UNIVERSITAS
PRIMA
INDONESIA

Perkembangan Sistem
BASIS DATA

Penulis

Yonata Laia, M.Kom
Windania Purba, M.Kom

Editor:

Yonata Laia, M.Kom

Buku Ajar Sitem Basis Data

Penulis
Yonata Laia, M.Kom
Windania Purba, M.Kom

Editor
Yonata Laia

FAKULTAS SAIN DAN TEKNOLOGI
UNIVERSITAS PRIMA INDONESI
2025

Buku Ajar Sitem Basis Data

Penulis:

Yonata Laia M Kom
Windania Purba, M.Kom

Desain Isi:

Yonata Laia M Kom

Desain Cover:
Yonata Laia M Kom

Penerbit:

UNIVERSITAS
PRIMA INDONESIA
UNPRI PRESS

BAB 1

PENGANTAR SISTEM BASIS DATA

1.1 Latar Belakang dan Sejarah

Perkembangan teknologi informasi mendorong kebutuhan akan pengelolaan data yang cepat, akurat, dan terintegrasi. Pada awalnya, pengolahan data dilakukan secara manual menggunakan arsip kertas. Sistem ini memiliki berbagai kelemahan seperti redundansi (duplikasi data), inkonsistensi data, sulitnya pencarian data, rentan terhadap kehilangan atau kerusakan, serta tidak efisien dalam pengolahan data berskala besar.

Seiring berkembangnya komputer, sistem pengolahan data mulai beralih ke sistem berbasis file (file processing system). Dalam sistem ini, setiap aplikasi memiliki file datanya sendiri. Namun sistem ini masih memiliki berbagai kelemahan, seperti data yang terpisah-pisah di berbagai file, ketergantungan tinggi antara program dan data, sulit melakukan integrasi data, serta keamanan data yang masih terbatas.

Untuk mengatasi berbagai permasalahan tersebut, dikembangkanlah Sistem Basis Data (Database System). Sistem Basis Data adalah suatu sistem yang terdiri dari basis data (database), DBMS (Database Management System), pengguna (user), serta perangkat keras dan perangkat lunak pendukung yang saling berinteraksi untuk mengelola data secara efisien.

Tujuan utama sistem basis data antara lain:
1.Mengurangi redundansi data
2.Menjamin konsistensi data
3.Meningkatkan keamanan data
4.Mempermudah akses dan manipulasi data
5. Mendukung pengambilan keputusan berbasis informasi

B. Sejarah Perkembangan Sistem Basis Data

1. Era Manual (Sebelum 1960)

Pada masa ini, data disimpan dalam bentuk arsip fisik dan diproses secara manual. Risiko kehilangan data sangat tinggi dan proses pencarian data memakan waktu lama.

2. Era File Processing System (1960-an)

Data mulai disimpan dalam komputer menggunakan sistem berbasis file. Namun setiap aplikasi memiliki file sendiri sehingga redundansi dan inkonsistensi masih sering terjadi.

3.Era Model Hirarki dan Jaringan (1970-an)

Model Hirarki dikembangkan oleh IBM melalui Information Management System (IMS) dengan struktur berbentuk pohon (tree). Model ini cocok untuk hubungan satu-ke-banyak.

Model Jaringan (Network Model) dikembangkan oleh CODASYL dan memungkinkan hubungan banyak-ke-banyak dengan struktur yang lebih fleksibel namun kompleks.

4. Era Model Relasional (1970–1980-an)

Tonggak penting perkembangan basis data terjadi ketika Edgar F. Codd memperkenalkan model relasional pada tahun 1970 melalui paper “A Relational Model of Data for Large Shared Data Banks.” Model ini menyimpan data dalam bentuk tabel (relasi) dengan baris (tuple) dan kolom (atribut). Bahasa yang digunakan adalah SQL (Structured Query Language).

DBMS relasional yang populer antara lain Oracle Database, MySQL, Microsoft SQL Server, dan PostgreSQL.

5. Era Object-Oriented dan Distributed Database (1990-an)

Perkembangan internet mendorong lahirnya basis data terdistribusi dan arsitektur client-server. Sistem basis data mulai mendukung data multimedia serta integrasi dengan pemrograman berorientasi objek.

6. Era NoSQL dan Big Data (2000–Sekarang)
Muncul kebutuhan untuk menangani data tidak terstruktur dengan volume sangat besar dan kecepatan akses tinggi. Contoh sistem NoSQL antara lain MongoDB, Apache Cassandra, dan Firebase. Sistem ini mendukung skalabilitas horizontal dan cocok untuk aplikasi web serta mobile modern.

C. Perkembangan Teori dalam Sistem Basis Data

Perkembangan teori basis data mencakup:
1. Teori Model Data: Model Hirarki, Model Jaringan, Model Relasional, Model Objek, dan Model NoSQL.
2. Teori Normalisasi: Bertujuan mengurangi redundansi dan mencegah anomali data melalui tahapan 1NF, 2NF, 3NF, dan BCNF.
3. Teori Transaksi (ACID): Atomicity, Consistency, Isolation, dan Durability sebagai prinsip utama dalam pengelolaan transaksi database.

D. Desain Sistem Basis Data

Desain sistem basis data meliputi:
1. Perancangan Konseptual menggunakan ERD (Entity Relationship Diagram).
2. Perancangan Logis dengan transformasi ERD ke model relasional dan proses normalisasi.
3. Perancangan Fisik meliputi penentuan indeks, struktur penyimpanan, serta optimasi query.

E. Implementasi Sistem Basis Data

Tahap implementasi meliputi instalasi DBMS, pembuatan database, pembuatan tabel, penentuan constraint, pembuatan query (DDL, DML, DCL), serta pengujian sistem secara menyeluruh.

1.2 Pengertian Data, Informasi, dan Basis Data

Sistem Basis Data adalah sistem yang terdiri atas sekumpulan data yang saling berhubungan (database) dan perangkat lunak pengelola (Database Management

System/DBMS), sehingga memungkinkan data disimpan, diakses, dikelola, dan dimanipulasi secara efisien.

Basis data merupakan kumpulan data yang terorganisir dan saling berkaitan untuk memenuhi kebutuhan informasi suatu organisasi. Sedangkan DBMS adalah perangkat lunak yang digunakan untuk mendefinisikan, membuat, memelihara, serta mengontrol akses terhadap basis data.

A. Tujuan sistem basis data:
1. Mengurangi redundansi dan inkonsistensi data
2. Menjaga integritas dan keamanan data
3. Mempermudah akses dan manipulasi data
4. Mendukung pengambilan keputusan
5. Meningkatkan efisiensi kerja organisasi

B. Komponen Sistem Basis Data

1. Perangkat Keras (Hardware)
Meliputi komputer, server, media penyimpanan (hard disk/SSD), serta perangkat jaringan.

2. Perangkat Lunak (Software)
Meliputi DBMS, sistem operasi, dan aplikasi pendukung lainnya.

3. Data
Data merupakan komponen inti yang dapat berupa angka, teks, gambar, suara, maupun video.

4. Pengguna (User)
Terdiri dari Database Administrator (DBA), programmer, dan end user.

5. Prosedur (Procedure)
Merupakan aturan operasional yang mengatur penggunaan, keamanan, backup, dan recovery sistem basis data.

C. Fungsi DBMS

1. Data Definition
2. Data Manipulation
3. Data Security
4. Data Integrity
5. Backup dan Recovery

D. Karakteristik Sistem Basis Data

1Data terintegrasi dan terpusat
2. Mengurangi redundansi
3. Mendukung multi-user
4. Memiliki mekanisme keamanan
5. Menangani transaksi dalam jumlah besar

E. Kesimpulan

Sistem basis data merupakan fondasi utama dalam sistem informasi modern karena mampu mengelola data secara efektif, efisien, dan terstruktur.

1.3 Sistem Pemrosesan File Tradisional vs. Pendekatan Basis Data

A. Pengertian Sistem Pemrosesan File Tradisional

Sistem pemrosesan file tradisional (traditional file processing system) merupakan metode pengelolaan data yang digunakan sebelum berkembangnya sistem basis data modern. Pada sistem ini, setiap aplikasi atau program memiliki file datanya sendiri yang disimpan secara terpisah. Data biasanya disimpan dalam bentuk file teks atau file biner yang dikelola langsung oleh program aplikasi tanpa adanya sistem manajemen basis data (DBMS).

Dalam sistem ini, struktur data ditentukan dan dikendalikan oleh masing-masing program. Artinya, jika terdapat perubahan struktur data, maka program yang menggunakan data tersebut juga harus diubah. Sistem ini banyak

digunakan pada era awal komputerisasi, terutama pada tahun 1960–1970 sebelum model relasional berkembang luas.

Karakteristik utama sistem pemrosesan file tradisional antara lain:

1. Data disimpan dalam file terpisah untuk setiap aplikasi.
2. Tidak ada integrasi antar file secara terpusat.
3. Ketergantungan tinggi antara program dan struktur data.
4. Kontrol keamanan dan integritas data masih terbatas.
5. Redundansi data sering terjadi.

B. Kelemahan Sistem Pemrosesan File Tradisional

Meskipun pada masanya sistem ini dianggap memadai, dalam praktiknya terdapat berbagai kelemahan mendasar, yaitu:

1. Redundansi dan Inkonsistensi Data

Karena setiap aplikasi memiliki file sendiri, data yang sama sering disimpan di beberapa tempat. Hal ini menyebabkan pemborosan ruang penyimpanan dan berpotensi menimbulkan inkonsistensi apabila terjadi perbedaan pembaruan data.

2. Kesulitan Akses Data

Pengguna harus menggunakan program tertentu untuk mengakses data. Tidak ada bahasa query umum yang fleksibel seperti SQL.

3. Ketergantungan Program terhadap Data

Perubahan struktur file mengharuskan perubahan pada program yang menggunakan file tersebut. Hal ini meningkatkan biaya pemeliharaan sistem.

4. Kurangnya Keamanan dan Kontrol Akses

Pengaturan hak akses pengguna belum terstruktur dengan baik sehingga rentan terhadap penyalahgunaan.

5. Sulitnya Integrasi Data
Karena data tersebar di berbagai file terpisah, integrasi data untuk kebutuhan analisis menjadi sulit dan memakan waktu.

C. Pengertian Pendekatan Basis Data

Pendekatan basis data (database approach) merupakan metode pengelolaan data yang menggunakan sistem terintegrasi dan dikelola oleh perangkat lunak khusus yang disebut Database Management System (DBMS). Dalam pendekatan ini, data disimpan dalam satu sistem terpusat yang dapat digunakan oleh berbagai aplikasi secara bersamaan.

DBMS bertindak sebagai perantara antara pengguna dan data. Sistem ini memungkinkan definisi, manipulasi, pengamanan, dan pemeliharaan data secara terstruktur dan terkontrol.

Karakteristik pendekatan basis data antara lain:
1. Data terintegrasi dalam satu sistem terpusat.
2. Mengurangi redundansi data.
3. Mendukung multi-user.
4. Memiliki sistem keamanan dan kontrol akses.
5. Mendukung bahasa query standar seperti SQL.

D. Keunggulan Pendekatan Basis Data

Pendekatan basis data memberikan berbagai keunggulan dibanding sistem file tradisional, yaitu:

1. Pengendalian Redundansi
Dengan proses normalisasi dan manajemen relasi antar tabel, duplikasi data dapat diminimalkan.

2. Independensi Data
Struktur data dapat diubah tanpa harus mengubah seluruh program aplikasi (data independence).

3. Keamanan Lebih Baik

DBMS menyediakan mekanisme autentikasi dan otorisasi pengguna.

4. Integritas Data

Tersedia aturan constraint dan validasi untuk menjaga konsistensi data.

5. Backup dan Recovery

DBMS menyediakan fasilitas pencadangan dan pemulihan data apabila terjadi kerusakan.

6. Efisiensi Akses Data

Bahasa query memungkinkan pencarian data secara cepat dan fleksibel.

E. Perbandingan Sistem File Tradisional dan Pendekatan Basis Data

Secara umum, perbandingan kedua sistem dapat dilihat dari beberapa aspek berikut:

1. Struktur Data

- Sistem File: Data tersebar dalam file terpisah.
- Basis Data: Data terpusat dalam satu sistem.

2. Redundansi

- Sistem File: Tinggi.
- Basis Data: Rendah dan terkontrol.

3. Keamanan

- Sistem File: Terbatas.
- Basis Data: Terstruktur dan berbasis hak akses.

4. Integritas Data

- Sistem File: Tidak terjamin.
- Basis Data: Dijaga melalui constraint dan aturan integritas.

5. Skalabilitas
- Sistem File: Sulit dikembangkan.
- Basis Data: Mudah dikembangkan sesuai kebutuhan.

6. Dukungan Multi-User
- Sistem File: Terbatas.
- Basis Data: Mendukung banyak pengguna secara bersamaan.

F. Kesimpulan

Sistem pemrosesan file tradisional merupakan metode awal pengelolaan data yang memiliki banyak keterbatasan, terutama dalam hal redundansi, integrasi, dan keamanan data. Sebaliknya, pendekatan basis data menawarkan solusi yang lebih terstruktur, efisien, dan aman dalam mengelola data.

Dengan adanya DBMS, organisasi dapat mengelola data secara terpusat, mengurangi duplikasi, menjaga konsistensi, serta meningkatkan efisiensi operasional. Oleh karena itu, pendekatan basis data menjadi standar dalam pengembangan sistem informasi modern.

1.4 Database Management System (DBMS)

A. Pengertian Database Management System (DBMS)

Database Management System (DBMS) adalah perangkat lunak (software) yang dirancang secara khusus untuk memungkinkan interaksi antara pengguna (user), aplikasi, dan basis data. DBMS berfungsi sebagai perantara (interface) yang menghubungkan pengguna dengan data yang tersimpan dalam sistem komputer.

Tanpa adanya DBMS, pengguna harus berinteraksi langsung dengan file data yang kompleks dan sulit dikelola. Dengan DBMS, proses penyimpanan, pengambilan, perubahan, serta penghapusan data dapat dilakukan secara terstruktur, aman, dan efisien.

Secara umum, DBMS menyediakan lingkungan yang nyaman dan terkendali untuk menyimpan serta menarik data secara aman, konsisten, dan terintegrasi.

B. Fungsi Utama DBMS

1. Data Definition
DBMS memungkinkan pengguna mendefinisikan struktur database melalui bahasa definisi data (Data Definition Language/DDL). Pengguna dapat membuat tabel, menentukan tipe data, serta menetapkan relasi antar tabel.

2. Data Manipulation
Melalui Data Manipulation Language (DML), DBMS memungkinkan penambahan (insert), perubahan (update), penghapusan (delete), dan penarikan (select) data.

3. Data Security dan Authorization
DBMS menyediakan sistem autentikasi dan otorisasi untuk mengatur hak akses pengguna terhadap data.

4. Data Integrity
DBMS menjaga konsistensi data melalui aturan integritas (constraint) seperti primary key, foreign key, unique, dan not null.

5. Backup dan Recovery
DBMS menyediakan mekanisme pencadangan (backup) dan pemulihan (recovery) untuk melindungi data dari kerusakan atau kehilangan.

6. Concurrency Control
DBMS mampu mengatur akses data oleh banyak pengguna secara bersamaan tanpa menyebabkan konflik atau inkonsistensi.

C. Komponen Utama DBMS

1. Query Processor

Bertugas menerjemahkan perintah SQL menjadi instruksi yang dapat dipahami oleh sistem.

2. Storage Manager

Mengelola penyimpanan fisik data di dalam media penyimpanan.

3. Transaction Manager

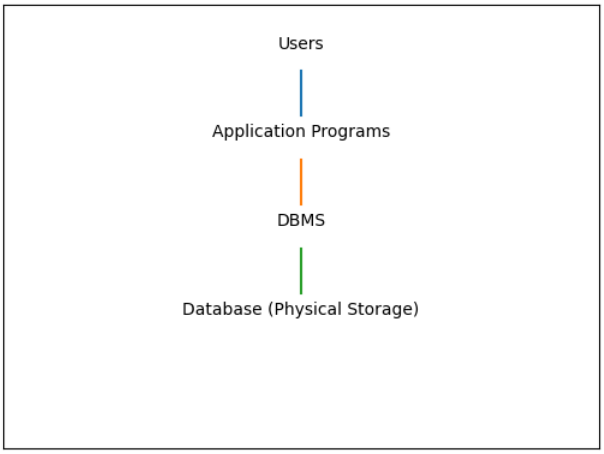
Mengatur transaksi agar memenuhi prinsip ACID (Atomicity, Consistency, Isolation, Durability).

4. Data Dictionary

Menyimpan metadata atau informasi tentang struktur database.

D. Arsitektur DBMS

Berikut merupakan gambaran sederhana arsitektur DBMS:



Arsitektur di atas menunjukkan bahwa pengguna (Users) tidak berinteraksi langsung dengan basis data fisik. Pengguna menggunakan aplikasi (Application Programs) yang kemudian mengirimkan instruksi atau query ke DBMS. DBMS bertugas memproses instruksi tersebut dan mengelola data yang tersimpan

secara fisik dalam media penyimpanan.

Dengan arsitektur ini, keamanan, integritas, serta efisiensi pengelolaan data dapat terjaga dengan baik karena seluruh akses data dikendalikan oleh DBMS.

E. Contoh DBMS Populer

Beberapa DBMS yang banyak digunakan saat ini antara lain:

1. MySQL – Banyak digunakan untuk aplikasi web dan sistem informasi skala menengah.
2. PostgreSQL – DBMS open-source yang kuat dan mendukung fitur lanjutan.
3. Oracle Database – Digunakan pada sistem enterprise berskala besar.
4. Microsoft SQL Server – Banyak digunakan dalam lingkungan bisnis berbasis Windows.

Setiap DBMS memiliki keunggulan dan karakteristik tersendiri, namun pada dasarnya memiliki fungsi utama yang sama, yaitu mengelola basis data secara efisien dan aman.

F. Kesimpulan

Database Management System (DBMS) merupakan komponen inti dalam sistem basis data modern. DBMS menyediakan berbagai fasilitas penting seperti pengelolaan struktur data, manipulasi data, keamanan, integritas, serta mekanisme backup dan recovery.

