian tidak diketahui, dan gaji Rina tidak diketahui. Dalam semua kasus ini, nilai dalam tabel tersebut adalah NULL.

Contoh: Menggunakan IS NULL

Jika kamu ingin mencari pegawai yang gajinya tidak diketahui, kamu bisa menggunakan pernyataan IS NULL.

```
SELECT *
FROM Pegawai
WHERE gaji IS NULL;
```

Tabel Output

id_pegawai	nama	jabatan	email	gaji	id_departemen
5	Rina	Manager	rina@mail.com	NULL	1

Contoh: Menggunakan IS NOT NULL

Sebaliknya, jika kamu ingin mencari pegawai yang gajinya diketahui, kamu bisa menggunakan pernyataan IS NOT NULL.

```
SELECT *
FROM Pegawai
WHERE gaji IS NOT NULL;
```

Tabel Output

id_pegawai	nama	jabatan	email	gaji	id_departemen
1	Budi	Manager	budi@mail.com	10000	1
2	Sari	Supervisor	sari@mail.com	8000	NULL
3	Agus	Staff	agus@mail.com	6000	2
4	Dian	NULL	dian@mail.com	6000	3

3.8 Menggunakan UPDATE

Dalam manajemen database SQL, terdapat situasi dimana kita perlu memperbarui data yang ada di dalam tabel. Operasi ini sangat penting dan biasa terjadi dalam operasi sehari-hari. SQL menyediakan perintah UPDATE untuk memperbarui data di dalam tabel. Perintah ini dapat digunakan untuk memperbarui satu atau lebih kolom pada satu atau lebih baris.

Misalkan kamu memiliki tabel Pegawai sebagai berikut:

Tabel Original: Pegawai

id_pegawai	nama	jabatan	email	gaji
1	Budi	Manager	budi@mail.com	10000
2	Sari	Supervisor	sari@mail.com	8000
3	Agus	Staff	agus@mail.com	6000
4	Dian	Staff	dian@mail.com	6000
5	Rina	Manager	rina@mail.com	12000

Contoh 1: Update Gaji untuk Satu Pegawai

Misalnya, kamu ingin memberikan kenaikan gaji untuk Agus menjadi 6500. Dalam kasus ini, kamu dapat menggunakan pernyataan UPDATE untuk memperbarui kolom gaji untuk Agus.

```
UPDATE Pegawai
```

```
SET gaji = 6500
WHERE id_pegawai = 3;
```

Tabel Output

id_pegawai	nama	jabatan	email	gaji
1	Budi	Manager	budi@mail.com	10000
2	Sari	Supervisor	sari@mail.com	8000
3	Agus	Staff	agus@mail.com	6500
4	Dian	Staff	dian@mail.com	6000
5	Rina	Manager	rina@mail.com	12000

Contoh 2: Update Gaji untuk Semua Staff

Misalnya, kamu ingin memberikan kenaikan gaji sebesar 500 untuk semua staff. Dalam kasus ini, kamu dapat menggunakan pernyataan UPDATE untuk memperbarui kolom gaji untuk semua staff.

```
UPDATE Pegawai
SET gaji = gaji + 500
WHERE jabatan = 'Staff';
```

Tabel Output

id_pegawai	nama	jabatan	email	gaji
1	Budi	Manager	budi@mail.com	10000
2	Sari	Supervisor	sari@mail.com	8000
3	Agus	Staff	agus@mail.com	7000
4	Dian	Staff	dian@mail.com	6500
5	Rina	Manager	rina@mail.com	12000

3.9 Menggunakan DELETE

Tidak semua data yang ada dalam database akan selalu relevan atau diperlukan. Ada kalanya kamu perlu menghapus beberapa baris data yang sudah tidak relevan atau mungkin karena adanya duplikasi data. Dalam SQL, kamu bisa melakukan ini dengan menggunakan perintah DELETE.

Contoh 1: Menghapus Satu Baris Data

Misalkan kamu memiliki tabel berikut:

Tabel Original: Pegawai

id_pegawai	nama	jabatan	email	gaji
1	Budi	Manager	budi@mail.com	10000
2	Sari	Supervisor	sari@mail.com	8000
3	Agus	Staff	agus@mail.com	6000
4	Dian	Staff	dian@mail.com	6000
5	Rina	Manager	rina@mail.com	10000

Misalkan kamu mendapatkan informasi bahwa pegawai dengan id_pegawai 4 sudah tidak bekerja lagi di perusahaan tersebut dan data tersebut perlu dihapus. Dalam kasus ini, kamu dapat menggunakan pernyataan DELETE sebagai berikut:

```
DELETE FROM Pegawai
WHERE id_pegawai = 4;
```

Tabel Output

id_pegawai	nama	jabatan	email	gaji
1	Budi	Manager	budi@mail.com	10000
2	Sari	Supervisor	sari@mail.com	8000
3	Agus	Staff	agus@mail.com	6000
5	Rina	Manager	rina@mail.com	10000

Contoh 2: Menghapus Semua Data Dengan Kondisi Tertentu

Misalkan sekarang kamu ingin menghapus semua data pegawai yang memiliki jabatan 'Staff'. Dalam kasus ini, kamu dapat menggunakan pernyataan DELETE sebagai berikut:

```
DELETE FROM Pegawai
WHERE jabatan = 'Staff';
```

Tabel Output

id_pegawai	nama	jabatan	email	gaji
1	Budi	Manager	budi@mail.com	10000
2	Sari	Supervisor	sari@mail.com	8000
5	Rina	Manager	rina@mail.com	10000

Perlu diingat, berhati-hatilah saat menggunakan pernyataan DELETE karena jika kamu lupa menambahkan klausa WHERE, SQL akan menghapus semua baris dari tabel tersebut, yang bisa berakibat fatal.

Bab 4. Fungsi SQL

4.1 SQL Aggregate Functions

Dalam memanipulasi data, kadang kamu membutuhkan perhitungan atau operasi yang melibatkan sekelompok data, bukan hanya data per baris. Dalam SQL, ada sekelompok fungsi yang disebut fungsi agregat, yang digunakan untuk melakukan operasi pada sekumpulan nilai dan mengembalikan satu nilai. Dalam subbab ini, kita akan membahas lima fungsi agregat paling umum, yaitu COUNT, AVG, SUM, MAX, dan MIN.

Misalkan kita memiliki tabel sebagai berikut:

Tabel Original: Pegawai

id_pegawai	nama	jabatan	email	gaji
1	Budi	Manager	budi@mail.com	10000
2	Sari	Supervisor	sari@mail.com	8000
3	Agus	Staff	agus@mail.com	6000
4	Dian	Staff	dian@mail.com	6000
5	Rina	Manager	rina@mail.com	10000

1. COUNT

Fungsi COUNT menghitung jumlah baris yang memenuhi kriteria tertentu.

Misalnya, kamu ingin mengetahui berapa banyak pegawai yang bekerja sebagai 'Manager'. Kamu dapat menggunakan fungsi COUNT seperti berikut:

```
SELECT COUNT(*) AS JumlahManager
FROM Pegawai
WHERE jabatan = 'Manager';
```

Tabel Output

JumlahManager
2

2. AVG

Fungsi AVG menghitung rata-rata dari sekumpulan nilai.

Misalnya, kamu ingin mengetahui gaji rata-rata dari semua pegawai. Kamu dapat menggunakan fungsi AVG seperti berikut:

```
SELECT AVG(gaji) AS RataRataGaji
FROM Pegawai;
```

Tabel Output

RataRataGaji
8400

3. SUM

Fungsi SUM menghitung total dari sekumpulan nilai.

Misalnya, kamu ingin mengetahui total gaji yang dibayarkan kepada semua pegawai. Kamu dapat menggunakan fungsi SUM seperti berikut:

```
SELECT SUM(gaji) AS TotalGaji
FROM Pegawai;
```

Tabel Output

TotalGaji
42000

4. MAX

Fungsi MAX menemukan nilai maksimum dalam sekumpulan nilai.

Misalnya, kamu ingin mengetahui gaji tertinggi yang diterima oleh pegawai. Kamu dapat menggunakan fungsi MAX seperti berikut:

```
SELECT MAX(gaji) AS GajiTertinggi
FROM Pegawai;
```

Tabel Output

GajiTertinggi
10000

5. MIN

Fungsi MIN menemukan nilai minimum dalam sekumpulan nilai.

Misalnya, kamu ingin mengetahui gaji terendah yang diterima oleh pegawai. Kamu dapat menggunakan fungsi MIN seperti berikut:

```
SELECT MIN(gaji) AS GajiTerendah
FROM Pegawai;
```

Tabel Output

GajiTerendah
6000

Fungsi agregat sangat berguna dalam melakukan analisis data pada SQL. Kamu dapat menggunakan fungsi-fungsi ini untuk mendapatkan insight dari data yang kamu miliki, seperti mencari nilai rata-rata, total, maksimum, minimum, atau menghitung jumlah data.

4.2 SQL Scalar functions

Fungsi skalar adalah fungsi yang bekerja pada setiap baris dan menghasilkan satu hasil untuk setiap baris. Mereka berbeda dari fungsi agregat yang menghasilkan satu hasil untuk banyak baris. Dalam subbab ini, kita akan membahas beberapa fungsi skalar yang paling umum digunakan dalam SQL, yaitu UCASE, LCASE, MID, LEN, ROUND, NOW, dan FORMAT.

Misalkan kita memiliki tabel berikut:

Tabel Original: Pegawai

id_pegawai	nama	jabatan	email	gaji
1	Budi	Manager	budi@mail.com	10000
2	Sari	Supervisor	sari@mail.com	8000
3	Agus	Staff	agus@mail.com	6000
4	Dian	Staff	dian@mail.com	6000
5	Rina	Manager	rina@mail.com	10000

1. UCASE

Fungsi UCASE mengubah semua karakter dalam string menjadi huruf besar.

Misalkan kamu ingin mengubah semua nama pegawai menjadi huruf besar. Kamu bisa menggunakan fungsi UCASE seperti berikut:

```
SELECT UCASE(nama) AS Nama
```

```
FROM Pegawai;
```

Tabel Output

Nama
BUDI
SARI
AGUS
DIAN
RINA

2. LCASE

Fungsi LCASE mengubah semua karakter dalam string menjadi huruf kecil.

Misalkan kamu ingin mengubah semua nama pegawai menjadi huruf kecil. Kamu bisa menggunakan fungsi LCASE seperti berikut:

```
SELECT LCASE(nama) AS Nama
FROM Pegawai;
```

Tabel Output

Nama
budi
sari
agus
dian
rina

3. MID

Fungsi MID digunakan untuk memilih substring dengan panjang tertentu dari suatu string. Fungsi ini memerlukan tiga argumen: string sumber, posisi awal, dan panjang substring.

Misalkan kamu ingin memilih tiga karakter pertama dari setiap nama pegawai. Kamu bisa menggunakan fungsi MID seperti berikut:

```
SELECT MID(nama, 1, 3) AS Nama
FROM Pegawai;
```

Tabel Output

Nama
Bud
Sar
Agu
Dia
Rin

4. LEN

Fungsi LEN digunakan untuk menghitung panjang string.

Misalkan kamu ingin mengetahui panjang nama dari setiap pegawai. Kamu bisa menggunakan fungsi LEN seperti berikut:

```
SELECT LEN(nama) AS PanjangNama
FROM Pegawai;
```

Tabel Output

PanjangNama
4
4
4
4
4

5. ROUND

Fungsi ROUND digunakan untuk membulatkan angka ke sejumlah desimal tertentu.

