SISTEM BASIS DATA

J Prayoga, Sinar Sinurat, Andy Rachman, Irmawati Carolina,
Yuwan Jumaryadi, Frieyadie, Andi Irmayana, Adi Supriyatna,
Pasnur, Budanis Dwi Meilani, Munawarah

GME
GRAHA MITRA EDUKASI

Sistem Basis Data

J Prayoga, Sinar Sinurat, Andy Rachman, Irmawati Carolina,
Yuwan Jumaryadi, Frieyadie, Andi Irmayana, Adi Supriyatna,
Pasnur, Budanis Dwi Meilani, Munawarah

GRAHA MITRA EDUKASI

Sistem Basis Data

J Prayoga, Sinar Sinurat, Andy Rachman, Irmawati Carolina, Yuwan Jumaryadi, Frieyadie, Andi Irmayana, Adi Supriyatna, Pasnur, Budanis Dwi Meilani, Munawarah

Editor : Muhammad Syahrizal

Layout : Sarwandi

Desain Cover : Sugi Hartono

Diterbitkan oleh Penerbit CV. Graha Mitra Edukasi

Komplek Senda Residence Jl. Payanibung Ujung D Dalu Sepuluh-B Tanjung Morawa, Kab. Deli Serdang Sumatera Utara

Distributor Tunggal:

CV. Graha Mitra Edukasi Komplek Senda Residence Jl. Payanibung Ujung D Dalu Sepuluh-B Tanjung Morawa Kab. Deli Serdang Sumatera Utara

Sistem Pendukung Keputusan

CV. Graha Mitra Edukasi, 2023
; 18 x 21 cm
ISBN: 978-623-09-4150-4
Cetakan Pertama, Juni 2023

Kata Pengantar

Buku ini hadir untuk memperkenalkan Anda pada konsep, prinsip, dan praktik yang terkait dengan manajemen sistem basis data. Sistem basis data merupakan elemen penting dalam dunia teknologi informasi, yang telah mengubah cara kita menyimpan, mengelola, dan memanfaatkan data secara efisien. Dalam buku ini, kami akan membahas berbagai aspek yang terkait dengan sistem basis data, mulai dari dasar-dasar pengertian hingga topik-topik yang lebih mendalam seperti model data, desain basis data, bahasa query, pemrosesan transaksi, optimisasi kinerja, dan keamanan basis data. Kami juga akan memperkenalkan beberapa teknologi dan alat yang digunakan dalam pengembangan dan pengelolaan sistem basis data.

Tidak diragukan lagi, sistem basis data memainkan peran krusial dalam mendukung kegiatan bisnis, penelitian, dan kehidupan sehari-hari kita. Kemampuan untuk menyimpan dan mengakses data dengan cepat dan akurat adalah fondasi dari transformasi digital yang sedang kita alami saat ini. Dalam buku ini, kami berharap dapat memberikan pemahaman yang kokoh tentang sistem basis data serta memberikan wawasan yang berguna bagi pembaca dalam menghadapi tantangan di dunia yang semakin terhubung.

Buku ini ditujukan untuk mahasiswa, profesional teknologi informasi, dan siapa pun yang ingin memperdalam pengetahuannya tentang sistem basis data. Kami berharap buku ini dapat menjadi panduan yang komprehensif dan bermanfaat bagi pembaca dalam memahami konsep dan prinsip-prinsip dasar, serta memberikan landasan yang kuat untuk mengembangkan dan mengelola sistem basis data yang efektif dan handal.

Medan, April 2023

Penulis

Daftar Isi

Bab 1
Mengenal *Database*

1.1 Database

Database adalah kumpulan terstruktur dari data yang disimpan secara elektronik. Ini adalah sistem yang digunakan untuk menyimpan, mengelola, dan mengambil informasi secara efisien. *Database* sering digunakan dalam berbagai konteks, mulai dari bisnis dan organisasi hingga aplikasi web dan perangkat lunak.

Database terdiri dari tabel yang terdiri dari baris dan kolom. Setiap baris dalam tabel mewakili satu entitas atau objek, sedangkan setiap kolom mewakili atribut atau karakteristik dari entitas tersebut. Misalnya, dalam sebuah *database* pelanggan, setiap baris mungkin mewakili satu pelanggan, dan kolom-kolomnya bisa berisi nama, alamat, nomor telepon, dan sebagainya.

Salah satu keuntungan utama penggunaan *database* adalah kemampuannya untuk menyimpan dan mengelola jumlah data yang besar dan kompleks. *Database* dapat dirancang untuk mendukung berbagai operasi, seperti pencarian, penyaringan, pengurutan, dan penggabungan data. Dengan menggunakan bahasa kueri seperti SQL *(Structured Query Language)*, pengguna dapat mengambil data yang spesifik dari *database* dengan cepat dan mudah.

Selain itu, *database* juga memungkinkan multipleks pengguna, artinya beberapa pengguna dapat mengakses dan memanipulasi data secara bersamaan. Ini menjadikannya alat yang sangat penting dalam lingkungan bisnis di mana banyak pengguna membutuhkan akses ke informasi yang sama.

Database juga menyediakan fitur keamanan yang penting untuk melindungi data dari akses yang tidak sah. Penggunaan hak akses dan izin dapat ditetapkan untuk mengontrol siapa yang dapat melihat, mengubah, atau menghapus data dalam *database.* Dalam beberapa tahun terakhir, perkembangan teknologi *database* telah melihat pergeseran menuju *database* yang lebih canggih dan inovatif, seperti *database NoSQL* yang dirancang untuk mengelola data yang tidak terstruktur atau semi-terstruktur dengan skala yang besar. Selain itu, *database* terdistribusi dan *database* berbasis *cloud* juga semakin populer, memungkinkan skalabilitas dan ketersediaan data yang lebih baik.