Dengan adanya DBMS, organisasi dapat mengelola data dalam jumlah besar secara terstruktur, aman, dan efisien. Oleh karena itu, DBMS menjadi fondasi utama dalam pengembangan sistem informasi di berbagai bidang.

SOAL LATIHAN BAB 1

1. Jelaskan perbedaan mendasar antara data dan informasi, berikan contoh di lingkungan kampus!
2. Sebutkan 3 kelemahan sistem pemrosesan file tradisional dibandingkan dengan menggunakan Database Management System (DBMS)!

BAB 2

ARSITEKTUR BASIS DATA

2.1 Entitas dan Atribut

A. Pengertian Abstraksi Data dalam Sistem Basis Data

Dalam sistem basis data, abstraksi data merupakan konsep penting yang digunakan untuk menyederhanakan interaksi pengguna dengan sistem database. Abstraksi data adalah proses penyembunyian detail teknis penyimpanan data sehingga pengguna tidak perlu memahami bagaimana data disimpan secara fisik di dalam media penyimpanan.

Konsep ini dikembangkan untuk mengatasi kompleksitas pengelolaan data dalam sistem yang besar dan multi-user. Dengan adanya abstraksi, sistem basis data dapat dipahami dalam beberapa tingkatan (level) yang berbeda sesuai kebutuhan pengguna. Pendekatan ini memungkinkan pemisahan antara cara data disimpan secara fisik dan cara data dilihat atau digunakan oleh pengguna.

Dalam arsitektur sistem basis data modern, terdapat tiga tingkatan utama abstraksi data, yaitu:

1. Tingkat Fisik (Physical Level)
2. Tingkat Logik/Konseptual (Logical/Conceptual Level)
3. Tingkat View (External Level)

Ketiga tingkatan ini membentuk kerangka kerja yang dikenal sebagai Three-Level Architecture atau Arsitektur Tiga Tingkat.

B. Tingkat Fisik (Physical Level)

Tingkat fisik merupakan level paling rendah dalam abstraksi data. Pada tingkat ini dijelaskan bagaimana data disimpan secara aktual di dalam media penyimpanan seperti hard disk atau SSD.

Level fisik berfokus pada:
- Struktur penyimpanan file
- Indeks
- Metode pengorganisasian data (sequential, indexed, hashing)
- Blok penyimpanan
- Manajemen ruang penyimpanan

Detail yang dibahas pada tingkat ini meliputi bagaimana record disimpan dalam blok disk, bagaimana pointer digunakan untuk menghubungkan data, serta bagaimana sistem mengoptimalkan kecepatan akses data.

Sebagai contoh, sebuah tabel mahasiswa pada tingkat fisik mungkin disimpan dalam bentuk file dengan struktur tertentu yang dioptimalkan menggunakan indeks B-Tree agar proses pencarian lebih cepat.

Pengguna umum tidak perlu memahami detail ini karena semuanya dikelola oleh DBMS. Namun, administrator basis data (DBA) perlu memahami tingkat fisik untuk melakukan optimasi performa dan pengelolaan penyimpanan.

C. Tingkat Logik atau Konseptual (Logical/Conceptual Level)

Tingkat logik atau konseptual merupakan level menengah dalam abstraksi data. Pada level ini dijelaskan struktur keseluruhan basis data secara logis tanpa memperhatikan bagaimana data disimpan secara fisik.

Level ini menggambarkan:
- Entitas (Entity)
- Atribut
- Relasi antar entitas
- Constraint atau aturan integritas

Pada tingkat ini, data biasanya direpresentasikan dalam bentuk model relasional (tabel), model ERD (Entity Relationship Diagram), atau model data lainnya.

Sebagai contoh, dalam sistem akademik terdapat entitas Mahasiswa, Dosen, dan Mata Kuliah. Relasi antara Mahasiswa dan Mata Kuliah dapat berupa relasi "mengambil". Pada level konseptual, hubungan ini digambarkan secara logis tanpa menjelaskan bagaimana data disimpan di disk.

Keuntungan dari tingkat logik adalah:
1. Memberikan gambaran menyeluruh tentang struktur database.
2. Memudahkan perancangan sistem.
3. Tidak bergantung pada detail teknis penyimpanan.

Level ini biasanya dirancang oleh analis sistem dan perancang basis data (database designer).

D. Tingkat View (External Level)

Tingkat view atau external level merupakan level tertinggi dalam abstraksi data. Level ini berhubungan langsung dengan pengguna akhir (end user).

Pada tingkat ini, pengguna hanya melihat sebagian data sesuai kebutuhan dan hak aksesnya. Setiap pengguna dapat memiliki view yang berbeda terhadap basis data yang sama.

Sebagai contoh:
- Bagian akademik hanya dapat melihat data nilai mahasiswa.
- Bagian keuangan hanya dapat melihat data pembayaran.
- Dosen hanya dapat melihat daftar mahasiswa dalam kelasnya.

View membantu meningkatkan:
1. Keamanan data
2. Kesederhanaan penggunaan
3. Relevansi informasi bagi pengguna tertentu

Dalam sistem relasional, view dapat dibuat menggunakan perintah SQL seperti CREATE VIEW, yang memungkinkan penyajian data tertentu tanpa mengubah struktur fisik database.

E. Hubungan Antar Tingkatan Abstraksi

Ketiga tingkatan abstraksi data saling berhubungan namun tetap independen. Konsep ini disebut sebagai data independence (independensi data).

Terdapat dua jenis independensi data:

1. Physical Data Independence
Kemampuan untuk mengubah struktur fisik penyimpanan tanpa memengaruhi struktur logik database.

2. Logical Data Independence
Kemampuan untuk mengubah struktur logik database tanpa memengaruhi view pengguna.

Independensi data merupakan salah satu keunggulan utama sistem basis data modern karena memungkinkan sistem berkembang tanpa mengganggu aplikasi yang sudah berjalan.

F. Manfaat Abstraksi Data

Penerapan abstraksi data memberikan berbagai manfaat, antara lain:

1. Menyederhanakan kompleksitas sistem.
2. Meningkatkan keamanan data.
3. Mendukung pengembangan sistem jangka panjang.
4. Memungkinkan pembagian tanggung jawab antara DBA, programmer, dan end user.
5. Mempermudah pemeliharaan dan pengembangan sistem.

G. Contoh Implementasi dalam Sistem Nyata

Dalam sistem perbankan, tingkat fisik mengatur bagaimana data transaksi disimpan dan diindeks. Tingkat logik menggambarkan relasi antara nasabah, rekening, dan transaksi. Sementara itu, tingkat view memungkinkan nasabah

melihat saldo dan riwayat transaksi melalui aplikasi mobile banking tanpa mengetahui struktur penyimpanan sebenarnya.

Contoh lain terdapat pada sistem akademik universitas, di mana mahasiswa hanya dapat melihat data pribadinya melalui portal akademik, sedangkan administrator dapat mengakses seluruh data mahasiswa.

H. Kesimpulan

Tingkatan abstraksi data (fisik, logik, dan view) merupakan konsep fundamental dalam sistem basis data yang bertujuan untuk menyederhanakan pengelolaan data dan meningkatkan keamanan serta fleksibilitas sistem. Dengan adanya tiga level ini, sistem basis data dapat dirancang secara modular, efisien, dan mudah dikembangkan.

Arsitektur tiga tingkat memungkinkan pemisahan antara detail teknis penyimpanan data dan kebutuhan informasi pengguna, sehingga sistem menjadi lebih stabil dan adaptif terhadap perubahan.

2.2 Kemandirian Data (Data Independence)

A. Pengertian Kemandirian Data (Data Independence)

Kemandirian data (data independence) merupakan salah satu konsep fundamental dalam sistem basis data modern yang berkaitan erat dengan arsitektur tiga tingkat (three-level architecture), yaitu tingkat fisik, tingkat logik (konseptual), dan tingkat view (eksternal).

Data independence adalah kemampuan sistem basis data untuk memungkinkan perubahan pada satu tingkat abstraksi tanpa harus memengaruhi tingkat lainnya. Konsep ini sangat penting dalam pengembangan dan pemeliharaan sistem informasi karena memungkinkan sistem berkembang secara fleksibel tanpa mengganggu aplikasi atau pengguna yang sudah ada.

Dalam sistem berbasis file tradisional, perubahan struktur data sering kali mengharuskan perubahan pada seluruh program aplikasi yang menggunakan data tersebut. Hal ini menyebabkan sistem menjadi kaku dan sulit dikembangkan. Sebaliknya, dalam sistem basis data modern, DBMS menyediakan mekanisme yang memungkinkan perubahan dilakukan secara terisolasi melalui prinsip kemandirian data.

B. Tujuan dan Pentingnya Data Independence

Kemandirian data bertujuan untuk:

1. Mengurangi ketergantungan antara program aplikasi dan struktur data.
2. Mempermudah pemeliharaan dan pengembangan sistem.
3. Meningkatkan fleksibilitas sistem dalam jangka panjang.
4. Mengurangi biaya modifikasi sistem.
5. Menjamin stabilitas aplikasi meskipun terjadi perubahan pada database.

Dalam organisasi besar yang memiliki banyak aplikasi dan pengguna, perubahan struktur database merupakan hal yang tidak terhindarkan. Tanpa adanya kemandirian data, setiap perubahan kecil dapat menimbulkan gangguan besar pada sistem secara keseluruhan.

C. Jenis-Jenis Kemandirian Data

Secara umum, terdapat dua jenis kemandirian data, yaitu:

1. Kemandirian Data Fisik (Physical Data Independence)
2. Kemandirian Data Logik (Logical Data Independence)

Kedua jenis ini berhubungan langsung dengan tiga tingkat abstraksi data dalam sistem basis data.

D. Kemandirian Data Fisik (Physical Data Independence)

Kemandirian data fisik adalah kemampuan untuk mengubah struktur penyimpanan fisik database tanpa memengaruhi struktur logik (konseptual) maupun view pengguna.

Perubahan pada tingkat fisik dapat meliputi:
- Perubahan metode penyimpanan data.
- Penambahan atau penghapusan indeks.
- Perubahan organisasi file (misalnya dari sequential menjadi indexed).
- Perubahan media penyimpanan (misalnya migrasi dari HDD ke SSD atau ke sistem cloud).
- Optimasi struktur penyimpanan untuk meningkatkan performa.

Sebagai contoh, administrator database dapat menambahkan indeks pada tabel mahasiswa untuk mempercepat pencarian data tanpa harus mengubah struktur tabel atau aplikasi yang menggunakan tabel tersebut.

Keunggulan utama kemandirian data fisik adalah:
1. Meningkatkan performa tanpa mengganggu aplikasi.
2. Memungkinkan optimasi sistem secara fleksibel.
3. Mengurangi risiko kesalahan akibat perubahan teknis.

Kemandirian data fisik relatif lebih mudah dicapai dibandingkan kemandirian data logik karena perubahan pada tingkat fisik tidak mengubah struktur logis data.

E. Kemandirian Data Logik (Logical Data Independence)

Kemandirian data logik adalah kemampuan untuk mengubah struktur logik (konseptual) database tanpa memengaruhi view pengguna atau program aplikasi.

Perubahan pada tingkat logik dapat meliputi:
- Penambahan atribut baru pada tabel.
- Penambahan entitas atau relasi baru.

- Perubahan struktur relasi antar tabel.
- Penggabungan atau pemisahan tabel.

Sebagai contoh, jika dalam sistem akademik ditambahkan atribut “Email Mahasiswa” pada tabel Mahasiswa, maka perubahan ini seharusnya tidak mengganggu aplikasi yang hanya menggunakan atribut NIM dan Nama, selama view yang digunakan tetap konsisten.

Kemandirian data logik lebih sulit dicapai dibandingkan kemandirian data fisik karena perubahan pada struktur logik berpotensi memengaruhi berbagai aplikasi yang menggunakan data tersebut. Oleh karena itu, DBMS menyediakan mekanisme seperti view dan schema mapping untuk menjaga konsistensi.

F. Hubungan Data Independence dengan Arsitektur Tiga Tingkat

Konsep data independence tidak dapat dipisahkan dari arsitektur tiga tingkat, yaitu:

1. Physical Level
2. Conceptual Level
3. External Level

Kemandirian data fisik berkaitan dengan pemisahan antara physical level dan conceptual level. Sedangkan kemandirian data logik berkaitan dengan pemisahan antara conceptual level dan external level.

Dengan adanya pemisahan ini, sistem basis data menjadi lebih modular dan mudah dikembangkan.

G. Manfaat Kemandirian Data dalam Sistem Informasi Modern

Penerapan data independence memberikan berbagai manfaat strategis, antara lain:

1. Fleksibilitas Pengembangan

Sistem dapat dikembangkan secara bertahap tanpa harus membangun ulang seluruh aplikasi.

2. Efisiensi Biaya

Perubahan struktur database tidak memerlukan perubahan besar pada aplikasi, sehingga menghemat waktu dan biaya.

3. Keamanan Sistem

Perubahan teknis dapat dilakukan tanpa mengganggu pengguna akhir.

4. Skalabilitas

Database dapat dikembangkan sesuai kebutuhan organisasi yang terus berkembang.

5. Stabilitas Operasional

Sistem tetap berjalan normal meskipun terjadi perubahan internal pada database.

H. Contoh Implementasi dalam Dunia Nyata

Dalam sistem perbankan, struktur penyimpanan data transaksi dapat diubah untuk meningkatkan performa tanpa memengaruhi aplikasi mobile banking yang digunakan nasabah. Ini merupakan contoh kemandirian data fisik.

Sementara itu, ketika bank menambahkan fitur baru seperti pencatatan kategori transaksi (misalnya belanja, transfer, pembayaran), struktur logik database dapat diperluas tanpa mengubah tampilan aplikasi yang digunakan nasabah, selama view tetap konsisten. Ini merupakan contoh kemandirian data logik.

Dalam sistem akademik universitas, perubahan struktur tabel internal tidak memengaruhi portal mahasiswa selama view yang disediakan tetap sama.

I. Tantangan dalam Mewujudkan Data Independence

Meskipun konsep ini sangat penting, penerapannya tidak selalu mudah. Beberapa tantangan yang sering dihadapi antara lain:

1. Kompleksitas sistem yang besar.
2. Ketergantungan aplikasi lama (legacy systems).
3. Kurangnya dokumentasi database.
4. Desain awal database yang kurang optimal.

Oleh karena itu, perancangan database yang baik sejak awal sangat menentukan tingkat keberhasilan penerapan kemandirian data.

J. Kesimpulan

Kemandirian data (data independence) merupakan salah satu prinsip utama dalam sistem basis data modern yang memungkinkan perubahan pada satu tingkat abstraksi tanpa memengaruhi tingkat lainnya.

Terdapat dua jenis kemandirian data, yaitu kemandirian data fisik dan kemandirian data logik. Keduanya berperan penting dalam menjaga fleksibilitas, efisiensi, dan stabilitas sistem informasi.

Dengan penerapan data independence yang baik, organisasi dapat mengembangkan sistem basis data secara berkelanjutan tanpa mengganggu aplikasi dan pengguna yang sudah ada, sehingga mendukung keberlangsungan operasional dalam jangka panjang

2.3 Bahasa Basis Data (DDL, DML, DCL)

A. Pengertian Bahasa Basis Data

Dalam sistem basis data modern, interaksi antara pengguna dengan database dilakukan melalui bahasa khusus yang dikenal sebagai bahasa basis data (database language). Bahasa ini digunakan untuk mendefinisikan struktur database, memanipulasi data, serta mengatur hak akses pengguna.

Pada sistem manajemen basis data relasional (RDBMS), bahasa yang paling umum digunakan adalah SQL (Structured Query Language). SQL menyediakan berbagai perintah yang dikelompokkan ke dalam beberapa kategori utama, yaitu Data Definition Language (DDL), Data Manipulation Language (DML), dan Data Control Language (DCL).

Ketiga kelompok bahasa ini memiliki fungsi yang berbeda namun saling melengkapi dalam pengelolaan basis data.

B. Data Definition Language (DDL)

Data Definition Language (DDL) adalah bagian dari bahasa basis data yang digunakan untuk mendefinisikan dan mengelola struktur database. DDL berhubungan dengan pembuatan, perubahan, dan penghapusan objek-objek dalam database.

Objek database yang dimaksud meliputi:
- Database
- Tabel
- View
- Index
- Constraint
- Schema

Beberapa perintah DDL yang umum digunakan antara lain:

1. CREATE
Digunakan untuk membuat objek baru dalam database.
Contoh:

```
CREATE TABLE Mahasiswa (
    NIM VARCHAR(10) PRIMARY KEY,
    Nama VARCHAR(100),
    Jurusan VARCHAR(50)
);
```

2. ALTER

Digunakan untuk mengubah struktur objek database yang sudah ada.

Contoh:

ALTER TABLE Mahasiswa ADD Email VARCHAR(100);

3. DROP

Digunakan untuk menghapus objek database secara permanen.

Contoh:

DROP TABLE Mahasiswa;

4. TRUNCATE

Digunakan untuk menghapus seluruh data dalam tabel tanpa menghapus struktur tabel.

DDL bersifat struktural karena perubahan yang dilakukan akan memengaruhi struktur database secara keseluruhan. Oleh karena itu, penggunaan DDL biasanya dilakukan oleh Database Administrator (DBA) atau perancang sistem.

C. Data Manipulation Language (DML)

Data Manipulation Language (DML) adalah bagian dari bahasa basis data yang digunakan untuk mengelola dan memanipulasi data yang tersimpan dalam tabel.

DML memungkinkan pengguna untuk:

- Menambahkan data
- Mengubah data
- Menghapus data
- Mengambil atau menampilkan data

Perintah DML yang umum digunakan meliputi:

1. INSERT

Digunakan untuk menambahkan data ke dalam tabel.

Contoh:

```
INSERT INTO Mahasiswa (NIM, Nama, Jurusan)
VALUES ('22001', 'Andi', 'Informatika');
```

2. SELECT

Digunakan untuk mengambil atau menampilkan data dari tabel.

Contoh:

```
SELECT * FROM Mahasiswa;
```

3. UPDATE

Digunakan untuk memperbarui data yang sudah ada.

