SISTEM BASIS DATA: KONSEP, DESAIN DAN IMPLEMENTASI

Muhammad Fairuzabadi
Angga Aditya Permana
Melissa Indah Fianty
Erick Fernando
Raymond Sunardi Oetama
Weri Sirait
Nur Azizah
Heru Saputra
Wirawan Istiono
Bambang Suprayogi
Boy Firmansyah
Suwito Pomalingo
Rahmat Taufik R.L Bau

GET PRESS INDONESIA

SISTEM BASIS DATA: KONSEP, DESAIN DAN IMPLEMENTASI

Penulis :
Muhammad Fairuzabadi
Angga Aditya Permana
Melissa Indah Fianty
Erick Fernando
Raymond Sunardi Oetama
Weri Sirait
Nur Azizah
Heru Saputra
Wirawan Istiono
Bambang Suprayogi
Boy Firmansyah
Suwito Pomalingo
Rahmat Taufik R.L Bau

ISBN :

Editor : Ariyanto M.Pd
Penyunting : Tri Putri Wahyuni,S.Pd
Desain Sampul dan Tata Letak : Atyka Trianisa, S.Pd

Penerbit : GET PRESS INDONESIA
Anggota IKAPI No. 033/SBA/2022

Redaksi :
Jln. Palarik Air Pacah No 26 Kel. Air Pacah
Kec. Koto Tangah Kota Padang Sumatera Barat
Website : www. getpress.co.id
Email : adm.getpress@gmail.com

Cetakan pertama, Oktober 2023

KATA PENGANTAR

Segala Puji dan syukur atas kehadirat Allah SWT dalam segala kesempatan. Sholawat beriring salam dan doa kita sampaikan kepada Nabi Muhammad SAW. Alhamdulillah atas Rahmat dan Karunia-Nya penulis telah menyelesaikan Buku Sistem Basis Data: Konsep, Desain Dan Implementasi ini.

Buku ini membahas Konsep Dasar Basis Data, Model-model Basis Data, Sistem Manajemen Basis Data, Desain Basis Data, Normalisasi, SQL: Data Definition Languange, SQL: Data Manipulatin Language, SQL: Fungsi-Fungsi, SQL: Sub Query, dan Join, SQL: View dan Store Prosedure, SQL: Transaksi dan Manajemen Pengguna, Basis Data NoSQL, Migrasi dan Konversi Basis Data.

Proses penulisan buku ini berhasil diselesaikan atas kerjasama tim penulis. Demi kualitas yang lebih baik dan kepuasan para pembaca, saran dan masukan yang membangun dari pembaca sangat kami harapkan.

Penulis ucapkan terima kasih kepada semua pihak yang telah mendukung dalam penyelesaian buku ini. Terutama pihak yang telah membantu terbitnya buku ini dan telah mempercayakan mendorong, dan menginisiasi terbitnya buku ini. Semoga buku ini dapat bermanfaat bagi masyarakat Indonesia.

Padang, Oktober 2023

Penulis

DAFTAR ISI

DAFTRA GAMBAR

DAFTAR TABEL

BAB 1
KONSEP DASAR BASIS DATA

Oleh Muhammad Fairuzabadi

1.1 Basis Data

Basis data, atau lebih dikenal dengan istilah database, merupakan fondasi penting dalam dunia teknologi informasi. Konsep dasar basis data memainkan peran krusial dalam memahami bagaimana data diorganisir, diakses, dan dikelola secara efisien. Dengan begitu, pemahaman akan konsep dasar basis data menjadi kunci dalam merancang sistem informasi yang handal dan responsif. Dalam konteks ini, artikel ini akan membahas secara komprehensif mengenai konsep dasar basis data dan relevansinya dalam era teknologi yang terus berkembang.

1.1.1 Definisi Data

Sebelum kita membahas definisi basis data maka perlu dipahami dulu tentanng definisi data. Data dapat didefinisikan sebagai kumpulan fakta, angka, karakter, atau simbol yang mewakili fakta-fakta mentah tentang sesuatu. Data belum diolah atau diberikan makna tertentu, dan merupakan representasi dari realitas yang dapat diamati atau diukur (Muttaqin, Sondakh, *et al.*, 2023).

Data dapat berupa berbagai bentuk, seperti teks, angka, gambar, suara, atau kombinasi dari semuanya. Penting untuk dipahami bahwa data sendiri belum memberikan pemahaman atau informasi yang bermakna. Namun, data menjadi bahan dasar yang diperlukan dalam proses analisis, interpretasi, dan pengolahan lebih lanjut untuk menghasilkan informasi yang berguna dan memiliki makna bagi penerima informasi (Kadir, 2014).

Pada tabel berikut, berbagai jenis data dijelaskan dengan deskripsi singkat, serta diberikan contoh untuk masing-masing jenis data. Tabel ini memudahkan pemahaman tentang beragam jenis data yang ada dalam konteks yang terstruktur dan mudah dibaca.

Tabel 1.6. Jenis-Jenis Data

Jenis	Deskripsi	Contoh
Data Angka	Angka atau bilangan	123456
	Desimal atau pecahan	3.14
	Bilangan bulat negatif	-25
Data Teks	Teks atau kata-kata	"Hello, World!"
	Kalimat atau frasa	"Data dan Informasi"
Data Tanggal dan Waktu	Tanggal dalam berbagai format	2023-08-03
	Tanggal dengan format lain	08/03/2023
	Tanggal dengan format bahasa Indonesia	03 Agustus 2023
	Waktu dalam format jam:menit:detik	12:30:45
Data Karakter	Karakter atau huruf	'A'
	Karakter dalam huruf kecil	'b'
	Simbol atau karakter khusus	'@'
Data Gambar	File gambar dalam format JPEG atau PNG	File gambar JPEG atau PNG

Jenis	Deskripsi	Contoh
Data Suara	File audio dalam format MP3, WAV, atau OGG	File audio dalam format MP3
Data Geospasial	Koordinat lintang dan bujur suatu lokasi	40.7128° N, 74.0060° W
Data Kategori	Kategori atau jenis produk	"Elektronik"
	Kategori lainnya	"Makanan"
Data Status	Status atau keadaan	"Aktif"
	Status lainnya	"Lulus"
Data Identitas	Nomor identitas individu, seperti NIS atau NIK	Nomor induk siswa (NIS)
	Nomor rekening bank	Nomor rekening bank

Berikut adalah contoh jenis data dalam bentuk tabulasi dengan beberapa atribut atau kolom yang mencakup jenis-jenis data yang berbeda:

Tabel 1.7. Contoh Data dengan Berbagai Jenis Data

No.	Nama	Gaji	Pekerjaan	Status Menikah	Tanggal Lahir
1	John Doe	5000	Engineer	Belum Menikah	1993-05-15
2	Jane Smith	4000	Marketing	Menikah	1998-09-28
3	Mark Johnson	6000	Sales	Belum Menikah	1993-11-10
4	Emily Brown	5500	Accountant	Menikah	1989-07-20

Tabel di atas menampilkan contoh data dalam bentuk tabulasi dengan beberapa jenis data yang berbeda, yaitu:

1. Data Teks: Nama, Pekerjaan, dan Status Menikah adalah contoh data teks yang merepresentasikan nama individu, jenis pekerjaan, dan status pernikahan (belum menikah atau menikah).
2. Data Angka: Gaji adalah contoh data angka yang merepresentasikan usia individu dalam tahun dan jumlah gaji dalam bentuk mata uang.
3. Data Tanggal: Tanggal Lahir adalah contoh data tanggal yang merepresentasikan tanggal lahir individu dalam format YYYY-MM-DD.

1.1.2 Definisi Basis Data

Basis data memiliki peran penting dalam mendukung aplikasi dan sistem informasi modern, termasuk sistem manajemen perusahaan, sistem e-commerce, sistem perbankan, sistem kepegawaian, dan banyak lagi. Dengan adanya basis data, organisasi dapat mengelola data dengan lebih terstruktur, aman, dan efisien, sehingga mendukung pengambilan keputusan yang lebih baik dan menyediakan layanan yang lebih baik kepada pengguna(Tantriawan *et al.*, 2023). Berikut adalah definisi basis data dari berbagai referensi:

Tabel 1.8. Definisi Basis Data dari Berbegai Refrensi

No	Sumber Referensi	Definisi
1	Oracle	Kumpulan data terstruktur yang disimpan dalam komputer dan dapat diakses serta dikelola menggunakan DBMS.
2	IBM	Kumpulan data terorganisir dan terstruktur dalam komputer yang diatur dalam tabel untuk akses dan

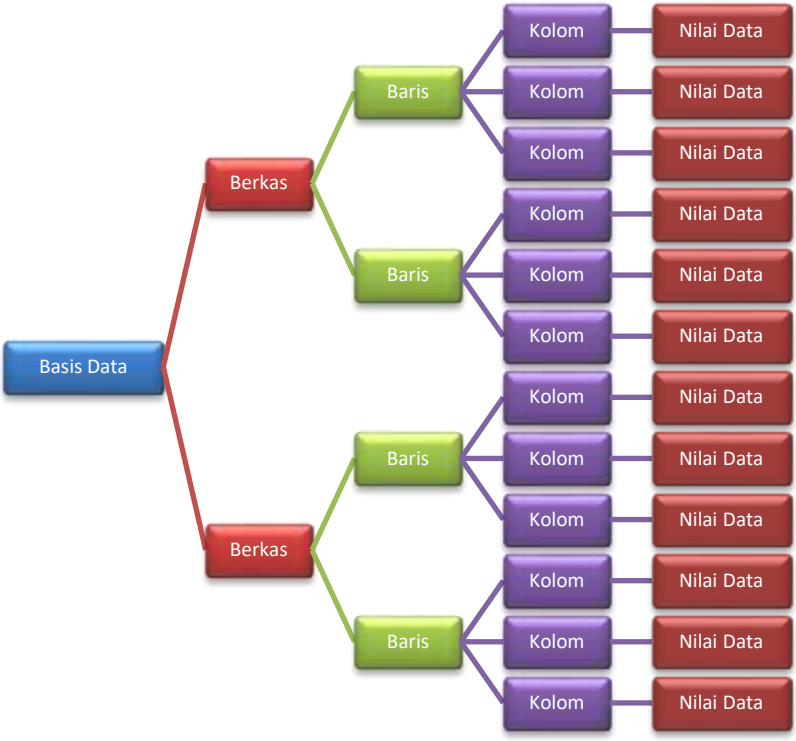
No	Sumber Referensi	Definisi
		pengelolaan.
3	Microsoft	Kumpulan data terstruktur yang diatur dengan baik dalam komputer, diakses, dan dikelola melalui perangkat lunak.
4	Fundamentals of Database Systems (Elmasri dan Navathe, 2015)	Kumpulan data terintegrasi yang disimpan bersama, mengizinkan akses dan pengelolaan data dengan efisien.
5	Database Management Systems (Ramakrishnan dan Gehrke, 2003)	Kumpulan data terorganisir yang diakses melalui perangkat lunak pengelola basis data untuk interaksi dan akses.

Merangkung definisi basis data yang disampaikan oleh berbagai referensi tersebut Basis data dapat didefinisikan kumpulan data terstruktur yang disimpan dalam komputer dan diakses serta dikelola melalui perangkat lunak pengelola basis data (DBMS). Data dalam basis data diorganisir dalam bentuk tabel yang terdiri dari baris dan kolom, memungkinkan akses dan pengelolaan data dengan efisien dan terstruktur. Basis data membantu menyimpan, mengatur, dan mengintegrasikan data sehingga memungkinkan pengguna dan aplikasi untuk mengakses serta memanipulasi data dengan mudah.

1.1.3 Hirarki Data

Hirarki data adalah konsep penting dalam dunia teknologi informasi yang mencerminkan struktur dan pengorganisasian data dari tingkatan terendah hingga tingkatan tertinggi. Hirarki data memungkinkan pengorganisasian data mulai dari unit terkecil (nilai data) hingga keseluruhan koleksi data dalam basis data,

mempermudah pengelolaan, akses, dan analisis data secara sistematis dan terstruktur. Berikut adalah penjelasan lebih detail tentang hirarki data dari yang terbesar ke yang terkecil.



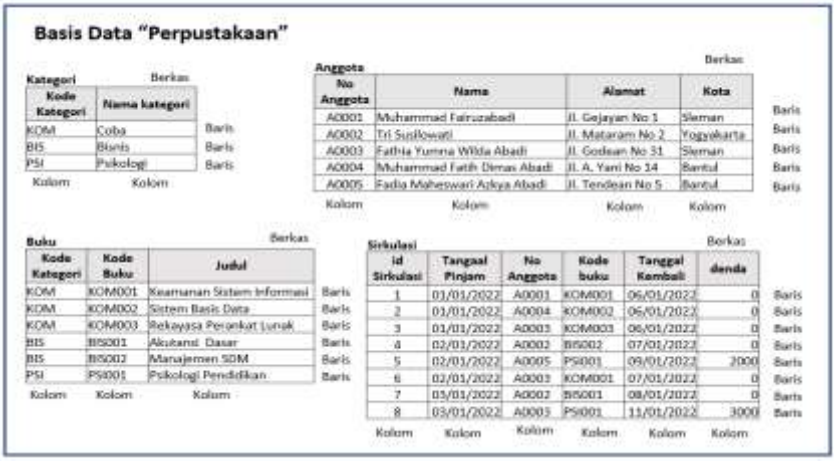
Gambar 1.6. Hirarki Data

1. Basis Data (*Database*)
   Basis data merupakan tingkatan tertinggi dalam hirarki data. Ini mencakup kumpulan file dan struktur data yang terorganisir dan terhubung dengan baik. Basis data berfungsi sebagai gudang sentral yang menyimpan berbagai informasi dari berbagai sumber. Dalam basis data, data diatur dalam

tabel-tabel yang memiliki relasi atau hubungan dengan data lainnya. Basis data memungkinkan penyimpanan data dalam skala besar dan kompleks, serta menyediakan mekanisme untuk mengakses, mengelola, dan memanipulasi data secara efisien. Basis data sering diimplementasikan menggunakan Sistem Manajemen Basis Data (*Database Management System* - DBMS) yang menyediakan antarmuka untuk berinteraksi dengan data.

2. Berkas (*File*)

   File adalah kumpulan record yang memiliki kesamaan atau hubungan tertentu. File merupakan unit penyimpanan data yang menyimpan beberapa rekaman terkait. File dalam basis data biasanya berisi data-data dalmm bentuk tabel yang terkait dengan suatu entitas tertentu atau suatu topik. Misalnya, dalam sistem basis data sebuah perpustakaan, ada file kategori, file buku, file anggota, dan file sirkulasi, masing-masing berisi informasi yang terkait dengan entitas atau topik tersebut. File diorganisir dengan struktur dan format tertentu untuk memudahkan pengelolaan data.



Gambar 1.7. Contoh Basis Data dan Hirarki

3. Baris (*Record*)
   Record atau baris adalah kumpulan field yang membentuk entitas data tunggal atau catatan. Setiap baris dalam tabel merupakan satu rekaman atau record yang berisi informasi lengkap tentang suatu entitas tertentu. Misalnya, dalam tabel data anggota, setiap baris berisi informasi lengkap tentang satu anggota perpusatakaan, termasuk no anggota, nama, alamat, dan kota. Record merupakan unit terorganisir dari data yang diatur dalam tabel dalam basis data.
4. Kolom (*Field*)
   Field atau kolom adalah kumpulan nilai data yang mewakili atribut atau informasi spesifik dalam suatu entitas data. Setiap kolom dalam tabel merepresentasikan jenis data tertentu. Misalnya, dalam tabel data anggota, kolom "No Anggota", "Nama", "Alamat", dan "Kota" adalah contoh field yang mewakili atribut-atribut tertentu dari data anggota perpustakaan. Field mengelompokkan data berdasarkan jenis atau tipe atributnya, dan setiap nilai data dalam field diisi sesuai dengan informasi yang relevan.
5. Nilai Data (*Data Value*)
   Nilai data atau data value adalah unit terkecil dalam hirarki data. Ini merupakan nilai tunggal seperti angka, teks, karakter, atau simbol yang berada dalam kolom atau field. Misalnya, nilai data "Muhammad Fairuzabadi" dalam kolom "Nama" atau nilai data "Sleman" dalam kolom "Kota" adalah contoh nilai data. Nilai data adalah elemen paling dasar dan mentah dari data yang diorganisir dalam tabel-tabel dan berkontribusi dalam membentuk informasi yang lebih besar dan berarti.

1.1.4 Tujuan Basis Data

Tujuan utama dari basis data adalah menyediakan cara yang efisien untuk menyimpan, mengelola, dan mengakses data dengan tujuan mendukung berbagai kegiatan dan aplikasi dalam

suatu organisasi atau sistem informasi. Beberapa tujuan penting dari basis data adalah(Foster dan Godbole, 2016):

1. **Efisiensi Penyimpanan**: Basis data dirancang untuk mengatur data secara terstruktur dan efisien di dalam komputer. Dengan menggunakan metode penyimpanan yang optimal, basis data dapat menghemat ruang penyimpanan dan memastikan data tersedia secara konsisten.

Gambar 1.8. Tujuan Basis Data

2. **Konsistensi Data**: Basis data memastikan bahwa data yang disimpan konsisten dan terintegrasi. Hal ini menghindari duplikasi data yang tidak perlu dan memastikan bahwa semua aplikasi yang menggunakan data tersebut mendapatkan versi yang sama.
3. **Pengamanan Data**: Basis data menyediakan mekanisme keamanan untuk melindungi data dari akses yang tidak sah. Penggunaan otorisasi, autentikasi, dan izin akses membantu menjaga kerahasiaan, integritas, dan ketersediaan data.
4. **Efisiensi Pencarian dan Pengambilan Data**: Dengan menggunakan indeks dan struktur yang terorganisir, basis data memungkinkan pencarian dan pengambilan data secara cepat dan efisien, bahkan dalam volume data yang besar.

5. **Ketersediaan Data**: Basis data dirancang untuk memberikan ketersediaan data yang tinggi, sehingga pengguna dan aplikasi dapat mengakses data yang dibutuhkan kapan pun diperlukan.
6. **Integritas Data**: Basis data menyediakan mekanisme untuk menjaga integritas data, yaitu memastikan bahwa data dalam basis data tetap konsisten, akurat, dan valid sepanjang waktu.
7. **Pembagian Data**: Basis data mendukung pembagian data dalam lingkungan yang terdistribusi, memungkinkan akses data dari lokasi yang berbeda secara transparan.
8. **Pengelolaan Transaksi**: Basis data memungkinkan pengelolaan transaksi untuk memastikan data tetap konsisten dan aman selama proses operasional.
9. **Dukungan Pengambilan Keputusan**: Basis data menyediakan data historis dan alat analisis yang membantu dalam proses pengambilan keputusan yang lebih baik dan informasi yang lebih tepat.

1.2 Arsitektur Basis Data

Arsitektur basis data merujuk pada struktur dan komponen dari suatu sistem basis data. Arsitektur ini merinci cara data disimpan, diatur, dan diakses oleh pengguna dan aplikasi. Arsitektur basis data biasanya mencakup beberapa tingkatan atau lapisan yang berinteraksi satu sama lain untuk menyediakan fungsionalitas lengkap dan efisien dalam mengelola data.

Level arsitektur basis data merupakan konsep penting dalam perancangan dan pengelolaan sistem basis data. Dengan memahami tingkatan-tingkatan ini, kita dapat memahami bagaimana data disimpan, diakses, dan dikelola dengan efisien, serta memastikan integritas dan konsistensi data. Sub bab ini akan menjelaskan secara mendalam tentang masing-masing level arsitektur basis data dan relevansinya dalam mendukung aplikasi dan sistem informasi modern.

Tabel berikut ini menyajikan perbedaan antara ketiga tingkatan dalam arsitektur basis data, yaitu level fisik yang berfokus pada implementasi fisik data, level logik yang berfokus pada representasi struktur data dan cara akses pengguna, dan level konseptual yang berfokus pada pengorganisasian data secara keseluruhan tanpa terikat pada detail fisik(Putikadea *et al.*, 2022). Masing-masing tingkatan memiliki peran penting dalam desain, pengelolaan, dan penggunaan basis data secara efisien dan efektif. Tabel berikut menunjukkan perbedaan antra tiga level tersebut:

Tabel 1.9. Perbedaan, Level Fisik, Level Logik dan Level Konseptual

No.	Aspek Perbedaan	Level Fisik	Level Logik	Level Konseptual
1	Fokus	Implementasi fisik data di perangkat penyimpanan	Representasi struktur data dan cara akses pengguna	Pengorganisasian data secara keseluruhan
2	Tingkat Detail	Detail implementasi fisik	Tinggi, melibatkan definisi tabel, kolom, hubungan, dsb.	Tinggi, namun lebih abstrak dari level logik
3	Representasi	Blok-blok disk, pengaturan memori	Bahasa kueri (SQL), hubungan antar tabel, indeks	Model entitas-hubungan (ER model), ERD
4	Penggunaan	Manajemen penyimpanan fisik	Pengaksesan data dan manipulasi oleh pengguna	Perancangan basis data dan definisi skema

No.	Aspek Perbedaan	Level Fisik	Level Logik	Level Konseptual
5	Fungsionalitas Utama	Pengaturan teknis data fisik	Penggunaan bahasa kueri, hubungan antar tabel	Definisi keseluruhan skema basis data
6	Contoh Komponen	Penyimpanan blok-blok disk, indeks fisik	Bahasa kueri SQL, definisi hubungan antar tabel	ERD (Entity-Relationship Diagram), skema basis data
7	Tingkat Abstraksi	Paling rendah	Tengah	Paling tinggi

1.2.1 Level Fisik

Level fisik adalah tingkatan terbawah dalam arsitektur basis data. Pada tingkatan ini, fokus utama adalah pada cara data fisik disimpan di perangkat penyimpanan, seperti hard disk. Level fisik mencakup aspek teknis seperti cara data diorganisir dalam blok-blok, bagaimana blok-blok tersebut disimpan di disk, dan bagaimana data dikelola secara fisik.

Level fisik dalam arsitektur basis data menggambarkan bagaimana data fisik disimpan di perangkat penyimpanan seperti hard disk. Pada level fisik, data diorganisir dalam blok-blok yang merupakan unit penyimpanan dasar. Berikut adalah contoh-contoh level fisik dalam basis data:

Gambar 1.9. Level Arsitektur Basis Data dan Representasinya

1. **Penyimpanan dalam Blok-Blok Disk**
   Data dipecah menjadi blok-blok disk yang merupakan unit dasar penyimpanan. Setiap blok memiliki ukuran tetap (misalnya 4KB) dan berisi sejumlah record data. Data diurutkan dalam blok-blok ini sehingga memungkinkan akses yang efisien saat membaca atau menulis data ke disk.

2. **Penggunaan Struktur Data Fisik**
   Level fisik mencakup definisi struktur data fisik seperti format record, ukuran field, dan tipe data yang digunakan. Misalnya, sebuah record pada basis data yang menyimpan informasi tentang mahasiswa mungkin terdiri dari field nama, usia, dan alamat, dengan ukuran dan tipe data tertentu untuk masing-masing field.
3. **Penggunaan Teknologi Penyimpanan yang Tepat**
   Level fisik juga mencakup pilihan teknologi penyimpanan yang sesuai, seperti menggunakan SSD (Solid State Drive) untuk meningkatkan kecepatan akses data, atau RAID (Redundant Array of Independent Disks) untuk meningkatkan keandalan dan ketersediaan data.
4. **Indeks Fisik**
   Untuk meningkatkan efisiensi pencarian data, level fisik mencakup implementasi indeks fisik. Indeks digunakan untuk mengurutkan atau mengorganisir data tertentu dalam blok-blok disk, sehingga pencarian data menjadi lebih cepat.
5. **Manajemen Ruang Penyimpanan**
   Level fisik juga melibatkan manajemen ruang penyimpanan, termasuk bagaimana data di-alokasikan dan di-dealokasikan pada disk. Sistem basis data harus mengelola ruang kosong dan penuh pada disk untuk memastikan ruang disk digunakan secara efisien.

1.2.2 Level Logika

Level logika dalam arsitektur basis data berkaitan dengan representasi struktur data yang terstruktur dan cara data diakses oleh pengguna dan aplikasi. Pada tingkatan ini, fokus utama adalah pada bagaimana data diorganisir dalam basis data dan bagaimana pengguna berinteraksi dengan data tersebut. Berikut adalah beberapa contoh dari level logika dalam basis data:

1. **Definisi Skema Database**
   Pada tingkatan logika, pengembang mendefinisikan skema database, yang mencakup tabel, kolom, hubungan antar tabel, dan konstrain data. Misalnya, sebuah skema database dapat mencakup tabel "Mahasiswa" dengan kolom "Nama," "Usia," dan "Jurusan," serta tabel "Mata Kuliah" dengan kolom "Nama Mata Kuliah" dan "Dosen Pengajar."
2. **Penggunaan Bahasa Kueri**
   Pengguna dapat berinteraksi dengan data melalui bahasa kueri seperti SQL (*Structured Query Language*). Bahasa ini memungkinkan pengguna untuk mengambil, menyisipkan, memperbarui, dan menghapus data dalam basis data. Contohnya, pengguna dapat menulis kueri "SELECT * FROM Mahasiswa WHERE Jurusan='Informatika'" untuk mengambil data mahasiswa yang memiliki jurusan Informatika.
3. **Hubungan Antara Tabel**
   Level logika mencakup hubungan antara tabel-tabel dalam basis data, seperti hubungan *one-to-many* atau *many-to-many*. Misalnya, tabel "Mahasiswa" dan tabel "Mata Kuliah" dapat memiliki hubungan *many-to-many* melalui tabel perantara "Daftar Mata Kuliah" yang menghubungkan mahasiswa dengan mata kuliah yang diambil.
4. **Konstrain Data**
   Pengembang dapat menetapkan konstrain data untuk memastikan integritas data. Misalnya, pengembang dapat menetapkan konstrain kunci utama pada kolom "ID Mahasiswa" untuk memastikan bahwa setiap mahasiswa memiliki ID yang unik.
5. **Views**
   Level logika juga mencakup penggunaan views, yang merupakan pandangan alternatif atau subset dari data dalam basis data. Views memungkinkan pengguna untuk melihat data yang spesifik tanpa harus mengakses seluruh tabel.

1.2.3 Level Konseptual

Level konseptual dalam arsitektur basis data adalah tingkatan teratas yang menggambarkan representasi keseluruhan basis data tanpa memperhatikan detail implementasi fisiknya. Pada tingkatan ini, fokus utama adalah pada pengorganisasian data secara keseluruhan, definisi keseluruhan skema database, dan hubungan antar tabel secara menyeluruh. Berikut adalah beberapa contoh dari level konseptual dalam basis data:

1. **Model Entitas-Hubungan**
   Level konseptual biasanya dijelaskan menggunakan model entitas-hubungan (ER model) yang menunjukkan entitas-entitas utama dan hubungan antara entitas-entitas tersebut. Misalnya, basis data untuk perpustakaan mungkin memiliki entitas "Buku," "Anggota," dan "Peminjaman," serta hubungan antara entitas tersebut seperti "Meminjam" dan "Dimiliki oleh."
2. **Definisi Skema Basis Data**
   Level konseptual mencakup definisi skema basis data secara menyeluruh, termasuk semua tabel, atribut-atributnya, serta keterkaitan dan hubungannya. Misalnya, skema konseptual dari sistem informasi karyawan dapat mencakup tabel "Karyawan" dengan atribut "Nama," "Usia," dan "Posisi," serta tabel "Departemen" dengan atribut "Nama Departemen" dan "Lokasi."
3. ***Entity-Relationship Diagram* (ERD)**
   ERD adalah bentuk visual dari model entitas-hubungan yang digunakan untuk menyajikan skema konseptual. Diagram ini menunjukkan entitas, atribut, hubungan antara entitas, dan kardinalitas hubungan.
4. **Data Dictionary atau Katalog Basis Data**
   Level konseptual mencakup data dictionary atau katalog basis data yang berisi metadata tentang seluruh skema basis data. Data dictionary mencatat semua informasi tentang tabel, atribut, indeks, kunci, dan konstrain data lainnya.

5. **Definisi Integritas Referensial**
   Level konseptual mencakup definisi integritas referensial yang mengatur hubungan antara tabel, sehingga menjaga konsistensi data dalam basis data. Misalnya, jika sebuah tabel memiliki kunci asing yang mengacu pada kunci utama tabel lain, definisi integritas referensial memastikan bahwa referensi tersebut selalu valid.

1.3 Sistem Basis Data

Sistem basis data (*database system*) merupakan fondasi penting dalam dunia teknologi informasi, menyediakan cara yang efisien dan terstruktur dalam menyimpan, mengelola, dan mengakses data. Pada dasarnya, sistem basis data adalah kumpulan data terstruktur yang diatur dengan bantuan perangkat lunak pengelola basis data (DBMS). DBMS memainkan peran sentral dalam memfasilitasi interaksi antara pengguna dan data, menyediakan antarmuka untuk melakukan operasi seperti pencarian, penyisipan, pembaruan, dan penghapusan data. Selain itu, sistem basis data juga melibatkan komponen perangkat keras (seperti server), mekanisme pengamanan data, pemulihan data dalam situasi darurat, dan optimisasi kinerja untuk memastikan ketersediaan dan keandalan data. Dalam paragraf berikutnya, kami akan menjelaskan lebih rinci tentang komponen-komponen penting dalam sistem basis data yang berperan dalam mendukung pengelolaan data yang efektif dan terpercaya.

1.3.1 Definisi Sistem Basis Data

Sistem basis data adalah kumpulan terstruktur dari data yang saling terkait, disimpan dalam perangkat penyimpanan komputer, dan diakses serta dikelola menggunakan perangkat lunak pengelola basis data (DBMS). Sistem basis data dirancang untuk menyimpan, mengatur, mengelola, dan memanipulasi data dengan efisien, serta menyediakan cara bagi pengguna dan aplikasi

untuk mengakses dan memanipulasi data tersebut(Widodo dan Kurnianingtyas, 2017).

Sistem basis data menyediakan antarmuka untuk pengguna dan aplikasi berinteraksi dengan data dalam basis data. Pengguna dapat melakukan berbagai operasi seperti mencari data, menambahkan data baru, mengubah data yang sudah ada, menghapus data, dan melihat laporan berdasarkan data yang tersimpan.

Sistem basis data juga menyediakan mekanisme untuk menjaga integritas dan konsistensi data, menghindari konflik antaroperasi, dan mengatur akses data agar sesuai dengan hak dan izin yang telah ditentukan.

1.3.2 Perbedaan Basis Data dan Sistem Basis Data

Perbedaan antara basis data (*database*) dan sistem basis data (*database system*) terletak pada konsep dan cakupan penggunaannya. Dengan kata lain, basis data adalah kumpulan data terstruktur yang disimpan dalam media penyimpanan, sedangkan sistem basis data adalah kumpulan dari basis data itu sendiri beserta perangkat lunak dan perangkat keras yang bekerja bersama-sama untuk mengelola data dengan cara yang efisien dan terstruktur. Sistem basis data mencakup lebih banyak fitur dan kemampuan daripada sekadar basis data, karena melibatkan komponen perangkat lunak pengelola basis data yang memungkinkan pengolahan data secara lebih canggih dan kompleks.

Basis data adalah kumpulan data terstruktur yang disimpan dalam media penyimpanan, sedangkan sistem basis data mencakup lebih banyak elemen termasuk basis data itu sendiri dan perangkat lunak pengelola basis data yang memungkinkan pengolahan data yang lebih canggih dan kompleks. Tabel berikut menyajikan perbedaan antara Basis Data (*Database*) dan Sistem Basis Data (*Database System*).

Tabel 1.10. Basis Data Versus Sistem Basis Data

No.	Aspek Perbedaan	Basis Data (*Database*)	Sistem Basis Data (*Database System*)
1	Definisi	Kumpulan data terstruktur dalam media penyimpanan	Kombinasi basis data dan perangkat lunak pengelola basis data
2	Cakupan	Hanya mencakup struktur data	Meliputi basis data, perangkat lunak DBMS, dan perangkat keras
3	Perangkat Lunak Pengelola	Tidak termasuk perangkat lunak pengelola basis data	Memiliki perangkat lunak pengelola basis data (DBMS)
4	Mekanisme Tambahan	Tidak menyediakan mekanisme tambahan seperti pengamanan data, pemulihan data, dll.	Menyediakan mekanisme pengamanan, pemulihan, dll.
5	Sifat	Biasanya bersifat statis, hanya menyimpan data	Bersifat lebih dinamis, mendukung manajemen data yang lebih efisien

1.3.3 Komponen Sistem Basis Data

Sistem basis data adalah kesatuan dari berbagai komponen penting yang bekerja bersama untuk mengelola, menyimpan, dan mengakses data secara efisien dan terstruktur. Sistem basis data merupakan fondasi penting dalam manajemen data modern. Terdiri dari berbagai komponen yang saling terkait, sistem basis data menyediakan cara yang efisien dan andal dalam mengelola data. Berikut adalah komponen-komponen utama dalam sistem basis data(Schaller dan Elmagarmid, 1993; Ramakrishnan dan Gehrke, 2003; Tantriawan *et al.*, 2023):

1. **Pengguna (*Users*)**
   Pengguna adalah individu atau entitas yang berinteraksi dengan sistem basis data. Level pengguna dalam basis data dapat diklasifikasikan ke dalam empat kategori berdasarkan tingkat akses dan kemampuan mereka dalam mengelola data. Berikut adalah level pengguna basis data:

Gambar 1.10. Komponen Sistem Basis Data

   a. **Pengguna Akhir (*End Users*)**
      Pengguna akhir adalah pengguna akhir dari sistem basis data yang memiliki akses terbatas untuk hanya membaca, menambah, mengubah, atau menghapus data yang sudah ada. Mereka tidak memiliki akses langsung ke struktur basis data atau kontrol administratif. Pengguna akhir umumnya

berinteraksi dengan basis data melalui antarmuka aplikasi yang telah disediakan, seperti aplikasi desktop, aplikasi web, atau aplikasi mobile. Contoh pengguna akhir termasuk karyawan yang mengisi formulir data, siswa yang mengakses informasi akademik, atau pelanggan yang melakukan pembelian online.

b. Pengguna Aplikasi (*Application Users*)

Pengguna aplikasi adalah pengguna yang memiliki pengetahuan dan keterampilan teknis dalam menggunakan aplikasi yang dikembangkan untuk mengakses dan memanipulasi basis data. Mereka dapat melakukan tugas yang lebih kompleks dalam aplikasi, seperti menghasilkan laporan, melakukan analisis data, atau mengelola proses bisnis. Pengguna aplikasi ini memiliki tingkat akses yang lebih tinggi daripada pengguna akhir, tetapi mereka tidak memiliki hak administratif atas struktur basis data.

c. Pengguna Database Administrator (DBA):

Pengguna database administrator adalah pihak yang bertanggung jawab atas manajemen, pemeliharaan, dan administrasi basis data secara keseluruhan. Mereka memiliki hak akses penuh untuk mengelola dan mengatur struktur basis data, melakukan pemulihan data, membuat cadangan, mengoptimalkan kinerja, dan mengamankan basis data dari potensi ancaman. DBA adalah peran yang sangat teknis dan membutuhkan pengetahuan mendalam tentang teknologi basis data.

d. Pengembang Basis Data (*Database Developer*)

Pengembang basis data adalah individu yang memiliki keahlian dalam merancang, mengembangkan, dan mengimplementasikan struktur basis data serta aplikasi yang berinteraksi dengan basis data. Mereka berfokus pada desain tabel, indeks, tampilan, dan prosedur yang mendukung fungsionalitas aplikasi. Pengembang basis data

juga bertanggung jawab untuk menulis dan mengoptimalkan kueri SQL (*Structured Query Language*) untuk mengambil, memperbarui, atau menghapus data dari basis data.

2. **Perangkat Keras (*Hardware*)**
   Komponen perangkat keras mencakup server komputer, perangkat penyimpanan data seperti hard disk dan SSD, serta perangkat jaringan yang memungkinkan koneksi antara server dan klien. Perangkat keras yang handal dan berkualitas tinggi penting untuk memastikan ketersediaan dan performa sistem basis data.
3. **Perangkat Lunak (*Software*)**
   Perangkat lunak utama dalam sistem basis data adalah perangkat lunak pengelola basis data (DBMS). DBMS menyediakan antarmuka untuk berinteraksi dengan data dan melakukan operasi seperti pencarian, penyisipan, pembaruan, dan penghapusan data. Contoh DBMS termasuk MySQL, Oracle, Microsoft SQL Server, dan PostgreSQL.
4. **Jaringan (*Network*)**
   Jaringan adalah infrastruktur yang menghubungkan server basis data dengan klien atau aplikasi yang membutuhkan akses ke data. Jaringan memungkinkan akses data dari lokasi yang berbeda secara transparan dan memastikan konektivitas yang andal antara pengguna dan sistem basis data(Muttaqin, Karim, *et al.*, 2023).
5. **Pengamanan dan Pemulihan (*Security and Recovery*)**
   Komponen pengamanan melibatkan langkah-langkah untuk melindungi data dari akses tidak sah dan mengatur izin akses yang tepat. Selain itu, mekanisme pemulihan data dalam situasi darurat atau kegagalan sistem juga penting untuk memastikan kontinuitas dan ketersediaan data.

6. **Manajemen dan Optimisasi (*Management and Optimization*)**
   Manajemen basis data mencakup tugas seperti pemantauan kinerja, perencanaan kapasitas, penjadwalan pemeliharaan, dan penanganan masalah yang terkait dengan basis data. Optimisasi bertujuan untuk meningkatkan performa basis data dengan penyesuaian indeks, struktur data, dan konfigurasi sistem.

DAFTAR PUSTAKA

Elmasri, R. dan Navathe, S.B. 2015. *Fundamentals of Database Systems*. Seventh Ed. Pearson Education. doi:10.1201/9781003304029-2.

Foster, E.C. dan Godbole, S. 2016. *Database systems: A pragmatic approach*, *Database Systems: A Pragmatic Approach*. doi:10.1007/978-1-4842-1191-5.

Kadir, A. 2014. *Pengenalan Sistem Informasi*. Revisi. Yogyakarta: ANDI.

Muttaqin, Karim, A., *et al.* 2023. *Jaringan Komputer dan Internet*. Medan, Indonesia: Yayasan Kita Menulis.

Muttaqin, Sondakh, D.E., *et al.* 2023. *Pengantar Teknologi Digital*. Medan: Yayasan Kita Menulis.

Putikadea, I. *et al.* 2022. *Manajemen Basis Data*. CV. Bintang Semesta Media.

Ramakrishnan, R. dan Gehrke, J. 2003. "Database Management Systems solution," *Database Management Systems solutions manual Third Edition*, 8(4), hal. 65–68. Tersedia pada: http://portal.acm.org.oca.ucsc.edu/citation.cfm?id=553046.

Schaller, T. dan Elmagarmid, A.K. 1993. "The Integration of Database Systems," 4(5), hal. 415–423.

Tantriawan, H. *et al.* 2023. *Basis Data: Teori dan Praktik*. Medan: Yayasan Kita Menulis.

Widodo, A.W. dan Kurnianingtyas, D. 2017. *Sistem Basis Data*. Malang, Indonesia: Universitas Brawijaya Press.

BAB 2
MODEL – MODEL BASIS DATA

Oleh Angga Aditya Permana

2.1 Pendahuluan

Gambar 2.1. Model Basis Data
Sumber : https://www.nesabamedia.com/

Model-model basis data adalah pondasi yang kritis dalam dunia teknologi informasi modern. Mereka adalah struktur konseptual yang memungkinkan organisasi untuk mengelola, mengorganisir, dan mengakses data dengan cara yang efisien. Dalam lingkup ini, model basis data mengacu pada kerangka kerja yang digunakan untuk menggambarkan bagaimana data disimpan, diatur, dan dihubungkan dalam sistem basis data. Pemahaman yang mendalam tentang berbagai model basis data sangat penting,

karena ini akan mempengaruhi cara data dikelola, diambil, dan dimanfaatkan dalam berbagai aplikasi. (Warden dan Warden 1984; Sharma 2022)

Salah satu model basis data yang paling dikenal dan sering digunakan adalah model relasional. Model ini menggambarkan data dalam bentuk tabel dua dimensi yang terhubung oleh relasi atau kunci. Dengan SQL (*Structured Query Language*) sebagai bahasa standar, model relasional telah menjadi tulang punggung sistem basis data bisnis dan aplikasi modern yang luas. Ini memberikan kemudahan dalam mengakses data, menjadikan basis data relasional pilihan utama untuk banyak aplikasi.

Selain model relasional, terdapat berbagai model basis data lainnya, seperti model hierarki, jaringan, objek, dokumen, kolom, dan grafik. Setiap model memiliki karakteristik unik yang cocok untuk jenis data tertentu dan skenario penggunaan khusus. Model basis data objek, misalnya, memungkinkan penyimpanan objek-objek kompleks dan perilaku yang dapat ditentukan pengguna dalam basis data. Model basis data grafik, di sisi lain, sangat efektif dalam menggambarkan dan menjelajahi hubungan kompleks antara data dalam lingkungan seperti jejaring sosial.

Dalam pengembangan dan pemeliharaan sistem basis data, pemilihan model yang tepat sangat penting. Model basis data yang salah dapat mengakibatkan kesulitan dalam mengelola data atau bahkan menghambat efisiensi operasi bisnis. Oleh karena itu, dalam panduan ini, kami akan menjelajahi berbagai model basis data secara rinci, menggali keunggulan dan kelemahan masing-masing, serta memberikan panduan tentang kapan dan bagaimana menggunakan model-model ini secara efektif dalam berbagai konteks.

Dengan pemahaman yang kuat tentang model-model basis data ini, Anda akan memiliki landasan yang kokoh untuk merancang, mengimplementasikan, dan mengelola sistem basis data yang sesuai dengan kebutuhan dan tujuan bisnis Anda. Mari

kita mulai dengan menjelajahi dunia yang luas dan menarik dari model-model basis data.

Ada beberapa model basis data yang digunakan untuk mengorganisir dan mengelola data dalam sebuah sistem basis data. Berikut ini adalah beberapa model basis data yang umum digunakan:

1. Model Basis Data Hierarki: Model ini mengorganisir data dalam struktur hierarki seperti pohon. Data disusun dalam bentuk hirarki dengan satu entitas (seperti orang atau barang) di atas yang lain. Model ini digunakan dalam beberapa sistem berbasis file awal dan terutama cocok untuk data yang memiliki hubungan satu ke banyak.
2. Model Basis Data Jaringan: Model ini mengatasi beberapa keterbatasan dari model hierarki dengan memungkinkan entitas memiliki beberapa "orang tua." Dalam model ini, data diorganisir dalam bentuk jaringan yang terhubung, mirip dengan struktur grafik. Model ini digunakan dalam beberapa sistem basis data kuno.
3. Model Basis Data Relasional: Model ini adalah salah satu yang paling umum digunakan saat ini. Data disimpan dalam tabel dua dimensi dengan relasi antara tabel. Ini memungkinkan fleksibilitas dalam mengakses data dan menyediakan bahasa SQL yang kuat untuk mengambil dan memanipulasi data. Basis data relasional sangat cocok untuk sebagian besar aplikasi bisnis.
4. Model Basis Data Objek: Model ini menggabungkan fitur dari basis data relasional dengan objek-objek dari pemrograman berorientasi objek. Ini memungkinkan penyimpanan data kompleks seperti objek, gambar, dan suara, serta perilaku yang dapat ditentukan pengguna dalam bentuk metode.
5. Model Basis Data Dokumen: Model ini digunakan untuk menyimpan data semi-struktur seperti dokumen, teks, dan data yang berubah-ubah. Basis data dokumen berfokus pada

penyimpanan dan pengelolaan koleksi dokumen dengan fitur pencarian yang kuat.

6. Model Basis Data Kolom: Model ini berfokus pada penyimpanan data dalam bentuk kolom daripada baris. Ini dapat efisien untuk aplikasi yang memerlukan pembacaan atau penulisan hanya pada beberapa kolom data, seperti analisis data dan data warehousing.
7. Model Basis Data Grafik: Model ini cocok untuk data yang memiliki struktur yang sangat rumit dan banyak hubungan yang kompleks. Data diorganisir dalam bentuk grafik dengan node dan edge yang mewakili entitas dan hubungan antara mereka. Model ini sering digunakan dalam aplikasi seperti jejaring sosial dan analisis jaringan.

Setiap model basis data memiliki kelebihan dan kelemahan sendiri serta situasi yang cocok untuk penggunaannya. Pemilihan model basis data yang tepat tergantung pada jenis data yang akan disimpan, kebutuhan aplikasi, dan kompleksitas hubungan antara data.

2.2 Penting dalam Menentukan Model Basis Data

Gambar 2.2. Icon Basis Data
Sumber : https://www.pngegg.com/

Menentukan penggunaan model basis data yang tepat adalah keputusan kunci dalam perancangan sistem informasi. Berikut adalah beberapa hal yang perlu diperhatikan saat memilih model basis data: (Warden dan Warden 1984; Kurniati *et al.* 2015; Sharma 2022)

1. **Jenis Data**: Pertimbangkan jenis data yang akan disimpan dalam basis data Anda. Apakah data tersebut terstruktur (seperti data tabel dalam basis data relasional), semi-struktur (seperti dokumen atau XML), atau tak terstruktur (seperti gambar atau suara)? Model basis data yang tepat akan bergantung pada karakteristik data ini.
2. **Kompleksitas Hubungan**: Pahami hubungan antara data Anda. Apakah data memiliki hubungan sederhana satu-ke-satu atau satu-ke-banyak, ataukah ada hubungan yang lebih kompleks dan berlapis? Model basis data relasional sering cocok untuk hubungan sederhana, sementara model basis data grafik mungkin lebih cocok untuk hubungan kompleks.
3. **Skalabilitas**: Pertimbangkan ukuran basis data Anda dan pertumbuhannya di masa depan. Model basis data harus dapat menangani pertumbuhan data tanpa mengorbankan kinerja. Model kolom atau basis data terdistribusi mungkin lebih cocok untuk skenario yang memerlukan skalabilitas tinggi.
4. **Kebutuhan Kinerja**: Perhatikan performa aplikasi Anda. Beberapa model basis data dapat memberikan performa yang lebih baik untuk jenis operasi tertentu. Sebagai contoh, basis data kolom sering lebih cepat dalam operasi analisis data dibandingkan dengan basis data relasional.
5. **Kemudahan Pengembangan dan Pemeliharaan**: Pertimbangkan kemudahan dalam mengembangkan dan memelihara aplikasi yang menggunakan model basis data tertentu. Beberapa model mungkin memerlukan lebih banyak usaha dalam pengembangan dan pemeliharaan, sementara yang lain mungkin lebih sederhana.

6. **Bahasa Pertanyaan dan Akses Data**: Apakah Anda memerlukan bahasa khusus untuk mengakses dan memanipulasi data? Basis data relasional menggunakan SQL secara umum, sementara basis data NoSQL (seperti basis data dokumen) mungkin menggunakan bahasa pertanyaan yang berbeda.
7. **Kebutuhan Keamanan**: Pertimbangkan tingkat keamanan yang diperlukan untuk data Anda. Beberapa model basis data memiliki fitur keamanan bawaan yang lebih kuat, termasuk autentikasi, otorisasi, dan enkripsi data.
8. **Fleksibilitas dan Perubahan**: Pikirkan tentang seberapa sering struktur data Anda berubah. Model basis data yang lebih fleksibel mungkin lebih sesuai jika Anda berurusan dengan perubahan struktur data yang sering.
9. **Ketersediaan Teknologi**: Pertimbangkan ketersediaan teknologi dan dukungan yang ada untuk model basis data yang Anda pertimbangkan. Pastikan ada infrastruktur dan alat yang memadai untuk mendukung penggunaan model tersebut.
10. **Biaya**: Akhirnya, pertimbangkan biaya pengembangan, implementasi, dan pemeliharaan sistem berdasarkan model basis data yang Anda pilih. Beberapa model basis data mungkin lebih mahal dalam hal infrastruktur atau lisensi perangkat lunak.

Mempertimbangkan semua faktor ini secara cermat akan membantu Anda menentukan model basis data yang paling sesuai untuk proyek atau aplikasi tertentu. Terkadang, kombinasi model basis data atau pendekatan hybrid juga dapat menjadi solusi yang tepat untuk mengatasi berbagai kebutuhan data yang beragam.

2.3 Tips Memilih Model Database yang Sesuai

Gambar 2.3. Tips DBA
Sumber : https://www.pngegg.com/

Sebagai seorang *Database Administrator* (DBA), memilih model database yang tepat adalah langkah kunci dalam merancang dan mengelola sistem basis data. Berikut beberapa tips yang dapat membantu Anda dalam proses pemilihan model database yang sesuai: (Warden dan Warden 1984; Kurniati *et al.* 2015; Sharma 2022)

1. **Pahami Kebutuhan Bisnis**: Langkah awal yang krusial adalah memahami sepenuhnya kebutuhan bisnis dan tujuan penggunaan basis data. Diskusikan dengan stakeholder untuk menentukan jenis data yang akan disimpan, bagaimana data tersebut akan digunakan, dan berapa besar volume data yang diharapkan.
2. **Evaluasi Karakteristik Data**: Teliti dengan cermat karakteristik data yang akan Anda simpan. Pertimbangkan jenis data (struktur, semi-struktur, atau tak terstruktur),

kompleksitas hubungan antara data, dan apakah data memiliki atribut yang berubah-ubah atau tidak.

3. **Skala dan Performa**: Pertimbangkan besarnya basis data dan perkiraan pertumbuhan data di masa depan. Pastikan model database yang Anda pilih mampu menangani skala tersebut dengan baik tanpa mengorbankan performa aplikasi.
4. **Pertimbangkan Keamanan**: Keamanan data adalah aspek yang sangat penting. Pastikan model database yang Anda pilih memiliki fitur keamanan yang memadai, termasuk autentikasi, otorisasi, dan enkripsi data.
5. **Bahasa Pertanyaan dan Akses Data**: Pertimbangkan jenis bahasa pertanyaan yang akan digunakan untuk mengakses dan memanipulasi data. Misalnya, basis data relasional menggunakan SQL, sementara basis data NoSQL mungkin memiliki bahasa pertanyaan yang berbeda.
6. **Fleksibilitas dan Kemudahan Pengembangan**: Pahami tingkat fleksibilitas yang Anda butuhkan. Model basis data yang lebih fleksibel mungkin lebih cocok jika struktur data berubah dengan sering. Selain itu, perhatikan kemudahan dalam mengembangkan aplikasi di sekitar model basis data tersebut.
7. **Ketersediaan Teknologi**: Pastikan infrastruktur dan alat yang dibutuhkan untuk model basis data tersebut tersedia dan dapat diakses. Juga, pertimbangkan apakah ada sumber daya atau komunitas yang dapat memberikan dukungan.
8. **Biaya**: Evaluasi biaya pengembangan, implementasi, dan pemeliharaan sistem berdasarkan model basis data yang Anda pilih. Beberapa model basis data mungkin memiliki biaya yang lebih tinggi dalam hal infrastruktur, lisensi perangkat lunak, atau kebutuhan sumber daya manusia.
9. **Kaji Pengalaman Tim**: Pertimbangkan pengetahuan dan pengalaman tim DBA dalam mengelola model basis data tertentu. Jika tim Anda sudah berpengalaman dengan model

tertentu, ini dapat memudahkan implementasi dan pemeliharaan.

10. **Pertimbangkan Hybrid atau Multi-Model**: Terkadang, memilih satu model basis data tidak cukup. Pertimbangkan apakah ada kebutuhan untuk pendekatan hybrid atau multi-model, di mana Anda menggunakan beberapa model basis data dalam satu sistem untuk menangani berbagai jenis data.
11. **Uji dan Evaluasi**: Sebelum mengambil keputusan akhir, lakukan uji coba dan evaluasi model basis data yang dipilih dalam lingkungan yang sesuai dengan kebutuhan Anda. Ini dapat membantu Anda memahami bagaimana model tersebut akan berkinerja dalam situasi nyata.
12. **Pemantauan dan Pemeliharaan Berkelanjutan**: Ingatlah bahwa pemilihan model basis data tidak hanya tentang pemilihan awal. DBA perlu terus memantau dan memelihara basis data sesuai dengan perkembangan kebutuhan bisnis dan perubahan data.

Dengan mempertimbangkan faktor-faktor ini secara cermat, Anda dapat membuat keputusan yang lebih tepat dalam memilih model basis data yang sesuai untuk proyek atau sistem Anda, yang akan berdampak positif pada kinerja dan keamanan sistem basis data Anda.

Mari kita lihat sebuah contoh nyata yang mengilustrasikan pemilihan model basis data. Bayangkan Anda adalah seorang DBA di sebuah perusahaan e-commerce yang sedang merencanakan untuk mengembangkan sistem basis data baru untuk mengelola informasi produk dan pesanan pelanggan.

Kebutuhan Bisnis: Bisnis ini memiliki kebutuhan untuk menyimpan informasi produk seperti nama, harga, stok, deskripsi, dan kategori produk. Mereka juga perlu melacak pesanan pelanggan, termasuk data pelanggan, produk yang dibeli, tanggal pesanan, dan status pengiriman.

Karakteristik Data:

1. Produk: Data terstruktur, dengan atribut seperti nama (teks), harga (numerik), stok (numerik), deskripsi (teks), dan kategori produk (teks).
2. Pesanan: Data terstruktur, dengan atribut seperti nomor pesanan (numerik), pelanggan (teks), produk yang dibeli (teks atau ID produk), tanggal pesanan (tanggal), dan status pengiriman (teks).

Skala dan Performa: Bisnis ini telah tumbuh pesat dan memproses ribuan pesanan per hari. Mereka berencana untuk terus berkembang.
Keamanan: Data pelanggan sangat sensitif, jadi keamanan harus menjadi prioritas.
Bahasa Pertanyaan dan Akses Data: Tim pengembang sudah sangat terampil dalam menggunakan SQL.
Fleksibilitas dan Kemudahan Pengembangan: Bisnis ini ingin dapat menambahkan kategori produk baru dengan mudah dan merespons perubahan dalam struktur data dengan cepat.
Ketersediaan Teknologi: Tim IT memiliki pengalaman dalam mengelola sistem basis data MySQL.
Biaya: Ada anggaran terbatas untuk infrastruktur dan lisensi perangkat lunak.
Pertimbangan Pengalaman Tim: Tim memiliki pengalaman yang kuat dalam penggunaan MySQL dan model basis data relasional.
Pemilihan Model Basis Data: Berdasarkan informasi di atas, dalam contoh ini, pemilihan model basis data relasional seperti MySQL atau PostgreSQL adalah pilihan yang masuk akal. Berikut ilustrasinya:

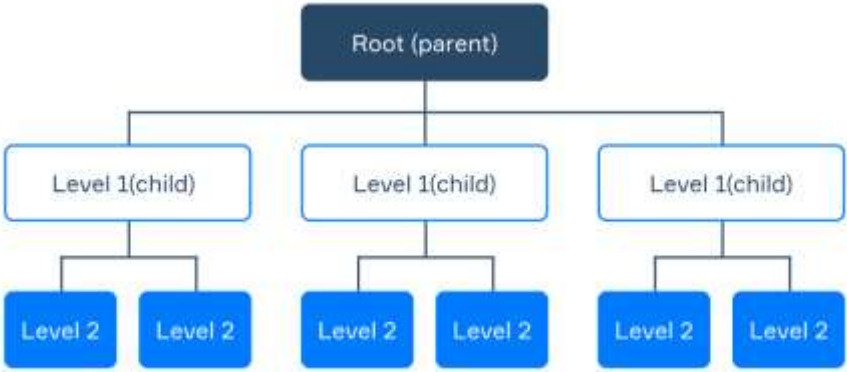
1. Produk dapat disimpan dalam tabel dengan kolom-kolom yang sesuai dengan atribut produk seperti nama, harga, stok, dan sebagainya.

2. Pesanan dapat disimpan dalam tabel terpisah dengan referensi ke produk yang dibeli dan pelanggan yang membuat pesanan.
3. SQL dapat digunakan untuk mengambil data produk dan pesanan dengan berbagai permintaan seperti mencari produk berdasarkan kategori atau melihat pesanan pelanggan tertentu.

Model basis data relasional memungkinkan untuk mengorganisir data dengan struktur yang baik, menjaga konsistensi data, dan memberikan fleksibilitas dalam mengambil data dengan bahasa SQL. Karena tim memiliki pengalaman dengan MySQL, ini juga memudahkan dalam pengembangan dan pemeliharaan.

Namun, penting untuk diingat bahwa pemilihan model basis data akan sangat bergantung pada kasus-kasus khusus dan kebutuhan bisnis yang berbeda. Dalam kasus lain dengan karakteristik data yang berbeda, seperti data semi-struktur atau tak terstruktur, model basis data NoSQL atau model basis data dokumen mungkin lebih cocok.

2.4 Model Basis Data Hierarki



Gambar 2.4. Basis Data Hierarki
Sumber : https://hyperskill.org/

Model Basis Data Hierarki adalah salah satu model basis data yang pertama kali diperkenalkan dalam pengembangan sistem basis data. Model ini mengorganisir data dalam struktur berbentuk hirarki atau pohon, di mana setiap entitas atau elemen data memiliki satu entitas "induk" di atasnya, kecuali akar dari pohon tersebut. Setiap entitas dapat memiliki satu atau lebih entitas "anak" yang berada di bawahnya dalam hierarki. Contohnya bisa diambil dalam konteks organisasi, di mana seorang karyawan memiliki atasan yang menjadi entitas "induk," sementara karyawan tersebut dapat memiliki bawahan yang menjadi entitas "anak." (Warden dan Warden 1984; Kurniati *et al.* 2015; Sharma 2022)
Dalam model basis data hierarki, entitas-entitas ini direpresentasikan sebagai node-node dalam pohon. Setiap node memiliki atribut-atributnya sendiri, dan hubungan antara node-node ini didefinisikan oleh struktur hirarkisnya. Salah satu contoh implementasi model ini adalah IMS (*Information Management System*) dari IBM yang populer pada tahun 1960-an dan 1970-an.

Salah satu keunggulan dari model basis data hierarki adalah kemudahan dalam menggambarkan hubungan hierarkis yang kuat antara entitas. Namun, model ini memiliki beberapa kelemahan, seperti kurangnya fleksibilitas dalam menggambarkan hubungan yang kompleks dan tidak sesuai untuk data dengan struktur yang dinamis. Oleh karena itu, dengan munculnya model basis data relasional yang lebih fleksibel, penggunaan model hierarki telah berkurang secara signifikan dalam pengembangan sistem basis data modern.

Contoh Basis Data Hierarki

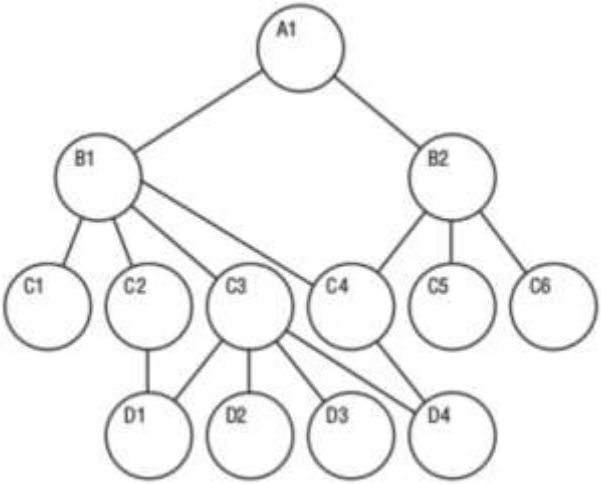
Ilustrasi berikut akan membantu Anda memahami bagaimana model basis data hierarki dapat digunakan. Mari kita anggap kita memiliki sebuah organisasi fiksi yang ingin mengelola struktur organisasi mereka dengan menggunakan model basis data hierarki.

Entitas Utama: Organisasi ini terdiri dari beberapa entitas utama, yaitu "Divisi," "Departemen," "Manajer," dan "Karyawan."
Hierarki Organisasi: Organisasi ini memiliki struktur hierarki yang jelas. Setiap "Divisi" memiliki satu atau lebih "Departemen" di bawahnya. Setiap "Departemen" dipimpin oleh satu "Manajer," dan setiap "Manajer" mengawasi beberapa "Karyawan."
Atribut Entitas: Setiap entitas memiliki atribut-atributnya sendiri. Misalnya, "Divisi" dapat memiliki atribut seperti "Nama Divisi" dan "Lokasi." "Karyawan" dapat memiliki atribut seperti "Nama," "Nomor Karyawan," dan "Gaji."
Ilustrasi Model Hierarki:

Dalam ilustrasi ini, "Divisi" adalah entitas utama dalam model basis data hierarki. Setiap "Divisi" memiliki atribut seperti "Nama Divisi" dan "Lokasi." Di bawah setiap "Divisi," terdapat "Departemen" yang memiliki "Manajer" yang memimpinnya. Departemen tersebut kemudian memiliki "Karyawan" yang bekerja di dalamnya.

Model basis data hierarki sangat berguna dalam menggambarkan struktur organisasi yang memiliki hierarki yang jelas. Dalam kasus ini, Anda dapat dengan mudah melihat hubungan antara divisi, departemen, manajer, dan karyawan dalam bentuk pohon hierarki. Namun, model ini tidak cocok untuk menggambarkan hubungan yang lebih kompleks atau dinamis yang mungkin ada dalam basis data modern. Itulah mengapa model basis data hierarki telah digantikan oleh model-model basis data yang lebih fleksibel seperti model relasional dan NoSQL dalam banyak kasus penggunaan saat ini.

2.5 Model Basis Data Jaringan



Gambar 2.5. Basis Data Jaringan
Sumber : https://mariadb.com/

Model Basis Data Jaringan adalah salah satu model basis data yang digunakan untuk mengorganisir dan menghubungkan data dalam suatu sistem basis data. Model ini dikembangkan sebagai evolusi dari model hierarki dan memperkenalkan konsep struktur data yang lebih fleksibel. Dalam model basis data jaringan, data diorganisir dalam bentuk grafik atau jaringan, di mana entitas (node) dapat memiliki lebih dari satu entitas "induk" atau "orang tua." (Warden dan Warden 1984; Kurniati *et al.* 2015; Sharma 2022)

Dalam model basis data jaringan, hubungan antara entitas diwakili oleh "set" dan "pointer." Set adalah kumpulan entitas yang memiliki hubungan yang sama, sementara pointer adalah referensi yang menghubungkan satu entitas dengan entitas lainnya. Model ini memungkinkan entitas untuk memiliki beberapa "orang tua," yang memungkinkan penggambaran hubungan yang lebih kompleks. Misalnya, dalam konteks organisasi, seorang karyawan dapat memiliki beberapa atasan atau dapat bekerja di beberapa departemen.