Secara keseluruhan, *database* adalah komponen inti dalam pengelolaan data modern. Dalam dunia yang semakin terhubung *dan data-driven*, *database* memainkan peran krusial dalam menyimpan, mengelola, dan menyediakan akses terhadap informasi yang berharga bagi organisasi dan pengguna.

1.2 Komponen Sistem Basis Data

Sistem basis data terdiri dari beberapa komponen utama yang bekerja bersama untuk mengelola dan menyediakan akses terhadap data. Berikut adalah beberapa komponen penting dalam sistem basis data:

1. **Data:** Data merupakan informasi yang disimpan dalam basis data. Data dapat berupa teks, angka, gambar, suara, atau jenis data lainnya. Data ini mewakili fakta atau entitas yang ingin kita simpan dan kelola.
2. **Struktur Data:** Struktur data merujuk pada cara data diatur dan disimpan dalam basis data. Ini melibatkan pemodelan entitas dan hubungan antara entitas tersebut. Struktur data yang umum digunakan dalam basis data relasional adalah tabel, kolom, dan baris.
3. **DBMS (*Database Management System):*** DBMS adalah perangkat lunak yang digunakan untuk mengelola basis data. Ini menyediakan antarmuka antara pengguna dan basis data, serta menyediakan alat untuk membuat, mengedit, dan mengelola data. Contoh DBMS yang populer adalah MySQL, Oracle, Microsoft SQL Server, dan PostgreSQL.
4. **Bahasa Kueri:** Bahasa kueri digunakan untuk mengambil dan memanipulasi data dalam basis data. SQL (*Structured Query Language*) adalah bahasa kueri yang paling umum digunakan dalam sistem basis data relasional. Bahasa kueri memungkinkan pengguna untuk menyusun perintah untuk mencari, menyaring, mengurutkan, dan menggabungkan data dalam basis data.
5. **Metode Penyimpanan:** Metode penyimpanan mengacu pada cara data disimpan dalam media penyimpanan fisik, seperti hard disk atau memori. Beberapa metode penyimpanan yang umum digunakan adalah penyimpanan berkas, penyimpanan berbasis blok, dan penyimpanan berbasis halaman. Setiap metode memiliki keunggulan dan kelemahan tertentu dalam hal kinerja dan efisiensi.
6. **Keamanan:** Komponen keamanan dalam sistem basis data melibatkan perlindungan data dari akses yang tidak sah. Ini meliputi pengaturan hak akses pengguna, enkripsi data, penggunaan kata sandi yang aman, dan tindakan keamanan lainnya untuk melindungi integritas dan kerahasiaan data.
7. ***Backup* dan Pemulihan:** Komponen ini melibatkan kegiatan membuat salinan cadangan data untuk mencegah kehilangan data akibat kegagalan sistem atau

kesalahan manusia. *Backup* dan pemulihan melibatkan proses menyimpan salinan data yang aman dan kemampuan untuk mengembalikan data ke keadaan sebelumnya jika terjadi kegagalan.

8. **Indeks:** Indeks adalah struktur tambahan yang digunakan untuk mempercepat pencarian dan pemulihan data dalam basis data. Indeks biasanya dibangun pada kolom atau atribut yang sering digunakan dalam operasi pencarian. Ini membantu mengurangi waktu yang dibutuhkan untuk menemukan data yang diperlukan.
9. **Integrasi Data:** Komponen ini melibatkan mengintegrasikan data dari berbagai sumber yang berbeda ke dalam basis data tunggal. Integrasi data dapat mencakup pemaduan data dari sistem yang berbeda, pemetaan dan transformasi data, serta menjaga konsistensi dan kesesuaian data antara sumber yang berbeda.

Komponen-komponen ini bekerja bersama untuk menciptakan sistem basis data yang efisien, aman, dan dapat diandalkan. Mereka membantu mengelola data dengan cara yang terstruktur dan terorganisir, serta memberikan akses yang cepat dan akurat kepada pengguna yang membutuhkannya

1.3 Perangkat Untuk Membuat Database

Ada beberapa perangkat lunak pembuat database yang tersedia di pasaran. Berikut adalah beberapa perangkat lunak populer yang digunakan untuk membuat dan mengelola basis data:

1. ***Microsoft Access:*** *Microsoft Access* adalah perangkat lunak basis data yang populer untuk *platform Windows.* Ini menyediakan antarmuka pengguna grafis yang intuitif untuk membuat basis data relasional. *Access* juga dilengkapi dengan alat untuk membangun formulir, laporan, dan kueri SQL.
2. **MySQL:** MySQL adalah sistem basis data open-source yang banyak digunakan. Ini merupakan pilihan populer bagi pengembang aplikasi web dan bisnis kecil. MySQL mendukung bahasa kueri SQL dan menyediakan antarmuka pengguna berbasis teks dan grafis.
3. **Oracle:** Oracle adalah sistem basis data yang kuat dan populer. Ini digunakan secara luas di perusahaan-perusahaan besar dan lingkungan bisnis yang membutuhkan kinerja dan skalabilitas tinggi. Oracle menyediakan berbagai fitur canggih untuk manajemen basis data.
4. **PostgreSQL:** PostgreSQL adalah sistem basis data open-source yang kuat dan sangat dapat diandalkan. Ini mendukung fitur-fitur maju seperti transaksi, kontrol keamanan yang ketat, dan replikasi data. PostgreSQL juga kompatibel dengan SQL dan menawarkan fleksibilitas tinggi dalam mengelola basis data.