Misalkan kamu memiliki tabel gaji pegawai dan kamu ingin membulatkan gaji mereka ke angka terdekat. Kamu bisa menggunakan fungsi ROUND seperti berikut:

```
SELECT ROUND(gaji) AS GajiBulat
FROM Pegawai;
```

Tabel Output

GajiBulat
10000
8000
6000

6000
10000

6. NOW

Fungsi NOW digunakan untuk mendapatkan tanggal dan waktu saat ini.

Misalkan kamu ingin mengetahui waktu saat ini. Kamu bisa menggunakan fungsi NOW seperti berikut:

```
SELECT NOW() AS WaktuSekarang;
```

Tabel Output

WaktuSekarang
2023-05-31 18:45:32

7. FORMAT

Fungsi FORMAT digunakan untuk mengubah format angka, tanggal, atau waktu.

Misalkan kamu ingin mengubah format gaji pegawai menjadi format mata uang. Kamu bisa menggunakan fungsi FORMAT seperti berikut:

```
SELECT FORMAT(gaji, 'C') AS GajiFormat
FROM Pegawai;
```

Tabel Output

GajiFormat
$10,000.00
$8,000.00
$6,000.00
$6,000.00
$10,000.00

Bab 5. Subquery dan Join

5.1. Pengertian Subquery

Dalam penanganan data yang kompleks, seringkali kita perlu menjalankan query SQL di dalam query lain. Query semacam ini dikenal sebagai subquery atau inner query atau nested query. Subquery adalah query yang berada di dalam query lain dan berfungsi untuk membantu query luar dalam mendapatkan hasil yang diinginkan.

Pentingnya subquery dapat dipahami jika kamu mempertimbangkan skenario berikut: Bayangkan kamu memiliki sebuah database yang menampung data pelanggan dan penjualan. Kamu ingin mengetahui pelanggan mana yang melakukan pembelian tertinggi dalam sebulan. Untuk menjawab pertanyaan ini, pertama-tama kamu harus mencari total penjualan per pelanggan dalam sebulan, kemudian membandingkan total tersebut untuk mengetahui pelanggan mana yang memiliki total tertinggi. Dalam situasi seperti ini, subquery sangat membantu. Kamu bisa membuat subquery untuk menghitung total penjualan per pelanggan, kemudian menggunakan hasil subquery tersebut dalam query utama untuk menentukan pelanggan dengan total tertinggi.

Subquery dapat berada di berbagai bagian dari query SQL, termasuk bagian SELECT, FROM, WHERE, dan HAVING. Beberapa database juga mendukung penggunaan subquery di bagian INSERT, UPDATE, dan DELETE.

Subquery biasanya diawali dengan tanda kurung buka dan diakhiri dengan tanda kurung tutup. Hasil dari subquery dapat digunakan oleh query utama, dan sebaliknya, subquery dapat menggunakan informasi dari query utama.

Perlu diingat bahwa subquery memproses data dari dalam ke luar. Ini berarti bahwa database akan menjalankan subquery terlebih dahulu, lalu menggunakan hasil subquery untuk menjalankan query utama.

Berikut adalah contoh umum penggunaan subquery:

- Untuk membandingkan nilai sebuah kolom dengan hasil dari subquery. Misalnya, kamu ingin mengetahui barang yang harganya di atas rata-rata harga semua barang.
- Untuk memilih baris dari tabel yang memiliki atau tidak memiliki kecocokan di tabel lain. Misalnya, kamu ingin mengetahui pelanggan yang belum melakukan pembelian.
- Untuk menghitung nilai agregat dari sekelompok baris. Misalnya, kamu ingin mengetahui total penjualan per pelanggan.

Meskipun subquery dapat membantu dalam menangani situasi yang kompleks, perlu diingat bahwa subquery bisa memperlambat kinerja database. Hal ini dikarenakan setiap subquery harus dijalankan sebelum query utama, dan jika ada banyak subquery atau subquery yang kompleks, ini bisa memakan waktu yang cukup lama. Oleh karena itu, sebaiknya gunakan subquery secara efisien dan hanya ketika diperlukan.

5.2. Penggunaan Subquery

Subquery adalah query yang ditempatkan di dalam query lain, baik di dalam klausa SELECT, FROM, WHERE, atau HAVING. Subquery juga dikenal dengan istilah inner query atau nested query. Sebelum kita memahami lebih lanjut tentang penggunaan subquery, mari kita lihat tabel contoh berikut:

Tabel Mahasiswa:

NIM	Nama	Prodi	Angkatan
1	Budi	Teknik Informatika	2018
2	Andi	Teknik Informatika	2018
3	Sari	Teknik Informatika	2019
4	Dian	Sistem Informasi	2020
5	Rini	Sistem Informasi	2020
6	Ahmad	Teknik Informatika	2019
7	Ratna	Sistem Informasi	2018

Contoh 1: Subquery di klausa WHERE

Misalkan, kamu ingin mencari mahasiswa yang angkatannya paling awal, kamu dapat menggunakan subquery dalam klausa WHERE. Berikut ini adalah contoh SQL untuk mencari mahasiswa dengan angkatan paling awal:

```
SELECT NIM, Nama, Prodi, Angkatan
FROM Mahasiswa
WHERE Angkatan = (SELECT MIN(Angkatan) FROM Mahasiswa);
```

Subquery SELECT MIN(Angkatan) FROM Mahasiswa akan dievaluasi terlebih dahulu dan menghasilkan nilai minimum dari kolom Angkatan. Setelah itu, nilai ini akan digunakan dalam kondisi WHERE dari query utama.

Hasil query di atas adalah:

NIM	Nama	Prodi	Angkatan
1	Budi	Teknik Informatika	2018

2	Andi	Teknik Informatika	2018
7	Ratna	Sistem Informasi	2018

Contoh 2: Subquery di klausa FROM

Subquery juga bisa digunakan dalam klausa FROM untuk membuat tabel sementara yang bisa kamu gunakan dalam query utama. Misalkan kamu ingin menghitung jumlah mahasiswa dalam setiap prodi, kamu dapat menggunakan subquery dalam klausa FROM seperti berikut:

```
SELECT Prodi, COUNT(NIM) AS JumlahMahasiswa
FROM (SELECT Prodi, NIM FROM Mahasiswa) AS TabelSementara
GROUP BY Prodi;
```

Subquery (SELECT Prodi, NIM FROM Mahasiswa) menghasilkan tabel sementara yang selanjutnya digunakan dalam query utama. Dalam tabel sementara ini, kolom yang ada hanya Prodi dan NIM.

Hasil query di atas adalah:

Prodi	JumlahMahasiswa
Teknik Informatika	4
Sistem Informasi	3

Contoh 3: Subquery di klausa SELECT

Subquery juga dapat digunakan dalam klausa SELECT. Misalkan, kamu ingin mengetahui jumlah mahasiswa dalam prodi yang sama untuk setiap mahasiswa. Kamu bisa melakukan hal ini dengan subquery di klausa SELECT. Berikut adalah contohnya:

```
SELECT NIM, Nama, Prodi, Angkatan,
(SELECT COUNT(NIM) FROM Mahasiswa AS M2 WHERE M1.Prodi = M2.Prodi) AS
JumlahMahasiswa
FROM Mahasiswa AS M1;
```

Subquery di sini berfungsi untuk menghitung jumlah mahasiswa dalam prodi yang sama. Setelah itu, hasil dari subquery ini ditampilkan sebagai kolom baru dalam hasil query utama.

Hasil query di atas adalah:

NIM	Nama	Prodi	Angkatan	JumlahMahasiswa
1	Budi	Teknik Informatika	2018	4

2	Andi	Teknik Informatika	2018	4
3	Sari	Teknik Informatika	2019	4
4	Dian	Sistem Informasi	2020	3
5	Rini	Sistem Informasi	2020	3
6	Ahmad	Teknik Informatika	2019	4
7	Ratna	Sistem Informasi	2018	3

Dalam penggunaan subquery, kamu harus memperhatikan bahwa subquery dijalankan terlebih dahulu, kemudian hasilnya digunakan dalam query utama. Oleh karena itu, subquery harus menghasilkan output yang valid dan dapat digunakan dalam konteks query utama.

5.3. Pengertian Join

Join dalam SQL digunakan untuk menggabungkan baris dari dua atau lebih tabel berdasarkan kolom terkait antara mereka. Join memungkinkan kamu untuk mengambil data dari dua atau lebih tabel dan menggabungkannya menjadi satu set hasil. Untuk memahami konsep Join, mari kita lihat dua tabel contoh berikut:

Tabel Mahasiswa:

NIM	Nama	Prodi
1	Budi	Teknik Informatika
2	Andi	Teknik Informatika
3	Sari	Teknik Informatika
4	Dian	Sistem Informasi
5	Rini	Sistem Informasi
6	Ahmad	Teknik Informatika
7	Ratna	Sistem Informasi

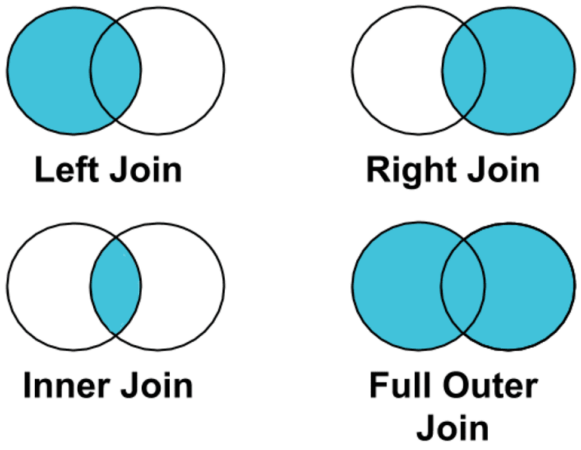
Tabel Nilai:

NIM	MataKuliah	Nilai
1	Matematika	85
2	Matematika	80
3	Matematika	90
4	Matematika	70
2	Fisika	88
5	Fisika	92

6	Fisika	76

Pada dasarnya, ada empat jenis join yang dapat kamu gunakan dalam SQL:

Inner Join: Menghasilkan baris saat ada kecocokan di kedua tabel.
Left (Outer) Join: Menghasilkan semua baris dari tabel kiri, dan baris yang cocok dari tabel kanan. Jika tidak ada kecocokan, hasilnya NULL pada sisi kanan.
Right (Outer) Join: Menghasilkan semua baris dari tabel kanan, dan baris yang cocok dari tabel kiri. Jika tidak ada kecocokan, hasilnya NULL pada sisi kiri.
Full (Outer) Join: Menghasilkan semua baris ketika ada kecocokan di salah satu tabel kiri atau kanan.



Source: https://dev.to/azibom/sql-joins-29j4

Contoh Inner Join

Misalkan, kamu ingin mendapatkan daftar nama mahasiswa dan nilai matematika yang mereka peroleh. Berikut adalah contoh penggunaan inner join:

```
SELECT Mahasiswa.Nama, Nilai.Nilai
FROM Mahasiswa
INNER JOIN Nilai ON Mahasiswa.NIM = Nilai.NIM
WHERE Nilai.MataKuliah = 'Matematika';
```

Hasilnya:

Nama	Nilai
Budi	85
Andi	80
Sari	90
Dian	70

Dalam query di atas, INNER JOIN menggabungkan baris dari tabel Mahasiswa dan Nilai jika NIM pada kedua tabel tersebut cocok.