Salah satu implementasi model basis data jaringan yang terkenal adalah CODASYL DBTG (*Conference on Data Systems and Languages* - Data Base Task Group), yang digunakan dalam beberapa sistem basis data pada tahun 1960-an dan 1970-an. Model ini memberikan fleksibilitas dalam menggambarkan hubungan yang kompleks antara data, tetapi juga dapat menjadi lebih rumit dalam desain dan penggunaannya dibandingkan dengan model relasional yang lebih modern.

Meskipun model basis data jaringan memberikan fleksibilitas dalam menggambarkan hubungan yang kompleks, kekurangan utamanya adalah kompleksitas desain dan implementasinya. Model relasional yang lebih sederhana dan efisien sering kali menjadi pilihan yang lebih populer dalam pengembangan sistem basis data modern.

Contoh Basis Data Jaringan:

Untuk memberikan ilustrasi penggunaan model basis data jaringan, mari kita anggap kita memiliki sebuah perpustakaan yang ingin mengelola informasi tentang buku, penulis, dan peminjamnya menggunakan model basis data jaringan.

Entitas Utama:

1. Buku
2. Penulis
3. Peminjam

Hubungan antara Entitas:

1. Buku dapat ditulis oleh satu atau lebih penulis.
2. Buku dapat dipinjam oleh satu atau lebih peminjam.

Atribut Entitas:

1. Buku memiliki atribut seperti "Judul," "Nomor ISBN," dan "Tahun Terbit."
2. Penulis memiliki atribut "Nama Penulis" dan "Tanggal Lahir."
3. Peminjam memiliki atribut "Nama Peminjam" dan "Nomor Anggota Perpustakaan."

Ilustrasi Model Basis Data Jaringan:

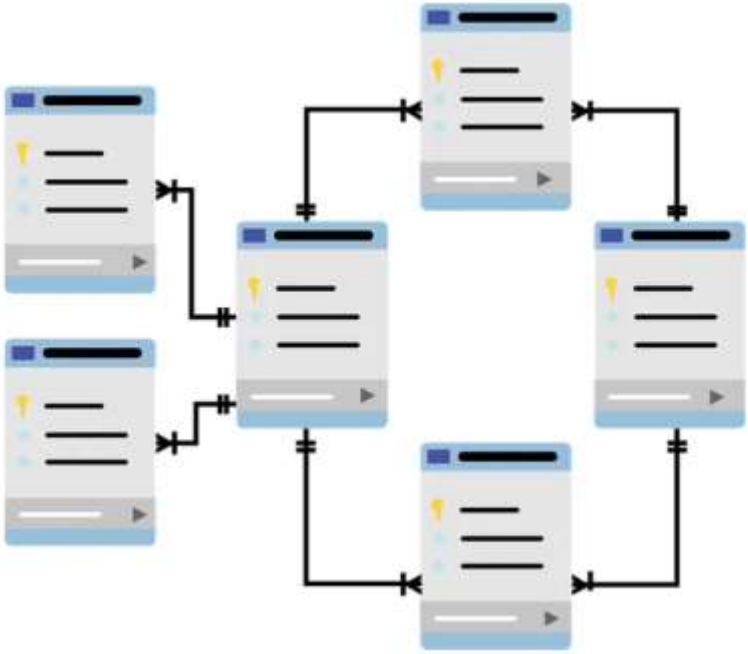
```
Buku:
- Judul: "The Great Gatsby"
- ISBN: 978-0743273565
- Tahun Terbit: 1925
  - Ditulis oleh: F. Scott Fitzgerald
  - Dipinjam oleh: (Daftar peminjam yang memiliki buku ini)

Penulis:
- Nama Penulis: F. Scott Fitzgerald
- Tanggal Lahir: 24 September 1896
  - Menulis: (Daftar buku yang ditulis oleh penulis ini)

Peminjam:
- Nama Peminjam: Alice Johnson
- Nomor Anggota Perpustakaan: 12345
  - Meminjam: (Daftar buku yang dipinjam oleh peminjam ini)
```

Dalam ilustrasi ini, "Buku," "Penulis," dan "Peminjam" adalah entitas utama dalam model basis data jaringan. Setiap buku memiliki atribut seperti judul, ISBN, dan tahun terbit. Namun, yang membuat model ini jaringan adalah hubungan antara entitas. Sebagai contoh, buku "The Great Gatsby" ditulis oleh F. Scott Fitzgerald, dan F. Scott Fitzgerald memiliki beberapa buku lain yang ditulisnya. Selain itu, peminjam Alice Johnson dapat meminjam beberapa buku, dan setiap buku dapat dipinjam oleh beberapa peminjam.

Model basis data jaringan memungkinkan representasi hubungan yang lebih kompleks seperti ini, di mana entitas dapat memiliki lebih dari satu "orang tua" atau dapat terlibat dalam berbagai hubungan. Meskipun model ini memiliki fleksibilitas dalam menggambarkan hubungan yang kompleks, perlu diingat

bahwa desain dan implementasinya bisa lebih rumit dibandingkan dengan model relasional.

2.6 Basis Data Relasional



Gambar 2.6. Basis Data Relasional
Sumber : https://towardsdatascience.com/

Model Basis Data Relasional adalah salah satu model basis data yang paling umum digunakan dalam pengembangan sistem basis data modern. Model ini didasarkan pada teori hubungan yang dikembangkan oleh Edgar F. Codd pada tahun 1970 dan mengorganisir data dalam bentuk tabel dua dimensi yang terdiri dari baris dan kolom. Dalam model ini, setiap tabel mewakili satu jenis entitas atau relasi, dan setiap baris dalam tabel mewakili satu

instansi entitas tersebut. (Warden dan Warden 1984; Kurniati *et al.* 2015; Sharma 2022)

Konsep yang sangat penting dalam model basis data relasional adalah kunci atau primary key, yang digunakan untuk mengidentifikasi secara unik setiap baris dalam tabel. Selain itu, model ini menggambarkan hubungan antara tabel dengan menggunakan kunci asing (foreign key) yang menghubungkan baris dalam satu tabel dengan baris dalam tabel lainnya. Ini memungkinkan untuk merepresentasikan hubungan satu-ke-satu, satu-ke-banyak, atau banyak-ke-banyak antara entitas.

Keuntungan utama dari model basis data relasional adalah kemampuannya untuk mengorganisir data dengan struktur yang terstruktur dan terorganisir dengan baik. SQL (*Structured Query Language*) digunakan untuk mengakses, mengambil, memperbarui, dan memanipulasi data dalam model ini, sehingga memudahkan pemrosesan data. Model relasional sangat cocok untuk aplikasi bisnis seperti manajemen inventaris, pelacakan pelanggan, dan analisis data.

Selain itu, model basis data relasional juga menawarkan keamanan yang baik melalui pengaturan hak akses pengguna dan enkripsi data. Hal ini membuatnya sesuai untuk mengelola data yang sensitif seperti informasi pelanggan dan keuangan perusahaan. Meskipun model ini sangat berguna dan serbaguna, pemilihan struktur tabel dan pengoptimalan kueri adalah aspek yang penting untuk memastikan kinerja yang baik dalam implementasinya.

Contoh Basis Data Relasional:

Untuk memberikan ilustrasi penggunaan model basis data relasional, mari kita anggap ada sebuah perusahaan fiksi yang ingin mengelola informasi tentang pelanggan, pesanan, dan produk menggunakan model ini.

Entitas Utama:
1. Pelanggan
2. Pesanan
3. Produk

Hubungan antara Entitas:
1. Setiap pelanggan dapat melakukan banyak pesanan.
2. Setiap pesanan hanya dimiliki oleh satu pelanggan.
3. Setiap pesanan berisi beberapa produk.
4. Setiap produk dapat ada dalam beberapa pesanan.

Atribut Entitas:
1. Pelanggan memiliki atribut seperti "Nama Pelanggan," "Alamat," dan "Nomor Telepon."
2. Pesanan memiliki atribut "Nomor Pesanan" dan "Tanggal Pesanan."
3. Produk memiliki atribut seperti "Nama Produk," "Harga," dan "Stok."

Ilustrasi Model Basis Data Relasional:
Tabel "Pelanggan":

ID Pelanggan	Nama Pelanggan	Alamat	Nomor Telepon
1	Alice Johnson	Jalan A No. 123	555-1234
2	Bob Smith	Jalan B No. 456	555-5678

Tabel "Pesanan":

ID Pesanan	Nomor Pesanan	Tanggal Pesanan	ID Pelanggan
101	P101	2023-01-15	1
102	P102	2023-02-10	2

Tabel "Produk":

ID Produk	Nama Produk	Harga	Stok
201	Buku	20.00	50
202	Laptop	800.00	10
203	Smartphone	400.00	30

Tabel "Pesanan_Produk" (tabel perantara yang menghubungkan pesanan dengan produk):

ID Pesanan	ID Produk
101	201
101	203
102	202
102	203

Dalam ilustrasi ini, model basis data relasional digunakan untuk mengelola data pelanggan, pesanan, dan produk. Setiap tabel mewakili satu jenis entitas atau relasi, dan relasi antar entitas dijelaskan melalui kunci asing. Misalnya, tabel "Pesanan" memiliki kunci asing "ID Pelanggan" yang menghubungkannya dengan tabel "Pelanggan."

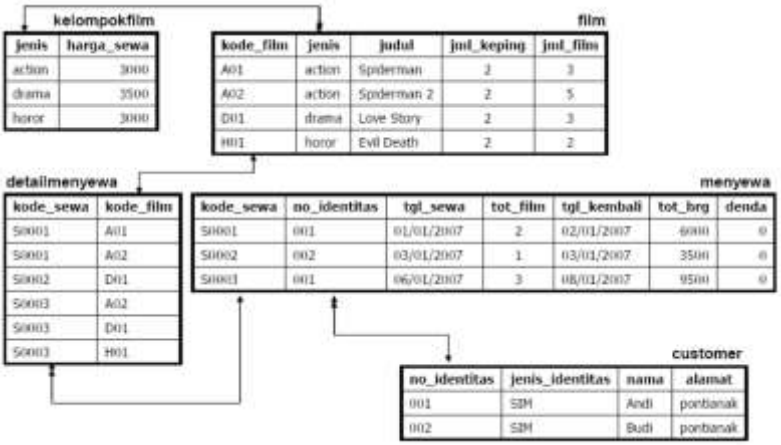
Dengan menggunakan SQL, Anda dapat mengambil data dengan berbagai cara. Contoh kueri SQL mungkin mencakup:

1. Menampilkan semua pesanan yang dilakukan oleh seorang pelanggan tertentu.

2. Menampilkan semua produk dalam satu pesanan.
3. Menghitung total harga pesanan.
4. Menambahkan atau memperbarui informasi pelanggan, pesanan, atau produk.

Model basis data relasional memungkinkan untuk mengatur, mengakses, dan mengelola data dengan struktur yang terstruktur dan kohesif, serta memudahkan kueri dan analisis data dalam aplikasi bisnis seperti sistem e-commerce atau manajemen inventaris.

2.7 Basis Data Objek



Gambar 2.7. Basis Data Objek
Sumber : https://sis.binus.ac.id/

Model Basis Data Objek adalah salah satu jenis model basis data yang lebih modern, yang memungkinkan penyimpanan dan manipulasi data yang lebih kompleks daripada model-model tradisional seperti model relasional. Dalam model ini, data

diorganisir sebagai objek-objek yang dapat memiliki atribut dan metode yang mirip dengan objek dalam pemrograman berorientasi objek. Ini memungkinkan data dan perilaku terkait untuk digabungkan menjadi satu unit, yang membuatnya cocok untuk aplikasi yang kompleks dan dinamis. (Warden dan Warden 1984; Kurniati *et al.* 2015; Sharma 2022)

Dalam model basis data objek, entitas atau objek-objek dapat didefinisikan bersama dengan atribut-atributnya. Sebagai contoh, dalam konteks sistem manajemen inventaris, produk dapat didefinisikan sebagai objek yang memiliki atribut seperti nama produk, harga, dan jumlah stok. Selain itu, objek-objek ini dapat memiliki metode atau fungsi yang memungkinkan operasi tertentu dilakukan pada objek tersebut, seperti menghitung total harga produk.

Salah satu keuntungan utama dari model basis data objek adalah kemampuannya untuk merepresentasikan hubungan yang kompleks antara objek dan menggabungkan data dan perilaku dalam satu kesatuan. Model ini juga mendukung konsep pewarisan, di mana satu objek dapat mewarisi atribut dan metode dari objek lain, yang memungkinkan pengembangan sistem yang lebih modular dan mudah diatur.

Namun, penting untuk diingat bahwa model basis data objek tidak selalu menjadi pilihan yang sesuai untuk setiap aplikasi. Penggunaannya lebih umum dalam pengembangan perangkat lunak berorientasi objek dan sering digunakan bersama dengan bahasa pemrograman berorientasi objek seperti Java dan C++. Model ini mungkin memerlukan pemahaman yang lebih mendalam dalam pemrograman berorientasi objek dan biasanya memiliki kompleksitas yang lebih tinggi dalam pengembangan dan pemeliharaan.

Contoh Basis Data Objek:

Ilustrasi penggunaan model basis data objek dapat diberikan dalam konteks aplikasi manajemen perpustakaan di

mana objek-objek seperti buku, anggota perpustakaan, dan peminjaman dapat diwakili dengan atribut dan metode. Berikut adalah ilustrasi penggunaan model basis data objek:

Objek Utama:

1. **Buku**: Objek yang mewakili setiap buku dalam perpustakaan. Buku memiliki atribut seperti "Judul," "Penulis," "Nomor ISBN," dan "Stok." Objek buku juga memiliki metode seperti "Peminjaman" dan "Pengembalian" untuk mengelola status peminjaman buku.
2. **Anggota Perpustakaan**: Objek yang mewakili anggota perpustakaan. Objek ini memiliki atribut seperti "Nama Anggota," "Nomor Anggota," dan "Alamat." Objek anggota perpustakaan juga memiliki metode seperti "Peminjaman" untuk meminjam buku dari perpustakaan.

Hubungan antara Objek:

1. Setiap anggota perpustakaan dapat melakukan beberapa peminjaman buku.
2. Setiap buku dapat dipinjam oleh beberapa anggota perpustakaan.

Ilustrasi Model Basis Data Objek:

```
# Definisi objek Buku
class Buku:
    def __init__(self, judul, penulis, isbn, stok):
        self.judul = judul
        self.penulis = penulis
        self.isbn = isbn
        self.stok = stok

    def peminjaman(self):
        if self.stok > 0:
            self.stok -= 1
            return f"Buku '{self.judul}' telah dipinjam."
        else:
            return "Maaf, buku ini sudah habis."

    def pengembalian(self):
        self.stok += 1
        return f"Buku '{self.judul}' telah dikembalikan."

# Definisi objek Anggota Perpustakaan
class AnggotaPerpustakaan:
    def __init__(self, nama, nomor_anggota, alamat):
        self.nama = nama
        self.nomor_anggota = nomor_anggota
        self.alamat = alamat
        self.peminjaman = []
```

```
    def peminjaman_buku(self, buku):
        if buku not in self.peminjaman:
            self.peminjaman.append(buku)
            return f"Buku '{buku.judul}' berhasil dipinjam oleh {self.nama}.
        else:
            return f"Anda sudah meminjam buku '{buku.judul}' sebelumnya."

    def daftar_peminjaman(self):
        return [buku.judul for buku in self.peminjaman]

# Contoh penggunaan model basis data objek
buku1 = Buku("Harry Potter and the Sorcerer's Stone", "J.K. Rowling", "978-0
buku2 = Buku("To Kill a Mockingbird", "Harper Lee", "978-0061120084", 8)

anggota1 = AnggotaPerpustakaan("Alice Johnson", "A12345", "Jalan A No. 123")
anggota2 = AnggotaPerpustakaan("Bob Smith", "B56789", "Jalan B No. 456")

anggota1.peminjaman_buku(buku1)
anggota1.peminjaman_buku(buku2)
anggota2.peminjaman_buku(buku1)

print("Daftar Peminjaman oleh Alice Johnson:", anggota1.daftar_peminjaman())
print("Daftar Peminjaman oleh Bob Smith:", anggota2.daftar_peminjaman())
```

Dalam ilustrasi ini, model basis data objek digunakan untuk mengelola data buku dan anggota perpustakaan. Objek-objek buku dan anggota perpustakaan memiliki atribut yang mewakili informasi mereka dan metode untuk mengelola operasi seperti peminjaman dan pengembalian buku. Hubungan antara objek-objek ini dijelaskan oleh cara anggota perpustakaan dapat meminjam buku, dan buku dapat dipinjam oleh beberapa anggota perpustakaan. Model ini mencerminkan cara data dan perilaku terkait digabungkan dalam satu unit dalam pemrograman berorientasi objek.

DAFTAR PUSTAKA

Kurniati ID, Setiawan R, Rohmani A, Lahdji A, Tajally A, Ratnaningrum K, Basuki R, Reviewer S, Wahab Z. 2015. *Basis Data*. MEDIA SAINS INDONESIA.

Sharma DA. 2022. *Database Management Systems/Managing Database DCAP402*. EXCEL BOOKS PRIVATE LIMITED Regd.

Warden WH, Warden BM. 1984. *An Introduction to Database Management Systems*. Volume ke-2.

BAB 3
SISTEM MANAJEMEN BASIS DATA

Oleh Melissa Indah Fianty

3.1 Apa itu Sistem Manajemen Basis Data?

Manajemen data yang efisien biasanya memerlukan penggunaan basis data komputer. Sebuah basis data adalah struktur komputer bersama yang menyimpan kumpulan hal-hal berikut:

1. *End-user data*, yaitu fakta-fakta mentah yang menarik bagi *end user*.
2. Metadata, atau data tentang data, melalui data end-user diintegrasikan dan dikelola.

Metadata menjelaskan karakteristik data dan kumpulan hubungan yang menghubungkan data yang ditemukan dalam basis data (Amiraslani, 2021). Sebagai contoh, komponen metadata mengandung informasi seperti nama masing-masing elemen data, jenis nilai (numerik, tanggal, atau teks) yang tersimpan di setiap elemen data, dan apakah elemen data tersebut memiliki opsi untuk dibiarkan kosong. Metadata memberikan tambahan informasi yang memperkaya nilai dan penggunaan data. Dengan kata lain, metadata memberikan pandangan yang lebih komprehensif tentang data yang ada dalam basis data. Dengan mempertimbangkan sifat metadata, mungkin Anda pernah mendengar deskripsi basis data sebagai "kumpulan data yang dapat menjelaskan dirinya sendiri.

Sistem manajemen basis data (*Database Management System* atau DBMS) terdiri dari serangkaian program yang bertanggung jawab untuk mengelola struktur basis data dan

mengontrol akses ke data yang tersimpan di dalamnya. Dalam arti tertentu, basis data dapat dibandingkan dengan rak penyimpanan digital yang sangat terorganisir, dan perangkat lunak andal seperti DBMS berperan penting dalam mengelola isi dari rak penyimpanan tersebut (Carlos Coronel, 2018).

Perangkat lunak manajemen basis data (*Database Management System* atau DBMS) adalah program yang berinteraksi dengan aplikasi pengguna dan basis data. Biasanya, DBMS memberikan layanan berikut:

1. DBMS memungkinkan pengguna untuk mendefinisikan basis data mereka sendiri, biasanya melalui Bahasa Definisi Data (DDL). DDL memungkinkan pengguna menentukan jenis data, struktur, dan batasan data yang akan disimpan dalam basis data.
2. Pengguna dapat memasukkan, memperbarui, menghapus, dan mengambil data dari basis data melalui Bahasa Manipulasi Data (DML). Dengan memiliki repositori pusat untuk semua data dan deskripsi data, DML menyediakan fasilitas kueri yang umum untuk data ini, yang disebut bahasa kueri. Penggunaan bahasa kueri mengatasi masalah yang muncul dalam sistem berbasis file, di mana pengguna harus berurusan dengan berbagai kueri yang bervariasi atau perlu membuat banyak program, yang mengakibatkan masalah manajemen perangkat lunak yang kompleks. Bahasa kueri yang paling umum digunakan adalah *Structured Query Language* (SQL), yang saat ini merupakan bahasa standar formal dan de facto untuk DBMS relasional.
3. DBMS menyediakan akses yang terkendali ke basis data. Misalnya, ini dapat mencakup:
    a. Sistem keamanan, yang mencegah akses tidak sah ke basis data.
    b. Sistem integritas, yang menjaga konsistensi data yang tersimpan.

c. Sistem kontrol konkurensi, yang memungkinkan akses bersama ke basis data.
d. Sistem pengendalian pemulihan, yang mengembalikan basis data ke keadaan konsisten setelah kegagalan perangkat keras atau perangkat lunak.
e. Katalog yang dapat diakses oleh pengguna, yang berisi deskripsi data dalam basis data.

4. Pengguna berinteraksi dengan basis data melalui berbagai program aplikasi yang digunakan untuk membuat, mengelola, dan menghasilkan informasi dari basis data. Program-program ini bisa berbentuk aplikasi batch tradisional atau lebih umum saat ini, aplikasi online. Program aplikasi tersebut dapat ditulis dalam berbagai bahasa pemrograman atau bahasa tingkat tinggi generasi keempat. (Tray et al., 2020).

Tampilan (Views)

Dengan fungsionalitas ini, DBMS adalah alat yang sangat kuat dan berguna. Namun, karena pengguna akhir tidak terlalu tertarik pada seberapa kompleks atau mudahnya tugas bagi sistem, bisa dikatakan bahwa DBMS telah membuat hal-hal menjadi lebih kompleks, karena sekarang mereka melihat lebih banyak data daripada yang mereka butuhkan atau inginkan (Curé & Blin, 2015).

Tingkat fungsionalitas sebenarnya yang ditawarkan oleh DBMS berbeda dari satu produk ke produk lainnya. Sebagai contoh, DBMS untuk komputer pribadi mungkin tidak mendukung akses bersama konkurensi, dan mungkin hanya menyediakan perlindungan keamanan, integritas, dan kontrol pemulihan yang terbatas (Xu, 2021). Namun, produk DBMS multi-pengguna modern yang besar menawarkan semua fungsi yang disebutkan dan banyak lagi. Sistem modern adalah perangkat lunak yang sangat kompleks yang terdiri dari jutaan baris kode, dengan dokumentasi yang terdiri dari banyak volume. Kompleksitas ini

adalah hasil dari harus menyediakan perangkat lunak yang mengatasi persyaratan yang lebih umum. Selain itu, penggunaan DBMS saat ini memerlukan sistem yang memberikan reliabilitas hampir total dan ketersediaan 24/7 (24 jam sehari, 7 hari seminggu), bahkan dalam situasi kegagalan perangkat keras atau perangkat lunak. DBMS terus berkembang dan memperluas diri untuk menghadapi kebutuhan pengguna baru. Sebagai contoh, beberapa aplikasi sekarang memerlukan penyimpanan gambar grafis, video, suara, dan sebagainya. Untuk mencapai pasar ini, DBMS harus berubah. Kemungkinan besar, fungsionalitas baru akan selalu diperlukan, sehingga fungsionalitas DBMS tidak akan pernah menjadi statis (Tray et al., 2020).

3.2 Peran Sistem Manajemen Basis Data?

Dalam konteks lingkungan DBMS, kita dapat mengenali empat peran yang berpartisipasi: administrator basis data dan data, perancang basis data, pengembang aplikasi, dan pengguna akhir.

3.2.1 Data and Database Administrators Basis data bersama dengan sistem manajemen basis data (DBMS)

Data and Database Administrators Basis data bersama dengan sistem manajemen basis data (DBMS) merupakan aset perusahaan yang harus dikelola dengan cara yang sama seperti aset lainnya. Administrasi data dan basis data adalah peran yang biasanya terkait dengan manajemen dan pengawasan DBMS serta informasinya.

Administrator Data (DA) Tanggung jawabnya meliputi pengelolaan aset data, termasuk perencanaan basis data, pembuatan serta pemeliharaan standar, kebijakan, dan prosedur, serta perancangan konseptual dan logis basis data. Dalam peran ini, Dia melakukan konsultasi dengan manajer senior dan memberikan nasihat, serta memastikan bahwa perkembangan basis data pada akhirnya akan mendukung tujuan perusahaan.

Administrator Basis Data (DBA) Tugasnya mencakup pelaksanaan aspek fisik basis data, seperti merancang dan menerapkan struktur fisik basis data, mengawasi keamanan dan integritas data, menjaga sistem operasional berjalan lancar, dan memastikan bahwa aplikasi berjalan dengan baik untuk pengguna. Peran Database Administrator (DBA) lebih berfokus pada aspek teknis dibandingkan dengan peran Data Administrator (DA), memerlukan pengetahuan mendalam tentang DBMS yang digunakan dan lingkungan sistem yang berkaitan. Di beberapa organisasi, tidak ada perbedaan yang jelas antara kedua peran ini; di tempat lain, pentingnya sumber daya perusahaan tercermin dalam penugasan tim khusus yang bertanggung jawab atas masing-masing peran ini.

3.2.2 Database Designers

Dalam proyek desain basis data yang besar, kita dapat mengidentifikasi dua jenis perancang: perancang basis data konseptual dan perancang basis data fisik. Perancang basis data konseptual berkaitan dengan mengidentifikasi struktur data (seperti entitas dan atribut), hubungan antara data, dan pembatasan data yang akan disimpan dalam basis data

Untuk sukses, perancang basis data konseptual harus melibatkan semua calon pengguna basis data dalam pengembangan model data, dan keterlibatan ini sebaiknya dimulai sejak awal dalam proses. Dalam konteks buku ini, kami memecah pekerjaan perancang basis data konseptual menjadi dua tahap:

1. Desain basis data konseptual adalah tahap awal yang tidak terkait dengan implementasi spesifik, seperti DBMS yang akan digunakan, aplikasi yang akan dibangun, bahasa pemrograman yang akan digunakan, atau pertimbangan fisik lainnya.

2. Desain basis data logis berfokus pada struktur data yang akan digunakan dalam model tertentu, seperti model relasional, jaringan, hierarki, atau berorientasi objek.

Perancang Perancang basis data fisik memiliki peran dalam menentukan bagaimana desain basis data logis akan diwujudkan secara fisik. Ini melibatkan:

1. Merubah desain basis data logis menjadi serangkaian tabel dan menetapkan pembatasan integritas yang relevan.
2. Memilih struktur penyimpanan yang tepat dan metode akses data khusus untuk mencapai performa yang optimal.
3. Menentukan langkah-langkah keamanan yang diperlukan untuk melindungi data

Banyak aspek dari desain basis data fisik sangat bergantung pada DBMS yang digunakan, dan seringkali ada beberapa cara untuk menerapkan mekanisme tersebut. Karena itu, perancang basis data fisik perlu memiliki pemahaman mendalam tentang kemampuan DBMS yang digunakan dan juga harus memahami kelebihan serta kekurangan dari berbagai opsi implementasi yang ada. Perancang basis data fisik harus memiliki kemampuan untuk memilih strategi penyimpanan yang sesuai, dengan mempertimbangkan penggunaannya. Sementara desain basis data konseptual dan logis berkaitan dengan apa yang harus dicapai, desain basis data fisik berkaitan dengan bagaimana mencapainya. Hal ini memerlukan keterampilan yang berbeda yang mungkin dimiliki oleh individu yang berbeda.

3.2.3 Application Developers

Setelah basis data selesai diimplementasikan, tahap berikutnya adalah mengembangkan program aplikasi yang akan memberikan fungsionalitas yang dibutuhkan oleh pengguna akhir. Tugas ini menjadi tanggung jawab dari pengembang aplikasi.

Secara umum, pengembang aplikasi akan bekerja berdasarkan spesifikasi yang telah disusun oleh analis sistem. Setiap program yang dibuat mengandung instruksi yang meminta DBMS untuk melakukan berbagai operasi pada basis data, termasuk pengambilan data, penambahan, pembaruan, dan penghapusan data. Program-program ini dapat dikodekan menggunakan bahasa pemrograman tingkat ketiga atau tingkat keempat, seperti yang telah dijelaskan sebelumnya.

3.2.4 End-Users

Pengguna akhir merupakan pihak yang "menggunakan" basis data yang telah dirancang, diimplementasikan, dan dipelihara untuk memenuhi kebutuhan informasi mereka. Kelompok pengguna akhir dapat dibagi berdasarkan cara mereka berinteraksi dengan sistem:

1. Pengguna awam (*naïve users*) umumnya tidak memiliki kesadaran mengenai adanya DBMS. Mereka mengakses basis data melalui program aplikasi yang dirancang khusus untuk menyederhanakan operasi. Mereka menggunakan operasi basis data dengan mengirimkan perintah sederhana atau memilih opsi dari menu. Dengan kata lain, mereka tidak perlu memiliki pengetahuan tentang basis data atau DBMS. Sebagai contoh, seorang asisten kasir di toko lokal menggunakan pembaca kode batang untuk menentukan harga barang. Namun, di balik layar, ada program aplikasi yang mengenali kode batang, mencari harga barang dalam basis data, mengurangi jumlah barang yang tersedia dalam stok, dan menampilkan harga di layar kasir.
2. Pengguna mahir (*sophisticated users*), di sisi lain, memiliki pemahaman yang mendalam tentang struktur basis data dan kemampuan yang diberikan oleh DBMS. Pengguna mahir mungkin menggunakan bahasa kueri tingkat tinggi seperti SQL untuk melakukan operasi yang diperlukan.

Bahkan, beberapa pengguna mahir mungkin menulis program aplikasi mereka sendiri untuk keperluan khusus mereka (Wang et al., 2018).

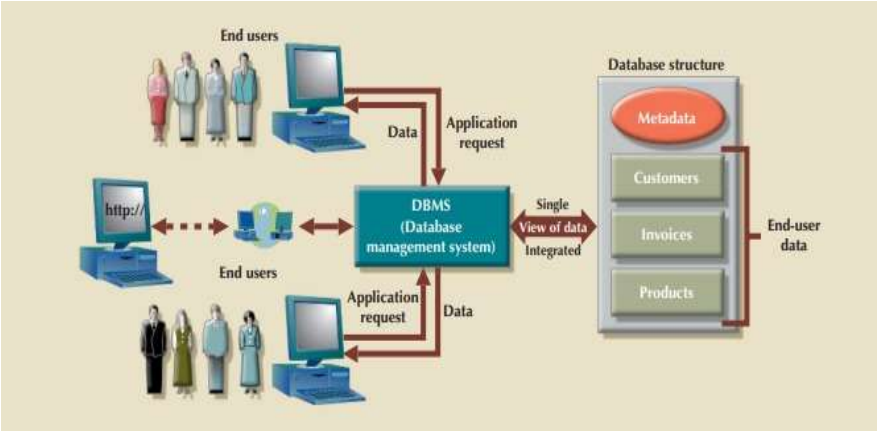
3.3 Keuntungan dan kekurangan Sistem Manajemen Basis Data?

Sistem Manajemen Basis Data (DBMS) berperan sebagai perantara antara pengguna dan basis data. Struktur dasar dari basis data disimpan sebagai koleksi berkas, dan satu-satunya cara untuk mengakses data yang ada dalam berkas-berkas tersebut adalah melalui DBMS. Dalam Gambar 3.1, terlihat dengan jelas bahwa DBMS menyajikan pandangan terpadu dan seragam kepada pengguna akhir (atau program aplikasi) tentang data dalam basis data. Setiap permintaan yang diajukan oleh aplikasi diterima oleh DBMS, dan DBMS kemudian menerjemahkannya menjadi operasi-operasi yang rumit yang diperlukan untuk memenuhi permintaan tersebut. Dengan melakukan ini, DBMS menyembunyikan sebagian besar kompleksitas internal dari basis data dari mata program aplikasi dan pengguna. Program aplikasi bisa dibuat oleh seorang pengembang menggunakan bahasa pemrograman seperti Visual Basic .NET, Java, atau C#, atau bahkan dibuat melalui utilitas yang disediakan oleh DBMS (Zhou, 2012).

Keberadaan DBMS sebagai perantara antara pengguna akhir dan basis data memberikan beberapa keunggulan yang signifikan. Pertama, DBMS memungkinkan data dalam basis data dapat dibagikan di antara berbagai aplikasi atau pengguna. Kedua, DBMS menggabungkan berbagai pandangan yang berbeda dari banyak pengguna terhadap data menjadi satu repository data yang komprehensif.

Data merupakan komponen penting sebagai bahan dasar untuk menghasilkan informasi. Sebagaimana diuraikan dalam buku ini, DBMS membantu dalam efisiensi dan efektivitas pengelolaan

data. Secara khusus, DBMS memberikan berbagai keuntungan, termasuk (Carral et al., 2019):



Gambar 3.1. DBMS mengelola interaksi antara *end user* dan basis data
(Sumber: (Carlos Coronel, 2018))

1. *Enhanced* data *sharing*. DBMS berperan dalam menciptakan lingkungan di mana pengguna akhir dapat dengan lebih baik mengakses sejumlah besar data yang terstruktur dan diatur dengan baik. Kemampuan ini memungkinkan pengguna akhir untuk merespons perubahan dalam lingkungan mereka dengan lebih cepat.
2. *Enhanced* data *security.* Semakin banyak pengguna yang mengakses data, semakin besar risiko pelanggaran keamanan data. Perusahaan meluangkan waktu, upaya, dan sumber daya finansial yang signifikan untuk memastikan penggunaan yang benar dari data perusahaan. DBMS menyediakan kerangka kerja untuk menerapkan dan mematuhi kebijakan privasi serta keamanan data yang lebih kuat.

3. *Enhanced* data *integration*. Akses yang lebih luas ke data yang teratur membantu dalam menciptakan pandangan yang lebih terpadu tentang operasi perusahaan serta pemahaman yang lebih baik tentang gambaran keseluruhan. Ini memudahkan dalam melihat bagaimana tindakan di satu bagian perusahaan dapat memengaruhi bagian lainnya.
4. *Minimized* data *inconsistency*. Inconsistensi data terjadi saat variasi dalam data yang sama terdeteksi di lokasi yang berbeda. Sebagai contoh, inkonsistensi data mungkin muncul ketika departemen penjualan sebuah perusahaan menyimpan nama seorang perwakilan penjualan sebagai Bill Brown sementara departemen sumber daya manusia perusahaan menyimpan nama yang sama sebagai William G. Brown. Kemungkinan terjadinya inkonsistensi data berkurang secara signifikan dalam basis data yang dirancang dengan baik.
5. *Enhanced* data *access*. DBMS memungkinkan untuk mendapatkan respons cepat terhadap pertanyaan-pertanyaan spontan. Dalam konteks basis data, query adalah permintaan khusus kepada DBMS untuk memanipulasi data, seperti membaca atau memperbarui data. Dengan kata lain, query adalah pertanyaan, dan query yang bersifat spontan (ad hoc) adalah pertanyaan yang muncul secara mendadak. DBMS memberikan hasil query kepada aplikasi. Sebagai contoh, ketika menghadapi volume data penjualan yang besar, pengguna akhir mungkin memerlukan respons cepat terhadap pertanyaan-pertanyaan seperti:
    a. Berapa total volume penjualan per produk selama enam bulan terakhir?
    b. Berapa bonus penjualan untuk setiap karyawan penjualan selama tiga bulan terakhir?

c. Berapa jumlah pelanggan yang memiliki saldo kredit sebesar $3.000 atau lebih?

6. Enhanced decision making. Manajemen data yang lebih baik dan akses yang ditingkatkan memungkinkan penghasilan informasi berkualitas yang lebih tinggi, yang membentuk dasar pengambilan keputusan yang lebih baik. Kualitas informasi yang dihasilkan sangat tergantung pada kualitas data yang menjadi dasarnya. Kualitas data adalah upaya komprehensif untuk meningkatkan akurasi, validitas, dan ketepatan waktu data. Meskipun DBMS tidak menjamin kualitas data, ia memberikan kerangka kerja yang memfasilitasi inisiatif perbaikan kualitas data.
7. Enhanced end-user productivity. Ketersediaan data, digabungkan dengan alat-alat yang mengubah data menjadi informasi yang berguna, memberdayakan pengguna akhir untuk membuat keputusan yang cepat dan terinformasi. Hal ini bisa menjadi faktor penentu antara kesuksesan dan kegagalan dalam konteks ekonomi global.

Keuntungan penggunaan DBMS tidak terbatas pada yang hanya disebutkan di atas. Faktanya, Anda akan menemukan banyak keuntungan lain saat Anda mempelajari lebih lanjut tentang detail teknis basis data dan desain yang benar.

DBMS terdapat beberapa kekurangan berikut:

1. *Complexity.* Penyediaan fungsionalitas yang kita harapkan dari DBMS yang baik membuat DBMS menjadi perangkat lunak yang sangat kompleks. Perancang dan pengembang basis data, administrator data dan basis data, serta pengguna akhir harus memahami fungsionalitas ini untuk memanfaatkannya sepenuhnya. Kegagalan dalam memahami sistem ini dapat mengakibatkan keputusan desain yang buruk, yang dapat memiliki konsekuensi serius bagi sebuah organisasi.

2. *Size.* Kompleksitas dan cakupan fungsionalitas membuat DBMS menjadi perangkat lunak yang sangat besar, memerlukan banyak megabita ruang disk dan memerlukan jumlah memori yang substansial untuk berjalan dengan efisien.
3. *Cost of DBMS* bervariasi secara signifikan, tergantung pada lingkungan dan fungsionalitas yang disediakan. Sebagai contoh, sebuah DBMS untuk satu pengguna pada komputer pribadi mungkin hanya biaya $100. Namun, sebuah DBMS multi-pengguna besar untuk mainframe yang melayani ratusan pengguna bisa sangat mahal, mungkin sekitar $100,000 atau bahkan $1,000,000. Ada juga biaya pemeliharaan tahunan berulang, yang biasanya merupakan persentase dari harga daftar.
4. *Additional hardware costs*. Kebutuhan penyimpanan disk untuk DBMS dan basis data mungkin memerlukan pembelian ruang penyimpanan tambahan. Selain itu, untuk mencapai kinerja yang diperlukan, mungkin perlu membeli mesin yang lebih besar, bahkan mungkin sebuah mesin yang didedikasikan untuk menjalankan DBMS. Pengadaan perangkat keras tambahan mengakibatkan pengeluaran lebih lanjut.
5. *Cost of conversion.* Dalam beberapa situasi, biaya DBMS dan perangkat keras tambahan mungkin relatif kecil dibandingkan dengan biaya mengkonversi aplikasi yang sudah ada untuk berjalan pada DBMS dan perangkat keras baru. Biaya ini juga mencakup biaya pelatihan staf untuk menggunakan sistem baru ini, dan mungkin juga perekrutan staf ahli untuk membantu dalam konversi dan pengoperasian sistem. Biaya ini adalah salah satu alasan utama mengapa beberapa organisasi merasa terikat pada sistem saat ini dan tidak dapat beralih ke teknologi basis data yang lebih modern. Istilah "sistem warisan" kadang-

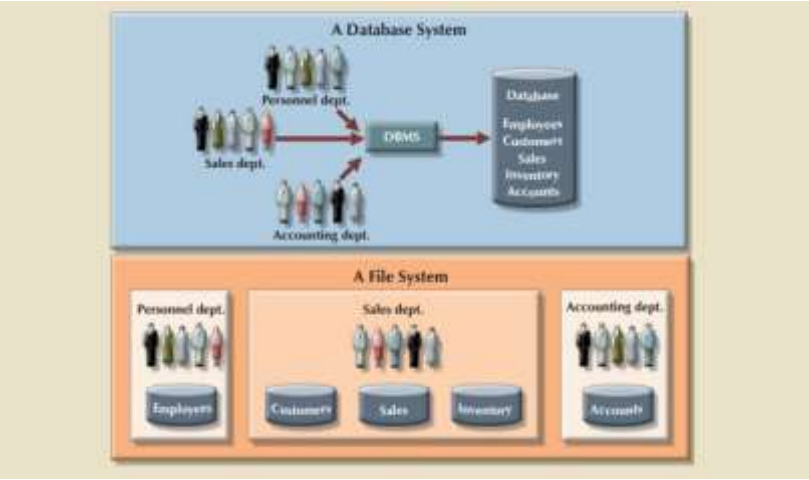
kadang digunakan untuk merujuk pada sistem yang lebih lama, dan biasanya lebih rendah mutunya.

6. Performance. Biasanya, sistem berbasis file ditulis untuk aplikasi tertentu, seperti faktur. Akibatnya, kinerjanya umumnya sangat baik. Namun, DBMS ditulis untuk lebih umum, untuk melayani banyak aplikasi daripada hanya satu. Hasilnya adalah bahwa beberapa aplikasi mungkin tidak berjalan secepat seperti sebelumnya.
7. *Greater impact of a failure.* Pusatnya sumber daya meningkatkan kerentanan sistem. Karena semua pengguna dan aplikasi mengandalkan ketersediaan DBMS, kegagalan komponen tertentu dapat menghentikan operasi.

3.4 Sistem Basis Data

Masalah-masalah yang melekat dalam sistem berkas membuat penggunaan sistem basis data menjadi sangat diinginkan. Berbeda dengan sistem berkas yang memiliki banyak berkas terpisah dan tidak berhubungan, sistem basis data terdiri dari data yang terkait secara logis yang disimpan dalam satu repositori data logis. (Label "logis" mencerminkan kenyataan bahwa repositori data ini tampak sebagai satu kesatuan bagi pengguna akhir, meskipun data mungkin tersebar secara fisik di berbagai fasilitas penyimpanan dan lokasi.) Karena repositori data basis data adalah satu kesatuan logis, basis data ini merupakan perubahan besar dalam cara data pengguna akhir disimpan, diakses, dan dikelola. DBMS basis data, yang ditampilkan dalam Gambar 3.2, memberikan banyak keunggulan dibandingkan dengan manajemen sistem berkas, yang ditampilkan dalam Gambar 3.1, dengan memungkinkan eliminasi sebagian besar masalah inkonsistensi data, anomali data, ketergantungan data, dan ketergantungan struktural dalam sistem berkas. Yang lebih baik lagi, generasi saat ini dari perangkat lunak DBMS menyimpan bukan hanya struktur data, tetapi juga hubungan antara struktur-

struktur tersebut dan jalur akses ke struktur-struktur tersebut—semuanya dalam lokasi pusat. Generasi saat ini dari perangkat lunak DBMS juga mengurus definisi, penyimpanan, dan manajemen semua jalur akses yang diperlukan ke komponen-komponen tersebut.



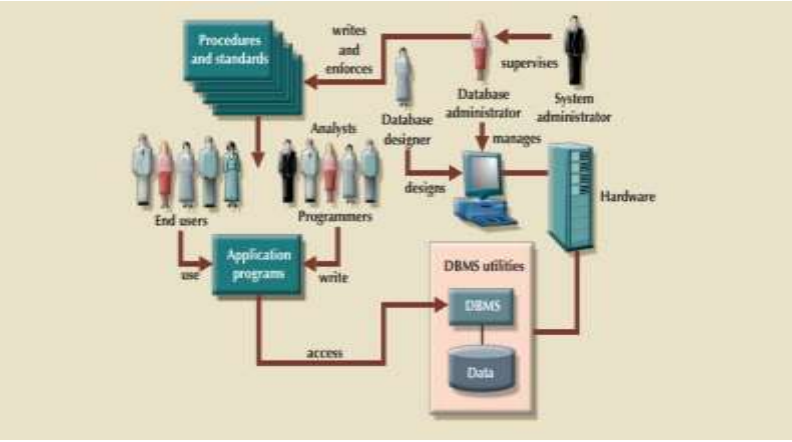
Gambar 3.2. Perbedaan basis data dan sistem file
(Sumber: (Carlos Coronel, 2018))

Ingatlah bahwa DBMS hanyalah salah satu komponen penting dari sistem basis data. DBMS bahkan bisa disebut sebagai "jantung" sistem basis data. Namun, seperti halnya manusia memerlukan lebih dari sekadar jantung untuk berfungsi, begitu juga sistem basis data memerlukan lebih dari sekadar DBMS untuk berfungsi. Dalam bagian-bagian berikutnya, Anda akan mempelajari apa itu sistem basis data, apa komponennya, dan bagaimana DBMS berperan dalam gambaran tersebut.

3.5 Komponen pada Sistem Manajemen Basis Data

Sistem basis data mengacu pada struktur organisasi yang terdiri dari elemen-elemen yang mendefinisikan, mengatur, serta mengelola pengumpulan dan penyimpanan data dalam lingkungan basis data. Dari perspektif manajemen umum, sistem basis data dapat dilihat sebagai kombinasi dari lima komponen inti yang terlihat dalam Gambar 3.3: perangkat keras (hardware), perangkat lunak (software), individu (individuals), prosedur (procedures), dan data. Mari kita periksa lebih detail kelima elemen ini yang tergambar dalam Gambar 3.3:

1. Perangkat Keras (Hardware): Perangkat keras merujuk pada semua perangkat fisik sistem, termasuk komputer (PC, tablet, workstation, server, dan superkomputer), perangkat penyimpanan, pencetak (printer), perangkat jaringan (hub, switch, router, serat optik), dan perangkat lainnya (mesin penarik otomatis, pembaca ID, dan sebagainya).



Gambar 3.3. Komponen pada Sistem Manajemen Basis Data
(Sumber: (Carlos Coronel, 2018))

2. Perangkat Lunak (Software): Meskipun perangkat lunak yang paling dikenal adalah Sistem Manajemen Basis Data (DBMS) itu sendiri, ada tiga jenis perangkat lunak yang diperlukan agar sistem basis data berfungsi sepenuhnya: perangkat lunak sistem operasi, perangkat lunak DBMS, dan program aplikasi dan utilitas.
    a. Perangkat lunak sistem operasi mengelola semua komponen perangkat keras dan membuatnya memungkinkan bagi semua perangkat lunak lainnya untuk berjalan di komputer. Contoh perangkat lunak sistem operasi termasuk Microsoft Windows, Linux, Mac OS, UNIX, dan MVS.
    b. Perangkat lunak DBMS mengelola basis data dalam sistem basis data. Beberapa contoh perangkat lunak DBMS termasuk Microsoft SQL Server, Oracle Corporation Oracle, MySQL Oracle, dan IBM DB2.
    c. Program aplikasi dan perangkat lunak utilitas digunakan untuk mengakses dan memanipulasi data dalam DBMS dan mengelola lingkungan komputer di mana akses dan manipulasi data berlangsung. Program aplikasi paling umum digunakan untuk mengakses data dalam basis data untuk menghasilkan laporan, tabel, dan informasi lainnya untuk memfasilitasi pengambilan keputusan. Utilitas adalah alat perangkat lunak yang digunakan untuk membantu mengelola komponen komputer sistem basis data. Sebagai contoh, semua vendor DBMS utama sekarang menyediakan antarmuka pengguna grafis (GUI) untuk membantu membuat struktur basis data, mengendalikan akses basis data, dan memantau operasi basis data.
3. Orang-orang (People): Komponen ini mencakup semua pengguna sistem basis data. Berdasarkan fungsi pekerjaan utama, ada lima jenis pengguna yang dapat diidentifikasi

dalam sistem basis data: administrator sistem, administrator basis data, perancang basis data, analis sistem dan pemrogram, dan pengguna akhir. Setiap jenis pengguna, yang dijelaskan selanjutnya, menjalankan fungsi yang unik dan saling melengkapi.

a. Administrator sistem mengawasi operasi umum sistem basis data.
b. Administrator basis data, juga dikenal sebagai DBA, mengelola DBMS dan memastikan bahwa basis data berfungsi dengan baik. Peran DBA cukup penting sehingga perlu dieksplorasi secara detail terkait dengan Administrasi dan Keamanan Basis Data.
c. Perancang basis data merancang struktur basis data. Mereka, pada dasarnya, adalah arsitek basis data. Jika desain basis data buruk, bahkan programmer aplikasi terbaik dan DBA yang paling berdedikasi tidak dapat menghasilkan lingkungan basis data yang berguna. Karena organisasi berusaha untuk mengoptimalkan sumber daya data mereka, deskripsi pekerjaan perancang basis data telah berkembang untuk mencakup dimensi-dimensi baru dan tanggung jawab yang berkembang.
d. Analis sistem dan pemrogram merancang dan mengimplementasikan program aplikasi. Mereka merancang dan membuat layar input data, laporan, dan prosedur yang digunakan oleh pengguna akhir untuk mengakses dan memanipulasi data dalam basis data.
e. *End users* adalah orang-orang yang menggunakan program aplikasi untuk menjalankan operasi harian organisasi. Misalnya, karyawan penjualan, supervisor, manajer, dan direktur semuanya diklasifikasikan sebagai pengguna akhir. Pengguna akhir tingkat tinggi

menggunakan informasi yang diperoleh dari basis data untuk membuat keputusan bisnis taktis dan strategis.

4. Prosedur. Prosedur adalah instruksi dan aturan yang mengatur desain dan penggunaan sistem basis data. Prosedur adalah komponen kritis, meskipun terkadang terlupakan, dalam sistem ini. Prosedur memainkan peran penting dalam sebuah perusahaan karena mereka menerapkan standar dengan mana bisnis dilakukan dalam organisasi dan dengan pelanggan. Prosedur juga membantu memastikan bahwa perusahaan memiliki cara yang terorganisir untuk memantau dan mengaudit data yang masuk ke dalam basis data dan informasi yang dihasilkan dari data tersebut.
5. Data. Kata data mencakup kumpulan fakta yang disimpan dalam basis data. Karena data adalah bahan dasar dari mana informasi dihasilkan, menentukan data mana yang akan dimasukkan ke dalam basis data dan bagaimana cara mengorganisasi data tersebut adalah bagian penting dari pekerjaan perancang basis data.

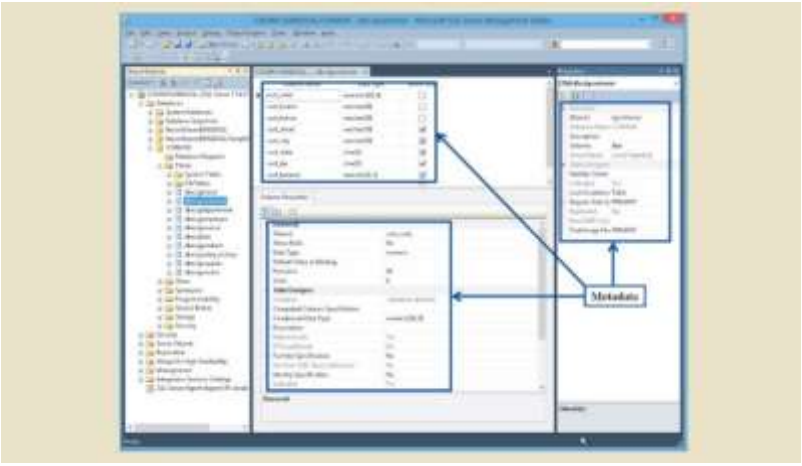
Sistem basis data menambah dimensi baru ke dalam struktur manajemen sebuah organisasi. Kompleksitas struktur manajerial ini bergantung pada ukuran organisasi, fungsi-fungsi organisasi tersebut, dan budaya perusahaan. Oleh karena itu, sistem basis data dapat dibuat dan dikelola pada tingkat kompleksitas yang berbeda dan dengan tingkat ketaatan standar yang berbeda pula. Sebagai contoh, bandingkan sistem toko convenience lokal dengan sistem klaim asuransi nasional. Sistem toko convenience mungkin dikelola oleh dua orang, perangkat keras yang digunakan mungkin hanya satu komputer, prosedur-prosedurnya mungkin sederhana, dan volume data cenderung rendah. Sistem klaim asuransi nasional kemungkinan besar memiliki setidaknya satu administrator sistem, beberapa DBA

penuh waktu, dan banyak perancang dan pemrogram; perangkat keras mungkin mencakup beberapa server di berbagai lokasi di seluruh Amerika Serikat; prosedur-prosedurnya kemungkinan banyak, kompleks, dan ketat; dan volume data cenderung tinggi.

Selain tingkat kompleksitas yang berbeda dari sistem basis data, manajer juga harus mempertimbangkan fakta penting lainnya: solusi basis data harus efektif dari segi biaya serta efektif secara taktis dan strategis. Menghasilkan solusi senilai jutaan dolar untuk masalah senilai ribuan dolar hampir tidak mencerminkan pemilihan atau desain dan manajemen sistem basis data yang baik. Terakhir, teknologi basis data yang sudah digunakan kemungkinan akan memengaruhi pemilihan sistem basis data.

3.6 Fungsi dari Sistem Manajemen Basis Data

Sebuah Sistem Manajemen Basis Data (DBMS) memiliki tanggung jawab dalam melaksanakan berbagai fungsi kunci yang bertujuan untuk menjaga integritas dan konsistensi data dalam basis data. Sebagian besar fungsi ini tidak dapat terlihat oleh pengguna akhir, dan untuk mengaksesnya, Anda perlu menggunakan DBMS (Soykan & Şimşek, 2017). Fungsi-fungsi ini melibatkan manajemen kamus data, pengelolaan penyimpanan data, transformasi dan penyajian data, pengaturan keamanan data, pengendalian akses multiuser, manajemen pencadangan dan pemulihan data, pengawasan integritas data, bahasa akses basis data, dan antarmuka komunikasi basis data. Setiap fungsi ini akan diuraikan sebagai berikut:



Gambar 3.4. Mengilustrasikan Metadata dengan Microsoft SQL Server Express
(Sumber: (Carlos Coronel, 2018))

1. *Data dictionary management.* DBMS menyimpan definisi elemen data dan hubungan mereka (metadata) dalam kamus data. Seluruh program yang mengakses data dalam basis data bekerja melalui DBMS. DBMS menggunakan kamus data untuk mencari struktur komponen data yang diperlukan dan hubungannya, sehingga menghindarkan Anda dari pengkodean hubungan yang kompleks dalam setiap program. Selain itu, semua perubahan yang dilakukan dalam struktur basis data secara otomatis dicatat dalam kamus data, sehingga Anda tidak perlu memodifikasi semua program yang mengakses struktur yang berubah. Dengan kata lain, DBMS memberikan abstraksi data, dan menghilangkan ketergantungan struktural dan data dari sistem. Sebagai contoh, Gambar 3.4 menunjukkan bagaimana Microsoft SQL Server Express menyajikan definisi data untuk tabel CUSTOMER.

2. Pengelolaan penyimpanan data. DBMS bertugas untuk menciptakan dan mengelola struktur yang rumit yang diperlukan untuk menyimpan data, yang mengurangi beban tugas Anda dalam mengartikan dan mengatur karakteristik fisik data. Sebuah DBMS modern tidak hanya menyediakan penyimpanan untuk data itu sendiri, tetapi juga untuk berbagai elemen terkait seperti formulir input data, definisi laporan, peraturan validasi data, kode prosedural, kerangka kerja untuk mengelola format video dan gambar, dan lain sebagainya. Pengelolaan penyimpanan data juga memiliki peran penting dalam peningkatan kinerja basis data. Peningkatan kinerja ini terkait dengan serangkaian tindakan yang membuat basis data berjalan dengan lebih efisien dalam hal penyimpanan dan akses yang lebih cepat.

Gambar 3.5. Mengilustrasikan Manajemen Penyimpanan Data dengan Oracle
(Sumber: (Carlos Coronel, 2018))

Meskipun pengguna melihat basis data sebagai unit penyimpanan data tunggal, DBMS sebenarnya menyimpan basis data dalam beberapa file data fisik. (Lihat Gambar 3.5) File-data semacam ini bahkan dapat disimpan pada media penyimpanan yang berbeda. Oleh karena itu, DBMS tidak harus menunggu permintaan disk satu selesai sebelum yang berikutnya dimulai. Dengan kata lain, DBMS dapat memenuhi permintaan basis data secara bersamaan.

3. *Data transformation and presentation*. DBMS mengubah data yang dimasukkan agar sesuai dengan struktur data yang diperlukan. DBMS menghilangkan keharusan Anda untuk membedakan antara format data logis dan format data fisik. Artinya, DBMS memformat data yang diambil secara fisik agar sesuai dengan harapan logis pengguna. Sebagai contoh, bayangkan sebuah basis data perusahaan multinasional. Seorang pengguna akhir di Inggris akan mengharapkan memasukkan tanggal 11 Juli 2017 sebagai "11/07/2017." Sebaliknya, tanggal yang sama akan dimasukkan di Amerika Serikat sebagai "07/11/2017." Terlepas dari format penyajian data, DBMS harus mengelola tanggal tersebut dalam format yang benar untuk setiap negara.
4. *Security management*. DBMS Manajemen keamanan. DBMS bertanggung jawab untuk membentuk sistem keamanan yang menegakkan tingkat keamanan dan kerahasiaan data. Aturan-aturan keamanan mengatur siapa saja yang memiliki akses ke basis data, elemen data apa yang dapat diakses oleh masing-masing pengguna, serta jenis operasi data (misalnya, membaca, menambah, menghapus, atau mengubah data) yang diizinkan untuk dilakukan oleh pengguna. Keamanan ini sangat krusial dalam lingkungan basis data multiuser. Sebagai tambahan, Administrasi dan

Keamanan Basis Data mengeksplorasi masalah-masalah yang berkaitan dengan keamanan data dan privasi secara lebih mendalam. Semua pengguna basis data mungkin diidentifikasi oleh DBMS melalui penggunaan nama pengguna dan kata sandi, atau melalui otentikasi biometrik seperti pemindaian sidik jari. Informasi otentikasi ini digunakan oleh DBMS untuk mengatur hak akses ke berbagai komponen basis data, termasuk kueri dan laporan.

5. *Multiuser access control.* Untuk memberikan integritas data dan konsistensi data, DBMS menggunakan algoritma-algoritma canggih untuk memastikan bahwa beberapa pengguna dapat mengakses basis data secara bersamaan tanpa mengompromikan integritasnya.
6. *Backup and recovery management*. DBMS menyediakan pencadangan dan pemulihan data untuk memastikan keamanan dan integritas data. Sistem DBMS saat ini menyediakan utilitas khusus yang memungkinkan DBA untuk melakukan prosedur pencadangan dan pemulihan rutin maupun khusus. Manajemen pemulihan berurusan dengan pemulihan basis data setelah kegagalan, seperti sektor buruk di disk atau pemadaman listrik. Kemampuan tersebut sangat penting untuk menjaga integritas basis data. Bab 16 membahas isu-isu pencadangan dan pemulihan.
7. *Data integrity management*. DBMS mempromosikan dan menegakkan aturan integritas, sehingga meminimalkan redundansi data dan memaksimalkan konsistensi data. Hubungan data yang disimpan dalam kamus data digunakan untuk menegakkan integritas data. Menjaga integritas data sangat penting dalam sistem basis data yang berorientasi pada transaksi.

8. *Database access languages and application programming interfaces.* DBMS menyediakan akses data melalui bahasa kueri. Bahasa kueri adalah bahasa nonprosedural, yang memungkinkan pengguna menentukan apa yang harus dilakukan tanpa harus menentukan bagaimana melakukannya. Bahasa Kueri Terstruktur (SQL) adalah bahasa kueri dan standar akses data yang didukung oleh sebagian besar vendor DBMS. Pengantar Bahasa Kueri Terstruktur (SQL), dan SQL Lanjutan, membahas penggunaan SQL. DBMS juga menyediakan antarmuka pemrograman aplikasi ke bahasa pemrograman prosedural seperti COBOL, C, Java, Visual Basic.NET, dan C#. Selain itu, DBMS menyediakan utilitas administrasi yang digunakan oleh DBA dan perancang basis data untuk membuat, mengimplementasikan, memantau, dan menjaga basis data.
9. *Database communication interfaces.* DBMS generasi saat ini menerima permintaan pengguna akhir melalui berbagai lingkungan jaringan yang berbeda. Sebagai contoh lainnya: Sebagai ilustrasi, DBMS mungkin memfasilitasi akses ke basis data melalui internet dengan memanfaatkan peramban web seperti Mozilla Firefox, Google Chrome, atau Microsoft Internet Explorer. Dalam konteks ini, interaksi dapat terjadi melalui berbagai metode komunikasi:
    a. Pengguna akhir dapat menghasilkan jawaban atas kueri dengan mengisi formulir layar melalui peramban web pilihan mereka.
    b. DBMS dapat secara otomatis menerbitkan laporan-laporan yang telah ditentukan sebelumnya di sebuah situs web.

c. DBMS dapat terhubung ke sistem-sistem pihak ketiga untuk mendistribusikan informasi melalui email atau aplikasi produktivitas lainnya.

3.7 Sejarah Sistem Manajemen Basis Data

Kami telah menyaksikan bahwa sebelum DBMS muncul, sistem berbasis berkas telah ada. Meskipun demikian, pendekatan basis data tidak pernah sepenuhnya menggantikan sistem berbasis berkas. Bahkan, dalam beberapa konteks, sistem berbasis berkas masih digunakan. Ada klaim bahwa DBMS memiliki akar yang dapat ditelusuri kembali ke proyek pendaratan manusia di bulan Apollo pada tahun 1960-an. Proyek ini dimulai sebagai tanggapan terhadap tantangan yang diajukan oleh Presiden Kennedy untuk mendaratkan seorang astronot di bulan sebelum akhir dekade tersebut. Saat itu, belum ada sistem yang dapat mengelola dan mengolah volume besar informasi yang dihasilkan oleh proyek ini (Connolly & Begg, 2017).

Sebagai respons terhadap kebutuhan tersebut, *North American Aviation* (NAA, sekarang Rockwell International), yang merupakan kontraktor utama proyek Apollo, mengembangkan perangkat lunak yang dikenal sebagai GUAM (*Generalized Update Access Method*). GUAM berdasarkan pada gagasan bahwa komponen-komponen kecil dapat digabungkan menjadi komponen yang lebih besar, dan seterusnya, hingga produk akhir terbentuk. Model ini, yang mirip dengan struktur pohon terbalik, juga dikenal sebagai struktur hirarkis. Pada pertengahan tahun 1960-an, IBM bergabung dengan NAA untuk mengembangkan GUAM menjadi sistem yang dikenal saat ini sebagai IMS (*Information Management System*). Awalnya, IBM membatasi IMS untuk mengelola data dalam struktur hirarkis untuk memenuhi kebutuhan pasar yang ada, terutama penggunaan perangkat penyimpanan serial seperti pita magnetik. Pembatasan ini kemudian dihapus. Meskipun IMS adalah salah satu DBMS komersial pertama yang ada, ia masih

dianggap sebagai sistem utama berbasis hirarki yang digunakan oleh banyak pusat komputasi besar.