5. **MongoDB:** MongoDB adalah salah satu contoh dari database NoSQL yang populer. Database NoSQL dirancang untuk mengelola data yang tidak terstruktur atau semi-terstruktur. MongoDB menyediakan fleksibilitas dalam menyimpan data JSON-style dan skema yang dinamis.
6. **Firebase:** Firebase adalah platform pengembangan aplikasi yang menyediakan layanan cloud, termasuk basis data real-time. Firebase Realtime Database adalah basis data NoSQL yang dapat disinkronkan secara real-time antara klien dan server.

Selain perangkat lunak ini, masih banyak lagi perangkat lunak pembuat basis data yang tersedia, baik yang berbayar maupun yang open-source. Pemilihan perangkat lunak tergantung pada kebutuhan spesifik, tingkat keahlian, dan lingkungan pengembangan yang digunakan.

1.4 Jenis Dan Tipe Database

Dalam database, terdapat beberapa jenis dan tipe yang digunakan untuk mengorganisasi dan mengelompokkan data. Berikut adalah beberapa jenis dan tipe umum dalam database:

1. **Basis Data Relasional:** Jenis basis data yang paling umum digunakan. Basis data relasional menggunakan model relasional untuk menyimpan dan mengelola data. Data diatur dalam tabel yang terdiri dari baris (tuple) dan kolom (atribut). Hubungan antara tabel didefinisikan melalui kunci primer dan kunci asing. Contoh perangkat lunak basis data relasional adalah MySQL, Oracle, dan PostgreSQL.
2. **Basis Data Hierarkis:** Jenis basis data di mana data disimpan dalam struktur hirarkis seperti pohon. Data dihubungkan melalui hubungan orang tua-anak. Basis data hierarkis biasanya digunakan dalam sistem legacy atau aplikasi yang membutuhkan struktur data yang beririsan.
3. **Basis Data Jaringan:** Jenis basis data yang mirip dengan basis data hierarkis, tetapi menggunakan struktur yang lebih kompleks. Data diorganisir dalam bentuk jaringan yang menghubungkan beberapa entitas melalui "pemilik" dan "anggota" setiap entitas. Model basis data jaringan lebih fleksibel daripada model hierarkis.
4. **Basis Data Objek:** Jenis basis data yang menggabungkan fitur dari basis data relasional dengan pemrograman berorientasi objek. Basis data objek memungkinkan penyimpanan objek kompleks dengan metode, properti, dan perilaku yang terkait. Mereka mendukung konsep seperti enkapsulasi, pewarisan, dan polimorfisme.

5. **Basis Data NoSQL:** Jenis basis data yang dirancang untuk mengelola data yang tidak terstruktur atau semi-terstruktur dengan skala yang besar. NoSQL (Not Only SQL) dapat mengelola data seperti dokumen, grafik, kolom, atau kunci-nilai. Basis data NoSQL lebih fleksibel dan skalabel daripada basis data relasional, dan umumnya digunakan dalam aplikasi web dan sistem terdistribusi.
6. **Basis Data Kolom:** Jenis basis data yang mengorganisir data berdasarkan kolom, bukan baris seperti pada basis data relasional. Basis data kolom cocok untuk kasus pengambilan data selektif yang melibatkan banyak kolom tetapi sedikit baris.
7. **Basis Data Grafik:** Jenis basis data yang dikhususkan untuk menyimpan dan mengelola data yang berhubungan dalam bentuk grafik. Grafik digunakan untuk merepresentasikan hubungan antara entitas dan simpul, dan basis data grafik menyediakan alat untuk mengeksplorasi dan menganalisis struktur tersebut.
8. **Basis Data Temporal:** Jenis basis data yang dirancang untuk menyimpan dan mengelola data yang berubah seiring waktu. Basis data temporal menyimpan riwayat data dan memungkinkan query pada data di waktu tertentu.

Setiap jenis dan tipe basis data memiliki kelebihan dan kelemahan serta digunakan untuk kasus penggunaan yang berbeda. Pemilihan jenis basis data yang tepat sangat tergantung pada kebutuhan aplikasi dan karakteristik data yang akan disimpan dan diakses.

1.5 Variabel Dan Model Database

Dalam konteks database, istilah "variabel" tidak umum digunakan. Sebagai gantinya, terdapat beberapa konsep dan istilah yang lebih relevan dalam konteks basis data. Berikut adalah beberapa istilah yang berkaitan dengan data dalam database:

1. **Kolom (Column):** Kolom dalam database merujuk pada atribut atau karakteristik yang didefinisikan untuk setiap entitas dalam tabel. Setiap kolom memiliki nama unik dan tipe data yang menentukan jenis nilai yang dapat disimpan di dalamnya.
2. **Baris (Row):** Baris dalam database, juga dikenal sebagai tuple, mewakili satu entitas atau rekaman dalam tabel. Setiap baris berisi nilai untuk setiap kolom yang sesuai dengan entitas yang direpresentasikan.
3. **Nilai (Value):** Nilai adalah data aktual yang disimpan dalam kolom. Setiap kolom dalam basis data memiliki nilai yang sesuai untuk setiap baris atau entitas.