5.4. Penggunaan Join

Setelah memahami pengertian Join, sekarang kita akan mendalami cara menggunakan berbagai jenis join dalam SQL.

INNER JOIN

Inner join menghasilkan baris saat ada kecocokan di kedua tabel. Untuk lebih jelasnya, mari kita lihat contoh penggunaannya.

```
SELECT Mahasiswa.Nama, Nilai.Nilai
FROM Mahasiswa
INNER JOIN Nilai ON Mahasiswa.NIM = Nilai.NIM
WHERE Nilai.MataKuliah = 'Matematika';
```

Hasil:

Nama	Nilai
Budi	85
Andi	80
Sari	90
Dian	70

LEFT (OUTER) JOIN

Left join menghasilkan semua baris dari tabel kiri, dan baris yang cocok dari tabel kanan. Jika tidak ada kecocokan, hasilnya NULL pada sisi kanan.

```
SELECT Mahasiswa.Nama, Nilai.Nilai
FROM Mahasiswa
LEFT JOIN Nilai ON Mahasiswa.NIM = Nilai.NIM
AND Nilai.MataKuliah = 'Fisika';
```

Hasil:

Nama	Nilai
Budi	NULL
Andi	88
Sari	NULL

Dian	NULL
Rini	92
Ahmad	76
Ratna	NULL

RIGHT (OUTER) JOIN

Right join menghasilkan semua baris dari tabel kanan, dan baris yang cocok dari tabel kiri. Jika tidak ada kecocokan, hasilnya NULL pada sisi kiri.

```
SELECT Mahasiswa.Nama, Nilai.Nilai
FROM Mahasiswa
RIGHT JOIN Nilai ON Mahasiswa.NIM = Nilai.NIM
AND Nilai.MataKuliah = 'Fisika';
```

Hasil:

Nama	Nilai
NULL	NULL
Andi	88
NULL	NULL
NULL	NULL
Rini	92
Ahmad	76
NULL	NULL

FULL (OUTER) JOIN

Full join menghasilkan semua baris ketika ada kecocokan di salah satu tabel kiri atau kanan.

```
SELECT Mahasiswa.Nama, Nilai.Nilai
FROM Mahasiswa
FULL JOIN Nilai ON Mahasiswa.NIM = Nilai.NIM;
```

Hasil:

Nama	Nilai
Budi	85
Andi	80
Andi	88
Sari	90

Dian	70
Rini	92
Ahmad	76
Ratna	NULL

Melalui contoh di atas, kamu dapat melihat bagaimana setiap jenis join berfungsi dan bagaimana cara menggunakannya dalam query SQL kamu. Pemahaman ini penting ketika kamu bekerja dengan database relasional, karena Join memungkinkan kamu untuk mengekstrak data yang kompleks dari beberapa tabel sekaligus.

Bab 6. Index dan Views

6.1. Pengertian dan Manfaat Index

Sebuah index pada dasarnya adalah struktur data yang memperbaiki kecepatan operasi pada database table. Index membantu dalam mempercepat pencarian dan query dengan menciptakan pointer ke data pada sebuah table.

Sebagai analogi, kamu bisa membayangkan index seperti daftar isi pada sebuah buku. Jika kamu ingin mencari informasi tertentu dalam buku, kamu bisa melihat daftar isi untuk menemukan halaman di mana informasi tersebut berada. Tanpa daftar isi, kamu harus membuka setiap halaman buku satu per satu hingga menemukan informasi yang kamu cari. Sama halnya dengan index dalam database, tanpa index, database harus melakukan 'full table scan', yaitu membaca setiap baris table satu per satu sampai menemukan row yang dicari.

Namun, seperti halnya daftar isi yang membutuhkan ruang ekstra di awal buku, index dalam database juga membutuhkan ruang disk tambahan, dan ada waktu tambahan yang diperlukan untuk memperbarui index setiap kali data ditambahkan, diubah, atau dihapus. Jadi, harus ada keseimbangan antara jumlah indeks (yang mempercepat query SELECT tetapi memperlambat operasi INSERT, UPDATE, dan DELETE) dan kebutuhan operasi dalam database.

Mari kita lihat contoh menggunakan index dalam praktiknya.

Kita mulai dengan tabel Mahasiswa seperti berikut:

Tabel Mahasiswa

NIM	Nama
111	Budi
222	Andi
333	Sari
444	Dian
555	Rini
666	Ahmad
777	Ratna

Untuk mencari informasi tentang mahasiswa dengan NIM 555, tanpa index, database harus melihat setiap baris, satu per satu, hingga menemukan baris dengan NIM 555. Ini membutuhkan banyak waktu jika table sangat besar.

Kita bisa membuat index pada kolom NIM. Setelah index dibuat, database bisa menggunakan index untuk menemukan baris dengan NIM 555 dengan jauh lebih cepat, mirip dengan cara kita menggunakan daftar isi buku.

Itulah pengertian dan manfaat menggunakan index dalam SQL. Kita akan membahas bagaimana membuat dan menghapus index pada subbab selanjutnya.

6.2. Membuat dan Menghapus Index

Setelah memahami apa itu index dan manfaatnya, selanjutnya kita akan membahas cara membuat dan menghapus index dalam database SQL. Melakukan ini akan memungkinkan kamu untuk mengoptimalkan database kamu dan memastikan bahwa operasi SELECT kamu berjalan seefisien mungkin.

Mari kita mulai dengan contoh praktis menggunakan tabel Mahasiswa dari subbab sebelumnya. Sebagai pengingat, berikut adalah tabel Mahasiswa:

Tabel Mahasiswa

NIM	Nama
111	Budi
222	Andi
333	Sari
444	Dian
555	Rini
666	Ahmad
777	Ratna

Membuat Index

Untuk membuat index, kamu dapat menggunakan perintah CREATE INDEX. Kita akan membuat index pada kolom NIM:

```
CREATE INDEX idx_mahasiswa_nim
ON Mahasiswa (NIM);
```

Perintah di atas akan membuat sebuah index pada kolom NIM pada tabel Mahasiswa. Index ini akan membantu mempercepat operasi SELECT yang melibatkan kolom NIM. Harap diperhatikan bahwa operasi ini mungkin akan memakan waktu jika tabel kamu sangat besar, karena database harus membaca setiap baris dalam tabel dan membangun struktur data index.

Menghapus Index

Di beberapa kasus, kamu mungkin perlu menghapus index yang sudah ada. Misalnya, jika kamu mengetahui bahwa index tersebut tidak pernah digunakan oleh query yang kamu jalankan, atau jika kamu perlu menghemat ruang disk.

Untuk menghapus index, kamu dapat menggunakan perintah DROP INDEX. Berikut cara menghapus index yang telah kita buat sebelumnya:

```
DROP INDEX idx_mahasiswa_nim;
```

Perintah di atas akan menghapus index idx_mahasiswa_nim dari tabel Mahasiswa.

Penting untuk dicatat bahwa menghapus index tidak akan menghapus data yang ada dalam tabel, hanya struktur data index yang dihapus.

Dengan memahami cara membuat dan menghapus index, kamu sekarang memiliki lebih banyak alat untuk mengoptimalkan database kamu dan memastikan bahwa query kamu berjalan dengan seefisien mungkin.

6.3. Pengertian, Cara Membuat, dan Menghapus Tabel View

View dalam SQL adalah tabel virtual yang dihasilkan dari query. Meski secara fisik view ini tidak ada dalam database (karena tidak ada data yang disimpan dalam view itu sendiri), tetapi view ini berisi baris dan kolom seperti yang ada dalam tabel sebenarnya.

Untuk memahami konsep ini, mari kita ambil contoh tabel mahasiswa dan nilai:

Membuat View

Tabel Mahasiswa

NIM	Nama
111	Budi
222	Andi
333	Sari

Tabel Nilai

NIM	Mata Kuliah	Nilai
111	Matematika	90

222	Matematika	80
333	Matematika	85
111	Biologi	88
222	Biologi	78
333	Biologi	90

Dari dua tabel di atas, kita bisa membuat sebuah view yang menampilkan nama mahasiswa dan nilai mereka dalam mata kuliah Matematika. View ini dapat dibuat dengan query berikut:

```
CREATE VIEW view_matematika AS
SELECT Mahasiswa.Nama, Nilai.Nilai
FROM Mahasiswa
JOIN Nilai ON Mahasiswa.NIM = Nilai.NIM
WHERE Nilai.Mata_Kuliah = 'Matematika';
```

Ketika kamu menjalankan query SELECT * FROM view_matematika;, kamu akan mendapatkan tabel berikut:

Tabel view_matematika

Nama	Nilai
Budi	90
Andi	80
Sari	85

Jadi, apa manfaat dari view ini?

- Simplifikasi Query: View memungkinkan kamu untuk menyembunyikan kompleksitas query. Misalnya, jika kamu memiliki query yang sangat kompleks dan panjang, kamu bisa menyembunyikannya di balik view dan kemudian kamu bisa memanggil view tersebut tanpa harus menulis ulang query yang panjang tersebut.
- Keamanan Data: Dengan view, kamu bisa membatasi akses pengguna ke beberapa kolom atau baris data dalam tabel. Misalnya, jika kamu tidak ingin pengguna melihat kolom gaji dalam tabel pegawai, kamu bisa membuat view yang tidak mencakup kolom gaji tersebut.
- Membantu dalam Modifikasi Logis Data: View bisa membantu dalam modifikasi logis data. Dengan kata lain, view dapat digunakan untuk mendekomposisi tabel yang sangat besar dan kompleks menjadi bagian-bagian yang lebih kecil dan lebih mudah dikelola.

Menghapus View

Jika kamu ingin menghapus view yang sudah dibuat, kamu bisa menggunakan perintah DROP VIEW. Berikut adalah sintaksnya:

```
DROP VIEW view_matematika;
```

Dengan menjalankan perintah di atas, view view_matematika akan dihapus dari database. Jika kamu mencoba menjalankan perintah SELECT * FROM view_matematika; setelah view dihapus, kamu akan mendapatkan pesan error karena view tersebut sudah tidak ada.

Bab 7. SQL Window Functions

7.1. Pengertian SQL Window Functions

Sebelum kita membahas lebih jauh tentang fungsi window di SQL, pertama-tama kita harus mengerti apa itu fungsi window. Fungsi window merupakan suatu fitur dalam SQL yang memungkinkan kita melakukan berbagai perhitungan pada suatu set data yang kita definisikan. Set data ini biasa disebut "window".

Fungsi window memiliki kemampuan yang unik dan kuat yang tidak dimiliki oleh fungsi SQL lainnya. Secara khusus, fungsi window dapat mengakses lebih dari satu baris dalam tabel tunggal tanpa harus menggabungkan tabel tersebut.

Berikut ini adalah beberapa karakteristik dari fungsi window:

- Melakukan perhitungan pada set data (window) yang kita tentukan.
- Fungsi window melakukan perhitungan pada sekumpulan baris yang terkait dengan baris saat ini. Ini berarti bahwa, setiap kali fungsi window dievaluasi, hasilnya bisa berbeda tergantung pada konten window.
- Tidak mengubah jumlah baris.
- Berbeda dengan fungsi agregat, fungsi window tidak mengubah jumlah baris dalam hasil. Setiap baris input sesuai dengan satu baris output.
- Mempunyai akses ke lebih dari satu baris dalam tabel tunggal tanpa harus melakukan join.