Di pertengahan tahun 1960-an, terjadi perkembangan yang signifikan, yaitu munculnya IDS (atau *Integrated Data Store*) yang dikembangkan oleh General Electric. Inisiatif ini diprakarsai oleh salah satu perintis awal dalam sistem basis data, yaitu Charles Bachmann. Kemunculan IDS membawa perkembangan baru dalam bentuk sistem basis data yang disebut sebagai DBMS jaringan, yang kemudian memiliki dampak besar pada perkembangan sistem informasi generasi tersebut. Sistem basis data jaringan diciptakan sebagian besar untuk mengatasi tantangan dalam merepresentasikan hubungan data yang lebih kompleks daripada yang bisa dijelaskan dengan menggunakan struktur hirarkis. Selain itu, salah satu tujuan utama adalah untuk menetapkan standar dalam domain basis data. Untuk memfasilitasi pembentukan standar ini, Konferensi Bahasa Sistem Data (CODASYL), yang terdiri dari perwakilan dari pemerintah AS dan sektor bisnis dan perdagangan, membentuk sebuah Kelompok Tugas Pengolahan Daftar pada tahun 1965. Kelompok ini kemudian berganti nama menjadi Kelompok Tugas Basis Data (DBTG) pada tahun 1967. Misi DBTG adalah merumuskan spesifikasi standar yang akan mengarah pada pembuatan basis data dan manipulasi data yang lebih terstandarisasi. Draf laporan pertama diterbitkan pada tahun 1969, dan laporan definitif pertama kali diterbitkan pada tahun 1971. Proposal dari DBTG mengidentifikasi tiga komponen utama:

1. skema jaringan—organisasi logis seluruh basis data seperti yang dilihat oleh DBA—yang mencakup definisi nama basis data, jenis setiap catatan, dan komponen dari setiap jenis catatan;
2. subskema—bagian dari basis data seperti yang dilihat oleh pengguna atau program aplikasi;
3. bahasa manajemen data untuk mendefinisikan karakteristik data dan struktur data, dan memanipulasi data.

Untuk standardisasi, DBTG menentukan tiga bahasa yang berbeda:

1. skema DDL, yang memungkinkan DBA untuk mendefinisikan skema;
2. subskema DDL, yang memungkinkan program aplikasi untuk mendefinisikan bagian basis data yang mereka butuhkan;
3. DML, untuk memanipulasi data.

Meskipun laporan tersebut tidak secara resmi diadopsi oleh *American National Standards Institute* (ANSI), beberapa sistem kemudian dikembangkan sesuai dengan proposal DBTG. Sistem-sistem ini saat ini dikenal sebagai sistem CODASYL atau DBTG. Pendekatan CODASYL dan pendekatan hirarkis yang digunakan mewakili generasi awal dari DBMS.

Pada tahun 1970, E. F. Codd dari IBM Research Laboratory menerbitkan makalah yang sangat berpengaruh tentang model data relasional ("A relational model of data for large, shared data banks," Codd, 1970). Makalah ini muncul pada waktu yang tepat dan mengatasi kelemahan dari pendekatan sebelumnya. Setelah itu, banyak DBMS relasional eksperimental diimplementasikan, dengan produk komersial pertama muncul pada akhir tahun 1970-an dan awal tahun 1980-an. Yang perlu dicatat adalah proyek System R yang dikembangkan di Laboratorium Riset IBM di San José, California, pada akhir tahun 1970 (Astrahan et al., 1976). Proyek ini bertujuan untuk membuktikan manfaat dari model relasional dengan memberikan implementasi struktur data dan operasi yang sesuai, dan proyek ini menghasilkan dua kemajuan utama:

1. Berkembangnya bahasa kueri terstruktur yang dikenal sebagai SQL, yang kemudian menjadi bahasa standar untuk DBMS relasional.

2. Munculnya berbagai produk komersial DBMS relasional selama dekade 1980-an, termasuk DB2 dan SQL/DS dari IBM, serta Oracle dari Oracle Corporation.

Saat Sekarang ada beberapa ratus DBMS relasional yang digunakan dalam lingkungan mainframe dan PC, walaupun banyak di antaranya melampaui konsep dasar model relasional. Contoh DBMS relasional multi-user meliputi MySQL dari Oracle Corporation, Ingres dari Actian Corporation, SQL Server dari Microsoft, dan Informix dari IBM. Di sisi lain, contoh DBMS relasional berbasis PC mencakup Office Access dan Visual FoxPro dari Microsoft, Interbase dari Embarcadero Technologies, serta R:Base dari R:Base Technologies. DBMS relasional sering disebut sebagai generasi kedua dari DBMS.

Meskipun model relasional memiliki banyak keunggulan, seperti yang telah disebutkan sebelumnya, namun memiliki beberapa keterbatasan dalam hal kemampuan pemodelan. Oleh karena itu, sejak saat itu, telah ada banyak penelitian yang berusaha mengatasi masalah tersebut. Pada tahun 1976, Chen memperkenalkan model Entity-Relationship, yang saat ini menjadi teknik yang secara luas diterima untuk desain basis data dan menjadi dasar untuk metodologi yang dijelaskan dalam Bab 16 dan 17 buku ini. Selanjutnya, pada tahun 1979, Codd sendiri berusaha mengatasi beberapa kekurangan dalam pekerjaannya yang asli dengan merilis versi perluasan dari model relasional yang dikenal sebagai RM/T (1979) dan kemudian RM/V2 (1990). Upaya untuk menyediakan model data yang lebih mendekati "dunia nyata" sering kali dikelompokkan sebagai pemodelan data semantik.

Menyikapi meningkatnya kompleksitas aplikasi basis data, muncul dua jenis sistem "baru": DBMS berorientasi objek (OODBMS) dan DBMS objek-relasional (ORDBMS). Namun, berbeda dari model-model sebelumnya, struktur sebenarnya dari model-model ini menjadi kurang jelas. Dekade 1990-an juga

melihat munculnya Internet, arsitektur tiga lapisan klien-server, dan permintaan untuk mengintegrasikan basis data perusahaan dengan aplikasi web. Akhir tahun 1990-an melihat perkembangan XML (eXtensible Markup Language), yang memiliki dampak besar pada banyak aspek IT, termasuk integrasi basis data, antarmuka grafis, sistem tertanam, sistem terdistribusi, dan sistem basis data.

DBMS khusus juga telah dibuat, seperti data warehouses, yang dapat menyimpan data yang diambil dari beberapa sumber data, mungkin dikelola oleh unit-unit operasi yang berbeda dari sebuah organisasi. Sistem-sistem ini menyediakan fasilitas analisis data komprehensif untuk memungkinkan pengambilan keputusan strategis berdasarkan, misalnya, tren historis. Semua vendor DBMS utama menyediakan solusi data warehouses. Contoh lainnya adalah sistem perencanaan sumber daya perusahaan (ERP), lapisan aplikasi yang dibangun di atas DBMS yang mengintegrasikan semua fungsi bisnis sebuah organisasi, seperti manufaktur, penjualan, keuangan, pemasaran, pengiriman, penagihan, dan sumber daya manusia. Sistem ERP populer adalah SAP R/3 dari SAP dan PeopleSoft dari Oracle.

DAFTAR PUSTAKA

Amiraslani, F. 2021. Analysis of quality of life across Tehran districts based on designated indicators and relational database management system. *Urban Governance*, *1*(2), 107–114. https://doi.org/https://doi.org/10.1016/j.ugj.2021.09.003

Carlos Coronel, S. M. 2018. *Database Systems: Design, Implementation, & Management* (13th ed., Vol. 13). Cengage Learning.

Carral, L., Alvarez-Feal, C., Jesús Rodríguez-Guerreiro, M., Vargas, A., Arean, N., & Carballo, R. 2019. Methodology for positioning a group of green artificial reef based on a database management system, applied in the estuary of Ares-Betanzos (Nw Iberian Peninsula). *Journal of Cleaner Production*, *233*, 1047–1060. https://doi.org/https://doi.org/10.1016/j.jclepro.2019.06.092

Connolly, T., & Begg, C. 2017. *Database systems. A practical approach to design, implementation and management*.

Curé, O., & Blin, G. (Eds.). 2015. Chapter Two - Database Management Systems. In *RDF Database Systems* (pp. 9–40). Morgan Kaufmann. https://doi.org/https://doi.org/10.1016/B978-0-12-799957-9.00002-X

Lee, H., Chapiro, J., Schernthaner, R., Duran, R., Wang, Z., Gorodetski, B., Geschwind, J.-F., & Lin, M. 2015. How I Do It: A Practical Database Management System to Assist Clinical Research Teams with Data Collection, Organization, and Reporting. *Academic Radiology*, *22*(4), 527–533. https://doi.org/https://doi.org/10.1016/j.acra.2014.12.002

Soykan, F., & Şimşek, B. 2017. Examining studies on learning management systems in SSCI database: A content analysis study. *Procedia Computer Science*, *120*, 871–876. https://doi.org/https://doi.org/10.1016/j.procs.2017.11.320

Tray, E., Leadbetter, A., Meaney, W., Conway, A., Kelly, C., Maoiléidigh, N. Ó., de Eyto, E., Moran, S., & Brophy, D. 2020. An open-source database model and collections management system for fish scale and otolith archives. *Ecological Informatics*, *59*, 101115.https://doi.org/https://doi.org/10.1016/j.ecoinf.2020.101115

Wang, Y., Liu, J., He, X., & Wang, B. 2018. Design and realization of rock salt gas storage database management system based on SQL Server. *Petroleum*, *4*(4), 466–472. https://doi.org/ https://doi.org/10.1016/j.petlm.2017.10.001

Xu, X. 2021. RETRACTED: Material database management system based on heterogeneous multi-processor and computer embedded system. *Microprocessors and Microsystems*, *82*, 103926.https://doi.org/https://doi.org/10.1016/j.micpro.2021.103926

Zhou, H. 2012. Design of Student Information Management Database Application System for Office and Departmental Target Responsibility System. *Physics Procedia*, *25*, 1660–1665. https://doi.org/https://doi.org/10.1016/j.phpro.2012.03.291

BAB 4
DESAIN BASIS DATA

Oleh Erick Fernando

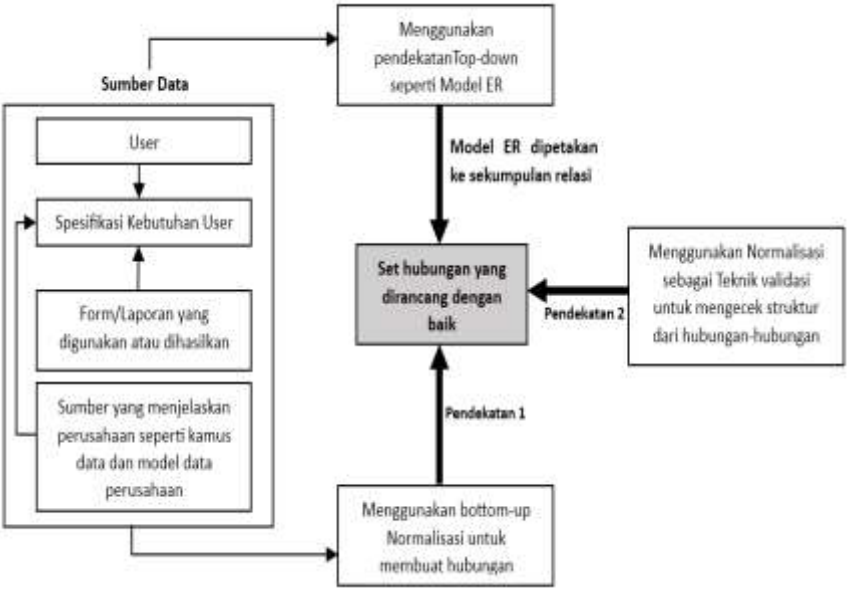
4.1 Pendahuluan

Desain basis data mengacu pada proses yang membantu pembuatan, pengembangan, dan implementasi sistem basis data perusahaan serta pemeliharaannya yang berkelanjutan. Proses perancangan basis data memiliki potensi untuk mempermudah pengelolaan basis data, meningkatkan konsistensi data, dan mengurangi biaya yang terkait dengan penyimpanan data (Coronel and Morris, 2023). Desain database dapat menentukan potongan data mana yang harus dikaitkan satu sama lain dan data mana yang harus disimpan. Pembuatan model desain konseptual, Logikal, dan fisik dari sistem basis data yang diusulkan adalah tujuan utama dari proses desain basis data. Agar dapat digunakan dengan Database Management System (DBMS), harus dimodifikasi terlebih dahulu (Conger, 2012; Connolly and Carolyn Begg, 2015).

Ada dua metodologi utama untuk desain database, yaitu pendekatan "*bottom-up*" dan "*top-down*". Teknik bottom-up dimulai dengan memeriksa kualitas dasar (seperti sifat entitas dan hubungan) dan kemudian mengaturnya ke dalam hubungan yang menggambarkan jenis entitas dan asosiasi antar entitas. Hal ini dicapai melalui investigasi hubungan antar atribut. Metode ini dianggap tepat untuk arsitektur basis data dasar yang dicirikan oleh jumlah properti yang relatif terbatas. (Connolly and Carolyn Begg, 2015).

Selain itu, metode *top-down* berkaitan dengan konseptualisasi model *Entity-Relationship* (ER), dimana langkah awal melibatkan identifikasi entitas dan hubungan yang ada di

antara mereka, sehingga memenuhi kebutuhan organisasi(Conger, 2012; Gupta and Aditya Mittal, 2017). Teknik top-down melibatkan serangkaian modifikasi yang diterapkan secara berurutan untuk menemukan entitas tingkat rendah, hubungan, dan properti terkait. Langkah awal melibatkan pembuatan model data yang mencakup banyak item tingkat tinggi dan hubungannya yang sesuai.



Gambar 4.1. Skema Pendekatan Desain Basis Data

4.2 Langkah-langkah dalam desain basis data

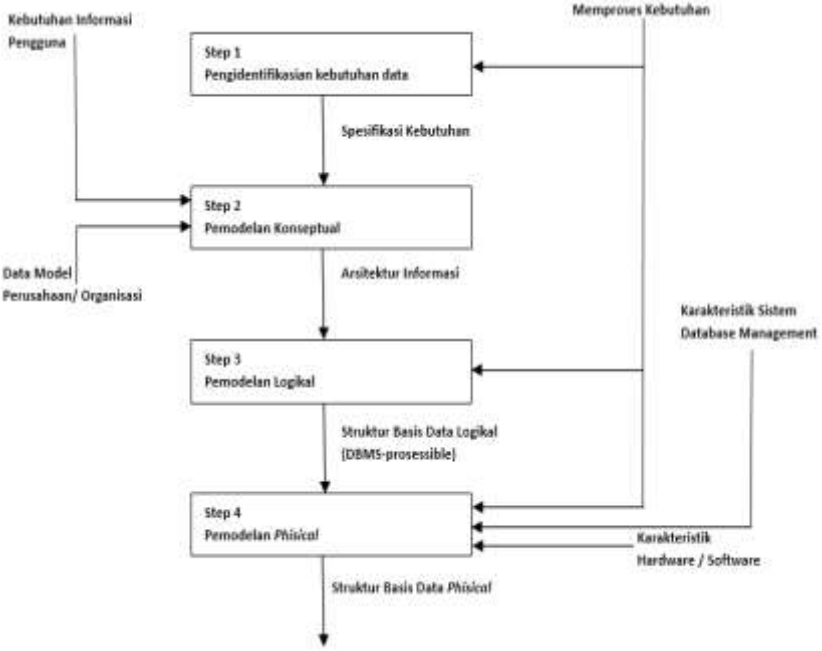
Desain basis data sesuai dengan *database development life cycle*. Langkah-langkah desain basis data sebagai berikut:

1. Pengidentifikasian kebutuhan data
2. Pemodelan Konseptual
3. Pemodelan Logikal
4. Pemodelan *Physical*

Model konseptual yang dapat digunakan untuk membangun representasi konseptual dari database. Model ini harus mencakup identifikasi item, hubungan, dan atribut yang signifikan. Praktik membangun model data yang digunakan dalam suatu organisasi sambil mengabaikan setiap dan semua pertimbangan fisik.

Dalam model logikal, penekanan ditempatkan pada persyaratan data dan data yang akan disimpan, bukan pada pertimbangan fisik. Tidak ada perbedaan mengenai cara penyimpanan data atau lokasi penyimpanan fisiknya.

Pemodelan *Physical*. Desain basis data logikal diekspor dari basis data dan kemudian diterjemahkan ke media fisik melalui berbagai sumber daya perangkat keras dan sistem perangkat lunak, seperti sistem manajemen basis data (DBMS). Ini adalah proses yang membentuk model desain data fisik.



Gambar 4.2. Langkah – langkah desain basis data

4.3 Pengidentifikasian kebutuhan data

Perencanaan dan definisi sistem memiliki dua tahap siklus hidup pengembangan basis data yang bertanggung jawab untuk mengidentifikasi kebutuhan data. Selama fase perencanaan, strategi organisasi untuk sistem informasinya serta persyaratan yang relevan dipertimbangkan. Tahap ini memerlukan sejumlah tindakan yang sangat signifikan.

Pertama dan terpenting, pemahaman yang komprehensif tentang tujuan perusahaan. Langkah selanjutnya adalah menentukan entitas utama yang perlu direfleksikan dalam database, serta atribut yang perlu disimpan untuk masing-masing entitas tersebut. Setelah itu, relasi yang terjalin antar entitas perlu dievaluasi juga, agar struktur database dapat dikembangkan secara efektif. Perkiraan jumlah data saat ini serta proyeksi perluasannya juga merupakan faktor penting untuk diperhitungkan selama prosedur ini.

Langkah selanjutnya adalah menentukan persyaratan integrasi yang harus dipenuhi untuk menjamin bahwa database akan kompatibel dengan sistem lain yang mungkin sudah ada. Ada aspek keamanan data lebih lanjut yang perlu ditentukan, seperti teknik melindungi data sensitif dengan prosedur enkripsi, otentikasi, dan otorisasi yang diperlukan.

Pentingnya mempertimbangkan kinerja basis data tidak boleh diabaikan, terutama ketika harus menyesuaikan basis data untuk mengakomodasi analisis, pelaporan, atau transaksi real-time. Selain itu, sangat penting untuk mempertimbangkan banyak aspek strategi manajemen data, termasuk tindakan pemeliharaan, pemulihan bencana, dan keamanan data, untuk menjaga integritas dan aksesibilitas data.

Untuk memahami dan memenuhi semua persyaratan bisnis secara efektif, sangat penting untuk melibatkan pemangku kepentingan terkait, termasuk pengguna bisnis, analis, pengembang, dan administrator basis data, dalam proses

menetapkan persyaratan data. Oleh karena itu, prosedur membedakan persyaratan data memfasilitasi pembangunan database yang sejalan dengan tuntutan bisnis dan menghasilkan keuntungan yang optimal bagi organisasi.

4.4 Pemodelan Konseptual

Tahap awal desain database melibatkan proses pemodelan konseptual. Desain basis data konseptual melibatkan pembangunan satu atau lebih model data konseptual berdasarkan persyaratan data organisasi. Pada tahap ini, konstruksi model data dilakukan dengan memanfaatkan informasi yang telah ditetapkan dalam spesifikasi kebutuhan pengguna. Desain basis data konseptual tidak dipengaruhi oleh spesifikasi implementasi, seperti pilihan perangkat lunak DBMS target, program aplikasi, bahasa pemrograman, platform perangkat keras, atau faktor fisik lainnya. Selama proses pengembangan model data konseptual, penting untuk menguji dan memvalidasi model terhadap kebutuhan pengguna.

Model data konseptual terdiri dari:

1. Tipe entitas;
2. Jenis hubungan (*relationship*);
3. Atribut dan domain atribut;

Langkah langkah tentang cara desain basis data konseptual.

4.4.1 Langkah Identifikasi tipe entitas

Langkah mengidentifikasi menentukan tipe entitas dengan cara memeriksa spesifikasi kebutuhan pengguna. Dari spesifikasi ini, mengidentifikasi kata benda atau frase kata benda yang disebutkan (misalnya, nomor staf, nama staf, nomor properti, alamat properti, sewa, jumlah kamar). Selain itu penentuan entitas pada objek utama, seperti orang, tempat, atau konsep minat. Indentifikasi tipe entitas dengan memberikan nama entintas yang

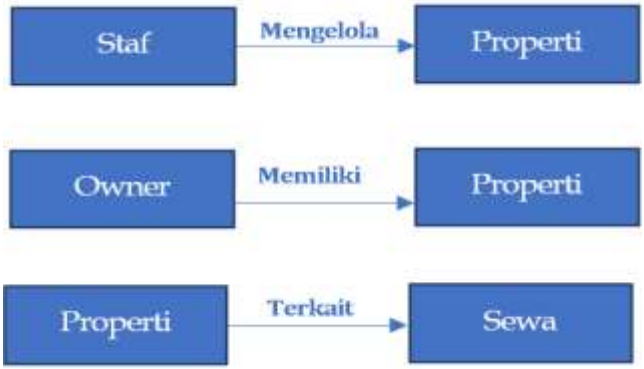
bermakna dan jelas. Setiap entitas harus memiliki nama yang berbeda.

Langkah ini bersifat subyektif, setiap desainer berbeda dapat menghasilkan interpretasi yang berbeda, tetapi bisa sama-sama valid. Jadi, langkah iini sangat bergantung pada penilaian dan pengalaman.

4.4.2 Langkah Identifikasi tipe hubungan

Langkah mengidentifikasi hubungan entitas adalah mencari kata benda dalam spesifikasi kebutuhan pengguna. Hubungan ditunjukkan dengan kata kerja atau ekspresi verbal. Misalnya:

1. Staf **Mengelola** Properti
2. Owner **Memiliki** Properti
3. Property **Terkait** Dengan Sewa



Gambar 4.3. Hubungan Relasi antar entitas

4.4.3 Langkah Identifikasi dan kaitkan atribut dengan tipe entitas atau relasi

Langkah ini melibatkan pembentukan asosiasi antara atribut dan entitas yang sesuai atau tipe relasi. Langkah ini melibatkan identifikasi relasi entitas dalam bentuk karakteristik sederhana atau gabungan, serta atribut bernilai tunggal atau multi,

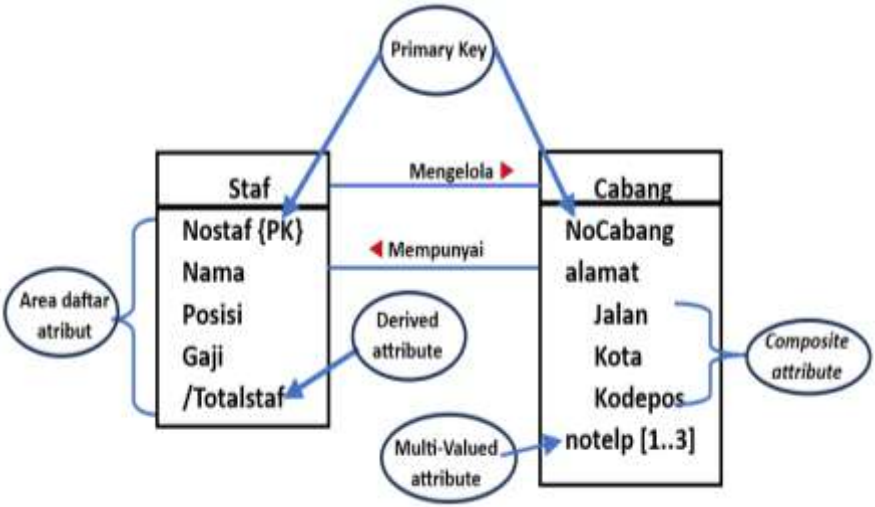
dan atribut turunan. Pada tahap ini, sangat penting untuk menjamin bahwa hubungan tersebut definitif dan akurat sesuai dengan kebutuhan spesifik seseorang.

Simple attribut adalah atribut yang tidak memiliki kemampuan untuk dibagi lagi menjadi atribut yang lebih kecil. Sebagai ilustrasi dapat dilihat variabel jumlah pegawai. Karakteristik ini umumnya digunakan untuk menetapkan identifikasi yang berbeda dan eksklusif bagi orang-orang dalam struktur organisasi.

Composite attributes mengacu pada kualitas yang memiliki karakteristik dapat dibagi menjadi sub-atribut yang lebih kecil, yang masing-masing memiliki makna tersendiri. Pada objek staff, terdapat atribut yang disebut nama_staf yang dapat didekomposisi menjadi sub-atribut yang lebih terperinci, yaitu nama_pertama dan nama_belakang.

Single/multi-valued attributes. multi-valued attributes. Di bidang manajemen basis data, atribut dapat diklasifikasikan sebagai bernilai tunggal atau bernilai banyak. Atribut multi-nilai mengacu pada atribut yang memiliki kemampuan menampung beberapa nilai yang terkait dengan kunci entitas tertentu. Sebagai gambaran, seorang staf bisa saja memiliki banyak nomor telepon. Oleh karena itu, properti yang dimaksud memiliki sifat multinilai. Di sisi lain, atribut nilai tunggal mengacu pada karakteristik yang memiliki nilai tunggal dan spesifik, sesuai dengan kebutuhan organisasi.

Derived attributes **merupakan atribut yang nilainya diturunkan dari atribut lain yang sudah ada dalam model. Contohnya, atribut tanggal lahir maka umur dapat diturunkan.**



Gambar 4.4. Representasi diagram entitas serta jenis atributnya

4.4.4 Langkah Tentukan domain atribut

Langkah ini menentukan domain untuk atribut dalam data konseptual model. Domain merupakan kumpulan nilai dari satu atau lebih atribut. Domain dapat didefinisikan:

1. Domain atribut nomor staf yang valid (staffNo) sebagai string dengan panjang variabel lima karakter, dengan dua karakter pertama sebagai huruf dan tiga karakter berikutnya sebagai digit dalam rentang 1–999;
2. Domain atribut dengan string karakter tunggal, contohmya nilai "M" atau "F." pada atribut jenis kelamin.
3. Model data yang dikembangkan sepenuhnya menentukan domain untuk setiap atribut dan mencakup kumpulan nilai yang diperbolehkan untuk atribut dan ukuran dan format atribut.

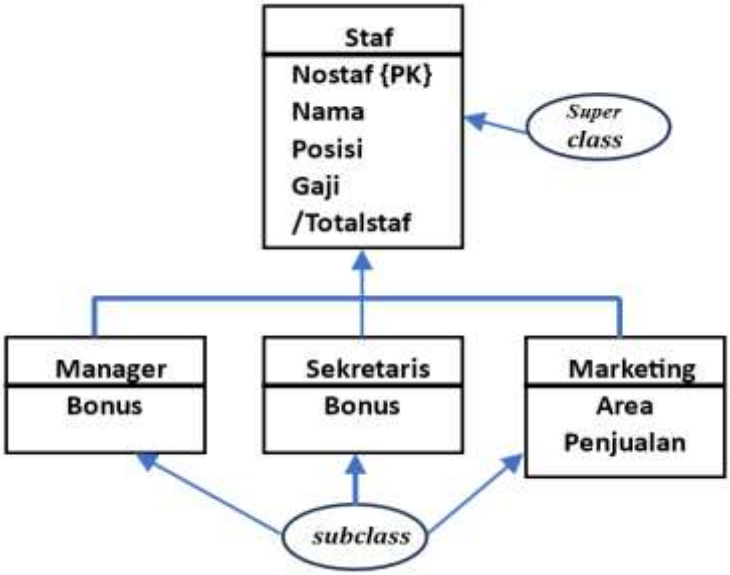
4.4.5 Langkah Tentukan atribut kandidat, primer, dan kunci alternatif

Langkah ini untuk mengidentifikasi kunci kandidat untuk setiap tipe entitas. Kunci kandidat dapat lebih dari satu kunci kandidat, dapat berupa kunci utama dan kunci alternatif. Cara memilih kunci utama antara lain :

1. Kunci kandidat yang nilainya paling kecil kemungkinannya diubah;
2. Kunci kandidat dengan karakter paling sedikit (untuk yang memiliki atribut tekstual);
3. Kunci kandidat dengan nilai maksimum terkecil (untuk yang memiliki atribut numerik);
4. Kunci kandidat yang paling mudah digunakan dari sudut pandang pengguna.

4.4.6 Langkah Pertimbangkan penggunaan konsep pemodelan yang disempurnakan (langkah opsional)

Pada tahap ini, digunakan konsep pemodelan yang disempurnakan seperti komposisi, agregasi, spesialisasi, dan generalisasi untuk mengembangkan model ER. Pendekatan spesialisasi mendefinisikan entitas dalam satu atau lebih *subclass* dari entitas *superclass*, sedangkan pendekatan generalisasi mendefinisikan entitas *superclass* yang menggeneralisasi dengan mengidentifikasi karakteristik yang sama. Pendekatan agregasi menunjukkan hubungan antara jenis entitas yang memiliki kepemilikan yang kuat dan masa hidup kebetulan antara "keseluruhan" dan "bagian". Pendekatan komposisi, atau jenis agregasi khusus, menunjukkan hubungan antara jenis entitas yang memiliki "memiliki" atau "adalah-bagian-dari" atau "keseluruhan" dan "bagian".



Gambar 4.5. Representasi Entitas *Subclass* dan *Superclass*

4.4.7 Langkah Periksa model untuk redundansi

Pada langkah ini, memeriksa atau mengidentifikasi redundansi dalam model data konseptual dan bahkan menghilangkan redundansi yang ada. Langkah ini terdiri dari tiga tugas:

1. Memeriksa kembali hubungan satu-ke-satu (1:1);
2. Menghilangkan hubungan yang berlebihan; dan
3. Mempertimbangkan dimensi waktu.

4.4.8 Langkah Validasi model data konseptual terhadap transaksi pengguna

Pada langkah ini, model data konseptual telah dibuat untuk mewakili kebutuhan data perusahaan. Langkah ini bertujuan untuk memastikan bahwa model mendukung transaksi yang diperlukan. Kami berupaya melakukan operasi secara manual dengan

menggunakan model tersebut. Jika kami dapat menyelesaikan semua transaksi dengan cara ini, kami telah memeriksa bahwa model data konseptual mendukung transaksi yang diperlukan. Tetapi jika kita tidak dapat melakukan transaksi secara manual, ada masalah dengan model data yang harus diselesaikan. Dalam situasi seperti ini, kami mungkin telah menghilangkan entitas, hubungan, atau fitur tertentu dari model data.

Untuk memastikan bahwa model data konseptual mendukung transaksi yang diperlukan, memiliki dua opsi:

1. Menjelaskan transaksi;
2. Menggunakan jalur transaksi.

4.4.9 Langkah Tinjau model data konseptual dengan pengguna

Langkah ini berfungsi untuk melakukan peninjauan terhadap model data konseptual dengan memperhatikan kesesuaian kebutuhan pengguna dan memastikan bahwa model tersebut merupakan representasi data yang "benar" dan memenuhi kebutuhan bisnis serta pengguna akhir. Proses peninjauan melibatkan partisipasi pengguna untuk mengurangi risiko kesalahan atau ketidakcocokan pada tahap selanjutnya dalam pengembangan sistem informasi.

4.5 Pemodelan Logikal

Fase kedua dalam desain basis data logikal adalah pemodelan logika, atau pembuatan model data logikal untuk perusahaan. Ini dilakukan dengan menggunakan DBMS tertentu dan mempertimbangkan basis data lain secara fisik. Fase ini menghasilkan pembuatan model data logikal untuk komponen yang dibutuhkan sistem suatu organisasi untuk dimodelkan.

Pada tahap ini, model data konseptual yang dibuat pada tahap sebelumnya diperbaiki untuk dapat dipetakan ke dalam model data logikal. Model data logikal didasarkan pada model data

target untuk basis data, seperti model data relasional. Model data logikal berfungsi sebagai sumber informasi untuk tahap berikutnya, yaitu desain basis data fisik, yang menyediak Selama tahap pemeliharaan operasional dari siklus hidup pengembangan sistem basis data, model logikal juga sangat penting. Model data yang baik dan terus diperbarui memungkinkan basis data menggambarkan data atau program aplikasi yang berubah secara akurat dan efektif. langkah-langkah metodologi yang digunakan untuk mendesain basis data logikal untuk model relasional.

4.5.1 Langkah Turunkan relasi untuk model data logikal

Proses ini memulai dengan mengidentifikasi hubungan dalam model data logikal untuk mewakili entitas, hubungan, dan atribut yang telah diidentifikasi. Selanjutnya, ia menurunkan hubungan dalam model data logikal untuk mewakili entitas, hubungan, dan atribut yang telah diidentifikasi. Nama relasi pertama kali disebutkan dalam langkah ini. Kemudian, atribut sederhana relasi disebutkan dan diapit oleh tanda kurung. Setelah itu, cari kunci utama, alternatif, dan/atau kunci lainnya dari relasi tersebut. Setelah menemukan semua kunci, berikan relasi yang mengandung kunci primer yang direferensikan. Selain itu, atribut turunan apa pun disertakan, serta metode yang digunakan untuk menghitung masing-masing atribut.

Mekanisme kunci utama atau kunci luar menunjukkan hubungan antara suatu entitas dan entitas lain. Pertama-tama, Anda harus menentukan entitas "parent" dan "child" dalam hubungan tersebut sebelum memutuskan di mana akan menempatkan atribut penting.

Langkah ini menjelaskan bagaimana hubungan diturunkan untuk struktur berikut yang mungkin terjadi dalam model data konseptual:

1. Jenis entitas yang kuat;
2. Jenis entitas yang lemah;
3. Jenis hubungan biner satu ke banyak (1:*);

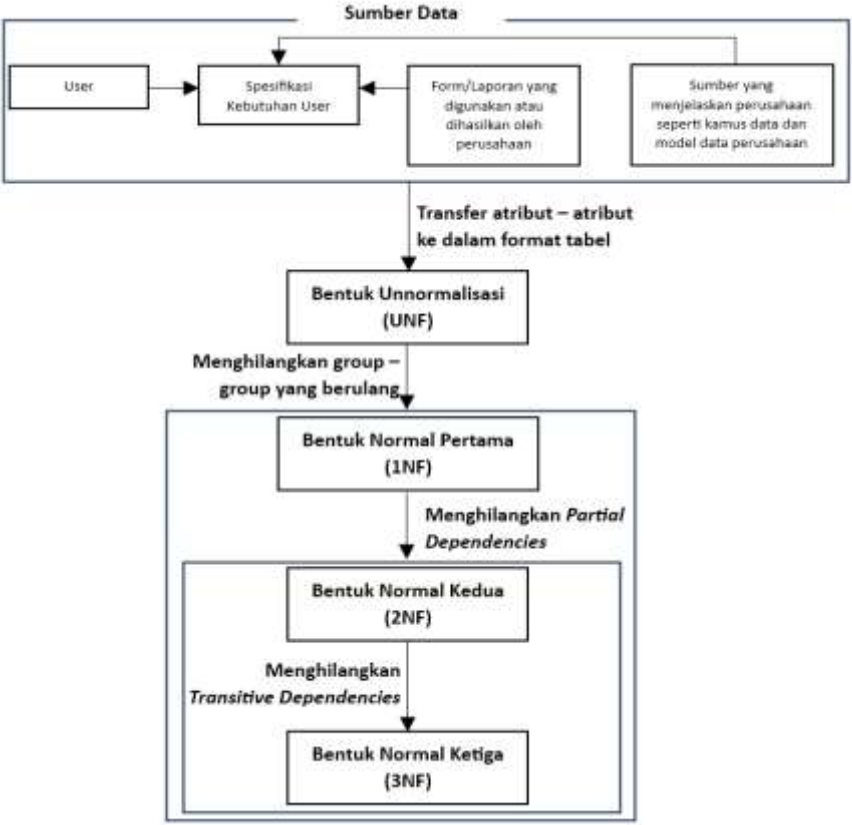
4. Jenis hubungan biner satu ke satu (1:*);
5. Jenis hubungan biner satu ke satu (1:*) rekursif;
6. Jenis hubungan super- dan subkelas;
7. Jenis hubungan biner banyak ke banyak (*:*);
8. Jenis hubungan kompleks; dan (9) sifat yang memiliki banyak nilai.

4.5.2 Langkah validasi relasi menggunakan normalisasi

Normalisasi adalah metode untuk membuat sekumpulan relasi antar tabel yang sangat penting untuk kebutuhan data perusahaan. Manfaat menggunakan basis data dengan kumpulan relasi yang tepat adalah bahwa akan lebih mudah bagi pengguna untuk mengakses, menyimpan, dan mengambil data.

Ciri-ciri himpunan relasi yang sesuai antara lain sebagai berikut:

1. Jumlah atribut yang diperlukan untuk memenuhi kebutuhan data perusahaan;
2. Atribut yang memiliki hubungan logis yang dekat ditemukan dalam hubungan yang sama; dan
3. Minimal redundansi, dengan pengecualian penting atribut yang membentuk semua atau sebagian kunci luar untuk menggabungkan hubungan terkait.
4. Proses Normalisasi mempunyai dua pendekatan utama
5. Pendekatan pertama, normalisasi dapat dilakukan digunakan sebagai teknik desain basis data mandiri bottom-up, dan
6. Dalam pendekatan kedua, normalisasi dapat digunakan sebagai metode validasi untuk memeriksa struktur relasi; ini dapat dibuat dengan menggunakan metode top-down seperti pemodelan ER.
7. Proses normalisasi membutuhkan beberapa tahap untuk dapat diimplementasikan.



Gambar 4.6. Skema Normalisasi

Tahap-tahap proses normalisasi terdiri dari :

1. Tahap bentuk tidak (un) normal (UNF)
   Bentuk tidak normal (UNF) adalah bentuk normalisasi yang dimana tabel -tabel masih memiliki data yang masih berulang. Pada bentuk tidak normal maka banyak data yang berada pada grup berulang. Grup ini mempunyai atribut pada tabel yang terdapat banyak nilai untuk satu kemunculan atribut kunci yang dinominasikan untuk tabel tersebut.

2. Bentuk normal pertama (1NF)

   Pada bentuk normal 1NF, semua data yang berulang dihilangkan sehingga hanya mempunyai satu field yang tidak akan berulang atau redudan dan tiap field mempunyai satu nilai.

   Ada dua pendekatan untuk menghapus grup berulang dari tabel yang tidak dinormalisasi:

   Mengisi kekosongan dengan menduplikasi data yang tidak berulang, jika diperlukan Metode ini dikenal sebagai "meratakan" tabel.

   Menggunakan relasi yang berbeda untuk data berulang dan salinan atribut kunci asli. Tabel yang tidak dinormalisasi kadang-kadang mengandung lebih dari satu kelompok berulang; dalam hal ini, metode ini diterapkan berulang kali hingga tidak ada lagi kelompok berulang.
3. Bentuk normal kedua (2NF)

   Pada bentuk normalisasi 2NF, field yang bukan kunci tergantung pada kunci utama secara fungsi. Bentuk normal kedua berlaku untuk relasi dengan kunci utama yang terdiri dari dua atribut atau lebih.

   Relasi dengan kunci utama dengan atribut tunggal setidaknya secara otomatis 2NF, tetapi relasi yang tidak ada dalam 2NF dapat mengalami anomali pembaruan. Misalnya, jika hanya satu tuples dalam relasi yang diperbarui tetapi dua tuples lainnya tidak, ini menyebabkan inkonsistensi data.

   Normalisasi relasi 1NF ke 2NF melibatkan penghilangan dependensi parsial. Jika ada ketergantungan parsial, akan menghapus atribut yang bergantung sebagian dari relasi dengan menempatkannya dalam relasi baru bersama dengan salinan determinannya.

4. Bentuk normal ketiga (3NF)
   Pada bentuk 3NF, bentuk normalisasi dianggap bergantung secara transitive pada primary key jika tidak ada field dalam bentuk normalisasi yang bukan kunci utama.
   Untuk mencapai bentuk normal ketiga (3NF) dari bentuk normal kedua (2NF), ketergantungan transitif perlu dihilangkan. Dalam situasi yang melibatkan *Transitive dependencies*, atribut *Transitive dependencies* perlu dimasukkan ke dalam relasi baru, bersama dengan duplikat determinan, untuk menghilangkan ketergantungan.
   Transitive dependencies merupakan atribut *non-primary-key* yang bergantung secara transitif pada *primary-key*.
5. Bentuk BCNF (*Boyce-Codd Normal Form*)
   Normalisasi pada bentuk ini dilakukan hanya jika dan hanya jika setiap determinant adalah kunci kandidat. BCNF didasarkan pada dependensi fungsional, yang memperhitungkan semua kunci kandidat dalam suatu relasi. Untuk menguji apakah suatu relasi termasuk dalam BCNF, langkahnya mengidentifikasi semua determinan dan memastikan bahwa semua determinan tersebut adalah kunci kandidat. Menurut proses ini, determinan adalah atribut, atau kumpulan atribut, di mana beberapa atribut bergantung sepenuhnya pada fungsi lainnya.
   Potensi pelanggaran BCNF dapat terjadi pada saat:
   Relasi berisi dua (atau lebih) kunci kandidat gabungan atau Kunci kandidat tumpang tindih, yaitu memiliki setidaknya satu atribut yang sama.

4.5.3 Langkah Validasi relasi terhadap transaksi pengguna

Langkah ini memvalidasi model data logikal untuk memastikan bahwa itu mendukung transaksi yang diperlukan sesuai dengan kebutuhan pengguna. Pada langkah ini, memeriksa hubungan yang dibuat pada langkah sebelumnya harus

mendukung hubungan dan memastikan bahwa tidak ada kesalahan saat membuat hubungan. Kami berupaya melakukan operasi secara manual dengan menggunakan relasi, diagram ER, kamus data, dan kunci utama/kunci luar yang ditampilkan dalam relasi. Proses memvalidasi model data logikal untuk transaksi tersebut juga selesai setelah semua indentifikasi selesai.

4.5.4 Langkah Periksa kendala integritas

Langkah ini untuk melindungi basis data dari informasi yang tidak lengkap, tidak akurat, atau tidak konsisten, digunakan pengaturan integritas. Pada titik ini, satu-satunya perhatian yang diberikan adalah desain tingkat tinggi, yang berarti menentukan *Integrity constraints* yang diperlukan, terlepas dari metode yang dapat digunakan untuk mencapainya. Gambaran "sebenarnya" dari kebutuhan data perusahaan dapat ditemukan dalam model data logikal, yang mencakup semua *Integrity constraints* penting. Jenis penghalang integritas berikut dipertimbangkan dalam *Integrity constraints*:

1. *Required data;*
2. *Attribute domain constraints;*
3. *Multiplicity;*
4. *Entity integrity;*
5. *Referential integrity;*
6. *General constraints.*

4.5.5 Langkah Tinjau model data logikal dengan pengguna

Langkah ini Untuk memastikan bahwa model data logikal mewakili persyaratan data perusahaan, langkah ini meninjau model dengan pengguna. Jika pengguna tidak puas dengan model, pengulangan langkah sebelumnya dalam metodologi mungkin diperlukan. Jika pengguna puas dengan model, langkah selanjutnya melakukan pendekatan integrasi tampilan.

4.5.6 Langkah Menggabungkan model data logikal ke dalam model data global (langkah opsional)

Langkah ini dapat dilakukan secara opsional dan diperlukan hanya untuk desain basis data yang memiliki banyak tampilan pengguna yang dikelola melalui pendekatan integrasi tampilan. Ini akan menggabungkan model data logikal lokal menjadi satu model data logikal global yang mewakili semua tampilan pengguna dalam basis data.

4.5.7 Langkah Periksa pertumbuhan di masa mendatang

Tahap Normalisasi

Langkah ini memastikan apakah ada perubahan signifikan yang mungkin terjadi di masa mendatang dan untuk menilai model data logikal sehingga dapat mengakomodasi perubahan tersebut.

4.6 Pemodelan *Phisical*

Pemodelan *Phisical* merupakan penjelasan implementasi basis data pada secondary storage; menguraikan basis relasi , organisasi file, rancangan indexes agar acces data menjadi *efficient* , semua yang berhubungan dengan kendala integrity, langkah-langkah keamanan terkait. Langkah-langkah adalah sebagai berikut:

4.6.1 Langkah Menterjemahkan Model Data Logikal untuk DBMS Target

Langkah pertama adalah membuat rencana basis data relasional dari model data logikal yang dapat digunakan dalam DBMS relasional target. Bagian pertama dari proses ini adalah menterjemahkan hubungan dalam model data logikal ke dalam bentuk yang dapat digunakan dalam DBMS relasional target. Proses ini memerlukan pemahaman mendalam tentang fungsi DBMS target. Bagian pertama proses menggunakan informasi ini untuk menghasilkan desain relasi dasar; bagian kedua memerlukan dokumentasi informasi yang dikumpulkan selama tahap

pengumpulan dan analisis persyaratan dan didokumentasikan dalam kamus data. Perancang harus memahami sistem yang mendukung definisi domain, kendala integritas relasional, dan kendala umum; sistem yang mendukung definisi kunci primer, kunci asing, dan kunci alternatif; sistem yang mendukung definisi data yang diperlukan (misalnya, apakah sistem mengizinkan atribut didefinisikan sebagai NOT NULL).

1. Langkah Desain relasi dasar
   Langkah ini menyusun dan mengasimilasi informasi tentang relasi yang dibuat selama desain basis data logikal. Informasi ini dapat diperoleh dari kamus data dan definisi relasi yang dijelaskan oleh DBDL. Definisi untuk setiap relasi yang ditemukan dalam model data logikal terdiri dari:
   Nama relasi;
   Daftar atribut sederhana dalam tanda kurung;
   Primer keys dan, lternate keys (AK) dan *foreign keys;*
   Referential integrity constraints untuk setiap *foreign keys* yang teridentifikasi.
   Dari kamus data, kami juga memiliki untuk setiap atribut:
   Domainnya, yang terdiri dari tipe data, panjang, dan *constraints* apa pun pada domain;
   Nilai default opsional untuk atribut;
   Atribut dapat menampung nol;
   Atribut diturunkan dan, jika demikian, bagaimana cara menghitungnya.
2. Langkah Desain Representasi dari Data Turunan
   Dalam langkah ini, diputuskan bagaimana menampilkan data turunan saat ini dalam model data logikal di DBMS yang dimaksud. Nilai atribut lain, yang disebut sebagai atribut turunan atau terhitung, dapat ditemukan dengan memeriksanya. Ini dilakukan dengan memeriksa model data logikal dan kamus data, yang menghasilkan daftar semua atribut turunan. Apakah atribut yang diturunkan disimpan

dalam basis data atau dihitung setiap kali dibutuhkan adalah perbedaan dari sudut pandang desain basis data phisical. Untuk menjaga penghitungan yang benar dan integritas basis data, sangat penting untuk memastikan bahwa perubahan ini dilakukan secara konsisten.

3. Langkah Merancang *General Constraints*
   Proses memperbarui hubungan ini mungkin dibatasi oleh *Integrity constraints* yang mengatur transaksi yang diwakili oleh pembaruan. Pada tahap ini, constraints umum yang tersisa harus dipertimbangkan. Rancangan kendala tersebut sekali lagi bergantung pada DBMS; beberapa sistem menyediakan lebih banyak fasilitas untuk mendefinisikan kendala umum daripada yang lain. Selain itu, karena sistem sesuai dengan standar SQL, beberapa batas dapat diimplementasikan dengan mudah.

4.6.2 Langkah Merancang Organisasi dan Indeks File

Langkah ini menemukan cara optimal untuk menyusun file guna menyimpan relasi utama dan indeks yang diperlukan untuk mencapai tingkat kinerja yang dapat diterima. Langkah ini juga menentukan lokasi tupel dan relasi yang akan digunakan untuk penyimpanan sekunder.

1. Langkah Menganalisis Transaksi
   Langkah ini membutuhkan pemahaman tentang fungsi transaksi yang akan dijalankan di basis data dan untuk menganalisis transaksi penting. Selain itu, untuk menjalankan desain basis data fisik secara efektif, Anda juga perlu memahami transaksi atau kueri yang akan terjadi di basis data. Ini memiliki data kuantitatif dan kualitatif. Untuk menganalisis transaksi, kriteria kinerja seperti:
   Transaksi yang sering berjalan dan akan berdampak signifikan terhadap kinerja;

Transaksi yang sangat penting untuk pengoperasian bisnis;
Waktu-waktu ketika akan ada permintaan tinggi pada basis data.

2. Langkah Pemilihan Organisasi File
   Langkah ini adalah proses memilih organisasi file terbaik untuk setiap relasi, jika DBMS target mengizinkannya. DBMS relasional mungkin tidak memberikan banyak pilihan untuk memilih organisasi file, tetapi beberapa dapat ditetapkan sebagai indeks yang ditentukan.
3. Langkah Pemilihan Indeks
   Langkah ini dalam upaya untuk meningkatkan kinerja sistem, langkah menentukan untuk menambahkan indeks adalah memilih organisasi file yang tepat untuk relasi untuk menjaga agar tupels tidak terurut dan membuat indeks sekunder sebanyak yang diperlukan.
4. Langkah Perkirakan Kebutuhan Ruang Disk
   Tahapan ini menghitung perkiraan jumlah space pada disk yang akan dibutuhkan oleh database. Selain itu, kebutuhan konfigurasi perangkat keras saat ini dapat menangani implementasi basis data fisik.

4.6.3 Langkah Rancang Tampilan Pengguna

Langkah ini merancang tampilan pengguna yang ditemukan selama tahap pengumpulan dan analisis, yang merupakan bagian dari persyaratan siklus hidup pengembangan sistem basis data.

4.6.4 Langkah Merancang Mekanisme Keamanan

Langkah ini menetapkan langkah-langkah keamanan yang akan dilakukan. Jenis fasilitas yang ditawarkan oleh sistem bervariasi. Sekali lagi, perancang basis data perlu memahami fasilitas yang ditawarkan oleh DBMS yang dimaksud. DBMS relasional biasanya menyediakan dua jenis keamanan basis data: keamanan sistem dan keamanan data.

Keamanan sistem mencakup akses dan penggunaan basis data pada tingkat sistem, seperti nama pengguna dan kata sandi, sedangkan keamanan data mencakup akses dan penggunaan objek basis data (seperti tampilan dan relasi), serta tindakan yang dapat dilakukan pengguna pada objek tersebut.

4.6.5 Monitor dan Tune Sistem Operasional

Tahap ini memastikan penyimpanan dan akses data yang efisien dan memantau sistem operasional untuk memperbaiki keputusan desain yang salah. Faktor yang digunakan untuk mengukur efisiensi operasional sistem adalah sebagai berikut:

1. Throughput transaksi:
   Sekumpulan transaksi yang dapat diproses dalam jangka waktu tertentu. Untuk sistem tertentu, seperti reservasi maskapai penerbangan, throughput transaksi yang tinggi sangat penting untuk keberhasilan sistem.
2. Waktu menanggapi.
   Ini adalah jumlah waktu yang telah dihabiskan untuk menyelesaikan satu transaksi. Faktor yang memengaruhi waktu respons yang mungkin tidak dapat dikendalikan oleh perancang termasuk waktu komunikasi dan pemuatan sistem. Waktu respons dapat dipersingkat dengan mengurangi perselisihan dan waktu tunggu, terutama waktu tunggu disk I/O; mengurangi jumlah sumber daya yang dibutuhkan; dan menggunakan komponen yang lebih cepat.
3. Penyimpanan disk:
   Pada disk adalah catatan jumlah ruang pada disk yang diperlukan untuk menyimpan file database. Jumlah disk yang digunakan harus ditebang sebanyak mungkin oleh perancang.

4.7 Penerapan Desain Basis Data pada DBMS Tertentu

Penerapan desain basis data untuk Sistem Manajemen Basis Data (DBMS) tertentu memerlukan pelaksanaan struktur dan skema basis data yang telah dirancang untuk DBMS yang dipilih. Paragraf berikutnya menguraikan prosedur dasar yang terlibat dalam melaksanakan desain database pada sistem manajemen database (DBMS) tertentu.

1. Memilih DBMS
   Sistem manajemen basis data (DBMS) yang akan digunakan untuk mengelola basis data harus dipilih terlebih dahulu. Sistem manajemen basis data (DBMS) seperti MySQL, PostgreSQL, Oracle, Microsoft SQL Server dan MongoDB adalah yang paling banyak digunakan. Pilih sistem manajemen basis data (DBMS) yang paling memenuhi kebutuhan perusahaan Anda dan fitur yang Anda inginkan.
2. Membuat Basis Data
   Setelah menentukan DBMS yang akan digunakan, langkah selanjutnya adalah membangun database baru dengan menggunakan sistem tersebut. Selama proses pembuatan database, kerangka dasar yang akan digunakan untuk menyimpan tabel, kolom, indeks, dan data lainnya akan dibuat.
3. Membuat Skema Tabel
   Langkah selanjutnya yang harus dilakukan adalah membuat tabel menggunakan arsitektur database yang telah ditentukan sebelumnya dalam proses. Setiap tabel harus memiliki kolom yang memberikan indikasi kualitas yang dipilih selama tahap desain.
4. Menentukan Kunci Utama dan Kunci Asing:
   Tetapkan kunci utama, juga dikenal sebagai pengidentifikasi utama, ke setiap tabel. Kunci ini adalah karakteristik pembeda yang dapat digunakan untuk mengidentifikasi setiap baris.

Selain itu, pastikan untuk menentukan kunci asing untuk mencirikan koneksi antar tabel secara memadai.

5. Mengatur Indeks:
   Prosedur pembentukan indeks pada kolom yang sering digunakan untuk tujuan pencarian atau pengurutan data. Penerapan indeks memiliki potensi untuk meningkatkan efisiensi operasi pencarian dalam database, sehingga meningkatkan kinerja secara keseluruhan.
6. Menentukan Tipe Data
   Sangat penting bahwa setiap kolom di dalam tabel diberi tipe data yang sesuai sesuai dengan sifat data yang ingin disimpan. Sebagai ilustrasi, kolom yang berisi data numerik diberi tipe data angka, sedangkan kolom yang berisi data tekstual diberi tipe data teks.
7. Menyusun Constraints
   Menerapkan *constraints* atau peraturan di atas meja untuk menjamin bahwa input data mematuhi norma yang ditetapkan. *Instance* dari *constraints* NOT NULL dapat digunakan untuk menjamin bahwa kolom tertentu tidak berisi nilai kosong. Demikian pula, kendala UNIQUE dapat digunakan untuk memverifikasi bahwa nilai yang berbeda hadir dalam kolom tertentu.
8. Mengimpor Data Awal
   Jika terdapat data awal yang memerlukan input ke dalam basis data, disarankan untuk menggunakan fungsi impor yang ditawarkan oleh sistem manajemen basis data (DBMS) untuk memfasilitasi transfer data tersebut ke dalam tabel yang relevan.
9. Mengoptimalkan Kinerja
   Tugas yang dihadapi melibatkan penerapan teknik pengoptimalan basis data, yang mencakup membuat keputusan berdasarkan informasi terkait pengindeksan, mengevaluasi kebutuhan untuk strategi partisi, dan

melakukan aktivitas penyetelan untuk meningkatkan kinerja sistem manajemen basis data (DBMS).

10. Uji Coba dan Perbaikan

    Setelah pengembangan database selesai, sangat penting untuk melakukan pengujian menyeluruh untuk memverifikasi bahwa fungsionalitas dan kinerja database memenuhi ekspektasi yang diantisipasi. Jika masalah teridentifikasi, sangat penting untuk melakukan perbaikan yang diperlukan dan terlibat dalam proses berulang sampai database berfungsi secara optimal.

    Pemanfaatan desain basis data dalam kaitannya dengan sistem manajemen basis data (DBMS) tertentu merupakan proses penting dalam membangun kerangka kerja data yang efektif yang memenuhi persyaratan aplikasi dan operasi bisnis. Eksekusi yang hati-hati dari proses ini diperlukan untuk memastikan keberhasilan keseluruhan dari sistem basis data.

DAFTAR PUSTAKA

Conger, S. 2012. *Hands-on database : an introduction to database design and development*. New Jersey: Pearson Education, Inc.

Connolly, T. and Carolyn Begg. 2015. *Database Systems : A Practical Approach to Design, Implementation, and Management*. 6th edn. Edinburgh Gate: Pearson Education Limited. Available at: http://link.springer.com/10.1007/978-1-4842-1191-5.

Coronel, C. and Morris, S. 2023. *Database Systems: Design, Implementation, and Management*. Edited by 14th. Cengage Learning.

Gupta, S.B. and Aditya Mittal. 2017. *An Introduction to Database Management Systems*. india: UNIVERSITY SCIENCE PRESS. Available at: https://doi.org/10.1108/eb047557.

BAB 5
NORMALISASI

Oleh Raymond Sunardi Oetama

5.1 Konsep normalisasi dan tujuannya

Salah satu metode untuk merancang sebuah basis data relasional adalah dengan menggunakan proses yang umumnya dikenal sebagai normalisasi (Silberschatz, Korth and Sudarshan, 2011). Dalam database, terdapat beberapa bentuk normal utama:

1. Bentuk Normal Pertama (1NF) berfungsi seperti aturan dasar yang mengatakan bahwa dalam tabel, tidak boleh memiliki kolom dengan nama yang sama lebih dari sekali, dan setiap sel dalam kolom harus sederhana, berisi satu nilai saja.
2. Bentuk Normal Kedua (2NF) adalah langkah kedua dalam merancang sebuah basis data yang terstruktur dengan baik. Tujuannya adalah untuk memastikan bahwa setiap informasi dalam tabel bergantung sepenuhnya pada kunci utama tabel tersebut, tanpa ada ketergantungan sebagian.
3. Bentuk Normal Ketiga (3NF) adalah tahap yang memastikan bahwa setiap atribut dalam tabel bergantung dengan benar pada kunci utama, tidak ada ketergantungan transitif.
4. Bentuk Normalisasi Tingkat BCNF merupakan lanjutan dari 3NF. BCNF memastikan bahwa setiap tabel dalam database harus dirancang sedemikian rupa sehingga tidak ada atribut bukan kunci yang bergantung pada atribut lain selain dari kunci super

Normalisasi data merupakan cara untuk mengatur dan merancang basis data agar lebih rapi, efisien, dan mudah digunakan. Ini seperti mengelola buku catatan Anda agar teratur dan mudah ditemukan. Misalnya, bayangkan Anda memiliki daftar pelanggan dan pesanan mereka dalam tabel. Normalisasi akan memastikan bahwa setiap pelanggan hanya terdaftar satu kali, sehingga tidak ada pengulangan. Ini membantu menghemat ruang dan menghindari kebingungan. Selain itu, normalisasi membuat data lebih mudah diperbarui dan dicari. Ini seperti memiliki daftar alamat yang diatur berdasarkan nama kota, sehingga Anda dapat dengan cepat menemukan alamat yang Anda cari. Dengan normalisasi, data lebih akurat, lebih mudah dikelola, dan lebih cepat dalam penggunaan, sehingga membantu bisnis dan analisis data berjalan lebih baik. Jadi, normalisasi adalah cara cerdas untuk mengatur data agar lebih efisien dan berguna.

5.2 Normalisasi Tingkat Pertama

Normalisasi Tingkat Pertama atau dalam istilah Inggaris disebut sebagai First Normal Form (1NF) adalah tingkatan pertama dalam normalisasi basis data. Dalam 1NF, sebuah tabel harus memenuhi dua kriteria utama (Silberschatz, Korth and Sudarshan, 2011):

1. Setiap kolom dalam tabel harus memiliki nama yang unik.. Ini berarti setiap atribut atau kolom dalam sebuah tabel harus diberi nama yang berbeda dan tidak boleh ada dua kolom dengan nama yang sama dalam tabel tersebut. Jika ada dua kolom dalam sebuah tabel dengan nama yang sama, ini akan menyebabkan kebingungan dan ambiguitas dalam struktur tabel dan penggunaan data. Masalah ini dapat menyulitkan pemahaman dan penggunaan data dalam basis data.
2. Setiap sel dalam tabel harus berisi nilai atomik. Ini berarti bahwa dalam sebuah tabel, setiap sel atau elemen dalam

tabel hanya boleh berisi satu nilai, dan tidak boleh berisi kombinasi atau struktur data yang lebih kompleks. Ketika data dalam sel tidak atomik, ini dapat menyebabkan masalah seperti ketidakkonsistenan, kesulitan dalam pencarian, duplikasi data, dan kesalahan dalam penggunaan data. Data yang terstruktur dengan baik, dengan setiap sel berisi nilai tunggal, lebih mudah dikelola dan digunakan.

Beberapa isu penting dalam kasus atomik adalah Alamat dan Nama. Alamat sering dianggap bukan nilai atomik karena terdiri dari beberapa bagian yang dapat diuraikan menjadi komponen-komponen lebih kecil, seperti nomor rumah/jalan, nama jalan, kota, negara, dan kode pos. Dalam konteks desain basis data, Nama biasanya dianggap sebagai nilai yang tidak atomik, karena nama seseorang terdiri dari beberapa komponen yang dapat dipecah menjadi elemen-elemen yang lebih kecil. Sebagian besar nama terdiri dari nama depan, nama tengah, dan nama belakang. Selain itu, ada situasi di mana nama seseorang dapat mencakup gelar akademik atau gelar kehormatan.

Selanjutkan kita akan membahas beberapa contoh permasalahan bentuk normal tingkat pertama berikut dengan solusinya.

Contoh 1:

Tabel Pelanggan_Lama mencatat nama pelanggan (Ani dan Budi) beserta jenis kelamin mereka (Pria dan Wanita) dengan notasi "Yes" dan "No". Kolom "Jenis_Kelamin" tidak memiliki nama Unik: Artinya, kolom ini memiliki nama yang sama untuk dua nilai yang berbeda, yaitu "Pria" dan "Wanita." Kolom ini seharusnya memiliki nama yang unik agar kita bisa mengidentifikasi setiap kolom dengan jelas.

Tabel 5.1. Pelanggan_Lama

Nama	Jenis_Kelamin	
	Pria	Wanita
Ani	No	Yes
Budi	Yes	No

Tabel Pelanggan_Lama belum memenuhi persyaratan 1NF karena salah satu kolomnya, yaitu "Jenis_Kelamin," tidak memiliki nama yang unik. Untuk memenuhi 1NF, sebaiknya kita menggabungkan kolom "Pria" dan "Wanita" menjadi satu kolom dengan nama "Jenis_Kelamin," menjadi Tabel Pelanggan Baru berikut ini.

Tabel 5.2. Pelanggan Baru

Nama	Jenis_Kelamin
Ani	Wanita
Budi	Pria

Perbedaan utama dari kedua tabel ini adalah bahwa tabel Pelanggan_Lama mencoba merepresentasikan jenis kelamin sebagai dua kolom terpisah ("Pria" dan "Wanita") yang menggunakan notasi "Yes" dan "No". Sedangkan tabel yang baru sudah menggunakan satu kolom "Jenis_Kelamin" dengan nilai yang lebih langsung dan jelas ("Pria" dan "Wanita"). Tabel Pelanggan_Baru ini sudah memenuhi persyaratan Normalisasi Tingkat Pertama (1NF). Setiap kolom memiliki nama yang unik, dan setiap sel dalam kolom berisi hanya satu nilai atomik.