Contoh:

```
UPDATE Mahasiswa
SET Jurusan = 'Sistem Informasi'
WHERE NIM = '22001';
```

4. DELETE

Digunakan untuk menghapus data tertentu dari tabel.

Contoh:

```
DELETE FROM Mahasiswa
WHERE NIM = '22001';
```

DML dapat dibedakan menjadi dua jenis, yaitu procedural dan non-procedural. SQL termasuk dalam bahasa non-procedural karena pengguna hanya menentukan data apa yang diinginkan tanpa harus menjelaskan bagaimana cara sistem mengambil data tersebut.

D. Data Control Language (DCL)

Data Control Language (DCL) adalah bagian dari bahasa basis data yang digunakan untuk mengatur hak akses dan keamanan database.

DCL berperan penting dalam menjaga keamanan data, terutama dalam sistem multi-user di mana banyak pengguna mengakses database secara bersamaan.

Beberapa perintah DCL yang umum digunakan antara lain:

1. GRANT

Digunakan untuk memberikan hak akses kepada pengguna tertentu.

Contoh:

GRANT SELECT, INSERT ON Mahasiswa TO user1;

2. REVOKE

Digunakan untuk mencabut hak akses yang telah diberikan.

Contoh:

REVOKE INSERT ON Mahasiswa FROM user1;

Melalui DCL, administrator dapat mengatur siapa yang berhak membaca, menambah, mengubah, atau menghapus data dalam database.

E. Hubungan antara DDL, DML, dan DCL

Ketiga jenis bahasa basis data ini saling melengkapi dalam pengelolaan database:

- DDL digunakan untuk membangun dan mengatur struktur database.
- DML digunakan untuk mengelola isi atau data dalam database.
- DCL digunakan untuk mengatur keamanan dan hak akses pengguna.

Tanpa DDL, struktur database tidak dapat dibuat. Tanpa DML, data tidak dapat dimasukkan atau diolah. Tanpa DCL, keamanan data tidak dapat dijaga dengan baik.

F. Peran Bahasa Basis Data dalam Sistem Informasi Modern

Bahasa basis data memiliki peran penting dalam pengembangan sistem informasi, antara lain:

1. Mempermudah Pengelolaan Data

SQL menyediakan sintaks yang sederhana namun kuat untuk mengelola data dalam jumlah besar.

2. Mendukung Multi-User Environment

Melalui DCL, akses pengguna dapat diatur sehingga tidak terjadi konflik atau penyalahgunaan data.

3. Mendukung Integritas dan Konsistensi

DDL memungkinkan penerapan constraint seperti primary key dan foreign key untuk menjaga integritas data.

4. Mendukung Analisis Data

Perintah SELECT dengan berbagai klausa seperti WHERE, GROUP BY, dan ORDER BY memungkinkan analisis data secara fleksibel.

G. Contoh Implementasi dalam Dunia Nyata

Dalam sistem akademik, DDL digunakan untuk membuat tabel Mahasiswa, Dosen, dan Mata Kuliah. DML digunakan untuk memasukkan data mahasiswa baru, memperbarui nilai, atau menampilkan daftar mahasiswa. DCL digunakan untuk membatasi akses sehingga mahasiswa hanya dapat melihat data pribadinya, sementara administrator dapat mengakses seluruh data.

Dalam sistem perbankan, DDL digunakan untuk mendefinisikan struktur rekening dan transaksi. DML digunakan untuk mencatat transaksi harian. DCL digunakan untuk memastikan bahwa hanya petugas tertentu yang dapat mengubah data rekening.

H. Kesimpulan

Bahasa basis data merupakan komponen penting dalam sistem manajemen basis data. DDL berfungsi untuk mendefinisikan struktur database, DML untuk memanipulasi data, dan DCL untuk mengatur keamanan serta hak akses pengguna.

Ketiga jenis bahasa ini bekerja secara terpadu untuk memastikan bahwa sistem basis data dapat berjalan secara terstruktur, aman, dan efisien dalam mendukung berbagai kebutuhan organisasi modern.

BAB 3

MODEL DATA DAN ENTITY RELATIONSHIP DIAGRAM (ERD)

3.1 Konsep Model Data

A. Pengertian Model Data

Model data merupakan representasi konseptual yang digunakan untuk menggambarkan struktur data, hubungan antar data, serta batasan (constraint) yang berlaku dalam suatu sistem basis data. Model data berfungsi sebagai kerangka kerja dalam perancangan database sehingga data dapat disusun secara sistematis, terstruktur, dan mudah dipahami.

Dalam sistem basis data, model data menjadi dasar utama sebelum database diimplementasikan secara fisik. Dengan adanya model data, perancang sistem

dapat menggambarkan kebutuhan informasi suatu organisasi dalam bentuk yang terstruktur dan logis.

Model data tidak hanya menjelaskan bagaimana data disimpan, tetapi juga bagaimana data tersebut saling berhubungan dan bagaimana aturan integritas diterapkan. Oleh karena itu, model data memiliki peran penting dalam proses analisis dan desain sistem basis data.

B. Tujuan Model Data

Penggunaan model data dalam perancangan basis data memiliki beberapa tujuan utama, yaitu:

1. Memberikan gambaran yang jelas mengenai struktur data.
2. Mengidentifikasi entitas dan atribut yang dibutuhkan dalam sistem.
3. Menentukan hubungan antar entitas.
4. Mengurangi redundansi data melalui perancangan yang terstruktur.
5. Menjadi dasar dalam pembuatan skema database.
6. Memudahkan komunikasi antara analis sistem, programmer, dan pengguna.

Dengan adanya model data, proses pengembangan sistem menjadi lebih terarah dan terencana.

C. Komponen Utama dalam Model Data

Secara umum, model data terdiri dari beberapa komponen utama, yaitu:

1. Entitas (Entity)
Entitas adalah objek atau konsep yang dapat dibedakan dan memiliki keberadaan dalam sistem. Contohnya adalah Mahasiswa, Dosen, Produk, Pelanggan, dan Transaksi.

2. Atribut (Attribute)
Atribut adalah karakteristik atau properti yang dimiliki oleh suatu entitas.

Misalnya, entitas Mahasiswa memiliki atribut seperti NIM, Nama, Alamat, dan Tanggal Lahir.

3. Relasi (Relationship)
Relasi adalah hubungan antara satu entitas dengan entitas lainnya. Contohnya, Mahasiswa “mengambil” Mata Kuliah, atau Pelanggan “melakukan” Transaksi.

4. Constraint
Constraint merupakan aturan atau batasan yang mengatur validitas data, seperti primary key, foreign key, dan aturan integritas lainnya.

D. Jenis-Jenis Model Data

Dalam perkembangan sistem basis data, terdapat beberapa jenis model data yang digunakan, antara lain:

1. Model Data Hirarki (Hierarchical Model)
Model ini menyusun data dalam struktur pohon (tree) dengan hubungan satu-ke-banyak. Setiap entitas anak hanya memiliki satu entitas induk.

2. Model Data Jaringan (Network Model)
Model ini memungkinkan hubungan banyak-ke-banyak antar entitas. Struktur model jaringan lebih fleksibel dibandingkan model hirarki.

3. Model Data Relasional (Relational Model)
Model relasional menyimpan data dalam bentuk tabel (relasi) yang terdiri dari baris (tuple) dan kolom (atribut). Model ini merupakan model yang paling banyak digunakan saat ini.

4. Model Data Berorientasi Objek (Object-Oriented Model)
Model ini mengintegrasikan konsep pemrograman berorientasi objek ke dalam sistem basis data, seperti objek, kelas, dan pewarisan (inheritance).

5. Model Data Konseptual
Model ini digunakan pada tahap awal perancangan untuk menggambarkan

kebutuhan sistem secara umum tanpa memperhatikan detail teknis implementasi.

E. Peran Model Data dalam Perancangan Database

Model data berperan sebagai dasar dalam proses desain database yang terdiri dari tiga tahap, yaitu:

1. Perancangan Konseptual
Pada tahap ini, model data digunakan untuk menggambarkan kebutuhan informasi secara umum, biasanya menggunakan Entity Relationship Diagram (ERD).

2. Perancangan Logis
Model data konseptual diterjemahkan ke dalam model relasional dengan menentukan tabel, atribut, dan relasi.

3. Perancangan Fisik
Model logis kemudian diimplementasikan ke dalam DBMS dengan mempertimbangkan aspek penyimpanan dan performa.

Tanpa model data yang baik, sistem database berisiko mengalami redundansi, inkonsistensi, dan kesulitan dalam pengembangan.

F. Contoh Penerapan Model Data

Sebagai contoh, dalam sistem akademik universitas, model data digunakan untuk menggambarkan entitas Mahasiswa, Dosen, Mata Kuliah, dan Nilai. Relasi antar entitas tersebut dirancang sebelum sistem diimplementasikan dalam bentuk tabel pada DBMS.

Dalam sistem e-commerce, model data digunakan untuk mendefinisikan entitas Produk, Pelanggan, Pesanan, dan Pembayaran. Dengan model data yang tepat, sistem dapat berjalan secara terstruktur dan efisien.

G. Kelebihan Penggunaan Model Data

Penggunaan model data memberikan berbagai keuntungan, antara lain:

1. Meningkatkan kualitas desain database.
2. Memudahkan proses dokumentasi sistem.
3. Mengurangi kesalahan dalam implementasi.
4. Mendukung integritas dan konsistensi data.
5. Mempermudah proses pemeliharaan dan pengembangan sistem.

H. Kesimpulan

Model data merupakan fondasi utama dalam perancangan sistem basis data. Dengan menggunakan model data, struktur data dan hubungan antar data dapat dirancang secara sistematis sebelum diimplementasikan ke dalam DBMS.

Keberadaan model data sangat penting dalam memastikan bahwa sistem basis data dapat berjalan secara efisien, terstruktur, dan mampu memenuhi kebutuhan informasi organisasi secara optimal.

3.2 Entitas dan Atribut

A. Pengantar Konsep Entitas dan Atribut

Dalam perancangan basis data, langkah awal yang sangat penting adalah memodelkan dunia nyata ke dalam bentuk konseptual. Dunia nyata yang kompleks harus disederhanakan agar dapat direpresentasikan dalam sistem basis data secara terstruktur dan sistematis. Proses ini dilakukan melalui identifikasi entitas dan atribut yang relevan dengan kebutuhan sistem.

Entitas dan atribut merupakan komponen dasar dalam Entity Relationship Diagram (ERD). Keduanya berperan sebagai fondasi dalam membangun model data konseptual sebelum diterjemahkan ke dalam bentuk tabel pada model relasional.

B. Pengertian Entitas (Entity)

Entitas adalah objek di dunia nyata yang memiliki keberadaan independen dan dapat dibedakan dari objek lain. Entitas dapat berupa benda fisik, individu, peristiwa, atau konsep yang informasinya perlu disimpan dalam sistem basis data.

Contoh entitas dalam berbagai sistem:
1. Sistem Akademik: Mahasiswa, Dosen, Mata Kuliah, Ruang Kelas.
2. Sistem Perbankan: Nasabah, Rekening, Transaksi.
3. Sistem E-Commerce: Produk, Pelanggan, Pesanan, Pembayaran.
4. Sistem Rumah Sakit: Pasien, Dokter, Rekam Medis.

Setiap entitas harus memiliki identitas yang unik sehingga dapat dibedakan dari entitas lainnya. Dalam implementasi database relasional, entitas biasanya direpresentasikan sebagai tabel.

C. Karakteristik Entitas

Beberapa karakteristik utama entitas antara lain:

1. Memiliki Keberadaan yang Jelas
Entitas harus merepresentasikan sesuatu yang nyata atau dapat didefinisikan secara konseptual.

2. Dapat Dibedakan dari Entitas Lain
Setiap entitas harus memiliki pembeda yang unik, biasanya dalam bentuk primary key.

3. Memiliki Atribut
Entitas selalu memiliki atribut yang menjelaskan karakteristiknya.

4. Berpartisipasi dalam Relasi
Entitas biasanya memiliki hubungan dengan entitas lain dalam sistem.

D. Jenis-Jenis Entitas

Dalam perancangan ERD, entitas dapat dibedakan menjadi beberapa jenis:

1. Entitas Kuat (Strong Entity)
Entitas yang dapat berdiri sendiri dan memiliki primary key. Contoh: Mahasiswa dengan NIM sebagai primary key.

2. Entitas Lemah (Weak Entity)
Entitas yang keberadaannya bergantung pada entitas lain dan tidak memiliki primary key sendiri secara lengkap. Contoh: Detail Nilai yang bergantung pada entitas Mahasiswa dan Mata Kuliah.

3. Entitas Asosiatif
Entitas yang terbentuk dari hubungan banyak-ke-banyak antara dua entitas dan biasanya memiliki atribut tambahan. Contoh: Entitas KRS yang menghubungkan Mahasiswa dan Mata Kuliah.

E. Pengertian Atribut (Attribute)

Atribut adalah properti atau karakteristik yang mendeskripsikan suatu entitas. Atribut memberikan informasi detail mengenai entitas yang bersangkutan.

Sebagai contoh, untuk entitas Mahasiswa, atributnya dapat berupa:
- NIM
- Nama
- Tanggal Lahir
- Alamat
- Program Studi
- Email

Tanpa atribut, entitas tidak memiliki informasi yang bermakna.

F. Jenis-Jenis Atribut

Atribut dalam ERD dapat diklasifikasikan menjadi beberapa jenis:

1. Atribut Sederhana (Simple Attribute)
Atribut yang tidak dapat dipecah lagi menjadi bagian yang lebih kecil. Contoh: NIM.

2. Atribut Komposit (Composite Attribute)
Atribut yang dapat dipecah menjadi beberapa bagian. Contoh: Alamat dapat terdiri dari Jalan, Kota, dan Kode Pos.

3. Atribut Tunggal (Single-Valued Attribute)
Atribut yang hanya memiliki satu nilai untuk setiap entitas. Contoh: Tanggal Lahir.

4. Atribut Multi-Nilai (Multi-Valued Attribute)
Atribut yang dapat memiliki lebih dari satu nilai. Contoh: Nomor Telepon.

5. Atribut Turunan (Derived Attribute)
Atribut yang nilainya diperoleh dari atribut lain. Contoh: Umur yang dihitung dari Tanggal Lahir.

6. Atribut Kunci (Key Attribute)
Atribut yang digunakan untuk mengidentifikasi entitas secara unik, seperti NIM pada entitas Mahasiswa.

G. Hubungan antara Entitas dan Atribut

Entitas dan atribut saling berkaitan erat. Entitas berfungsi sebagai wadah informasi, sedangkan atribut menjelaskan detail dari entitas tersebut. Dalam ERD, entitas digambarkan dengan persegi panjang, sedangkan atribut digambarkan dengan bentuk oval yang terhubung ke entitas.

Pemilihan atribut yang tepat sangat penting untuk menghindari redundansi data dan memastikan integritas database.

H. Contoh Studi Kasus

Misalnya dalam sistem akademik:

Entitas: Mahasiswa

Atribut:

- NIM (Primary Key)
- Nama
- Tanggal Lahir
- Alamat
- Program Studi

Entitas: Mata Kuliah

Atribut:

- Kode MK (Primary Key)
- Nama Mata Kuliah
- SKS

Dalam sistem e-commerce:

Entitas: Produk

Atribut:

- ID Produk (Primary Key)
- Nama Produk
- Harga
- Stok

Entitas: Pelanggan

Atribut:

- ID Pelanggan (Primary Key)
- Nama
- Alamat
- Email

I. Pentingnya Identifikasi Entitas dan Atribut yang Tepat

Identifikasi entitas dan atribut yang tepat akan menentukan kualitas desain database. Kesalahan dalam menentukan entitas atau atribut dapat menyebabkan:

1. Redundansi data.
2. Inkonsistensi data.
3. Kesulitan dalam pengembangan sistem.
4. Kompleksitas yang tidak perlu.

Oleh karena itu, analisis kebutuhan sistem harus dilakukan secara menyeluruh sebelum menentukan entitas dan atribut.

J. Peran Entitas dan Atribut dalam Implementasi Database

Setelah tahap perancangan konseptual selesai, entitas akan diterjemahkan menjadi tabel dalam database relasional, sedangkan atribut akan menjadi kolom dalam tabel tersebut. Primary key akan menjadi penanda unik setiap record, dan relasi antar entitas akan direpresentasikan melalui foreign key.

Dengan struktur yang tepat, sistem database dapat berjalan secara efisien, aman, dan mudah dikembangkan di masa depan.

K. Kesimpulan

Entitas dan atribut merupakan komponen fundamental dalam perancangan basis data. Entitas merepresentasikan objek atau konsep di dunia nyata yang perlu disimpan informasinya, sedangkan atribut mendeskripsikan karakteristik dari entitas tersebut.

Pemahaman yang baik mengenai jenis-jenis entitas dan atribut serta hubungan di antara keduanya akan menghasilkan desain database yang terstruktur, konsisten, dan efisien. Oleh karena itu, tahap identifikasi entitas dan atribut

harus dilakukan secara cermat dan sistematis dalam proses perancangan sistem basis data.

3.3 Relasi dan Derajat Relasi

A. Pengertian Relasi dalam Model Data

Dalam perancangan basis data, relasi merupakan komponen penting yang menghubungkan satu entitas dengan entitas lainnya. Relasi menggambarkan bagaimana objek-objek di dunia nyata saling berinteraksi atau memiliki keterkaitan.

Sebagai contoh, dalam sistem akademik, entitas “Mahasiswa” memiliki relasi “Mengambil” dengan entitas “Mata Kuliah”. Relasi tersebut menunjukkan bahwa mahasiswa mengikuti atau mengambil mata kuliah tertentu dalam suatu semester.

Relasi tidak hanya menunjukkan hubungan, tetapi juga membantu perancang sistem memahami alur data dan bagaimana informasi mengalir di dalam sistem sebelum diimplementasikan ke dalam tabel-tabel relasional.

B. Unsur-Unsur Relasi

Relasi dalam ERD memiliki beberapa unsur utama, yaitu:

1. Nama Relasi
Nama relasi biasanya berupa kata kerja yang menggambarkan hubungan antar entitas, seperti Mengambil, Mengajar, Membeli, atau Memiliki.