Hal ini sangat membantu dalam situasi di mana kamu perlu membandingkan nilai dari baris saat ini dengan nilai dari baris sebelumnya atau berikutnya.

Untuk memberikan contoh konkret dari fungsi window, mari kita lihat kasus berikut. Misalkan kita memiliki tabel penjualan berikut:

Tabel Penjualan

ID	Nama Produk	Jumlah
1	Produk A	5
2	Produk B	3
3	Produk C	7
4	Produk D	2
5	Produk E	6

Kita ingin mengetahui total penjualan yang berjalan (running total). Dengan fungsi window, kita dapat melakukannya dengan mudah. Berikut adalah contoh querynya:

```
SELECT
  ID,
  Nama Produk,
  Jumlah,
  SUM(Jumlah) OVER (ORDER BY ID) AS Running_Total
FROM
  Penjualan;
```

Hasilnya akan seperti ini:

ID	Nama Produk	Jumlah	Running_Total
1	Produk A	5	5
2	Produk B	3	8
3	Produk C	7	15
4	Produk D	2	17
5	Produk E	6	23

Kolom Running_Total menghitung total penjualan yang berjalan. Untuk setiap baris, nilai Running_Total adalah total penjualan dari semua produk dengan ID kurang atau sama dengan ID produk saat ini.

Dari contoh di atas, bisa kita lihat bahwa fungsi window sangat berguna untuk melakukan perhitungan yang melibatkan lebih dari satu baris dalam satu tabel. Fungsi ini sangat kuat dan memiliki banyak aplikasi, terutama dalam analisis data dan pengolahan data.

Dalam bab-bab berikutnya, kita akan membahas lebih jauh tentang berbagai jenis fungsi window di SQL, termasuk ROW_NUMBER(), RANK(), DENSE_RANK(), LAG(), LEAD(), FIRST_VALUE(), LAST_VALUE(), dan cara menggunakan fungsi agregat seperti SUM(), AVG(), MIN(), dan MAX() dengan klausa OVER() dan PARTITION BY. Semua fungsi ini memiliki fitur dan karakteristik yang unik, dan masing-masing memiliki kegunaannya sendiri dalam berbagai skenario analisis data. Oleh karena itu, penting bagi kamu untuk memahaminya dengan baik.

7.2. Fungsi ROW_NUMBER()

Fungsi ROW_NUMBER() merupakan salah satu fungsi window yang disediakan oleh SQL. Fungsi ini akan memberikan nomor unik untuk setiap baris dalam hasil query, berdasarkan urutan yang ditentukan dalam klausa ORDER BY yang diberikan dalam klausa OVER.

Sintaks dari fungsi ROW_NUMBER() adalah sebagai berikut:

```
ROW_NUMBER() OVER (ORDER BY column)
```

Di mana:

ROW_NUMBER() adalah fungsi yang akan memberikan nomor urut untuk setiap baris.
OVER digunakan untuk mendefinisikan window atau set baris yang akan diperhitungkan oleh fungsi ROW_NUMBER().
ORDER BY column digunakan untuk menentukan urutan baris dalam window.
Mari kita lihat contoh penggunaannya. Misalkan kita memiliki tabel Penjualan sebagai berikut:

Tabel Penjualan

ID_Transaksi	Nama_Produk	Jumlah
1	Produk A	5
2	Produk B	3
3	Produk C	7
4	Produk D	2
5	Produk E	6

Kita ingin memberikan nomor urut untuk setiap transaksi berdasarkan jumlah penjualan. Berikut adalah query yang bisa kita gunakan:

```
SELECT
  ID_Transaksi,
  Nama_Produk,
  Jumlah,
  ROW_NUMBER() OVER (ORDER BY Jumlah DESC) AS Nomor_Urut
FROM
  Penjualan;
```

Hasilnya akan seperti ini:

ID_Transaksi	Nama_Produk	Jumlah	Nomor_Urut
3	Produk C	7	1
5	Produk E	6	2
1	Produk A	5	3
2	Produk B	3	4
4	Produk D	2	5

Dalam contoh di atas, fungsi ROW_NUMBER() memberikan nomor urut untuk setiap baris berdasarkan jumlah penjualan. Transaksi dengan jumlah penjualan tertinggi mendapatkan nomor urut 1, dan seterusnya.

Dalam banyak kasus, fungsi ROW_NUMBER() sangat berguna. Misalnya, jika kamu ingin mengetahui posisi relatif dari suatu baris dalam suatu set data, atau jika kamu ingin menghilangkan baris duplikat berdasarkan beberapa kriteria.

Namun, perlu kamu ingat bahwa hasil dari fungsi ROW_NUMBER() mungkin tidak konsisten jika tidak ada klausa ORDER BY dalam klausa OVER, atau jika ada lebih dari satu baris yang memiliki nilai yang sama untuk kolom yang digunakan dalam klausa ORDER BY. Oleh karena itu, selalu pastikan bahwa kamu memiliki klausa ORDER BY yang jelas dan spesifik saat menggunakan fungsi ROW_NUMBER().

7.3. Fungsi RANK() dan DENSE_RANK()

Fungsi RANK() dan DENSE_RANK() juga merupakan bagian dari fungsi window di SQL yang digunakan untuk memberikan peringkat pada setiap baris dalam hasil query.

RANK()

Fungsi RANK() memberikan peringkat unik untuk setiap baris, tetapi jika terdapat dua baris atau lebih yang memiliki nilai yang sama, semua baris tersebut akan mendapatkan peringkat yang sama dan peringkat berikutnya akan dilewati. Misalnya, jika dua baris mendapatkan peringkat 1, peringkat berikutnya yang akan diberikan adalah 3, bukan 2.

Berikut adalah sintaks dari fungsi RANK():

```
RANK() OVER (ORDER BY column)
```

DENSE_RANK()

Berbeda dengan fungsi RANK(), fungsi DENSE_RANK() tidak melewatkan peringkat jika terdapat baris yang memiliki nilai yang sama. Misalnya, jika dua baris mendapatkan peringkat 1, peringkat berikutnya yang akan diberikan adalah 2, bukan 3.

Berikut adalah sintaks dari fungsi DENSE_RANK():

```
DENSE_RANK() OVER (ORDER BY column)
```

Untuk memahami lebih lanjut, mari kita lihat contoh penggunaan fungsi RANK() dan DENSE_RANK().

Misalkan kita memiliki tabel Penjualan sebagai berikut:

Tabel Penjualan

ID_Transaksi	Nama_Produk	Jumlah
1	Produk A	5
2	Produk B	3
3	Produk C	5
4	Produk D	2
5	Produk E	6

Kita ingin memberikan peringkat untuk setiap produk berdasarkan jumlah penjualan. Berikut adalah query yang bisa kita gunakan:

```
SELECT
  ID_Transaksi,
  Nama_Produk,
  Jumlah,
  RANK() OVER (ORDER BY Jumlah DESC) AS Rank,
  DENSE_RANK() OVER (ORDER BY Jumlah DESC) AS Dense_Rank
FROM
  Penjualan;
```

Hasilnya akan seperti ini:

ID_Transaksi	Nama_Produk	Jumlah	Rank	Dense_Rank
5	Produk E	6	1	1
1	Produk A	5	2	2
3	Produk C	5	2	2
2	Produk B	3	4	3
4	Produk D	2	5	4

Dalam contoh di atas, fungsi RANK() dan DENSE_RANK() memberikan peringkat untuk setiap baris berdasarkan jumlah penjualan. Produk dengan jumlah penjualan tertinggi mendapatkan peringkat 1.

Kamu dapat melihat bahwa Produk A dan Produk C yang sama-sama memiliki jumlah penjualan 5 mendapatkan peringkat 2 oleh kedua fungsi. Namun, untuk peringkat berikutnya, fungsi RANK() melewatkan peringkat 3 dan langsung ke peringkat 4 (untuk Produk B), sementara fungsi DENSE_RANK() tidak melewatkan peringkat dan memberikan peringkat 3 untuk Produk B.

Pemahaman yang baik tentang perbedaan antara RANK() dan DENSE_RANK() penting dalam berbagai situasi, terutama saat kamu harus memilih fungsi mana yang harus digunakan berdasarkan kebutuhan spesifik kamu.

7.4. Fungsi LAG() dan LEAD()

Fungsi LAG() dan LEAD() dalam SQL adalah fungsi window yang digunakan untuk membandingkan nilai suatu baris dengan nilai baris lain dalam kumpulan hasil. Kedua fungsi ini sangat berguna saat perlu membandingkan nilai antara baris saat melakukan analisis data.

Fungsi LAG()

Fungsi LAG() digunakan untuk mengambil nilai dari baris sebelumnya dalam kumpulan hasil. Fungsi ini menerima dua argumen: nama kolom dan jarak. Jarak (opsional) merujuk pada seberapa jauh kamu ingin melihat ke belakang. Jika jarak tidak disebutkan, secara default akan dianggap 1.

Berikut adalah sintaks fungsi LAG():

```
LAG(column, offset) OVER (ORDER BY column)
```

Fungsi LEAD()

Sementara itu, fungsi LEAD() digunakan untuk melihat ke depan, atau mengambil nilai dari baris berikutnya dalam kumpulan hasil. Fungsi ini juga menerima dua argumen yang sama dengan fungsi LAG().

Berikut adalah sintaks fungsi LEAD():

```
LEAD(column, offset) OVER (ORDER BY column)
```

Untuk lebih memahami fungsi LAG() dan LEAD(), mari kita lihat contoh berikut. Misalkan kita memiliki tabel Penjualan sebagai berikut:

Tabel Penjualan

Tanggal	Jumlah
2023-01-01	100
2023-01-02	150
2023-01-03	200
2023-01-04	250

Kita ingin melihat perubahan jumlah penjualan dari hari ke hari. Berikut adalah query yang bisa kita gunakan:

```
SELECT
  Tanggal,
  Jumlah,
  LAG(Jumlah) OVER (ORDER BY Tanggal) AS Jumlah_Hari_Sebelumnya,
  LEAD(Jumlah) OVER (ORDER BY Tanggal) AS Jumlah_Hari_Berikutnya
FROM
  Penjualan;
```

Hasilnya akan seperti ini:

Tanggal	Jumlah	Jumlah_Hari_Sebelumnya	Jumlah_Hari_Berikutnya
2023-01-01	100	NULL	150
2023-01-02	150	100	200
2023-01-03	200	150	250
2023-01-04	250	200	NULL

Dalam contoh di atas, fungsi LAG() memberikan jumlah penjualan pada hari sebelumnya dan fungsi LEAD() memberikan jumlah penjualan pada hari berikutnya untuk setiap baris.

Fungsi LAG() dan LEAD() sangat berguna saat melakukan analisis tren atau perbandingan over-time di SQL. Dengan menggunakan fungsi ini, kamu bisa dengan mudah membandingkan nilai suatu baris dengan baris lain dalam hasil query, tanpa perlu melakukan join tabel dengan dirinya sendiri atau membuat subquery yang kompleks.

7.5. Fungsi FIRST_VALUE() dan LAST_VALUE()

Dalam SQL, fungsi FIRST_VALUE() dan LAST_VALUE() adalah bagian dari fungsi Window yang dapat digunakan untuk mengambil nilai pertama atau terakhir dari suatu set data dalam jendela yang telah ditentukan.