Contoh 2:

Tabel Produk_Lama ini sebenarnya digunakan untuk mencatat produk beserta warnanya. Namun, ada satu masalah di

sini. Pada kolom Produk terdapat dua baris yang disatukan. Kolom Produk sebaiknya diisi nama produk tanpa disatukan.

Tabel 5.3. Produk Lama

Produk	Warna
Mouse	Putih
	Hitam

Pada tabel Produk_Lama, kolom Produk disatukan biasanya terjadi pada spreadsheet. Tabel ini belum memenuhi syarat 1NF karena ada sel dalam tabel yang tidak berisi data yang atomik. Pada aturan 1NF kolom Produk harus dilengkapi sehingga menjadi sebagai berikut.

Tabel 5.4. Produk Baru

Produk	Warna
Mouse	Putih
Mouse	Hitam

Sekarang Tabel ini sudah memenuhi syarat 1NF, karena setiap kolom dalam tabel sudah memiliki nama yang unik dan setiap sel dalam tabel sudah berisi hanya satu nilai. Dalam tabel ini, ada dua entri untuk produk "Mouse," masing-masing dengan warna yang berbeda. Meskipun ini masih memiliki duplikasi produk, itu sudah memenuhi persyaratan 1NF karena setiap baris dalam tabel adalah entri yang unik. Hal ini yang menjadi kelemahan pada bentuk 1NF. Namun demikian, Untuk menghindari duplikasi produk yang mungkin membingungkan, Anda dapat mempertimbangkan untuk menggunakan identifikasi unik seperti nomor seri produk jika ada.

Contoh 3:

Perhatikan pada Tabel Produk_Lama dengan kasus yang lain di bawah ini. Tabel ini mencatat informasi tentang produk bersama dengan warna-warna yang terkait. Namun, ada beberapa masalah dalam tabel ini. Dalam kolom "Produk," terdapat beberapa entri yang menggabungkan dua produk, seperti "Keyboard dan Mouse". Ini membuat tabel ini tidak memenuhi syarat 1NF, di mana setiap sel dalam Tabel harus berisi satu nilai atomik. Kemudian, kolom Warna juga menggabungkan beberapa warna untuk beberapa produk. Ini membuat tabel ini tidak memenuhi prinsip 1NF, di mana setiap sel dalam Tabel harus berisi satu nilai atomik.

Tabel 5.5. Produk

Produk	Warna
Keyboard, Mouse	Putih
Printer	Hitam, Putih

Untuk membuat tabel tersebut memenuhi syarat 1NF, Anda perlu mengorganisasi data dalam tabel sehingga setiap sel dalam Tabel hanya berisi satu nilai atomik. Setelah dipisahkan, tabelnya akan terlihat seperti tabel di bawah ini.

Tabel 5.6. Produk Baru

Produk	Warna
Keyboard	Putih
Mouse	Putih
Printer	Hitam
Printer	Putih

Dalam tabel Produk_Baru ini, kolom Produk sudah memiliki nilai yang atomik. Kolom Warna memisahkan produk dengan warna yang berbeda menjadi baris yang terpisah. Ini membuat tabel lebih terstruktur daripada tabel sebelumnya, di mana warna-warna yang berbeda digabungkan menjadi satu sel dalam satu kolom. Sekarang Tabel ini sudah memenuhi syarat 1NF, karena setiap kolom dalam tabel sudah memiliki nama yang unik dan setiap sel dalam tabel sudah berisi hanya satu nilai.

5.3 Normalisasi Tingkat Kedua

Normalisasi Tingkat Kedua, atau dalam istilah Inggrisnya disebut dengan *Second Normal Form* (2NF), merupakan langkah kedua dalam proses normalisasi dan mengacu pada beberapa aturan yang harus dipenuhi oleh sebuah tabel agar memenuhi 2NF. Aturan utama dari 2NF adalah sebagai berikut (Silberschatz, Korth and Sudarshan, 2011) :

1. Tabel harus memenuhi syarat 1NF. Setiap kolom memiliki nama yang unik serta setiap sel dalam tabel harus berisi nilai atomik.
2. Setiap kolom yang bukan kunci utama harus memiliki dependensi fungsional penuh pada kunci utama . Jika ada kolom bukan kunci dalam tabel yang tidak memiliki dependensi fungsional penuh pada kunci utama, itu dapat menghasilkan masalah dalam desain database yang dapat mempengaruhi integritas data dan kekonsistenan. Oleh karena itu, aturan 2NF dirancang untuk menghindari masalah semacam itu dengan memastikan bahwa setiap kolom bukan kunci tergantung sepenuhnya pada kunci utama.

Pada persyaratan 2NF ini terdapat istilah kunci utama. Kunci utama , atau dalam istilah bahasa Inggrisnya primary key, adalah kolom atau sekelompok kolom yang digunakan untuk

secara unik mengidentifikasi setiap baris dalam tabel dalam basis data. Fungsi utama dari kunci utama adalah memastikan bahwa tidak ada duplikat data dalam tabel dan memungkinkan untuk dengan mudah mencari, mengakses, dan merujuk ke data yang spesifik dalam tabel.

Sifat-sifat dari sebuah kunci utama adalah:

1. Unik: Setiap nilai dalam kolom kunci utama harus unik di seluruh tabel. Ini berarti tidak ada dua baris dalam tabel yang memiliki nilai kunci utama yang sama.
2. Tidak Boleh *Null*: Kolom kunci utama tidak boleh memiliki nilai *null* (kosong) karena nilai *null* tidak unik dan tidak dapat digunakan untuk mengidentifikasi secara eksklusif baris tertentu.
3. Konsisten: Nilai-nilai dalam kolom kunci utama harus tetap konsisten sepanjang waktu. Ini berarti bahwa kunci utama tidak boleh berubah setelah data dimasukkan ke dalam tabel.

Pada persyaratan 2NF, juga terdapat istilah dependensi fungsional. Ketergantungan fungsional kolom B terhadap kolom A ditulis A → B dalam relasi R berarti bahwa setiap nilai A secara unik berhubungan dengan satu nilai B, dimana A serta B dapat terdiri dari satu atau beberapa kolom. Terdapat bebepa jenis dependensi fungsional sbb:

1. Dependensi fungsional penuh antara kolom B dan A terjadi ketika B sepenuhnya bergantung pada A, dimana A berupa kunci utama.
2. Ketergantungan fungsional parsial kolom B dan A terjadi ketika B hanya bergantung pada sebagian dari A, dimana A berupa kunci utama.
3. Ketergantungan transitif terjadi ketika kolom C memiliki ketergantungan fungsional pada kolom B dan kolom B memiliki ketergantungan fungsional pada kolom A.

Pada aturan 2NF, dalam setiap Tabel, semua kolom hanya memiliki dependensi fungsional penuh kepada kunci utama dan menghindari dependensi fungsional parsial. Sedangkan permasalahan dependensi transitif akan diatur pada 3NF.

Bila sebuah Tabel memiliki Kunci Utama tunggal yang terdiri satu kolom saja. Maka sudah dipastikan tidak akan terjadi masalah dependensi fungsional parsial. Bila kunci utama berupa gabungan beberapa kolom, maka ada kemungkinan terjadinya masalah dependensi fungsional parsial.

Dalam sebuah Tabel berisi kolom-kolom dilambangkan dengan himpunan R{A,B,C,D, ...Z} yang memiliki kolom A, B, C, ... Z. Untuk A$\rightarrow$ C, maka C memiliki dependensi fungsional penuh ke A. Bila {A,B} $\rightarrow$ D, D memiliki dependensi fungsional penuh ke {A,B}. Namun bila ternyata B $\rightarrow$ E, maka E memiliki dependensi fungsional penuh ke B namun tidak ke A sehingga E memiliki dependensi fungsional parsial ke {A,B}. Hal inilah yang dilarang dalam bentuk 2NF.

Dependensi parsial dapat mengakibatkan masalah seperti redundansi data, anomali data, dan kesulitan pemeliharaan.

1. **Redundansi Data**: Dependensi parsial dapat mengakibatkan duplikasi data dalam tabel. Misalnya, jika beberapa atribut bergantung pada sebagian dari kunci utama, informasi tersebut perlu diulang-ulang setiap kali kunci utama berubah. Ini mengakibatkan pemborosan ruang penyimpanan dan membuat tabel lebih sulit dipelihara.
2. **Anomali Data**: Dependensi parsial dapat menyebabkan anomali data seperti anomali pencatatan, anomali penghapusan, dan anomali pembaruan. Ini dapat terjadi ketika perubahan dalam data tertentu (misalnya, menghapus atau memperbarui sebuah baris) dapat mengganggu integritas data dan menghasilkan hasil yang tidak diharapkan.

3. **Kesulitan Pemeliharaan**: Tabel dengan dependensi parsial dapat menjadi sulit untuk dikelola dan pemeliharaan. Karena data terduplikasi, perubahan pada satu tempat mungkin memerlukan perubahan di banyak tempat lain dalam tabel, yang bisa membingungkan dan meningkatkan risiko kesalahan.

Selanjutkan kita akan membahas beberapa contoh permasalahan bentuk normal tingkat kedua yang diikuti dengan solusinya.

Contoh 1:

Tabel Penjualan_Lama di bawah ini sudah memenuhi 1NF karena semua kolom sudah memiliki nilai yang atomik dan setiap sel memiliki nilai yang atomik. Tabel ini juga sudah memiliki kunci utama {Penjualan_id, Produk_id} yang terdiri dari gabungan kolom Penjualan_id dan kolom Produk_id. Untuk kolom Jumlah sudah dependensi fungsional penuh ke kunci utama gabungan ini. Namun pada kolom Nama memiliki dependensi fungsional parsial kepada kunci utama gabungan ini karena kolom Nama hanya memiliki depedensi kepada Penjualan_id, dan tidak kepada kolom Produk_Id. Sehingga Tabel ini belum memenuhi syarat 2NF.

Tabel 5.7. Penjualan Lama

Penjualan_Id	Produk_Id	Nama	Jumlah
1	1	Agus	3
1	2	Agus	4
2	1	Budi	5
2	2	Budi	6

Supaya Tabel Penjualaan_Lama di atas memenuhi persyaratan 2NF, maka tabel ini akan dipecah menjadi 2 tabel yaitu tabel Penjualan_Master dan tabel Penjualan_Detil seperti terlihat di bawah ini.

Tabel 5.8. Penjualan_Master

Penjualan_Id	Nama
1	Agus
2	Budi

Tabel 5.9. Penjualan Detil

Penjualan_Id	Produk_Id	Jumlah
S1	P1	3
S1	P2	4
S2	P1	5
S2	P2	6

Pada tabel Penjualan_Master, kolom Nama memiliki dependensi fungsional penuh ke kunci utama Penjualan_id sehingga memenuhi 2NF. Sedangkan pada Tabel Penjualan Detil, kolom Jumlah juga memiliki dependensi penuh ke kunci utama gabungan kolom Penjualan_id dan Produk_id sehingga sudah memenuhi syarat 2NF.

Contoh 2:

Tabel Penjualan_Lama di bawah ini sudah memenuhi 1NF dan juga sudah memiliki kunci utama gabungan yang terdiri dari kolom Penjualan_Id dan kolom Produk_Id. Untuk kolom Jumlah memiliki dependensi fungsional penuh ke kunci utama gabungan. Namun pada kolom Produk memiliki dependensi fungsional parsial karena kolom Produk hanya memiliki depedensi fungsional hanya kepada Produk_id, dan tidak kepada kolom Penjualan_id. Sehingga Tabel ini belum memenuhi syarat 2NF.

Tabel 5.10. Penjualan_Lama

Penjualan_Id	Produk_Id	Produk	Jumlah
S1	P1	Baju	3
S1	P2	Celana	4
S2	P1	Baju	5
S2	P2	Celana	6

Supaya Tabel Penjualan ini memenuhi persyaratan 2NF, maka tabel ini akan dipecah menjadi 2 tabel yaitu tabel Penjualan dan tabel Produk seperti terlihat di bawah ini. Pada tabel Penjualan, kolom Jumlah memiliki dependensi fungsional penuh ke kunci utama gabungan Penjualan_id dan Produk_id sehingga memenuhi 2NF. Sedangkan pada Tabel Produk, Produk sudah memiliki dependesi fungsional penuh ke Produk_id. Sehingga Tabel Penjualan sudah memenuhi syarat 2NF.

Tabel 5.11. Penjualan

Penjualan_Id	Produk_Id	Jumlah
S1	P1	3
S1	P2	4
S2	P1	5
S2	P2	6

Tabel 5.12. Produk

Produk_Id	Produk
P1	Baju
P2	Celana

5.4 Normalisasi Tingkat Ketiga

Normalisasi tingkat Ketiga (3NF) adalah langkah ketiga dalam proses normalisasi dan mengacu pada sejumlah aturan yang harus dipenuhi oleh sebuah tabel agar memenuhi 3NF. Aturan utama dari 3NF adalah sebagai berikut (Silberschatz, Korth and Sudarshan, 2011):

1. Tabel harus memenuhi syarat 2NF.
2. Tidak boleh ada dependensi fungsional transitif dari kolom yang bukan kunci utama pada kolom lainnya yang bukan kunci utama.

Dalam sebuah Tabel berisi kolom-kolom dilambangkan dengan himpunan R{A,B,C,D, ...Z} yang memiliki kolom A, B, C, ... Z. A adalah kunci utama sehingga A→ {A, B, C, ... Z}. Setaip kolom memiliki dependensi fungsional penuh ke A. Bila ditemukan kolom B sedemikian sehingga B→C dimana B bukan kunci utama, maka terjadi dependensi fungsional transitif, karena C dependensi fungsional ke B, B depensi fungsional ke A. Dikatakan bahwa C dependensi transitif ke A melalui B.

Dalam konteks normalisasi database, perbedaan utama antara 2NF dan 3NF adalah bahwa 2NF fokus pada dependensi parsial, sementara 3NF fokus pada dependensi transitif. 3NF adalah tingkat normalisasi yang lebih tinggi daripada 2NF, yang berarti tabel yang mencapai 3NF juga akan otomatis mencapai 2NF. Akibat dari dependensi fungsonal transitif sama dengan dependensi fungsional parsial yaitu redundansi data, anomali data, dan kesulitan pemeliharaan.

Selanjutnya, kita akan membahas beberapa contoh masalah dalam bentuk Normalisasi Tingkat Ketiga beserta solusinya.

Contoh:

Tabel Penjualan ini memiliki kunci utama pada kolom Penjualan_id dimana semua kolom memiliki dependensi ke kunci utama ini. Karena kunci utama tidak memiliki kunci kandidat, maka tidak terjadi dependensi fungsional parsial. Sehingga memenuhi 2NF. Namun bila diperhatikan, kolom Produk juga memiliki dependensi penuh pada kolom Produk_id, sedangkan kolom Produk_id bukan kunci utama. Terjadi dependensi transitif disini sehingga tidak memenuhi syarat 3NF.

Tabel 5.13. Penjualan

Penjualan_Id	Tanggal	Produk_id	Produk	Harga	Jumlah
S1	1/12/2023	P1	Baju	500000	2
S1	1/12/2023	P2	Celana	600000	3
S2	2/12/2023	P3	Topi	200000	1
S2	2/12/2023	P4	Kacamata	250000	2

Kolom Tanggal, Produk_id, dan Jumlah memiliki dependensi fungsional penuh ke Penjualan_id, sedangkan Produk dan Harga memiliki dependensi transitif ke Penjualan_id melalui Produk_id. Untuk memperbaikinya, maka Tabel Penjualan ini bisa dipisahkan menjadi 2 tabel yaitu Tabel Penjualan, Tabel Produk seperti terlihat di bawah ini.

Tabel 5.14. Penjualan

Penjualan_Id	Tanggal	Produk_id	Jumlah
S1	1/12/2023	P1	2
S1	1/12/2023	P2	3
S2	2/12/2023	P3	1
S2	2/12/2023	P4	2

Tabel 5.15. Produk

Produk_id	Produk	Harga
P1	Baju	500000
P2	Celana	600000
P3	Topi	200000
P4	Kacamata	250000

Pada Tabel Penjulan yang baru, kolom Tanggal, Produk, dan Jumlah memiliki dependensi penuh ke kunci_utama Penjualan_id. Sedangkan pada Tabel Produk, Produk dan Harga memiliki dependensi penuh ke kunci utama. Sehingga baik Tabel Penjualan maupun Tabel Produk tidak memiliki Dependensi Transitif.

Dengan demikian, kedua Tabel ini sudah memenuhi persyaratan 3NF.

5.5 Normalisasi Tingkat BCNF

Bentuk Normal 1NF, 2NF, 3NF merupakan bentuk yang paling sering digunakan dalam proses normalisasi database. Namun ada satu bentuk yang sukup populer untuk tingkat lanjutan yaitu *Boyce Codd Normal Form* (BCNF). BCNF adalah tingkat normalisasi dalam desain database yang lebih tinggi daripada 3NF. Tujuan utama BCNF adalah menghilangkan dependensi fungsional yang tidak diinginkan dalam tabel database. Aturan utama dari BCNF adalah sebagai berikut (Silberschatz, Korth and Sudarshan, 2011):

1. Tabel harus memenuhi syarat 3NF
2. Bila terjadi dependensi fungsional dari kolom B ke kolom A, maka kolom A adalah kunci super. Sebuah superkey dari sebuah relasi adalah kumpulan satu atau lebih kolom yang nilai-nilainya dijamin dapat mengidentifikasi baris dalam Tabel tersebut secara unik.

Selanjutnya, kita akan membahas beberapa contoh masalah dalam bentuk Normalisasi Tingkat BNCF beserta solusinya.
Contoh:

Misalkan kita memiliki tabel Pesanan_Lama dengan atribut (kolom) berikut: Nomor_Pesanan (Kunci utama), Nama_Pelanggan, Nomor_Telepon_Pelanggan. Nama kolom sudah unik, nilai sel sudah atomik (1NF). Tidak terjadi dependensi fungsional parsial (2NF) serta Tidak terjadi dependensi fungsional trsansitif (3NF).

Tabel 5.16. Pesanan_Lama

Nomor_Pesanan	Nama_Pelanggan	Nomor_Telepon_Pelanggan
123	Roni	454488
124	Rudi	453459
125	Roni	454488

Sekarang kita periksa beberapa dependensi fungsional antar kolom-kolom dengan kunci utamanya. Pada dependensi fungsional Nomor_Pesanan → Nama_Pelanggan sesuai dengan BCNF karena Nomor_Pesanan adalah kunci utama. Pada dependensi fungsional Nomor_Pesanan → Nomor_Telepon_Pelanggan masih sesuai dengan BCNF karena Nomor_Pesanan adalah kunci utama. Namun bila diperhatikan, juga terjadi dependensi fungsional Nama_Pelanggan → Nomor_Telepon_Pelanggan, namun disini Nama_Pelanggan bukan kunci super dan juga bukan kunci kandidat. Sehingga belum memenuhi syarat BCNF.

Supaya memenuhi syarat BCNF, Tabel Pesanan_Lama akan dipisah menjadi 2 tabel yaitu Tabel Pesanan_Baru dan Tabel Pelanggan. Tabel Pesanan_Baru memiliki dua kolom, yaitu Nomor_Pesanan sebagai kunci utama dan Pelanggan_Id sebagai *foreign key*. Tabel Pelanggan memiliki Pelanggan_id sebagai kunci utama dan Nama_Pelanggan dan Nomor_Telepon_Pelanggan sebagai kolom bukan kunci.

Tabel 5.17. Pesanan_Baru

Nomor_Pesanan	Pelanggan_Id
123	1
124	2
125	1

Tabel 5.18. Pelanggan

Pelanggan_Id	Nama_Pelanggan	Nomor_Telepon_Pelanggan
1	Roni	454488
2	Rudi	453459

Untuk menilai apakah tabel ini memenuhi syarat BCNF (*Boyce-Codd Normal Form*), kita perlu memeriksa apakah semua dependensi fungsional yang ada telah dipenuhi. Tabel Pesanan_Baru ini memenuhi syarat BCNF karena Nomor_Pesanan adalah kunci super yang unik untuk tabel ini. Semua dependensi fungsional telah dipenuhi dan tidak ada masalah dependensi parsial atau transitif dalam tabel ini. Tabel Pelanggan juga memenuhi syarat BCNF karena setiap kolom sepenuhnya bergantung pada kunci super (Pelanggan_Id), dan tidak ada dependensi fungsional yang tidak diinginkan yang melibatkan kolom bukan kunci.

5.6 Kelebihan dan Kekurangan Normalisasi

Proses normalisasi dalam desain database memiliki beberapa kelebihan yang sangat penting dalam pengelolaan data dan pemeliharaan database. Berikut adalah beberapa kelebihan utama dari proses normalisasi:

1. **Mengurangi Redundansi Data**: Salah satu tujuan utama normalisasi adalah mengurangi redundansi data, yaitu menyimpan data yang sama berulang-ulang dalam tabel yang berbeda. Ini menghemat ruang penyimpanan dan memastikan konsistensi data. Dengan data yang disimpan hanya sekali, perubahan data hanya perlu dilakukan satu kali.
2. **Meningkatkan Integritas Data**: Normalisasi membantu menjaga integritas data dengan memastikan bahwa data

yang disimpan sesuai dengan aturan dan batasan yang telah ditentukan. Ini membantu mencegah data yang salah atau tidak konsisten.

3. **Meningkatkan Performa *Query*:** Normalisasi dapat meningkatkan performa *query* karena tabel yang lebih kecil dan terfokus memungkinkan pencarian dan pengambilan data menjadi lebih efisien. Ini dapat mengurangi waktu yang diperlukan untuk menjalankan *query* pada database.
4. **Memudahkan Pemeliharaan**: Dengan struktur yang terorganisasi dengan baik, pemeliharaan database menjadi lebih mudah. Perubahan pada data atau aturan bisnis dapat diimplementasikan dengan lebih efisien tanpa mengganggu seluruh database.
5. **Mengurangi Anomali Data**: Normalisasi membantu mengurangi kemungkinan terjadinya anomali data seperti anomali pencatatan, anomali penghapusan, dan anomali pembaruan. Ini memastikan bahwa operasi CRUD (*Create*, *Read*, *Update*, *Delete*) dapat dilakukan dengan benar.
6. **Meningkatkan Fleksibilitas**: Normalisasi memungkinkan fleksibilitas dalam mengubah struktur database jika diperlukan. Dengan tabel yang terpisah dengan baik, Anda dapat menambahkan atau menghapus entitas atau atribut dengan relatif mudah.
7. **Mengurangi Duplikasi Data yang Tidak Diperlukan**: Dengan normalisasi yang tepat, data yang sama tidak perlu disimpan dalam banyak tempat. Ini mengurangi potensi kesalahan dan memudahkan pemeliharaan.
8. **Meningkatkan Keamanan Data**: Normalisasi memungkinkan Anda untuk mengendalikan akses ke data dengan lebih baik. Anda dapat menerapkan izin akses yang tepat pada tabel yang sesuai.
9. **Memudahkan Penggabungan Data**: Normalisasi memisahkan data menjadi entitas yang berdiri sendiri,

yang memudahkan penggabungan data dari berbagai sumber dan aplikasi yang berbeda.

10. **Memudahkan Pemahaman Database**: Normalisasi menghasilkan struktur database yang lebih terstruktur dan terorganisasi, sehingga memudahkan pemahaman dan dokumentasi database.

Meskipun proses normalisasi memiliki banyak kelebihan, ada beberapa kekurangan yang perlu dipertimbangkan dalam desain database:

1. **Kompleksitas Desain**: Normalisasi bisa menghasilkan struktur database yang kompleks dengan banyak tabel terpisah. Ini dapat membuat desain database lebih sulit dipahami, terutama untuk pemula, dan memerlukan waktu lebih lama untuk mengembangkan dan memelihara.
2. **Pemeliharaan yang Lebih Rumit**: Normalisasi yang berlebihan atau tidak tepat dapat mengakibatkan pemeliharaan yang lebih rumit. Mengganti struktur tabel atau menambahkan kolom baru dapat memerlukan perubahan yang rumit dalam banyak tempat.
3. **Waktu Eksekusi *Query* yang Lebih Lama**: Normalisasi yang berlebihan dapat menyebabkan waktu eksekusi *query* yang lebih lama karena adanya banyak penggabungan tabel. Ini dapat menjadi masalah dalam aplikasi yang membutuhkan respons cepat.
4. **Penggunaan Ruang Penyimpanan yang Lebih Besar**: Normalisasi dapat menghasilkan penggunaan ruang penyimpanan yang lebih besar karena adanya banyak tabel dan indeks. Ini bisa menjadi masalah dalam lingkungan dengan batasan penyimpanan.

DAFTAR PUSTAKA

Silberschatz, A., Korth, H.F. and Sudarshan, S. 2011. 'Database system concepts'.

BAB 6
SQL : DATA DEFINITION LANGUAGE

Oleh Weri Sirait

6.1 Pendahuluan

Structured Query Language (SQL) merupakan bahasa standar yang paling banyak digunakan dalam mengakses dan mengelola data didalam database modern seperti MySQL, SQL Server, Oracle dan sebagainya. SQL memungkinkan kita untuk berkomunikasi dengan database. Query MySQL merupakan dasar yang sangat penting dalam mempelajari ilmu sistem basis data.
Secara umum SQL sendiri terbagi menjadi tiga bahasa yaitu:

1. *Data Definition Language* (DDL)
   DDL digunakan untuk mengelola kerangka atau struktur database termasuk tabel.
2. *Data Manipulation Language* (DML)
   DML digunakan untuk mengelola dan manipulasi data-data pada tabel.
3. *Data Control Language* (DCL)
   DCL digunakan untuk pengaturan manajemen user baik itu pembuatan dan hak akses user.

Ketiga query tersebut memiliki sebuah manfaat dimana untuk memberikan pengertian terhadap objek yang ada di sebuah sistem basis data seperti dalam pembuatan tabel, memanipulasi database dan mengontrol database (Andriani, 2019).

6.2 Data Defenition Language

Pada *Structured Query Language* (SQL) perintah yang pertama digunakan adalah *Data Defenition Language* (DDL). Perintah ini digunakan untuk menspesifikasikan skema basis data. Dengan DDL user dimungkinkan untuk membuat database, tabel dan lainnya (Meidi and Ken, 2019). Selain membuat database dan membuat tabel perintah DDL juga dapat digunakan untuk merubah nama database, tabel, struktur field pada tabel, menambahkan key pada tabel serta menghapus database atau tabel (Anwar and Purnamasari, 2018).

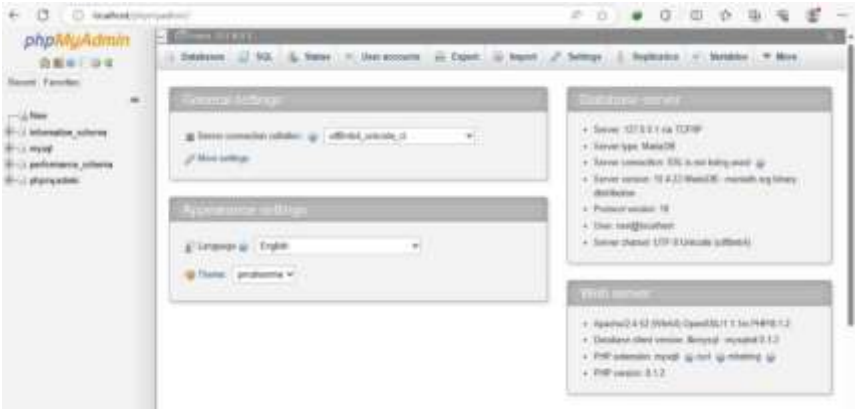
6.3 Operasi Data Defenition Language

Pada dasarnya perintah DDL terdiri dari lima jenis perintah yaitu **CREATE, DROP, RENAME, ALTER,** dan **SHOW**.

6.3.1 Perintah CREATE

Create merupakan perintah yang berfungsi untuk membuat database, tabel, procedure, function atau yang lainnya pada suatu database.

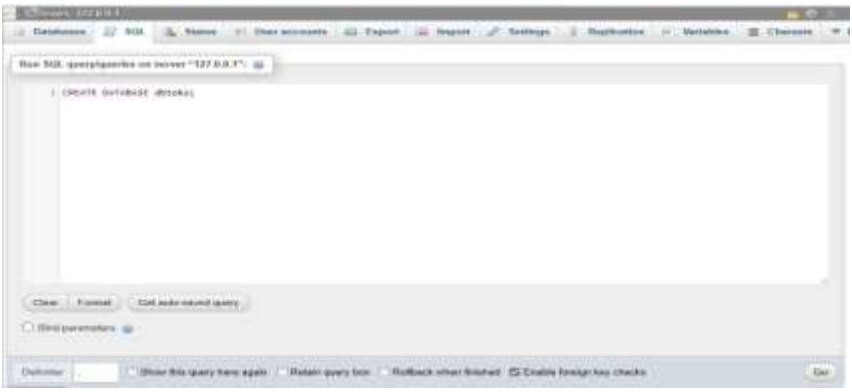
Pada contoh disini menggunakan MySQL pada XAMPP untuk membuat database, tabel dan perintah lainnya pada Data Defenition Language. Untuk mengaktifkan MySQL, aktifkan control panel XAMPP kemudian buak browser dan ketikkan localhost/phpmyadmin, maka akan tampil jendela phpMyAdmin seperti terlihat pada gambar 6.1 berikut ini



Gambar 6.1. Jendela phpMyAdmin (MySQL)

1. Sintaks SQL untuk melakukan pembuatan database baru di dalam basis data sebagai berikut:
   a. CREATE DATABASE nama_database;
   b. Pembuatan database baru (CREATE DATABASE) identik dengan pembuatan lemari arsip yang baru. Sebagai contoh jika akan membuat database dbtoko, maka sintaks SQL adalah sebagai berikut:
   c. **Contoh:** Membuat database dengan nama dbtoko
   d. CREATE DATABASE dbtoko;

   Untuk membuat database menggunakan perintah SQL, kita bisa melakukan dengan cara aktifkan tab SQL pada phpMyAdmin maka akan tampil jendela SQL Editor. Selanjutnya kita dapat menuliskan perintah untuk membuat database.



Gambar 6.2. Perintah SQL untuk membuat database

Setalah selesai menuliskan perintah SQL, klik button GO yang terletak di bawah jendela, maka database dbtoko akan dibuat oleh MySQL. Perintah create juga dapat digunakan untuk membuat tabel pada database yang telah kita buat, selain itu juga kita dapat menambahkan atau membuat indeks pada tabel yang kita buat.

2. Perintah untuk membuat tabel dengan perintah SQL adalah:
   a. CREATE TABLE nama_tabel (
   b. field1 tipe(panjang),
   c. field2 tipe(panjang),
   d. field n tipe(panjang),
   e. PRIMARY KEY (field_key)
   f.);
   g. di mana:
      1. Nama_tabel adalah nama tabel yang kan dibuat.
      2. Field1, field2,..., field n adalah nama-nama atribut yang akan terdapat di dalam tabel nama_tabel.

3. Tipe(panjang) adalah domain nilai masing-masing atribut tersebut ditentukan berdasarkan tipe datanya.
4. Primary key adalah sintaks yang digunakan untuk membuat primary key.
5. (field_key) adalah nama atribut yang akan dijadikan primary key.

Pembuatan tabel baru identik dengan penambahan kotak arsip baru ke sebuah lemari arsip yang telah ada. Misalnya dalam sebuah database dbtoko yang sudah ada akan ditambahkan sebuah tabel baru dengan nama tabel barang.
Contoh: Membuat tabel barang
Ketentuan sebagai berikut:

Tabel 6.1. Tabel barang

FieldName	Type	Lenght
Kode_barang	VARCHAR	7
nama_barang	VARCHAR	50
satuan	VARCHAR	50
harga	INT	
stok	INT	

Maka, perintah SQL nya adalah:

```
CREATE TABLE barang (
kode_barang VARCHAR(7) NOT NULL,
nama_barang VARCHAR(50) NOT NULL,
satuan VARCHAR(50) NOT NULL,
harga INT NOT NULL,
stok INT NOT NULL,
PRIMARY KEY (kode_barang)
);
```

Gambar 6.3. Perintah SQL untuk membuat tabel

Penjelasan dari perintah di atas adalah:

1. **CREATE TABLE** merupakan perintah dasar untuk membuat sebuah tabel.
2. **Kode_barang, nama_barang, satuan, harga, stok** merupakan nama field untuk tabel barang.
3. **VARCHAR, INT** merupakan tipe data untuk field-field yang dibuat.
4. **Angka 7, 50, dan 50** merupakan panjang maksimal dari suatu field. Untuk tipe data tertentu seperti tipe DATE, INT dan tipe-tipe data numerik lainnya dapat tidak diberikan panjang maksimalnya.
5. **PRIMARY KEY** merupakan perintah yang digunakan untuk pembuatan primary key atau atribut kunci. Perintah ini bisa ditambahkan setelah nama field yang akan diberi primary key atau diletakkan di akhir perintah pembuatan tabel.

6.3.2 Perintah SHOW

1. Perintah Show digunakan untuk menampilkan database atau tabel yang ada atau yang telah dibuat. Caranya aktifkan pilihan SQL pada SQL Editor kemudian ketikkan perintah:
   SHOW DATABASES;

Gambar 6.4. Hasil Perintah Show databases pada MySQL

2. Untuk menampilkan tabel yang telah dibuat maka perintah yang digunakan adalah berikut:
   SHOW TABLES;
   Untuk menampilkan tabel pada database dbtoko dapat dilihat pada gambar dibawah ini:

Gambar 6.5. Perintah Show Tables pada MySQL

Setelah menuliskan perintah SQL, klik GO. Maka akan tampil tabel yang pernah dibuat pada database dbtoko. Hasil perintah dapat dilihat pada gambar dibawah ini:

Gambar 6.6. Hasil Perintah Show Tables pada MySQL

6.3.3 Perintah ALTER

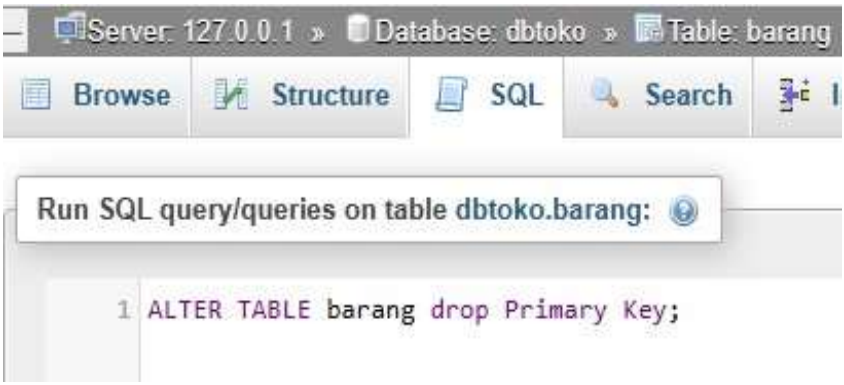
Perintah ini digunakan untuk mengubah struktur tabel yang sebelumnya sudah ada. Perubahan bisa jadi mengubah nama tabel, nama field, penambahan atau penghapusan field yang ada. Penulisan untuk perintah alter sebagai berikut:

1. Membuat Primary Key
   Perintah Alter dapat digunakan untuk menambahkan primary key apabila dalam pembuatan tabel belum ada atribut kunci. Perintah SQLnya sebagai berikut:
   ALTER TABLE nama_table add Primary Key (nama_field_primary);
   Contoh: Menambahkan Primary Key pada field kode_barang di tabel barang
   ALTER TABLE barang add Primary Key (kode_barang);

Gambar 6.7. Perintah Alter Add Primary Key pada MySQL

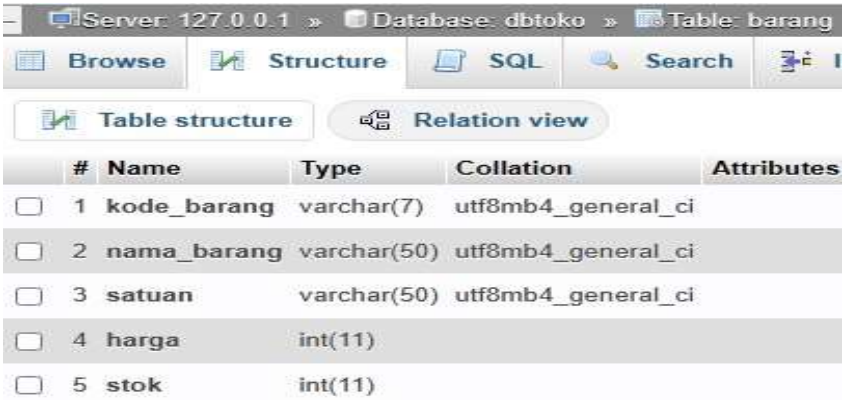
Gambar 6.8. Hasil Alter Add Primary Key pada MySQL

2. Menghapus Primary Key
   Apabila sebuah tabel sudah memiliki primary key dan kita ingin menghapusnya maka perintah SQLnya sebagai berikut:
   ALTER TABLE name_table drop Primary Key;
   Contoh: Menghapus nilai primary key pada tabel barang
   ALTER TABLE barang drop Primary Key;



Gambar 6.9. Perintah Alter Drop Primary Key pada MySQL

Hasil perintah diatas dapat dilihat pada gambar berikut ini:



Gambar 6.10. Hasil Drop Primary Key pada MySQL

3. Menambah Field

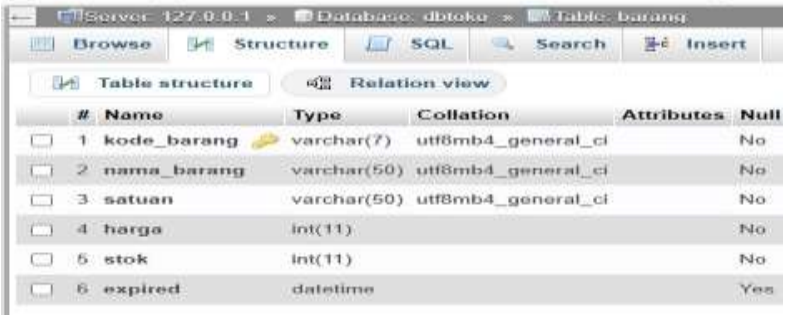
   Perintah alter bisa juga digunakan untuk menambahkan field pada tabel yang sudah ada. Perintah SQLnya sebagai berikut:

   ALTER TABLE nama_table add Spesifikasikode;

Contoh: kita akan menambahkan field baru dengan nama expired dengan tipe data datetime dimana letaknya disisipkan setelah stok.
ALTER TABLE barang add expired datetime after stok;

Gambar 6.11. Alter tambah field pada MySQL

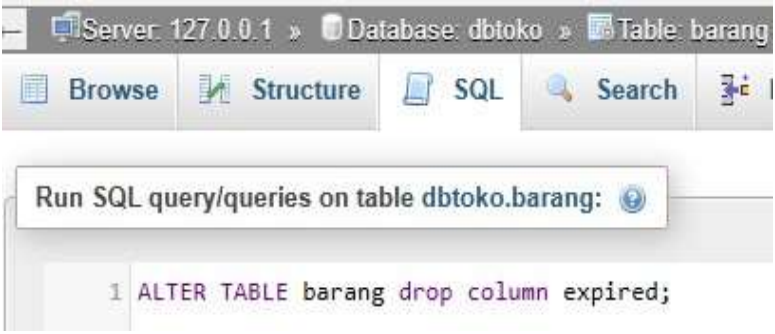
Setelah perintah diatas dieksekusi maka struktur tabel barang akan terdapat penambahan kolom baru yaitu field expired yang akan berada pada posisi setelah field stok. Hasil perintah diatas dapat dilihat pada gambar berikut ini:



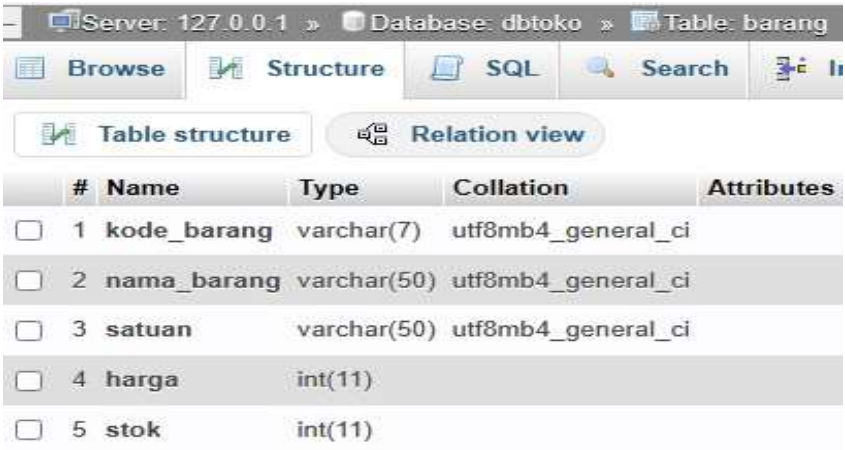
#	Name	Type	Collation	Attributes	Null
1	kode_barang	varchar(7)	utf8mb4_general_ci		No
2	nama_barang	varchar(50)	utf8mb4_general_ci		No
3	satuan	varchar(50)	utf8mb4_general_ci		No
4	harga	int(11)			No
5	stok	int(11)			No
6	expired	datetime			Yes

Gambar 6.12. Hasil Alter tambah field pada MySQL

4. Mengurangi Field
   Saat kita membuat tabel mungkin ada kesalahan dalam membuat field yang tidak terpakai atau salah. Untuk melakukan penghapusan dapat dilakukan dengan perintah SQL berikut:
   ALTER TABLE nama_table drop column field;
   Contoh: Tabel barang ada field stok yang ingin dihapus
   ALTER TABLE barang drop column expired;



Gambar 6.13. Perintah Alter mengurangi field pada MySQL



Gambar 6.14. Hasil Alter mengurangi field pada MySQL

5. Mengubah Nama Field

Perintah alter bisa juga digunakan untuk mengubah nama field. Perintah SQL sebagai berikut:

ALTER TABLE nama_tabel change field_lama field_baru (length);

Contoh: Mengganti nama field dari nama_barang menjadi nm_barang dengan type varchar panjang fieldnya 40

ALTER TABLE BARANG change nama_barang nm_barang VARCHAR (40);

Gambar 6.15. Perintah Alter merubah field pada MySQL

Gambar 6.16. Hasil Alter merubah field pada MySQL

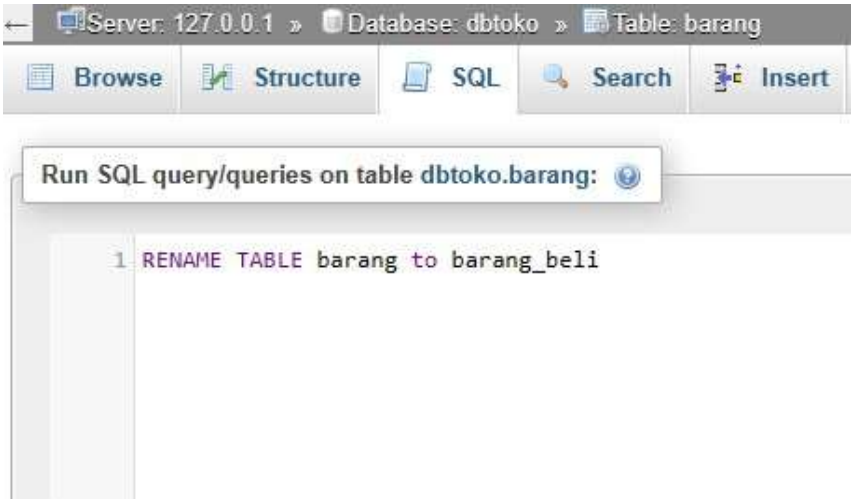
6.3.4 Perintah RENAME

Suatu perintah yang dapat kita gunakan untuk merubah nama database atau tabel yang ada. Pada perubahan tertentu perintah rename dapat digunakan bersamaan dengan perintah Alter.

Format penulisan perintah Renama adalah sebagai berikut :
RENAME TABLE [nama_tabel_lama] to [nama_tabel_baru]

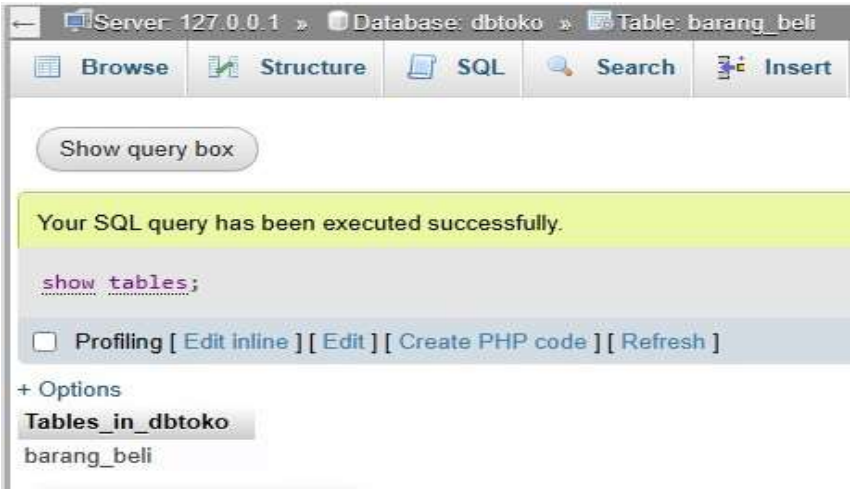
Contoh:
RENAME TABLE barang to barang_beli

Perintah di atas digunakan untuk mengganti nama tabel barang menjadi tabel barang_beli. Tampilan SQL pada MySQL dapat dilihat pada gambar diberikut ini.



Gambar 6.17. Perintah Rename SQL pada MySQL

Setelah perintah diatas dieksekusi maka tabel barang akan berubah nama menjadi tabel barang_beli. Hasil perintah diatas dapat diliha pada gambar berikut ini:



Gambar 6.18. Hasil Perintah Rename SQL pada MySQL

6.3.5 Perintah DROP

Drop merupakan perintah yang dapat digunakan untuk menghapus database, tabel, fuction, index, procedure ataupun trigger.

1. Penghapusan tabel dari suatu database (DROP TABLE) identik dengan penghapusan kotak arsip lama yang ada disebuah lemari arsip berserta isinya jika ada. Misalnya akan menghapus tabel barang_beli dalam database dbtoko maka perintah SQL dalam penghapusan tabel adalah :
   DROP TABLE nama_table;
   Contoh: DROP TABLE barang_beli;
   Dimana barang_beli adalah nama tabel yang ada didatabase dbtoko yang akan dihapus. Setelah perintah diatas dieksekusi maka tabel barang_beli pada database dbtoko akan terhapus.
2. Penghapusan database (DROP DATABASE) identik dengan penghapusan seluruh lemari arsip sekaligus beserta isinya

jika ada. Perintah SQL dalam penghapusan database adalah:

DROP DATABASE nama_database;

Contoh: DROP DATABASE dbtoko;

Setelah perintah diatas dieksekusi maka database dengan nama dbtoko beserta tabel-tabel didalamnya akan terhapus.

DAFTAR PUSTAKA

Andriani, A. . P. B. E. 2019. DESAIN DATABASE dengan ERD dan LRS. 1st edn. Yogyakarta: TEKNOSAIN.

Anwar, I. and Purnamasari, K. K. 2018. 'Penerjemahan Teks Bahasa Indonesia Menjadi Data Definition Language (DDL) dengan Penanganan Kalimat Majemuk', Jurnal Ilmiah Komputer dan Informatika (KOMPUTA), (Ddl), p. 6. Available at: https://repository.unikom.ac.id/id/eprint/59392.

Meidi, D. P. P. and Ken, K. P. 2019. 'Penerjemah Teks Bahasa Indonesia Ke Dalam Dml (Data Manipulation Language) dengan Sub-Query', Jurnal Ilmiah Komputer dan Informatika, 2(9), pp. 1–8.

BAB 7
SQL : DATA MANIPULATION LANGUAGE

Oleh Nur Azizah

7.1 Pendahuluan

SQL merupakan bahasa standar yang paling banyak digunakan dalam DBMS, SQL sendiri terbagi menjadi dua jenis yaitu DDL (*Data Definition Language*) untuk mendefinisikan tabel dalam database dan DML (*Data Manipulation Language*) untuk memanipulasi data atau informasi dalam suatu *database* sehingga itu menjadi informasi yang berguna, *Subquery* adalah bagian dari DML yang jika dipahami adalah *query* di dalam *query*, jadi untuk mendapatkan informasi dalam tabel atau tabel relasional yang sangat kompleks, kita memiliki kondisi yang dapat digunakan dengan *subquery* tertentu. *Subquery* sendiri dapat menghasilkan perintah SQL yang relatif lebih sederhana dibandingkan fungsi join, sehingga memperpendek perintah dan dapat menggunakan fungsi logika yang lebih mudah dipahami dibandingkan *query* biasa (Meidi and Ken, 2019). *Data manipulation language* (DML) adalah sub bahasa SQL yang digunakan untuk memanipulasi data dalam basis data yang telah dibuat. Perintah-perintah DML digunakan untuk mengelola data yang merupakan isi dari tabel. Dengan kata lain DML dapat dipahami sebagai perintah yang memanipulasi data dengan cara memasukkan data baru, mencari data lama, menghapus data, memodifikasi data, dan lain-lain. Pada dasarnya perintah DML terdiri dari empat macam yaitu INSERT, SELECT, UPDATE dan *DELETE*.

7.2 Operasi Data Manipulation Language

Pada dasarnya perintah DML terdiri dari empat macam yaitu INSERT, SELECT, UPDATE dan *DELETE* (Andriani, 2019).

7.2.1 Insert

Insert merupakan perintah yang digunakan untuk menyisipkan atau memasukkan data baru ke dalam tabel. Format perintah untuk memasukkan data yaitu:

INSERT INTO nama_table (field1, field2, field3, ...) VALUES(nilai1, nilai2, nilai3, ...);

INTO sifatnya opsional. Boleh ditulis, boleh tidak. Daftar nama kolom (field1, field2, field3) boleh ditulis, boleh tidak. Jika tidak ditulis, maka dianggap sesuai urutan nama kolom dalam pendefinisian tabel. Gunakan koma sebagai pemisah. {VALUE | VALUES} wajib ditulis. Boleh VALUE atau VALUES.

Terdapat dua cara untuk menginput data ke dalam tabel dengan menggunakan perintah INSERT. Yang pertama perintah INSERT dapat dituliskan secara lengkap dengan menyebutkan daftar field di dalam tabel yang diikuti daftar value atau data data yang akan dimasukkan ke dalam tabel. Cara penggunaan perintah INSERT yang kedua yaitu menyingkat perintah tanpa menyebutkan daftar nama field yang terdapat di dalam tabel. Untuk penguinputan data yang lengkap, maka lebih tepat menggunakan cara yang kedua karena akan menyingkat waktu penulisan perintah SQL. Tetapi untuk penginputan data yang tidak lengkap ke dalam tabel, maka lebih tepat menggunakan cara yang pertama, karena apabila kita menginputkan data yang tidak lengkap dengan cara yang kedua akan menimbulkan error.

Contoh 1 : menggunakan printah INSERT dengan cara pertama (memasukkan data tabel dengan menyebutkan daftar field name dalam tabel secara lengkap)

INSERT INTO mahasiswa (NIM nama_mahasiswa, tempat_lahir, tanggal_lahir, no_hp, alamat) values ('172005029',

'Nur Azizah', 'Padang Panjang', '1996-09-07', '0812346789', 'Batusangkar');

Contoh 2 : Menggunakan perintah INSERT dengan cara kedua (memasukkan data ke dalam tabel tanpa menyebutkan daftar field name)

INSERT INTO mahasiswa values ('172005029', 'Nur Azizah', 'Padang Panjang', '1996-09-07', '0812346789','Batusangkar');

Contoh 3 :

insert into member (no, nama, tgllhr, berat, status) value (null, 'Adi','1980-06-10',65.5,'A');

Keterangan : Menggunakan INTO, menggunakan nama filed, menggunakan VALUE, kolom yang AUTO_INCREMENT diisi dengan nilai NULL, pengisian data untuk setiap kolom ditulis sesuai urutan kolom dalam pendefinisian tabel.

Contoh 4 :

insert member values(null,'Budi','1985-12-10',50,'T');

Keterangan : Tidak menggunakan INTO, tidak menggunakan field , menggunakan VALUES, kolom yang AUTO_INCREMENT diisi dengan nilai NULL, pengisian data untuk setiap kolom ditulis sesuai urutan kolom dalam pendefinisian tabel.

7.2.2 Select

Select adalah perintah untuk mengambil atau menampilkan data dari tabel. Perintah SELECT dapat digunakan untuk pengambilan data secara sederhana dapat juga digunakan untuk pengambilan data dengan berbagai macam kriteria sesuai dengan data seperti apa yang ingin diambil. Penguasaan perintah SELECT ini nantinya akan sangat berguna dalam pembuatan query.

Untuk pegambilan data dengan perintah SELECT terbagi menjadi beberapa penggunaan yaitu pengambilan data hanya beberapa kolom atau field saja, dan pengambilan semua kolom atau field tetapi berdasarkan kriteria tertentu.

1. Menampilkan semua data
   Perintah SELECT ini merupakan perintah dasar yang fungsina yaitu mengambil semua data dari tabel tanpa terkecuali. Format dasar query select yang menampilkan seluruh isi tabel adalah:
   SELECT * FROM nama_tabel
   Contoh : mengambil dan menampilkan SEMUA data yang ada di dalam tabel Mahasiswa
   SELECT * from Mahasiswa;
   Tanda bintang (*) adalah wildcard sebagai pengganti 'pilih semua kolom'.
2. Menampilkan data dalam beberapa kolom saja
   Perintah SELECT ini merupakan perintah SELECT yang digunakan untuk mengambil semua data pada kolomatau field tertentu saja.
   Format dasarnya adalah:
   SELECT nama_kolom1, nama_kolom2,... FROM nama_tabel
   nama_kolom1 dan nama_kolom2 adalah nama kolom yang ingin ditampilkan. Misalnya kita ingin menampilkan kolom nama_dosen dan alamat dari tabel daftar_dosen, maka querynya adalah sebagai berikut:
   SELECT nama_dosen, alamat FROM daftar_dosen;
3. Menampilkan data berdasarkan kriteria
   Kondisi WHERE pada perintah SELECT digunakan untuk memilih data yang diperlukan saja, tanpa menampilkan data yang tidak memenuhi kondisi tersebut.
   Format dasar query SELECT…WHERE adalah:
   SELECT nama_kolom1, nama_kolom2,... FROM nama_tabel WHERE kondisi
   Contoh :
   SELECT nama_dosen FROM daftar_dosen WHERE alamat='Padang';

(*query* yang menampilkan data tentang dosen yang beralamat di Padang)

Operator yang dapat digunakan untuk membuat kriteria dalam perintah SELECT diantaranya :

Tabel 7.1. Daftar Operator yang bisa digunakan untuk membuat kriteria

Operator	**Keterangan**
=	Sama dengan
<> atau !=	Tidak sama dengan
>	Lebih besar dari
<	Lebih kecil dari
>=	Lebih besar atau sama dengan
<=	Lebih kecil atau sama dengan
BETWEEN .. AND	Antara dua nilai
LIKE	Mencari suatu pola
ORDER by	Mengurutkan data
AND, OR	Menggabungkan dua atau lebih kondisi
IN	Digunakan jika ingin mengetahui nilai pasti yang akan diambil
ALIAS	Mengganti nama kolom dan nama tabel
JOIN	Menampilkan data dari dua atau lebih

Sumber:(Andriani, 2019)

Contoh 1:
Penggunaan perintah WHERE dan "=" (menampilkan atau mengambil data mahasiswa yang NIMnya sama dengan 171200059)
SELECT * From Mahasiswa WHERE NIM = '1711200059';
Contoh 2:
Penggunaan perintah WHERE dan ">" (menampilkan data mahasiswa yang tahun masuknya di atas 2017)
SELECT * From Mahasiswa WHERE tahun_masuk > 2017;
Contoh 3 :
Penggunaan perintah WHERE dan "BETWEEN" (menampilkan data mahasiswa yang tahun masuknya antara 2017 dan 2022)
SELECT * From Mahasiswa WHERE tahun_masuk BETWEEN 2017 AND 2022;
Contoh 4 :
Penggunaan perintah WHERE dan "LIKE" dengan ketentuan huruf depan tertentu (menampilkan data mahasiswa yang nama awalnya diawali huruf J)
SELECT * From Mahasiswa WHERE nama_mahasiswa 'J%';
Contoh 5 :
Penggunaan perintah WHERE dan "LIKE" dengan ketentuan huruf depan tertentu (menampilkan data mahasiswa yang namanya di akhiri huruf j)
SELECT * From Mahasiswa WHERE nama_mahasiswa '%j';
Contoh 6 :
Penggunaan perintah ORDER By untuk membuat pengurutan tampilan data secara ascending (menampilkan data mahasiswa yang diurutkan berdasarkan abjad)
SELECT * From Mahasiswa ORDER By Nm_Mhs;

Contoh 7 :

Penggunaan perintah ORDER By untuk membuat pengurutan tampilan data secara ascending (menampilkan data mahasiswa yang diurutkan berdasarkan abjad terbalik)

SELECT * From Mahasiswa ORDER By Nm_Mhs DESC;

Contoh 8 :

Penggunaan perintah kombinasi WHERE, LIKE dan AND untuk menampilkan data dengan kriteria tertentu (menampilkan data mahasiswa dengan nama "Philip" dan tempat lahir "Medan")

SELECT * From Mahasiswa WHERE nama_mahasiswa LIKE 'John%' AND tempat_lahir='Medan';

4. Menampilkan data dengan fungsi aggregate

   Penggunaan perintah SELECT dapat juga digunakan untuk perhitungan matematika seperti penggunaan fungsi statistik yang dalam hal ini disebut fungsi aggregate selain untuk menampilkan data berdasarkan kriteria. Contoh fungsi aggregate yang bisa digunakan.

Tabel 7.2. Daftar fungsi aggregate

Operator	**Keterangan**
Sum()	Untuk menghitung jumlah nilai pada suatu kolom
Avg()	Untuk menghtiung nilai rata-rata pada suatu kolom
Max()	Untuk mencari nilai maksimal pada suatu kolom
Min()	Untuk mencati nilai minimal pada suatu kolom
Count()	Untuk menghitung jumlah baris pada suatu kolom

Sumber: (Andriani, 2019)

Contoh penggunaan fungsi Aggregate :
Buatlah tabel Nilai dengan format tabel sebagai berikut :

NIM	Nama_matakuliah	Nilai_Akhir
171200059	Perancangan Basis Data	85
171200060	Algoritma dan Struktur Data	79
171200061	Kewirausahaan	80
171200062	Digital Marketing	81

Contoh 1 : menampilkan total dari Nilai_Akhir
SELECT SUM (Nilai_Akhir) from Nilai;
Contoh 2 : menampilkan rata-rata dari Nilai_Akhir
SELECT AVG (Nilai_Akhir) from Nilai;
Contoh 1 : menampilkan nilai tertinggi dari Nilai_Akhir
SELECT MAX (Nilai_Akhir) from Nilai;
Contoh 1 : menampilkan nilai terendah dari Nilai_Akhir
SELECT MIN (Nilai_Akhir) from Nilai;
Contoh 1 : menampilkan jumlah data Nilai_Akhir
SELECT COUNT (Nilai_Akhir) from Nilai;

5. Cara Mengurutkan hasil tampilan data MySQL (SELECT…ORDER BY)
   Query dasar untuk SELECT…ORDER BY adalah:
   SELECT nama_kolom1,... FROM nama_tabel WHERE kondisi ORDER BY nama_kolom_urut.
   nama_kolom_urut adalah kolom yang akan diurutkan. Pengurutan bisa dari yang terkecil ke terbesar atau terbesar ke terkecil. Opsi ini dapat diatur dengan menambahkan instruksi ASC (singkatan dari ascending) untuk pengurutan kecil ke besar, dan DESC (singkatan dari descending) untuk urutan dari besar ke kecil. Kecuali

disebutkan lain, perintah ORDER BY MySQL menggunakan ASC secara default.
Contoh :
menampilkan semua data dosen pada tabel daftar_dosen dan diurutan kolom alamat secara abjad, maka querynya adalah sebagai berikut:
SELECT * FROM daftar_dosen ORDER BY alamat ASC;

7.2.3 Update

Update merupakan perintah yang digunakan untuk mengubah data atau memperbaharui data lama dengan data baru.
Bentuk umum pemanggilan UPDATE adalah :
UPDATE nama_tabel SET nama_kolom = data_baru WHERE kondisi
nama_tabel adalah nama tabel yang record/barisnya sedang diperbaharui (*update*), nama_kolom adalah nama kolom dari tabel yang sedang diperbaharui, data_baru adalah nilai data yang akan dimasukkan sebagai nilai baru dari kolom tersebut, dan kondisi adalah kondisi atau keadaan dari baris yang akan diubah.
Contoh :
Merubah alamat dari Mahasiswa yang mempunyai NIM = 171200059 UPDATE member SET alamat = “Padang“ WHERE no=“171200059"

7.2.4 Delete

Delete berfungsi untuk menghapus record dari *database*
Bentuk umum pemanggilan *DELETE* adalah :
DELETE FROM nama_table WHERE kondisi
nama_tabel adalah nama dari tabel yang record/barisnya akan dihapus (*delete*), kondisi adalah kondisi atau syarat dari baris yang akan dihapus.
Contoh : DELETE FROM Mahasiswa WHERE NIM=“171200059“;

DAFTAR PUSTAKA

Andriani, A.. P.B.E. 2019. *DESAIN DATABASE dengan ERD dan LRS*. 1st edn. Yogyakarta: TEKNOSAIN.

Meidi, D.P.P. and Ken, K.P. 2019. 'Penerjemah Teks Bahasa Indonesia Ke Dalam Dml (Data Manipulation Language) dengan Sub-Query', *Jurnal Ilmiah Komputer dan Informatika*, 2(9), pp. 1–8.