4. **Tabel (Table):** Tabel adalah struktur utama dalam basis data relasional yang mengorganisir data dalam baris dan kolom. Tabel merepresentasikan entitas atau kategori data tertentu dalam basis data.
5. **Kunci (Key):** Kunci dalam database adalah atribut atau kombinasi atribut yang unik untuk mengidentifikasi secara unik setiap baris dalam tabel. Kunci utama (primary key) adalah kunci yang unik dan membedakan setiap baris dalam tabel. Kunci asing (foreign key) digunakan untuk menghubungkan tabel yang berbeda dalam basis data relasional.
6. **Relasi (Relationship):** Relasi dalam basis data relasional mengacu pada hubungan antara tabel yang berbeda. Relasi ditentukan oleh penggunaan kunci asing yang menghubungkan kolom dalam tabel yang berbeda.
7. **Indeks (Index):** Indeks adalah struktur data yang digunakan untuk mempercepat pencarian dan pengambilan data dalam basis data. Indeks dibangun pada kolom atau atribut tertentu dalam tabel untuk memungkinkan akses yang lebih efisien ke data.
8. **Skema (Schema):** Skema dalam basis data mengacu pada struktur, pengaturan, dan definisi keseluruhan basis data, termasuk tabel, kolom, kunci, dan relasi antara tabel. Skema menentukan bagaimana data diorganisir dan diatur dalam basis data.

Inilah beberapa istilah yang lebih umum digunakan dalam konteks basis data untuk menggambarkan komponen dan karakteristik data dalam database. Dalam basis data relasional, ada beberapa model database yang digunakan untuk merancang dan mengorganisir struktur data dalam sistem basis data. Berikut adalah beberapa model database yang umum digunakan:

1. **Model Entitas-Relasi (Entity-Relationship Model):** Model Entitas-Relasi (ER) adalah model konseptual yang digunakan untuk memodelkan hubungan antara entitas dalam basis data. Dalam model ini, entitas direpresentasikan sebagai objek dengan atribut yang menggambarkan karakteristik mereka. Hubungan antara entitas dijelaskan melalui relasi, yang menggambarkan bagaimana entitas berinteraksi satu sama lain.
2. **Model Relasional:** Model Relasional adalah model yang paling umum digunakan dalam basis data relasional. Model ini menggunakan tabel untuk mewakili entitas dan hubungan antara entitas dalam basis data. Setiap tabel terdiri dari kolom (atribut) yang mewakili karakteristik entitas dan baris (tuple) yang mewakili rekaman atau entitas individu.
3. **Model Hierarkis:** Model Hierarkis menggambarkan data dalam struktur pohon yang memiliki simpul induk dan anak. Setiap simpul anak hanya memiliki satu simpul induk di atasnya. Model ini biasanya digunakan dalam sistem legacy atau aplikasi yang membutuhkan struktur data hierarkis.

4. **Model Jaringan:** Model Jaringan adalah model yang mirip dengan model hierarkis, tetapi lebih kompleks. Data diorganisir dalam bentuk jaringan yang menghubungkan beberapa entitas melalui "pemilik" dan "anggota" setiap entitas. Model jaringan memberikan fleksibilitas yang lebih tinggi dalam menggambarkan hubungan antara entitas daripada model hierarkis.
5. **Model Objek-Relasional:** Model Objek-Relasional (OR) menggabungkan fitur dari model relasional dengan pemrograman berorientasi objek. Model ini memungkinkan penyimpanan objek kompleks dengan metode, properti, dan perilaku yang terkait. Model OR menggabungkan kekuatan model relasional dengan kelebihan pemrograman berorientasi objek.
6. **Model Dimensional:** Model Dimensional digunakan dalam basis data olap (Online Analytical Processing) yang dirancang untuk analisis dan pelaporan data. Model ini berfokus pada pengorganisasian data dalam dimensi dan fakta. Dimensi mewakili cara data dihierarki dan dimensi-dimensi tersebut saling berhubungan dengan fakta yang menggambarkan data numerik yang diukur.

Setiap model database memiliki pendekatan dan konsep yang berbeda untuk merancang struktur data. Pilihan model tergantung pada kebutuhan dan sifat data yang akan disimpan dalam basis data tersebut.

1.6 Tahapan Perancangan Database

Tahapan perancangan database melibatkan serangkaian langkah yang harus diikuti untuk merancang dan mengembangkan struktur database yang efisien dan sesuai dengan kebutuhan aplikasi. Berikut adalah tahapan umum dalam perancangan database:

1. **Analisis Kebutuhan:** Tahap pertama dalam perancangan database adalah menganalisis kebutuhan bisnis dan mengidentifikasi persyaratan data. Ini melibatkan berkomunikasi dengan pemangku kepentingan untuk memahami tujuan, proses bisnis, dan data yang diperlukan dalam aplikasi.
2. **Perancangan Konseptual:** Pada tahap ini, perancang database membuat model konseptual yang menggambarkan hubungan antara entitas dan atribut utama dalam basis data. Model ini biasanya menggunakan diagram ER (Entity-Relationship) yang membantu memvisualisasikan entitas, atribut, dan hubungan antara entitas.
3. **Perancangan Logikal:** Setelah perancangan konseptual, tahap selanjutnya adalah merancang struktur database secara logis. Ini melibatkan pemetaan model konseptual ke dalam model relasional, yang terdiri dari tabel, kolom, dan hubungan antara tabel melalui kunci primer dan kunci asing.
4. **Normalisasi:** Normalisasi adalah proses untuk mengorganisir tabel dan mengurangi redundansi data. Dalam tahap ini, perancang database menerapkan

aturan normalisasi untuk memastikan struktur database dalam keadaan normal (minimal redundansi dan anomali data).