2. Entitas yang Terlibat
Relasi selalu melibatkan minimal dua entitas.

3. Derajat Relasi
Menunjukkan jumlah entitas yang terlibat dalam satu relasi.

4. Kardinalitas

Menentukan batasan jumlah instansiasi yang dapat terjadi dalam hubungan tersebut.

C. Derajat Relasi

Derajat relasi menunjukkan jumlah entitas yang terlibat dalam suatu hubungan.

1. Relasi Unary (Derajat 1)

Relasi yang terjadi pada satu entitas dengan dirinya sendiri. Contoh: entitas Pegawai dengan relasi “Membawahi” pada entitas Pegawai.

2. Relasi Binary (Derajat 2)

Relasi yang melibatkan dua entitas. Ini adalah bentuk relasi yang paling umum digunakan. Contoh: Mahasiswa – Mengambil – Mata Kuliah.

3. Relasi Ternary (Derajat 3)

Relasi yang melibatkan tiga entitas sekaligus. Contoh: Mahasiswa – Mengambil – Mata Kuliah – pada Semester tertentu.

Relasi dengan derajat lebih tinggi memungkinkan pemodelan yang lebih kompleks, terutama dalam sistem yang memiliki banyak variabel interaksi.

D. Kardinalitas Relasi

Kardinalitas relasi menentukan batasan jumlah instansiasi antara entitas yang berhubungan. Terdapat tiga jenis kardinalitas utama:

1. One-to-One (1:1)

Satu entitas pada sisi pertama hanya berhubungan dengan satu entitas pada sisi kedua, dan sebaliknya.
Contoh: Satu Mahasiswa memiliki satu Kartu Mahasiswa.

2. One-to-Many (1:N)

Satu entitas pada sisi pertama dapat berhubungan dengan banyak entitas pada

sisi kedua, tetapi setiap entitas pada sisi kedua hanya berhubungan dengan satu entitas pada sisi pertama.
Contoh: Satu Dosen mengajar banyak Mata Kuliah.

3. Many-to-Many (M:N)
Banyak entitas pada sisi pertama dapat berhubungan dengan banyak entitas pada sisi kedua.
Contoh: Mahasiswa dapat mengambil banyak Mata Kuliah, dan satu Mata Kuliah dapat diambil oleh banyak Mahasiswa.

Relasi M:N biasanya dipecah menjadi dua relasi 1:N melalui entitas perantara ketika diimplementasikan dalam model relasional.

E. Partisipasi Relasi

Selain kardinalitas, terdapat konsep partisipasi relasi, yaitu:

1. Partisipasi Total
Semua entitas harus terlibat dalam relasi.

2. Partisipasi Parsial
Tidak semua entitas harus terlibat dalam relasi.

Konsep ini membantu menentukan apakah suatu hubungan bersifat wajib atau opsional.

F. Contoh Studi Kasus Akademik

Dalam sistem akademik sederhana:

Entitas:
- Mahasiswa
- Mata Kuliah

Relasi:
- Mengambil

Kardinalitas:
Mahasiswa (M) — Mengambil — (N) Mata Kuliah

Relasi ini menunjukkan hubungan many-to-many yang kemudian akan diterjemahkan menjadi tabel KRS dalam database relasional.

G. Peran Relasi dalam Implementasi Database

Relasi yang dirancang dalam ERD akan diterjemahkan menjadi foreign key dalam model relasional. Kardinalitas 1:N akan menghasilkan foreign key pada sisi N, sedangkan relasi M:N akan menghasilkan tabel penghubung (junction table).

Pemahaman relasi yang tepat sangat penting untuk menghindari redundansi data dan menjaga integritas referensial.

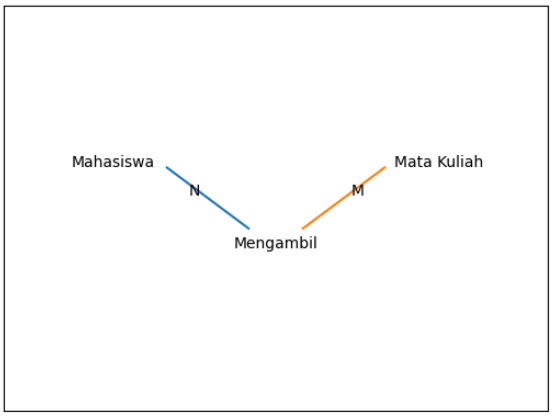
H. Pentingnya Relasi dalam Perancangan Sistem

Relasi membantu:
1. Menggambarkan alur data secara jelas.
2. Menentukan struktur tabel dalam database.
3. Menjaga konsistensi dan integritas data.
4. Menghindari kesalahan desain sistem.

Tanpa relasi yang jelas, database akan kehilangan struktur dan sulit untuk dikembangkan.

I. Gambar Contoh ERD

Berikut merupakan contoh ERD sederhana dalam lingkungan akademik:



Gambar di atas mengilustrasikan ERD sederhana di lingkungan akademik, di mana terlihat jelas bagaimana entitas Mahasiswa dan Mata Kuliah terhubung melalui relasi Mengambil dengan kardinalitas many-to-many (M:N).

Visualisasi ini membantu desainer sistem melihat struktur hubungan sebelum diimplementasikan ke dalam tabel fisik (relasional). Dengan adanya ERD, proses transformasi dari model konseptual ke model logis menjadi lebih sistematis dan terarah.

J. Kesimpulan

Relasi dan derajat relasi merupakan elemen penting dalam Entity Relationship Diagram (ERD). Relasi menggambarkan hubungan antar entitas, sedangkan kardinalitas dan derajat relasi menentukan batasan serta kompleksitas hubungan tersebut.

Pemahaman yang baik mengenai relasi, kardinalitas (1:1, 1:N, M:N), serta partisipasi akan menghasilkan desain database yang efisien, terstruktur, dan mudah diimplementasikan dalam sistem manajemen basis data. Oleh karena itu, analisis relasi harus dilakukan secara cermat pada tahap perancangan

konseptual sebelum database diimplementasikan secara fisik.

SOAL LATIHAN BAB 3

1. Sebuah perpustakaan universitas ingin membuat sistem basis data. Identifikasi minimal 3 Entitas dan masing-masing 3 Atribut dari entitas tersebut!
2. Gambarkan Entity Relationship Diagram (ERD) sederhana yang menghubungkan entitas Peminjam, Buku, dan Petugas Perpustakaan beserta kardinalitasnya.

BAB 4

MODEL DATA RELASIONAL

4.1 Konsep Tabel, Baris (Tuple), dan Kolom (Attribute)

A. Pengantar Model Data Relasional

Model data relasional merupakan model basis data yang paling banyak digunakan dalam sistem manajemen basis data modern. Model ini diperkenalkan oleh Edgar F. Codd pada tahun 1970 dan menjadi dasar bagi sistem Relational Database Management System (RDBMS) seperti MySQL, PostgreSQL, Oracle, dan Microsoft SQL Server.

Dalam model relasional, data direpresentasikan dalam bentuk tabel (relation). Setiap tabel terdiri dari baris (tuple) dan kolom (attribute). Struktur ini

memungkinkan data disimpan secara terorganisir, konsisten, dan mudah dimanipulasi menggunakan bahasa SQL.

Model relasional didasarkan pada teori himpunan (set theory) dan logika predikat (predicate logic), sehingga memiliki dasar matematis yang kuat. Oleh karena itu, pemahaman mengenai tabel, tuple, dan atribut sangat penting dalam memahami bagaimana database relasional bekerja.

B. Konsep Tabel (Relation)

Dalam model relasional, tabel disebut sebagai relation. Tabel merupakan struktur utama yang digunakan untuk menyimpan data. Setiap tabel merepresentasikan satu entitas atau satu jenis objek di dunia nyata.

Sebagai contoh, dalam sistem akademik, terdapat tabel Mahasiswa, tabel Dosen, dan tabel Mata Kuliah. Setiap tabel menyimpan data yang berkaitan dengan entitas tersebut.

Karakteristik utama tabel dalam model relasional adalah:

1. Memiliki nama yang unik dalam satu database.
2. Terdiri dari sejumlah kolom (attribute).
3. Terdiri dari sejumlah baris (tuple).
4. Tidak memiliki urutan baris yang tetap.
5. Tidak memiliki duplikasi tuple yang identik.
6. Setiap kolom memiliki domain atau tipe data tertentu.

Secara konseptual, tabel dapat digambarkan sebagai struktur dua dimensi yang terdiri dari baris dan kolom.

C. Konsep Baris (Tuple)

Baris dalam tabel disebut sebagai tuple. Tuple merepresentasikan satu record atau satu instansi dari entitas yang bersangkutan.

Sebagai contoh, dalam tabel Mahasiswa, satu baris dapat berisi:
NIM: 22001
Nama: Andi
Program Studi: Informatika

Baris tersebut merepresentasikan satu mahasiswa tertentu.

Karakteristik tuple dalam model relasional:

1. Setiap tuple harus unik.
2. Setiap tuple memiliki nilai untuk setiap atribut.
3. Tidak boleh ada dua tuple yang sepenuhnya identik.
4. Urutan tuple tidak memengaruhi makna data.

Keunikan tuple biasanya dijamin melalui penggunaan primary key, yaitu atribut yang secara unik mengidentifikasi setiap baris dalam tabel.

D. Konsep Kolom (Attribute)

Kolom dalam tabel disebut sebagai attribute. Attribute merepresentasikan karakteristik atau properti dari entitas.

Sebagai contoh, dalam tabel Mahasiswa, atributnya dapat berupa:
- NIM
- Nama
- Tanggal Lahir
- Alamat
- Program Studi

Setiap atribut memiliki domain, yaitu himpunan nilai yang diperbolehkan untuk atribut tersebut. Misalnya, atribut NIM bertipe VARCHAR(10), sedangkan atribut Tanggal Lahir bertipe DATE.

Karakteristik atribut:

1. Memiliki nama yang unik dalam satu tabel.
2. Memiliki tipe data tertentu.
3. Memiliki domain nilai.
4. Dapat memiliki constraint seperti NOT NULL, UNIQUE, atau DEFAULT.

E. Hubungan antara Tabel, Tuple, dan Attribute

Dalam model relasional, ketiga komponen ini saling berkaitan:

- Tabel adalah struktur utama.
- Tuple adalah baris dalam tabel.
- Attribute adalah kolom dalam tabel.

Setiap tabel terdiri dari sekumpulan tuple, dan setiap tuple terdiri dari nilai-nilai untuk setiap attribute.

Struktur ini memungkinkan data disimpan secara sistematis dan mudah diolah menggunakan query SQL.

F. Domain dan Tipe Data

Domain merupakan himpunan nilai yang diperbolehkan untuk suatu atribut. Domain ditentukan melalui tipe data yang digunakan dalam database.

Contoh tipe data yang umum digunakan dalam model relasional:

1. INT (bilangan bulat)
2. VARCHAR (teks dengan panjang tertentu)
3. DATE (tanggal)
4. FLOAT (bilangan desimal)
5. BOOLEAN (nilai benar/salah)

Penentuan domain yang tepat sangat penting untuk menjaga integritas dan konsistensi data.

G. Primary Key dan Integritas Entitas

Dalam model relasional, setiap tabel harus memiliki primary key. Primary key adalah atribut atau kombinasi atribut yang secara unik mengidentifikasi setiap tuple dalam tabel.

Contoh:
Pada tabel Mahasiswa, atribut NIM dapat dijadikan sebagai primary key.

Fungsi primary key:
1. Menjamin keunikan setiap baris.
2. Mencegah duplikasi data.
3. Menjadi acuan dalam pembuatan relasi dengan tabel lain (foreign key).

Konsep ini dikenal sebagai entity integrity, yaitu aturan bahwa primary key tidak boleh bernilai NULL.

H. Representasi Formal Model Relasional

Secara matematis, sebuah tabel (relation) dapat direpresentasikan sebagai himpunan tuple yang memiliki atribut tertentu.

Misalnya:
Mahasiswa (NIM, Nama, Program_Studi)

Setiap tuple dalam relasi tersebut merupakan kombinasi nilai yang sesuai dengan domain masing-masing atribut.

I. Keunggulan Struktur Tabel dalam Model Relasional

Struktur tabel memiliki beberapa keunggulan, antara lain:

1. Sederhana dan mudah dipahami.
2. Fleksibel dalam pengolahan data.
3. Mendukung operasi relasional seperti SELECT, PROJECT, JOIN.
4. Mudah dikembangkan dan dipelihara.
5. Memiliki dasar teoritis yang kuat.

J. Contoh Implementasi dalam Sistem Nyata

Dalam sistem perbankan, tabel Rekening menyimpan data nasabah, sedangkan tabel Transaksi menyimpan riwayat transaksi. Setiap baris dalam tabel Transaksi merepresentasikan satu transaksi tertentu.

Dalam sistem e-commerce, tabel Produk menyimpan informasi produk, dan setiap baris merepresentasikan satu produk.

Struktur tabel yang baik akan mempermudah analisis data dan pelaporan.

K. Tantangan dalam Perancangan Tabel

Meskipun terlihat sederhana, perancangan tabel harus memperhatikan:

1. Normalisasi untuk menghindari redundansi.
2. Penentuan tipe data yang tepat.
3. Penentuan primary key yang sesuai.
4. Hubungan antar tabel melalui foreign key.

Kesalahan dalam tahap ini dapat menyebabkan inkonsistensi dan kesulitan dalam pengembangan sistem.

L. Kesimpulan

Konsep tabel, baris (tuple), dan kolom (attribute) merupakan inti dari model data relasional. Tabel berfungsi sebagai wadah penyimpanan data, tuple merepresentasikan satu record, dan attribute mendeskripsikan karakteristik data.

Pemahaman yang mendalam mengenai ketiga konsep ini sangat penting dalam perancangan dan implementasi sistem basis data relasional. Dengan struktur yang tepat, database dapat berjalan secara efisien, konsisten, dan mudah dikembangkan sesuai kebutuhan organisasi.

4.2 Jenis-jenis Kunci (Super Key, Candidate Key, Primary Key, Foreign Key)

A. Pengantar Konsep Kunci dalam Model Relasional

Dalam model data relasional, konsep kunci (key) merupakan elemen yang sangat penting untuk menjamin keunikan data, menjaga integritas, serta membangun hubungan antar tabel. Kunci digunakan untuk mengidentifikasi setiap baris (tuple) dalam tabel secara unik dan untuk menghubungkan satu tabel dengan tabel lainnya.

Tanpa adanya kunci, database akan mengalami kesulitan dalam membedakan satu record dengan record lainnya. Selain itu, relasi antar tabel tidak dapat dibangun dengan baik sehingga berpotensi menimbulkan inkonsistensi data.

Konsep kunci berkembang dari teori himpunan dalam model relasional dan menjadi fondasi dalam penerapan integritas entitas (entity integrity) dan integritas referensial (referential integrity).

B. Super Key

Super key adalah satu atau lebih atribut dalam suatu tabel yang dapat digunakan untuk mengidentifikasi setiap tuple secara unik.

Super key tidak harus minimal. Artinya, super key dapat terdiri dari kombinasi atribut yang sebenarnya tidak semuanya diperlukan untuk menjamin keunikan.

Contoh:
Pada tabel Mahasiswa dengan atribut:
(NIM, Nama, Email, Program_Studi)

Jika NIM bersifat unik, maka:
- NIM adalah super key.

- NIM + Nama juga merupakan super key.
- NIM + Email juga merupakan super key.

Namun kombinasi tersebut tidak efisien karena mengandung atribut yang tidak diperlukan.

Karakteristik Super Key:
1. Menjamin keunikan setiap baris.
2. Dapat terdiri dari satu atau lebih atribut.
3. Bisa mengandung atribut tambahan yang tidak diperlukan.

C. Candidate Key

Candidate key adalah super key yang minimal, artinya tidak memiliki atribut berlebih. Jika satu atribut dihilangkan, maka sifat uniknya akan hilang.

Dengan kata lain, candidate key adalah super key yang paling efisien.

Contoh:
Dalam tabel Mahasiswa:
- NIM (unik)
- Email (unik)

Maka NIM dan Email masing-masing merupakan candidate key, karena keduanya dapat mengidentifikasi setiap mahasiswa secara unik tanpa atribut tambahan.

Karakteristik Candidate Key:
1. Bersifat unik.
2. Tidak memiliki atribut berlebih.
3. Bisa lebih dari satu dalam satu tabel.

Jika terdapat lebih dari satu candidate key, maka salah satunya akan dipilih sebagai primary key.

D. Primary Key

Primary key adalah candidate key yang dipilih untuk menjadi identitas utama suatu tabel. Primary key harus memiliki sifat unik dan tidak boleh bernilai NULL.

Fungsi Primary Key:
1. Mengidentifikasi setiap baris secara unik.
2. Menjamin integritas entitas (entity integrity).
3. Menjadi acuan dalam pembentukan relasi dengan tabel lain.

Contoh:
Pada tabel Mahasiswa:
NIM dipilih sebagai primary key.

Syarat Primary Key:
1. Tidak boleh bernilai NULL.
2. Harus unik.
3. Stabil (tidak sering berubah).
4. Sebaiknya sederhana dan ringkas.

Primary key dapat berupa:
- Single key (satu atribut).
- Composite key (gabungan beberapa atribut).

Contoh composite primary key:
Pada tabel KRS:
(NIM, Kode_MK) dapat menjadi primary key karena kombinasi keduanya unik.

E. Foreign Key

Foreign key adalah atribut dalam suatu tabel yang merujuk pada primary key di tabel lain. Foreign key digunakan untuk membangun relasi antar tabel dan menjaga integritas referensial.

Contoh:

Tabel Mahasiswa:

NIM (Primary Key)

Tabel KRS:

NIM (Foreign Key)

Kode_MK (Foreign Key)

Di sini, NIM pada tabel KRS merujuk ke NIM pada tabel Mahasiswa.

Fungsi Foreign Key:

1. Menghubungkan tabel yang berbeda.
2. Menjaga konsistensi data.
3. Mencegah data yatim (orphan data).

Aturan integritas referensial:

1. Nilai foreign key harus ada dalam primary key tabel referensi.
2. Dapat bernilai NULL jika relasi bersifat opsional.

F. Hubungan antar Jenis Kunci

Hubungan antar jenis kunci dapat dijelaskan sebagai berikut:

1. Semua candidate key adalah super key.
2. Tidak semua super key adalah candidate key.
3. Primary key dipilih dari salah satu candidate key.
4. Foreign key merujuk ke primary key di tabel lain.