BAB 8
SQL: FUNGSI-FUNGSI

Oleh Heru Saputra

8.1 Pendahuluan

SQL (*Structured Query Language*) adalah langkah pertama yang penting dalam memahami interaksi database. SQL adalah bahasa pemrograman yang digunakan untuk mengelola dan mengambil data dalam database relasional (Fikry et al., 2018). Dalam pengembangan perangkat lunak dan manajemen basis data, SQL memainkan peran penting karena memungkinkan pengguna untuk melakukan berbagai operasi seperti menyimpan, memperbarui, menghapus, dan mengambil data secara efisien. Salah satu aspek penting dari SQL adalah fungsi yang memungkinkan pengguna melakukan operasi matematika, manipulasi string, dan manipulasi data lainnya. SQL memiliki fungsi-fungsi yang berperan penting dalam mengolah data dalam database agar hasilnya dapat digunakan untuk mengambil keputusan yang lebih baik. Pada bab ini, kita akan mengeksplorasi beberapa fungsi dasar SQL dan bagaimana fungsi tersebut dapat digunakan untuk memproses data secara efisien.

8.1.1 Pentingnya Fungsi dalam SQL

Salah satu alasan utama mengapa fungsi SQL penting adalah untuk mempermudah pemrosesan data. Dengan menggunakan fungsi SQL, kita dapat melakukan operasi matematika, manipulasi string, dan pemrosesan data lainnya dengan mudah dan efisien. Misalnya, kita bisa menggunakan fungsi *COUNT* untuk menghitung jumlah baris dalam tabel atau fungsi *SUM* untuk menjumlahkan nilai kolom tertentu. Salah satu fungsi tersebut memungkinkan kita dengan cepat mengambil informasi yang relevan dari database

tanpa harus melakukan perhitungan manual atau memanipulasi data secara manual. Selain itu, fungsi SQL juga memungkinkan kita mengelompokkan dan menggabungkan data, seperti penggunan fungsi *GROUP BY* dan *HAVING* memungkinkan kita mengelompokkan data berdasarkan kriteria tertentu seperti kategori atau rentang nilai. Hal ini sangat berguna ketika kita ingin melakukan analisis atau membuat laporan berdasarkan kelompok data tertentu. Misalnya, kita dapat menggunakan fungsi *AVG* untuk menghitung nilai rata-rata di setiap grup atau fungsi *MAX* untuk mencari nilai terbesar di setiap grup. Dengan fungsi ini, kita dapat dengan mudah menganalisis data dengan berbagai cara tergantung kebutuhan kita.

SQL juga memungkinkan kita melakukan transformasi dan manipulasi data yang kompleks. Dengan menggunakan fungsi seperti *CONCAT*, *SUBSTRING*, *UPPER* dan *LOWER* kita dapat menggabungkan, memotong, dan mengganti elemen data dalam kolom string. Misalnya, kita bisa menggunakan fungsi *CONCAT* untuk menggabungkan dua kolom menjadi satu, atau fungsi *SUBSTRING* untuk mengambil bagian tertentu dari sebuah string. Dengan fitur ini kita dapat dengan mudah melakukan manipulasi data yang kompleks seperti: Misalnya, Anda dapat mengubah format tanggal atau menghapus karakter tertentu di kolom.

8.2 Fungsi Agregasi

Salah satu fungsi utama perintah SQL agregasi adalah untuk melakukan penghitungan statistik pada data (*Data Analysis Expressions (DAX) di Power Pivot, n.d.)*. Dengan menggunakan perintah seperti COUNT, SUM, AVG, MIN dan MAX, kita dapat dengan mudah menghitung jumlah baris, menambahkan nilai tertentu, menghitung rata-rata, dan menemukan nilai terkecil dan terbesar di kolom tabel. Operasi agregasi ini sangat berguna ketika kita ingin mendapatkan informasi statistik dari database, seperti: Total penjualan, rata-rata, atau nilai maksimum dan minimum

dalam kumpulan data. Fungsi lain dari perintah agregasi adalah mengelompokkan dan mengagregasi data berdasarkan kriteria tertentu. Dengan menggunakan perintah GROUP BY, kita dapat mengelompokkan data berdasarkan kolom tertentu, seperti mengelompokkan data berdasarkan kategori produk atau wilayah geografis. Setelah dikelompokkan, kita dapat menggunakan perintah SQL gabungan untuk menghitung statistik dalam setiap grup, seperti jumlah penjualan di setiap kategori produk atau nilai rata-rata di setiap wilayah. Selain itu, fungsi perintah agregasi juga memungkinkan kita memfilter atau memilih data berdasarkan hasil operasi agregasi, seperti perintah HAVING memungkinkan kita menentukan kriteria spesifik untuk memfilter data yang dihasilkan dari operasi agregasi. Misalnya, kita bisa menggunakan perintah HAVING untuk memfilter hanya data yang total penjualannya lebih besar dari nilai tertentu, atau hanya data yang nilai rata-ratanya kurang dari nilai tertentu. Dengan kemampuan filter ini, kita dapat dengan mudah melakukan analisis data yang lebih detail dan relevan.

8.2.1 Fungsi COUNT

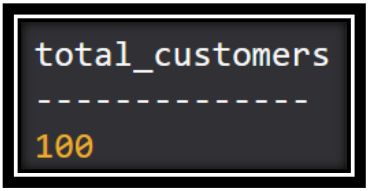
Perintah *COUNT* digunakan untuk menghitung jumlah baris atau record dalam suatu tabel. Tujuan dari perintah ini adalah untuk memberikan informasi tentang seberapa banyak data yang memenuhi suatu kondisi atau berada dalam suatu kolom. Gambar 8.1 merupakan contoh kode perintah *COUNT*:

```
SELECT COUNT(*) AS total_customers
FROM customers;
```

Gambar 8.1. Coding *Count*

Gambar 8.1 menjelaskan cara menghitung jumlah baris pada tabel *customers*. Tanda bintang (*) digunakan sebagai

parameter untuk menghitung seluruh baris dalam tabel. Kita juga bisa menggunakan nama kolom tertentu jika ingin menghitung jumlah baris yang memenuhi kriteria tertentu. kita dapat menggunakan "total_*customers*" untuk memberi nama hasil penghitungan jumlah baris, berikut hasilnya:



Gambar 8.2. Hasil SQL dari *Count*

Hasil dari perintah *COUNT* adalah 100 yang berarti tabel *customers* berisi 100 baris atau *record* seperti yang terlihat pada gambar 8.2. Perintah *COUNT* biasa digunakan di SQL untuk menghitung jumlah data dalam sebuah tabel, seperti jumlah pelanggan, jumlah transaksi, atau jumlah produk yang tersedia. Ini berguna untuk analisis data, pelaporan.

8.2.2 Fungsi *SUM*

Perintah *SUM* digunakan untuk menjumlahkan nilai kolom tabel. Fungsi dari perintah ini adalah untuk menghitung jumlah nilai pada kolom tertentu. Gambar 8.3 merupakan contoh kode perintah *SUM*:

```
SELECT SUM(quantity) AS total_quantity
FROM sales;
```

Gambar 8.3. Coding *Sum*

Gambar 8.3, menjelaskan penggunaan perintah SUM untuk menjumlahkan nilai dalam kolom "quantity" dalam tabel "sales". Kita mengambil hasil perhitungan dan menetapkan "*total_quantity*" untuk nama hasil tersebut, berikut hasilnya:

Gambar 8.4. Hasil SQL dari *Sum*

Hasil dari perintah *SUM* adalah 500 yang menunjukkan jumlah total nilai pada kolom "*quantity*" pada tabel "*sales*". Pernyataan *SUM* sering digunakan dalam SQL untuk menghitung jumlah kolom tertentu (seperti total penjualan, inventaris, atau total nilai transaksi).

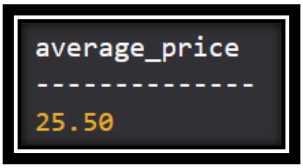
8.2.3 Fungsi AVG

Perintah *AVG* digunakan untuk menghitung rata-rata dari nilai dalam sebuah kolom dalam tabel. Fungsi ini menghasilkan nilai rata-rata dari nilai-nilai yang ada dalam kolom tertentu. Gambar 8.5 merupakan contoh kode perintah *AVG*:

```
SELECT AVG(price) AS average_price
FROM products;
```

Gambar 8.5. Coding *AVG*

Gambar 8.5, menjelaskan penggunaan perintah *AVG* untuk menghitung rata-rata nilai di kolom "*price*" pada tabel "*products*". Kemudian kita beri nama alias yang dihasilkan degan nama "*average_price*".



Gambar 8.6. Hasil SQL dari *AVG*

Hasil dari perintah *AVG* adalah 25,50 yang mewakili rata-rata nilai pada kolom "*price*" pada tabel "*products*". Perintah *AVG* sering digunakan dalam SQL untuk menghitung rata-rata pada kolom tertentu, seperti rata-rata harga produk, rata-rata peringkat pelanggan, atau rata-rata usia data yang ada.

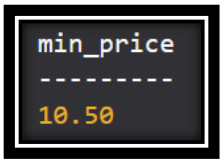
8.2.4 Fungsi MIN

Tujuan dari perintah *MIN* adalah untuk mendapatkan nilai terkecil dari suatu kolom tabel pada suatu tabel. Perintah ini digunakan untuk mencari nilai minimum atau minimal pada suatu kolom tertentu. Berikut contoh kode perintah SQL *MIN*:

```
SELECT MIN(price) AS min_price
FROM products;
```

Gambar 8.7. Coding *MIN*

Pada gambar 8.7, kita menggunakan perintah *MIN* untuk mencari nilai terkecil di kolom "*prices*" pada tabel "*products*". Hasil perhitungan disimpan dengan alias "*min_price*".



Gambar 8.8. Hasil SQL dari *MIN*

Hasil dari perintah MIN adalah 10,50 yang merupakan nilai terkecil pada kolom " *prices* " pada tabel " *products* ". Perintah *MIN* sering digunakan dalam SQL untuk mencari nilai terkecil pada kolom seperti harga produk, nilai tanggal, atau nilai numerik lainnya.

8.2.5 Fungsi *MAX*

Tujuan dari perintah *MAX* adalah untuk mendapatkan nilai terbesar dari suatu kolom tabel. Perintah ini digunakan untuk mencari nilai tertinggi atau terbesar pada kolom tertentu. Berikut contoh kode penggunaan perintah SQL *MAX*:

Gambar 8.9. Coding *MAX*

Pada gambar 8.8, kita menggunakan perintah *MAX* untuk mencari nilai terbesar dalam kolom "*price*" pada tabel "*products*". Hasil perhitungan akan disimpan dengan alias "*max_price*".

Gambar 8.10. Hasil SQL dari *MAX*

Hasil dari perintah MAX adalah 99,99 yang merupakan nilai terbesar pada kolom " *price*" pada tabel " *products* ". Pernyataan *MAX* sering digunakan dalam SQL untuk menemukan nilai terbesar di kolom seperti harga produk, tanggal, atau nilai numerik lainnya.

8.3 Fungsi String

Fungsi string dalam database SQL sangat penting dalam memproses dan memanipulasi data teks(MySQL, 2020). SQL memiliki berbagai operasi string yang dapat digunakan untuk memanipulasi string, seperti menggabungkan string (*concatenation*), memotong string (*substring*), mencari kata atau karakter dalam string (*search*), mengubah ukuran huruf dan seterusnya. Dengan menggunakan fungsi-fungsi ini, kita dapat memproses dan memanipulasi data teks dalam SQL dengan lebih efisien dan fleksibel.

8.3.1 Fungsi *CONCAT*

Perintah *CONCAT* digunakan untuk menggabungkan string teks atau kolom dari kolom atau nilai dalam dua tabel atau lebih. Fungsi ini memungkinkan Kita menggabungkan teks atau kolom dengan tanda ‘ ‘ untuk membentuk satu string yang lebih panjang. Berikut contoh kode perintah SQL *CONCAT*:

```
SELECT CONCAT(first_name, ' ', last_name) AS full_name
FROM employees;
```

Gambar 8.11. Coding *Concat*

Pada gambar 8.11, kita menggunakan perintah *CONCAT* untuk menggabungkan nilai kolom "*first_name*" dan "*last_name*" dalam tabel "*employees*" dengan tanda ‘ ‘. Hasil perhitungan akan disimpan dengan alias "full_name".

```
full_name
----------------
John Doe
Jane Smith
Mike Johnson
```

Gambar 8.12. Hasil SQL dari *Concat*

Hasil dari perintah *CONCAT* adalah gabungan dari nilai kolom "*first_name*" dan "*last_name*" dalam tabel "*employees*". Misalnya, "*John Doe*" adalah hasil penggabungan nilai "*John*" dari kolom "*first_name*" dan "*Doe*" dari kolom "*last_name*". Perintah *CONCAT* sering digunakan dalam SQL untuk menggabungkan nilai-nilai string atau kolom dalam rangka membangun teks yang lebih kompleks, seperti menggabungkan nama depan dan nama belakang menjadi satu nama lengkap, atau menggabungkan beberapa kolom menjadi satu string.

8.3.2 Fungsi *LENGTH*

Perintah *LENGTH* digunakan untuk menghitung jumlah karakter dalam sebuah string. Perintah ini digunakan untuk mengukur panjang atau jumlah karakter dalam sebuah nilai teks atau kolom teks dalam tabel. Berikut contoh kode perintah SQL *LENGTH*:

Gambar 8.13. Coding *Length*

Pada gambar 8.13, kita menggunakan perintah *LENGTH* untuk menghitung panjang atau jumlah karakter dalam kolom "*product_name*" pada tabel "*products*". Hasil perhitungan akan disimpan dengan alias "*name_length*".

Gambar 8.14. Hasil SQL dari *Length*

Hasil dari perintah *LENGTH* adalah jumlah karakter dalam kolom "*product_name*" pada setiap baris dalam tabel "*products*". Misalnya, "8" menunjukkan bahwa ada 8 karakter dalam "*product_name*" pada baris pertama. Perintah *LENGTH* sering digunakan dalam SQL untuk menghitung panjang string, memvalidasi batasan karakter, atau melakukan manipulasi teks.

8.3.3 Fungsi *SUBSTRING*

Fungsi dari perintah *SUBSTRING* adalah untuk mengambil sepotong atau substring dari sebuah string. Perintah ini digunakan untuk membagi atau memotong string menjadi bagian-bagian yang lebih kecil berdasarkan posisi karakter awal dan panjang substring yang diinginkan. Berikut contoh kode perintah SQL *SUBSTRING*:

```
SELECT SUBSTRING(product_name, 1, 5) AS substring_name
FROM products;
```

Gambar 8.15. Coding *Substring*

Pada gambar 8.15, kita menggunakan perintah *SUBSTRING* untuk mengambil substring dari kolom "*product_name*" pada tabel "*products*". Parameter pertama adalah string yang akan dipotong, parameter kedua adalah posisi karakter awal, dan parameter ketiga adalah panjang substring yang diinginkan. Hasil perhitungan akan disimpan dengan alias "*substring_name*".

```
substring_name
--------------
Apple
Banana
Orange
```

Gambar 8.16. Hasil SQL dari *Substring*

Hasil dari perintah *SUBSTRING* adalah *substring* dari kolom "*product_name*" pada setiap baris dalam tabel "*products*". Misalnya, "*Apple*" adalah *substring* pertama yang diambil dari "*product_name*" pada baris pertama, dimulai dari karakter pertama hingga karakter kelima. Perintah *SUBSTRING* sering digunakan dalam SQL untuk memisahkan atau memotong bagian tertentu dari string, seperti mengambil nama depan dari sebuah nama lengkap atau mengambil kode wilayah dari nomor telepon.

8.3.4 Fungsi UPPER dan LOWER

Perintah *UPPER* dan *LOWER* merupakan perintah SQL untuk mengubah teks menjadi huruf kapital atau huruf kecil. Perintah *UPPER* digunakan untuk mengubah semua karakter dalam sebuah string menjadi huruf kapital, sedangkan perintah *LOWER* digunakan untuk mengubah semua karakter dalam sebuah string menjadi huruf kecil. Berikut contoh kode perintah SQL *UPPER* dan *LOWER*:

```
SELECT UPPER(product_name) AS upper_case_name, LOWER(product_name) AS lower_case_name
FROM products;
```

Gambar 8.17. Coding *UPPER* dan *LOWER*

Pada gambar 8.17, kita menggunakan perintah *UPPER* dan *LOWER* untuk mengubah nilai kolom "*product_name*" pada tabel "*products*". Perintah *UPPER* akan mengubah semua karakter dalam "*product_name*" menjadi huruf kapital, sedangkan perintah *LOWER* akan mengubah semua karakter menjadi huruf kecil. Hasil perhitungan akan disimpan dengan alias "*upper_case_name*" dan "*lower_case_name*".

upper_case_name	lower_case_name
APPLE	apple
BANANA	banana
ORANGE	orange

Gambar 8.18. Hasil SQL dari *UPPER* dan *LOWER*

8.4 Fungsi Tanggal dan Waktu

Fungsi Tanggal dan Waktu pada SQL memainkan peran penting dalam pengelolaan untuk memanipulasi dan mengelola data terkait tanggal dan waktu dalam database. Dalam lingkungan SQL, ada beberapa fungsi yang dapat digunakan untuk mengakses dan memanipulasi informasi tanggal dan waktu. Salah satu fungsi yang sering digunakan adalah fungsi *DATE, TIME, TIMESTAMP* dan *DATEPART* Fungsi-fungsi ini sangat penting dalam SQL karena memungkinkan pengguna untuk mengelola, memanipulasi, dan menganalisis data tanggal dan waktu dengan lebih efisien. Dengan memahami dan menggunakan fungsi-fungsi ini, pengguna dapat mengoptimalkan penggunaan data tanggal dan waktu dalam database.

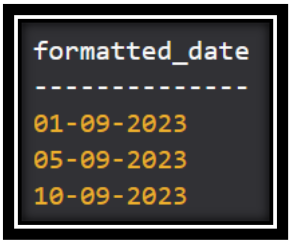
8.4.1 Fungsi DATE

Fungsi *DATE* pada SQL digunakan untuk mengambil atau memanipulasi bagian tanggal dari sebuah data dalam database. Fungsi ini biasanya digunakan untuk melakukan operasi seperti menampilkan tanggal, mengekstrak bagian tanggal seperti tahun atau bulan, atau melakukan perbandingan antara tanggal. Berikut contoh kode perintah SQL *DATE*:

```
SELECT DATE_FORMAT(order_date, '%d-%m-%Y') AS formatted_date
FROM orders;
```

Gambar 8.19. Coding *Date*

Pada gambar 8.19, kita menggunakan perintah *DATE_FORMAT* untuk mengubah format tanggal pada kolom "*order_date*" dalam tabel "*orders*". Parameter pertama adalah nilai tanggal yang akan diformat, dan parameter kedua adalah format yang diinginkan. Dalam contoh ini, format yang diinginkan adalah '%d-%m-%Y', yang akan menghasilkan tanggal dengan format "hari-bulan-tahun". Hasil perhitungan akan disimpan dengan alias "*formatted_date*".



Gambar 8.19. Hasil SQL dari *Date*

Hasil dari perintah *DATE_FORMAT* adalah tanggal yang diformat sesuai dengan format yang ditentukan. Misalnya, "01-09-2023" adalah hasil dari pemformatan tanggal "*order_date*" pada baris pertama. Perintah DATE sering digunakan dalam SQL untuk melakukan operasi terkait tanggal, seperti pengambilan bagian-bagian tertentu dari tanggal, perhitungan selisih tanggal, atau pengurutan berdasarkan tanggal.

8.4.2 Fungsi *TIME*

TIME merupakan sintak SQL yang digunakan untuk mengambil atau memanipulasi bagian waktu dari sebuah data dalam pada database. Fungsi ini memungkinkan kita menampilkan waktu seperti jam, menit, atau detik, atau melakukan perbandingan antara waktu. Berikut contoh kode perintah untuk memanipulasi data waktu (*time*) dalam database:

```
SELECT TIME_FORMAT(order_time, '%H:%i:%s') AS formatted_time
FROM orders;
```

Gambar 8.20. Coding *Time*

Pada gambar 8.20, kita menggunakan perintah *TIME_FORMAT* untuk mengubah format data waktu pada kolom "order_time" dalam tabel "*orders*". Parameter pertama adalah nilai data waktu yang akan diformat, dan parameter kedua adalah format yang diinginkan. Dalam contoh ini, format yang diinginkan adalah '%H:%i:%s', yang akan menghasilkan waktu dengan format "jam:menit:detik". Hasil perhitungan akan disimpan dengan alias "*formatted_time*".

Gambar 8.21. Hasil SQL dari *Time*

Hasil dari perintah *TIME_FORMAT* adalah waktu yang diformat sesuai dengan format yang ditentukan. Misalnya, "08:30:00" adalah hasil dari pemformatan data waktu pada baris pertama. Perintah *TIME* sering digunakan dalam SQL untuk melakukan operasi terkait data waktu, seperti pengambilan bagian-bagian tertentu dari data waktu, perhitungan selisih waktu, atau pengurutan berdasarkan data waktu.

8.4.3 Fungsi *TIMESTAMP*

TIMESTAMP pada SQL digunakan untuk menampilkan nilai datetime berdasarkan tanggal dan waktu saat ini. Fungsi ini menghasilkan tanggal, jam, menit, detik atau nilai timestamp dalam

format YYYY-MM-DD HH: MM: SS. Berikut contoh kode perintah untuk memanipulasi data waktu (*timestamp*) dalam database:

```
SELECT TIMESTAMPDIFF(MINUTE, created_at, NOW()) AS time_difference
FROM orders;
```

Gambar 8.22. Coding *Timestamp*

Pada gambar 8.22, kita menggunakan perintah *TIMESTAMPDIFF* untuk menghitung selisih waktu antara kolom "*created_at*" dalam tabel "*orders*" dengan waktu saat ini (*NOW*()). Parameter pertama adalah satuan waktu yang ingin dihitung, dalam contoh ini adalah MENIT (*MINUTE*). Parameter kedua adalah *timestamp* awal (*created_at*), dan parameter ketiga adalah timestamp akhir (*NOW*()). Hasil perhitungan akan disimpan dengan alias "*time_difference*".

```
time_difference
---------------
15
35
60
```

Gambar 8.23. Hasil SQL dari *Timestamp*

Hasil dari perintah *TIMESTAMPDIFF* adalah selisih waktu dalam satuan menit antara timestamp "created_at" pada setiap baris dengan waktu saat ini. Misalnya, "15" menunjukkan bahwa terdapat selisih waktu 15 menit antara waktu pembuatan pada baris pertama dengan waktu saat ini. Perintah *TIMESTAMP* sering digunakan dalam SQL untuk melacak dan memanipulasi data waktu dengan presisi yang lebih tinggi.

8.4.4 Fungsi *DATEPART*

DATEPART pada SQL digunakan untuk mengambil bagian tertentu dari sebuah tanggal, seperti tahun, bulan, hari, jam, menit, atau detik. Fungsi ini mengembalikan nilai bilangan bulat yang sesuai dengan bagian yang ditentukan dari tanggal yang diberikan. Berikut contoh kode perintah untuk memanipulasi data waktu (*datepart*) dalam database:

```
SELECT DATEPART(year, order_date) AS order_year,
       DATEPART(month, order_date) AS order_month,
       DATEPART(day, order_date) AS order_day
FROM orders;
```

Gambar 8.24. Coding *Datepart*

Pada gambar 8.24, kita menggunakan perintah *DATEPART* untuk mengambil bagian-bagian tertentu dari kolom "*order_date*" dalam tabel "*orders*". Parameter pertama adalah bagian yang ingin diambil, seperti "*year*" (tahun), "*month*" (bulan), atau "*day*" (hari). Parameter kedua adalah nilai tanggal yang ingin diambil bagian-bagiannya. Dalam contoh ini, kita mengambil tahun, bulan, dan hari dari "*order_date*". Hasil perhitungan akan disimpan dengan alias "*order_year*", "*order_month*", dan "*order_day*".

order_year	order_month	order_day
2023	9	12
2023	9	5
2023	9	1

Gambar 8.25. Hasil SQL dari *Datepart*

Hasil dari perintah *DATEPART* adalah menampilkan nilai-nilai spesifik dari tanggal pada kolom "*order_date*". Misalnya, pada

baris pertama, kita mendapatkan nilai tahun "2023", bulan "9", dan hari "12" dari tanggal "*order_date*". Perintah *DATEPART* digunakan dalam SQL untuk mengambil informasi yang spesifik dari tanggal atau waktu yang ada dalam database.

8.5 Fungsi Logika

Fungsi logika dalam SQL di dalam database adalah kemampuan untuk melakukan evaluasi dan manipulasi data berdasarkan kondisi logika yang diberikan. Fungsi logika dalam SQL memungkinkan kita untuk melakukan operasi logika seperti percabangan, penggabungan kondisi, dan pengaturan data berdasarkan kondisi yang ditentukan. Salah satu fungsi logika yang umum digunakan dalam SQL adalah fungsi IF(). Fungsi ini digunakan untuk mengevaluasi suatu kondisi logika dan mengembalikan nilai berdasarkan kondisi tersebut. Kita dapat menggunakan fungsi IF() untuk membuat percabangan dalam SQL, di mana tindakan yang diambil akan bergantung pada apakah kondisi logika terpenuhi atau tidak, Selain itu, SQL juga menyediakan fungsi logika lain seperti fungsi CASE(). Dengan menggunakan fungsi logika dalam SQL, kita dapat menjalankan operasi logika yang kompleks dan mengatur data berdasarkan kondisi yang ditentukan. Hal ini memungkinkan kita untuk melakukan analisis data yang lebih lanjut dan pengolahan yang lebih efisien di dalam database.

8.5.1 Fungsi *IF*

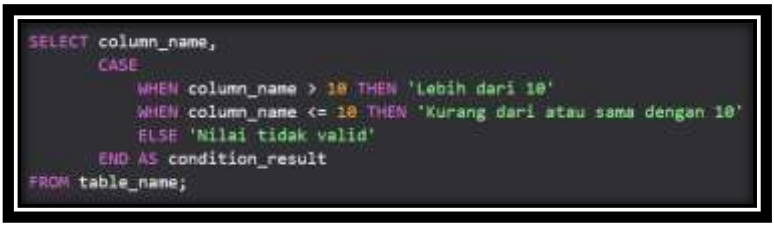
Perintah SQL IF digunakan dalam database untuk melakukan evaluasi kondisi dan mengembalikan nilai berdasarkan hasil evaluasi tersebut. Perintah ini memungkinkan kita untuk membuat logika kondisional dalam pernyataan SQL. Dengan menggunakan IF, kita dapat mengatur bagaimana data akan diolah atau dihasilkan berdasarkan kondisi yang ada, Berikut contoh kode IF:

Gambar 8.26. Coding *IF*

Pada gambar 8.26, kita menggunakan perintah *IF* untuk mengevaluasi kondisi apakah total_pembelian pelanggan lebih dari 1.000.000. Jika kondisi terpenuhi, maka akan mengembalikan string 'Pelanggan VIP', jika tidak terpenuhi, maka akan mengembalikan string 'Pelanggan Reguler'. Hasil evaluasi akan disimpan dalam kolom dengan alias "tipe_pelanggan". Perintah IF dalam SQL sangat berguna dalam melakukan evaluasi kondisi dan membuat logika kondisional yang kompleks. Hal ini memungkinkan kita untuk menghasilkan output yang berbeda berdasarkan kondisi yang ada dalam data kita.

8.5.2 Fungsi *CASE*

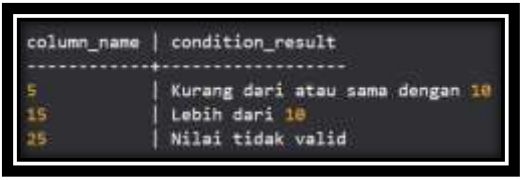
Perintah ini memungkinkan kita untuk membuat logika kondisional yang lebih kompleks dengan beberapa kondisi yang berbeda. Berikut adalah contoh coding dan penjelasan penggunaan perintah SQL *CASE*:



Gambar 8.27. Coding *CASE*

Pada gambar 8.27, kita menggunakan perintah *CASE* untuk mengevaluasi kondisi berdasarkan nilai dalam kolom "*column_name*". Jika nilai dalam kolom tersebut lebih besar dari 10,

maka akan mengembalikan string 'Lebih dari 10'. Jika nilai dalam kolom tersebut kurang dari atau sama dengan 10, maka akan mengembalikan string 'Kurang dari atau sama dengan 10'. Jika tidak ada kondisi yang terpenuhi, maka akan mengembalikan string 'Nilai tidak valid'. Hasil evaluasi akan disimpan dalam kolom dengan alias "*condition_result*".



Gambar 8.28. Hasil SQL dari *Case*

Hasil dari perintah *CASE* akan mengembalikan nilai berdasarkan hasil evaluasi kondisi pada kolom "*column_name*". Misalnya, pada baris pertama, nilai dalam kolom "*column_name*" adalah 5, yang kurang dari atau sama dengan 10, sehingga hasil evaluasi adalah 'Kurang dari atau sama dengan 10'. Jika nilai tidak memenuhi kondisi apa pun, maka akan mengembalikan string 'Nilai tidak valid'.

8.5.3 Fungsi *COALESCE*

Fungsi dari perintah SQL *COALESCE* dalam database adalah untuk mengembalikan nilai pertama yang tidak NULL dari beberapa ekspresi. Perintah ini digunakan ketika kita ingin mengganti nilai NULL dengan nilai alternatif yang dapat ditentukan. Berikut contoh coding penggunaan perintah SQL *COALESCE*:

```
SELECT column_name,
       COALESCE(column_name, 'Nilai Alternatif') AS result
FROM table_name;
```

Gambar 8.29. Hasil SQL dari *Coalesce*

Pada gambar 8.29, kita menggunakan perintah COALESCE untuk mengembalikan nilai pertama yang tidak NULL dari kolom "column_name". Jika nilai dalam kolom tersebut tidak NULL, maka akan mengembalikan nilai aslinya. Namun, jika nilai dalam kolom tersebut NULL, maka akan mengembalikan nilai alternatif yang ditentukan, dalam contoh ini adalah 'Nilai Alternatif'. Hasil evaluasi akan disimpan dalam kolom dengan alias "result".

```
column_name | result
------------+------------------
5           | 5
NULL        | Nilai Alternatif
15          | 15
```

Gambar 8.30. Hasil SQL dari *Coalesce*

Hasil dari perintah *COALESCE* akan mengembalikan nilai pertama yang tidak NULL dari kolom "*column_name*". Misalnya, pada baris pertama, nilai dalam kolom "*column_name*" adalah 5, yang bukan NULL, sehingga hasil evaluasi adalah 5. Pada baris kedua, nilai dalam kolom "column_name" adalah NULL, sehingga mengembalikan nilai alternatif 'Nilai Alternatif'. Pada baris ketiga, nilai dalam kolom "column_name" adalah 15, yang bukan NULL, sehingga hasil evaluasi adalah 15.

8.6 Kesimpulan

Pentingnya SQL dan fungsi-fungsi dalam database dalam pengolahan data tidak dapat diremehkan. SQL (*Structured Query Language*) adalah bahasa pemrograman khusus yang digunakan untuk mengelola, melakukan manipulasi data dalam database. Dengan menggunakan SQL pengguna dapat membuat, mengubah, dan menghapus tabel, serta mengatur hubungan antara tabel-tabel tersebut. SQL juga memungkinkan pengguna untuk mengambil data yang diperlukan dengan menggunakan query yang spesifik, yang mempermudah proses pencarian dan filtrasi data. Selain itu, fungsi-fungsi dalam SQL juga memainkan peran penting dalam pengolahan data. Fungsi-fungsi ini memungkinkan pengguna untuk melakukan operasi matematika, logika, dan manipulasi data lainnya. Misalnya, fungsi agregasi seperti *SUM(), COUNT()*, dan *AVG()* memungkinkan pengguna untuk melakukan perhitungan pada data dalam database. Kemudian terdapat fungsi string seperti *CONCAT* dan *LENGTH* memungkinkan pengguna untuk memanipulasi data string, sedangkan fungsi tanggal dan waktu memungkinkan pengguna untuk memanipulasi data terkait tanggal dan waktu.

Dengan menggunakan SQL dan fungsi-fungsi ini, pengguna dapat mengolah data dalam database dengan lebih efisien dan efektif. Mereka dapat melakukan analisis data, mengambil informasi yang diperlukan, dan menghasilkan laporan yang relevan. Dengan demikian, SQL dan fungsi-fungsi dalam database berperan penting dalam pengolahan data yang akurat, terstruktur, dan dapat diandalkan.

DAFTAR PUSTAKA

Fikry, M., Yusra, Y., & Hidayat, T. 2018. Pembangkitan Formulir Web Berdasarkan Metadata SQL dan Spesifikasi W3C. *Jurnal CoreIT: Jurnal Hasil Penelitian Ilmu Komputer Dan Teknologi Informasi*, *3*(2), 63. doi:10.24014/coreit. v3i2.4417

Data Analysis Expressions (DAX) di Power Pivot (no date). Available at: https://support.microsoft.com/id-id/office/data-analysis-expressions-dax-di-power-pivot-bab3fbe3-2385-485a-980b-5f64d3b0f730.

Data Analysis Expressions (DAX) di Power Pivot (no date). Available at: https://support.microsoft.com/id-id/office/data-analysis-expressions-dax-di-power-pivot-bab3fbe3-2385-485a-980b-5f64d3b0f730.

BAB 9
SQL SUB QUERY DAN JOIN

Oleh Wirawan Istiono

9.1 Dasar Subquery

Subkueri yang terkandung dalam ekspresi dievaluasi sebagai subkueri skalar. Daftar SELECT dan klausa WHERE sering menggunakan subkueri skalar. Subkueri skalar diperlukan untuk mengidentifikasi satu kolom. Kesalahan analisis akan dihasilkan dari upaya memilih beberapa kolom. Daftar SELECT yang berisi satu ekspresi adalah metode yang paling efisien untuk memilih satu kolom. Ungkapan ini menentukan sifat dari hasil skalar subquery (Mode.com, 2019).

Subquery adalah pernyataan SELECT yang disematkan di dalam pernyataan T-SQL lainnya. Pernyataan SELECT subquery akan mengembalikan kumpulan hasil jika dijalankan secara independen dari pernyataan T-SQL yang memuatnya. Dengan kata lain, pernyataan SELECT subquery tidak tergantung pada pernyataan di mana ia bersarang. Pernyataan SELECT subquery dapat mengembalikan sejumlah nilai dan dapat ditemukan dalam daftar kolom pernyataan SELECT, klausa FROM, GROUP BY, HAVING, dan/atau ORDER BY dari pernyataan T-SQL. Subquery juga dapat digunakan sebagai parameter fungsi. Subquery pada dasarnya dapat digunakan di mana saja ekspresi dapat digunakan (Larsen, 2015).

Subkueri, juga dikenal sebagai kueri dalam atau kueri bersarang, mengacu pada kueri yang dimuat di dalam kueri SQL lain dan disematkan di dalam klausa WHERE. Sebuah subquery mengacu pada pernyataan SELECT yang dilampirkan dalam klausa dari pernyataan SQL lainnya. Pemanfaatan kueri terbukti sangat

menguntungkan dalam proses pengambilan baris tertentu dari database, bergantung pada kondisi yang didasarkan pada data dalam tabel yang sama atau berbeda. Subquery digunakan untuk mengambil data yang kemudian akan digunakan dalam query utama sebagai syarat untuk lebih membatasi data yang akan diambil. Subquery dapat diposisikan dalam berbagai klausa SQL, yaitu klausa WHERE, klausa HAVING, dan klausa FROM (Harikamoluguram27, 2022).

Alternatifnya, SELECT AS STRUCT dapat digunakan untuk mendefinisikan subquery yang memilih satu nilai STRUKTUR yang bidangnya ditentukan oleh satu atau lebih ekspresi.

Jika subquery mengembalikan satu baris, nilai singular itu adalah hasil subquery skalar. Jika subquery tidak mengembalikan baris, hasilnya adalah NULL. Kueri gagal dengan kesalahan waktu proses jika subkueri mengembalikan beberapa baris (Mode.com, 2019).

Subkueri (juga dikenal sebagai kueri dalam atau kueri bersarang) adalah metode untuk melakukan operasi multi-langkah. Misalnya, jika Anda ingin mengambil jumlah dari beberapa kolom dan kemudian menghitung rata-ratanya, Anda perlu melakukan setiap agregasi dalam fase terpisah.

Dimungkinkan untuk menggunakan subkueri di beberapa lokasi dalam kueri, tetapi paling mudah untuk memulai dengan pernyataan DARI. Berikut adalah contoh sederhana dari subquery:

```
SELECT sub.*
 FROM (
    SELECT *
     FROM bookcp.sample_2023_08
     WHERE day_of_week = 'Monday'
   ) sub
WHERE sub.notes = 'NONE'
```

Subkueri harus memiliki nama, yang disisipkan setelah tanda kurung dengan cara yang sama seperti alias untuk tabel normal. Dalam hal ini, kami menggunakan istilah "sub".

Catatan singkat tentang pemformatan: saat menggunakan subkueri, sangat penting untuk menyediakan metode bagi pembaca untuk menentukan bagian mana dari kueri yang akan dieksekusi secara bersamaan. Sebagian besar individu melakukannya dengan mengindentasi subquery. Dalam tutorial ini, contoh diindentasi sampai ke tanda kurung. Ini tidak praktis jika Anda menyarangkan banyak subkueri, jadi umumnya hanya membuat indentasi dua atau tiga spasi.

Contoh sebelumnya dan soal latihan tidak memerlukan subkueri karena mereka mengatasi masalah yang juga dapat diselesaikan dengan menambahkan beberapa kondisi ke klausa WHERE. Bagian berikut memberikan contoh masalah yang subkuerinya merupakan solusi terbaik atau satu-satunya.

Sebuah subquery juga disebut sebagai inner query atau inner select, sedangkan pernyataan yang berisi subquery disebut sebagai outer query atau outer select (Elhemali*et al.*, 2007).

Banyak pernyataan Transact-SQL yang berisi subkueri juga dapat ditulis sebagai gabungan. Hanya subkueri yang dapat digunakan untuk mengajukan pertanyaan tambahan. Di Transact-SQL, biasanya tidak ada perbedaan kinerja antara pernyataan yang berisi subquery dan pernyataan yang setara secara semantik tanpa subquery. Lihat pemrosesan pernyataan SQL untuk informasi arsitektur tentang cara SQL Server memproses kueri. Namun, dalam situasi tertentu di mana keberadaan harus diverifikasi, JOIN memberikan kinerja yang unggul. Jika tidak, kueri bersarang harus dijalankan untuk setiap hasil kueri luar untuk memastikan bahwa duplikat dihapus. Dalam situasi seperti itu, pendekatan JOIN akan menghasilkan hasil yang unggul (MSFT, 2023).

9.2 Dasar Subquery

Komponen berikut terdiri dari subquery yang terkandung dalam pernyataan SELECT luar (MSFT, 2023):

1. Kueri SELECT standar yang berisi elemen daftar pilih standar.
2. Klausa FROM konvensional yang mencakup nama satu atau beberapa tabel atau tampilan.
3. Klausa WHERE diskresioner.
4. Kemungkinan klausa GROUP BY.
5. Kemungkinan klausa HAVING.

Selalu sertakan kueri SELECT dari subkueri dalam tanda kurung. Itu tidak bisa berisi klausa COMPUTE atau FOR BROWSE, dan hanya bisa berisi klausa ORDER BY jika klausa TOP juga ditentukan.

Sebuah subquery mungkin bersarang di dalam klausa WHERE atau HAVING dari pernyataan SELECT, INSERT, UPDATE, atau DELETE eksterior, serta di dalam subquery lain. Ada dukungan hingga 32 tingkat penumpukan, meskipun batas sebenarnya bergantung pada memori yang tersedia dan kerumitan ekspresi lain dalam kueri. Kueri individu mungkin tidak mendukung hingga 32 tingkat penyarangan. Di mana pun ekspresi dapat digunakan, subkueri yang mengembalikan nilai singular juga dapat digunakan (MSFT, 2023).

Jika tabel hanya muncul dalam subkueri dan bukan dalam kueri luar, kolomnya tidak dapat disertakan dalam output (daftar pilihan kueri luar).

Pernyataan yang berisi subquery biasanya mengikuti salah satu format berikut:

1. WHERE ekspresi [NOT] IN (subkueri)
2. WHERE ekspresi operator_perbandingan [ANY | ALL] (subkueri)
3. WHERE [NOT] EXIST (subkueri)

Dalam pernyataan Transact-SQL tertentu, subkueri dapat dievaluasi seolah-olah merupakan kueri independen. Secara konseptual, hasil subquery dimasukkan ke dalam outer query; namun, tidak selalu demikian cara SQL Server memproses pernyataan Transact-SQL dengan subkueri.

Ada tiga kategori subquery mendasar. Orang-orang yang:

1. Lakukan operasi pada daftar yang diperkenalkan dengan IN atau yang dimodifikasi oleh ANY atau ALL menggunakan operator pembanding.
2. Harus menghasilkan nilai tunggal dan diperkenalkan dengan operator pembanding yang tidak dimodifikasi.
3. Tes keberadaan yang dimulai dengan EXISTS.

9.3 Aturan subkueri

Subquery diatur oleh batasan berikut (MSFT, 2023):

1. Daftar pilih dari subkueri yang diperkenalkan dengan operator pembanding hanya boleh berisi satu ekspresi atau nama kolom (dengan pengecualian operator EXISTS dan IN, yang masing-masing dijalankan pada SELECT * dan daftar).
2. Jika klausa WHERE kueri eksterior menyertakan nama kolom, itu harus kompatibel dengan kolom daftar pilih subkueri.
3. Tipe data nteks, teks, dan gambar tidak dapat digunakan dalam daftar pilih subkueri.
4. Subkueri yang diperkenalkan oleh operator perbandingan yang tidak dimodifikasi (yang tidak diikuti oleh kata kunci ANY atau ALL) tidak dapat berisi klausa GROUP BY dan HAVING karena harus mengembalikan nilai singular.
5. Kata kunci DISTINCT tidak kompatibel dengan subkueri yang berisi klausa GROUP BY.
6. Klausa COMPUTE dan INTO tidak dapat ditentukan.
7. ORDER BY hanya dapat ditentukan bersama dengan TOP.
8. Tampilan yang dihasilkan oleh subquery tidak dapat dimodifikasi.

9. Dengan konvensi, daftar pilih subquery yang diperkenalkan dengan EXISTS berisi tanda bintang (*) daripada nama kolom tunggal. Kriteria untuk subkueri yang diperkenalkan dengan EXISTS identik dengan kriteria untuk daftar pilihan standar, karena EXISTS membuat pengujian keberadaan dan mengembalikan TRUE atau FALSE alih-alih data.

9.4 Jenis subkueri

Beberapa lokasi mengizinkan spesifikasi subkueri (MSFT, 2023):

1. Memiliki nama.
2. Dengan IN atau NOT IN. Lihat Subkueri dengan IN dan Subkueri dengan NOT IN untuk informasi tambahan.
3. Dalam pernyataan INSERT, UPDATE, dan DELETE. Rujuk ke Subkueri dalam Pernyataan UPDATE, DELETE, dan INSERT untuk informasi tambahan.
4. berisi operator pembanding. Lihat Subkueri dengan Operator Perbandingan untuk informasi lebih lanjut.
5. Dengan ANY, SOME, dan EVERY. Lihat Operator Perbandingan yang Dimodifikasi oleh ANY, SOME, atau ALL untuk informasi tambahan.
6. Dengan IS NOT DIFFERENT FROM. Lihat IS [NOT] DISTINCT FROM (Transact-SQL) untuk informasi lebih lanjut.
7. Dengan operator EXISTS dan NOT EXISTS. Lihat Subqueries dengan EXISTS dan Subqueries dengan NOT EXISTS untuk informasi tambahan.
8. Melainkan ekspresi. Lihat Subkueri yang digunakan sebagai pengganti Ekspresi untuk informasi tambahan.

9.5 Penggunaan Subquery dalam Daftar Kolom Pernyataan SELECT

Misalkan Anda ingin melihat OrderID dan OrderDate untuk pesanan terbaru yang dikirim ke Paris. Misalkan Anda juga ingin melihat OrderDate untuk pesanan terbaru dikirimkan, terlepas dari

ShipCity. Selain itu, Anda ingin menghitung jumlah hari yang memisahkan dua TanggalPesanan yang berbeda. Inilah pernyataan T-SQL SELECT saya untuk tujuan ini (Larsen, 2015):

```
select top 1 OrderId,convert(char(10),
    OrderDate,121) Last_Paris_Order,
   (select convert(char(10),max(OrderDate),121) from
       Northwind.dbo.Orders) Last_OrderDate,
    datediff(dd,OrderDate,
   (select Max(OrderDate)from Northwind.dbo.Orders)) Day_Diff
from Northwind.dbo.Orders
where ShipCity = 'Paris'
order by OrderDate desc
```

Kode sebelumnya mencakup dua subkueri. Subquery pertama mengambil OrderDate untuk pesanan terbaru yang dikirimkan, terlepas dari ShipCity, dan subquery kedua menghitung jumlah hari antara dua OrderDates. Di sini, saya menggunakan subquery pertama untuk mengambil nilai kolom untuk set hasil akhir. Subquery kedua digunakan sebagai parameter fungsi. Subquery ini menyediakan fungsi DATEDIFF dengan tanggal "max(OrderDate)".

9.6 Penggunaan Subquery dalam klausa WHERE

Dengan mengontrol record mana yang memenuhi kondisi klausa WHERE, subquery dapat digunakan untuk memodifikasi record yang dikembalikan oleh SELECT. Dalam hal ini, hasil subquery akan digunakan sebagai bagian dari kondisi klausa WHERE. Berikut ilustrasinya (Larsen, 2015):

```
SELECT distinct country from Northwind.dbo.Customers
where country not in (select distinct country from
Northwind.dbo.Suppliers)
```

Berikut adalah daftar negara tempat tinggal pelanggan, tetapi tidak ada pemasok di negara tersebut. Saya berasumsi bahwa jika Anda ingin memberikan waktu pengiriman yang lebih cepat kepada pelanggan Anda, Anda akan mencari pemasok tambahan di negara-negara tersebut. Misalkan sebuah perusahaan ingin terlibat dalam pemasaran yang ditargetkan. Pemasaran tertarget ini akan menjangkau konsumen di negara dengan volume pemesanan terendah. Pemasaran bertarget ini dimaksudkan untuk meningkatkan penjualan secara keseluruhan di negara yang ditargetkan. Berikut adalah contoh subquery yang mengembalikan informasi kontak pelanggan untuk negara dengan pesanan paling sedikit (Larsen, 2015):

```
select Country,CompanyName, ContactName, ContactTitle, Phone
from Northwind.dbo.Customers
where country =
  (select top 1 country
      from Northwind.dbo.Customers C
              join
         Northwind.dbo.Orders O
         on C.CustomerId = O.CustomerID
   group by country
   order by count(*))
```

Di sini, saya telah membuat subkueri yang menggabungkan tabel Pelanggan dan Pesanan untuk memastikan jumlah total pesanan untuk setiap negara. Klausul "TOP 1" digunakan untuk mengembalikan negara dengan jumlah pesanan terkecil. Negara dengan jumlah pesanan terkecil kemudian digunakan dalam klausa WHERE untuk menentukan Informasi Pelanggan mana yang ditampilkan.

9.7 Penggunaan Subquery dalam klausa FROM

Biasanya, klausa FROM mengidentifikasi tabel yang digunakan oleh pernyataan T-SQL. Setiap tabel yang ditentukan dalam klausa FROM dapat dilihat sebagai kumpulan record. Karena subquery hanyalah kumpulan record, subquery dapat digunakan dalam klausa FROM seperti halnya tabel. Berikut ini menunjukkan penggunaan subkueri dalam klausa FROM dari pernyataan SELECT (Larsen, 2015).

```
select au_lname, au_fname, title from
   (select au_lname, au_fname, au_id from pubs.dbo.authors
      where state = 'CA')  as a
            join
   pubs.dbo.titleauthor ta on a.au_id=ta.au_id
            join
   pubs.dbo.titles t on ta.title_id = t.title_id
```

Di sini, saya telah menggunakan subkueri untuk memilih hanya informasi rekaman penulis jika kolom status dari rekaman penulis berisi nilai "CA." Saya telah memberikan set hasil dari subquery ini penunjukan tabel "a." Seperti yang saya lakukan di klausa "ON" dari kriteria "JOIN", saya kemudian dapat menggunakan alias ini di tempat lain dalam pernyataan T-SQL untuk merujuk ke kolom dari subquery dengan mengawalinya dengan "a". Menggunakan subkueri dalam klausa FROM terkadang dapat mengurangi ukuran himpunan yang harus digabungkan. Mengurangi jumlah rekaman yang akan digabungkan meningkatkan kemanjuran menggabungkan baris dan dengan demikian mempercepat eksekusi kueri. Saya menggunakan subquery dalam klausa FROM dari pernyataan UPDATE seperti yang ditunjukkan di bawah ini.

```
set nocount on
create table x(i int identity,
        a char(1))
insert into x values (‘a’)
insert into x values (‘b’)
insert into x values (‘c’)
insert into x values (‘d’)
select * from x
update x
set a = b.a
from (select max(a) as a from x) b
where i > 2
select * from x
drop table x
```

Saya telah membuat tabel berjudul "x" dengan empat bidang. Kemudian saya memperbarui baris di mana "i" lebih besar dari 2 dengan nilai maksimum kolom "a". Saya menggunakan subquery di dalam klausa FROM dari pernyataan UPDATE untuk menentukan nilai maksimum kolom "a."

9.8 Penggunaan Subquery dalam klausa HAVING

Dalam ilustrasi ini, subkueri digunakan untuk memastikan jumlah buku yang diterbitkan oleh penerbit tertentu, tidak termasuk penerbit yang berlokasi di negara bagian California. Untuk mencapai ini, subquery digunakan dalam klausa HAVING. Disajikan di bawah ini adalah kode yang telah saya kembangkan (Larsen, 2015):

```
Select pub_name, count(*) bookcnt
from pubs.dbo.titles t
      join
   pubs.dbo.publishers p
```

```
   on t.pub_id = p.pub_id
group by pub_name
having p.pub_name in
 (select pub_name from pubs.dbo.publishers where state <> 'CA')
```

Subquery menghasilkan nilai pub_name untuk penayang yang nilai statusnya tidak sama dengan "CA". Kondisi HAVING selanjutnya memverifikasi apakah pub_name ada di dalam set yang diperoleh dari subquery.

9.9 Menggunakan subkueri untuk menggabungkan dalam beberapa tahap

Bagaimana jika Anda ingin menentukan jumlah insiden yang dilaporkan setiap hari dalam seminggu? Bagaimana jika ingin mengetahui jumlah rata-rata kejadian yang terjadi pada hari Jumat di bulan Desember? Bagaimana dengan Januari? Proses ini terdiri dari dua langkah: menghitung jumlah insiden setiap harinya (inner query) dan menghitung rata-rata bulanan (*outer query*).

```
SELECT LEFT(sub.date, 2) AS cleaned_month,
    sub.day_of_week,
    AVG(sub.incidents) AS average_incidents
 FROM (
     SELECT day_of_week,
          date,
          COUNT(incidnt_num) AS incidents
       FROM tutorial.sf_crime_incidents_2014_01
      GROUP BY 1,2
    ) sub
GROUP BY 1,2
ORDER BY 1,2
```

Jika seseorang menemui kesulitan dalam memahami situasinya, disarankan untuk mengeksekusi inner query secara terpisah untuk mendapatkan pemahaman yang lebih baik tentang tampilan hasilnya. Biasanya, akan lebih mudah untuk memulai proses penulisan dengan merumuskan dan menyempurnakan pertanyaan batin sampai hasilnya selaras dengan pemahaman seseorang, sebelum melanjutkan ke pertanyaan luar.

9.10 Subkueri dalam logika kondisional

Subkueri dapat digunakan dalam logika bersyarat, khususnya dalam kaitannya dengan pernyataan WHERE, JOIN/ON, atau CASE. Kueri yang disediakan mengambil semua entri dari kumpulan data mulai dari tanggal paling awal. Namun, karena pemformatan kolom tanggal yang tidak memadai, kueri sebenarnya mengembalikan nilai yang diurutkan terlebih dahulu dalam urutan abjad.

```
SELECT *
 FROM tutorial.sf_crime_incidents_2014_01
WHERE Date = (SELECT MIN(date)
          FROM tutorial.sf_crime_incidents_2014_01
        )
```

Kueri di atas valid karena subkueri mengembalikan satu sel. Sebagian besar logika kondisional kompatibel dengan subkueri yang mengembalikan hasil sel tunggal. Namun, IN adalah satu-satunya bentuk logika kondisional yang valid saat kueri dalam mengembalikan beberapa hasil.

```
SELECT *
 FROM tutorial.sf_crime_incidents_2014_01
WHERE Date IN (SELECT date
          FROM tutorial.sf_crime_incidents_2014_01
          ORDER BY date
          LIMIT 5
         )
```

Penting untuk diperhatikan bahwa penyertaan alias tidak diperlukan saat membuat subquery dalam ekspresi bersyarat. Hal ini terjadi karena subquery ditafsirkan sebagai nilai tunggal atau kumpulan nilai dalam kasus operator IN, bukannya dianggap sebagai tabel.

9.11 Apa Joins di SQL??

JOINS adalah klausa SQL yang memungkinkan Anda menghubungkan dua tabel atau lebih, memungkinkan Anda mengambil data dari semua tabel yang digabungkan. Tabel digabungkan berdasarkan kolom dengan nilai identik di kedua (atau semua) tabel. Misalnya, jika tabel penulis berisi kolom id, Anda juga dapat menyertakan kolom author_id di tabel buku dan seri. Ketiga kolom ini akan berisi nilai yang identik (yaitu, nomor ID penulis) dan dapat digunakan untuk menggabungkan data dari ketiga tabel (Babic, 2021).

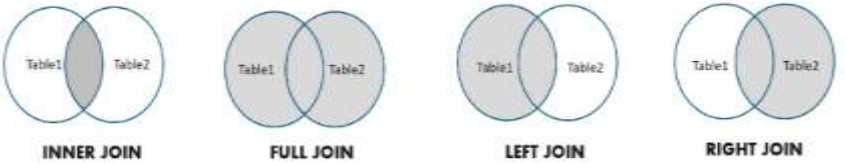
Perintah JOINS dalam SQL digunakan untuk menggabungkan catatan dari dua atau lebih tabel berdasarkan kolom umum. Saat mencoba mengekstrak data dari tabel dengan hubungan satu-ke-banyak atau banyak-ke-banyak, mereka paling sering digunakan (Kappagantula, 2023).

Operasi JOIN adalah jenis kueri basis data yang menggabungkan data dari banyak tabel. Operasi JOIN dijalankan ketika ada banyak tabel yang ditentukan dalam klausa FROM kueri.

Pernyataan SELECT dalam kueri memiliki kemampuan untuk memilih dan mengambil kolom dari salah satu tabel yang tersedia. Jika kondisi bergabung dihilangkan atau dianggap tidak valid, produk Cartesian dihasilkan. Untuk menghindari kesalahpahaman, penting untuk mengkualifikasikan kolom dengan tabel atau nama alias tabel masing-masing dalam kueri jika ada dua tabel yang memiliki nama kolom yang sama. Sebagian besar kueri JOIN biasanya memiliki minimal satu syarat JOIN, yang dapat ditemukan di klausa FROM atau klausa WHERE (Harikamoluguram27, 2022).

9.12 Tipe JOINS SQL

Ada empat jenis JOIN yang harus Anda pahami. Berikut ini adalah (Kappagantula, 2023):
FULL JOIN, INNER JOIN, LEFT JOIN, RIGHT JOIN
Silakan lihat gambar di bawah ini.



Gambar 9.1. Ilustrasi Jenis SQL JOINS
(Sumber: Kappagantula, 2023)

Mari kita periksa masing-masing secara individual. Untuk meningkatkan pemahaman pengertian ini, tiga tabel berikutnya akan diperiksa untuk mendemonstrasikan proses melakukan operasi Join pada tabel-tabel tersebut.

Tabel 11.1. Tabel Employee

EmpID	Nama penerima	Nama Perusahaan	Usia	EmailID	No Telepon	Alamat
1	Viore	Komala	22	vardy@abc.com	9876543210	Jakarta
2	Henny	Sinta	32	Henny@abc.com	9977554422	Tangerang
3	Amanda	Sherleen	24	Amanda@abc.com	9977555121	Malang
4	Herman	Sinta	25	Herman@abc.com	9876545666	Medan
5	Susan	Karen	26	Susan@abc.com	9544567777	Bekasi

Tabel 11.2. Tabel Project

ID Proyek	EmpID	ID Klien	Nama Proyek	Tanggal Mulai Proyek
111	1	3	Project1	2019-04-21
222	2	1	Project2	2019-02-12
333	3	5	Project3	2019-01-10
444	3	2	Project4	2019-04-16
555	5	4	Project5	2019-05-23
666	9	1	Project6	2019-01-12
777	7	2	Project7	2019-07-25
888	8	3	Project8	2019-08-20

Tabel 11.3. Tabel Client

ID Klien	ClientFname	ClientLname	Usia	ClientEmailID	No Telepon	Alamat	EmpID
1	Susan	Smith	30	susan@adn.com	9765411231	Malang	3
2	Mois	Ali	27	mois@jsq.com	9876543561	Malang	3
3	Soma	Paul	22	soma@wja.com	9966332211	Jakarta	1

ID Klien	ClientFname	ClientLname	Usia	ClientEmailID	No Telepon	Alamat	EmpID
4	Zainab	Daginawala	40	zainab@qkq.com	9955884422	Bekasi	5
5	Bhaskar	Reddy	32	bhaskar@xyz.com	9636963269	Tangerang	2

9.12.1 Inner Join

Jenis JOIN ini mengembalikan rekaman yang memiliki nilai yang cocok di kedua tabel. Jadi, jika Anda melakukan operasi JOIN INNER antara tabel Karyawan dan tabel Proyek, semua tupel yang memiliki nilai yang cocok di kedua tabel akan diberikan sebagai output.

```
SELECT Employee.EmpID, Employee.EmpFname, Employee.EmpLname, Projects.ProjectID, Projects.ProjectName
FROM Employee
```

Tabel 11.4. Tabel Hasil Inner Join

EmpID	EmpFname	EmpLname	ProjectID	ProjectName
1	Viore	Komala	111	Project1
2	Henny	Sinta	222	Project2
3	Amanda	Sherleen	333	Project3
4	Amanda	Sherleen	444	Project4
5	Susan	Karen	555	Project5

9.12.2 Full Joins

Gabung Penuh, juga dikenal sebagai JOINS Luar Penuh, mengambil semua entri yang memiliki kecocokan di tabel kiri (Tabel 11.1) atau tabel kanan (Tabel 11.2).

```
SELECT Employee.EmpFname, Employee.EmpLname,
Projects.ProjectID
FROM Employee
FULL JOIN Projects
ON Employee.EmpID = Projects.EmpID;
```

Tabel 11.5. Tabel Hasil Full Join

EmpFname	EmpLname	ProjectID
Viore	Komala	111
Henny	Sinta	222
Amanda	Sherleen	333
Amanda	Sherleen	444
Herman	Sinta	NULL
Susan	Karen	555
NULL	NULL	666
NULL	NULL	777
NULL	NULL	888

9.12.3 Right Joins

RIGHT JOIN, juga dikenal sebagai RIGHT OUTER JOIN, mengambil semua rekaman dari tabel kanan dan menyertakan semua rekaman dari tabel kiri yang memenuhi kondisi tertentu. Jika tidak ada entri yang cocok di tabel kiri, keluaran atau rangkaian hasil akan menyertakan nilai NULL.

```
PILIH Employee.EmpFname, Employee.EmpLname,
Projects.ProjectID, Projects.ProjectName
DARI Karyawan
GABUNG KANAN
ON Employee.EmpID = Projects.EmpID;
```

Tabel 11.6. Tabel Hasil Right Join

EmpFname	EmpLname	ProjectID	ProjectName
Viore	Komala	111	Project1
Henny	Sinta	222	Project2
Amanda	Sherleen	333	Project3
Amanda	Sherleen	444	Project4
Susan	Karen	555	Project5
NULL	NULL	666	Project6
NULL	NULL	777	Project7
NULL	NULL	888	Project8

9.13 Kelebihan dan Kekurangan SQL Join dan Subquery

Saat ada persyaratan untuk menggabungkan data dari berbagai tabel di SQL, ada dua pendekatan utama yang tersedia: subkueri dan gabungan. Kedua metode dapat menghasilkan hasil yang serupa, namun memiliki kelebihan dan kekurangan yang berbeda. Artikel ini bertujuan untuk memberikan pemahaman tentang subquery dan join, mekanismenya, dan kriteria untuk menentukan pemanfaatannya yang tepat dalam upaya pengembangan database (Harikamoluguram27, 2022).

9.13.1 Keuntungan JOIN

1. Salah satu manfaat memanfaatkan operasi JOIN adalah kemampuannya untuk beroperasi dengan kecepatan yang lebih cepat.
2. Dalam banyak kasus, waktu eksekusi kueri yang menggunakan JOIN biasanya lebih cepat dibandingkan dengan subkueri.
3. Pemanfaatan JOIN dalam operasi basis data memungkinkan pengurangan beban komputasi dengan menggabungkan

banyak kueri menjadi kueri JOIN tunggal. Ini menyiratkan bahwa pengguna dapat mengoptimalkan pemanfaatan fungsionalitas database seperti mencari, memfilter, menyortir, dan sebagainya.

9.13.2 Kerugian JOINS

1. Salah satu kelemahan yang terkait dengan penggunaan JOIN adalah bahwa mereka relatif kurang dapat dipahami dibandingkan dengan subkueri.
2. Dimasukkannya lebih banyak JOIN dalam kueri memerlukan peningkatan beban kerja di server basis data, yang menghasilkan prosedur pengambilan data yang lebih memakan waktu. Adanya banyak tipe join dapat menimbulkan kebingungan mengenai pemilihan tipe join yang sesuai untuk mencapai set hasil yang diinginkan.
3. Pemanfaatan JOIN tidak dapat dihindari saat mengekstraksi data dari database yang dinormalisasi. Namun, sangat penting untuk mengeksekusi JOIN secara akurat, karena JOIN yang salah dapat menyebabkan penurunan kinerja yang signifikan dan menghasilkan hasil kueri yang salah.

9.13.3 Keunggulan Subquery

1. Subkueri digunakan untuk mempartisi kueri kompleks menjadi komponen yang berbeda, sehingga memungkinkan dekonstruksi kueri kompleks menjadi urutan operasi logis.
2. Pemahaman materi difasilitasi, dan tugas menjaga kode juga disederhanakan.
3. Pemanfaatan subkueri memungkinkan penggabungan hasil dari kueri sebelumnya ke dalam kueri utama.
4. Dalam kasus tertentu, subkueri memiliki potensi untuk berfungsi sebagai alternatif untuk operasi JOIN dan JOIN yang rumit.