Fungsi FIRST_VALUE()

Fungsi FIRST_VALUE() digunakan untuk mengambil nilai pertama dalam jendela data yang telah ditentukan. Berikut adalah sintaks fungsi FIRST_VALUE():

```
FIRST_VALUE(column) OVER (ORDER BY column)
```

Fungsi LAST_VALUE()

Sebaliknya, fungsi LAST_VALUE() digunakan untuk mengambil nilai terakhir dalam jendela data yang telah ditentukan. Berikut adalah sintaks fungsi LAST_VALUE():

```
LAST_VALUE(column) OVER (ORDER BY column)
```

Untuk memahami fungsi FIRST_VALUE() dan LAST_VALUE(), mari kita lihat contoh berikut.

Misalkan kita memiliki tabel Penjualan sebagai berikut:

Tabel Penjualan

Tanggal	Jumlah
2023-01-01	100
2023-01-02	150
2023-01-03	200
2023-01-04	250

Kita ingin melihat penjualan pertama dan terakhir dalam periode waktu yang ditentukan. Berikut adalah query yang bisa kita gunakan:

```
SELECT
  Tanggal,
  Jumlah,
  FIRST_VALUE(Jumlah) OVER (ORDER BY Tanggal) AS Penjualan_Pertama,
  LAST_VALUE(Jumlah) OVER (ORDER BY Tanggal ROWS BETWEEN UNBOUNDED
PRECEDING AND UNBOUNDED FOLLOWING) AS Penjualan_Terakhir
FROM
  Penjualan;
```

Hasilnya akan seperti ini:

Tanggal	Jumlah	Penjualan_Pertama	Penjualan_Terakhir
2023-01-01	100	100	250
2023-01-02	150	100	250
2023-01-03	200	100	250
2023-01-04	250	100	250

Perhatikan bahwa dalam fungsi LAST_VALUE(), kita perlu menambahkan klausa ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING untuk memastikan bahwa fungsi ini mempertimbangkan semua baris dalam set data.

Fungsi FIRST_VALUE() dan LAST_VALUE() sangat berguna saat perlu mengetahui nilai pertama atau terakhir dalam set data, misalnya saat melakukan analisis tren waktu atau mengevaluasi perubahan nilai sepanjang waktu.

7.6 Fungsi SUM(), AVG(), MIN(), MAX() dengan OVER() dan PARTITION BY

Fungsi agregat seperti SUM(), AVG(), MIN(), dan MAX() biasanya digunakan dalam SQL untuk melakukan operasi matematika pada sekelompok nilai dan mengembalikan hasil tunggal. Saat digunakan dengan fungsi window OVER() dan PARTITION BY, fungsi-fungsi ini dapat memberikan insight yang lebih besar tentang data kamu.

Mari kita mulai dengan penjelasan singkat tentang masing-masing fungsi.

- SUM(): Menghitung total atau jumlah dari serangkaian nilai.
- AVG(): Menghitung rata-rata dari serangkaian nilai.
- MIN(): Menemukan nilai terkecil di antara serangkaian nilai.
- MAX(): Menemukan nilai terbesar di antara serangkaian nilai.

Mari kita lihat contoh tabel berikut yang berisi data penjualan per hari untuk dua produk:

Tabel Penjualan

Tanggal	Produk	Jumlah
2023-01-01	A	100
2023-01-01	B	150
2023-01-02	A	200
2023-01-02	B	250
2023-01-03	A	300
2023-01-03	B	350
2023-01-04	A	400
2023-01-04	B	450

Berikut query SQL yang menggunakan fungsi agregat dengan fungsi window OVER() dan PARTITION BY:

```
SELECT
  Tanggal,
  Produk,
  Jumlah,
  SUM(Jumlah) OVER (PARTITION BY Produk) AS Total_Terakumulasi,
  AVG(Jumlah) OVER (PARTITION BY Produk) AS Rata2_Terakumulasi,
```

```
  MIN(Jumlah) OVER (PARTITION BY Produk) AS Min_Terakumulasi,
  MAX(Jumlah) OVER (PARTITION BY Produk) AS Max_Terakumulasi
FROM
  Penjualan;
```

Hasilnya akan seperti ini:

Tanggal	Produk	Jumlah	Total_Terakumulasi	Rata2_Terakumulasi	Min_Terakumulasi	Max_Terakumulasi
2023-01-01	A	100	1000	250	100	400
2023-01-02	A	200	1000	250	100	400
2023-01-03	A	300	1000	250	100	400
2023-01-04	A	400	1000	250	100	400
2023-01-01	B	150	1200	300	150	450
2023-01-02	B	250	1200	300	150	450
2023-01-03	B	350	1200	300	150	450
2023-01-04	B	450	1200	300	150	450

Dari tabel output di atas, kamu dapat melihat bagaimana setiap fungsi agregat berperilaku. Fungsi SUM() memberikan total penjualan terakumulasi untuk setiap produk. Fungsi AVG() memberikan rata-rata penjualan terakumulasi untuk setiap produk. Fungsi MIN() dan MAX() memberikan nilai penjualan minimum dan maksimum yang terakumulasi untuk setiap produk.

Sebagai catatan, kamu perlu berhati-hati saat menggunakan fungsi agregat dengan fungsi window. Fungsi agregat beroperasi pada seluruh set data kecuali jika kamu menentukan partisi dengan PARTITION BY, sementara fungsi window secara default hanya beroperasi pada baris 'sejauh ini' dalam set data kecuali jika kamu menentukan frame window lain dengan ROWS BETWEEN.

Bab 8. Stored Procedure dan Triggers

8.1 Pengertian Stored Procedure

Sebagai pengguna SQL, kemungkinan besar kamu sudah familiar dengan konsep dan pemakaian query. Query adalah instruksi atau pernyataan yang kamu berikan kepada database untuk melakukan sesuatu, baik itu mengambil data, mengubah data, menghapus data, atau melakukan sesuatu yang lain. Tapi apa jadinya jika kamu memiliki serangkaian instruksi yang harus dijalankan berulang kali? Atau mungkin ada logika tertentu yang ingin kamu kapsulasi dan gunakan kembali? Inilah di mana stored procedure masuk.

Stored procedure adalah kumpulan pernyataan SQL yang disimpan dan didefinisikan di database. Stored procedure biasanya digunakan untuk mengotomatisasi tugas yang berulang, untuk menyederhanakan beban kerja yang kompleks dan juga digunakan untuk meningkatkan keamanan dan efisiensi.

Stored procedure adalah suatu fungsi yang ada di dalam database dan dapat dipanggil oleh aplikasi yang membutuhkan fungsionalitas yang disediakan oleh stored procedure tersebut. Ketika sebuah aplikasi memanggil stored procedure, database akan menjalankan sejumlah perintah SQL yang terdapat di dalam stored procedure tersebut.

Misalnya, bayangkan situasi di mana sebuah aplikasi e-commerce perlu menghitung total biaya pesanan untuk setiap pelanggan. Untuk menghitung total ini, aplikasi perlu menjalankan beberapa query SQL untuk mengambil harga dari setiap produk dalam pesanan, mengalikan harga dengan jumlah produk, dan kemudian menambahkan semua total untuk mendapatkan total biaya pesanan. Semua ini bisa jadi cukup rumit dan memakan waktu. Namun, dengan menggunakan stored procedure, semua perintah ini dapat digabungkan menjadi satu prosedur yang disimpan di dalam database. Kemudian, aplikasi hanya perlu memanggil prosedur ini dengan ID pesanan sebagai input, dan database akan menghitung total biaya pesanan dan mengembalikannya sebagai output.

Keuntungan lain dari menggunakan stored procedure adalah meningkatkan keamanan. Karena logika dan operasi data disimpan di server database, pengguna atau aplikasi yang berinteraksi dengan database tidak perlu memiliki akses langsung ke tabel database. Sebagai gantinya, mereka dapat berinteraksi dengan database melalui stored procedure, yang dapat dirancang untuk memberikan kontrol yang lebih baik atas apa yang dapat dan tidak dapat dilakukan pengguna.

Selain itu, karena stored procedure telah dikompilasi dan disimpan di server database, eksekusinya bisa lebih cepat daripada menjalankan sejumlah besar pernyataan SQL secara individual, yang setiap kali harus dikirim dari aplikasi ke server database dan dikompilasi sebelum dijalankan.

Namun, meski memiliki banyak keuntungan, penggunaan stored procedure juga harus dipertimbangkan dengan cermat. Stored procedure bisa menjadi rumit dan sulit untuk dipahami, terutama jika mereka mencakup banyak logika atau operasi kompleks. Selain itu, perubahan pada stored procedure bisa mempengaruhi banyak bagian dari aplikasi, sehingga perlu hati-hati saat melakukan perubahan. Dan terakhir, meskipun stored procedure umum di banyak sistem database manajemen (DBMS), sintaks dan fungsionalitasnya bisa berbeda-beda, yang bisa menyulitkan migrasi atau portabilitas antar platform.

Secara umum, stored procedure bisa menjadi alat yang sangat berguna dalam toolbox seorang pengembang database. Mereka dapat membantu mengotomatisasi tugas yang berulang, menyederhanakan operasi yang kompleks, dan memberikan kontrol yang lebih baik atas akses dan manipulasi data. Namun, seperti semua alat, mereka harus digunakan dengan bijak dan dengan pemahaman yang baik tentang kekuatan dan keterbatasan mereka.

8.2 Membuat dan Menggunakan Stored Procedure

Dalam subbab ini, kita akan membahas bagaimana membuat dan menggunakan stored procedure. Untuk membuatnya lebih jelas, mari kita ambil contoh dari sebuah skenario real. Misalkan kita memiliki sebuah aplikasi e-commerce dan kita perlu menghitung total pesanan untuk setiap pelanggan. Untuk mempermudah proses ini, kita bisa membuat sebuah stored procedure.

Berikut ini adalah struktur dari tabel orders dan order_details:

Tabel orders:

OrderID	CustomerID	OrderDate
1	CUST01	2023-02-21
2	CUST02	2023-02-22
3	CUST01	2023-02-23

Tabel order_details:

OrderDetailID	OrderID	ProductID	Quantity	UnitPrice
1	1	PROD01	2	50.00
2	1	PROD02	1	100.00
3	2	PROD01	3	50.00
4	3	PROD03	1	150.00

Berikut ini adalah contoh bagaimana membuat stored procedure di SQL Server:

```
CREATE PROCEDURE GetOrderTotal
  @OrderID INT
AS
BEGIN
  SELECT OrderID, SUM(Quantity * UnitPrice) as OrderTotal
  FROM order_details
  WHERE OrderID = @OrderID
  GROUP BY OrderID;
END
```

Stored procedure ini menghitung total pesanan berdasarkan ID pesanan. Kita menggunakan parameter @OrderID untuk menentukan pesanan mana yang kita ingin hitung totalnya.

Setelah stored procedure ini dibuat, kita bisa memanggilnya seperti berikut:

```
EXEC GetOrderTotal 1;
```

Dan outputnya akan seperti berikut:

OrderID	OrderTotal
1	200.00

Output tersebut menunjukkan bahwa total dari pesanan dengan OrderID 1 adalah 200.00.

Itulah dasar-dasar membuat dan menggunakan stored procedure. Namun, penting untuk diingat bahwa ini hanyalah contoh sederhana. Dalam praktiknya, stored procedure bisa jauh lebih kompleks, dan mungkin melibatkan beberapa tabel, kondisi, dan operasi lainnya.