5. **Perancangan Fisik:** Pada tahap ini, perancang database merinci implementasi fisik dari struktur database. Ini termasuk menentukan tipe data untuk setiap kolom, indeks yang diperlukan untuk mempercepat kueri, batasan integritas, dan alokasi ruang penyimpanan.
6. **Implementasi:** Tahap implementasi melibatkan pembuatan skrip atau perintah SQL untuk membuat tabel, indeks, kunci, dan batasan dalam sistem basis data yang digunakan. Selain itu, data awal juga dimasukkan ke dalam basis data.
7. **Pengujian:** Setelah basis data diimplementasikan, tahap pengujian dilakukan untuk memastikan bahwa struktur database berfungsi dengan baik. Ini melibatkan pengujian fungsionalitas, kinerja, keandalan, dan pengoptimalan kueri.
8. P**emeliharaan dan Peningkatan:** Setelah database beroperasi, perancangan database juga mencakup pemeliharaan dan peningkatan. Ini melibatkan pemeliharaan rutin, seperti backup dan pemulihan, serta penyesuaian struktur database saat kebutuhan aplikasi berubah.

Setiap tahap dalam perancangan database merupakan bagian penting untuk memastikan basis data yang efisien, andal, dan sesuai dengan kebutuhan bisnis. Penting untuk melakukan perencanaan yang cermat dan melibatkan pemangku kepentingan dalam setiap tahap perancangan.

Bab 2
Lingkungan Basis Data

2.1 Konfigurasi Model Basis Data

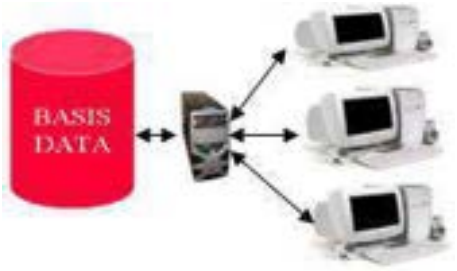
Secara umum penerapan basis data dalam program komputer dengan suatu aplikasi atau bahasa pemrograman komputer membutuhkan media penyimpanan data sedemikian rupa sehingga fungsi manajemen data dalam sistem akan dapat dilakukan dengan baik. Pengolahan basis data butuh pengetahuan tentang organisasi data, sehingga basis data akan menjadi unsur yang vital dalam sistem informasi. Penggunaan basis data untuk membangun sistem informasi (*database system*) mendapatkan prioritas utama. Beberapa defenisi yang mendukung pengertian basis data sebagai berikut:

1. Kumpulan dari beberapa arsip yang saling berhubungan satu sama lain yang tersusun sedemikian rupa yang digunakan mengeksplorasi informasi lain dengan berbagai *flatform* penyimpanan.
2. Beberapa data yang terekam dalam media tanpa pengulangan (redudansi) suatu media simpan untuk digunakan dalam membangun sebuah informasi.
3. Sekelompok berkas yang tersimpan dalam media penyimpanan yang dikelola secara elektronis untuk membangun informasi.

Penggunaan basis data dengan bahasa pemrograman atau perangkat lunak aplikasi dengan suatu metode akan menghasilkan informasi baru bagi kalangan tertentu sebagai bagian dari pengolahan data khusus. Contoh : Basis data Biro Perjalanan Bus Antar Kota, yang menjadi Objek yang dikelola dapat berupa : jadwal keberangkatan, jenis kendaraan, nomor plat bus, calon-calon penumpang, nomor kursi dan lain sebagainya. Contoh lain basis data universitas, objek yang dikelola dapat berupa : Daftar Mahasiswa Baru dan Mahasiswa lama, Daftar Dosen Pengampu Mata Kuliah, Daftar Mata Kuliah, dan lain sebagainya. Ada 2 jenis arsitektur basis data yang sering digunakan antara lain :

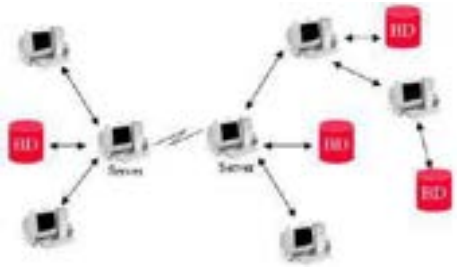
1. **Basis Data Terpusat,** pada bagian ini merekam dan mengambil data dilakukan dalam *server*, ditunjukkan pada gambar 2.1

Contoh :



Gambar 2.1 Basis Data Terpusat

2. **Basis Data Tersebar,** secara logika bahwa server yang aktif mendistribusikan data keberbagai lokasi untuk dapat digunakan oleh *client* yang sedang terhubung artinya *client* hanya bisa menggunakan data yang terlah didistribusi oleh *sever*, diperlihatkan pada gambar 2.2
   Contoh :



Gambar 2.2 Basis Data Tersebar

Beberapa kriteria basis data yang sesuai untuk ke dua bentuk di atas adalah:

1. Berorientasi pada Data dan bukan berorientasi pada program.
2. Digunakan oleh beberapa program aplikasi yang aktif tanpa terjadi perubahan pada struktur basis data sumbernya.
3. Mudah dikembangkan baik terhadap volume ataupun strukturnya.
4. Memenuhi kriteria sistem baru yang membutuhkan dengan cara yang berbeda .
5. Terpenuhi kerangkapan data seminimal mungkin.