9.13.4 Kelemahan Subquery

1. Proses pengoptimalan untuk MYSQL lebih baik dikembangkan untuk JOIN dibandingkan dengan subkueri. Akibatnya, dalam banyak kasus, kueri yang menyertakan subkueri dapat dijalankan dengan lebih efisien dengan mengulanginya sebagai operasi gabungan.
2. Tidak mungkin melakukan modifikasi tabel dan pemilihan tabel dalam subquery dalam pernyataan SQL yang sama.

9.13.5 Subquery vs join: kinerja

Performa adalah pertimbangan penting saat memutuskan antara subkueri dan gabungan. Dalam konteks umum, dapat dikatakan bahwa JOIN cenderung menunjukkan kinerja yang unggul dibandingkan dengan subkueri karena kemampuannya untuk memanfaatkan indeks dan menerapkan berbagai strategi pengoptimalan. Sebaliknya, subkueri mungkin memerlukan sumber daya dan memori komputasi tambahan, terutama ketika mereka menghasilkan hasil yang substansial atau rumit. Namun demikian, penting untuk dicatat bahwa pernyataan tersebut tidak berlaku dalam semua skenario. Performa database dapat bervariasi berdasarkan beberapa faktor seperti mesin database yang digunakan, jumlah data, kompleksitas kueri, dan keberadaan indeks.

9.13.6 Subquery vs join: keterbacaan

Keterbacaan adalah aspek tambahan yang harus diperhitungkan saat membuat keputusan antara subkueri dan gabungan. Konsep keterbacaan berkaitan dengan tingkat kemudahan kode kueri dapat dipahami dan dipelihara. Secara umum, dapat dikatakan bahwa subquery lebih kondusif untuk keterbacaan dibandingkan dengan join, karena subquery menawarkan cara yang lebih mudah untuk mengekspresikan logika yang rumit. Subkueri memiliki manfaat tambahan untuk

mengurangi duplikasi dan redundansi melalui pemanfaatan kueri yang sama dalam banyak hal. Sebaliknya, JOIN mungkin memerlukan kode dan ketentuan tambahan, sehingga berpotensi memperumit kueri dan menjadikannya lebih menantang untuk dipahami dan dipecahkan.

9.13.7 Subquery vs join: fleksibilitas

Fleksibilitas adalah aspek tambahan yang harus diperhitungkan saat membuat keputusan antara subquery dan join. Fleksibilitas berkaitan dengan tingkat kemudahan query dapat dimodifikasi dan disesuaikan untuk mengakomodasi berbagai kondisi dan tuntutan. Secara umum, subkueri menunjukkan fleksibilitas yang lebih besar dibandingkan dengan gabungan, karena mereka memiliki kemampuan untuk menangani beragam jenis dan proses data. Subkueri dapat digunakan dalam konteks di mana penggabungan dilarang, seperti dalam klausa SELECT, HAVING, atau ORDER BY. Sebaliknya, JOIN mungkin memiliki beberapa kendala dan batasan, bergantung pada jenis JOIN tertentu yang digunakan dan mesin basis data yang mendasarinya.

9.14 Kesimpulan untuk Join dan Subquery

Salah satu kendala yang ditemui dalam proses pembuatan query SQL melibatkan proses pengambilan keputusan pemilihan antara penggunaan subquery atau operasi JOIN. Ada banyak skenario di mana operasi JOINS terbukti menjadi pendekatan yang paling optimal, sementara dalam kasus lain, subkueri menunjukkan kemanjuran yang unggul (Piesco, 2021).

Subkueri digunakan dalam kueri SQL yang rumit. Biasanya, kueri luar disertai dengan satu atau beberapa subkueri yang bersarang di dalam kueri luar.

Subquery dapat diklasifikasikan sebagai sederhana atau berkorelasi. Subkueri dasar tidak tergantung pada kolom di kueri

luar, sedangkan subkueri yang dikorelasikan membuat referensi ke data dari kueri luar (Piesco, 2021).

Klausa JOINS tidak mencakup kueri tambahan. Operasi SQL JOIN menetapkan hubungan antara dua atau lebih tabel dan mengambil data dari tabel tersebut untuk membentuk kumpulan hasil terpadu. Tujuan utama penggunaannya adalah untuk memfasilitasi penggabungan tabel melalui pemanfaatan primary dan foreign key (Oktavia dan Sujarwo, 2014).

Baik subkueri maupun JOINS dapat digunakan dalam kueri canggih untuk mengambil data dari beberapa tabel; namun, mereka menggunakan metodologi yang berbeda untuk mencapai tujuan ini. Dalam kasus tertentu, individu mungkin dihadapkan pada pilihan di antara dua alternatif; namun, ada skenario ketika memanfaatkan subquery menjadi satu-satunya tindakan yang layak (Wdzięczna, 2020).

Berbagai aplikasi subkueri telah diperiksa, bersama dengan skenario di mana subkueri tertentu dapat diganti dengan operasi GABUNG. Dalam sebagian besar situasi, operasi JOINS dianggap lebih efisien. Namun, ada skenario tertentu di mana konstruksi lain tidak layak dan subquery diperlukan. Meskipun subkueri dapat dianggap lebih mudah diakses oleh pengguna pemula, pemrogram SQL yang berpengalaman cenderung menganggap JOINS lebih mudah dipahami, terutama saat menangani kueri yang rumit. Dianjurkan untuk tidak menggunakan beberapa level subkueri bersarang karena kurangnya keterbacaan dan kinerja yang kurang optimal. Dalam konteks umum, disarankan untuk membuat kueri menggunakan operasi JOINS daripada subkueri, khususnya ketika subkueri menunjukkan korelasi.

DAFTAR PUSTAKA

Babic, T. 2021. *Top 10 SQL JOIN Articles for Beginners*, *learnsql.com*. Available at: https://learnsql.com/blog/top-sql-join-articles/ (Accessed: 10 August 2023).

Elhemali, M. *et al.* 2007. 'Execution strategies for SQL subqueries', *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pp. 993–1004. doi: 10.1145/1247480.1247598.

Harikamoluguram27. 2022. *SQL Join vs Subquery*, *geeksforgeeks.org*. Available at: https://www.geeksforgeeks.org/sql-join-vs-subquery/ (Accessed: 10 August 2023).

Kappagantula, S. 2023. *Understanding SQL Joins – All You Need To Know About SQL Joins*, *edureka.co*. Available at: https://www.edureka.co/blog/sql-joins-types (Accessed: 10 August 2023).

Larsen, G. 2015. *Using a Subquery in a T-SQL Statement*, *databasejournal.com*. Available at: https://www.databasejournal.com/ms-sql/using-a-subquery-in-a-t-sql-statement/ (Accessed: 10 August 2023).

Mode.com. 2019. *Writing Subqueries in SQL*, *Mode.com*. Available at: https://mode.com/sql-tutorial/sql-sub-queries/#:~:text=Subqueries (also known as inner,aggregation in a distinct step. (Accessed: 10 August 2023).

MSFT, W. Da. 2023. *Subqueries (SQL Server)*, *learn.microsoft.com*. Available at: https://learn.microsoft.com/en-us/sql/relational-databases/performance/subqueries?view=sql-server-ver16 (Accessed: 10 August 2023).

Oktavia, T. and Sujarwo, S. 2014. 'Evaluation of sub query performance in SQL server', *EPJ Web of Conferences*, 68, pp. 1–5. doi: 10.1051/epjconf/20146800033.

Piesco, N. G. 2021. *How do you decide when to use a subquery or a join in SQL?*, *linkedin.com*. Available at: https://www.linkedin.com/advice/0/how-do-you-decide-when-use-subquery-join-sql#:~:text=Subquery vs join%3A performance&text=Generally speaking%2C joins are faster,return large or complex results. (Accessed: 10 August 2023).

Wdzięczna, D. 2020. *Subquery vs. JOIN*, *learnsql.com*. Available at: https://learnsql.com/blog/subquery-vs-join/ (Accessed: 10 August 2023).

BAB 10
VIEW DAN STORE PROCEDURE

Oleh Bambang Suprayogi

10.1 Pengertian Dan Manfaat *View*

View merupakan salah satu objek yang berisikan kolom-kolom dari satu atau beberapa tabel yang berelasi menyimpan referensi atau pointer ke record pada tabel-tabel yang berkaitan serta menyimpan dalam bentuk *query* baik dari satu tabel atau lebih dalam sebuah tabel virtual yang tidak menyimpan data secara fisik di dalam database.

Kegunaan *view* dalam database antara lain:

1. Untuk membatasi akses ke database guna pengamanan keamanan data.
2. Dapat menampilkan atau membuat sebuah *query* yang kompleks secara penuh dan mudah.
3. Mengijinkan untuk melakukan perubahan struktur data dalam database tanpa melakukan perubahan pada program yang memproses data tersebut.
4. Dapat menampilkan data yang berbeda dengan sumber data yang sama
5. Untuk mempermudah dalam pengambilan data dari *query* tertentu.
6. Menyembunyikan kolom atau baris suatu tabel atau *query*
7. Menampilkan hasil dari penghitungan baik dalam satu tabel atau beberapa tabel yang berelasi.
8. Menyembunyikan penulisan peritah sql.
9. Termasuk fasilitas yang disediakan oleh database untuk memanifulasi data(*built-in layer).*

10. Menyediakan batasan bagi user sesuai dengan levelnya dalam penggunaan data tabel dan *view* data.
11. Memberikan hak pengunaan untuk *view* yang berbeda dalam pengelolaan tabel yang sama.
12. Ada perlakuan *trigger* dari tabel yang sama pada *view* yang berbeda.
13. Pengambilan kolom dalam tabel.
14. Pengambilan baris dalam tabel.
15. Pengambilan kolom dan baris dalam tabel.
16. Dapat menampilkan *subview* dari *view* lain.
17. Dapat memanipulasi data dalam *record* dengan menggunakan operasi *join*.

Aturan dalam DML untuk membentuk operasi *view* antara lain :

1. Pada *simple view* jika berisi fungsi *group, klausa group by, keyword distinct* pad baris data maka *view* tidak dapat dihapus.
2. Kolom yang telah didefinisikan dalam suatu ekspresi dan kolom *rownum* maka data pada *view* tidak bisa diubah.
3. Jika *view* berisi 5 kondisi dan terdapat kolom *not null* pada table darimana *view* tersebut maka v*iew* tersebut tidak bisa ditambahkan data.

Fitur	Simple view	Complex view
Jumlah table	Satu	Satu atau lebih
Berisi fungsi	Tidak	Ya
Berisi group data	Tidak	Ya
DML melalui view	Ya	Tidak selalu

Gambar 10.1. Fitur *View*
Sumber : http://2.bp.blogspot.com

Tipe *view* dibagi menjadi 2 bagian yaitu *simple view* yang merupakan *view* sederhana hanya dapat berisi satu tabel dasar dan *complex view* merupakan *view* yang berisi lebih dari satu tabel dasar, secara khusus, *view complex* dapat berisi *join conditions, a group by clause, order by clause.*
Berikut ini perbedaan *Simple View* dan *Complex View* antara lain:

Parameter	Tampilan Sederhana dalam SQL	Tampilan Kompleks dalam SQL
Arti dan Definisi	Tampilan Sederhana dalam SQL mengacu pada tampilan yang pembuatannya hanya melibatkan satu tabel. Dengan kata sederhana, Tampilan ini hanya terdiri dari satu tabel dasar di SQL.	Tampilan Kompleks dalam SQL mengacu pada tampilan yang pembuatannya melibatkan lebih dari satu tabel. Dengan kata sederhana, Tampilan ini melibatkan proyeksi beberapa tabel dalam SQL.
Asosiasi	Karena Tampilan ini hanya memiliki satu tabel dalam konteks, seseorang tidak perlu menerapkan asosiasi utama untuk Tampilan Tunggal dalam SQL.	Tampilan ini, di sisi lain, berisi banyak tabel dalam konteks. Dengan demikian, seseorang perlu menerapkan asosiasi umum di dalamnya. Ini termasuk pesanan dengan klausa, grup dengan klausa, dan kondisi bergabung.
Fungsi Grup	Karena Tampilan Sederhana berisi satu tabel, seseorang tidak dapat menggunakan fungsi grup apa pun seperti Hitung (), MAX (), dll.	Tampilan Kompleks, di sisi lain, melibatkan banyak tabel. Dengan demikian, kita dapat memanfaatkan berbagai fungsi grup di dalamnya.
Operasi yang Diizinkan	Seseorang dapat dengan mudah melakukan operasi DML dalam kasus Simple View.	Dalam kasus Tampilan Kompleks, seseorang tidak selalu dapat melakukan operasi DML.
Perubahan	Karena kita dapat melakukan operasi DML, kita dapat dengan mudah menggunakan UPDATE, DELETE, dan INSERT dalam Simple View di SQL.	Karena kita tidak bisa selalu melakukan operasi DML, kita tidak bisa selalu menerapkan UPDATE, DELETE, dan INSERT dalam Tampilan Kompleks di SQL.
Kolom NULL	Anda tidak selalu dapat menggunakan tabel dasar untuk menyertakan kolom NOT NULL.	Seseorang selalu dapat menyertakan kolom NOT NULL jika Tampilan Kompleks dalam SQL.
Contoh	CREATE VIEW Employee AS SELECT Empid, Empname FROM Employee WHERE Empid = '030314';	CREATE VIEW EmployeeByDepartment AS SELECT e.emp_id, d.dept_id, e.emp_name FROM Employee e, Department d WHERE e.dept_id=d.dept_id;

Gambar10.2. Perbedaan *Simple View* dan *Complex View*
Sumber : https://www.geeksforgeeks.org/

10.2 Skema Database dan Tabel

Dalam skema pembuatan database dan tabel penggunaan *script* TSQL sebagai berikut :

```
'------------Membuat Database Swalayan -----------------'

Create Database Swalayan
use Swalayan

'------------Membuat Tabel Tabel ----------------------'

Create Table Pembeli
(IdCustomer varchar(5) Primary Key, NmCustomer varchar(50),
Alamat varchar(150), Telp varchar(20))

Create Table Pegawai
(IdPegawai varchar(5) Primary Key, NmPegawai varchar(50),
AlamatPeg varchar(150), Telp varchar(20))

Create Table Suplier
(IdSuplier varchar(5) Primary Key, NmSuplier varchar(50),
AlamatSup varchar(150), Telp varchar(20))

Create table Barang
(Kdbarang varchar(5) Primary key, Nmbarang varchar(35),
JnsBarang varchar(15), HrgBarang int, JmlBarang int)

Create table Master_Transaksi
(NoFaktur varchar(5) Primary Key, IdCustomer varchar(5),
TotalBayar int)

Create table Detail_Transaksi
    (NoFaktur varchar(5), KdBarang varchar (5), JmlBeli int)
```

```
Create Table Master_Pesan
(NoPO varchar (5) Primary Key, IdPegawai varchar (5), TotalBayar
int)

Create Table Detail_Pesan
(NoPO varchar (5), KdBarang varchar (5) ,IdSuplier Varchar(5),
JmlPO int)
```

Gambar 10.3. Struktur Tabel dalam Database
Sumber : Modul Praktikum Politeknik LP3I Bandung

10.3 Membuat, Menampilkan, Memodifikasi dan Menghapus View

Penulisan *Syntax View* dalam *SQL Server* sebagai berikut:

```
CREATE [ OR ALTER ] VIEW [ schema_name . ] view_name [ (column
[ ,...n ] ) ]
[ WITH <view_attribute> [ ,...n ] ]
AS select_statement
[ WITH CHECK OPTION ]
[ ; ]

<view_attribute> ::=
{
    [ ENCRYPTION ]
    [ SCHEMABINDING ]
    [ VIEW_METADATA ]
}
```

Keterangan :

- *Schema_name*

Nama skema yang untuk *view*

- *View_name*

Nama *view* yang dibuat dalam database

- *With check option*

Menentukan baris data yang boleh di-*insert* atau di-*update, constraint_name* nama yang diberikan untuk *constraint check option*.

- *Encryption*

Mengenkripsi teks pernyataan *alter view* di *sys.syscomments.*

- *Schemabinding*

Tabel dalam *view* tidak dapat dimodifikasi

- *View_metadata*

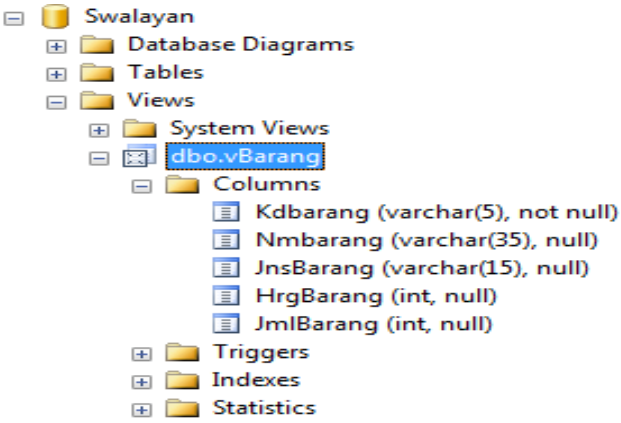
View akan memiliki metadata dalam *sql server*

10.3.1 Membuat *View*

Membuat *View* dengan *Simple View* sebagai berikut :

```
CREATE VIEW vBarang
AS
SELECT Kdbarang, Nmbarang, JnsBarang, HrgBarang, JmlBarang
FROM Barang
GO
```

Membuat sebuah *view* dengan nama vBarang yang akan menampilkan data dari tabel Barang.



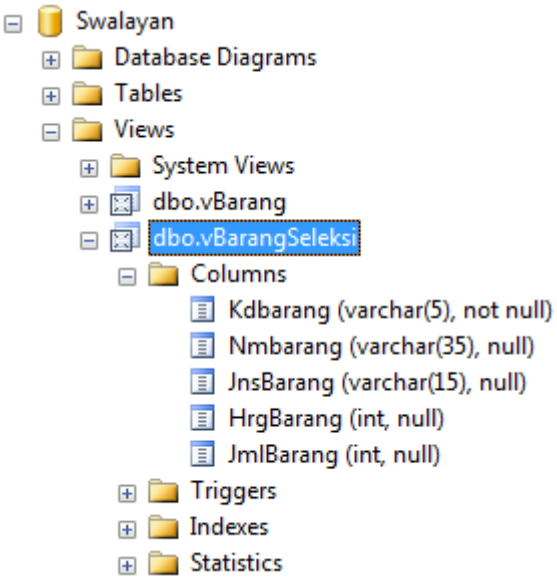
Gambar 10.4. *Simple View*

Sumber : Modul Praktikum Politeknik LP3I Bandung

Membuat *View* seleksi dengan *Simple View* sebagai berikut :

```
CREATE VIEW vBarangSeleksi
AS
SELECT Kdbarang, Nmbarang, JnsBarang, HrgBarang, JmlBarang
FROM Barang where JnsBarang='minuman'
GO
```

Membuat sebuah *view* dengan nama vBarangSeleksi yang akan menampilkan data dari tabel Barang berdasarkan seleksi jenis barang berupa minuman.

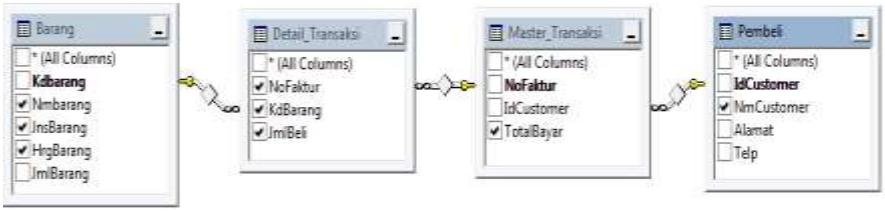


Gambar 10.5. *Simple View* dengan kondisi
Sumber : Modul Praktikum Politeknik LP3I Bandung

Membuat *Complex View* dengan menambahkan fungsi lain dalam subquery seperti *join, group function* dan *set operator, d*alam *TSQL* maka *script* sebagai berikut :

```
Create View Complex_View As
    SELECT Detail_Transaksi.NoFaktur, Pembeli.NmCustomer,
    Detail_Transaksi.KdBarang,Barang.Nmbarang,
    Barang.JnsBarang, Barang.HrgBarang,
    Detail_Transaksi.JmlBeli, Master_Transaksi.TotalBayar FROM
    Barang INNER JOIN Detail_Transaksi ON Barang.Kdbarang =
    Detail_Transaksi.KdBarang INNER JOIN Master_Transaksi ON
    Detail_Transaksi.NoFaktur = Master_Transaksi.NoFaktur
```

```
INNER JOIN Pembeli ON Master_Transaksi.IdCustomer =
Pembeli.IdCustomer
Go
```



Gambar 10.6. Skema *Complex View*
Sumber : Modul Praktikum Politeknik LP3I Bandung

Hasil :

Results | Messages

	NoFaktur	NmCustomer	KdBarang	Nmbarang	JnsBarang	HrgBarang	JmlBeli	TotalBayar
17	F006	Deni	B0011	Energen Ch...	Minuman	10000	23	12000
18	F006	Deni	B0012	SGM Isopro ...	Susu	100000	3	12000
19	F007	Fanni	B0013	Mie Oven Ig...	Mie	3000	2	40000
20	F007	Fanni	B0011	Energen Ch...	Minuman	10000	11	40000
21	F007	Fanni	B0012	SGM Isopro ...	Susu	100000	21	40000
22	F008	Fanni	B004	Nata De Coco	Jelly	15000	12	60000
23	F008	Fanni	B008	Fruit Tea	Minuman	3000	4	60000
24	F009	Fanni	B0014	Mie Oven A...	Mie	3000	6	23000
25	F009	Fanni	B0012	SGM Isopro ...	Susu	100000	23	23000
26	F009	Fanni	B0021	Supero Crac...	Biskuit	5000	3	23000
27	F0010	Andri	B0013	Mie Oven Ig...	Mie	3000	2	34000
28	F0010	Andri	B0021	Supero Crac...	Biskuit	5000	11	34000
29	F0010	Andri	B0022	Taro Puff	Chiki	5000	21	34000

Gambar 10.7. *Complex View*
Sumber : Modul Praktikum Politeknik LP3I Bandung

10.3.2 Menampilkan *View*

Untuk menampilkan view yang dibuat, cukup menggunakan perintah *select* dan menyebutkan nama *view* nya seperti berikut:

```
select * from vBarang.
```

Hasil :

Results | Messages

	Kdbarang	Nmbarang	JnsBarang	HrgBarang	JmlBarang
1	B001	Teh Milk Tea	Minuman	3000	10
2	B0010	Nata De Coco	Jelly	15000	20
3	B0011	Energen Champion	Minuman	10000	20
4	B0012	SGM Isopro Soy	Susu	100000	23
5	B0013	Mie Oven Iga Bakar	Mie	3000	10
6	B0014	Mie Oven Ayam Bawang	Mie	3000	30
7	B0015	Mie Oven Gulai	Mie	3000	23
8	B0016	Samyang Carbo	Mie	15000	20
9	B0017	Samyang Spicy	Mie	10000	20
10	B0018	Samyang Original Spicy	Mie	100000	23
11	B0019	Wafer Classic	Biskuit	13000	10
12	B002	Fruit Tea	Minuman	3000	30
13	B0020	Malkis Marie Susu	Biskuit	5000	30
14	B0021	Supero Cracker	Biskuit	5000	23
15	B0022	Taro Puff	Chiki	5000	20
16	B0023	Nextar	Biskuit	10000	20
17	B0024	Potato	Chiki	100000	23
18	B0025	Tango Wafer Coklat	Biskuit	100000	23
19	B003	Yogurt Drink	Minuman	3000	23
20	B004	Nata De Coco	Jelly	15000	20
21	B005	Energen Champion	Minuman	10000	20
22	B006	SGM Isopro Soy	Susu	100000	23
23	B007	Teh Milk Tea	Minuman	3000	10
24	B008	Fruit Tea	Minuman	3000	30
25	B009	Yogurt Drink	Minuman	3000	23

Gambar 10.8. Hasil *View* Data Barang dalam Database
Sumber : Modul Praktikum Politeknik LP3I Bandung

Menampilkan *View* dengan kondisi seleksi seperti berikut :

```
Select * From vBarangSeleksi
```

Query diatas akan menampilkan data dari tabel Barang dengan kondisi seleksi dengan jenis barang berupa minuman.

Hasil :

Results | Messages

	Kdbarang	Nmbarang	JnsBarang	HrgBarang	JmlBarang
1	B001	Teh Milk Tea	Minuman	3000	10
2	B0011	Energen Champion	Minuman	10000	20
3	B002	Fruit Tea	Minuman	3000	30
4	B003	Yogurt Drink	Minuman	3000	23
5	B005	Energen Champion	Minuman	10000	20
6	B007	Teh Milk Tea	Minuman	3000	10
7	B008	Fruit Tea	Minuman	3000	30
8	B009	Yogurt Drink	Minuman	3000	23

Gambar 10.9. Hasil *View* Seleksi Data Barang dalam Database
Sumber : Modul Praktikum Politeknik LP3I Bandung

10.3.3 Memodifikasi *View*

Untuk merubah *view* yang dibuat cukup menggunakan perintah *sp_rename* dengan menyebutkan nama *view* barunya seperti berikut:

Syntax :

```
EXEC sp_rename @objname = 'schema.oldviewname', @newname
= 'newviewname';
```

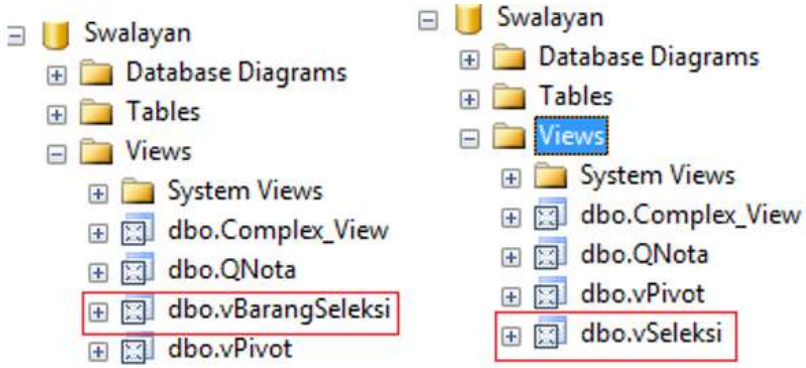
Keterangan :

@objname = 'schema.oldviewname' : Nama *View* lama
@newname = 'newviewname' : Nama *View* baru

Dalam TSQL maka *script* untuk mengganti nama *view* adalah sebagai berikut :

```
EXEC sp_rename vbarangseleksi, vSeleksi
```

Dengan menggunakan *sp_rename*, bila sudah ada *view* yang sudah dibuat dan ingin menganti nama *view* dengan nama baru



Gambar 10.10. Hasil merubah nama *View* dalam Database
Sumber : Modul Praktikum Politeknik LP3I Bandung

10.3.4 Menghapus *View*

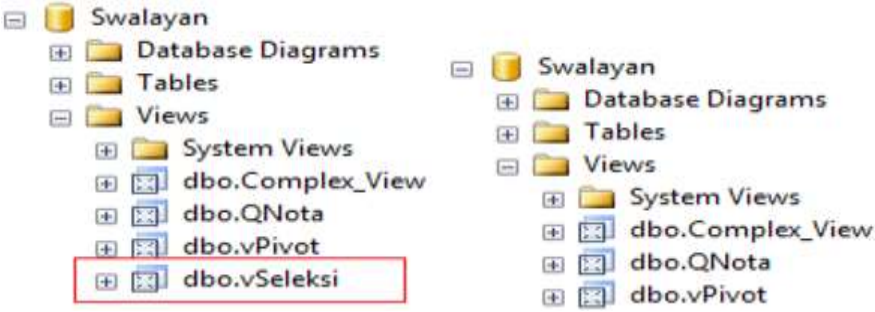
Syntax :

```
DROP VIEW nama_view
```

Dalam TSQL maka *script* untuk menghapus *view* adalah sebagai berikut :

```
DROP VIEW vSeleksi
```

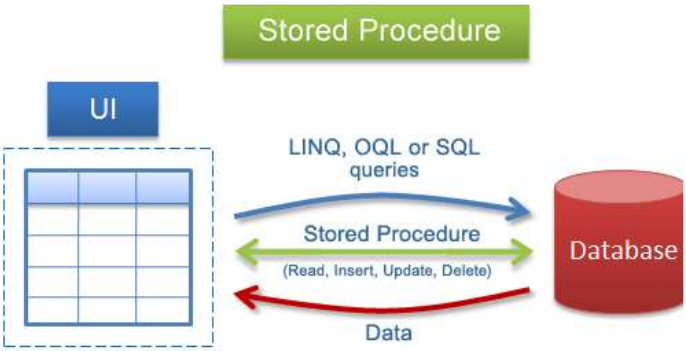
Dengan menggunakan *Drop View*, maka *view* akan dihapus dari database



Gambar 10.11. Hasil menghapus *View* dalam Database
Sumber : Modul Praktikum Politeknik LP3I Bandung

10.4 Pengertian Stored Procedure

Stored procedure merupakan sebuah prosedur SQL yang berisikan *script* perintah, pendefinisian variabel, dan alur logika program dapat menuliskan pernyataan *IF* dan menghasilkan data dengan menggunakan *output parameter, return code, result set* dari *statement select* yang digunakan secara manual atau dijalankan melalui program lain yang tersimpan dalam database.



Gambar 10.12. *Store Procedure*
Sumber : https://www.nesabamedia.com

Kemampuan yang dimiliki dalam *stored procedure* antara lain :

1. Dapat menerima parameter input dan mengembalikan nilai parameter *output* dari sebuah *procedure* yang dipanggil.
2. Dalam *statement* pemrograman dapat memanggil *stored procedure* lain untuk melakukan operasi terhadap database.
3. Memberikan sebuah pesan berhasil atau tidak eksekusi sebuah *stored procedure*.
4. Bersifat pemrograman modular, cukup dibuat sekali dan dapat dipakai berulang-ulang untuk kompilasi yang sama.

5. Eksekusi sebuah *store procedure* akan tersimpan dalam memori sehingga eksekusi berikutnya akan lebih cepat.
6. Mengurangi antrian jaringan barisan kode *transact-sql* sebagai mekanise *security*.

10.4.1 Manfaat dari Stored Procedure

Untuk aplikasi database setelah *procedure* dijalankan di *server* akan menjadi lebih mudah dan cepat karena eksekusi telah tersimpan di dalam memory. Menggunakan *stored procedure* memiliki banyak manfaat diantaranya:

1. Karena disimpan dan dijalankan di *server* maka tidak memberatkan klien.
2. Dapat menjalankan banyak perintah secara bersamaan.
3. Dapat membatasi hak akses suatu *user* terhadap suatu data dan meminimalkan kerusakan dan kebocoran data.
4. Dengan adanya *stored procedure*, *SQL injection* dapat dicegah.
5. Mengurangi akses ke database dan meningkatkan performa aplikasi.
6. Lebih fleksibel dalam proses yang dilakukan database *server* sehingga lebih cepat dikarenakan aplikasi cukup memanggil *stored procedure* dan mengirim parameter tersebut.

Dalam penggunaan *stored procedure* memiliki keterbatasan dikarenakan di dalam *stored procedure* tidak dapat dibuatkan *function, rules, trigger, view* dan *schemas*. Adapun kerugian penggunaan dalam *stored procedure* antara lain :

1. Database server dalam menjalankan *stored procedure* membutuhkan *memory* dan *prosessor* yang memadai.
2. Dalam penulisan *Stored procedure* dengan kompleksitas logika sangat sulit karena hanya berisi SQL deklaratif.

3. Dalam pengelolaan dan memaintain *stored procedure* dibutuhkan keahlian khusus seorang developer dikarenakan beban kerja server meningkat karena prosedur yang seharusnya dapat dikerjakan pada aplikasi pemanggil harus dikerjakan pada server.
4. Lebih sulit dalam *debugging* ketika *store procedure* yang dibuat semakin banyak.

10.5 Perintah Stored Procedure

Perintah dalam membuat *stored procedure* dengan perintah sebagai berikut:

```
CREATE PROCEDURE [NAMA_PROCEDURE]
   ([@NAMA_VARIABEL TIPE DATA])
AS
BEGIN
   [SQL STATEMENT]
END
```

Terdapat tiga komponen dalam membuat *stored procedure*

1. *Input parameters* : menangani input
2. *Output paramters* : menangani output yang dihasilkan
3. *Body* : berisi statement SQL untuk dieksekusi

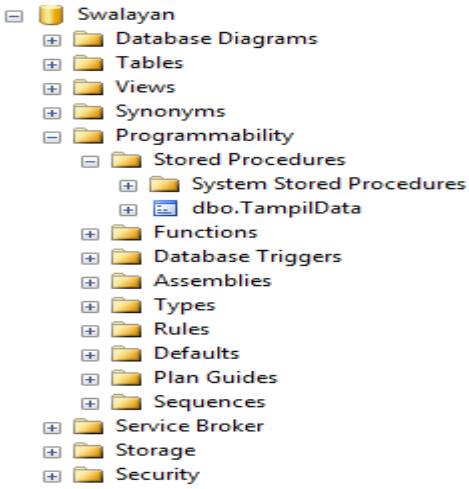
10.5.1 Membuat Stored Procedure

Syntax membuat *Stored Procedure* sebagai berikut :

```
CREATE PROCEDURE nama_procedure
(@parameter1 tipe_data, @parameter2 tipe_data, @parameter_n
tipe_data)
AS BEGIN TRANSACTION
//letakkan Query disini

IF @@ERROR = 0
COMMIT TRANSACTION
ELSE
ROLLBACK TRANSACTION
```

Stored Procedure untuk menampilkan data barang tanpa ada pemilihan parameter, contoh sebagai berikut :

```
CREATE PROCEDURE TampilData
AS
BEGIN
 SET NOCOUNT ON;
                SELECT * from Barang
END
GO
```

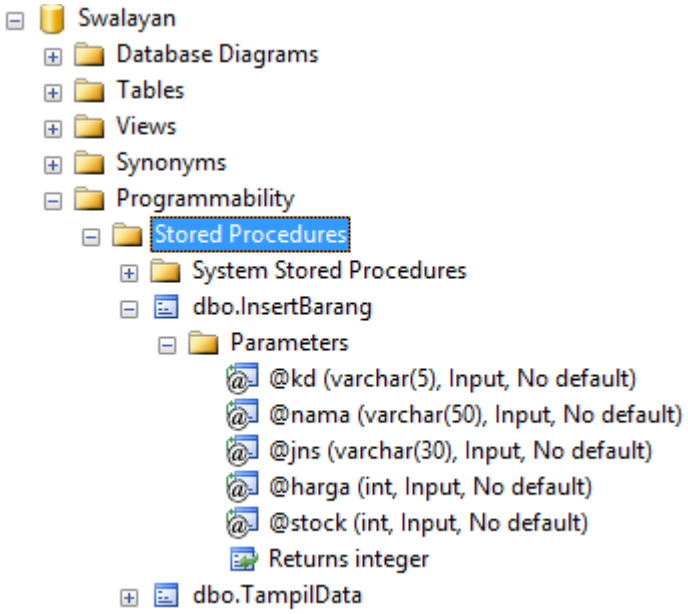


Gambar 10.13. Pembuatan *Store Procedure* tanpa Parameter
Sumber : Modul Praktikum Politeknik LP3I Bandung

Stored Procedure untuk menampilkan data barang dengan pemilihan parameter, contoh sebagai berikut :

```
CREATE PROCEDURE InsertBarang
 -- Tambah parameters stored procedure
 @kd varchar(5),
```

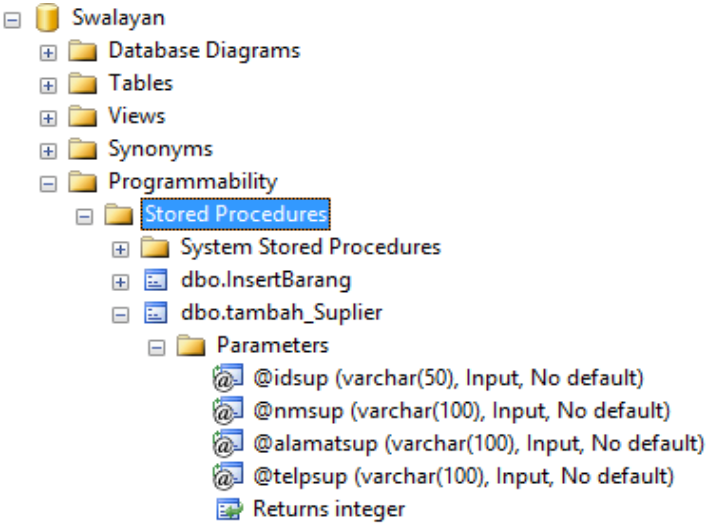
```
@nama varchar(50),
@jns varchar(30),
@hrg int,
@stock int
AS
BEGIN
-- SET NOCOUNT ON untuk menambahkan prefent dalam  SELECT
statements.
 SET NOCOUNT ON;
-- Insert statements store procedure
 insert  into  Barang(Kdbarang,  Nmbarang,  JnsBarang,  HrgBarang,
JmlBarang) values (@kd, @nama,@jns, @hrg, @stock)
END
GO
```



Gambar 10.14. Pembuatan *Store Procedure dengan Parameter*
Sumber : Modul Praktikum Politeknik LP3I Bandung

Stored Procedure untuk menambahkan data suplier dengan pemilihan parameter dan kondisi *error*, contoh sebagai berikut :

```
CREATE PROCEDURE tambah_Suplier
(@idsup VARCHAR(50), @nmsup VARCHAR(100), @alamatsup
VARCHAR(100), @telpsup VARCHAR(100))
AS BEGIN TRANSACTION
INSERT INTO Suplier  (idsuplier, nmsuplier,alamatSup, telp)
VALUES
(@idsup, @nmsup, @alamatsup, @telpsup)

IF @@ERROR = 0
COMMIT TRANSACTION
ELSE
ROLLBACK TRANSACTION
```



Gambar 10.15. Pembuatan *Store Procedure dengan Parameter*
Sumber : Modul Praktikum Politeknik LP3I Bandung

10.5.2 Menjalankan Store Procedure

Untuk meng*execute stored procedure* yang telah dibuat dengan menggunakan perintah EXEC

```
EXEC nama_procedure
atau
EXEC nama_procedure 'parameter1',…,'parameter_n'
```

Menjalankan *store procedure* TampilData sebagai berikut :

```
EXEC TampilData
```

Hasil :

Results | Messages

	Kdbarang	Nmbarang	JnsBarang	HrgBarang	JmlBarang
1	B001	Teh Milk Tea	Minuman	3000	10
2	B0010	Nata De Coco	Jelly	15000	20
3	B0011	Energen Champion	Minuman	10000	20
4	B0012	SGM Isopro Soy	Susu	100000	23
5	B0013	Mie Oven Iga Bakar	Mie	3000	10
6	B0014	Mie Oven Ayam Bawang	Mie	3000	30
7	B0015	Mie Oven Gulai	Mie	3000	23
8	B0016	Samyang Carbo	Mie	15000	20
9	B0017	Samyang Spicy	Mie	10000	20
10	B0018	Samyang Original Spicy	Mie	100000	23
11	B0019	Wafer Classic	Biskuit	13000	10
12	B002	Fruit Tea	Minuman	3000	30
13	B0020	Malkis Marie Susu	Biskuit	5000	30
14	B0021	Supero Cracker	Biskuit	5000	23

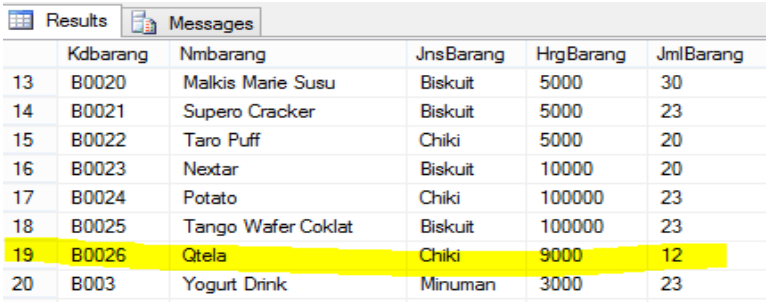
Gambar 10.16. *Execute Store Procedure*
Sumber : Modul Praktikum Politeknik LP3I Bandung

10.5.3 Mengubah Stored Procedure

Menjalankan *store procedure* InsertBarang dengan parameter @kd, @nama, @jns, @harga, @stock sebagai berikut :

```
EXEC InsertBarang'B0026','Qtela','Chiki',9000,12
```

Hasil :



Results | Messages

	Kdbarang	Nmbarang	JnsBarang	HrgBarang	JmlBarang
13	B0020	Malkis Marie Susu	Biskuit	5000	30
14	B0021	Supero Cracker	Biskuit	5000	23
15	B0022	Taro Puff	Chiki	5000	20
16	B0023	Nextar	Biskuit	10000	20
17	B0024	Potato	Chiki	100000	23
18	B0025	Tango Wafer Coklat	Biskuit	100000	23
19	B0026	Qtela	Chiki	9000	12
20	B003	Yogurt Drink	Minuman	3000	23

Gambar 10.17. *Execute Insert Store Procedure*
Sumber : Modul Praktikum Politeknik LP3I Bandung

10.5.4 Menghapus Stored Procedure

Sintax:

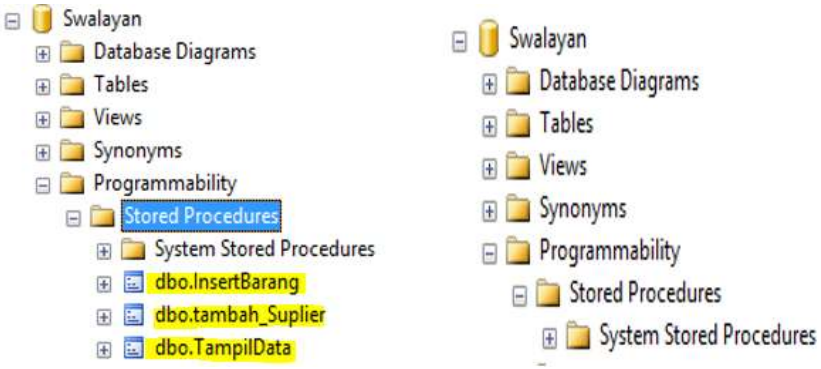
```
DROP { PROC | PROCEDURE } { [ schema_name. ] procedure } [ ,...n ]
```

Menghapus sebuah *stored procedure* bernama TampilData dengan *schema dbo*:

```
DROP PROCEDURE dbo.TampilData
```

Menghapus beberapa (banyak) *stored procedure*, sebagai contoh 3 buah *stored procedure* bernama TampilData, InsertBarang dan tambah_suplier dengan *schema dbo*:

```
DROP PROCEDURE dbo.TampilData, dbo.InsertBarang,
dbo.tambah_Suplier
```



Gambar 1018. *Execute Drop Store Procedure*
Sumber : Modul Praktikum Politeknik LP3I Bandung

Sementara menghapus *stored procedure* dengan melakukan cek terlebih dahulu apakah *stored procedure* ada atau tidak, dapat menggunakan *query* dengan format sebagai berikut:

```
IF EXISTS
       (select * from dbo.sysobjects
       where id = object_id(n'[dbo].[namastoredprocedure]')
       and objectproperty(id, n'isprocedure') = 1)
       DROP PROCEDURE [dbo].[NAMASTOREDPROCEDURE]
GO
```

DAFTAR PUSTAKA

Andi. 2012. *SQL Server 2012*. Semarang: Andi Offset.

Bagui, S. and Richard, E. 2003. *Database Design Using Entity Relationship Diagrams*. New York: Auerbach Publications.

Connoly, T.M. and Begg, C.E. 2005. *Database System : A Practical Approach to Design, Impelmentation, and Management*. 4th edn. United States of America: Addison Wesley inc.

Prabawati. 2013. *SQL Server 2012*. Yogyakarta: Andi.

Ross, M. and Misnes Stacia. 2012. *Introducing Microsoft SQL Server 2012*. United States of America: Mikcrosoft Press.

BAB 11
SQL: TRANSAKSI PENGGUNA

By Boy Firmansyah

Perintah SQL "transaksi pengguna" adalah istilah yang agak umum, dan dalam konteks basis data, itu biasanya merujuk pada penggunaan transaksi oleh pengguna atau aplikasi dalam basis data. Saya akan memberikan deskripsi lengkap tentang apa itu transaksi pengguna dalam SQL.

11.1 Transaksi Dalam Sql

Transaksi adalah serangkaian perintah SQL yang dieksekusi sebagai satu unit tunggal, yang berarti semua perintah dalam transaksi harus berhasil atau gagal secara bersamaan. Jika salah satu perintah dalam transaksi gagal, transaksi tersebut dapat digulirkan kembali (rollback) sehingga tidak ada perubahan yang disimpan dalam basis data, atau jika semuanya berhasil, maka perubahan dapat dikonfirmasi (commit).

11.2 Transaksi Pengguna

Transaksi pengguna adalah transaksi yang dinyatakan oleh pengguna atau aplikasi yang berinteraksi dengan basis data. Ini bisa menjadi serangkaian perintah SQL yang digunakan oleh pengguna untuk menambahkan, mengubah, atau menghapus data dalam basis data. Misalnya, ketika seorang pengguna ingin menambahkan entri baru ke dalam tabel, ia dapat melakukan serangkaian perintah SQL dalam satu transaksi.

11.3 Fitur Utama Transaksi Pengguna

Atomicity: Transaksi pengguna memiliki sifat atomic, yang berarti semua perintah dalam transaksi harus berhasil atau gagal bersamaan. Jika salah satu perintah gagal, perubahan yang dilakukan oleh perintah sebelumnya dibatalkan (*rollback*).

Consistency: Transaksi harus menjaga konsistensi basis data sebelum dan sesudah transaksi. Artinya, perubahan yang dilakukan oleh transaksi harus mematuhi aturan dan kendala yang telah ditetapkan dalam struktur basis data.

Isolation: Transaksi pengguna harus diisolasi dari transaksi lain yang sedang berlangsung dalam basis data. Ini memastikan bahwa satu transaksi tidak dapat melihat perubahan yang sedang dilakukan oleh transaksi lain sebelum transaksi tersebut dikonfirmasi (*commit*).

Durability: Setelah transaksi pengguna dikonfirmasi (*commit*), perubahan yang dibuat harus tetap ada dalam basis data bahkan jika sistem mengalami kegagalan. Ini memastikan keamanan dan keandalan data.

Contoh Transaksi Pengguna:

Misalnya, jika seorang pengguna ingin menambahkan informasi pelanggan baru ke dalam basis data toko online, transaksi pengguna mungkin terdiri dari serangkaian perintah SQL seperti INSERT INTO customers VALUES (...), INSERT INTO orders VALUES (...), dan UPDATE inventory SET .... Semua perintah ini akan dikemas dalam satu transaksi, sehingga jika salah satunya gagal (misalnya, karena kegagalan validasi), semua perubahan akan dibatalkan.

11.4 Konsep Dasar Transaksi SQL

Dalam rangka mengoptimalkan penggunaan transaksi pengguna, penting untuk memahami konsep dasar transaksi SQL dan memastikan bahwa transaksi tersebut dirancang dan

dieksekusi dengan benar agar memenuhi persyaratan bisnis dan menjaga integritas data dalam basis data.

Untuk melakukan transaksi pengguna dalam SQL, Anda biasanya akan menggunakan pernyataan-pernyataan seperti BEGIN, COMMIT, dan ROLLBACK bersama dengan perintah-perintah DML (Data Manipulation Language) seperti INSERT, UPDATE, dan DELETE. Contoh sederhana dari urutan perintah SQL untuk transaksi pengguna adalah:

```
BEGIN; -- Memulai transaksi
UPDATE saldo_pengguna SET saldo = saldo - 100 WHERE user_id = 123; -- Mengurangkan saldo
INSERT INTO riwayat_transaksi (user_id, jenis_transaksi, jumlah) VALUES (123, 'Pengurangan Saldo', 100); -- Mencatat transaksi
COMMIT; -- Menyimpan perubahan dalam database
```

Jika terjadi masalah, Anda bisa menggunakan ROLLBACK untuk membatalkan transaksi:

```
BEGIN;
UPDATE saldo_pengguna SET saldo = saldo - 100 WHERE user_id = 123;
-- Jika ada masalah, perintah di atas bisa dibatalkan dengan:
ROLLBACK; -- Membatalkan transaksi dan mengembalikan keadaan awal
```

Pastikan Anda menggantikan nama tabel, kolom, nilai, dan kondisi sesuai dengan struktur basis data dan kebutuhan transaksi Anda.

11.5 Perintah SQL Begin

Pernyataan SQL BEGIN digunakan untuk memulai sebuah transaksi dalam sistem manajemen basis data (DBMS) yang mendukung transaksi. Transaksi adalah serangkaian tindakan atau perubahan yang harus dilakukan sebagai satu kesatuan, artinya semua tindakan dalam transaksi harus berhasil atau semuanya dibatalkan.

Berikut adalah penjelasan rinci tentang pernyataan SQL BEGIN:

Tujuan Transaksi: Transaksi digunakan untuk memastikan integritas data dan konsistensi dalam basis data. Transaksi memastikan bahwa perubahan data dilakukan dengan benar dan aman, serta memungkinkan pemulihan ke keadaan semula jika ada kesalahan.

Transaksi dan ACID Properties:

***Atomicity* (Atomic)**: Transaksi dianggap atomik, artinya entah semua perubahan dalam transaksi berhasil dilakukan atau tidak ada perubahan sama sekali. Jika ada satu perubahan yang gagal, maka seluruh transaksi dibatalkan.

***Consistency* (Konsistensi)**: Transaksi harus memastikan bahwa basis data berada dalam keadaan yang konsisten sebelum dan sesudah transaksi. Ini berarti aturan integritas data harus tetap terjaga.

***Isolation* (Isolasi)**: Transaksi yang berjalan secara bersamaan tidak boleh saling mengganggu satu sama lain. Setiap transaksi seolah-olah berjalan sendiri tanpa interferensi dari transaksi lainnya.

***Durability* (Daya Tahan)**: Setelah transaksi selesai dan berhasil dilakukan, perubahan yang dilakukan harus tetap ada dan aman, bahkan jika terjadi kegagalan sistem setelahnya.

Pemakaian: Pernyataan BEGIN biasanya digunakan bersama dengan pernyataan COMMIT dan ROLLBACK untuk mengendalikan transaksi. Pernyataan BEGIN menandai awal transaksi, diikuti oleh pernyataan-pernyataan DML (Data Manipulation Language) seperti INSERT, UPDATE, dan DELETE yang akan membentuk inti dari transaksi.

Contoh Penggunaan:

Misalkan Anda ingin mengurangkan saldo pengguna dan mencatat transaksi yang sesuai:

BEGIN; -- Memulai transaksi
UPDATE saldo_pengguna SET saldo = saldo - 100 WHERE user_id = 123; -- Mengurangkan saldo

INSERT INTO riwayat_transaksi (user_id, jenis_transaksi, jumlah) VALUES (123, 'Pengurangan Saldo', 100); -- Mencatat transaksi
COMMIT; -- Menyimpan perubahan dalam database

Rollback: Jika ada masalah selama transaksi, Anda dapat menggunakan perintah ROLLBACK untuk membatalkan semua perubahan yang dilakukan dalam transaksi tersebut dan mengembalikan basis data ke keadaan semula sebelum transaksi dimulai.

Penanganan Kesalahan: Dalam kasus kesalahan atau kegagalan selama transaksi, penting untuk menggunakan ROLLBACK untuk memastikan data tetap konsisten dan tidak rusak.

Pernyataan BEGIN adalah langkah pertama dalam membangun transaksi yang aman dan konsisten dalam SQL.

11.6 Perintah SQL Commit

Tujuan Transaksi dan Commit: Dalam SQL, sebuah transaksi adalah serangkaian perintah yang dijalankan sebagai satu kesatuan, sehingga semua perubahan dalam transaksi harus berhasil atau semuanya dibatalkan. Pernyataan COMMIT digunakan untuk menyelesaikan dan menyimpan semua perubahan yang dilakukan dalam transaksi ke dalam basis data, sehingga perubahan tersebut menjadi permanen dan dapat dilihat oleh pengguna lain.

Transaksi dan ACID Properties:

***Atomicity* (Atomic)**: Pernyataan COMMIT menandai akhir dari transaksi dan mengonfirmasi bahwa semua perintah dalam transaksi berhasil dilakukan. Ini menjaga prinsip atomisitas, yaitu entah semua perubahan berhasil dilakukan atau tidak ada perubahan sama sekali.

***Consistency* (Konsistensi)**: Sebelum pernyataan COMMIT dieksekusi, basis data harus dalam keadaan yang konsisten sesuai dengan aturan integritas data. Setelah COMMIT, perubahan-perubahan tersebut menjadi permanen.

Pemakaian: Setelah semua perintah DML (*Data Manipulation Language*) dalam transaksi selesai dan tidak ada masalah, Anda dapat menggunakan pernyataan COMMIT untuk menyimpan perubahan-perubahan tersebut. Setelah COMMIT dieksekusi, perubahan akan menjadi permanen dan dapat dilihat oleh pengguna lain.

Contoh Penggunaan:

Misalnya, Anda telah mengeksekusi beberapa perintah dalam transaksi untuk mengurangkan saldo pengguna dan mencatat transaksi:

BEGIN; -- Memulai transaksi

UPDATE saldo_pengguna SET saldo = saldo - 100 WHERE user_id = 123; -- Mengurangkan saldo

INSERT INTO riwayat_transaksi (user_id, jenis_transaksi, jumlah) VALUES (123, 'Pengurangan Saldo', 100); -- Mencatat transaksi

COMMIT; -- Menyimpan perubahan dalam database

Konfirmasi dan Pembatalan: Setelah COMMIT dieksekusi, semua perubahan dalam transaksi menjadi permanen dan tidak dapat dibatalkan dengan mudah. Oleh karena itu, pastikan semua perintah dalam transaksi telah diperiksa dengan cermat sebelum melakukan COMMIT.

Durability (Daya Tahan): Salah satu dari properti ACID adalah "Durability", yang berarti bahwa setelah COMMIT, perubahan akan tetap ada bahkan jika terjadi kegagalan sistem setelahnya. Basis data harus dapat mempertahankan perubahan yang telah di-commit.

Pernyataan COMMIT adalah langkah akhir dalam sebuah transaksi, yang menjadikan perubahan dalam basis data menjadi permanen dan tersedia bagi pengguna lain.

11.7 Pernyataan SQL ROLLBACK

Tujuan Transaksi dan Rollback: Dalam SQL, sebuah transaksi adalah serangkaian perintah yang dijalankan sebagai satu kesatuan. Pernyataan ROLLBACK digunakan untuk membatalkan semua perubahan yang dilakukan dalam transaksi yang sedang berjalan. Ini memungkinkan pengembalian basis data ke keadaan semula sebelum transaksi dimulai.

Transaksi dan ACID Properties:

Atomicity **(Atomic)**: Pernyataan ROLLBACK mengimplementasikan prinsip atomisitas. Jika ada salah satu perintah dalam transaksi yang gagal, maka seluruh transaksi dibatalkan dan perubahan tidak akan mempengaruhi basis data.

Consistency **(Konsistensi)**: Rollback memastikan bahwa setelah transaksi dibatalkan, basis data kembali ke keadaan yang konsisten sesuai dengan aturan integritas data sebelum transaksi dimulai.

Isolation **(Isolasi)**: Rollback juga memastikan bahwa perubahan yang dilakukan oleh transaksi yang sedang berjalan tidak terlihat oleh transaksi lain. Ini mencegah pengaruh negatif dari transaksi yang tidak berhasil pada transaksi lainnya.

Durability **(Daya Tahan)**: Pernyataan ROLLBACK mengembalikan basis data ke keadaan sebelum transaksi dimulai, mengonfirmasi daya tahan perubahan.

Pemakaian: Jika ada masalah dalam transaksi atau jika Anda ingin membatalkan semua perubahan dalam transaksi, Anda dapat menggunakan pernyataan ROLLBACK. Ini akan mengembalikan basis data ke keadaan yang ada sebelum transaksi dimulai.

Contoh Penggunaan:

Misalnya, jika terjadi masalah dalam transaksi yang mengurangkan saldo pengguna dan mencatat transaksi:

```
BEGIN; -- Memulai transaksi
UPDATE saldo_pengguna SET saldo = saldo - 100 WHERE user_id = 123; -- Mengurangkan saldo
```

INSERT INTO riwayat_transaksi (user_id, jenis_transaksi, jumlah) VALUES (123, 'Pengurangan Saldo', 100); -- Mencatat transaksi
ROLLBACK; -- Membatalkan transaksi dan mengembalikan basis data ke keadaan semula
Pentingnya Pemeriksaan: Sebelum menggunakan ROLLBACK, pastikan untuk memeriksa dan memahami alasan mengapa transaksi perlu dibatalkan. Rollback mengembalikan semua perubahan dalam transaksi, dan ini mungkin tidak dapat dikembalikan jika perubahan telah ada di level yang lebih tinggi.
Kemungkinan Penggunaan Transaksi Ulang: Setelah ROLLBACK, Anda mungkin perlu menjalankan kembali transaksi dengan perbaikan atau penyesuaian yang diperlukan untuk mencapai hasil yang diinginkan.

Pernyataan ROLLBACK adalah cara untuk membatalkan transaksi yang sedang berjalan dan mengembalikan basis data ke keadaan sebelum transaksi dimulai. Hal ini sangat penting untuk menjaga integritas dan konsistensi data dalam sistem manajemen basis data.

11.8 Manajemen Pengguna

Perintah SQL untuk manajemen pengguna biasanya melibatkan operasi-operasi yang berkaitan dengan pemberian izin (permissions) pada pengguna, pembuatan, pengubahan, atau penghapusan pengguna. Di bawah ini adalah beberapa perintah SQL umum yang digunakan dalam manajemen pengguna:

1. **Membuat Pengguna:**
   CREATE USER username IDENTIFIED BY 'password';
   Ini akan membuat pengguna baru dengan nama username dan kata sandi password.
   Tujuan Pernyataan CREATE USER: Pernyataan CREATE USER dalam SQL digunakan untuk membuat pengguna baru dalam basis data. Pengguna ini dapat digunakan untuk

mengakses dan memanipulasi data dalam basis data sesuai dengan izin yang diberikan.

Batasan dan Variasi: Perintah CREATE USER tidak selalu ada di semua sistem manajemen basis data (DBMS), dan sintaksisnya bisa bervariasi antara DBMS yang berbeda. Misalnya, MySQL, PostgreSQL, dan SQL Server memiliki sintaksis yang sedikit berbeda.

Sintaksis Dasar:

CREATE USER username IDENTIFIED BY 'password';

username: Nama pengguna yang ingin Anda buat. Biasanya dalam bentuk teks atau huruf kecil.

'password': Kata sandi yang akan diatur untuk pengguna ini. Ini perlu dilakukan dengan hati-hati untuk menjaga keamanan.

Contoh Penggunaan:

Misalnya, jika Anda ingin membuat pengguna baru dengan nama "john" dan kata sandi "secretpass":

CREATE USER john IDENTIFIED BY 'secretpass';

Setelah pernyataan di atas dijalankan, pengguna baru dengan nama "john" akan dibuat dalam basis data dengan kata sandi yang sesuai.

Hak Akses dan Izin: Setelah pengguna dibuat, Anda perlu memberikan izin pada pengguna tersebut untuk mengakses tabel atau objek lain dalam basis data. Ini biasanya dilakukan dengan menggunakan perintah GRANT.

Perlindungan Kata Sandi: Pastikan kata sandi yang Anda atur kuat dan aman. Penggunaan kata sandi yang lemah dapat mengakibatkan risiko keamanan yang serius.

Pengaturan Ekstra: Terkadang, Anda mungkin perlu menyertakan parameter ekstra seperti lokasi IP dari mana pengguna diperbolehkan masuk atau batasan lainnya tergantung pada DBMS yang Anda gunakan.

Ketergantungan pada DBMS: Ingatlah bahwa setiap sistem manajemen basis data (MySQL, PostgreSQL, SQL Server, dsb.)

mungkin memiliki variasi sintaksis dan fitur tambahan yang dapat digunakan saat membuat pengguna.

Perintah CREATE USER adalah langkah pertama dalam memberikan akses kepada individu atau aplikasi ke dalam basis data. Pastikan untuk merencanakan izin dan keamanan dengan baik saat membuat pengguna baru.

2. **Memberikan Izin:**

   GRANT permission_type ON database_name.table_name TO username;

   Ini memberikan izin tertentu (misalnya SELECT, INSERT, UPDATE, DELETE) pada tabel dalam basis data kepada pengguna tertentu.

   Tujuan Pernyataan GRANT: Pernyataan GRANT dalam SQL digunakan untuk memberikan izin (permissions) tertentu kepada pengguna atau peran (roles) dalam basis data. Izin ini mengatur apa yang dapat dilakukan oleh pengguna atau peran terhadap objek-objek seperti tabel, tampilan (views), atau prosedur-prosedur dalam basis data.

   Tipe Izin: Izin yang dapat diberikan meliputi berbagai jenis aktivitas, seperti SELECT (membaca data), INSERT (menambahkan data), UPDATE (mengubah data), DELETE (menghapus data), CREATE (membuat objek baru), ALTER (mengubah struktur objek), DROP (menghapus objek), dan banyak lainnya, tergantung pada sistem manajemen basis data (DBMS) yang Anda gunakan.

   Sintaksis Dasar:

   GRANT permission_type(s) ON object TO user_or_role;

   permission_type(s): Izin yang ingin Anda berikan, bisa berupa satu atau lebih jenis izin yang dipisahkan dengan koma. Misalnya, SELECT, INSERT, UPDATE.

   object: Objek dalam basis data yang akan diberikan izin. Ini bisa berupa tabel, tampilan, atau objek lainnya.

   user_or_role: Pengguna atau peran yang akan diberikan izin.

Contoh Penggunaan:
Misalnya, jika Anda ingin memberikan izin kepada pengguna "john" untuk membaca data dari tabel "pegawai":
GRANT SELECT ON pegawai TO john;
Setelah pernyataan di atas dijalankan, pengguna "john" akan dapat menjalankan perintah SELECT pada tabel "pegawai."
Revoke Izin: Jika Anda perlu mencabut izin yang telah diberikan sebelumnya, Anda dapat menggunakan perintah REVOKE. Misalnya, untuk mencabut izin SELECT dari pengguna "john":
REVOKE SELECT ON pegawai FROM john;
Izin Global dan Spesifik Objek: Anda dapat memberikan izin baik pada tingkat global (ke seluruh basis data) maupun pada objek tertentu dalam basis data. Izin yang lebih spesifik akan menggantikan izin yang lebih umum jika ada konflik.
Perlindungan Keamanan: Penting untuk merencanakan izin dengan cermat untuk memastikan bahwa pengguna hanya memiliki akses yang diperlukan dan tidak lebih. Ini membantu menjaga keamanan dan integritas data.
Peran (Roles): Selain memberikan izin kepada pengguna individu, Anda juga dapat membuat peran yang mengelompokkan izin-izin bersama-sama. Ini membuat manajemen izin lebih efisien, terutama dalam skenario dengan banyak pengguna.
Ketergantungan pada DBMS: Ingatlah bahwa sintaksis dan kemampuan GRANT dapat berbeda-beda antara DBMS yang berbeda (misalnya, MySQL, PostgreSQL, SQL Server). Pastikan untuk merujuk pada dokumentasi DBMS yang Anda gunakan untuk detail yang lebih spesifik.
Pernyataan GRANT adalah alat penting dalam manajemen izin basis data yang memungkinkan Anda mengendalikan siapa yang memiliki akses ke data dan apa yang dapat mereka lakukan dengan data tersebut.