Hierarki ini menunjukkan bahwa konsep kunci saling berkaitan dan membentuk struktur yang sistematis dalam model relasional.

G. Contoh Studi Kasus Akademik

Tabel Mahasiswa:

- NIM (Primary Key)

- Email (Candidate Key)
- Nama

Tabel Mata_Kuliah:
- Kode_MK (Primary Key)
- Nama_MK

Tabel KRS:
- NIM (Foreign Key)
- Kode_MK (Foreign Key)
- Nilai

Primary Key: (NIM, Kode_MK)

Dalam contoh ini:
- NIM dan Email adalah candidate key pada tabel Mahasiswa.
- NIM dipilih sebagai primary key.
- NIM pada tabel KRS menjadi foreign key.

H. Peran Kunci dalam Normalisasi

Kunci memiliki peran penting dalam proses normalisasi. Identifikasi candidate key membantu menentukan dependensi fungsional dan bentuk normal suatu tabel.

Tanpa pemahaman kunci yang baik, proses normalisasi akan sulit dilakukan dan berpotensi menghasilkan desain database yang tidak optimal.

I. Kesalahan Umum dalam Penentuan Kunci

Beberapa kesalahan umum dalam menentukan kunci antara lain:

1. Memilih atribut yang sering berubah sebagai primary key.
2. Tidak mendefinisikan primary key.
3. Menggunakan atribut yang tidak unik sebagai primary key.
4. Tidak menerapkan foreign key untuk menjaga integritas referensial.

Kesalahan tersebut dapat menyebabkan inkonsistensi dan kerusakan struktur database.

J. Keuntungan Penggunaan Kunci yang Tepat

1. Menjamin keunikan data.
2. Menjaga integritas referensial.
3. Mempermudah relasi antar tabel.
4. Mendukung performa query.
5. Menghindari duplikasi data.

K. Kesimpulan

Jenis-jenis kunci dalam model relasional meliputi super key, candidate key, primary key, dan foreign key. Super key menjamin keunikan, candidate key merupakan super key minimal, primary key adalah candidate key terpilih, dan foreign key membangun relasi antar tabel.

Pemahaman yang mendalam mengenai jenis-jenis kunci sangat penting dalam perancangan database relasional agar sistem dapat berjalan secara konsisten, aman, dan efisien.

4.3 Integritas Entitas dan Referensial

A. Pengantar Konsep Integritas dalam Model Relasional

Dalam sistem basis data relasional, integritas data merupakan aspek yang sangat penting untuk memastikan bahwa data yang tersimpan tetap akurat, konsisten, dan dapat dipercaya. Integritas data berkaitan dengan aturan-aturan yang menjaga validitas dan konsistensi informasi dalam database.

Dua konsep utama dalam integritas model relasional adalah integritas entitas (entity integrity) dan integritas referensial (referential integrity). Kedua konsep

ini menjadi dasar dalam menjaga kualitas dan keandalan data dalam sistem database.

Tanpa penerapan aturan integritas yang tepat, database dapat mengalami berbagai masalah seperti duplikasi data, data yang tidak lengkap, atau hubungan antar tabel yang tidak konsisten.

B. Integritas Entitas (Entity Integrity)

Integritas entitas adalah aturan yang menyatakan bahwa setiap tabel dalam model relasional harus memiliki primary key, dan nilai primary key tersebut tidak boleh bernilai NULL.

Tujuan utama integritas entitas adalah memastikan bahwa setiap baris (tuple) dalam tabel dapat diidentifikasi secara unik.

1. Prinsip Integritas Entitas
- Setiap tabel harus memiliki primary key.
- Nilai primary key tidak boleh NULL.
- Nilai primary key harus unik.

2. Contoh Integritas Entitas

Misalnya pada tabel Mahasiswa:

NIM (Primary Key)
Nama
Program Studi

Jika terdapat dua baris dengan NIM yang sama, maka integritas entitas dilanggar. Demikian pula jika terdapat baris dengan NIM bernilai NULL.

3. Pentingnya Integritas Entitas

Integritas entitas penting karena:
- Menjamin keunikan setiap record.

- Mencegah duplikasi data.
- Mempermudah proses pencarian dan pengolahan data.
- Menjadi dasar pembentukan relasi dengan tabel lain.

C. Integritas Referensial (Referential Integrity)

Integritas referensial adalah aturan yang memastikan bahwa hubungan antar tabel tetap konsisten. Aturan ini menyatakan bahwa nilai foreign key pada suatu tabel harus sesuai dengan nilai primary key pada tabel yang dirujuk.

Dengan kata lain, tidak boleh ada nilai foreign key yang merujuk ke data yang tidak ada.

1. Prinsip Integritas Referensial
- Nilai foreign key harus ada pada tabel referensi.
- Jika relasi bersifat opsional, foreign key dapat bernilai NULL.
- Perubahan atau penghapusan data pada tabel utama harus mempertimbangkan tabel yang terkait.

2. Contoh Integritas Referensial

Tabel Mahasiswa:
NIM (Primary Key)

Tabel KRS:
NIM (Foreign Key)
Kode_MK

Jika pada tabel KRS terdapat NIM = 22005, maka NIM tersebut harus ada dalam tabel Mahasiswa. Jika tidak, maka terjadi pelanggaran integritas referensial.

3. Mekanisme Penegakan Integritas Referensial

DBMS menyediakan beberapa opsi ketika terjadi perubahan pada tabel referensi, antara lain:

- RESTRICT: Menolak penghapusan data jika masih digunakan oleh tabel lain.
- CASCADE: Menghapus atau memperbarui data terkait secara otomatis.
- SET NULL: Mengatur foreign key menjadi NULL.
- SET DEFAULT: Mengatur foreign key ke nilai default tertentu.

D. Perbedaan Integritas Entitas dan Referensial

Integritas Entitas:
- Berfokus pada keunikan dan identitas setiap baris dalam tabel.
- Berkaitan dengan primary key.
- Tidak boleh bernilai NULL.

Integritas Referensial:
- Berfokus pada konsistensi hubungan antar tabel.
- Berkaitan dengan foreign key.
- Menjamin bahwa relasi antar tabel tetap valid.

E. Hubungan Integritas dengan Constraint dalam SQL

Dalam implementasi database, aturan integritas diterapkan melalui constraint dalam SQL, seperti:

- PRIMARY KEY
- FOREIGN KEY
- NOT NULL
- UNIQUE

Contoh penerapan integritas entitas:

```
CREATE TABLE Mahasiswa (
    NIM VARCHAR(10) PRIMARY KEY,
    Nama VARCHAR(100) NOT NULL
```

Contoh penerapan integritas referensial:

```
CREATE TABLE KRS (
    NIM VARCHAR(10),
    Kode_MK VARCHAR(10),
    FOREIGN KEY (NIM) REFERENCES Mahasiswa(NIM)
```

Constraint ini membantu DBMS menjaga konsistensi data secara otomatis.

F. Dampak Pelanggaran Integritas

Jika integritas tidak dijaga, dapat terjadi:

1. Data Duplikat
Tanpa primary key, data yang sama dapat muncul lebih dari sekali.

2. Data Yatim (Orphan Data)
Foreign key merujuk pada primary key yang sudah tidak ada.

3. Inkonsistensi Data
Perbedaan data pada tabel yang seharusnya saling terhubung.

4. Kesalahan dalam Analisis Data
Data yang tidak valid dapat menghasilkan laporan yang salah.

G. Studi Kasus Sistem Akademik

Dalam sistem akademik:

Tabel Mahasiswa memiliki primary key NIM.
Tabel Nilai memiliki foreign key NIM yang merujuk ke Mahasiswa.

Jika mahasiswa dihapus dari tabel Mahasiswa, DBMS harus menentukan apakah data Nilai ikut dihapus (CASCADE) atau penghapusan ditolak (RESTRICT).

Pengaturan ini sangat penting untuk menjaga konsistensi sistem.

H. Peran Integritas dalam Keamanan dan Kualitas Data

Integritas data tidak hanya menjaga konsistensi, tetapi juga meningkatkan kualitas dan keandalan sistem. Database yang memiliki aturan integritas yang kuat akan:

1. Mengurangi kesalahan input.
2. Menjaga stabilitas sistem.
3. Meningkatkan kepercayaan terhadap data.
4. Mendukung pengambilan keputusan yang akurat.

I. Hubungan Integritas dengan Normalisasi

Integritas data juga berkaitan dengan proses normalisasi. Tabel yang telah dinormalisasi dengan baik akan lebih mudah menjaga integritas karena struktur datanya sudah terorganisir secara efisien.

Primary key dan foreign key yang didefinisikan dengan benar akan membantu menjaga konsistensi hubungan antar tabel.

J. Kesimpulan

Integritas entitas dan integritas referensial merupakan dua prinsip utama dalam model data relasional. Integritas entitas memastikan bahwa setiap baris dalam tabel memiliki identitas unik melalui primary key, sedangkan integritas referensial memastikan bahwa hubungan antar tabel tetap konsisten melalui foreign key.

Penerapan aturan integritas yang tepat akan menghasilkan sistem basis data yang stabil, konsisten, dan dapat dipercaya. Oleh karena itu, pemahaman mengenai integritas data menjadi bagian penting dalam perancangan dan implementasi database relasional.

BAB 5

NORMALISASI BASIS DATA

5.1 Tujuan Normalisasi

A. Pengantar Normalisasi Basis Data

Normalisasi merupakan proses sistematis dalam perancangan basis data relasional yang bertujuan untuk mengorganisasi data agar terstruktur dengan baik serta mengurangi redundansi dan ketergantungan yang tidak diinginkan. Konsep normalisasi diperkenalkan sebagai bagian dari teori model relasional untuk memastikan bahwa struktur tabel memenuhi prinsip integritas dan efisiensi.

Dalam praktiknya, normalisasi dilakukan dengan membagi tabel-tabel besar yang kompleks menjadi tabel-tabel yang lebih kecil dan lebih terfokus, kemudian mendefinisikan hubungan antar tabel tersebut menggunakan primary key dan foreign key.

Proses normalisasi dilakukan melalui tahapan yang disebut bentuk normal (normal forms), seperti 1NF, 2NF, 3NF, hingga BCNF. Namun sebelum memahami tahapan tersebut, penting untuk memahami tujuan utama dari normalisasi itu sendiri.

B. Mengurangi Redundansi Data

Salah satu tujuan utama normalisasi adalah mengurangi redundansi data. Redundansi terjadi ketika data yang sama disimpan berulang kali dalam satu atau beberapa tabel.

Sebagai contoh, dalam tabel yang menyimpan data Mahasiswa dan Mata Kuliah sekaligus, informasi mengenai nama mata kuliah dapat berulang kali muncul untuk setiap mahasiswa yang mengambil mata kuliah tersebut. Hal ini menyebabkan pemborosan ruang penyimpanan dan meningkatkan risiko inkonsistensi.

Dengan normalisasi, data yang berulang dipisahkan ke dalam tabel tersendiri sehingga setiap informasi hanya disimpan satu kali. Pendekatan ini membuat database lebih efisien dan terorganisir.

C. Menghindari Anomali Data

Tujuan penting lainnya adalah mencegah terjadinya anomali data, yaitu ketidakkonsistenan atau kesalahan yang muncul akibat desain tabel yang tidak terstruktur dengan baik. Terdapat tiga jenis anomali utama:

1. Anomali Penyisipan (Insertion Anomaly)
Terjadi ketika data tidak dapat dimasukkan ke dalam tabel tanpa adanya data lain yang belum tersedia.

2. Anomali Penghapusan (Deletion Anomaly)
Terjadi ketika penghapusan suatu data menyebabkan hilangnya informasi lain yang sebenarnya masih dibutuhkan.

3. Anomali Pembaruan (Update Anomaly)
Terjadi ketika perubahan data harus dilakukan di banyak tempat, sehingga berisiko menimbulkan inkonsistensi jika tidak diperbarui secara menyeluruh.

Normalisasi membantu mengatasi ketiga anomali tersebut dengan menyusun tabel berdasarkan dependensi fungsional yang tepat.

D. Menjamin Integritas Data

Normalisasi bertujuan untuk memastikan bahwa struktur database mendukung integritas entitas dan integritas referensial. Dengan memisahkan data ke dalam tabel-tabel yang sesuai dan mendefinisikan kunci dengan benar, sistem dapat menjaga konsistensi hubungan antar data.

Sebagai contoh, dengan memisahkan tabel Mahasiswa dan tabel Mata Kuliah, serta menghubungkannya melalui tabel KRS, integritas hubungan antar data dapat dijaga dengan lebih baik.

E. Mempermudah Pemeliharaan dan Pengembangan Sistem

Database yang telah dinormalisasi lebih mudah dipelihara dan dikembangkan. Ketika struktur data sudah terorganisir dengan baik, perubahan pada satu bagian sistem tidak akan memberikan dampak besar pada bagian lain.

Hal ini sangat penting dalam sistem berskala besar yang terus berkembang. Dengan struktur yang modular, penambahan atribut atau entitas baru dapat dilakukan tanpa merombak keseluruhan sistem.

F. Meningkatkan Konsistensi dan Keakuratan Data

Normalisasi membantu memastikan bahwa setiap fakta hanya disimpan di satu tempat. Dengan demikian, jika terjadi perubahan data, cukup dilakukan pada satu tabel saja.

Pendekatan ini meningkatkan konsistensi dan mengurangi risiko kesalahan akibat data yang tidak sinkron di berbagai tabel.

G. Mendukung Efisiensi Penyimpanan

Meskipun pada beberapa kasus normalisasi dapat meningkatkan jumlah tabel, secara keseluruhan normalisasi membantu menghemat ruang penyimpanan karena menghilangkan duplikasi data.

Struktur yang efisien juga mendukung performa sistem dalam jangka panjang, terutama dalam pengelolaan database berukuran besar.

H. Mendukung Proses Query yang Lebih Terstruktur

Database yang telah dinormalisasi memungkinkan query dilakukan secara lebih terstruktur dan logis. Relasi antar tabel yang jelas memudahkan penggunaan operasi JOIN dalam SQL.

Meskipun query mungkin menjadi lebih kompleks karena melibatkan beberapa tabel, hasilnya akan lebih akurat dan konsisten.

I. Hubungan Normalisasi dengan Dependensi Fungsional

Tujuan normalisasi tidak dapat dipisahkan dari konsep dependensi fungsional. Dependensi fungsional menggambarkan hubungan antara atribut dalam suatu tabel.

Normalisasi memastikan bahwa setiap atribut non-kunci bergantung sepenuhnya pada primary key dan tidak bergantung pada atribut lain yang bukan kunci. Hal ini menjadi dasar dalam pembentukan bentuk normal kedua (2NF) dan ketiga (3NF).

J. Contoh Sederhana Tujuan Normalisasi

Misalnya terdapat tabel berikut:

NIM | Nama | Kode_MK | Nama_MK | Dosen

Jika satu mahasiswa mengambil beberapa mata kuliah, maka data Nama akan berulang. Dengan normalisasi, tabel tersebut dipecah menjadi:

1. Tabel Mahasiswa (NIM, Nama)
2. Tabel Mata Kuliah (Kode_MK, Nama_MK, Dosen)
3. Tabel KRS (NIM, Kode_MK)

Struktur ini menghilangkan redundansi dan menjaga konsistensi data.

K. Keseimbangan antara Normalisasi dan Performa

Meskipun normalisasi memiliki banyak tujuan positif, dalam praktik tertentu dilakukan denormalisasi untuk meningkatkan performa query. Oleh karena itu, tujuan normalisasi harus dipertimbangkan secara seimbang dengan kebutuhan performa sistem.

Desain database yang baik adalah desain yang mampu menjaga integritas sekaligus mempertimbangkan efisiensi akses data.

L. Kesimpulan

Tujuan normalisasi basis data adalah untuk mengorganisasi data secara sistematis agar mengurangi redundansi, mencegah anomali, menjaga integritas, serta meningkatkan konsistensi dan efisiensi sistem.

Dengan melakukan normalisasi, database menjadi lebih terstruktur, mudah dipelihara, dan mampu mendukung pengembangan sistem dalam jangka panjang. Oleh karena itu, normalisasi merupakan tahap penting dalam perancangan database relasional.

5.2 Anomali Data (Insert, Update, Delete)

A. Pengantar Anomali Data

Dalam perancangan basis data relasional, struktur tabel yang tidak dirancang dengan baik dapat menimbulkan berbagai permasalahan yang dikenal sebagai anomali data. Anomali data adalah ketidakkonsistenan atau kesalahan yang muncul akibat redundansi dan dependensi yang tidak tepat dalam suatu tabel.

Anomali biasanya terjadi pada tabel yang belum dinormalisasi dengan baik. Ketika satu tabel menyimpan terlalu banyak informasi yang saling bergantung, maka operasi penyisipan, pembaruan, atau penghapusan data dapat menyebabkan masalah serius.

Terdapat tiga jenis anomali utama dalam basis data, yaitu:
1. Anomali Penyisipan (Insert Anomaly)
2. Anomali Pembaruan (Update Anomaly)
3. Anomali Penghapusan (Delete Anomaly)

B. Anomali Penyisipan (Insert Anomaly)

Anomali penyisipan terjadi ketika data tidak dapat dimasukkan ke dalam tabel tanpa memasukkan data lain yang belum tersedia atau tidak relevan.

Contoh Kasus:

Misalkan terdapat tabel berikut:

NIM | Nama | Kode_MK | Nama_MK | Dosen

Jika ingin menambahkan data mata kuliah baru tetapi belum ada mahasiswa yang mengambilnya, maka kita tidak dapat memasukkan data tersebut karena kolom NIM dan Nama harus diisi. Hal ini menyebabkan keterbatasan dalam penyimpanan data.

Dampak Insert Anomaly:
1. Data tidak dapat disimpan secara fleksibel.
2. Informasi penting tertunda untuk dimasukkan.
3. Struktur tabel menjadi tidak efisien.

Penyebab utama insert anomaly adalah penggabungan beberapa entitas dalam satu tabel tanpa pemisahan yang tepat.