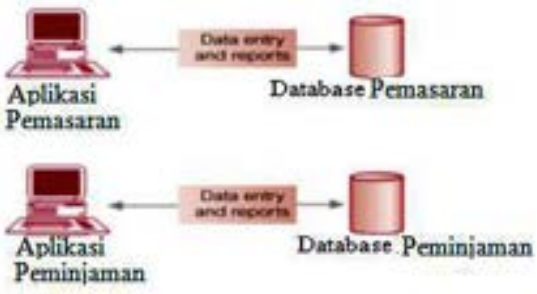
Manfaat utama mengelola basis data dengan baik agar data yang dibutuhkan lebih sesuai dan akses yang cepat dan tepat. Sedangkan tujuan dari penggunaan basis data sebagai berikut:

1. Terpenuhi kecepatan dan kemudahan akses (***Speed***).
2. Merjamin adanya efisiensi ruang penyimpanan yang digunakan (***Space***).
3. Adanya akurasi penyimpanan dan pengambilan data (***Accuracy***).
4. Dipastikan ketersediaan data dalam media simpan (***Availability***).

5. Tidak terjadi penyimpangan data saat dipanggil oleh program (***Completeness***).
6. Dipastikan aman dan tidak bias terhadap kegiatan lain (***Security***).
7. Penggunaan data oleh beberapa *client* terpenuhi (***Sharability***).

Dalam memangun suatu basis data yang perlu diperhatikan antara lain:

1. **Pendekatan dengan kebiasaan (tradisional),** pendekatan ini memiliki kekurangan antara lain:
   a. Data ganda (*data redundancy*), hal ini mungkin disebabkan :
      1) Adanya perubahan terhadap data-data yang tersimpan sebelumnya sehingga menjadi tidak efisien.
      2) Penggunaan media simpan yang berlebihan.
   b. Tidak terdapat hubungan yang jelas antar data (*data relatability*), diperhatikan gambar 2.3



Gambar 2.3 Basis Data Pendekatan Tradisional

Beberapa kelemahan pendekatan sistem basis data secara tradisional:

a. **Minimnya relasi data,** kurangnya relasi ini terlihat pada saat diimplementasikan dengan sebuah aplikasi sehingga tidak memberikan hasil sesuai dengan kebutuhan.
b. **Redundansi data,** redundansi terjadi apabila aplikasi baru menggunakannya yang mengakibatkan bahwa data yang sama tersimpan secara berulang. Contoh nama dan alamat pelanggan akan disimpan dalam berkas piutang dan juga dalam berkas pemasaran. Jika hal ini terjadi dapat mengakibatkan ketidak konsistenan data yang akan diinformasikan, misalnya jika terjadi perubahan alamat dari pelanggan tanpa melakukan perubahan di seluruh *file* yang menyimpankan data alamat akan memberikan informasi yang berbeda.
c. **Kurangnya standarisasi,** kekurangan ini dimaksudkan bahwa tidak terdapat kumpulan perangkat lunak secara terpusat. Dalam sebuah sistem basis data secara tradisional adalah tidak terbangun sebuah kamus data terpusat sehingga programmer tidak memperhatikan standard. Bisa saja dalam satu berkas nama pelanggan dibuat **nm_plgn** dan dalam berkas yang lain dibuat **nama_pelanggan**.

d. Akibat tidak ada standar produktifitas maka aplikasi baru mengakibatkan buruknya hasil pengolahan. Jika seorang programmer harus menghabiskan waktu untuk melihat apakah data yang dibutuhkan aplikasi yang baru sudah tersedia atau tidak dalam berkas, atau harus mengecek organisasi dari berkas untuk melihat apakah dapat dipakai dalam aplikasi yang baru, maka programmer akan kehabisan waktu untuk menemukan jawaban dari pada membuat aplikasi yang baru.

2. **Pendekatan manajemen sistem pengolahan basis data (*Database Management System* (DBMS)),** Pendekatan ini merupakan langkah dalam memperbaiki kelemahan yang terjadi pada pendekatan basis data secara kebiasaan (tradisional) yang mungkin terjadi karena disebabkan:
   a. Terjadi penggandaa data (*data redundancy*).
   b. Peningkatan relasi antar data (*data relatability*).

   Ciri-ciri pendekatan sistem manajemen basis data:
   a. Banyak aplikasi untuk satu database.
   b. Tingkat redundansi lebih kecil.
   c. Tingkat keamanan data lebih terjamin.
   d. Setiap data terintegrasi.

   Beberapa kelebihan dari pedekatan sistem manajemen basis data (DBMS):
   a. Terjamin integrasi data sehingga dalam merancang informasi dengan beberapa atau sebuah aplikasi saat membutuhkan data dapat terpenuhi.
   b. Meningkatnya kemampuan aplikasi dalam mengakses data tanpa harus membangun program baru.
   c. Meningkatkan keterpaduan data. Dalam hal ini akan mengurangi data yang tidak konsisten saat dipanggil.
   d. Pemeliharaan dan pengembangan aplikasi menjadi lebih berkualitas. Dengan DBMS para pemrogram tidak perlu khawatir terhadap struktur, dan lokasi dari sebuah *file*.
   e. Pengamanan data lebih fleksibel terhadap perkembangan aplikasi.

   Terdapat sejumlah kelemahan sistem manajemen basis data (DBMS) yaitu:
   a. Konstruksi basis data terlalu kompleks.
   b. Menungkinkan beberapa aplikasi menjadi lebih lambat mengakses data.
   c. Penyesuaian perkembangan teknologi dengan biaya yang lebih mahal.