Satu hal lagi yang perlu diperhatikan adalah bahwa stored procedure biasanya spesifik untuk database tertentu. Artinya, cara membuat dan menggunakan stored procedure bisa berbeda tergantung pada jenis database yang kamu gunakan. Contohnya, MySQL menggunakan perintah CALL untuk memanggil stored procedure, sedangkan SQL Server menggunakan EXEC. Oleh karena itu, sangat penting untuk memahami dokumentasi database kamu dan memastikan bahwa stored procedure kamu kompatibel.

8.3 Pengertian Triggers

Triggers adalah salah satu fitur SQL yang berguna dan kuat. Dalam konteks basis data, trigger adalah kode khusus yang dijalankan atau 'dipicu' oleh server basis data sebagai

reaksi terhadap peristiwa tertentu yang terjadi di dalam tabel. Peristiwa tersebut bisa berupa operasi CRUD (Create, Read, Update, Delete).

Triggers memiliki banyak manfaat dalam membangun dan mengelola basis data yang efisien dan efektif. Salah satunya adalah kemampuannya untuk menjaga integritas data. Kamu bisa membuat trigger untuk mencegah perubahan yang tidak valid pada data atau untuk melakukan tindakan khusus (seperti logging) saat perubahan tertentu terjadi.

Mari kita ilustrasikan konsep ini dengan sebuah contoh.

Misalkan kita memiliki sebuah sistem manajemen toko buku online dan kita memiliki tabel orders dan inventory sebagai berikut:

Tabel orders:

OrderID	BookID	Quantity
1	B01	3
2	B02	1
3	B03	2

Tabel inventory:

BookID	Stock
B01	10
B02	5
B03	7

Kita bisa membuat trigger untuk memastikan bahwa saat order baru ditambahkan, jumlah stok buku di tabel inventory akan berkurang sesuai dengan jumlah yang dipesan.

Berikut adalah contoh pembuatan trigger di SQL:

```
CREATE TRIGGER UpdateInventory
AFTER INSERT ON orders
FOR EACH ROW
BEGIN
  UPDATE inventory
  SET Stock = Stock - NEW.Quantity
  WHERE BookID = NEW.BookID;
END;
```

Trigger UpdateInventory ini dibuat untuk dijalankan setelah operasi INSERT dilakukan pada tabel orders. Saat trigger ini dipicu, ia akan mengurangi jumlah stok buku di tabel inventory

sejumlah NEW.Quantity, dimana NEW merujuk ke baris yang baru saja dimasukkan ke tabel orders.

Misalkan kita menambahkan order baru ke tabel orders seperti berikut:

```
INSERT INTO orders (OrderID, BookID, Quantity)
VALUES (4, 'B01', 2);
```

Tabel orders setelah operasi INSERT:

OrderID	BookID	Quantity
1	B01	3
2	B02	1
3	B03	2
4	B01	2

Setelah operasi INSERT ini, trigger UpdateInventory akan dipicu dan mengurangi jumlah stok buku B01 di tabel inventory sebanyak 2.

Berikut adalah bagaimana tabel inventory akan terlihat setelah trigger dijalankan:

BookID	Stock
B01	8
B02	5
B03	7

Dengan demikian, kita bisa melihat bagaimana trigger dapat membantu menjaga integritas data kita.

Catatan: Kamu harus memastikan bahwa basis data kamu mendukung penggunaan trigger dan bahwa kamu memiliki hak akses yang tepat untuk membuat dan mengubah trigger. Sintaks untuk membuat trigger dapat sedikit berbeda tergantuan jenis basis data yang kamu gunakan, jadi pastikan untuk memeriksa dokumentasi basis data kamu.

8.4 Membuat dan Menggunakan Triggers

Triggers adalah prosedur khusus yang dijalankan ketika peristiwa tertentu terjadi dalam database. Peristiwa ini biasanya terkait dengan perubahan data dalam tabel tertentu. Kita bisa membuat trigger untuk menangani perubahan ini dan memastikan bahwa data kita tetap konsisten dan sesuai dengan aturan yang kita tetapkan.

Mari kita lihat sebuah contoh nyata tentang bagaimana membuat dan menggunakan trigger. Sebagai contoh, kita akan menggunakan skenario yang sama seperti pada subbab 8.3 yaitu sistem manajemen toko buku online dengan dua tabel: orders dan inventory.

Tabel orders:

OrderID	BookID	Quantity
1	B01	3
2	B02	1
3	B03	2

Tabel inventory:

BookID	Stock
B01	10
B02	5
B03	7

Kita ingin memastikan bahwa stok buku di inventory akan berkurang setiap kali ada order baru. Untuk itu kita akan membuat trigger dengan nama UpdateInventory yang akan dijalankan setiap kali ada operasi INSERT pada tabel orders.

Berikut adalah syntax untuk membuat trigger tersebut:

```
CREATE TRIGGER UpdateInventory
AFTER INSERT ON orders
FOR EACH ROW
BEGIN
  UPDATE inventory
  SET Stock = Stock - NEW.Quantity
  WHERE BookID = NEW.BookID;
END;
```

Trigger ini akan berjalan setelah operasi INSERT di tabel orders dan akan mengurangi jumlah stok buku di inventory sesuai dengan jumlah buku yang dipesan.

Sekarang, mari kita lihat bagaimana trigger ini berfungsi. Misalkan kita menambahkan order baru sebagai berikut:

```
INSERT INTO orders (OrderID, BookID, Quantity)
VALUES (4, 'B01', 2);
```

Setelah operasi ini, tabel orders akan terlihat seperti ini:

OrderID	BookID	Quantity
1	B01	3
2	B02	1
3	B03	2
4	B01	2

Setelah operasi INSERT, trigger UpdateInventory akan dijalankan dan jumlah stok buku 'B01' di inventory akan berkurang sebanyak 2. Tabel inventory setelah trigger berjalan akan terlihat seperti ini:

BookID	Stock
B01	8
B02	5
B03	7

Dengan demikian, kita bisa melihat bagaimana membuat dan menggunakan trigger bisa membantu kita menjaga integritas data dan mempermudah proses manajemen database kita. Tentunya, ini hanya salah satu contoh penggunaan trigger. Kemampuannya bisa sangat luas tergantung pada kebutuhan aplikasi dan database kamu.

Bab 9. Manajemen Database dan Keamanan

9.1 Membuat Database

Sebelum kita bisa bekerja dengan data dalam SQL, kita perlu tempat untuk menyimpan data tersebut. Tempat ini adalah database. Dalam SQL, database adalah kumpulan tabel yang terstruktur dan saling terhubung, lengkap dengan data dan relasi antar tabel. Oleh karena itu, dalam langkah awal manajemen database, kita harus tahu bagaimana membuat database.

9.1.1 Pengertian Database

Secara singkat, database adalah struktur yang digunakan untuk menyimpan dan mengelola data. Data disimpan dalam bentuk tabel yang saling terkait, dan bisa dengan mudah ditemukan, diakses, dan dikelola dengan bantuan SQL. Database juga berisi metadata yang menjelaskan struktur dan organisasi data dalam tabel.

9.1.2 Membuat Database

Pembuatan database di SQL cukup sederhana. Untuk membuat database, kamu hanya perlu menggunakan perintah CREATE DATABASE, diikuti oleh nama database yang kamu inginkan.

Berikut adalah sintaks SQL untuk membuat database:

```
CREATE DATABASE nama_database;
```

Ketika membuat database, pastikan bahwa nama database:

- Unik dalam server SQL kamu.
- Tidak mengandung spasi.
- Tidak berisi karakter khusus.

Contoh membuat database:

```
CREATE DATABASE BookStore;
```

Perintah di atas akan membuat database baru dengan nama BookStore.

9.1.3 Memilih Database

Setelah membuat database, kamu harus memilih database yang akan digunakan dengan perintah USE diikuti oleh nama database.

Berikut adalah sintaks SQL untuk menggunakan database:

```
USE nama_database;
```

Contoh menggunakan database:

```
USE BookStore;
```

Perintah di atas akan memilih database BookStore untuk digunakan dalam sesi SQL saat ini.

9.1.4 Mengonfirmasi Pembuatan Database

Untuk memastikan bahwa database telah berhasil dibuat, kamu bisa menggunakan perintah SHOW DATABASES. Perintah ini akan menampilkan semua database yang ada di server SQL kamu.

Berikut adalah sintaks SQL untuk menampilkan semua database:

```
SHOW DATABASES;
```

Contoh menampilkan semua database:

```
SHOW DATABASES;
```

Output:

Database
information_schema
mysql
performance_schema

Seperti yang terlihat dalam output contoh di atas, database BookStore telah berhasil dihapus.

Menghapus database adalah tindakan besar dalam manajemen database. Meski prosesnya sederhana, tapi harus dilakukan dengan hati-hati karena semua data dalam database akan hilang secara permanen. Oleh karena itu, sebelum melakukan penghapusan, pastikan kamu telah membackup semua data penting dan memastikan tidak ada user atau aplikasi yang sedang menggunakan database tersebut.

9.4 Menghapus Tabel

Setelah kita membahas bagaimana membuat tabel dalam database, sekarang saatnya untuk memahami bagaimana menghapus tabel. Menghapus tabel dalam database bisa menjadi tugas yang sangat penting dan juga berisiko tinggi karena kamu akan kehilangan semua data dalam tabel tersebut. Oleh karena itu, penting untuk memahami bagaimana dan kapan harus menghapus tabel dalam database.

9.4.1 Pengertian Menghapus Tabel

Menghapus tabel berarti menghilangkan struktur tabel beserta semua data yang ada di dalamnya dari database. Seperti halnya menghapus database, ini adalah tindakan yang tidak bisa dibatalkan. Oleh karena itu, sebelum menghapus tabel, pastikan kamu sudah mem-backup semua data yang penting.

9.4.2 Menghapus Tabel

Untuk menghapus tabel, kamu bisa menggunakan perintah DROP TABLE, diikuti oleh nama tabel yang ingin kamu hapus.

Berikut adalah sintaks SQL untuk menghapus tabel:

```
DROP TABLE nama_tabel;
```

Ketika menghapus tabel, pastikan bahwa:

Kamu telah membackup semua data yang penting.
Tidak ada user atau aplikasi yang sedang menggunakan tabel tersebut.

Contoh menghapus tabel:

Misalkan kamu memiliki tabel 'Orders' dengan data sebagai berikut:

OrderID	CustomerID	OrderDate
1	2	2023-01-01
2	1	2023-02-02
3	3	2023-03-03

Syntax untuk menghapus tabel 'Orders':

```
DROP TABLE Orders;
```

Perintah di atas akan menghapus tabel Orders.

9.4.3 Mengonfirmasi Penghapusan Tabel

Untuk memastikan bahwa tabel telah berhasil dihapus, kamu bisa menggunakan perintah SHOW TABLES. Jika tabel yang telah kamu hapus tidak muncul dalam daftar, itu berarti penghapusan berhasil.

Berikut adalah sintaks SQL untuk menampilkan semua tabel:

```
SHOW TABLES;
```

Menghapus tabel adalah langkah penting dalam manajemen database. Meski prosesnya sederhana, tapi harus dilakukan dengan hati-hati karena semua data dalam tabel akan hilang secara permanen. Oleh karena itu, sebelum melakukan penghapusan, pastikan kamu telah membackup semua data penting dan memastikan tidak ada user atau aplikasi yang sedang menggunakan tabel tersebut.