3. **Mengubah Kata Sandi Pengguna:**
   ALTER USER old_username RENAME TO new_username;
   Ini memungkinkan pengguna untuk mengubah kata sandi mereka.
   Tujuan Pernyataan ALTER USER: Pernyataan ALTER USER dalam SQL digunakan untuk mengubah pengaturan atau atribut seorang pengguna dalam basis data. Ini memungkinkan Anda untuk melakukan tindakan seperti mengganti kata sandi pengguna, mengubah hak akses, atau mengatur opsi pengguna lainnya.
   Batasan dan Variasi: Perintah ALTER USER tidak selalu ada di semua sistem manajemen basis data (DBMS), dan sintaksisnya bisa bervariasi antara DBMS yang berbeda. Misalnya, MySQL, PostgreSQL, dan SQL Server memiliki sintaksis yang sedikit berbeda.
   Sintaksis Dasar:
   ALTER USER username [OPTIONS];
   username: Nama pengguna yang ingin Anda ubah atributnya.
   [OPTIONS]: Opsi tambahan yang bergantung pada DBMS tertentu. Ini mencakup pengaturan seperti mengubah kata sandi, mengganti nama pengguna, mengaktifkan atau menonaktifkan pengguna, dan banyak lagi.
   Contoh Penggunaan:
   Mengubah kata sandi pengguna:
   ALTER USER john IDENTIFIED BY 'new_password';
   Ini akan mengubah kata sandi pengguna "john" menjadi "new_password".
   Mengganti nama pengguna:
   ALTER USER old_username RENAME TO new_username;
   Ini akan mengganti nama pengguna "old_username" menjadi "new_username".
   Pengaturan Kata Sandi: Jika Anda menggunakan pernyataan ALTER USER untuk mengubah kata sandi pengguna, pastikan

kata sandi yang baru adalah aman dan kuat, mengikuti kebijakan keamanan yang tepat.

Hak Akses: Di beberapa DBMS, Anda mungkin dapat menggunakan pernyataan ALTER USER untuk mengubah hak akses pengguna, seperti memberikan atau mencabut izin tertentu. Namun, hal ini bisa bervariasi tergantung pada DBMS yang Anda gunakan.

Mengaktifkan dan Menonaktifkan Pengguna: Anda juga dapat menggunakan pernyataan ALTER USER untuk mengaktifkan atau menonaktifkan pengguna. Ini dapat berguna jika Anda ingin sementara menonaktifkan akses pengguna tanpa menghapusnya sepenuhnya dari basis data.

Opsi Lainnya: Selain perubahan yang umum seperti yang disebutkan di atas, tergantung pada DBMS, Anda mungkin memiliki opsi lain yang dapat Anda atur dengan pernyataan ALTER USER. Pastikan untuk merujuk pada dokumentasi DBMS yang Anda gunakan untuk opsi yang spesifik.

Ketergantungan pada DBMS: Ingatlah bahwa sintaksis dan kemampuan ALTER USER dapat berbeda-beda antara DBMS yang berbeda. Pastikan untuk merujuk pada dokumentasi DBMS Anda untuk detail yang lebih spesifik.

Perintah ALTER USER adalah alat penting dalam manajemen pengguna basis data, yang memungkinkan Anda mengubah atribut-atribut pengguna sesuai kebutuhan, mengelola kata sandi, dan mengatur hak akses pengguna.

4. **Mengubah Nama Pengguna:**

```
ALTER USER old_username RENAME TO new_username;
```

Ini memungkinkan pengubahan nama pengguna.

Tujuan Pernyataan ALTER USER Untuk Mengubah Nama Pengguna: Pernyataan ALTER USER dalam SQL digunakan untuk mengubah nama pengguna dalam basis data. Ini berguna jika Anda perlu mengganti nama pengguna yang telah ada tanpa harus membuat pengguna baru.

Batasan dan Variasi: Perintah ALTER USER untuk mengubah nama pengguna tidak selalu ada di semua sistem manajemen basis data (DBMS), dan sintaksisnya bisa bervariasi antara DBMS yang berbeda. Pastikan untuk merujuk pada dokumentasi DBMS Anda untuk memahami cara kerjanya dalam DBMS tertentu.

Sintaksis Dasar:

ALTER USER old_username RENAME TO new_username;

old_username: Nama pengguna yang ingin Anda ganti.

new_username: Nama baru yang ingin Anda terapkan pada pengguna.

Contoh Penggunaan:

Misalnya, jika Anda ingin mengubah nama pengguna "john" menjadi "jane":

ALTER USER john RENAME TO jane;

Ini akan mengubah nama pengguna "john" menjadi "jane" dalam basis data.

Ketentuan Tambahan: Beberapa DBMS mungkin memiliki ketentuan tambahan atau batasan saat mengubah nama pengguna. Misalnya, beberapa DBMS mungkin memerlukan hak istimewa tertentu untuk menjalankan perintah ini atau membatasi perubahan nama pengguna tertentu.

Dampak pada Objek: Penting untuk diingat bahwa mengubah nama pengguna dapat memiliki dampak pada objek-objek dalam basis data yang mungkin merujuk ke pengguna yang lama. Anda perlu memeriksa dan memperbarui objek-objek ini jika diperlukan.

Ketergantungan pada DBMS: Ingatlah bahwa sintaksis dan kemampuan ALTER USER untuk mengubah nama pengguna dapat berbeda-beda antara DBMS yang berbeda. Pastikan untuk merujuk pada dokumentasi DBMS Anda untuk detail yang lebih spesifik.

Pernyataan ALTER USER untuk mengubah nama pengguna adalah alat yang berguna dalam manajemen pengguna basis data, terutama ketika Anda perlu mengganti nama pengguna tanpa harus membuat ulang pengguna baru dan memindahkan izin dan objek terkait.

5. **Menghapus Pengguna:**
   DROP USER username;
   Ini akan menghapus pengguna beserta izin-izinnya.
   Tujuan Pernyataan DROP USER: Pernyataan DROP USER dalam SQL digunakan untuk menghapus pengguna (user) dari basis data. Ketika Anda tidak lagi memerlukan akses pengguna ke basis data, Anda dapat menggunakan perintah ini untuk menghapus pengguna beserta izin-izinnya.
   Batasan dan Variasi: Perintah DROP USER tidak selalu ada di semua sistem manajemen basis data (DBMS), dan sintaksisnya bisa bervariasi antara DBMS yang berbeda. Pastikan untuk merujuk pada dokumentasi DBMS Anda untuk memahami cara kerjanya dalam DBMS tertentu.
   Sintaksis Dasar:
   DROP USER username;
   username: Nama pengguna yang ingin Anda hapus dari basis data.
   Contoh Penggunaan:
   Misalnya, jika Anda ingin menghapus pengguna "john":
   Ini akan menghapus pengguna "john" beserta semua izin yang dimilikinya dalam basis data.
   Hapus Izin Terlebih Dahulu: Sebelum Anda dapat menghapus pengguna, pastikan Anda telah mencabut semua izin yang dimilikinya. Ini mencegah izin-izin yang diberikan oleh pengguna tetap ada dalam basis data setelah pengguna dihapus.
   Objek yang Merujuk ke Pengguna: Pastikan bahwa tidak ada objek lain dalam basis data yang masih merujuk ke pengguna

yang akan dihapus. Ini termasuk tampilan, prosedur, atau fungsi yang menggunakan pengguna sebagai pemilik atau pengguna dalam definisinya.

Ketergantungan pada DBMS: Ingatlah bahwa sintaksis dan kemampuan DROP USER dapat berbeda-beda antara DBMS yang berbeda. Beberapa DBMS mungkin memiliki parameter tambahan atau persyaratan tertentu saat menghapus pengguna.

Pertimbangan Keamanan: Pastikan untuk memastikan bahwa pengguna yang dihapus tidak lagi memiliki akses yang diperlukan atau relevan ke basis data. Ini penting untuk menjaga keamanan data.

Rekam Jejak: Beberapa DBMS mungkin memiliki rekam jejak atau log yang mencatat tindakan menghapus pengguna. Ini adalah praktik baik untuk mencatat dan memonitor tindakan semacam ini.

Pernyataan DROP USER adalah alat penting dalam manajemen pengguna basis data yang memungkinkan Anda menghapus pengguna yang tidak lagi diperlukan dan memastikan keamanan dan konsistensi basis data.

6. **Menampilkan Pengguna yang Ada:**

SELECT * FROM information_schema.users;

Ini akan menampilkan daftar pengguna yang ada dalam basis data.

Tujuan Pernyataan SQL: Pernyataan SELECT * FROM information_schema.users dalam SQL digunakan untuk mengambil seluruh data yang ada dalam tabel sistem informasi pengguna (information_schema.users). Tabel ini berisi informasi tentang pengguna dan hak akses mereka dalam basis data.

Tabel Sistem Informasi Pengguna: Tabel sistem informasi pengguna adalah tabel yang ada dalam basis data yang menyimpan metadata atau informasi tentang pengguna yang

memiliki akses ke basis data. Nama tabelnya dapat bervariasi tergantung pada sistem manajemen basis data (DBMS) yang digunakan. Misalnya, dalam PostgreSQL, tabel ini adalah pg_catalog.pg_user, sedangkan dalam MySQL, tabelnya adalah mysql.user.

Sintaksis Dasar:

SELECT * FROM information_schema.users;

*: Mengambil semua kolom dari tabel information_schema.users.

information_schema.users: Nama tabel sistem informasi pengguna dalam basis data.

Hasil Query: Perintah ini akan mengambil dan menampilkan semua data yang ada dalam tabel information_schema.users. Hasilnya akan berupa semua kolom dalam tabel, termasuk informasi seperti nama pengguna (username), jenis izin (permission_type), alamat email (email), dan atribut pengguna lainnya.

Ketergantungan pada DBMS: Nama tabel dan kolom yang ada dalam tabel ini dapat bervariasi antara DBMS yang berbeda. Pastikan untuk merujuk pada dokumentasi DBMS Anda untuk menentukan tabel dan kolom yang sesuai.

Hak Akses: Pastikan Anda memiliki hak akses yang memadai untuk menjalankan pernyataan SELECT ini. Biasanya, perintah ini dapat dijalankan oleh pengguna dengan hak akses baca (read) ke tabel sistem informasi pengguna.

Perlindungan Data: Data dalam tabel sistem informasi pengguna adalah data sensitif dan harus dilindungi dengan baik. Pastikan untuk menjaga keamanan data ini dan memberikan hak akses hanya kepada pengguna yang memerlukan informasi tersebut.

Penggunaan yang Umum: Perintah SELECT * FROM information_schema.users sering digunakan oleh administrator basis data atau pengembang untuk memeriksa

daftar pengguna yang ada dalam sistem dan informasi penting tentang mereka.

Perintah ini adalah cara efektif untuk mengambil informasi tentang pengguna yang ada dalam basis data dan memahami hak akses serta atribut lainnya yang terkait dengan pengguna tersebut. Namun, perlakukan data pengguna dengan hati-hati untuk menjaga keamanan dan privasi.

7. **Menampilkan Izin Pengguna:**

   SHOW GRANTS FOR username;

   Ini akan menampilkan izin-izin yang dimiliki oleh pengguna tertentu.

   Tujuan Pernyataan SHOW GRANTS FOR: Pernyataan SHOW GRANTS FOR dalam SQL digunakan untuk menampilkan izin (permissions) yang dimiliki oleh pengguna atau peran (roles) dalam basis data. Ini memungkinkan Anda untuk melihat hak akses spesifik yang telah diberikan kepada pengguna atau peran tertentu.

   Sintaksis Dasar:

   SHOW GRANTS FOR user_or_role;

   user_or_role: Nama pengguna atau peran yang izinnya ingin Anda lihat.

   Contoh Penggunaan:

   Misalnya, jika Anda ingin melihat izin yang dimiliki oleh pengguna "john":

   SHOW GRANTS FOR 'john';

   Ini akan menampilkan daftar izin yang diberikan kepada pengguna "john" dalam basis data.

   Ketergantungan pada DBMS: Sintaksis dan fungsionalitas perintah SHOW GRANTS FOR dapat berbeda-beda antara sistem manajemen basis data (DBMS) yang berbeda. Oleh karena itu, perlu merujuk pada dokumentasi DBMS yang digunakan untuk memahami detail spesifik.

Hasil Query: Hasil dari perintah ini akan menampilkan pernyataan SQL yang merupakan izin yang diberikan kepada pengguna atau peran yang Anda tentukan. Misalnya, hasilnya mungkin termasuk pernyataan seperti GRANT SELECT ON database.table TO 'john'@'localhost';.

Hak Akses: Biasanya, perintah SHOW GRANTS FOR dapat dijalankan oleh pengguna dengan hak akses SHOW DATABASES atau hak serupa. Namun, dalam beberapa kasus, Anda mungkin memerlukan hak akses tambahan, terutama jika Anda ingin melihat izin pengguna lain.

Perlindungan Keamanan: Informasi tentang izin adalah informasi sensitif yang harus dilindungi dengan baik. Pastikan bahwa hanya pengguna yang sah yang dapat mengakses informasi ini dan hindari berbagi izin lebih dari yang diperlukan.

Konsultasi dengan Administrator: Biasanya, penggunaan perintah SHOW GRANTS FOR adalah tugas administrator basis data atau DBA (Database Administrator) yang memeriksa dan mengelola izin pengguna. Pastikan untuk berkoordinasi dengan administrator jika Anda memerlukan akses atau pemahaman lebih lanjut tentang izin dalam basis data.

Pernyataan SHOW GRANTS FOR adalah alat yang berguna dalam manajemen izin basis data yang memungkinkan Anda melihat izin-izin yang dimiliki oleh pengguna atau peran tertentu. Ini membantu dalam memahami dan mengaudit hak akses dalam basis data.

8. **Pengaturan Hak Akses Global:**

   GRANT permission_type ON *.* TO username;

   Ini memberikan izin global kepada pengguna untuk semua basis data dan tabel.

   Tujuan Pernyataan GRANT: Pernyataan GRANT dalam SQL digunakan untuk memberikan izin (permissions) tertentu kepada pengguna dalam basis data. Pernyataan ini

memberikan penggunaan yang lebih tinggi atas berbagai jenis objek dalam basis data, seperti tabel, tampilan, atau prosedur.
Sintaksis Dasar:
GRANT permission_type ON database_name.* TO 'username'@'hostname';
permission_type: Izin yang ingin Anda berikan. Ini bisa berupa satu atau lebih jenis izin yang dipisahkan dengan koma, misalnya, SELECT, INSERT, UPDATE.
database_name: Nama basis data (database) yang izin akan diberikan. Tanda bintang (*) pada kedua sisi titik (.) mengindikasikan semua tabel dan objek dalam basis data tersebut.
username: Nama pengguna yang akan diberikan izin.
hostname: Alamat host yang diizinkan untuk pengguna. Ini bisa berupa alamat IP atau nama host.
Contoh Penggunaan:
Misalnya, jika Anda ingin memberikan izin kepada pengguna "john" untuk melakukan SELECT dan INSERT pada semua tabel dalam basis data "mydb" dari host "localhost":
GRANT SELECT, INSERT ON mydb.* TO 'john'@'localhost';
Ini akan memberikan izin kepada pengguna "john" untuk membaca dan menambahkan data ke semua tabel dalam basis data "mydb" dari host "localhost."
Kombinasi Izin: Anda dapat memberikan berbagai jenis izin sekaligus, seperti SELECT, INSERT, UPDATE, DELETE, dan lainnya. Sesuaikan izin yang Anda berikan sesuai dengan kebutuhan aplikasi atau pengguna.
Hostname: Menentukan host dalam pernyataan GRANT adalah penting untuk mengendalikan dari mana pengguna dapat mengakses basis data. Dengan menentukan host, Anda dapat membatasi akses ke basis data hanya dari lokasi tertentu.

Pengaturan IP: Sebagai alternatif, Anda dapat menggunakan alamat IP sebagai host untuk membatasi akses berdasarkan alamat IP tertentu.

Wildcards: Anda juga dapat menggunakan wildcard seperti % untuk host jika Anda ingin memberikan izin dari host apa pun. Misalnya, 'john'@'%' akan memungkinkan akses dari semua host.

Revoke Izin: Jika Anda perlu mencabut izin yang telah diberikan sebelumnya, Anda dapat menggunakan perintah REVOKE. Misalnya, untuk mencabut izin SELECT dari pengguna "john":

REVOKE SELECT ON mydb.* FROM 'john'@'localhost';

Perlindungan Keamanan: Pastikan untuk memberikan izin hanya kepada pengguna yang memerlukan akses tersebut dan selalu mengamankan kata sandi dan konfigurasi host untuk melindungi keamanan basis data.

Ketergantungan pada DBMS: Ingatlah bahwa sintaksis dan kemampuan GRANT dapat berbeda-beda antara DBMS yang berbeda. Pastikan untuk merujuk pada dokumentasi DBMS yang Anda gunakan untuk detail yang lebih spesifik.

Perintah GRANT adalah alat penting dalam manajemen izin basis data yang memungkinkan Anda mengendalikan hak akses pengguna dan memastikan bahwa pengguna hanya memiliki izin yang sesuai dengan kebutuhan aplikasi atau tugas mereka.

9. **Pemberian Izin Spesifik pada Basis Data:**

GRANT permission_type ON database_name.* TO username;

Ini memberikan izin tertentu pada semua tabel dalam basis data tertentu.

Tujuan Pernyataan GRANT: Pernyataan GRANT dalam SQL digunakan untuk memberikan izin (permissions) tertentu kepada pengguna dalam basis data. Ini memungkinkan

pengguna untuk melakukan tindakan tertentu seperti membaca, menulis, atau mengelola objek dalam basis data.
Sintaksis Dasar:
GRANT permission_type ON database_name.* TO 'username'@'hostname';
permission_type: Jenis izin yang ingin Anda berikan. Ini bisa berupa salah satu dari berbagai jenis izin, seperti SELECT, INSERT, UPDATE, DELETE, dan sebagainya. Anda juga dapat memberikan izin yang lebih umum seperti ALL PRIVILEGES untuk memberikan semua jenis izin.
database_name: Nama basis data (database) yang akan menerima izin. Tanda bintang (*) pada kedua sisi titik (.) mengindikasikan bahwa izin diberikan untuk semua objek (tabel, tampilan, dll.) dalam basis data tersebut.
username: Nama pengguna yang akan diberikan izin.
hostname: Alamat host dari mana pengguna diizinkan untuk mengakses basis data. Ini bisa berupa alamat IP atau nama host.
Contoh Penggunaan:
Misalnya, jika Anda ingin memberikan izin SELECT dan INSERT pada semua objek dalam basis data "mydb" kepada pengguna "john" dari host "localhost":
GRANT SELECT, INSERT ON mydb.* TO 'john'@'localhost';
Ini akan memberikan izin kepada pengguna "john" untuk membaca dan menambahkan data ke semua objek dalam basis data "mydb" dari host "localhost."
Kombinasi Izin: Anda dapat memberikan berbagai jenis izin sekaligus, seperti SELECT, INSERT, UPDATE, DELETE, dan lainnya, sesuai dengan kebutuhan aplikasi atau pengguna.
Hostname: Menentukan host dalam pernyataan GRANT adalah penting untuk mengendalikan dari mana pengguna dapat mengakses basis data. Dengan menentukan host, Anda dapat membatasi akses ke basis data hanya dari lokasi tertentu.

Wildcards: Jika Anda ingin memberikan izin dari host apa pun, Anda dapat menggunakan wildcard seperti % dalam host. Misalnya, 'john'@'%' akan memungkinkan akses dari semua host.
Revoke Izin: Jika Anda perlu mencabut izin yang telah diberikan sebelumnya, Anda dapat menggunakan perintah REVOKE. Misalnya, untuk mencabut izin SELECT dari pengguna "john":
REVOKE SELECT ON mydb.* FROM 'john'@'localhost';
Perlindungan Keamanan: Pastikan Anda memberikan izin hanya kepada pengguna yang memerlukan akses tersebut dan selalu mengamankan kata sandi dan konfigurasi host untuk melindungi keamanan basis data.
Ketergantungan pada DBMS: Ingatlah bahwa sintaksis dan kemampuan GRANT dapat berbeda-beda antara DBMS yang berbeda. Pastikan untuk merujuk pada dokumentasi DBMS yang Anda gunakan untuk detail yang lebih spesifik.
Perintah GRANT adalah alat penting dalam manajemen izin basis data yang memungkinkan Anda mengendalikan hak akses pengguna dan memastikan bahwa pengguna hanya memiliki izin yang sesuai dengan kebutuhan aplikasi atau tugas mereka.

Pastikan untuk mengganti username, password, permission_type, database_name, dan table_name sesuai dengan kebutuhan Anda. Selain itu, perlu diingat bahwa istilah dan sintaksis yang digunakan dapat bervariasi tergantung pada sistem manajemen basis data (misalnya MySQL, PostgreSQL, SQL Server) yang Anda gunakan.

DAFTAR PUSTAKA

BAB 12
BASIS DATA NOSQL

Oleh Suwito Pomalingo

12.1 Pengantar ke NoSQL

12.1.1 Apa itu NoSQL?

NoSQL, yang berarti "Not Only SQL," adalah jenis basis data yang dirancang untuk menyediakan cara yang lebih fleksibel, skalabel, dan performa tinggi untuk menyimpan dan mengelola data dibandingkan dengan sistem basis data relasional tradisional yang menggunakan SQL (*Structured Query Language*). Istilah ini pertama kali muncul pada akhir tahun 1990-an tetapi menjadi lebih populer pada dekade berikutnya dengan munculnya aplikasi web skala besar dan kebutuhan untuk menangani Big Data. Menunrut Stonebrakse, NoSQL, singkatan dari "Not Only SQL," adalah kategori dari sistem manajemen basis data yang menyediakan mekanisme untuk penyimpanan dan pengambilan data yang dimodelkan dengan cara selain dari relasi tabular yang digunakan dalam basis data relasional (Stonebraker *et al.*, 2010)

Database NoSQL memanfaatkan berbagai macam struktur data untuk mengelola dan mengakses informasi. Database jenis ini dirancang khusus untuk mendukung aplikasi yang membutuhkan kapasitas data yang besar, kecepatan akses yang tinggi, dan fleksibilitas dalam struktur data. Ini dicapai dengan meminimalisir batasan konsistensi data yang biasanya ada pada database konvensional.

Mari kita pertimbangkan sebuah contoh sederhana tentang pemodelan skema untuk database yang berisi informasi tentang buku.

Dalam konteks database relasional, informasi tentang buku biasanya dinormalisasi dan disimpan dalam beberapa tabel terpisah. Hubungan antar-tabel diatur melalui kunci primer dan kunci asing. Misalnya, tabel 'Buku' mungkin memiliki kolom untuk ISBN, Judul, dan Edisi, sementara tabel 'Penulis' memiliki kolom untuk ID Penulis dan Nama Penulis. Tabel relasional lain, yang bisa disebut 'Penulis-ISBN', akan menghubungkan ID Penulis dengan ISBN buku. Pendekatan ini dirancang untuk memastikan integritas data antar-tabel, mengurangi duplikasi data, dan biasanya dioptimalkan untuk efisiensi penyimpanan.

Sebaliknya, dalam database NoSQL, informasi tentang sebuah buku biasanya disimpan dalam satu dokumen JSON. Dalam dokumen ini, semua informasi relevan seperti ISBN, Judul Buku, Edisi, Nama Penulis, dan ID Penulis akan disimpan sebagai atribut dalam satu entitas data. Pendekatan ini memprioritaskan kemudahan pengembangan dan kemampuan untuk diskalakan secara horizontal.

12.1.2 Sejarah dan Evolusi NoSQL

Sejarah NoSQL tidak bisa dilepaskan dari kebutuhan untuk menangani volume data yang semakin besar dan kompleksitas yang semakin meningkat dalam dunia komputasi. Meskipun konsep NoSQL pertama kali muncul pada akhir tahun 1990-an, istilah ini menjadi lebih populer dan mendapatkan traksi pada awal 2000-an dengan munculnya aplikasi web skala besar dan kebutuhan untuk menangani Big Data (Cattell, 2011). Awalnya, NoSQL digunakan sebagai solusi untuk mengatasi keterbatasan database relasional, terutama dalam hal skalabilitas horizontal dan fleksibilitas schema.

Pada pertengahan 2000-an, perusahaan teknologi besar seperti Google dan Amazon memulai pengembangan sistem penyimpanan data terdistribusi yang nantinya akan menjadi cikal bakal dari beberapa jenis database NoSQL. Google merilis paper

tentang Bigtable pada tahun 2006 (Chang *et al.*, 2008), sementara Amazon merilis paper tentang Dynamo pada tahun 2007 (DeCandia *et al.*, 2007). Kedua paper ini memberikan fondasi teoritis dan praktis untuk pengembangan database NoSQL dan mempengaruhi banyak proyek open-source yang muncul kemudian, seperti Apache Cassandra dan MongoDB.

Dekade terakhir telah melihat perkembangan pesat dalam ekosistem NoSQL. Berbagai jenis database NoSQL telah dikembangkan, masing-masing dengan keunggulan dan kekurangannya sendiri. Misalnya, database berorientasi dokumen seperti MongoDB menjadi populer untuk aplikasi yang memerlukan fleksibilitas schema, sementara database berorientasi grafik seperti Neo4j digunakan untuk analisis hubungan kompleks (Robinson, Webber and Eifrem, 2015). Selain itu, NoSQL juga mulai mendapatkan penerimaan di industri di luar teknologi, termasuk keuangan, perawatan kesehatan, dan pemerintahan.

Dalam konteks ini, NoSQL telah berkembang dari menjadi alternatif niche menjadi bagian penting dari arsitektur data modern. Dengan kebutuhan untuk *real-time analytics, Internet of Things* (IoT), dan aplikasi mobile yang memerlukan latensi rendah dan skalabilitas tinggi, NoSQL terus beradaptasi dan berkembang untuk memenuhi kebutuhan ini (Leavitt, 2010). Dengan demikian, NoSQL tidak hanya menjadi solusi untuk masalah spesifik tetapi juga menjadi enabler untuk inovasi dalam pengelolaan data.

12.1.3 Perbedaan antara SQL dan NoSQL

Selama beberapa dekade, model data relasional telah menjadi standar dalam pengembangan aplikasi, digunakan oleh sistem basis data seperti Oracle, DB2, SQL Server, MySQL, dan PostgreSQL. Namun, dari pertengahan hingga akhir 2000-an, model data alternatif mulai mendapat penerimaan dan penggunaan yang luas. Istilah "NoSQL" kemudian muncul untuk membedakan dan mengklasifikasikan jenis-jenis database dan

model data baru ini. Istilah "NoSQL" sering digunakan secara sinonim dengan "nonrelasional."

Walaupun ada beragam tipe database NoSQL dengan fitur yang berbeda-beda, berikut beberapa hal yang membedakan antara database SQL dan NoSQL.

1. Struktur Data
   SQL: Menggunakan struktur tabel relasional yang ketat, di mana skema harus didefinisikan terlebih dahulu sebelum data dapat disimpan. Setiap tabel memiliki kolom yang telah ditentukan dan setiap baris adalah sebuah rekaman.
   NoSQL: Menawarkan struktur data yang lebih fleksibel, seperti key-value, document-oriented, column-family, atau graph databases. Skema bisa dinamis dan berubah sesuai kebutuhan.
2. Skalabilitas
   SQL: Biasanya diskalakan secara vertikal, yang berarti jika Anda perlu menangani lebih banyak beban, Anda biasanya perlu meningkatkan spesifikasi mesin (CPU, RAM, Storage).
   NoSQL: Dirancang untuk skalabilitas horizontal, memungkinkan penambahan mesin ke dalam jaringan untuk meningkatkan kapasitas dan performa.
3. Konsistensi dan Transaksi
   SQL: Menawarkan transaksi ACID (*Atomicity, Consistency, Isolation, Durability*), yang memastikan keandalan dalam setiap transaksi.
   NoSQL: Kebanyakan database NoSQL mengikuti model konsistensi eventual dan tidak mendukung transaksi ACID secara penuh, meskipun beberapa database NoSQL modern mulai menambahkan fitur ini.
4. Bahasa Query
   SQL: Memiliki bahasa query standar, yaitu SQL (*Structured Query Language*), yang digunakan untuk mendefinisikan dan memanipulasi data.

NoSQL: Tidak memiliki bahasa query standar. Setiap database NoSQL biasanya memiliki bahasa query atau API sendiri untuk interaksi dengan data.

5. Relasi dan Join
   SQL: Mendukung operasi join yang kompleks antar tabel, memungkinkan penggabungan data dari berbagai sumber dalam satu query.
   NoSQL: Kebanyakan tidak mendukung operasi join atau mendukungnya dengan cara yang lebih terbatas dibandingkan SQL.
6. Keamanan
   SQL: Umumnya memiliki fitur keamanan yang lebih matang, termasuk dukungan untuk autentikasi dan otorisasi tingkat lanjut.
   NoSQL: Meskipun menawarkan fitur keamanan, tetapi biasanya tidak sekomprehensif SQL.
7. Kasus Penggunaan
   SQL: Lebih cocok untuk aplikasi yang memerlukan struktur data yang kompleks dan relasi antar tabel, seperti sistem manajemen, aplikasi keuangan, dan lain-lain.
   NoSQL: Lebih cocok untuk aplikasi yang memerlukan skalabilitas tinggi, latensi rendah, dan kemampuan untuk menangani data yang tidak terstruktur atau semi-terstruktur, seperti aplikasi media sosial, *real-time analytics, dan Internet of Things* (IoT).

Kedua jenis database ini memiliki kelebihan dan kekurangan masing-masing, dan pilihan antara SQL dan NoSQL seringkali bergantung pada kebutuhan spesifik proyek atau aplikasi.

12.1.4 Kenapa harus menggunakan database NoSQL?

Database NoSQL adalah pilihan yang ideal untuk aplikasi modern termasuk aplikasi mobile, web, dan permainan video, karena menawarkan fleksibilitas, skalabilitas, performa tinggi, dan fitur-fitur canggih yang diperlukan untuk memberikan pengalaman pengguna yang memuaskan.

Berikut beberapa alasan kenapa harus menggunakan database NoSQL :

1. Fleksibilitas Struktur Data
   Salah satu keuntungan utama dari database NoSQL adalah fleksibilitas dalam struktur data. Tidak seperti database relasional yang memerlukan skema yang tetap, NoSQL memungkinkan Anda untuk menambahkan atau mengubah field dengan mudah. Ini sangat berguna dalam aplikasi yang memerlukan adaptasi cepat terhadap perubahan kebutuhan atau di mana struktur data belum sepenuhnya didefinisikan.
2. Skalabilitas Horizontal
   Database NoSQL dirancang untuk skalabilitas horizontal, yang memungkinkan penambahan mesin ke dalam jaringan untuk meningkatkan kapasitas dan performa. Ini berbeda dari database relasional yang biasanya diskalakan secara vertikal, yang bisa menjadi lebih mahal dan kompleks. Skalabilitas horizontal memungkinkan NoSQL menjadi pilihan yang lebih efisien untuk aplikasi dengan volume data yang besar atau yang mengalami pertumbuhan data yang cepat.
3. Performa dan Latensi Rendah
   NoSQL biasanya menawarkan latensi baca dan tulis yang lebih rendah dibandingkan dengan database relasional, terutama dalam beban kerja yang melibatkan akses ke sejumlah besar data dengan pola akses yang tidak terduga. Ini membuatnya ideal untuk aplikasi real-time, analitik, dan *Internet of Things* (IoT) yang memerlukan respon cepat.

4. Kemudahan dalam Menangani Data Tidak Terstruktur
   Dalam era Big Data, kemampuan untuk menangani data yang tidak terstruktur atau semi-terstruktur adalah penting. NoSQL sangat cocok untuk menyimpan jenis data seperti ini, termasuk data JSON, XML, atau data dari sensor IoT. Ini memungkinkan organisasi untuk memanfaatkan lebih banyak jenis data untuk mendapatkan wawasan yang lebih baik.

Secara keseluruhan, database NoSQL menawarkan fleksibilitas, skalabilitas, dan performa yang sering kali lebih sesuai untuk kebutuhan aplikasi modern dibandingkan dengan database relasional. Namun, penting untuk mempertimbangkan kebutuhan spesifik proyek Anda, karena NoSQL tidak selalu merupakan solusi yang paling tepat untuk setiap kasus.

12.2 Jenis-Jenis Basis Data NoSQL

12.2.1 Document-oriented

Document-oriented database adalah salah satu jenis basis data NoSQL yang dirancang untuk menyimpan, mengelola, dan mengambil data dalam bentuk dokumen. Berbeda dengan basis data relasional yang menggunakan tabel untuk menyimpan data, document-oriented database menggunakan dokumen biasanya dalam format JSON, BSON, atau XML. Setiap dokumen ini berfungsi sebagai unit penyimpanan data yang independen dan dapat berisi berbagai jenis informasi, termasuk data terstruktur, semi-terstruktur, atau bahkan data yang tidak terstruktur.

Untuk menggunakan document-oriented database, Anda pertama-tama perlu memilih platform database yang sesuai, seperti MongoDB, Couchbase, atau RavenDB. Setelah itu, Anda dapat mulai menyimpan data dalam bentuk dokumen. Operasi CRUD (Create, Read, Update, Delete) biasanya dilakukan melalui API atau query language yang disediakan oleh platform tersebut.

Misalnya, dalam MongoDB, Anda bisa menggunakan bahasa query khusus MongoDB untuk melakukan operasi ini. Selain itu, basis data ini juga mendukung fitur seperti indeksasi, pencarian teks penuh, dan agregasi data.

Sebagai contoh, misalkan Anda memiliki aplikasi e-commerce dan ingin menyimpan informasi tentang produk. Dalam document-oriented database seperti MongoDB, Anda bisa memiliki sebuah dokumen untuk setiap produk, yang berisi informasi seperti nama produk, harga, deskripsi, dan ulasan dari pelanggan. Dokumen ini bisa dengan mudah di-update atau dihapus tanpa mempengaruhi dokumen lainnya dalam database.

Contoh sederhana implementasi database document-oriented menggunakan MongoDB dan bahasa pemrograman Python. Dalam contoh ini, kita akan menggunakan library pymongo untuk berinteraksi dengan MongoDB.

```
from pymongo import MongoClient

# Koneksi ke server MongoDB
client = MongoClient('mongodb://localhost:27017/')

# Membuat atau memilih database bernama 'ecommerce'
db = client['ecommerce']

# Membuat atau memilih koleksi bernama 'products'
products = db['products']

# Menambahkan satu dokumen (produk) ke dalam koleksi 'products'
new_product = {
  "name": "Laptop",
  "price": 1000,
  "description": "A high-quality laptop",
  "reviews": [
```

```
        {"user": "Alice", "comment": "Excellent!", "rating": 5},
        {"user": "Bob", "comment": "Pretty good", "rating": 4}
    ]
}

insert_result = products.insert_one(new_product)
print(f'Inserted product with ID: {insert_result.inserted_id}")

# Membaca dokumen dari koleksi 'products'
for product in products.find():
    print(product)

# Update dokumen
update_result = products.update_one(
    {"name": "Laptop"},
    {"$set": {"price": 900}}
)
print(f'Modified {update_result.modified_count} document")

# Hapus dokumen
delete_result = products.delete_one({"name": "Laptop"})
print(f'Deleted {delete_result.deleted_count} document")
```

Pada contoh ini, akan dilakukan operasi CRUD sederhana:

1. Membuat sebuah dokumen baru yang mewakili sebuah produk.
2. Membaca dan menampilkan semua produk dari database.
3. Mengupdate harga dari produk "Laptop".
4. Menghapus produk "Laptop" dari database.

Kelebihan utama dari document-oriented database adalah fleksibilitasnya dalam menangani berbagai jenis data dan struktur yang kompleks. Ini memungkinkan pengembang untuk lebih cepat

dalam iterasi dan penyesuaian schema. Namun, ada juga kekurangan, seperti potensi inkonsistensi data dan kompleksitas dalam menjaga integritas transaksional antar dokumen. Oleh karena itu, penting untuk mempertimbangkan kebutuhan spesifik proyek Anda sebelum memilih menggunakan document-oriented database.

12.2.2 Key-value stores

Key-value stores adalah salah satu jenis basis data NoSQL yang paling sederhana dan efisien dalam hal penyimpanan dan pengambilan data. Seperti namanya, model ini menggunakan pasangan kunci-nilai (*key-value*) untuk menyimpan data. Kunci adalah identifikasi unik yang digunakan untuk mengakses nilai yang terkait. Nilai bisa berupa tipe data sederhana seperti string atau integer, atau struktur data lebih kompleks seperti JSON, array, atau bahkan blob biner. Beberapa contoh basis data key-value yang populer adalah Redis, Amazon DynamoDB, dan Riak.

Menggunakan key-value stores biasanya sangat sederhana. Anda hanya perlu menentukan kunci dan nilai yang ingin disimpan. Operasi dasarnya meliputi GET untuk mengambil nilai berdasarkan kunci, SET untuk menyimpan pasangan kunci-nilai, dan DELETE untuk menghapus entri berdasarkan kunci. Karena strukturnya yang sederhana, key-value stores sering digunakan untuk caching, penyimpanan sesi, dan aplikasi yang membutuhkan akses data berkecepatan tinggi.

Sebagai contoh, kita bisa menggunakan Redis untuk implementasi key-value store. Redis adalah basis data in-memory yang sangat cepat dan mendukung berbagai jenis struktur data. Berikut adalah contoh kode Python menggunakan library redis-py:

```
import redis

# Membuat koneksi ke Redis server
r = redis.Redis(host='localhost', port=6379, db=0)

# Menyimpan pasangan kunci-nilai
r.set('name', 'John')

# Mengambil nilai berdasarkan kunci
print(f"Name: {r.get('name').decode('utf-8')}")

# Menghapus kunci
r.delete('name')
```

Kelebihan utama dari key-value stores adalah kecepatan dan skalabilitas. Mereka sangat cepat untuk operasi baca dan tulis, dan mudah untuk diskalakan secara horizontal. Namun, mereka kurang cocok untuk operasi yang membutuhkan query kompleks atau relasi antar data. Selain itu, karena biasanya tidak mendukung ACID transactions, Anda mungkin perlu mengimplementasikan logika transaksional pada tingkat aplikasi.

12.2.3 Column-family stores

Column-family stores adalah salah satu jenis basis data NoSQL yang dirancang untuk menyimpan data dalam bentuk kolom daripada baris, yang berbeda dengan model relasional tradisional. Dalam model ini, data disimpan dalam keluarga kolom (column families), di mana setiap keluarga kolom berisi sejumlah kolom yang terkait. Model ini sangat efisien untuk operasi baca dan tulis pada skala besar dan sering digunakan dalam aplikasi yang membutuhkan kinerja tinggi untuk baca/tulis data. Contoh dari

basis data column-family adalah Apache Cassandra, HBase, dan ScyllaDB.

Untuk menggunakan basis data column-family, Anda pertama-tama perlu mendefinisikan skema yang mencakup keluarga kolom dan kolom yang terkait. Setelah itu, Anda bisa melakukan operasi CRUD (*Create, Read, Update, Delete*) pada data. Biasanya, Anda akan menggunakan bahasa query khusus atau API yang disediakan oleh platform tersebut. Misalnya, dalam Apache Cassandra, Anda bisa menggunakan bahasa query CQL (*Cassandra Query Language*) untuk berinteraksi dengan data.

Sebagai contoh, kita akan menggunakan Apache Cassandra dan bahasa query CQL. Pertama, kita membuat sebuah tabel dengan keluarga kolom:

```
CREATE TABLE users (
  user_id UUID PRIMARY KEY,
  first_name TEXT,
  last_name TEXT,
  email TEXT
);
```

Kemudian, kita bisa memasukkan data ke dalam tabel:

```
INSERT INTO users (user_id, first_name, last_name, email) VALUES (uuid(), 'John', 'Doe', 'john.doe@example.com');
```

Dan untuk membaca data:

```
SELECT * FROM users WHERE user_id = ?;
```

Kelebihan dari column-family stores termasuk skalabilitas horizontal yang tinggi, latensi rendah untuk operasi baca/tulis, dan kemampuan untuk menangani data yang sangat besar. Namun, model ini bisa menjadi kompleks dan memerlukan pemahaman yang mendalam tentang bagaimana data disimpan dan diakses.

Selain itu, karena fokusnya pada kinerja, beberapa fitur seperti transaksi ACID mungkin tidak selalu tersedia.

12.2.4 Graph databases

Graph databases adalah jenis basis data NoSQL yang dirancang untuk menangani data dalam bentuk graf. Dalam model ini, data disimpan sebagai simpul (nodes) dan hubungan (edges) antar simpul. Setiap simpul dan hubungan bisa memiliki atribut atau properti yang menambah detail atau konteks pada data. Graph databases sangat efisien dalam menjawab pertanyaan yang melibatkan hubungan antar data, seperti jaringan sosial, sistem rekomendasi, atau analisis jaringan. Beberapa contoh dari basis data graf adalah Neo4j, ArangoDB, dan OrientDB.

Untuk menggunakan graph databases, Anda pertama-tama perlu mendefinisikan simpul dan hubungan dalam graf Anda. Setelah itu, Anda bisa melakukan operasi CRUD (*Create, Read, Update, Delete*) pada simpul dan hubungan tersebut. Biasanya, Anda akan menggunakan bahasa query khusus atau API yang disediakan oleh platform tersebut. Misalnya, dalam Neo4j, Anda bisa menggunakan bahasa query Cypher untuk berinteraksi dengan data.

Sebagai contoh, kita akan menggunakan Neo4j untuk membuat sebuah graf sederhana yang merepresentasikan hubungan antar orang. Pertama, kita membuat simpul untuk beberapa orang:

```
CREATE (Alice:Person {name: 'Alice'}),
    (Bob:Person {name: 'Bob'}),
    (Charlie:Person {name: 'Charlie'});
```

Kemudian, kita menambahkan hubungan antar mereka:

```
MATCH (a:Person {name: 'Alice'}), (b:Person {name: 'Bob'})
CREATE (a)-[r:FRIEND]->(b);
```

Untuk query, misalnya kita ingin menemukan semua teman dari Alice:

```
MATCH (a:Person {name: 'Alice'})-[r:FRIEND]->(b)
RETURN b.name;
```

Kelebihan dari graph databases adalah kemampuannya untuk dengan mudah dan cepat menjelajah hubungan antar data, yang bisa menjadi sangat kompleks dalam basis data lain. Ini sangat berguna dalam aplikasi yang membutuhkan analisis hubungan yang mendalam. Namun, kekurangannya termasuk kompleksitas dalam desain dan implementasi, serta potensi overhead kinerja untuk operasi yang tidak memanfaatkan kekuatan graf.

12.3 Arsitektur dan Model Data

12.3.1 Bagaimana NoSQL disimpan dan diorganisir

Dalam basis data NoSQL, arsitektur dan model data sangat beragam, tergantung pada jenis basis data NoSQL yang digunakan. Namun, ada beberapa prinsip umum yang sering diterapkan. Pertama, NoSQL biasanya lebih fleksibel dalam hal skema data; Anda seringkali tidak perlu mendefinisikan struktur tabel atau kolom sebelum menyimpan data. Kedua, NoSQL biasanya dirancang untuk skalabilitas horizontal, yang berarti Anda bisa menambahkan lebih banyak mesin ke dalam kluster untuk meningkatkan kapasitas dan kinerja. Ketiga, NoSQL sering menggunakan partisi data dan replikasi untuk meningkatkan ketersediaan dan ketahanan.

Untuk memahami bagaimana data disimpan dan diorganisir dalam NoSQL, Anda perlu memahami konsep-konsep seperti partisi, replikasi, dan konsistensi. Partisi adalah teknik membagi data ke dalam beberapa bagian yang bisa disimpan di server yang berbeda. Replikasi adalah proses menyalin data ke beberapa lokasi untuk meningkatkan ketersediaan. Konsistensi mengacu pada bagaimana perubahan data disinkronkan antar partisi dan replika.

Setiap jenis basis data NoSQL memiliki cara sendiri dalam mengimplementasikan konsep-konsep ini.

Sebagai contoh, dalam Apache Cassandra, data disimpan dalam bentuk partisi yang didistribusikan di seluruh node dalam kluster. Cassandra menggunakan model konsistensi yang disebut "*eventual consistency*," di mana perubahan data akan disebarkan ke semua replika seiring waktu. Anda bisa menentukan tingkat konsistensi saat membaca atau menulis data, yang memungkinkan Anda untuk menyeimbangkan antara ketersediaan dan konsistensi.

```
// Menulis data dengan tingkat konsistensi QUORUM
CONSISTENCY QUORUM;
INSERT INTO table (key, value) VALUES ('key1', 'value1');

// Membaca data dengan tingkat konsistensi ONE
CONSISTENCY ONE;
SELECT * FROM table WHERE key = 'key1';
```

Memahami bagaimana data disimpan dan diorganisir dalam NoSQL adalah kunci untuk memilih jenis basis data yang paling sesuai dengan kebutuhan aplikasi Anda. Faktor-faktor seperti skalabilitas, ketersediaan, dan konsistensi perlu dipertimbangkan dengan cermat. Selain itu, setiap jenis basis data NoSQL memiliki kelebihan dan kekurangan sendiri dalam hal arsitektur dan model data, sehingga pemilihan yang tepat akan sangat bergantung pada kasus penggunaan spesifik Anda.

12.3.2 CAP Theorem

CAP Theorem, yang diusulkan oleh Eric Brewer pada tahun 2000, adalah sebuah prinsip dasar yang digunakan untuk memahami dan membandingkan sifat dari sistem basis data terdistribusi. CAP sendiri adalah akronim dari *Consistency* (Konsistensi), *Availability* (Ketersediaan), dan *Partition Tolerance*

(Ketahanan Partisi). Teorema ini menyatakan bahwa dalam sebuah sistem basis data terdistribusi, hanya dua dari tiga sifat ini yang bisa dicapai secara bersamaan. Misalnya, sebuah sistem yang fokus pada konsistensi dan ketersediaan mungkin akan mengalami masalah saat ada kegagalan partisi.

Berbagai jenis basis data NoSQL biasanya mengoptimalkan dua dari tiga aspek dalam CAP Theorem. Sebagai contoh, Apache Cassandra mengoptimalkan untuk *Availability* dan *Partition Tolerance* tetapi mengorbankan konsistensi ketat (*eventual consistency*). Di sisi lain, basis data seperti HBase lebih fokus pada Consistency dan Partition Tolerance tetapi mungkin mengorbankan ketersediaan.

Sebagai contoh, dalam sistem menggunakan Apache Cassandra, Anda bisa menentukan tingkat konsistensi saat query, yang memungkinkan Anda untuk menyeimbangkan antara ketersediaan dan konsistensi sesuai kebutuhan aplikasi. Misalnya, Anda bisa menggunakan tingkat konsistensi QUORUM untuk menulis atau membaca data, yang memastikan bahwa operasi akan dikonfirmasi oleh mayoritas node dalam kluster sebelum dianggap berhasil.

```
CONSISTENCY QUORUM;
INSERT INTO table (key, value) VALUES ('key1', 'value1');
```

Dalam kasus ini, jika sebagian node tidak tersedia atau mengalami partisi, operasi masih bisa berlanjut asalkan mayoritas node (quorum) dapat diakses.

Memahami CAP Theorem adalah penting untuk memilih dan mengkonfigurasi basis data NoSQL yang paling sesuai dengan kebutuhan aplikasi Anda. Ini akan membantu Anda memahami trade-off antara konsistensi, ketersediaan, dan ketahanan partisi, sehingga Anda bisa membuat keputusan yang lebih tepat sesuai dengan kasus penggunaan Anda.

12.3.3 Eventual Consistency vs Strong Consistency

Dalam konteks basis data terdistribusi, konsistensi mengacu pada sejauh mana semua replika data menunjukkan keadaan yang sama sepanjang waktu. Eventual Consistency dan Strong Consistency adalah dua model konsistensi yang sering digunakan. Dalam model Eventual Consistency, sistem menjamin bahwa jika tidak ada operasi baru yang dilakukan pada data, semua replika akan akhirnya berada dalam keadaan yang sama. Namun, ini bisa memakan waktu. Sebaliknya, dalam model Strong Consistency, setiap operasi baca akan mengembalikan nilai terbaru yang ditulis, sehingga semua replika selalu dalam keadaan yang sama.

Basis data NoSQL sering memilih antara Eventual dan Strong Consistency berdasarkan kebutuhan aplikasi dan arsitektur. Apache Cassandra, misalnya, biasanya menggunakan model Eventual Consistency dan memungkinkan pengguna untuk menentukan tingkat konsistensi saat query. Di sisi lain, Google Cloud Spanner menggunakan model Strong Consistency dengan memanfaatkan algoritma konsensus dan jam atom global untuk memastikan bahwa semua replika selalu sinkron.

Dalam Apache Cassandra, Anda bisa menentukan tingkat konsistensi seperti ONE, QUORUM, atau ALL saat melakukan operasi baca atau tulis. Misalnya, menggunakan tingkat konsistensi ONE berarti operasi akan dianggap berhasil jika setidaknya satu replika telah diperbarui.

```
CONSISTENCY ONE;
INSERT INTO table (key, value) VALUES ('key1', 'value1');
```

Di Google Cloud Spanner, Anda tidak perlu menentukan tingkat konsistensi karena sistem secara otomatis menjamin Strong Consistency. Setiap operasi baca atau tulis akan menunggu hingga semua replika diperbarui.

Pemilihan antara Eventual dan Strong Consistency akan sangat bergantung pada kebutuhan aplikasi Anda. Jika aplikasi Anda memerlukan bacaan yang paling up-to-date dan tidak bisa mentolerir inkonsistensi, maka Strong Consistency mungkin lebih sesuai. Namun, jika aplikasi Anda memerlukan skalabilitas tinggi dan bisa mentolerir sedikit latensi dalam konsistensi, maka Eventual Consistency bisa menjadi pilihan yang lebih baik.

12.4 Desain dan Model Data NoSQL

12.4.1 Schema-less Design

Dalam basis data relasional, skema atau struktur tabel dan kolom biasanya harus didefinisikan terlebih dahulu sebelum data dapat disimpan. Sebaliknya, dalam model Schema-less Design yang sering digunakan oleh basis data NoSQL, Anda memiliki fleksibilitas untuk menyimpan data tanpa harus mendefinisikan struktur awalnya. Ini memungkinkan Anda untuk dengan mudah menambahkan atau mengubah atribut data tanpa perlu melakukan migrasi atau perubahan struktural pada seluruh basis data. Fleksibilitas ini sangat berguna dalam aplikasi yang memerlukan iterasi cepat atau memiliki struktur data yang dinamis.

Basis data NoSQL seperti MongoDB, Couchbase, dan DynamoDB adalah contoh yang baik dari implementasi Schema-less Design. Dalam basis data ini, Anda bisa menyimpan dokumen atau entitas dengan atribut yang berbeda-beda dalam koleksi atau tabel yang sama. Misalnya, dalam satu koleksi MongoDB, satu dokumen bisa memiliki atribut nama dan alamat, sementara dokumen lain dalam koleksi yang sama bisa memiliki atribut nama, email, dan tanggal_lahir.

Sebagai contoh, dalam MongoDB, Anda bisa menyimpan dokumen dengan struktur yang berbeda dalam koleksi yang sama tanpa perlu mendefinisikan skema terlebih dahulu:

```
// Menyimpan dokumen dengan atribut nama dan alamat
db.users.insert({name: "Alice", address: "123 Main St"});

// Menyimpan dokumen dengan atribut nama, email, dan tanggal_lahir
db.users.insert({name: "Bob", email: "bob@email.com", birthdate: "1990-01-01"});
```

Kedua dokumen ini bisa disimpan dalam koleksi yang sama (users) meskipun memiliki atribut yang berbeda.

Schema-less Design menawarkan fleksibilitas yang signifikan dalam pengembangan dan skalabilitas aplikasi, tetapi juga datang dengan tantangan sendiri, seperti manajemen data yang lebih kompleks dan potensi inkonsistensi. Oleh karena itu, penting untuk mempertimbangkan trade-off antara fleksibilitas dan kompleksitas saat memilih menggunakan model Schema-less untuk aplikasi Anda.

12.4.2 Denormalization

Denormalization adalah proses desain data di mana data yang sama disimpan di beberapa tempat untuk meningkatkan efisiensi query. Dalam basis data relasional, normalisasi—yaitu, pengurangan duplikasi data—biasanya dianggap sebagai praktik terbaik. Namun, dalam konteks basis data NoSQL, denormalisasi sering digunakan untuk memaksimalkan kinerja dan meminimalkan latensi query. Dengan menyimpan data yang sering diakses bersama dalam struktur yang sama, Anda bisa mengurangi jumlah operasi baca yang diperlukan untuk mengambil informasi yang relevan.

Denormalisasi sangat umum dalam basis data NoSQL seperti MongoDB, Cassandra, dan Firebase Realtime Database. Misalnya, dalam aplikasi e-commerce, informasi produk dan ulasan bisa disimpan dalam dokumen atau entitas yang sama, meskipun

mereka bisa juga disimpan sebagai entitas terpisah dalam model relasional. Dengan melakukan ini, satu query saja sudah cukup untuk mengambil semua informasi tentang sebuah produk beserta ulasannya.

Sebagai contoh, dalam MongoDB, Anda bisa menyimpan informasi produk dan ulasan dalam satu dokumen di koleksi products:

```
db.products.insert({
  name: "Laptop",
  price: 1000,
  reviews: [
    {user: "Alice", comment: "Excellent!", rating: 5},
    {user: "Bob", comment: "Pretty good", rating: 4}
  ]
});
```

Dengan struktur ini, satu query ke koleksi products akan mengembalikan detail produk beserta semua ulasannya, sehingga mengurangi kebutuhan untuk query tambahan.

Meskipun denormalisasi bisa meningkatkan kinerja, ia juga memiliki kekurangannya, termasuk potensi inkonsistensi data dan penggunaan disk yang lebih besar. Oleh karena itu, penting untuk mempertimbangkan trade-off antara kinerja dan kompleksitas manajemen data. Denormalisasi biasanya paling efektif ketika digunakan untuk data yang jarang berubah tetapi sering diakses.

12.4.3 Data Modeling Techniques

Teknik pemodelan data dalam basis data NoSQL berbeda secara signifikan dari pendekatan tradisional yang digunakan dalam basis data relasional. Karena NoSQL tidak memerlukan skema yang ketat, desain data bisa lebih fleksibel dan disesuaikan dengan kebutuhan aplikasi. Beberapa teknik umum termasuk

denormalisasi, penggunaan tipe data terstruktur seperti JSON atau BSON, dan pengoptimalan query melalui indeksasi dan partisi. Selain itu, model seperti dokumen, graf, atau kolom keluarga memungkinkan untuk representasi data yang lebih kompleks dan hubungan antar data.

Dalam basis data NoSQL, pemodelan data sering dimulai dengan memahami pola akses data. Misalnya, jika aplikasi Anda membutuhkan bacaan cepat dan query kompleks, model dokumen atau kolom keluarga mungkin lebih sesuai. Jika aplikasi Anda memerlukan analisis hubungan antar entitas, model graf bisa menjadi pilihan yang baik. Selain itu, teknik seperti sharding atau partisi bisa digunakan untuk mendistribusikan data secara efisien di seluruh kluster.

Sebagai contoh, dalam aplikasi e-commerce yang menggunakan MongoDB, Anda bisa memodelkan data produk sebagai dokumen dengan atribut seperti nama, harga, dan kategori. Anda juga bisa menambahkan indeks pada atribut yang sering digunakan dalam query untuk meningkatkan kinerja.

```
// Membuat koleksi dengan indeks pada atribut 'kategori'
db.createCollection("products");
db.products.createIndex({ "kategori": 1 });

// Menyimpan produk sebagai dokumen
db.products.insert({
  nama: "Laptop",
  harga: 1000,
  kategori: "Elektronik"
});
```

Dalam contoh ini, indeks pada kategori memungkinkan query berdasarkan kategori untuk dijalankan lebih cepat.

Teknik pemodelan data dalam NoSQL memerlukan pendekatan yang berbeda dari basis data relasional, dengan fokus pada fleksibilitas, skalabilitas, dan pola akses data. Memahami kebutuhan spesifik aplikasi Anda dan memilih teknik pemodelan yang sesuai adalah kunci untuk memaksimalkan kinerja dan efisiensi.

12.5 Implementasi NoSQL

12.5.1 Instalasi dan Konfigurasi

Instalasi dan konfigurasi adalah langkah awal penting dalam implementasi basis data NoSQL. Proses ini melibatkan pemasangan perangkat lunak basis data pada server atau kluster server, serta penyesuaian berbagai parameter untuk memenuhi kebutuhan spesifik aplikasi Anda. Parameter ini bisa mencakup pengaturan untuk skalabilitas, ketersediaan, konsistensi data, serta keamanan. Setiap jenis basis data NoSQL memiliki prosedur instalasi dan konfigurasi yang berbeda, dan biasanya disertai dengan dokumentasi yang detil untuk membantu Anda melalui proses ini.

Langkah-langkah instalasi dan konfigurasi akan sangat bergantung pada jenis basis data NoSQL yang Anda pilih. Misalnya, untuk MongoDB, Anda bisa menggunakan manajer paket seperti apt di Ubuntu untuk menginstal perangkat lunak:

```
sudo apt update
sudo apt install -y mongodb
```

Setelah instalasi, Anda bisa memulai layanan MongoDB dan memverifikasi statusnya:

```
sudo systemctl start mongodb
sudo systemctl status mongodb
```

Contoh lainnya, dalam Apache Cassandra, Anda mungkin perlu mengedit file konfigurasi cassandra.yaml untuk menyesuaikan pengaturan seperti alamat IP kluster, jumlah replika, dan strategi partisi. Misalnya, untuk mengubah alamat IP kluster, Anda bisa mengubah baris berikut dalam cassandra.yaml:

```
listen_address: <Your_Cluster_IP>
```

Setelah melakukan perubahan, Anda perlu merestart layanan Cassandra:

```
sudo systemctl restart cassandra
```

Memahami proses instalasi dan konfigurasi adalah esensial untuk implementasi basis data NoSQL yang sukses. Ini tidak hanya melibatkan pemasangan perangkat lunak tetapi juga penyesuaian berbagai parameter untuk memastikan bahwa basis data berfungsi sesuai dengan kebutuhan aplikasi Anda. Selalu rujuk ke dokumentasi resmi dan komunitas pengembang untuk mendapatkan panduan dan best practices.

12.5.2 Contoh Implementasi dengan MongoDB, Cassandra, dan Neo4j

1. MongoDB: Implementasi Aplikasi E-commerce
   MongoDB, dengan skema yang fleksibel dan performa baca/tulis yang cepat, adalah pilihan yang sangat baik untuk aplikasi e-commerce. Dalam aplikasi semacam ini, Anda akan menemukan berbagai jenis data seperti produk, pelanggan, pesanan, ulasan, dan lain-lain. MongoDB memungkinkan Anda untuk menyimpan semua jenis data ini dalam format dokumen JSON, yang bisa dengan mudah diubah strukturnya sesuai kebutuhan.

2. Membuat Database dan Koleksi: Pertama, buat database dan koleksi untuk produk dan pelanggan.

```
use ecommerceDB
db.createCollection("products")
db.createCollection("customers")
```

3. Menyimpan Produk: Simpan informasi produk dalam koleksi products.

```
db.products.insert({
  name: "Laptop",
  price: 1000,
  stock: 50,
  categories: ["Electronics", "Computers"]
});
```

4. Menyimpan Pelanggan: Simpan informasi pelanggan dalam koleksi customers.

```
db.customers.insert({
  name: "Alice",
  email: "alice@email.com",
  orders: []
});
```

5. Query: Anda bisa melakukan query untuk menemukan produk berdasarkan kategori atau harga.

```
db.products.find({ categories: "Electronics", price: { $lt: 1500 } })
```

6. Cassandra: Implementasi Sistem Monitoring Real-time
   Cassandra dirancang untuk menangani beban data yang besar dan distribusi data yang luas, membuatnya ideal untuk sistem monitoring real-time. Dalam kasus ini, Anda mungkin memiliki data dari berbagai sensor atau server yang mengirimkan data secara terus-menerus. Cassandra memungkinkan Anda untuk menulis data ini dengan latensi yang sangat rendah.

a. Membuat Keyspace dan Tabel: Pertama, buat keyspace dan tabel untuk menyimpan metrik.

```
CREATE KEYSPACE Monitoring WITH replication = {'class': 'SimpleStrategy', 'replication_factor': 1};
USE Monitoring;
CREATE TABLE metrics (
  server_id UUID,
  timestamp TIMESTAMP,
  cpu_load DOUBLE,
  PRIMARY KEY (server_id, timestamp)
);
```

b. Menyimpan Metrik: Simpan data metrik ke dalam tabel.

```
INSERT INTO metrics (server_id, timestamp, cpu_load) VALUES (uuid(), '2023-09-04 12:00', 0.5);
```

c. Query: Anda bisa melakukan query untuk menemukan metrik berdasarkan server atau rentang waktu.

```
SELECT * FROM metrics WHERE server_id = ? AND timestamp >= ? AND timestamp <= ?;
```

7. Neo4j: Implementasi Sistem Rekomendasi

Neo4j, sebagai database graf, sangat cocok untuk memodelkan hubungan antar entitas, seperti dalam sistem rekomendasi. Dalam kasus ini, Anda bisa memodelkan produk sebagai node dan hubungan antara pelanggan dan produk sebagai tepi dalam graf.

a. Membuat Node: Pertama, buat node untuk produk dan pelanggan.

```
CREATE (p:Product {name: 'Book'})
CREATE (c:Customer {name: 'Alice'})
```

b. Menambahkan Hubungan: Tambahkan hubungan "PURCHASED" antara pelanggan dan produk.

```
MATCH (p:Product {name: 'Book'}), (c:Customer {name: 'Alice'})
CREATE (c)-[:PURCHASED]->(p)
```

c. Query Rekomendasi: Anda bisa menggunakan query graf untuk menemukan rekomendasi produk berdasarkan pembelian sebelumnya.

```
MATCH (c:Customer {name: 'Alice'})-[:PURCHASED]->(:Product)<-[:PURCHASED]-(other:Customer)-[:PURCHASED]->(rec:Product)
RETURN rec.name AS Recommended_Product, COUNT(*) AS Common_Buyers
ORDER BY Common_Buyers DESC
```

Dengan contoh-contoh implementasi ini, Anda akan mendapatkan gambaran yang komprehensif tentang bagaimana MongoDB, Cassandra, dan Neo4j bisa digunakan dalam berbagai jenis aplikasi, dari e-commerce hingga monitoring real-time dan sistem rekomendasi.