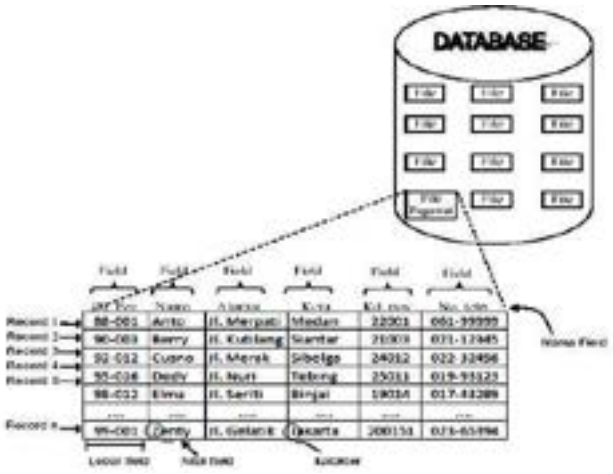
   Jenjang data dan tipe basis data dapat diuraikan sebagai berikut:

1. Basis data yang terdiri datai sejumlah *file* (berkas) yang saling berhubungan menjadi satu kesatuan yang tidak terpisahkan.
2. *File* adalah sejumlah *record* yang dibangun memiliki struktur *field* yang sama dan sejenis.

3. *Record* merupakan sejumlah *field* yang bergabung dan menggambarkan unit data individu.
4. *Field* adalah sejumlah atribut dalam *record* yang menunjukkan satu item dari data seperti nama, nim, alamat, dan lain sebagainya.
5. *Byte* merupakan atribut dari *field* berupa karakter yang membentuk nilai dari sebuah *field*.
6. Bit adalah bagian terkecil dari data secara keseluruhan, dapat berupa karakter ASCII nol atau satu yang merupakan komponen pembentuk *byte*.

Tipe *File* dalam basis data:

1. ***File* Induk,** sebuah *file* penting yang harus tersedia dalam basis data. *File* ini terbagi dalam 2 bagian:
   a. File induk bersifat rujukan adalah bahwa seluruh *record* yang tersimpan bersifat relatif statis (tidak berubah secara signifikan). Misalnya nomor antrian, identitas seseorang, dan lain sebagainya.
   b. *File* induk bersifat dinamis adalah bahwa seluruh *record* yang tersimpan sering terjadi perubahan setiap kali terjadi transaksi. Misalnya dalam *file* untuk jumlah stok barang.
2. ***File* Transaksi,** merupakan aktivitas perekaman data dalam sebuah transaksi. Misalnya pada *file* untuk jumlah penjualan, pengambilan data supplier.
3. ***File* Laporan,** merupakan hasil yang ditampilkan dari sebuah kegiatan pengolahan data
4. ***File History,*** merupakan data-data yang tersimpan cukup lama yang mungkin tidak pernah digunakan lagi
5. ***File Backup,*** merupakan salinan dari data aktif dalam waktu tertentu. Untuk lebuh jelas perlihatkan pada gambar 2.4



Gambar 2.4 Pendekatan Basis Data Terkomputerisasi

2.2 Evolusi Teknologi Basis Data

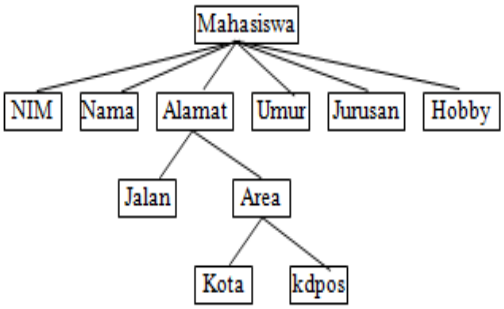
Perkembangan basis data hingga sampai saat ini telah melewati beberapa tahapan proses antara lain:

1. **Flat *File* Basis Data,** *record* diorganisir secara primitif, misalnya *file* teks, dimana panjang teks terbatas (*tab delimited*), setiap memasukkan data dipisahkan dengan satu atau beberapa karakter khusus (*delimiter*), contoh dengan penggunaan *vertical delimiter* (|), karena bentuknya datar dan tidak melebihi batasan lembaran kertas disebut *flat file*. Jenis aplikasi modern yang menggunakan *flat file* dan masih ada hingga saat ini adalah : *Spread Sheet* misalnya : *Excel*, *Lotus 1-2-3*, *OpenOffice Calc*, dan sebagainya. Ciri-ciri *flat file* bahwa datanya tidak saling terikat atau berdiri sendiri. Ciri khusus dalam *Flat File* masih terlihat dalam bentuk basis data modern misalnya yang menonjol adanya *record* sebagai salah satu elemen penyusunnya.