9.5 Membuat User dan Roles

Dalam manajemen database, konsep user dan roles sangat penting. User dan roles memungkinkan kita untuk mengatur siapa saja yang dapat mengakses dan memodifikasi data dalam database. Dalam subbab ini, kita akan membahas bagaimana cara membuat user dan roles.

9.5.1 Pengertian User dan Roles

User adalah individu atau aplikasi yang dapat mengakses database. Setiap user akan memiliki hak akses atau privileges tertentu yang menentukan apa yang bisa mereka lakukan dalam database. Misalnya, user bisa memiliki hak untuk melihat data, memodifikasi data, atau bahkan menghapus data.

Role adalah kumpulan hak akses yang dapat diberikan kepada user. Dengan roles, kita tidak perlu mengatur hak akses untuk setiap user secara individu. Cukup buat role dengan kumpulan hak akses tertentu, lalu berikan role tersebut kepada user.

9.5.2 Membuat User
Membuat user dalam database biasanya dilakukan oleh administrator database. Prosesnya melibatkan penentuan nama user, password, dan hak akses yang akan dimiliki user tersebut.

Berikut adalah sintaks SQL untuk membuat user:

```
CREATE USER 'nama_user'@'localhost' IDENTIFIED BY 'password';
```

Misalkan kita ingin membuat user baru dengan nama 'userbaru' dan password 'passwordbaru'. Kita bisa melakukannya dengan perintah berikut:

```
CREATE USER 'userbaru'@'localhost' IDENTIFIED BY 'passwordbaru';
```

9.5.3 Membuat Roles

Setelah membuat user, langkah selanjutnya adalah membuat roles. Role adalah cara yang efisien untuk mengelola hak akses dalam database. Dengan role, kita bisa menentukan kumpulan hak akses, lalu menerapkannya ke beberapa user sekaligus.

Berikut adalah sintaks SQL untuk membuat role:

```
CREATE ROLE nama_role;
```

Misalkan kita ingin membuat role baru dengan nama 'rolebaru'. Kita bisa melakukannya dengan perintah berikut:

```
CREATE ROLE rolebaru;
```

9.5.4 Memberikan Hak Akses ke User dan Roles

Setelah membuat user dan roles, langkah selanjutnya adalah memberikan hak akses kepada mereka. Hak akses menentukan apa yang bisa dilakukan user atau role dalam database.

Berikut adalah sintaks SQL untuk memberikan hak akses ke user:

```
GRANT hak_akses ON nama_database.* TO 'nama_user'@'localhost';
```

Dan berikut adalah sintaks SQL untuk memberikan hak akses ke role:

```
GRANT hak_akses ON nama_database.* TO nama_role;
```

Misalkan kita ingin memberikan hak akses SELECT, INSERT, dan UPDATE ke user 'userbaru' pada database 'dbtest'. Kita bisa melakukannya dengan perintah berikut:

```
GRANT SELECT, INSERT, UPDATE ON dbtest.* TO 'userbaru'@'localhost';
```

Kemudian, kita bisa memberikan hak akses yang sama ke role 'rolebaru' dengan perintah berikut:

```
GRANT SELECT, INSERT, UPDATE ON dbtest.* TO rolebaru;
```

Dalam prakteknya, setelah membuat user dan role, biasanya akan diikuti dengan proses pengaturan hak akses atau privileges, seperti yang akan kita bahas di subbab selanjutnya.

Dengan pengaturan yang tepat, kita bisa memastikan bahwa setiap user hanya memiliki akses yang mereka butuhkan untuk melaksanakan tugas mereka, sehingga membantu menjaga keamanan dan integritas data dalam database.

Bab 10. Tips dan Trik

10.1 Optimasi Query SQL

Pemrograman database adalah suatu seni, dan seperti seni lainnya, ada banyak cara untuk mencapai hasil yang diinginkan. Tetapi tidak semua cara sama efisien atau cepatnya. Di sinilah optimasi query SQL masuk. Dalam subbab ini, kita akan membahas beberapa teknik dan strategi untuk mempercepat query SQL dan membuat database bekerja lebih efisien. Ingatlah bahwa setiap database unik, dan apa yang bekerja paling baik mungkin berbeda dari kasus ke kasus. Namun, dengan menggunakan prinsip-prinsip dasar ini sebagai panduan, kamu bisa merancang dan mengimplementasikan query SQL yang lebih cepat dan efisien.

10.1.1 Mengerti Struktur Data dan Indexing

Salah satu aspek paling penting dari optimasi SQL adalah memahami struktur data dan bagaimana database mengindeks dan mencari data tersebut. Indeks adalah struktur data khusus yang mempercepat pencarian dan akses ke data dalam database.

Misalkan kita memiliki tabel 'pegawai' dengan kolom 'id', 'nama', dan 'gaji'. Jika kita ingin mencari pegawai dengan gaji tertentu, database harus membaca setiap baris tabel 'pegawai' untuk mencari nilai 'gaji' yang kita cari. Ini bisa sangat lambat jika tabel 'pegawai' berisi jutaan baris. Tetapi jika kita membuat indeks pada kolom 'gaji', database bisa mencari nilai 'gaji' yang kita cari jauh lebih cepat.

```
CREATE INDEX idx_gaji
ON pegawai (gaji);
```

10.1.2 Memilih Kolom yang Dibutuhkan

Salah satu kesalahan umum yang dilakukan programmer adalah menggunakan SELECT * dalam query mereka. Ini berarti bahwa database harus mengambil semua kolom dari tabel, yang bisa memperlambat query jika tabel memiliki banyak kolom atau beberapa kolom berisi banyak data.

Sebagai gantinya, sebaiknya pilih hanya kolom yang kamu butuhkan dalam query. Misalkan kamu hanya perlu 'id' dan 'nama' dari tabel 'pegawai', kamu bisa menulis query seperti ini:

```
SELECT id, nama
FROM pegawai;
```

10.1.3 Menghindari Fungsi dalam WHERE Clause

Menggunakan fungsi dalam WHERE clause bisa membuat query berjalan lambat, karena database harus menjalankan fungsi tersebut untuk setiap baris dalam tabel. Misalkan kamu memiliki query seperti ini:

```
SELECT id, nama
FROM pegawai
WHERE YEAR(tanggal_lahir) = 1990;
```

Dalam query ini, database harus menjalankan fungsi YEAR() untuk setiap baris dalam tabel 'pegawai'. Sebagai gantinya, coba tulis querymu agar database tidak perlu menjalankan fungsi untuk setiap baris. Contoh:

```
SELECT id, nama
FROM pegawai
WHERE tanggal_lahir BETWEEN '1990-01-01' AND '1990-12-31';
```

10.1.4 Menggunakan JOIN dengan Bijak

JOIN adalah operasi yang mahal dari segi kinerja, karena database harus mencocokkan baris dari satu tabel dengan baris di tabel lain. Selalu coba untuk meminimalkan jumlah JOIN dalam query kamu. Dan jika kamu harus menggunakan JOIN, pastikan kamu menggunakan tipe JOIN yang paling efisien untuk situasimu.

```
SELECT p.id, p.nama, d.nama AS departemen
FROM pegawai p
INNER JOIN departemen d ON p.departemen_id = d.id;
```

Di query ini, kita menggunakan INNER JOIN untuk menggabungkan tabel 'pegawai' dan 'departemen'. Tapi jika kita hanya ingin melihat pegawai yang memiliki departemen, kita bisa menggunakan LEFT JOIN yang biasanya lebih cepat:

```
SELECT p.id, p.nama, d.nama AS departemen
FROM pegawai p
LEFT JOIN departemen d ON p.departemen_id = d.id;
```

10.1.5 Menggunakan LIMIT

Jika kamu hanya membutuhkan sejumlah hasil tertentu, pastikan untuk menggunakan LIMIT dalam query kamu. Misalkan kamu hanya ingin melihat 10 pegawai dengan gaji tertinggi, kamu bisa menulis:

```
SELECT id, nama, gaji
FROM pegawai
ORDER BY gaji DESC
```

```
LIMIT 10;
```

Ini jauh lebih cepat daripada mengambil semua pegawai dan kemudian menyaring hasilnya di aplikasi kamu.

10.1.6 Memanfaatkan Explain Plan

Banyak database SQL modern memiliki fitur yang disebut explain plan, yang bisa membantu kamu memahami bagaimana database akan menjalankan query kamu. Ini bisa sangat berguna untuk mengidentifikasi bagian dari query kamu yang mungkin memperlambat kinerja.

```
EXPLAIN SELECT id, nama, gaji
FROM pegawai
ORDER BY gaji DESC;
```

Dengan memahami dan menerapkan teknik-teknik ini, kamu bisa membuat query SQL yang lebih cepat dan efisien. Tetapi ingatlah bahwa optimasi adalah proses iteratif, dan sering kali diperlukan beberapa percobaan untuk menemukan solusi terbaik.

10.2 Menghindari SQL Injection

SQL Injection adalah serangan yang dapat mengeksploitasi celah keamanan yang terjadi saat software tidak memvalidasi input yang diberikan oleh user dan memasukkannya langsung ke dalam perintah SQL. Serangan ini bisa sangat merusak, memungkinkan penyerang untuk melihat data yang seharusnya tidak mereka lihat, memodifikasi atau menghapus data, atau bahkan menjalankan perintah arbitrer pada server. Untuk menghindari kerusakan ini, sangat penting untuk mengetahui cara mencegah SQL Injection.

10.2.1 Apa itu SQL Injection?

Sebagai contoh awal, kita akan melihat skenario yang umum. Misalkan kamu memiliki aplikasi web yang memungkinkan pengguna untuk log in. Back-end aplikasi kamu mungkin menjalankan query SQL semacam ini untuk memeriksa apakah username dan password benar:

```
SELECT * FROM users WHERE username = '[username]' AND password =
'[password]';
```

Di sini, [username] dan [password] adalah input yang diberikan oleh user. Jika seorang penyerang mengisi form login dengan nilai tertentu, mereka bisa membuat query yang

benar-benar berbeda. Misalkan mereka memasukkan ' OR '1'='1 sebagai username dan password. Query yang dihasilkan adalah:

```
SELECT * FROM users WHERE username = '' OR '1'='1' AND password = '' OR
'1'='1';
```

Karena '1'='1' selalu benar, query ini akan mengembalikan setiap baris dari tabel users, sehingga memungkinkan penyerang untuk log in sebagai pengguna pertama dalam tabel tersebut.

10.2.2 Cara Mencegah SQL Injection

Ada beberapa cara untuk mencegah SQL Injection. Metode yang paling ampuh adalah dengan menggunakan prepared statements dan parameterized queries. Ini adalah fitur dari banyak bahasa pemrograman yang memungkinkan kamu untuk menulis query SQL dengan placeholder untuk input user, dan kemudian menyediakan input tersebut sebagai parameter terpisah.

Misalnya, di PHP, kamu bisa menggunakan fungsi prepare dan bind_param dari kelas mysqli untuk membuat query yang aman dari SQL Injection:

```
$stmt = $mysqli->prepare("SELECT * FROM users WHERE username = ? AND
password = ?");
$stmt->bind_param("ss", $username, $password);

$username = $_POST["username"];
$password = $_POST["password"];

$stmt->execute();
```

Di sini, "?" dalam query adalah placeholder untuk input user. Fungsi bind_param kemudian digunakan untuk menyediakan nilai sebenarnya untuk placeholder tersebut. "ss" menunjukkan bahwa kedua parameter adalah string. Fungsi prepare dan bind_param secara otomatis akan memastikan bahwa input user tidak dapat mempengaruhi struktur query SQL itu sendiri, sehingga mencegah SQL Injection.