C. Anomali Pembaruan (Update Anomaly)

Anomali pembaruan terjadi ketika perubahan data harus dilakukan di banyak baris sekaligus. Jika salah satu baris tidak diperbarui, maka akan terjadi inkonsistensi data.

Contoh:

Jika satu dosen mengajar beberapa mahasiswa, maka nama dosen akan muncul berulang kali dalam tabel. Ketika nama dosen berubah (misalnya karena gelar akademik bertambah), maka perubahan harus dilakukan di semua baris terkait.

Jika ada satu baris yang terlewat, maka database menjadi tidak konsisten.

Dampak Update Anomaly:
1. Inkonsistensi data.
2. Kesalahan laporan.
3. Kesulitan dalam pemeliharaan sistem.

Update anomaly biasanya terjadi akibat redundansi data yang tinggi dalam satu tabel.

D. Anomali Penghapusan (Delete Anomaly)

Anomali penghapusan terjadi ketika penghapusan suatu data menyebabkan hilangnya informasi lain yang sebenarnya masih diperlukan.

Contoh:

Jika satu mahasiswa adalah satu-satunya yang mengambil mata kuliah tertentu, kemudian data mahasiswa tersebut dihapus, maka informasi mengenai mata kuliah dan dosennya juga ikut hilang.

Hal ini menyebabkan kehilangan data penting yang seharusnya tetap tersimpan.

Dampak Delete Anomaly:
1. Kehilangan informasi penting.
2. Hilangnya histori data.
3. Gangguan pada integritas database.

E. Penyebab Terjadinya Anomali Data

Anomali data umumnya disebabkan oleh:

1. Tabel yang belum dinormalisasi.
2. Penggabungan beberapa entitas dalam satu tabel.
3. Ketergantungan fungsional yang tidak tepat.
4. Tidak adanya pemisahan primary key dan foreign key secara jelas.

Desain database yang buruk akan meningkatkan kemungkinan terjadinya anomali.

F. Peran Normalisasi dalam Mengatasi Anomali

Normalisasi bertujuan untuk mengatasi anomali data dengan memecah tabel menjadi beberapa tabel yang lebih kecil berdasarkan dependensi fungsional.

Sebagai contoh, tabel:

NIM | Nama | Kode_MK | Nama_MK | Dosen

Dipecah menjadi:

1. Tabel Mahasiswa (NIM, Nama)
2. Tabel Mata_Kuliah (Kode_MK, Nama_MK, Dosen)
3. Tabel KRS (NIM, Kode_MK)

Dengan struktur ini:
- Insert anomaly dapat dihindari karena mata kuliah dapat dimasukkan tanpa mahasiswa.
- Update anomaly dapat dihindari karena perubahan dosen hanya dilakukan di satu tabel.
- Delete anomaly dapat dihindari karena penghapusan mahasiswa tidak menghapus data mata kuliah.

G. Dampak Anomali terhadap Sistem Informasi

Jika anomali tidak ditangani, maka sistem dapat mengalami:

1. Penurunan kualitas data.
2. Kesalahan dalam pengambilan keputusan.
3. Kerusakan hubungan antar tabel.
4. Menurunnya kepercayaan pengguna terhadap sistem.

Dalam sistem perbankan atau rumah sakit, anomali data dapat berdampak serius terhadap operasional dan keamanan informasi.

H. Studi Kasus Sederhana

Misalkan terdapat tabel Penjualan:

ID_Pelanggan | Nama_Pelanggan | Produk | Harga | Jumlah

Jika satu pelanggan membeli beberapa produk, maka Nama_Pelanggan akan berulang. Ketika pelanggan mengganti nama atau alamat, semua baris harus diperbarui.

Dengan normalisasi, tabel dipecah menjadi:

1. Pelanggan (ID_Pelanggan, Nama_Pelanggan)
2. Produk (ID_Produk, Harga)
3. Transaksi (ID_Pelanggan, ID_Produk, Jumlah)

Struktur ini mengurangi redundansi dan mencegah anomali.

I. Hubungan Anomali dengan Bentuk Normal

Anomali data biasanya muncul pada tabel yang belum memenuhi bentuk normal tertentu.

1. 1NF menghilangkan nilai berulang dan atribut multi-nilai.
2. 2NF menghilangkan dependensi parsial.
3. 3NF menghilangkan dependensi transitif.

Semakin tinggi tingkat normalisasi, semakin kecil kemungkinan terjadinya anomali.

J. Kesimpulan

Anomali data (insert, update, dan delete anomaly) merupakan masalah yang timbul akibat desain tabel yang tidak terstruktur dengan baik. Ketiga anomali tersebut dapat menyebabkan inkonsistensi, kehilangan data, dan kesalahan dalam sistem.

Normalisasi basis data menjadi solusi utama untuk mengatasi anomali dengan memecah tabel menjadi struktur yang lebih terorganisir dan sesuai dengan dependensi fungsional.

Pemahaman mengenai anomali data sangat penting dalam perancangan database agar sistem yang dibangun memiliki kualitas data yang tinggi, konsisten, dan dapat diandalkan dalam jangka panjang.

5.3 Bentuk Normal Pertama (1NF) hingga Ketiga (3NF)

A. Pengantar Bentuk Normal (Normal Forms)

Dalam proses normalisasi basis data, struktur tabel diperbaiki secara bertahap melalui tahapan yang disebut bentuk normal (normal forms). Setiap bentuk normal memiliki aturan tertentu yang harus dipenuhi agar tabel terbebas dari redundansi dan anomali data.

Tiga bentuk normal pertama—First Normal Form (1NF), Second Normal Form (2NF), dan Third Normal Form (3NF)—merupakan fondasi utama dalam desain database relasional. Ketiga bentuk ini bertujuan untuk mengorganisasi data berdasarkan dependensi fungsional sehingga setiap atribut berada pada tempat yang tepat.

B. Bentuk Normal Pertama (First Normal Form / 1NF)

1NF adalah tahap pertama dalam normalisasi. Sebuah tabel dikatakan berada dalam 1NF apabila:

1. Setiap kolom hanya memiliki nilai atomik (tidak dapat dipecah lagi).
2. Tidak terdapat atribut multi-nilai.
3. Tidak terdapat kelompok data berulang dalam satu baris.
4. Setiap baris dapat diidentifikasi secara unik (memiliki primary key).

Contoh Tabel Tidak 1NF:

NIM | Nama | Mata Kuliah
22001 | Andi | Basis Data, Algoritma
22002 | Budi | Pemrograman

Pada tabel di atas, kolom Mata Kuliah mengandung lebih dari satu nilai dalam satu sel, sehingga melanggar prinsip atomisitas.

Perbaikan ke 1NF:

NIM | Nama | Mata Kuliah
22001 | Andi | Basis Data
22001 | Andi | Algoritma
22002 | Budi | Pemrograman

Setelah dipecah menjadi baris terpisah, tabel telah memenuhi 1NF karena setiap kolom hanya berisi satu nilai atomik.

Tujuan 1NF:
- Menghilangkan data berulang.
- Menjamin setiap atribut bersifat atomik.
- Membentuk struktur tabel yang lebih teratur.

C. Bentuk Normal Kedua (Second Normal Form / 2NF)

Sebuah tabel dikatakan berada dalam 2NF jika:

1. Telah memenuhi 1NF.
2. Tidak memiliki dependensi parsial terhadap primary key.

Dependensi parsial terjadi ketika atribut non-kunci bergantung hanya pada sebagian dari primary key (pada tabel dengan composite key).

Contoh:

Tabel KRS:
(NIM, Kode_MK, Nama_Mahasiswa, Nama_MK, Nilai)

Primary Key: (NIM, Kode_MK)

Masalah:
- Nama_Mahasiswa bergantung hanya pada NIM.
- Nama_MK bergantung hanya pada Kode_MK.

Artinya terdapat dependensi parsial.

Perbaikan ke 2NF:

1. Tabel Mahasiswa (NIM, Nama_Mahasiswa)
2. Tabel Mata_Kuliah (Kode_MK, Nama_MK)
3. Tabel KRS (NIM, Kode_MK, Nilai)

Setelah pemisahan, semua atribut non-kunci bergantung sepenuhnya pada primary key masing-masing tabel.

Tujuan 2NF:
- Menghilangkan dependensi parsial.
- Mengurangi redundansi data.
- Memperjelas struktur relasi.

D. Bentuk Normal Ketiga (Third Normal Form / 3NF)

Sebuah tabel berada dalam 3NF jika:

1. Telah memenuhi 2NF.
2. Tidak memiliki dependensi transitif.

Dependensi transitif terjadi ketika atribut non-kunci bergantung pada atribut non-kunci lainnya.

Contoh:

Tabel Mahasiswa:
NIM | Nama | Kode_Prodi | Nama_Prodi

Primary Key: NIM

Masalah:
Nama_Prodi bergantung pada Kode_Prodi, bukan langsung pada NIM.

Ini disebut dependensi transitif karena:
NIM → Kode_Prodi → Nama_Prodi

Perbaikan ke 3NF:

1. Tabel Mahasiswa (NIM, Nama, Kode_Prodi)
2. Tabel Program_Studi (Kode_Prodi, Nama_Prodi)

Dengan pemisahan ini, setiap atribut non-kunci hanya bergantung langsung pada primary key.

Tujuan 3NF:
- Menghilangkan dependensi transitif.
- Meningkatkan integritas data.
- Mengurangi risiko anomali pembaruan.

E. Ringkasan Perbedaan 1NF, 2NF, dan 3NF

1NF:
Fokus pada atomisitas dan penghapusan atribut multi-nilai.

2NF:
Fokus pada penghapusan dependensi parsial pada composite key.

3NF:
Fokus pada penghapusan dependensi transitif antar atribut non-kunci.

Setiap tahap normalisasi memperbaiki struktur tabel agar semakin efisien dan konsisten.

F. Manfaat Penerapan Hingga 3NF

Penerapan hingga 3NF memberikan manfaat sebagai berikut:

1. Mengurangi redundansi data secara signifikan.
2. Menghindari anomali insert, update, dan delete.
3. Menjaga integritas referensial.
4. Mempermudah pemeliharaan sistem.
5. Mendukung pengembangan database jangka panjang.

G. Studi Kasus Lengkap Transformasi

Misalkan terdapat tabel awal:

NIM | Nama | Kode_MK | Nama_MK | Dosen | Nilai

Langkah 1 (1NF):
Memastikan tidak ada atribut multi-nilai.

Langkah 2 (2NF):
Pisahkan atribut yang bergantung sebagian:
Mahasiswa (NIM, Nama)
Mata_Kuliah (Kode_MK, Nama_MK, Dosen)
KRS (NIM, Kode_MK, Nilai)

Langkah 3 (3NF):
Jika Dosen memiliki atribut tambahan seperti Ruang_Kantor yang bergantung pada Dosen, maka dibuat tabel terpisah:
Dosen (Kode_Dosen, Nama_Dosen, Ruang_Kantor)

Struktur akhir menjadi lebih terorganisir dan bebas anomali.

H. Hubungan Normalisasi dan Kinerja Sistem

Meskipun normalisasi meningkatkan integritas, terkadang database yang terlalu terfragmentasi dapat memerlukan banyak operasi JOIN. Oleh karena itu, dalam praktik tertentu dilakukan denormalisasi untuk meningkatkan performa.

Namun secara umum, 3NF dianggap sebagai standar optimal dalam perancangan database relasional.

I. Kesimpulan

Bentuk Normal Pertama (1NF), Kedua (2NF), dan Ketiga (3NF) merupakan tahapan penting dalam proses normalisasi basis data. 1NF memastikan setiap atribut bersifat atomik, 2NF menghilangkan dependensi parsial, dan 3NF menghilangkan dependensi transitif.

Dengan menerapkan ketiga bentuk normal ini, database menjadi lebih terstruktur, konsisten, efisien, dan bebas dari anomali data. Oleh karena itu, pemahaman dan penerapan normalisasi hingga 3NF sangat penting dalam perancangan sistem basis data relasional.

BAB 6

STRUCTURED QUERY LANGUAGE (SQL) DASAR

6.1 Pengenalan SQL

A. Pengertian Structured Query Language (SQL)

Structured Query Language (SQL) adalah bahasa standar yang digunakan untuk berkomunikasi dengan sistem manajemen basis data relasional (Relational Database Management System/RDBMS). SQL dirancang untuk mengelola data dalam database relasional, baik dalam hal pembuatan struktur data, manipulasi isi data, maupun pengaturan hak akses pengguna.

SQL pertama kali dikembangkan oleh IBM pada awal tahun 1970-an berdasarkan model relasional yang diperkenalkan oleh Edgar F. Codd. Sejak saat itu, SQL menjadi bahasa standar internasional yang digunakan oleh berbagai sistem database seperti MySQL, PostgreSQL, Oracle Database, dan Microsoft SQL Server.

Sebagai bahasa standar, SQL memungkinkan pengguna untuk melakukan berbagai operasi terhadap database tanpa perlu memahami detail teknis

penyimpanan data secara fisik. SQL bekerja pada tingkat logis, sehingga pengguna dapat berfokus pada data yang diinginkan tanpa memikirkan bagaimana data tersebut disimpan di dalam sistem.

B. Fungsi Utama SQL

SQL memiliki beberapa fungsi utama dalam sistem basis data relasional, antara lain:

1. Membuat dan Mengelola Struktur Database
SQL digunakan untuk membuat database baru, tabel, indeks, dan objek lainnya.

2. Memasukkan dan Memanipulasi Data
SQL memungkinkan pengguna untuk menambahkan, memperbarui, menghapus, dan mengambil data.

3. Mengontrol Akses Pengguna
SQL digunakan untuk mengatur hak akses dan keamanan data.

4. Mengelola Transaksi
SQL mendukung pengelolaan transaksi untuk menjaga konsistensi data.

Dengan fungsi-fungsi tersebut, SQL menjadi alat utama dalam pengelolaan database modern.

C. Karakteristik SQL

SQL memiliki beberapa karakteristik penting:

1. Bersifat Deklaratif
Pengguna menyatakan data apa yang diinginkan, bukan bagaimana cara mendapatkannya.

2. Mudah Dipahami
Sintaks SQL relatif sederhana dan menyerupai bahasa Inggris.

3. Standar Internasional

SQL distandarisasi oleh ANSI dan ISO.

4. Mendukung Multi-User

SQL dirancang untuk lingkungan database yang dapat diakses oleh banyak pengguna secara bersamaan.

D. Kategori Perintah dalam SQL

Perintah dasar dalam SQL dikategorikan menjadi beberapa kelompok utama, yaitu:

1. Data Definition Language (DDL)

DDL digunakan untuk mendefinisikan dan mengelola struktur database.

Perintah DDL meliputi:

- CREATE (membuat tabel atau database)
- ALTER (mengubah struktur tabel)
- DROP (menghapus tabel atau database)
- TRUNCATE (menghapus seluruh isi tabel)

Contoh:

```
CREATE TABLE Mahasiswa (
    NIM VARCHAR(10) PRIMARY KEY,
    Nama VARCHAR(100),
    Program_Studi VARCHAR(50)
);
```

2. Data Manipulation Language (DML)

DML digunakan untuk memanipulasi data yang terdapat dalam tabel.

Perintah DML meliputi:

- INSERT (menambahkan data)
- SELECT (mengambil data)

- UPDATE (memperbarui data)
- DELETE (menghapus data)

Contoh:

```
INSERT INTO Mahasiswa VALUES ('22001', 'Andi', 'Informatika');

SELECT * FROM Mahasiswa;
```

3. Data Control Language (DCL)

Digunakan untuk mengatur hak akses pengguna.

Contoh:

```
GRANT SELECT ON Mahasiswa TO user1;
```

4. Transaction Control Language (TCL)

Digunakan untuk mengelola transaksi.

Contoh:

```
COMMIT;
ROLLBACK;
```

E. Struktur Dasar Perintah SQL

Struktur umum perintah SELECT sebagai perintah paling dasar dalam SQL adalah:

```
SELECT nama_kolom
FROM nama_tabel
WHERE kondisi;
```

Bagian-bagian tersebut memiliki fungsi sebagai berikut:

- SELECT menentukan kolom yang ingin ditampilkan.
- FROM menentukan tabel sumber data.
- WHERE menentukan kondisi penyaringan data.

F. SQL dalam Sistem RDBMS

SQL digunakan dalam berbagai sistem RDBMS. Meskipun terdapat perbedaan kecil dalam implementasi, secara umum sintaks SQL tetap sama karena mengikuti standar.

Beberapa RDBMS populer antara lain:
1. MySQL – Banyak digunakan untuk aplikasi web.
2. PostgreSQL – Open-source dan mendukung fitur lanjutan.
3. Oracle Database – Digunakan dalam sistem enterprise.
4. Microsoft SQL Server – Banyak digunakan dalam lingkungan bisnis berbasis Windows.

G. Keunggulan Penggunaan SQL

SQL memiliki berbagai keunggulan, antara lain:

1. Efisien dalam Pengolahan Data
SQL dapat menangani data dalam jumlah besar secara cepat.

2. Fleksibel
Dapat digunakan untuk berbagai jenis operasi database.

3. Aman
Mendukung pengaturan hak akses pengguna.

4. Terintegrasi dengan Bahasa Pemrograman
SQL dapat digunakan bersama bahasa seperti Java, Python, dan PHP.

H. Peran SQL dalam Pengembangan Sistem Informasi

Dalam pengembangan sistem informasi, SQL berperan sebagai jembatan antara aplikasi dan database. Aplikasi seperti sistem akademik, e-commerce, dan perbankan menggunakan SQL untuk menyimpan dan mengambil data dari database.

Tanpa SQL, pengelolaan data dalam sistem relasional akan menjadi sangat kompleks dan tidak efisien.

I. Tantangan dalam Penggunaan SQL

Meskipun SQL relatif mudah dipelajari, terdapat beberapa tantangan dalam penggunaannya:

1. Query yang kompleks dapat sulit dipahami.
2. Kesalahan sintaks dapat menyebabkan error.
3. Optimasi query memerlukan pemahaman indeks dan struktur tabel.
4. Penggunaan yang tidak tepat dapat menimbulkan risiko keamanan seperti SQL Injection.