12.6 Manajemen Transaksi di NoSQL

12.6.1 ACID vs BASE

ACID (*Atomicity, Consistency, Isolation, Durability*) adalah set prinsip yang digunakan untuk menjamin keandalan transaksi dalam sistem basis data relasional. "Atomicity" memastikan bahwa semua operasi dalam satu transaksi berhasil atau gagal sebagai satu unit. "*Consistency*" menjamin bahwa sistem selalu berada dalam keadaan yang konsisten. "Isolation" memastikan bahwa transaksi paralel tidak saling mengganggu. "Durability" menjamin bahwa setelah transaksi selesai, perubahan yang telah dibuat akan tetap ada.

Dalam basis data relasional seperti MySQL, Anda bisa menggunakan perintah BEGIN, COMMIT, dan ROLLBACK untuk mengelola transaksi.

```
BEGIN;
UPDATE accounts SET balance = balance - 100 WHERE id = 1;
UPDATE accounts SET balance = balance + 100 WHERE id = 2;
COMMIT;
```

Jika salah satu UPDATE gagal, Anda bisa menggunakan ROLLBACK untuk membatalkan semua perubahan.

BASE (*Basically Available, Soft State, Eventually Consistent*) adalah model yang lebih fleksibel dibandingkan ACID dan biasanya digunakan dalam basis data NoSQL. "Basically Available" menunjukkan bahwa sistem tetap berfungsi meskipun sebagian dari komponennya gagal. "Soft State" berarti keadaan sistem bisa berubah seiring waktu, bahkan tanpa input. "Eventually Consistent" berarti sistem akan menjadi konsisten dalam waktu tertentu, tetapi tidak menjamin konsistensi seketika.

Dalam basis data seperti Cassandra, Anda bisa menentukan tingkat konsistensi saat membaca atau menulis data. Misalnya, Anda bisa memilih untuk menulis ke satu node saja dan menerima bacaan yang mungkin sudah usang, untuk meningkatkan ketersediaan dan kecepatan.

```
CONSISTENCY ONE;
INSERT INTO table (id, value) VALUES (1, 'one');
SELECT * FROM table WHERE id = 1;
```

Dalam contoh ini, CONSISTENCY ONE berarti operasi akan berhasil asalkan satu node memberikan respons, meskipun data mungkin belum terdistribusi ke semua node.

Dengan memahami perbedaan antara ACID dan BASE, Anda bisa lebih memahami bagaimana manajemen transaksi bekerja di berbagai jenis basis data dan memilih model yang paling sesuai dengan kebutuhan aplikasi Anda. ACID lebih cocok untuk aplikasi yang memerlukan konsistensi dan keandalan tinggi, sementara

BASE lebih sesuai untuk aplikasi yang memerlukan skalabilitas dan ketersediaan tinggi.

12.6.2 Transaksi dalam Praktik

Transaksi dalam basis data NoSQL adalah sebuah topik yang kompleks dan beragam, terutama karena berbagai jenis basis data NoSQL (seperti dokumen, kolom, graf, dan kunci-nilai) memiliki pendekatan yang berbeda untuk menangani transaksi. Sementara beberapa basis data NoSQL modern telah mulai menawarkan fitur transaksi yang mirip dengan model ACID, banyak yang lebih fokus pada model transaksi yang lebih fleksibel dan terdistribusi, sesuai dengan prinsip BASE (*Basically Available, Soft State, Eventually Consistent*).

Transaksi dalam basis data NoSQL sering kali berbeda dari model ACID yang kita kenal dalam basis data relasional. Meskipun beberapa basis data NoSQL, seperti MongoDB, telah mulai menawarkan dukungan untuk transaksi ACID, banyak yang lebih fokus pada model transaksi yang lebih fleksibel dan terdistribusi. Dalam praktiknya, transaksi di NoSQL sering kali melibatkan operasi yang lebih sederhana, yang bisa dengan mudah dibatalkan atau digulirkan kembali secara manual jika diperlukan.

a. MongoDB: Transaksi Multi-Dokumen
   MongoDB, mulai dari versi 4.0, mendukung transaksi multi-dokumen, memungkinkan operasi CRUD (*Create, Read, Update, Delete*) pada beberapa dokumen dalam satu sesi transaksi. Ini memberikan tingkat konsistensi dan isolasi yang lebih tinggi dibandingkan operasi standar.
   Dalam Python, Anda bisa menggunakan pymongo untuk memulai sesi transaksi, melakukan beberapa operasi, dan kemudian melakukan commit atau abort sesuai kebutuhan.

```
from pymongo import MongoClient

client = MongoClient('mongodb://localhost:27017/')
```

```
db = client['ecommerce']
with client.start_session() as session:
  with session.start_transaction():
    db['products'].update_one({'sku': '12345'}, {'$inc': {'stock': -1}}, session=session)
    db['orders'].insert_one({'sku': '12345', 'quantity': 1}, session=session)
```

b. Cassandra: Batch Operations
Meskipun Cassandra tidak mendukung transaksi multi-tabel, ia menawarkan "batch operations" yang memungkinkan Anda untuk mengelompokkan beberapa operasi CQL dalam satu aksi. Ini berguna untuk memastikan konsistensi pada tingkat aplikasi, meskipun tidak menjamin konsistensi ACID.
Berikut adalah contoh menggunakan batch operations di Cassandra:

```
BEGIN BATCH
INSERT INTO users (id, name) VALUES (1, 'Alice');
INSERT INTO orders (user_id, order_id, product) VALUES (1, 101, 'Laptop');
APPLY BATCH;
```

c. Redis: Transaksi dengan MULTI dan EXEC
Redis menawarkan mekanisme transaksi sederhana melalui perintah MULTI dan EXEC. Semua perintah antara MULTI dan EXEC akan dieksekusi secara atomik, memastikan bahwa semua operasi berhasil atau tidak ada yang berhasil.
Berikut adalah contoh transaksi di Redis:

```
MULTI
INCR counter
DECR counter
EXEC
```

Dengan memahami berbagai pendekatan untuk menangani transaksi dalam basis data NoSQL, Anda akan lebih siap untuk memilih model yang paling sesuai dengan kebutuhan aplikasi Anda. Selain itu, Anda juga akan lebih memahami bagaimana mendesain aplikasi Anda agar lebih tahan terhadap kegagalan dan lebih efisien dalam pengelolaan data.

12.7 Pentingnya Skalabilitas dan Ketersediaan Tinggi

12.7.1 Horizontal Scaling vs Vertical Scaling

Skalabilitas adalah kemampuan sebuah sistem untuk menangani beban kerja yang meningkat tanpa mengorbankan performa dan ketersediaan. Dalam konteks basis data, ada dua pendekatan utama untuk skalabilitas: Horizontal Scaling dan Vertical Scaling. Kedua pendekatan ini memiliki kelebihan dan kekurangan masing-masing, dan pilihan antara keduanya seringkali bergantung pada kebutuhan spesifik aplikasi dan infrastruktur yang digunakan.

Horizontal Scaling, juga dikenal sebagai "scale-out," melibatkan penambahan lebih banyak mesin ke dalam kluster basis data yang ada. Ini memungkinkan basis data untuk menyebar beban di antara banyak server, sehingga meningkatkan kapasitas dan ketersediaan. Basis data NoSQL seperti Cassandra dan MongoDB sering dirancang dengan horizontal scaling dalam pikiran.

Dalam Cassandra, Anda bisa menambahkan lebih banyak node ke dalam kluster tanpa downtime. Setelah node baru ditambahkan, data akan otomatis di-rebalance nodetool join.
Dalam MongoDB, Anda bisa menggunakan sharding untuk mendistribusikan data di antara beberapa server.
sh.addShard("newShardHostname:27017")

Vertical Scaling, atau "*scale-up*," melibatkan peningkatan kapasitas mesin yang ada, misalnya dengan menambahkan lebih

banyak CPU, RAM, atau ruang penyimpanan. Meskipun ini adalah cara yang lebih sederhana untuk meningkatkan kapasitas, ia memiliki batasan fisik dan seringkali memerlukan downtime untuk pemeliharaan.

Dalam basis data relasional seperti MySQL, vertical scaling biasanya melibatkan peningkatan spesifikasi hardware dari server yang ada. Ini bisa melibatkan peningkatan CPU, memori, atau penyimpanan, dan biasanya memerlukan waktu downtime untuk implementasinya.

Pilihan antara horizontal scaling dan vertical scaling akan sangat bergantung pada jenis beban kerja, kebutuhan kapasitas, dan toleransi terhadap downtime. Horizontal scaling biasanya lebih fleksibel dan bisa memberikan ketersediaan yang lebih tinggi, tetapi mungkin memerlukan perubahan arsitektur yang lebih kompleks. Sementara itu, vertical scaling biasanya lebih mudah diimplementasikan tetapi memiliki batasan fisik dan potensi downtime. Memahami perbedaan antara keduanya adalah kunci untuk memilih strategi skalabilitas yang paling sesuai untuk aplikasi Anda.

12.7.2 Replication dan Sharding

Dalam dunia basis data, Replication dan Sharding adalah dua teknik utama yang digunakan untuk mencapai skalabilitas dan ketersediaan tinggi. Meskipun keduanya memiliki tujuan yang sama, pendekatan dan implementasinya berbeda. Replication berfokus pada peningkatan ketersediaan dan ketahanan data, sementara Sharding berfokus pada peningkatan performa dan efisiensi penyimpanan.

Replication adalah proses menyalin data dari satu database ke database lain untuk menciptakan redundansi. Ini meningkatkan ketersediaan dan memungkinkan sistem untuk tetap berfungsi meskipun ada kegagalan pada beberapa node. Replication bisa

dilakukan dalam berbagai mode, seperti master-slave atau peer-to-peer, tergantung pada jenis basis data NoSQL yang digunakan.

Dalam MongoDB, Anda bisa mengatur Replica Set yang terdiri dari beberapa node. Satu node akan bertindak sebagai primary, sementara yang lainnya sebagai secondary.

rs.initiate()

Dalam Cassandra, replication diatur melalui Replication Factor, yang menentukan berapa banyak salinan data yang harus disimpan.

CREATE KEYSPACE myKeyspace WITH replication {'class': 'SimpleStrategy', 'replication_factor': 3};

Sharding adalah teknik membagi data ke dalam beberapa partisi yang disebut shard. Setiap shard bisa ditempatkan pada server yang berbeda, memungkinkan query untuk diproses oleh beberapa server secara paralel. Ini meningkatkan performa dan memungkinkan basis data untuk menangani lebih banyak data daripada yang bisa ditampung oleh satu mesin.

Replication dan Sharding adalah dua strategi yang sangat efektif untuk meningkatkan skalabilitas dan ketersediaan dalam basis data NoSQL. Replication memberikan keuntungan dalam hal ketersediaan dan ketahanan terhadap kegagalan, sementara Sharding menawarkan keuntungan dalam performa dan efisiensi penyimpanan. Memahami keduanya akan memungkinkan Anda untuk merancang dan mengimplementasikan solusi basis data yang robust dan skalabel.

12.7.3 Partitioning Strategies

Partitioning adalah salah satu aspek kunci dalam desain dan implementasi basis data NoSQL yang skalabel dan berkinerja tinggi. Teknik ini memungkinkan data untuk dibagi dan dikelola di beberapa node atau server, sehingga memaksimalkan efisiensi dan performa. Ada beberapa strategi partitioning yang umum digunakan, termasuk Range Partitioning, Hash Partitioning, dan

Directory-based Partitioning. Pilihan strategi akan sangat bergantung pada jenis data, pola akses, dan kebutuhan aplikasi.

Dalam Range Partitioning, data dibagi berdasarkan rentang nilai dari satu atau lebih kolom. Misalnya, semua data dengan ID antara 1 dan 1000 mungkin disimpan pada satu partisi, sementara data dengan ID antara 1001 dan 2000 disimpan pada partisi lain. Strategi ini sering digunakan dalam basis data yang memiliki kunci yang terdistribusi secara teratur. Dalam basis data relasional, Anda bisa menggunakan PARTITION BY RANGE untuk membuat tabel yang memanfaatkan *Range Partitioning*.

Hash Partitioning menggunakan fungsi hash pada kunci partisi untuk menentukan di mana sebuah rekaman harus disimpan. Ini menghasilkan distribusi data yang lebih seragam di antara partisi, tetapi bisa membuat operasi berbasis rentang lebih sulit atau tidak efisien. Dalam Cassandra, Hash Partitioning digunakan secara default. Partition key di-hash untuk menentukan di mana data akan disimpan.

Directory-based Partitioning menggunakan direktori atau peta yang menyimpan informasi tentang lokasi partisi. Ini memungkinkan sistem untuk dengan cepat mengetahui di mana data untuk kunci tertentu berada, dan mempermudah proses rebalancing dan redistribusi data. Dalam basis data seperti MongoDB, Anda bisa menggunakan Config Server sebagai direktori yang mengelola informasi partisi dalam skenario sharding.

Memahami berbagai strategi partitioning adalah kunci untuk merancang dan mengimplementasikan basis data NoSQL yang skalabel dan berkinerja tinggi. Setiap strategi memiliki kelebihan dan kekurangan tersendiri, dan pilihan yang tepat akan sangat bergantung pada kebutuhan spesifik aplikasi Anda. Dengan memilih strategi partitioning yang paling sesuai, Anda bisa memaksimalkan efisiensi, performa, dan ketersediaan sistem basis data Anda.

12.8 Keamanan dalam NoSQL

12.8.1 Authentication dan Authorization

Keamanan data adalah salah satu aspek kritis dalam manajemen basis data, dan ini juga berlaku untuk basis data NoSQL. Dua komponen utama dari keamanan basis data adalah *Authentication* (otentikasi) dan *Authorization* (otorisasi). *Authentication* adalah proses verifikasi identitas pengguna atau sistem yang mencoba mengakses basis data, sementara *Authorization* adalah proses yang menentukan apa yang bisa dilakukan pengguna setelah berhasil masuk.

Authentication dalam basis data NoSQL biasanya melibatkan mekanisme seperti username dan password, sertifikat SSL, atau bahkan autentikasi berbasis token. Beberapa basis data NoSQL juga mendukung autentikasi multi-faktor untuk meningkatkan keamanan.

Dalam MongoDB, Anda bisa mengimplementasikan autentikasi menggunakan metode SCRAM (*Salted Challenge Response Authentication Mechanism*) yang lebih aman, atau bahkan menggunakan x.509 untuk autentikasi berbasis sertifikat.

```
# Mengaktifkan autentikasi SCRAM di MongoDB
mongod --auth

# Mengaktifkan autentikasi x.509
mongod --sslMode requireSSL --sslPEMKeyFile /etc/ssl/mongodb.pem
```

Setelah autentikasi berhasil, langkah selanjutnya adalah menentukan apa yang bisa dilakukan oleh pengguna atau sistem tersebut. Ini adalah bagian dari Authorization. Dalam basis data NoSQL, otorisasi biasanya dikelola melalui *Role-Based Access Control* (RBAC), di mana setiap peran memiliki serangkaian izin yang telah ditentukan.

Dalam Cassandra, Anda bisa menggunakan CQL (*Cassandra Query Language*) untuk mendefinisikan peran dan izin dengan sangat granular. Misalnya, Anda bisa memberikan izin baca saja pada sebuah tabel kepada sebuah peran.

```
CREATE ROLE readonly WITH PASSWORD = 'password' AND LOGIN = true;
GRANT SELECT ON TABLE mytable TO readonly;
```

Dalam MongoDB, Anda bisa mendefinisikan peran yang sangat spesifik yang hanya memungkinkan operasi tertentu.

```
db.createRole({
  role: "readWriteOnSpecificFields",
  privileges: [
    {
      resource: { db: "mydb", collection: "mycollection" },
      actions: [ "find", "update" ]
    }
  ],
  roles: []
})
```

Keamanan adalah aspek yang sangat penting dalam manajemen basis data, dan ini termasuk basis data NoSQL. Melalui mekanisme *Authentication d*an *Authorization* yang tepat, Anda bisa memastikan bahwa data Anda aman dari akses yang tidak sah dan manipulasi. Dengan memahami dan menerapkan metode autentikasi dan otorisasi yang lebih canggih, Anda bisa membangun lapisan keamanan yang lebih kuat untuk basis data Anda.

12.8.2 Encryption

Enkripsi adalah salah satu metode keamanan yang paling efektif untuk melindungi data sensitif. Dalam konteks basis data NoSQL, enkripsi bisa diterapkan pada berbagai tingkatan—mulai dari enkripsi data saat transit (data yang sedang dikirim antar sistem) hingga enkripsi data at rest (data yang disimpan di disk). Enkripsi membantu memastikan bahwa meskipun data jatuh ke tangan yang salah, informasi tersebut tetap tidak bisa diakses atau dibaca tanpa kunci enkripsi yang sesuai.

a. Enkripsi Data in Transit

Enkripsi data in transit melindungi data saat sedang dikirim antara klien dan server atau antar node dalam kluster basis data. Protokol keamanan seperti TLS (*Transport Layer Security*) atau SSL (*Secure Sockets Layer*) biasanya digunakan untuk ini.

Dalam MongoDB, Anda bisa mengaktifkan enkripsi TLS dengan opsi --tlsMode dan menyediakan sertifikat yang valid.

```
mongod --tlsMode requireTLS --tlsCertificateKeyFile /etc/ssl/mongodb.pem
```

Dalam Cassandra, Anda bisa mengaktifkan SSL untuk komunikasi antar node dan antara klien dengan node melalui konfigurasi di cassandra.yaml.

```
# Aktifkan SSL untuk komunikasi antar node
internode_encryption: all
```

b. Enkripsi Data at Rest

Enkripsi data at rest melindungi data yang disimpan di media penyimpanan. Ini penting untuk melindungi data dari akses fisik yang tidak sah. Beberapa basis data NoSQL menawarkan enkripsi at rest sebagai fitur bawaan, sementara yang lain membutuhkan solusi pihak ketiga.

Dalam MongoDB Enterprise, Anda bisa menggunakan WiredTiger storage engine untuk enkripsi data at rest.

```
mongod --enableEncryption --encryptionKeyFile /path/to/keyfile
```

Dalam Cassandra, Anda bisa menggunakan fitur *Transparent Data Encryption* (TDE) untuk enkripsi data at rest.

```
# Aktifkan TDE di cassandra.yaml
transparent_data_encryption_options:
  enabled: true
```

Enkripsi adalah salah satu pilar keamanan dalam manajemen basis data NoSQL. Dengan menerapkan enkripsi baik pada data in transit maupun data at rest, Anda bisa menambah lapisan keamanan yang signifikan untuk melindungi data sensitif. Memahami dan menerapkan enkripsi dengan benar adalah kunci untuk memitigasi risiko keamanan yang terkait dengan akses data yang tidak sah.

12.8.3 Best Practices dalam mengamankan basis data NoSQL

Mengamankan basis data NoSQL adalah tugas yang kompleks yang memerlukan perhatian terhadap berbagai aspek, mulai dari autentikasi pengguna hingga enkripsi data. Berikut adalah beberapa best practices yang direkomendasikan:

1. Aktifkan Autentikasi dan Otorisasi
   Tujuan: Memastikan hanya pengguna atau sistem yang terotentikasi yang dapat mengakses basis data.
   Cara Implementasi: Gunakan fitur autentikasi dan otorisasi bawaan dari basis data NoSQL yang Anda gunakan. Misalnya, dalam MongoDB, Anda bisa mengaktifkan mode autentikasi dengan opsi --auth.
2. Enkripsi Data
   Tujuan: Melindungi data sensitif dari akses yang tidak sah.

Cara Implementasi: Gunakan enkripsi data in transit dengan protokol seperti TLS. Untuk data at rest, gunakan fitur enkripsi bawaan atau solusi pihak ketiga.

3. Pemantauan dan Audit
   Tujuan: Mendeteksi dan merespons aktivitas yang mencurigakan secepat mungkin.
   Cara Implementasi: Gunakan alat pemantauan dan audit, seperti ELK Stack atau fitur audit bawaan dari basis data NoSQL Anda.
4. Isolasi Jaringan dan Firewall
   Tujuan: Membatasi akses ke basis data hanya dari IP atau jaringan yang telah disetujui.
   Cara Implementasi: Gunakan firewall atau fitur keamanan jaringan lainnya untuk membatasi akses ke port basis data Anda.
5. Backup dan Pemulihan
   Tujuan: Meminimalkan dampak dari kehilangan data atau kegagalan sistem.
   Cara Implementasi: Buat backup teratur dari data Anda dan pastikan Anda memiliki prosedur pemulihan yang efektif.
6. Patching dan Pembaruan
   Tujuan: Menjaga basis data Anda tetap up-to-date dengan patch keamanan terbaru.
   Cara Implementasi: Ikuti siklus hidup pembaruan dan patching yang direkomendasikan oleh penyedia basis data NoSQL Anda.
7. Prinsip Hak Akses Minimal
   Tujuan: Memberikan hak akses hanya yang diperlukan untuk melakukan tugas tertentu.
   Cara Implementasi: Gunakan Role-Based Access Control (RBAC) untuk membatasi hak akses sesuai kebutuhan.
8. Logging dan Monitoring

Tujuan: Mencatat dan memantau semua aktivitas yang terjadi di basis data.
Cara Implementasi: Aktifkan fitur logging bawaan dan gunakan alat pemantauan untuk menganalisis log.

Keamanan adalah aspek yang sangat penting dalam manajemen basis data NoSQL. Dengan mengadopsi best practices ini, Anda tidak hanya akan meminimalkan risiko tetapi juga akan mempersiapkan infrastruktur Anda terhadap ancaman yang mungkin belum diketahui. Selalu tetap up-to-date dengan perkembangan terbaru dalam keamanan informasi untuk memastikan keamanan basis data Anda.

12.9 Kesimpulan

Basis data NoSQL menawarkan kelebihan signifikan dalam hal skalabilitas, fleksibilitas, dan performa, terutama untuk aplikasi yang memerlukan penanganan data dalam volume besar, kecepatan tinggi, atau struktur yang kompleks. Namun, NoSQL juga memiliki kekurangannya, termasuk kurangnya standar konsistensi dan transaksi yang kuat, serta potensi kompleksitas dalam manajemen dan keamanan data. Oleh karena itu, keputusan untuk menggunakan NoSQL harus didasarkan pada kebutuhan spesifik proyek atau aplikasi Anda. Jika Anda berhadapan dengan Big Data, membutuhkan skalabilitas horizontal, atau memerlukan struktur data yang lebih fleksibel seperti JSON atau XML, NoSQL bisa menjadi pilihan yang sangat tepat. Sebaliknya, untuk aplikasi yang memerlukan transaksi kompleks dan relasi antar tabel yang kuat, basis data relasional mungkin lebih sesuai.

Dalam konteks keamanan, NoSQL memerlukan pendekatan yang berbeda dibandingkan dengan basis data relasional. Meskipun banyak basis data NoSQL modern telah memasukkan fitur keamanan canggih, tetap penting untuk memahami dan

menerapkan best practices keamanan, termasuk autentikasi, otorisasi, enkripsi, dan pemantauan, untuk melindungi data Anda.

Secara keseluruhan, NoSQL adalah teknologi yang matang dan berkembang, yang menawarkan solusi efisien untuk tantangan data modern. Namun, seperti semua teknologi, ia memiliki kelebihan dan kekurangan yang perlu dipertimbangkan secara cermat. Memilih untuk menggunakan NoSQL berarti memahami dan menerima trade-off ini, dan mempersiapkan diri untuk menavigasi kompleksitas yang mungkin muncul.

DAFTAR PUSTAKA

Cattell, R. 2011. 'Scalable SQL and NoSQL data stores', *ACM SIGMOD Record*, 39(4), pp. 12–27. Available at: https://doi.org/10.1145/1978915.1978919.

Chang, F. *et al.* 2008. 'Bigtable: A Distributed Storage System for Structured Data', *ACM Transactions on Computer Systems*, 26(2), pp. 1–26. Available at: https://doi.org/10.1145/1365815.1365816.

DeCandia, G. *et al.* 2007. 'Dynamo: Amazon's highly available key-value store', *ACM SIGOPS Operating Systems Review*, 41(6), pp. 205–220. Available at: https://doi.org/10.1145/1323293.1294281.

Leavitt, N. 2010. 'Will NoSQL Databases Live Up to Their Promise?', *Computer*, 43(2), pp. 12–14. Available at: https://doi.org/10.1109/MC.2010.58.

Robinson, I., Webber, J. and Eifrem, E. 2015. *Graph Databases*. O'Reilly Media, Inc.

Stonebraker, M. *et al.* 2010. 'MapReduce and parallel DBMSs', *Communications of the ACM*, 53(1), pp. 64–71. Available at: https://doi.org/10.1145/1629175.1629197.

BAB 13
MIGRASI DAN KONVERSI BASIS DATA

Oleh Rahmat Taufik R.L Bau

13.1 Pendahuluan

Di era teknologi yang terus berkembang dengan cepat seperti saat ini, basis data memainkan peran yang krusial dalam mendukung operasional dan pertumbuhan bisnis. Basis data merupakan fondasi dari sistem informasi yang mengandung informasi berharga dan kritis bagi organisasi. Namun, seiring berjalannya waktu, kebutuhan akan perubahan dan adaptasi teknologi semakin mendesak. Maka dari itu, migrasi dan konversi basis data menjadi suatu keharusan bagi organisasi untuk terus berinovasi dan bertransformasi guna tetap kompetitif dan relevan.

Gambar 13.1. Ilustrasi migrasi data dari penyimpanan lokal ke database online.
(Sumber: Petanicode)

Urgensi migrasi dan konversi data terletak pada kemampuan untuk menghadirkan basis data yang lebih modern, efisien, dan responsif terhadap perubahan pasar. Dengan teknologi yang terus berkembang, kebutuhan untuk menggunakan platform dan sistem yang lebih canggih menjadi tidak terelakkan. Migrasi dan konversi basis data merupakan solusi yang tepat untuk memperbarui dan meningkatkan kinerja sistem, serta mengintegrasikannya dengan teknologi terkini.

Selain itu, migrasi dan konversi basis data juga berperan penting dalam menghadapi tantangan keamanan dan privasi data. Dalam lingkungan yang semakin terhubung dan rentan terhadap ancaman siber, organisasi perlu memastikan data mereka tetap aman dan terlindungi. Basis data yang sudah lama dapat memiliki kerentanannya sendiri, dan dengan migrasi ke platform yang lebih aman, risiko kebocoran atau pencurian data dapat diminimalisir.

Tidak hanya itu, adaptasi teknologi juga berperan dalam meningkatkan efisiensi operasional dan produktivitas. Basis data yang efisien dapat mengoptimalkan proses bisnis, mengurangi waktu respon, dan memberikan informasi yang lebih tepat waktu. Dengan migrasi dan konversi basis data, organisasi dapat mengalami peningkatan produktivitas dan keunggulan kompetitif.

13.2 Pengenalan Migrasi dan Konversi Data

Dalam dunia teknologi informasi, basis data menjadi inti dari sistem informasi yang digunakan oleh organisasi atau perusahaan untuk menyimpan, mengelola, dan mengakses data. Namun, kebutuhan bisnis yang terus berkembang seringkali memerlukan perubahan dalam struktur atau format data. Dalam hal ini, migrasi dan konversi basis data menjadi dua metode

penting yang sering digunakan untuk menghadapi tantangan tersebut.

13.2.1 Definisi Migrasi Basis Data

Migrasi basis data adalah proses yang kompleks yang melibatkan perpindahan seluruh sistem basis data dari satu lingkungan atau platform ke lingkungan atau platform yang lain. Lingkungan basis data mencakup perangkat keras, perangkat lunak, dan infrastruktur yang mendukung penyimpanan, pengelolaan, dan akses data. Proses migrasi ini berfokus pada mengubah seluruh sistem basis data untuk mencapai tujuan tertentu, seperti meningkatkan performa, mengganti teknologi yang sudah usang, atau mengkonsolidasikan beberapa sistem basis data menjadi satu.

13.2.2 Tujuan dan Manfaat Migrasi Basis Data

Tujuan utama dari migrasi basis data adalah untuk meningkatkan efisiensi dan keandalan sistem basis data. Melalui migrasi, organisasi dapat memperbarui infrastruktur dan teknologi basis data, sehingga meningkatkan performa dan kinerja sistem secara keseluruhan. Selain itu, migrasi juga berfungsi untuk meningkatkan keamanan data dengan adopsi teknologi basis data yang lebih baru dan dilengkapi dengan fitur keamanan yang lebih baik.

Proses migrasi juga memungkinkan sistem basis data untuk menjadi lebih skalabel, sehingga dapat menangani volume data yang semakin bertambah seiring dengan pertumbuhan organisasi. Dengan migrasi, pemeliharaan sistem juga menjadi lebih mudah, karena basis data yang lebih baru seringkali memiliki fitur pemeliharaan yang lebih baik, sehingga memudahkan tim IT dalam mengelola dan memantau kesehatan sistem secara berkala.

13.2.3 Definisi Konversi Basis Data

Konversi basis data adalah proses yang berkaitan dengan perubahan format atau struktur data tanpa harus mengganti seluruh sistem basis data. Fokus utama dari konversi basis data adalah untuk mengubah data ke dalam format yang lebih sesuai dengan kebutuhan bisnis atau teknis, tanpa mengubah lingkungan atau platform yang digunakan.

13.2.4 Tujuan dan Manfaat Konversi Basis Data

Tujuan utama dari konversi basis data adalah untuk meningkatkan efisiensi penggunaan data dan memenuhi kebutuhan bisnis yang berubah. Melalui proses konversi, organisasi dapat mengatasi masalah ketidaksesuaian data dengan perangkat lunak baru yang digunakan. Konversi juga memungkinkan organisasi untuk memperbarui struktur data yang sudah usang atau tidak sesuai dengan kebutuhan saat ini.

Proses konversi basis data juga dapat membantu meningkatkan integritas data dengan menghapus duplikasi dan memperbaiki data yang tidak akurat. Selain itu, konversi data memungkinkan organisasi untuk menyatukan data dari berbagai sumber ke dalam satu format standar, memungkinkan analisis dan pengambilan keputusan yang lebih baik. Dengan mengoptimalkan struktur dan indeks data, konversi juga dapat membantu meningkatkan performa sistem.

13.2.5 Perbedaan antara Migrasi dan Konversi Basis Data

Meskipun migrasi dan konversi basis data sama-sama berhubungan dengan perubahan data, ada perbedaan mendasar antara keduanya. Migrasi basis data melibatkan perpindahan seluruh sistem basis data dari satu lingkungan atau platform ke lingkungan atau platform yang lain, termasuk perangkat keras dan perangkat lunaknya. Sementara itu, konversi basis data lebih fokus

pada perubahan struktur atau format data tanpa mengganti seluruh sistem.

Tabel 13.1. Perbedaan antara Migrasi dan Konversi basis data

Aspek	**Migrasi Basis Data**	**Konversi Basis Data**
Ruang Lingkup	Meliputi seluruh sistem	Terfokus pada struktur/format data saja
Kompleksitas	Lebih kompleks	Relatif lebih sederhana
Waktu Pelaksanaan	Memakan waktu yang lebih lama	Biasanya lebih cepat
Resiko	Lebih tinggi	Relatif lebih rendah

Migrasi basis data umumnya lebih kompleks karena melibatkan banyak aspek teknis dan bisnis. Sebaliknya, konversi basis data cenderung lebih sederhana karena berfokus pada perubahan struktur data yang lebih terbatas.

Migrasi basis data memakan waktu yang lebih lama dibandingkan dengan konversi. Konversi basis data biasanya lebih cepat karena hanya berfokus pada bagian-bagian tertentu dari basis data.

Resiko migrasi basis data umumnya lebih tinggi karena melibatkan perpindahan keseluruhan sistem. Sebaliknya, resiko konversi basis data relatif lebih rendah karena lebih terbatas dalam cakupan perubahannya.

13.3 Metode dan Tools

Dalam proses migrasi dan konversi basis data, pemilihan metode yang sesuai menjadi kunci utama keberhasilan. Sub bab ini akan menjelaskan dengan mendalam tentang berbagai metode migrasi dan konversi yang tersedia, serta penggunaan alat dan

teknik bantu yang dapat digunakan untuk memfasilitasi proses tersebut.

13.3.1 Metode Migrasi dan Konversi Data

Pemilihan metode migrasi dan konversi yang tepat memegang peranan kunci dalam keberhasilan proses migrasi dan konversi basis data. Proses ini harus dipertimbangkan secara matang, mengingat beberapa faktor penting. Pertama, ukuran dan kompleksitas basis data menjadi faktor penting dalam menentukan metode yang tepat. Basis data yang besar dan kompleks mungkin memerlukan pendekatan yang berbeda dibandingkan dengan basis data yang lebih kecil dan sederhana.

Gambar 13.2. Ilustrasi proses migrasi dan konversi data yang memerlukan perencanaan yang matang
(Sumber: Stellaxius)

Selain itu, tingkat ketersediaan sumber daya seperti anggaran, personel, dan infrastruktur teknis juga harus dipertimbangkan. Setiap metode mungkin memerlukan sumber daya yang berbeda, dan pemilihan metode harus disesuaikan dengan ketersediaan yang ada. Kemudian, kebutuhan downtime

selama proses migrasi atau konversi juga menjadi pertimbangan penting. Dalam banyak kasus, waktu downtime harus diminimalkan untuk menghindari dampak negatif pada operasional bisnis.

Terakhir, keselamatan data harus menjadi prioritas utama. Pemilihan metode yang tepat harus memastikan keamanan data selama proses migrasi atau konversi untuk menghindari kerugian atau kebocoran data yang dapat berdampak serius pada organisasi.

Dengan mempertimbangkan faktor-faktor di atas, berikut adalah metode migrasi dan konversi yang sesuai dengan kebutuhan dan tujuan mereka :

1. Metode Penuh (*Full Migration/Conversion*): Metode penuh melibatkan migrasi atau konversi seluruh basis data secara menyeluruh. Ini berarti semua data, struktur, dan konfigurasi dari basis data lama akan dipindahkan atau diubah ke dalam basis data baru. Metode ini sering digunakan ketika organisasi melakukan perubahan teknologi yang drastis atau beralih ke platform yang berbeda secara keseluruhan. Contoh Penerapan: Sebuah perusahaan e-commerce yang sudah lama menggunakan basis data relasional memutuskan untuk beralih ke basis data NoSQL untuk meningkatkan skalabilitas dan kinerja. Dalam hal ini, perusahaan akan melakukan migrasi penuh dari basis data relasional ke basis data NoSQL, termasuk seluruh data produk, informasi pelanggan, dan riwayat transaksi.
2. Metode Parsial (*Partial Migration/Conversion*): Metode parsial melibatkan migrasi atau konversi hanya sebagian dari basis data. Tidak seluruh data dan tabel dari basis data lama dipindahkan atau diubah ke dalam basis data baru. Metode ini umumnya digunakan ketika ada beberapa data atau tabel yang dianggap kritis atau penting dan perlu segera beralih ke basis data baru, sementara bagian lain

tetap menggunakan basis data lama. Contoh Penerapan: Sebuah universitas memutuskan untuk mengganti sistem manajemen akademiknya ke sistem yang lebih modern. Namun, untuk menghindari risiko yang tinggi dan memastikan kesinambungan operasional, universitas hanya melakukan konversi parsial dengan memindahkan data mahasiswa, jadwal kuliah, dan hasil ujian dari sistem lama ke sistem baru, sementara data kepegawaian dan keuangan tetap di sistem lama hingga periode tertentu.

3. Metode Paralel (*Parallel Migration/Conversion*): Metode paralel melibatkan menjalankan dua sistem basis data secara paralel selama proses migrasi atau konversi. Data dan transaksi akan diperbarui pada kedua sistem secara bersamaan, dan pengguna secara bertahap akan beralih ke sistem basis data baru. Setelah yakin bahwa sistem baru berfungsi dengan baik, sistem lama akan dinonaktifkan. Contoh Penerapan: Sebuah perusahaan perbankan memutuskan untuk beralih ke platform basis data yang lebih modern dan skalabel. Untuk mengurangi risiko dan memastikan integritas data selama proses migrasi, perusahaan akan menjalankan sistem basis data lama dan baru secara paralel. Pengguna akan secara bertahap diberikan akses ke sistem baru, dan proses migrasi data akan dilakukan secara terus-menerus hingga seluruh data dan transaksi berada di sistem baru. Setelah yakin bahwa sistem baru berfungsi dengan baik, sistem lama akan dinonaktifkan secara resmi.

13.3.2 Tools Dalam Proses Migrasi dan Konversi Data

Dalam proses migrasi dan konversi basis data, penggunaan alat dan teknik bantu dapat sangat membantu untuk mengurangi kompleksitas dan risiko. Berikut ini adalah pengelompokan tools berdasarkan fungsinya untuk migrasi atau konversi data:

1. Tools ETL (*Talend Open Studio, SAP Data Services, Pentaho Data Integration*): Tools dalam kategori ini fokus pada proses ekstraksi (*extraction*), transformasi (*transform*), dan pemuatan data (*loading*).
   Fungsi utama dari ETL tools adalah untuk melakukan ekstraksi, transformasi, dan pemuatan data dari berbagai sumber ke dalam basis data atau gudang data yang lebih terstruktur dan siap untuk analisis atau penggunaan lainnya. Dalam tahap ekstraksi, data diambil dari sumber asal dan disimpan sementara sebelum dilakukan transformasi. Setelah itu, dalam tahap transformasi, data diubah, dibersihkan, diatur ulang, atau dihitung ulang agar sesuai dengan kebutuhan bisnis atau tujuan analisis. Terakhir, pada tahap pemuatan, data yang sudah diolah dimasukkan ke dalam basis data tujuan atau gudang data, sehingga siap digunakan oleh pengguna atau aplikasi lainnya. Dengan menggunakan ETL tools, proses ini dapat dilakukan dengan lebih efisien dan akurat, memastikan bahwa data yang dipindahkan atau diubah tetap konsisten, tepat, dan siap digunakan dalam lingkungan basis data baru.
2. *Database-Specific Tools, seperti Microsoft* SQL *Server Migration Assistant* (SSMA) dan Oracle Data Pump, memiliki fungsi utama yang sangat relevan dalam proses migrasi dan konversi data.
   Pertama, tools ini menyediakan optimalisasi dan pemrosesan khusus untuk basis data tertentu. Dengan demikian, mereka dapat menghadirkan teknik dan algoritma khusus yang dioptimalkan sesuai dengan karakteristik basis data target. Dalam konteks ini, tools tersebut mampu meminimalkan waktu dan sumber daya yang dibutuhkan untuk migrasi atau konversi,

meningkatkan efisiensi dan kinerja keseluruhan dalam lingkungan basis data tujuan.

Selain itu, tools database-specific juga menawarkan kemudahan integrasi dengan basis data spesifik. Mereka telah dioptimalkan untuk berfungsi secara sinergis dengan fitur dan layanan lain yang sudah ada dalam lingkungan basis data tersebut. Integrasi yang mudah memudahkan koordinasi dan interaksi dengan sistem yang sudah ada, sehingga meminimalkan potensi konflik atau kesalahan selama proses migrasi atau konversi. Dengan fokus pada basis data tertentu, tools ini dapat memberikan solusi yang lebih terarah dan terpadu untuk memastikan kesuksesan migrasi atau konversi data dalam lingkungan yang kompleks.

3. *Cloud-Based Data Integration Tools* (AWS *Database Migration Service, Microsoft Azure Data Factory*):

   Tools cloud-based data integration menawarkan fungsionalitas yang penting dalam proses migrasi dan konversi data. Pertama, tools ini menawarkan skalabilitas dan performa tinggi, yang menjadi keunggulan utama layanan cloud. Dengan kapabilitas skalabilitas yang tinggi, proses migrasi data besar dan kompleks dapat dijalankan dengan efisien, tanpa mengalami kendala kapasitas yang biasa terjadi pada infrastruktur on-premises. Hal ini memungkinkan perusahaan untuk dengan mudah mengatasi pertumbuhan data yang cepat dan meningkatkan produktivitas.

   Selain itu, tools ini menawarkan kemudahan integrasi dengan lingkungan cloud. Migrasi data antara sistem on-premises dan layanan cloud menjadi lebih mudah dan lancar, mengurangi hambatan dalam mengadopsi teknologi cloud. Proses migrasi menjadi lebih cepat dan lebih aman, karena tools ini telah dioptimalkan untuk berfungsi dengan

baik dalam lingkungan cloud. Kemampuan integrasi ini memungkinkan perusahaan untuk dengan mulus memanfaatkan layanan cloud yang dapat meningkatkan efisiensi operasional dan skalabilitas infrastruktur, sambil tetap menjaga integritas dan keamanan data. Dengan demikian, tools cloud-based data integration menjadi solusi yang andal dan efektif untuk memfasilitasi proses migrasi dan konversi data menuju lingkungan cloud yang lebih modern dan adaptif.

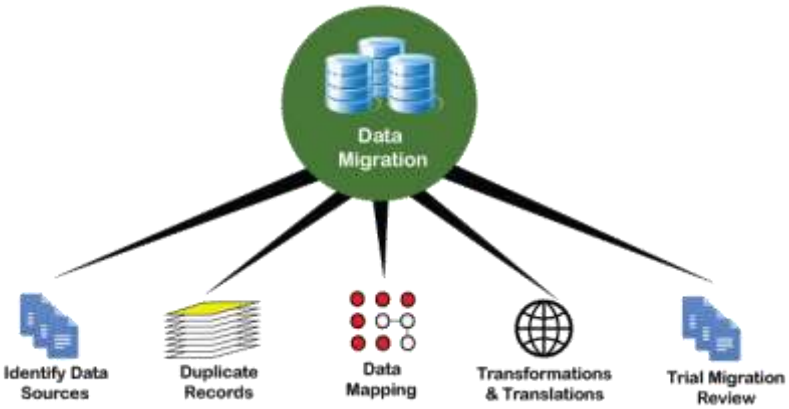
4. *Database Schema Versioning Tools* (Flyway).
   Tools database schema, seperti Flyway, memiliki peran penting dalam mengelola perubahan skema basis data dari waktu ke waktu. Fungsionalitas utama dari Flyway adalah untuk melacak dan mengatur perubahan skema secara terstruktur dan terkelola. Dalam konteks ini, tools ini berfungsi sebagai alat pengatur yang membantu memastikan konsistensi skema basis data, baik dalam lingkungan pengembangan maupun saat proses migrasi.
   Dengan Flyway, pengguna dapat dengan mudah menambahkan, mengubah, atau menghapus perubahan skema dalam database secara sistematis. Proses ini menjadi lebih terorganisir dan terdokumentasi dengan baik, sehingga memudahkan tim pengembang dalam bekerja dengan perubahan skema yang lebih kompleks. Selain itu, tools ini memastikan bahwa seluruh tim dapat bekerja secara sinergis dan selaras dalam mengelola perubahan skema, mengurangi risiko terjadinya konflik dan kesalahan yang dapat berdampak negatif pada integritas data.
   Dengan kemampuan pengaturan perubahan skema yang handal, Flyway juga memudahkan proses migrasi dan pemeliharaan basis data. Ketika basis data perlu ditingkatkan atau diubah untuk memenuhi kebutuhan

baru, tools ini memastikan bahwa perubahan skema dilakukan secara aman dan sesuai dengan rencana. Dengan demikian, pengguna dapat lebih percaya diri dalam menjalankan proses migrasi dan pemeliharaan, dengan peningkatan kualitas dan keamanan basis data secara keseluruhan.

5. GUI-*Based Database Tools* (MySQL Workbench, MongoDB Compass):
   Tools GUI-*based database* menawarkan fungsionalitas yang sangat bermanfaat bagi pengguna yang tidak terbiasa dengan perintah baris atau bahasa pemrograman dalam proses migrasi dan konversi data. Antarmuka grafis yang disediakan oleh tools ini memungkinkan pengguna untuk dengan mudah berinteraksi dengan perangkat lunak tanpa harus menguasai kode atau sintaks tertentu. Pengguna dapat dengan cepat memahami dan mengoperasikan tools tersebut tanpa menghadapi kompleksitas teknis yang seringkali menjadi hambatan dalam proses migrasi dan konversi.
   Salah satu keunggulan dari antarmuka grafis adalah kemampuannya untuk memvisualisasikan data secara intuitif. Pengguna dapat melihat struktur dan konten basis data dalam bentuk grafis atau tabel yang lebih mudah dipahami. Hal ini mempermudah pengguna dalam memahami hubungan antar tabel, skema data, atau informasi penting lainnya. Visualisasi data membantu pengguna dalam mengidentifikasi pola, kesalahan, atau anomali yang mungkin terjadi dalam basis data. Dengan tools GUI-based database, pengguna memiliki akses yang lebih mudah dan jelas ke informasi yang relevan, sehingga mempercepat proses pengambilan keputusan dan meningkatkan produktivitas dalam proses migrasi dan konversi data.

13.4 Proses Implementasi Migrasi dan Konversi Data

Proses implementasi migrasi atau konversi basis data adalah tahap kritis dalam proyek tersebut. Setelah perencanaan dan persiapan yang matang, langkah selanjutnya adalah melakukan migrasi atau konversi secara menyeluruh dengan koordinasi yang baik dan tindakan yang hati-hati.



Gambar 13.3. Proses utama dalam mitigasi data (Sumber: Javatpoint)

Pertama, langkah awal dalam proses ini adalah menghentikan operasi basis data lama. Jika menggunakan metode paralel, proses ini melibatkan penghentian semua operasi basis data lama, termasuk transaksi dan operasi lainnya, untuk memastikan tidak ada data yang hilang atau proses yang tertunda saat migrasi atau konversi berlangsung.

Setelah operasi basis data lama dihentikan, tahap selanjutnya adalah melakukan migrasi atau konversi data. Data akan dipindahkan atau diubah sesuai dengan perencanaan yang telah dilakukan sebelumnya. Pemetaan data dari basis data lama ke basis data baru akan dijalankan dengan hati-hati untuk

memastikan integritas dan konsistensi data. Selama proses ini, perangkat lunak atau alat bantu seperti ETL (*Extract, Transform, Load*) dapat membantu memfasilitasi pemindahan dan transformasi data dengan efisiensi.

Setelah data berhasil dipindahkan, basis data baru perlu dikonfigurasi dan diintegrasikan dengan baik. Konfigurasi dilakukan untuk menyesuaikan basis data baru dengan kebutuhan dan persyaratan bisnis. Integrasi basis data baru dengan aplikasi atau sistem lain yang bergantung padanya juga perlu diperiksa untuk memastikan bahwa ketergantungan fungsional antara basis data dan aplikasi tetap berjalan lancar.

Uji coba fungsionalitas menjadi langkah berikutnya. Pada tahap ini, sistem basis data baru akan diuji untuk memastikan bahwa semua fungsionalitas dan proses bisnis berjalan dengan baik dan sesuai dengan harapan. Uji coba ini meliputi pengujian integrasi, pengujian fungsional, dan pengujian kinerja. Hasil dari uji coba ini akan membantu memastikan bahwa basis data baru beroperasi dengan efektif dan sesuai dengan kebutuhan bisnis.

Terakhir, dalam beberapa kasus, proses implementasi migrasi atau konversi dapat dilakukan secara bertahap atau per bagian. Hal ini dilakukan untuk mengurangi risiko dan dampak perubahan pada operasional bisnis. Dengan mengimplementasikan secara bertahap, perubahan dapat diperiksa dan diverifikasi sebelum seluruh sistem berpindah sepenuhnya ke basis data baru.

Proses implementasi migrasi atau konversi basis data memerlukan tim yang terkoordinasi dengan baik dan perencanaan yang matang. Setiap langkah harus dijalankan dengan hati-hati untuk memastikan kesuksesan proses migrasi atau konversi dan menjaga integritas data yang dipindahkan atau diubah. Dengan mengikuti langkah-langkah ini, organisasi dapat mencapai hasil yang diharapkan dan mendapatkan manfaat dari basis data yang lebih modern dan efisien.

13.5 Resiko dalam Proses Migrasi dan Konversi Data

Meskipun proses migrasi atau konversi telah direncanakan dengan matang, tetap ada beberapa risiko yang mungkin timbul selama implementasi. Pemahaman mendalam tentang risiko ini sangat penting agar langkah-langkah mitigasi yang tepat dapat diambil untuk mengurangi dampak negatifnya.

Salah satu risiko yang mungkin terjadi adalah kehilangan atau korupsi data. Selama migrasi atau konversi, data dapat terhapus atau rusak, mengakibatkan hilangnya informasi yang berharga atau data yang tidak valid. Untuk mengatasi risiko ini, strategi mitigasi yang perlu dilakukan adalah dengan melakukan backup lengkap dan teratur dari basis data lama sebelum melakukan migrasi atau konversi. Backup ini akan menjadi cadangan yang berharga untuk mengembalikan data jika terjadi masalah selama proses.

Risiko lain adalah downtime yang berlebihan atau tidak terduga. Jika proses migrasi atau konversi memerlukan waktu lama dan mengakibatkan downtime yang berarti, ini dapat mengganggu operasional bisnis dan menyebabkan kerugian finansial. Untuk mengurangi risiko ini, tim perlu melakukan perencanaan yang cermat dan menyusun jadwal migrasi atau konversi yang meminimalkan downtime. Metode paralel atau migrasi bertahap dapat digunakan untuk memastikan kontinuitas operasional dan membatasi waktu downtime.

Masalah kompatibilitas teknologi juga dapat menjadi risiko yang perlu diatasi. Basis data baru mungkin tidak sepenuhnya kompatibel dengan aplikasi atau sistem lain yang bergantung padanya, menyebabkan masalah dalam integrasi atau fungsionalitas. Strategi mitigasi dalam hal ini adalah dengan melakukan uji coba dan pengujian integrasi sebelum migrasi atau konversi penuh dilakukan. Uji coba ini akan

membantu mengidentifikasi dan menyelesaikan masalah kompatibilitas sebelum mencapai tahap implementasi.

Selain itu, risiko keamanan data juga menjadi perhatian penting. Selama proses migrasi atau konversi, data sensitif dapat menjadi rentan terhadap akses tidak sah atau kebocoran informasi. Strategi mitigasi dalam hal ini melibatkan penerapan protokol keamanan yang ketat, penggunaan enkripsi data, dan pemantauan aktif selama proses berlangsung.

Tidak kalah pentingnya, risiko kesalahan manusia dapat menjadi ancaman. Tim yang terlibat dalam migrasi atau konversi mungkin melakukan kesalahan saat mengelola data atau memindahkan informasi. Strategi mitigasi dalam hal ini adalah dengan memberikan pelatihan dan panduan yang komprehensif kepada tim terkait, serta melibatkan verifikasi data secara berulang untuk memastikan akurasi dan integritas.

Dengan pemahaman yang mendalam tentang risiko dan strategi mitigasi yang tepat, proses migrasi atau konversi basis data dapat dilakukan dengan lebih efisien dan aman. Tim akan lebih siap menghadapi potensi masalah dan dapat mengatasi tantangan yang muncul dengan lebih baik. Dengan demikian, kesuksesan implementasi migrasi atau konversi dapat tercapai, dan organisasi dapat mencapai manfaat dari basis data yang lebih modern, efisien, dan andal.

DAFTAR PUSTAKA

Fong, J., & Bloor, C. 1994. Data conversion rules from network to relational databases. Inf. Softw. Technol., 36, 141-153. https://doi.org/10.1016/0950-5849(94)90053-1.

Shin, S., Lee, M., Lee, W., Yoon, H., Sung, W., & Kim, K. 2011. A Data Migration Model and Case Study for Building Management System of Science and Technology Contents. Journal of the Korea Society of Computer and Information, 16, 123-135. https://doi.org/10.9708/JKSCI.2011.16.11.123.

Hartono, N., Utami, E., & Amborowati, A. 2016. Migrasi dan Optimalisasi Database Sistem Informasi Manajemen Universitas Cokroaminoto Palopo. Jurnal Buana Informatika, 7(4).

Handayanto, R. T., Sejati, N. A. S., Emeraldi, M. A., Ilham, M., Purwadhana, R., Fahreja, A., ... & Irvan, M. 2019. Migrasi dari Sistem Relasional ke Basis Data Objek. INFORMATION MANAGEMENT FOR EDUCATORS AND PROFESSIONALS: Journal of Information Management, 3(2), 173-178.

Setiyani, A., & Juarna, A. 2012. RESTRUKTURISASI BASIS DATA DISTRIBUTION PLANNING SYSTEM DI PT. XYZ.

Nugroho, R. S. A. 2005. Sistem Pertukaran Data antar Basis Data dengan XML. In *Seminar Nasional Aplikasi Teknologi Informasi (SNATI)*.

Tutorial Codeigniter 4: Migrasi Database. https://www.petanikode.com/codeigniter4-migration/

BIODATA PENULIS

Muhammad Fairuzabadi
Dosen pada Program Sarjana Informatika Fakultas Sains dan Teknologi Universitas PGRI Yogyakarta

Muhammad Fairuzabadi lahir di Bulukumba, Sulawesi Selatan tahun 1974. Alumni S1 Program Sarjana Ilmu Komputer FakultaS MIPA Universitas Kristen Immanuel (UKRIM) Yogyakarta Tahun 1998 dan S2 Program Magister Ilmu Komputer Universitas Gadjah Mada Tahun 2006. Bekerja sebagai dosen pada Program Sarjana Informatika Fakultas Sains dan Teknologi Universitas PGRI Yogyakarta. Selain itu, sebagai tenaga ahli dalam berbagai proyek pengembangan siistem informasi, *master plan* dan *blue print* TIK diintansi Pemerintah Daerah, Kementerian dan Lembaga. Diantaranya: Masterplan Jogja Smart Province, Cetak Biru Sistem Informasi Terintegrasi PDLKWS Kementerian Kehutanan dan Lingkungan Hidup, dan Rencana Induk Sistem Pemerintahan Berbasis Elektronik (SPBE) Pemerintah Kota Metro.

BIODATA PENULIS

Angga Aditya Permana
Dosen Program Studi Informatika
Fakultas Teknik dan Informatika
Universitas Multimedia Nusantara

Angga Aditya Permana (lahir pada Desember 1989 di Jakarta) seorang dosen full time di bidang Computer Science pada Universitas Multimedia Nusantara sejak tahun 2021, fokus penelitian pada kriptografi, steganografi serta keamanan computer, namun pada tahun 2021 sedang mendalami topik bioinformatika dan network science. Angga juga memiliki hobi yaitu touring dan juga bermain bulutangkis, saat ini sedang menjadi mahasiswa program doctoral pada IPB University. Memulai karir pertama kali sebagai Network Enginner tahun 2011 dan memulai profesinya sebagai dosen pada 2013 di kampus swasta yang ada di Jakarta, pernah juga mengajar di kampus negri yang berada di Jakarta dan Tangerang, juga pernah di undang menjadi dosen tamu untuk mengenalkan Bioinformatika di kampus negeri di Jakarta. Terimakasih..

BIODATA PENULIS

Melissa Indah Fianty, S.Kom., MMSI
Dosen Program Studi Sistem Informasi Fakultas Teknik dan Informatika Universitas Multimedia Nusantara

Penulis adalah dosen full-time pada Program Studi Sistem Informasi Fakultas Teknik dan Informatika, Universitas Multimedia Nusantara. Menyelesaikan pendidikan S1 pada Teknik Informatika dan melanjutkan S2 pada Manajemen Sistem Informasi di Bina Nusantara University.

BIODATA PENULIS

Dr. Erick Fernando, S.Kom., M.S.I.
Dosen Program Studi Sistem Informasi
Fakultas Teknik dan Informatik

Penulis saat ini merupakan dosen tetap pada Program Studi Sistem Informasi Fakultas Teknik dan Informatik, Universitas Multimedia Nusantara. Menyelesaikan pendidikan S1 pada Jurusan Teknik Informatika dan melanjutkan S2 pada Jurusan Magister Sistem Informasi. Serta Menyelesaikan S3 pada Doktor of Computer Science. Penulis menekuni bidang Basis data, Teknologi Blockchain, Adopsi Teknologi, Big Data, Sistem Informasi Geografis, *Emmbeded System* Dlll.

BIODATA PENULIS

Ir. Raymond Sunardi Oetama, MCIS.
Dosen Program Studi Sistem Informasi
Fakultas Teknik dan Informatika Universitas Multimedia Nusantara

Penulis adalah seorang profesional akademik yang sangat berprestasi dengan latar belakang pendidikan yang kuat. Ia memiliki gelar sarjana dalam matematika dari Institut Teknologi Sepuluh November yang terkenal di Surabaya, Indonesia, dan gelar magister dalam ilmu komputasi dan ilmu informasi dari Auckland University of Technology di Selandia Baru. Saat ini, ia menjabat sebagai Dosen dan Peneliti di Departemen Sistem Informasi di Universitas Multimedia Nusantara. Untuk pertanyaan atau informasi lebih lanjut, Raymond dapat dihubungi melalui email di raymond@umn.ac.id.

BIODATA PENULIS

Weri Sirait, S.Kom., M.Kom.
Dosen Program Studi Informatika
Universitas Metamedia

Penulis lahir di Dharmasraya tanggal 02 September 1993. Penulis merupakan Dosen tetap di Universitas Metamedia Program Studi Informatika. Penulis telah menyelesaikan pendidikan S1 pada Jurusan Sistem Informasi dan melanjutkan S2 pada Jurusan Teknik Informatika konsentrasi Sistem Informasi.

BIODATA PENULIS

Nur Azizah, S.Kom., M.Kom.
Dosen Program Studi Bisnis Digital
Universitas Metamedia

Penulis lahir di Padang Panjang tanggal 09 Juli 1996. Penulis adalah dosen tetap pada Program Studi Bisnis Digital, Universitas Metamedia. Menyelesaikan pendidikan S1 pada Jurusan Sistem Informasi dan melanjutkan S2 pada Jurusan Teknologi Informatika.

BIODATA PENULIS

Heru Saputra, S.Kom., M.Kom.
Dosen Program Studi Sistem Informasi
Fakultas Teknologi Informasi dan Industri Kreatif

Penulis lahir di Banda Aceh pada tanggal 01 Mei 1984. Penulis merupakan anak pertama dari tiga bersaudara dari pasangan Sugiarsono dan Supiana dan sudah berkeluarga dan memiliki 1 anak dengan nama Rania Navisha Shanum dan istri bernama Ilfa Stephane. Penulis merupakan dosen tetap pada Program Studi Sistem Informasi, Fakultas Teknologi Informasi dan Industri Kreatif, Universitas Metamedia. Menyelesaikan pendidikan S1 pada Jurusan Sistem Informasi dan melanjutkan S2 pada Jurusan Sistem Informasi. Penulis menekuni beberapa bidang ilmu, seperti Blockchain, Machine Learning, VR/AR dan Artificial Intelligence. Penulis berperan aktif sebagai editor jurnal Indonesian Journal of Computer Science (IJCS), reviwer di acara seminar nasional SENATIKA 3 dan SENATIKA 4 dan sebagai narasumber di berbagai acara seminal nasional.

BIODATA PENULIS

Wirawan Istiono, S.Kom., M.Kom.
Dosen Informatika
Fakultas Teknik dan Informatika, Universitas Multimedia Nusantara

Penulis lahir di Jakarta tanggal 13 April 1983. Penulis adalah dosen pada Program Studi Informatika Fakultas Teknik dan Informatika, Universitas Multimedia Nusantara. Menyelesaikan pendidikan S1 pada Jurusan Teknik Informatika dan melanjutkan S2 pada bidang jurusan yang Teknik Informatika. Pada saat ini, penulis menekuni bidang Penelitian terkait komputer sains, sistem komputer dan teknologi informasi

BIODATA PENULIS

Bambang Suprayogi, S.E., M.Si.
Dosen Program Studi Manajemen Informatika
POLITEKNIK LEMBAGA PENDIDIKAN DAN PENGEMBANGAN
PROFESI INDONESIA BANDUNG

Penulis adalah dosen tetap pada Program Studi Manajemen Informatika Politeknik Lembaga Pendidikan Dan Pengembangan Profesi Indonesia Bandung. Alumni SMAN 20 Bandung dan menyelesaikan pendidikan S1 pada Jurusan Manajemen di Jakarta dan melanjutkan S2 pada Jurusan Ilmu Administrasi di Jakarta. Penulis menekuni bidang Manajemen dan Computer Science dengan keahlian bidang Database Developer, Full Stack Developer, Network Operation System and Internet Cisco Certified Network Associate, .Net Programming, Java and Android Programming.

BIODATA PENULIS

Suwito Pomalingo, S.Kom., M.Kom
Dosen Program Studi Informatika
Universitas Multimedia Nusantara

Penulis adalah dosen tetap pada Program Studi Informatika Fakultas Teknik dan Informatika Universitas Multimedia Nusantara. Menyelesaikan pendidikan S1 pada Jurusan Teknik Informatika dan melanjutkan S2 pada Program Magister Informatika Universitas Islam Indonesia.

Saat ini sementara menyelesaikan studi Doktoral dengan topik penelitian di bidang Security Science dengan pendekatan Deep Learning. Penulis menekuni bidang keilmuan meliputi Cyber Security, Malware Analytics, Data Science, Machine Learning, dan Deep Learning, selain itu, penulis juga memiliki beberapa serifikasi internasional di bidang Cyber Security, Digital Forensics, Data Science dan Machine Learning.

BIODATA PENULIS

Rahmat Taufik R.L Bau, M.Kom.
Dosen Program Studi Sistem Informasi
Fakultas Teknik Universitas Negeri Gorontalo

Penulis lahir di Gorontalo tanggal 27 Juli 1989. Penulis adalah seorang dosen pada Program Studi Sistem Informasi Fakultas Teknik, Universitas Negeri Gorontalo. Menyelesaikan pendidikan S1 dan S2 pada Jurusan Teknologi Informasi. Penulis mengajar dan meneliti di bidang keahlian analisis dan perancangan. Sejak awal tahun 2023 bergabung di komunitas menulis bab chapter oleh Get Press Indonesia.