10.2.3 Keberlanjutan Perlindungan

Meskipun penggunaan prepared statements dan parameterized queries akan melindungi aplikasi kamu dari SQL Injection dalam banyak kasus, penting untuk tetap waspada terhadap ancaman lain. Pastikan kamu selalu memperbarui software dan library yang kamu gunakan ke versi terbaru, dan memahami praktik keamanan terbaik dalam pengembangan aplikasi web.

Juga penting untuk melakukan sanitasi dan validasi input. Sanitasi input berarti membersihkan input pengguna dari elemen yang tidak diinginkan atau berbahaya, seperti tag HTML yang mungkin digunakan dalam serangan cross-site scripting (XSS). Validasi input berarti memeriksa apakah input pengguna sesuai dengan ekspektasi kamu, misalnya dengan mengecek apakah alamat email yang dimasukkan memiliki format yang valid.

Secara keseluruhan, menghindari SQL Injection melibatkan kombinasi dari menulis kode yang aman, menggunakan software yang up-to-date dan aman, dan memiliki pemahaman yang baik tentang keamanan web.

10.3 Debugging SQL

Debugging SQL sering menjadi tantangan tersendiri. SQL atau Structured Query Language merupakan bahasa khusus yang digunakan untuk berinteraksi dengan database, dan tidak seperti bahasa pemrograman lainnya, kamu tidak dapat dengan mudah memperiksa nilai variabel, memasukkan pernyataan cetak untuk melihat apa yang terjadi, atau menghentikan eksekusi di tengah jalan untuk memeriksa kondisi sistem. Oleh karena itu, debugging SQL memerlukan pendekatan yang berbeda dan alat yang berbeda. Berikut ini adalah beberapa tips dan teknik untuk debugging SQL.

10.3.1 Memahami Kesalahan

Seperti dalam semua bentuk debugging, langkah pertama dalam debugging SQL adalah memahami pesan kesalahan. Pesan kesalahan dari sistem manajemen basis data (DBMS) biasanya akan memberi tahu kamu baris mana dalam kode SQL yang menyebabkan kesalahan, dan jenis kesalahan apa yang terjadi. Misalnya, kesalahan sintaks akan muncul jika kamu membuat kesalahan ketik dalam pernyataan SQL, atau jika kamu mencoba menggunakan fitur yang tidak didukung oleh DBMS kamu.

Jika pesan kesalahan tidak cukup jelas, kamu dapat mencari kode kesalahan atau teks pesan di dokumentasi DBMS atau di web. Kemungkinan besar, kamu akan menemukan penjelasan yang lebih rinci tentang apa yang bisa menyebabkan kesalahan tersebut.

10.3.2 Menyederhanakan Masalah

Jika query SQL kamu rumit dan menghasilkan hasil yang tidak diharapkan, sering kali membantu untuk mencoba dan menyederhanakan masalah. Ini bisa berarti memecah query menjadi bagian-bagian yang lebih kecil dan mengeksekusi masing-masing bagian secara terpisah, untuk melihat apakah mereka menghasilkan hasil yang diharapkan.

Misalnya, jika kamu memiliki query SELECT yang melibatkan beberapa JOIN, cobalah untuk menjalankan masing-masing JOIN secara terpisah untuk melihat apakah mereka

menghasilkan baris yang benar. Atau, jika kamu memiliki subquery yang rumit, coba ekstrak dan jalankan subquery tersebut sendiri.

10.3.3 Menggunakan EXPLAIN PLAN

Sebagian besar DBMS memiliki perintah atau fungsi yang dapat menunjukkan "rencana eksekusi" untuk query SQL: urutan langkah yang diambil oleh DBMS untuk menjalankan query tersebut. Dalam MySQL dan PostgreSQL, kamu bisa mendapatkan rencana ini dengan menambahkan EXPLAIN sebelum pernyataan SQL kamu.

Rencana eksekusi dapat memberi kamu wawasan tentang bagaimana DBMS memahami query kamu, dan mungkin bisa membantu kamu melihat mengapa query tidak berfungsi seperti yang kamu harapkan. Misalnya, rencana tersebut dapat menunjukkan bahwa DBMS melakukan full table scan ketika kamu mengharapkan index untuk digunakan, yang mungkin menunjukkan bahwa ada masalah dengan index tersebut atau dengan cara kamu menulis query kamu.

10.3.4 Menggunakan Alat Debugging SQL

Ada juga alat debugging SQL yang bisa kamu gunakan, tergantung pada DBMS dan lingkungan kerja kamu. Misalnya, beberapa IDE pengembangan memiliki fitur debugging SQL built-in, yang dapat menampilkan hasil dari setiap langkah dalam query SQL, menunjukkan nilai dari variabel dan parameter, dan sebagainya.

Namun, sebelum kamu menggunakan alat ini, pastikan bahwa kamu sudah memiliki pemahaman yang cukup tentang SQL dan bagaimana DBMS bekerja. Alat-alat ini bisa sangat berguna, tetapi mereka tidak dapat menggantikan pengetahuan dan pemahaman kamu sendiri tentang apa yang kamu coba capai dengan kode SQL kamu.

10.4 Best Practices dalam Menulis Query SQL

Menulis query SQL yang efisien dan mudah dipahami bukanlah tugas yang mudah. Tapi, dengan pemahaman yang baik tentang SQL dan beberapa best practices, kamu bisa menulis query SQL yang optimal dan mudah dipelihara. Berikut adalah beberapa best practices dalam menulis query SQL yang harus kamu perhatikan.

Gunakan Nama Kolom yang Jelas

Ketika menulis query SQL, selalu berikan nama yang jelas dan deskriptif untuk setiap kolom dalam tabel kamu. Nama kolom yang jelas akan membuat kode SQL kamu lebih mudah dibaca dan dipahami, baik oleh kamu atau orang lain yang mungkin perlu memelihara kode kamu di masa mendatang.

Sebagai contoh, daripada menggunakan nama kolom seperti 'c1', 'c2', dan 'c3', gunakan nama yang lebih deskriptif seperti 'customer_name', 'order_date', dan 'total_purchase'.

Gunakan Aliasing dengan Bijak

Aliasing adalah teknik yang digunakan untuk memberikan nama baru sementara untuk tabel atau kolom dalam query SQL. Aliasing bisa sangat membantu dalam membuat kode SQL kamu lebih mudah dibaca, terutama ketika kamu bekerja dengan banyak tabel atau kolom dengan nama yang panjang atau rumit.

Namun, pastikan juga bahwa alias yang kamu gunakan masih membuat sense dan tidak menyesatkan. Sebagai contoh, menggunakan alias 'o' untuk tabel 'orders' dan 'c' untuk 'customers' mungkin masuk akal, tetapi menggunakan 'x' dan 'y' mungkin membuat kode kamu lebih sulit untuk dipahami.

Hindari Penggunaan Wildcards secara Berlebihan

Penggunaan wildcard (misalnya, SELECT * FROM table_name) bisa menjadi praktis dalam beberapa situasi, tetapi biasanya bukan ide yang baik untuk digunakan dalam produksi. Penggunaan wildcard dapat menambah beban pada database karena memerlukan waktu dan memori lebih banyak untuk mengambil semua kolom. Selalu lebih baik untuk secara spesifik menyebutkan kolom apa saja yang kamu butuhkan dalam query kamu.

Gunakan Komentar

Komentar sangat penting dalam kode apapun, termasuk SQL. Mereka dapat digunakan untuk menjelaskan apa yang dilakukan oleh sebagian kode, kenapa kamu membuat keputusan tertentu, atau untuk membuat catatan tentang hal-hal yang mungkin perlu diperhatikan atau diubah di masa mendatang.

Dalam SQL, kamu bisa membuat komentar dengan menggunakan simbol -- untuk komentar satu baris, atau /* ... */ untuk komentar yang lebih panjang.

Format Query SQL dengan Baik

Penyusunan dan indentasi query SQL dengan baik akan membuatnya lebih mudah dibaca dan dipahami. Beberapa DBMS juga menyediakan alat untuk membantu kamu dengan ini.

Sebagai contoh, kamu bisa memulai setiap klausa baru (seperti SELECT, FROM, WHERE, dan sebagainya) di baris baru, dan menggunakan indentasi untuk menunjukkan struktur query.

Test dan Review Query Kamu

Sebelum kamu menjalankan query SQL dalam lingkungan produksi, pastikan untuk selalu mengujinya terlebih dahulu dalam lingkungan pengembangan. Ini akan membantu kamu untuk menemukan dan memperbaiki kesalahan sebelum mereka berdampak pada sistem kamu.

Selain itu, penting juga untuk melakukan code review. Minta rekan kerja kamu untuk melihat kode kamu, atau gunakan alat otomatis untuk membantu kamu menemukan potensi masalah.

Ketahui Kapan Harus Menggunakan Index

Indexing adalah teknik yang digunakan untuk meningkatkan kinerja query dalam banyak situasi, tetapi mereka juga bisa mempengaruhi kinerja negatif jika digunakan secara tidak tepat.

Sebagai contoh, jika kamu sering melakukan query pada kolom tertentu, mungkin akan bermanfaat untuk membuat index pada kolom tersebut. Tetapi jika tabel tersebut sering mengalami update, insert, atau delete, maka index mungkin akan memperlambat operasi tersebut.

Jadi, penting untuk memahami cara kerja index, dan tahu kapan harus dan tidak harus menggunakannya.

Pelajari dan Pahami Fitur DBMS Kamu

Setiap DBMS memiliki fitur dan sintaks yang berbeda-beda. Kamu harus memahami fitur apa saja yang ditawarkan oleh DBMS yang kamu gunakan, dan bagaimana kamu bisa memanfaatkannya untuk memaksimalkan efisiensi dan kinerja query SQL kamu.

Misalnya, beberapa DBMS memiliki fungsi dan operator khusus yang bisa membantu kamu menyelesaikan tugas tertentu dengan lebih efisien. Beberapa DBMS juga memiliki fitur optimasi query otomatis, yang bisa membantu kamu menulis query yang lebih efisien tanpa perlu mengubah kode kamu.

10.4.9 Gunakan Stored Procedure dan Views jika Memungkinkan
Stored procedure dan views bisa sangat membantu dalam memperbaiki kinerja dan memudahkan pemeliharaan kode SQL kamu. Stored procedure memungkinkan kamu untuk menyimpan dan menjalankan blok kode SQL yang sering digunakan, sementara views memungkinkan kamu untuk menyimpan dan menjalankan query yang kompleks dengan lebih mudah.

Tapi, seperti halnya fitur lainnya, penting untuk memahami kapan dan bagaimana menggunakan stored procedure dan views, dan juga tahu kapan sebaiknya tidak digunakan.

Penutup

Belajar SQL bukanlah hal yang bisa dilakukan dalam satu hari. Kamu harus terus belajar dan berlatih untuk menjadi mahir dalam SQL. Cobalah untuk selalu update dengan fitur terbaru dari DBMS yang kamu gunakan, dan jangan takut untuk mencoba hal baru.

Ingatlah juga bahwa menulis query SQL yang baik bukan hanya tentang membuatnya bekerja, tetapi juga tentang membuatnya mudah dipahami, mudah dipelihara, dan efisien.

Terima Kasih