Oleh karena itu, penggunaan SQL harus dilakukan dengan pemahaman yang baik mengenai struktur database.

J. Kesimpulan

Structured Query Language (SQL) adalah bahasa standar yang digunakan untuk berkomunikasi dengan sistem manajemen basis data relasional. SQL memungkinkan pengguna untuk membuat, mengelola, dan memanipulasi data secara efisien dan terstruktur.

Perintah SQL dikategorikan ke dalam DDL dan DML sebagai dasar utama, serta didukung oleh DCL dan TCL untuk pengelolaan keamanan dan transaksi. Dengan memahami dasar-dasar SQL, pengguna dapat mengelola database secara profesional dan mendukung pengembangan sistem informasi modern.

6.2 Data Definition Language (DDL)

A. Pengertian Data Definition Language (DDL)

Data Definition Language (DDL) adalah bagian dari Structured Query Language (SQL) yang digunakan untuk mendefinisikan struktur atau skema basis data. DDL berfokus pada pembuatan, pengubahan, dan penghapusan objek-objek dalam database, seperti database, tabel, indeks, view, dan constraint.

DDL tidak digunakan untuk mengelola isi data, melainkan untuk mengatur kerangka atau struktur tempat data tersebut disimpan. Oleh karena itu, DDL memiliki peran yang sangat penting dalam tahap awal perancangan dan implementasi database.

B. Fungsi Utama DDL

Secara umum, DDL memiliki beberapa fungsi utama, yaitu:

1. Membuat database baru.
2. Membuat tabel dan menentukan struktur kolom.
3. Menentukan tipe data dan domain atribut.
4. Menetapkan constraint seperti PRIMARY KEY dan FOREIGN KEY.
5. Mengubah struktur tabel yang sudah ada.
6. Menghapus objek database yang tidak diperlukan.

Melalui DDL, administrator database dapat mengontrol struktur sistem secara menyeluruh.

C. Perintah-perintah DDL

Beberapa perintah utama dalam DDL antara lain:

1. CREATE

Perintah CREATE digunakan untuk membuat objek baru dalam database, seperti database, tabel, atau indeks.

Contoh membuat database:

```
CREATE DATABASE Akademik;
```

Contoh membuat tabel:

```
CREATE TABLE Mahasiswa (
    NIM VARCHAR(10) PRIMARY KEY,
    Nama VARCHAR(100) NOT NULL,
```

```
    Tanggal_Lahir DATE,
    Program_Studi VARCHAR(50)
);
```

Perintah ini mendefinisikan struktur tabel beserta tipe data dan constraint yang berlaku.

2. ALTER

Perintah ALTER digunakan untuk mengubah struktur tabel yang sudah ada.

Contoh menambahkan kolom baru:

```
ALTER TABLE Mahasiswa
ADD Email VARCHAR(100);
```

Contoh mengubah tipe data:

```
ALTER TABLE Mahasiswa
MODIFY Nama VARCHAR(150);
```

Perintah ALTER sangat penting ketika terjadi perubahan kebutuhan sistem.

3. DROP

Perintah DROP digunakan untuk menghapus objek database secara permanen.

Contoh:

```
DROP TABLE Mahasiswa;
```

```
DROP DATABASE Akademik;
```

Penggunaan DROP harus dilakukan dengan hati-hati karena data yang dihapus tidak dapat dikembalikan.

4. TRUNCATE

Perintah TRUNCATE digunakan untuk menghapus seluruh isi tabel tanpa menghapus struktur tabel tersebut.

Contoh:

TRUNCATE TABLE Mahasiswa;

Berbeda dengan DELETE, TRUNCATE bekerja lebih cepat karena tidak mencatat setiap penghapusan baris secara individual.

D. Komponen dalam Pembuatan Tabel

Dalam perintah CREATE TABLE, terdapat beberapa komponen penting, yaitu:

1. Nama Tabel
Setiap tabel harus memiliki nama yang unik dalam satu database.

2. Kolom dan Tipe Data
Setiap kolom harus memiliki tipe data tertentu, seperti INT, VARCHAR, DATE, atau BOOLEAN.

3. Constraint
Constraint digunakan untuk menjaga integritas data, seperti:
- PRIMARY KEY
- FOREIGN KEY
- NOT NULL
- UNIQUE
- CHECK
- DEFAULT

Contoh penggunaan constraint:

```
CREATE TABLE Mata_Kuliah (
    Kode_MK VARCHAR(10) PRIMARY KEY,
    Nama_MK VARCHAR(100) NOT NULL,
```

```
    SKS INT CHECK (SKS > 0)
);
```

E. Skema (Schema) dalam DDL

DDL juga digunakan untuk mengatur skema database. Skema adalah kumpulan objek database yang dikelompokkan secara logis.

Contoh membuat schema:

```
CREATE SCHEMA Akademik;
```

Dengan penggunaan schema, pengelolaan database menjadi lebih terstruktur dan terorganisir.

F. Peran DDL dalam Perancangan Database

DDL memiliki peran penting dalam tahap desain database, antara lain:

1. Menentukan struktur tabel berdasarkan model relasional.
2. Mengimplementasikan hasil perancangan ERD.
3. Menetapkan aturan integritas sejak awal.
4. Menjamin konsistensi struktur database.

Tanpa DDL, database tidak memiliki struktur yang jelas dan tidak dapat digunakan untuk menyimpan data.

G. Perbedaan DDL dan DML

DDL berfokus pada struktur database, sedangkan DML berfokus pada isi data. Perubahan yang dilakukan oleh DDL bersifat struktural dan biasanya memengaruhi keseluruhan sistem.

Contohnya:
- CREATE dan ALTER termasuk DDL.
- INSERT dan UPDATE termasuk DML.

H. Dampak Penggunaan DDL terhadap Sistem

Perintah DDL biasanya bersifat auto-commit dalam banyak sistem RDBMS, artinya perubahan langsung disimpan secara permanen. Oleh karena itu, penggunaannya harus dilakukan dengan perencanaan yang matang.

Kesalahan dalam penggunaan DDL dapat menyebabkan:
1. Hilangnya data penting.
2. Perubahan struktur yang tidak diinginkan.
3. Gangguan pada aplikasi yang terhubung dengan database.

I. Contoh Studi Kasus Implementasi DDL

Dalam sistem akademik, DDL digunakan untuk membuat tabel Mahasiswa, Dosen, Mata_Kuliah, dan KRS. Setiap tabel memiliki primary key dan foreign key untuk menjaga integritas referensial.

Contoh relasi menggunakan DDL:

```
CREATE TABLE KRS (
    NIM VARCHAR(10),
    Kode_MK VARCHAR(10),
    Nilai CHAR(2),
    PRIMARY KEY (NIM, Kode_MK),
    FOREIGN KEY (NIM) REFERENCES Mahasiswa(NIM),
    FOREIGN KEY (Kode_MK) REFERENCES Mata_Kuliah(Kode_MK)
```

Dengan struktur ini, sistem mampu menjaga konsistensi hubungan antar tabel.

J. Keunggulan Penggunaan DDL

1. Membantu membangun struktur database yang terorganisir.
2. Mendukung integritas data melalui constraint.

3. Mempermudah pengembangan sistem.
4. Memberikan kontrol penuh terhadap skema database.

K. Kesimpulan

Data Definition Language (DDL) merupakan bagian penting dari SQL yang digunakan untuk mendefinisikan struktur atau skema basis data. Perintah seperti CREATE, ALTER, DROP, dan TRUNCATE memungkinkan pengelolaan struktur database secara sistematis.

Dengan penggunaan DDL yang tepat, database dapat dirancang secara terstruktur, konsisten, dan siap mendukung berbagai operasi manipulasi data dalam sistem informasi modern.

6.3 Data Manipulation Language (DML): INSERT, SELECT, UPDATE, DELETE

A. Pengertian Data Manipulation Language (DML)

Data Manipulation Language (DML) adalah bagian dari Structured Query Language (SQL) yang digunakan untuk mengelola dan memanipulasi data yang tersimpan di dalam tabel pada sistem manajemen basis data relasional (RDBMS). Jika Data Definition Language (DDL) berfungsi untuk membentuk struktur database, maka DML berfungsi untuk mengisi, mengubah, mengambil, dan menghapus data dari struktur tersebut.

Perintah-perintah DML merupakan perintah yang paling sering digunakan dalam operasi sehari-hari sistem informasi, karena hampir seluruh aktivitas aplikasi seperti sistem akademik, perbankan, e-commerce, dan administrasi pemerintahan melibatkan manipulasi data.

Perintah utama dalam DML meliputi:
1. INSERT
2. SELECT

3. UPDATE
4. DELETE

B. Perintah INSERT

Perintah INSERT digunakan untuk menambahkan data baru ke dalam tabel. Data yang dimasukkan harus sesuai dengan struktur tabel, termasuk tipe data dan constraint yang telah ditentukan.

1. Sintaks Dasar INSERT

```
INSERT INTO nama_tabel (nama_kolom1, nama_kolom2, ...)
VALUES (nilai1, nilai2, ...);
```

2. Contoh Penggunaan

Misalnya terdapat tabel Mahasiswa:

```
INSERT INTO Mahasiswa (NIM, Nama, Program_Studi)
VALUES ('22001', 'Andi', 'Informatika');
```

Perintah ini akan menambahkan satu baris baru ke dalam tabel Mahasiswa.

3. INSERT Tanpa Menyebutkan Kolom

Jika seluruh kolom diisi sesuai urutan tabel:

```
INSERT INTO Mahasiswa
VALUES ('22002', 'Budi', 'Sistem Informasi');
```

Namun cara ini kurang direkomendasikan karena dapat menimbulkan kesalahan jika struktur tabel berubah.

4. INSERT Multiple Rows

```
INSERT INTO Mahasiswa (NIM, Nama, Program_Studi)
VALUES
```

('22003', 'Citra', 'Informatika'),
('22004', 'Dina', 'Sistem Informasi');

5. INSERT dari Tabel Lain

INSERT INTO Mahasiswa_Arsip
SELECT * FROM Mahasiswa;

Perintah ini menyalin data dari satu tabel ke tabel lain.

C. Perintah SELECT

SELECT adalah perintah DML yang digunakan untuk mengambil atau menampilkan data dari tabel. SELECT merupakan perintah yang paling fleksibel dan paling sering digunakan dalam SQL.

1. Sintaks Dasar SELECT

SELECT nama_kolom
FROM nama_tabel;

2. Menampilkan Semua Kolom

SELECT * FROM Mahasiswa;

3. Menggunakan WHERE untuk Penyaringan

SELECT Nama
FROM Mahasiswa
WHERE Program_Studi = 'Informatika';

4. Menggunakan ORDER BY

SELECT * FROM Mahasiswa
ORDER BY Nama ASC;

5. Menggunakan GROUP BY

```
SELECT Program_Studi, COUNT(*)
FROM Mahasiswa
GROUP BY Program_Studi;
```

6. Menggunakan JOIN

```
SELECT Mahasiswa.Nama, Mata_Kuliah.Nama_MK
FROM Mahasiswa
JOIN KRS ON Mahasiswa.NIM = KRS.NIM
JOIN Mata_Kuliah ON KRS.Kode_MK = Mata_Kuliah.Kode_MK;
```

SELECT memungkinkan analisis data secara kompleks dan mendalam.

D. Perintah UPDATE

Perintah UPDATE digunakan untuk memperbarui atau mengubah data yang sudah ada dalam tabel.

1. Sintaks Dasar UPDATE

```
UPDATE nama_tabel
SET nama_kolom = nilai_baru
WHERE kondisi;
```

2. Contoh UPDATE

```
UPDATE Mahasiswa
SET Program_Studi = 'Teknik Informatika'
WHERE NIM = '22001';
```

3. UPDATE Tanpa WHERE

```
UPDATE Mahasiswa
SET Program_Studi = 'Informatika';
```

Perintah ini akan mengubah seluruh data dalam kolom tersebut, sehingga harus digunakan dengan sangat hati-hati.

E. Perintah DELETE

DELETE digunakan untuk menghapus data dari tabel.

1. Sintaks Dasar DELETE

```
DELETE FROM nama_tabel
WHERE kondisi;
```

2. Contoh DELETE

```
DELETE FROM Mahasiswa
WHERE NIM = '22004';
```

3. DELETE Tanpa WHERE

```
DELETE FROM Mahasiswa;
```

Perintah ini akan menghapus seluruh isi tabel, sehingga sangat berisiko jika tidak digunakan dengan benar.

F. Perbedaan DELETE dan TRUNCATE

DELETE termasuk DML dan dapat menggunakan WHERE untuk menghapus data tertentu. DELETE juga mencatat setiap penghapusan dalam log transaksi.

TRUNCATE termasuk DDL dan menghapus seluruh data tanpa mencatat setiap baris secara individual, sehingga lebih cepat.

G. Transaksi dalam DML

Operasi DML biasanya dapat dikontrol menggunakan perintah transaksi seperti:

```
BEGIN;
COMMIT;
ROLLBACK;
```

COMMIT menyimpan perubahan secara permanen, sedangkan ROLLBACK membatalkan perubahan sebelum disimpan.

H. Dampak DML terhadap Integritas Data

Operasi DML harus memperhatikan integritas data. Misalnya:

1. INSERT harus memenuhi constraint seperti PRIMARY KEY dan FOREIGN KEY.
2. UPDATE tidak boleh melanggar integritas referensial.
3. DELETE harus mempertimbangkan relasi antar tabel.

Jika tidak dikelola dengan baik, manipulasi data dapat menyebabkan inkonsistensi.

I. Studi Kasus Sederhana

Dalam sistem akademik:

INSERT digunakan untuk menambahkan mahasiswa baru.
SELECT digunakan untuk menampilkan daftar mahasiswa.
UPDATE digunakan untuk mengubah data mahasiswa.
DELETE digunakan untuk menghapus mahasiswa yang sudah lulus.

Semua operasi tersebut dilakukan melalui DML.

J. Keunggulan DML

1. Memungkinkan pengelolaan data secara fleksibel.
2. Mendukung analisis data melalui SELECT.
3. Mendukung pemeliharaan data secara dinamis.
4. Terintegrasi dengan sistem transaksi.

K. Kesimpulan

Data Manipulation Language (DML) merupakan bagian penting dari SQL yang digunakan untuk mengelola isi data dalam database. Perintah INSERT,

SELECT, UPDATE, dan DELETE memungkinkan pengguna untuk menambahkan, mengambil, mengubah, dan menghapus data secara sistematis.

Pemahaman yang baik terhadap DML sangat penting dalam pengelolaan sistem basis data karena hampir seluruh aktivitas operasional sistem informasi melibatkan manipulasi data.

SOAL LATIHAN BAB 6

1. **(DDL)** Buatlah sebuah tabel bernama Dosen dengan ketentuan kolom: NIDN (Varchar 15, Primary Key), Nama_Dosen (Varchar 100, Not Null), dan Fakultas (Varchar 50).
2. **(DDL)** Ubah struktur tabel Dosen dengan menambahkan kolom No_Telepon (Varchar 15).
3. **(DML)** Masukkan 3 baris data fiktif ke dalam tabel Dosen menggunakan perintah INSERT.
4. **(DML)** Tampilkan hanya Nama_Dosen dan Fakultas dari tabel Dosen yang berasal dari Fakultas "Teknologi dan Ilmu Komputer".
5. **(DML)** Ubah data Fakultas menjadi "Ilmu Komputer dan Teknologi Informasi" untuk Dosen yang memiliki NIDN tertentu.

BAB 7

SQL LANJUTAN

7.1 Agregasi Data (SUM, AVG, COUNT, dll)

A. Pengertian Agregasi Data dalam SQL

Agregasi data dalam SQL adalah proses pengolahan sekumpulan data untuk menghasilkan satu nilai ringkasan (summary value). Fungsi agregasi digunakan untuk melakukan perhitungan terhadap sekumpulan baris dalam suatu tabel, seperti menghitung jumlah, rata-rata, nilai maksimum, nilai minimum, dan jumlah data.

Fungsi agregasi sangat penting dalam analisis data karena memungkinkan pengguna memperoleh informasi statistik secara cepat dan efisien tanpa harus menghitung secara manual. Dalam sistem informasi seperti perbankan, akademik, e-commerce, dan perusahaan, fungsi agregasi sering digunakan untuk laporan dan pengambilan keputusan.

B. Jenis-jenis Fungsi Agregasi

Beberapa fungsi agregasi utama dalam SQL antara lain:

1. COUNT()
COUNT digunakan untuk menghitung jumlah baris dalam suatu tabel atau hasil query.

Contoh:
SELECT COUNT(*) FROM Mahasiswa;

Perintah ini menghitung jumlah seluruh mahasiswa dalam tabel.

COUNT juga dapat digunakan pada kolom tertentu:
SELECT COUNT(NIM) FROM Mahasiswa;

COUNT tidak menghitung nilai NULL jika menggunakan nama kolom.

2. SUM()
SUM digunakan untuk menjumlahkan nilai numerik dalam suatu kolom.

Contoh:
SELECT SUM(SKS) FROM Mata_Kuliah;

Perintah ini menghitung total SKS dari seluruh mata kuliah.

3. AVG()
AVG digunakan untuk menghitung rata-rata nilai numerik.

Contoh:
SELECT AVG(Nilai) FROM KRS;

Perintah ini menghitung rata-rata nilai mahasiswa.

4. MIN()
MIN digunakan untuk mencari nilai terkecil dalam suatu kolom.

Contoh:
SELECT MIN(Nilai) FROM KRS;

5. MAX()
MAX digunakan untuk mencari nilai terbesar dalam suatu kolom.

Contoh:
SELECT MAX(Nilai) FROM KRS;

C. Penggunaan GROUP BY dalam Agregasi

Fungsi agregasi sering digunakan bersama klausa GROUP BY untuk mengelompokkan data berdasarkan kolom tertentu.

Contoh:
SELECT Program_Studi, COUNT(*)
FROM Mahasiswa
GROUP BY Program_Studi;

Perintah ini menghitung jumlah mahasiswa per program studi.

Contoh lain:
SELECT Program_Studi, AVG(Nilai)
FROM Mahasiswa

```
JOIN KRS ON Mahasiswa.NIM = KRS.NIM
GROUP BY Program_Studi;
```

Perintah ini menghitung rata-rata nilai berdasarkan program studi.

D. Penggunaan HAVING dalam Agregasi

HAVING digunakan untuk memfilter hasil agregasi yang sudah dikelompokkan.

Contoh:

```
SELECT Program_Studi, COUNT(*)
FROM Mahasiswa
GROUP BY Program_Studi
HAVING COUNT(*) > 50;
```

Perintah ini hanya menampilkan program studi yang memiliki lebih dari 50 mahasiswa.

Perbedaan WHERE dan HAVING:
- WHERE digunakan sebelum proses pengelompokan.
- HAVING digunakan setelah proses GROUP BY.

E. Kombinasi Agregasi dengan JOIN

Fungsi agregasi dapat dikombinasikan dengan JOIN untuk analisis lintas tabel.

Contoh:

```
SELECT Dosen.Nama, COUNT(Mata_Kuliah.Kode_MK)
FROM Dosen
JOIN Mata_Kuliah ON Dosen.Kode_Dosen = Mata_Kuliah.Kode_Dosen
GROUP BY Dosen.Nama;
```

Perintah ini menghitung jumlah mata kuliah yang diajar oleh setiap dosen.

F. Penanganan NULL dalam Fungsi Agregasi

Fungsi agregasi umumnya mengabaikan nilai NULL, kecuali COUNT(*). Oleh karena itu, penting untuk memastikan data yang dianalisis tidak mengandung nilai NULL yang tidak diinginkan.

Contoh penggunaan COALESCE untuk mengatasi NULL:
SELECT AVG(COALESCE(Nilai, 0)) FROM KRS;

G. Studi Kasus Sistem Akademik

Misalnya dalam sistem akademik:

1. Menghitung jumlah mahasiswa:
SELECT COUNT(*) FROM Mahasiswa;

2. Menghitung total SKS yang diambil mahasiswa tertentu:
SELECT SUM(SKS)
FROM KRS
JOIN Mata_Kuliah ON KRS.Kode_MK = Mata_Kuliah.Kode_MK
WHERE NIM = '22001';

3. Menghitung nilai tertinggi dan terendah:
SELECT MAX(Nilai), MIN(Nilai) FROM KRS;

4. Menghitung rata-rata nilai per mata kuliah:
SELECT Kode_MK, AVG(Nilai)
FROM KRS
GROUP BY Kode_MK;

H. Manfaat Agregasi dalam Pengambilan Keputusan

Fungsi agregasi membantu dalam:

1. Penyusunan laporan keuangan.
2. Analisis performa mahasiswa.
3. Evaluasi penjualan produk.

4. Analisis tren bisnis.
5. Pengukuran kinerja organisasi.

Dengan agregasi, data mentah dapat diubah menjadi informasi yang bermakna.

I. Kinerja dan Optimasi Agregasi

Penggunaan agregasi pada tabel besar dapat mempengaruhi performa sistem. Oleh karena itu, penting untuk:

1. Menggunakan indeks pada kolom yang sering digunakan dalam GROUP BY.
2. Menghindari SELECT * jika tidak diperlukan.
3. Menggunakan filter WHERE sebelum GROUP BY untuk memperkecil dataset.

J. Kesimpulan

Agregasi data dalam SQL merupakan teknik penting untuk menghasilkan informasi ringkasan dari sekumpulan data. Fungsi seperti SUM, AVG, COUNT, MIN, dan MAX memungkinkan analisis data secara efisien.

Dengan kombinasi GROUP BY dan HAVING, SQL mampu melakukan analisis data yang kompleks dan mendalam. Pemahaman yang baik terhadap fungsi agregasi sangat penting dalam SQL lanjutan untuk mendukung pengambilan keputusan berbasis data.

7.2 Pengelompokan (GROUP BY dan HAVING)

A. Pengantar Pengelompokan Data dalam SQL

Dalam pengolahan data menggunakan SQL, sering kali diperlukan analisis terhadap sekumpulan data yang memiliki karakteristik tertentu. Untuk tujuan tersebut, SQL menyediakan mekanisme pengelompokan data melalui klausa GROUP BY dan HAVING.

Pengelompokan memungkinkan data yang memiliki nilai sama pada kolom tertentu dikumpulkan dalam satu kelompok, sehingga dapat dilakukan

perhitungan agregasi seperti COUNT, SUM, AVG, MIN, dan MAX terhadap masing-masing kelompok tersebut.

Fitur ini sangat penting dalam pembuatan laporan, analisis statistik, serta pengambilan keputusan berbasis data.

B. Konsep Dasar GROUP BY

Klausa GROUP BY digunakan untuk mengelompokkan baris-baris data berdasarkan satu atau lebih kolom tertentu.

Sintaks dasar:

```
SELECT nama_kolom, fungsi_agregasi(nama_kolom)
FROM nama_tabel
GROUP BY nama_kolom;
```

Ketika GROUP BY digunakan, setiap kolom dalam SELECT harus:
1. Termasuk dalam GROUP BY, atau
2. Menggunakan fungsi agregasi.

Contoh sederhana:

```
SELECT Program_Studi, COUNT(*)
FROM Mahasiswa
GROUP BY Program_Studi;
```

Perintah ini menghitung jumlah mahasiswa untuk setiap program studi.

C. Mekanisme Kerja GROUP BY

Proses kerja GROUP BY dapat dijelaskan sebagai berikut:

1. SQL membaca seluruh data dari tabel.
2. Data dikelompokkan berdasarkan nilai kolom yang ditentukan.
3. Fungsi agregasi diterapkan pada setiap kelompok.
4. Hasil akhir ditampilkan dalam bentuk ringkasan per kelompok.

Contoh lain:

```
SELECT Kode_MK, AVG(Nilai)
FROM KRS
GROUP BY Kode_MK;
```

Query ini menghitung rata-rata nilai untuk setiap mata kuliah.

D. GROUP BY dengan Lebih dari Satu Kolom

GROUP BY dapat digunakan untuk mengelompokkan berdasarkan lebih dari satu kolom.

Contoh:

```
SELECT Program_Studi, Angkatan, COUNT(*)
FROM Mahasiswa
GROUP BY Program_Studi, Angkatan;
```

Query ini menghasilkan jumlah mahasiswa berdasarkan kombinasi program studi dan angkatan.

E. Penggunaan HAVING

HAVING digunakan untuk memfilter hasil pengelompokan yang sudah diproses oleh GROUP BY. Jika WHERE memfilter sebelum pengelompokan, maka HAVING memfilter setelah pengelompokan.

Sintaks dasar:

```
SELECT kolom, fungsi_agregasi(kolom)
FROM tabel
GROUP BY kolom
HAVING kondisi;
```

Contoh:

```
SELECT Program_Studi, COUNT(*)
FROM Mahasiswa
GROUP BY Program_Studi
HAVING COUNT(*) > 50;
```

Query ini hanya menampilkan program studi dengan jumlah mahasiswa lebih dari 50.

F. Perbedaan WHERE dan HAVING

Perbedaan utama antara WHERE dan HAVING:

WHERE:
- Digunakan sebelum GROUP BY.
- Tidak dapat menggunakan fungsi agregasi.

HAVING:
- Digunakan setelah GROUP BY.
- Dapat menggunakan fungsi agregasi.

Contoh kombinasi:

```
SELECT Program_Studi, AVG(Nilai)
FROM Mahasiswa
JOIN KRS ON Mahasiswa.NIM = KRS.NIM
WHERE Angkatan = 2022
GROUP BY Program_Studi
HAVING AVG(Nilai) > 75;
```

Query ini:
1. Memfilter mahasiswa angkatan 2022.
2. Mengelompokkan berdasarkan program studi.
3. Menampilkan hanya program studi dengan rata-rata nilai di atas 75.

G. Penggunaan GROUP BY dengan JOIN

GROUP BY sering dikombinasikan dengan JOIN untuk analisis lintas tabel.

Contoh:

```
SELECT Dosen.Nama, COUNT(Mata_Kuliah.Kode_MK)
FROM Dosen
JOIN Mata_Kuliah ON Dosen.Kode_Dosen = Mata_Kuliah.Kode_Dosen
GROUP BY Dosen.Nama;
```

Query ini menghitung jumlah mata kuliah yang diajar oleh setiap dosen.

H. Studi Kasus Sistem Penjualan

Misalkan terdapat tabel Penjualan:

ID_Penjualan | ID_Produk | Jumlah | Harga

Untuk menghitung total penjualan per produk:

```
SELECT ID_Produk, SUM(Jumlah * Harga) AS Total_Penjualan
FROM Penjualan
GROUP BY ID_Produk;
```

Untuk menampilkan hanya produk dengan total penjualan di atas 1.000.000:

```
SELECT ID_Produk, SUM(Jumlah * Harga) AS Total_Penjualan
FROM Penjualan
GROUP BY ID_Produk
HAVING SUM(Jumlah * Harga) > 1000000;
```

I. Kesalahan Umum dalam Penggunaan GROUP BY

Beberapa kesalahan umum:

1. Memasukkan kolom dalam SELECT yang tidak termasuk dalam GROUP BY dan tidak menggunakan fungsi agregasi.

2. Menggunakan HAVING tanpa GROUP BY.
3. Tidak memahami urutan eksekusi SQL.

Urutan eksekusi SQL secara umum:
1. FROM
2. WHERE
3. GROUP BY
4. HAVING
5. SELECT
6. ORDER BY

Memahami urutan ini penting agar query berjalan sesuai harapan.

J. Optimasi Query dengan GROUP BY

Pada tabel besar, GROUP BY dapat mempengaruhi performa. Untuk optimasi:

1. Gunakan indeks pada kolom yang sering digunakan dalam GROUP BY.
2. Gunakan WHERE untuk memperkecil jumlah data sebelum pengelompokan.
3. Hindari penggunaan fungsi yang tidak perlu dalam SELECT.

K. Manfaat Pengelompokan Data

Pengelompokan data membantu dalam:

1. Penyusunan laporan statistik.
2. Analisis tren data.
3. Evaluasi performa.
4. Pengambilan keputusan berbasis data.
5. Penyederhanaan data mentah menjadi informasi ringkas.

L. Kesimpulan

GROUP BY dan HAVING merupakan fitur penting dalam SQL lanjutan yang memungkinkan pengelompokan serta penyaringan hasil agregasi data. GROUP

BY digunakan untuk mengelompokkan data berdasarkan kolom tertentu, sedangkan HAVING digunakan untuk memfilter hasil pengelompokan tersebut.

Pemahaman yang baik terhadap kedua klausa ini sangat penting dalam analisis data karena memungkinkan pengguna menghasilkan laporan yang terstruktur, akurat, dan informatif.

7.3 Penggabungan Tabel (INNER JOIN, LEFT JOIN, RIGHT JOIN)

A. Pengantar Penggabungan Tabel (JOIN)

Dalam sistem basis data relasional, data umumnya disimpan dalam beberapa tabel yang saling berhubungan. Pemisahan tabel dilakukan untuk menghindari redundansi dan menjaga integritas data melalui proses normalisasi. Namun, ketika data tersebut perlu ditampilkan secara terpadu, maka diperlukan mekanisme untuk menggabungkan tabel-tabel tersebut. Mekanisme ini dikenal dengan istilah JOIN.

JOIN digunakan untuk mengkombinasikan baris dari dua atau lebih tabel berdasarkan kolom yang memiliki hubungan (biasanya melalui primary key dan foreign key). Dengan JOIN, pengguna dapat menampilkan data yang tersebar di beberapa tabel menjadi satu hasil query yang utuh dan informatif.

B. Konsep Dasar JOIN

JOIN bekerja dengan cara mencocokkan nilai pada kolom tertentu antara dua tabel. Kolom yang digunakan sebagai penghubung biasanya adalah:

1. Primary Key pada tabel utama.
2. Foreign Key pada tabel terkait.

Sintaks dasar JOIN:

```
SELECT kolom1, kolom2
FROM tabel1
```

JOIN tabel2
ON tabel1.kolom = tabel2.kolom;

Klausa ON menentukan kondisi pencocokan antara kedua tabel.

C. INNER JOIN

INNER JOIN adalah jenis JOIN yang hanya menampilkan data yang memiliki pasangan (match) di kedua tabel.

Sintaks:

SELECT kolom
FROM tabel1
INNER JOIN tabel2
ON kondisi;

Contoh:

SELECT Mahasiswa.Nama, Mata_Kuliah.Nama_MK
FROM Mahasiswa
INNER JOIN KRS ON Mahasiswa.NIM = KRS.NIM
INNER JOIN Mata_Kuliah ON KRS.Kode_MK = Mata_Kuliah.Kode_MK;

Penjelasan:
Query ini hanya menampilkan mahasiswa yang memiliki data pada tabel KRS dan mata kuliah yang sesuai. Jika tidak ada pasangan, data tidak akan ditampilkan.

Karakteristik INNER JOIN:
1. Menampilkan data yang cocok di kedua tabel.
2. Menghilangkan data yang tidak memiliki pasangan.
3. Digunakan untuk menampilkan data relasional yang lengkap.

D. LEFT JOIN (LEFT OUTER JOIN)

LEFT JOIN menampilkan semua data dari tabel sebelah kiri (tabel pertama) dan data yang cocok dari tabel sebelah kanan. Jika tidak ada pasangan, maka kolom dari tabel kanan akan bernilai NULL.

Sintaks:

```
SELECT kolom
FROM tabel1
LEFT JOIN tabel2
ON kondisi;
```

Contoh:

```
SELECT Mahasiswa.Nama, KRS.Kode_MK
FROM Mahasiswa
LEFT JOIN KRS ON Mahasiswa.NIM = KRS.NIM;
```

Penjelasan:
Query ini menampilkan seluruh mahasiswa, termasuk yang belum mengambil mata kuliah. Jika mahasiswa belum memiliki data di tabel KRS, maka kolom Kode_MK akan bernilai NULL.

Karakteristik LEFT JOIN:
1. Semua data dari tabel kiri ditampilkan.
2. Data dari tabel kanan hanya ditampilkan jika cocok.
3. Data yang tidak cocok di tabel kanan bernilai NULL.

LEFT JOIN sering digunakan untuk melihat data yang belum memiliki relasi.

E. RIGHT JOIN (RIGHT OUTER JOIN)

RIGHT JOIN adalah kebalikan dari LEFT JOIN. RIGHT JOIN menampilkan semua data dari tabel sebelah kanan dan data yang cocok dari tabel sebelah kiri.

Sintaks:

```
SELECT kolom
FROM tabel1
RIGHT JOIN tabel2
ON kondisi;
```

Contoh:

```
SELECT Mahasiswa.Nama, KRS.Kode_MK
FROM Mahasiswa
RIGHT JOIN KRS ON Mahasiswa.NIM = KRS.NIM;
```

Penjelasan:
Query ini menampilkan seluruh data dari tabel KRS, termasuk jika ada data KRS yang tidak memiliki pasangan di tabel Mahasiswa (meskipun dalam praktik integritas referensial biasanya hal ini tidak terjadi).

Karakteristik RIGHT JOIN:
1. Semua data dari tabel kanan ditampilkan.
2. Data dari tabel kiri hanya ditampilkan jika cocok.
3. Data yang tidak cocok di tabel kiri bernilai NULL.

F. Perbandingan INNER, LEFT, dan RIGHT JOIN

INNER JOIN:
- Menampilkan hanya data yang memiliki pasangan di kedua tabel.

LEFT JOIN:
- Menampilkan seluruh data dari tabel kiri dan pasangan dari tabel kanan.

RIGHT JOIN:
- Menampilkan seluruh data dari tabel kanan dan pasangan dari tabel kiri.

Pemilihan jenis JOIN bergantung pada kebutuhan analisis data.

G. Studi Kasus Sistem Akademik

Tabel Mahasiswa:
NIM | Nama

Tabel KRS:
NIM | Kode_MK

Tabel Mata_Kuliah:
Kode_MK | Nama_MK

1. INNER JOIN digunakan untuk melihat mahasiswa yang mengambil mata kuliah.
2. LEFT JOIN digunakan untuk melihat seluruh mahasiswa termasuk yang belum mengambil mata kuliah.
3. RIGHT JOIN digunakan untuk melihat seluruh data KRS beserta mahasiswa yang terkait.

H. JOIN dengan Lebih dari Dua Tabel

JOIN dapat digunakan untuk menggabungkan lebih dari dua tabel.

Contoh:

```
SELECT Mahasiswa.Nama, Mata_Kuliah.Nama_MK, Dosen.Nama
FROM Mahasiswa
INNER JOIN KRS ON Mahasiswa.NIM = KRS.NIM
INNER JOIN Mata_Kuliah ON KRS.Kode_MK = Mata_Kuliah.Kode_MK
INNER JOIN Dosen ON Mata_Kuliah.Kode_Dosen = Dosen.Kode_Dosen;
```

Query ini menampilkan nama mahasiswa, mata kuliah, dan dosen pengampu dalam satu hasil.

I. Pentingnya JOIN dalam Analisis Data

JOIN memungkinkan:
1. Integrasi data lintas tabel.
2. Penyusunan laporan kompleks.

3. Analisis hubungan antar entitas.
4. Penyajian informasi terpadu.

Tanpa JOIN, database relasional tidak dapat menampilkan hubungan antar tabel secara efektif.

J. Optimasi JOIN

JOIN pada tabel besar dapat mempengaruhi performa. Untuk optimasi:

1. Gunakan indeks pada kolom yang digunakan dalam ON.
2. Hindari SELECT * jika tidak diperlukan.
3. Gunakan filter WHERE untuk membatasi data.
4. Pastikan struktur relasi telah dinormalisasi dengan baik.

K. Kesimpulan

Penggabungan tabel melalui INNER JOIN, LEFT JOIN, dan RIGHT JOIN merupakan fitur penting dalam SQL lanjutan. INNER JOIN menampilkan data yang memiliki pasangan di kedua tabel, LEFT JOIN menampilkan seluruh data dari tabel kiri, dan RIGHT JOIN menampilkan seluruh data dari tabel kanan.

Pemahaman yang baik mengenai JOIN sangat penting dalam pengolahan data relasional karena memungkinkan integrasi data dari berbagai tabel menjadi satu kesatuan informasi yang utuh dan bermakna.