	A	B	C	D
245		1	21110174	Advent Halawa
246		2	21110172	Afri Nirmalasari Halawa
247		3	21110109	Ahmad Dedi Ramadhan Nasution
248		4	21110059	Alexander Franklin Sitorus
249		5	21110200	Ardin Fau
250		6	22110210	Bastian Syahputra Manurung
251		7	21110138	Citra Lisensya Manurung
252		8	21110179	Dewi Maulida Sari Tanjung

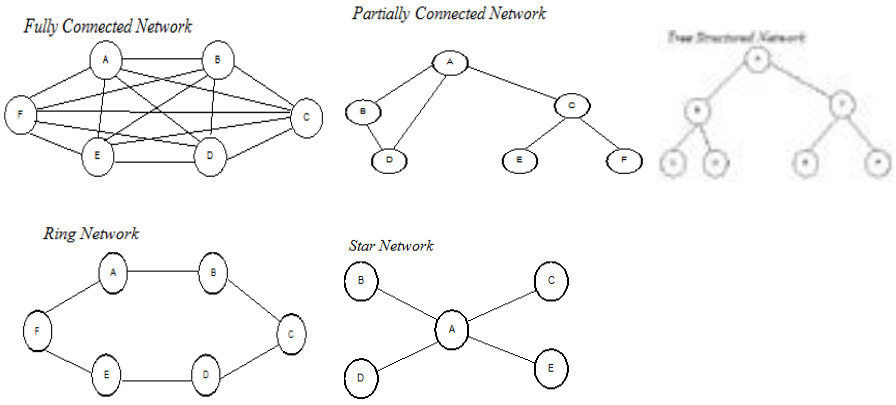
Gambar 2.5 Bentuk Flat *File* Dalam MsExcel

2. **Data Hirarki**, Model ini berbeda dengan *Flat File*, dimana ‘hirarki’ berarti adalah tingkatan, hirarki dapat berbentuk pohon biner dengan representasi hubungan antara induk (*parent*) dan anak (*child*) dengan sebuah *pointer* yang menunjuk posis data. Keterhubungan antar atribut prinsip susunan adalah *one to many* dalam pembentukan basis data. Pengaksesan data sangat bergantung dari hirarki sehingga seakan-akan secara fisik dalam media penyimpanan dan bukan bergantung terhadap kelompok *file*. Faktanya model jarang digunakan bahkan terkesan tidak ada, meskipun hubungan atau mata rantai dalam evolusi basis data masih melekat sifat relasi *one to many* sebagai salah satu ciri dari basis data modern, lebih jelas ditunjukkan pada gambar 2.6



Gambar 2.6 Bentuk Hirarki Dalam Basis Data

3. **Model *Network*,** sering disebut Basis data terdistribusi. Model ini adalah pengembagan dari hirarki. Keterhubungan antar unsus menunjukkan adanya relasi *many to many*. Rancangan basis data ini dinilai tidak terlalu praktis penerapannya, meski demikian basis data sebelumnya masih tetap dipakai sebagai dasar pengembangan selanjunya, seperti relasi yang mempunyai sifat *one to many* bahkan *many to many*.

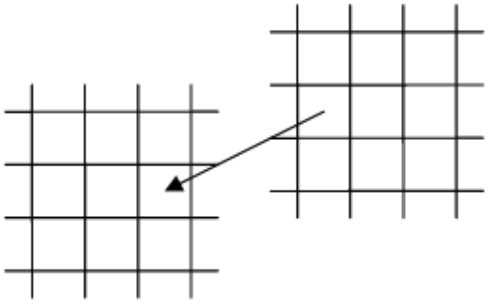


Gambar 2.7 Bentuk *Network* Dalam Basis Data

Kelebihan dan kekurangan dari penggunaan bentuk *network*. Kelebihan **terhubung penuh** (*Fully connected)* adalah jika sebuah simpul (*node*) rusak maka yang simpul yang lainnya masih berfungsi sedangkan kekurangannya manajemen data sulit dilakukan. Kelebihan **terhubung sebagian** (*Partially Connected)* adalah biaya dapat ditekan sedangkan kelemahannya kehandalan rendah. Kelebihan **terhubung pohon** (*Tree Connected)* adalah data tersimpan secara terpusat, pengolahan data lebih nyaman sedangkan kelemahannya bahwa jika simpul pusat rusak semua simpul yang lainnya menjadi tidak berfungsi. Kelebihan **terhubung lingkar** (*Ring Connected)* adalah bila ada salah satu

simpul yang rusak, maka simpul yang lainnya masih dapat berfungsi sebagai mestinya sedangkan Kelemahannya bahwa pengolahan data kurang nyaman sebab sifatnya tersebabar ke seluruh simpul lainnya. Kelebihan **terhubung bintang** (*Star Connected)* adalah pengolahan data lebih terjamin, karena sifatnya terpusat sedangkan kelemahannya jika simpul pusat rusak maka semua simpul lainnya hilang hubungan dan kehandalan rendah.

4. **Model *Relational*,** Model ini paling banyak digunakan hingga sekarang, sebab ketangguhan dan kemudahannya lebih mudah dalam pengelolaannya. Teknik penyusunan *file* adalah data tersimpan dalam tabel yang tersusun dengan adanya baris (*row*) dan kolom (*column*). Baris-baris (rows) menyatakan banyaknya *record* sedangkan kolom–kolom menyatakan *field*. Pemasukan data *record* dengan sejumlah atribut. Terdapat sejumlah syarat harus dipenuhi adalah proses normaliasi. Sifat relasi *one to one*, *one to many* dan *many to many* dalam mendefinisikan entitas. Relasi *many to many* jarang digunakan meskipun bisa dilakukan tetapi harus menyederhanakan bentuk relasi dengan beberapa relasi *one to many*.



Gambar 2.8 Bentuk Relational Dalam Basis Data

Karakteristik dari *relational* dalam basis data adalah:

a. Setiap unsur dari baris dan kolom harus memiliki nilai atomik.
b. Setiap unsur dalam relaso kolom tertentu dengan jenis yang sama.
c. Setiap kolom terdapat penamaan (atribut) yang unik.
d. Dalam *file* yang sama tidak terdapat lebih dari 2 baris identitas yang sama.

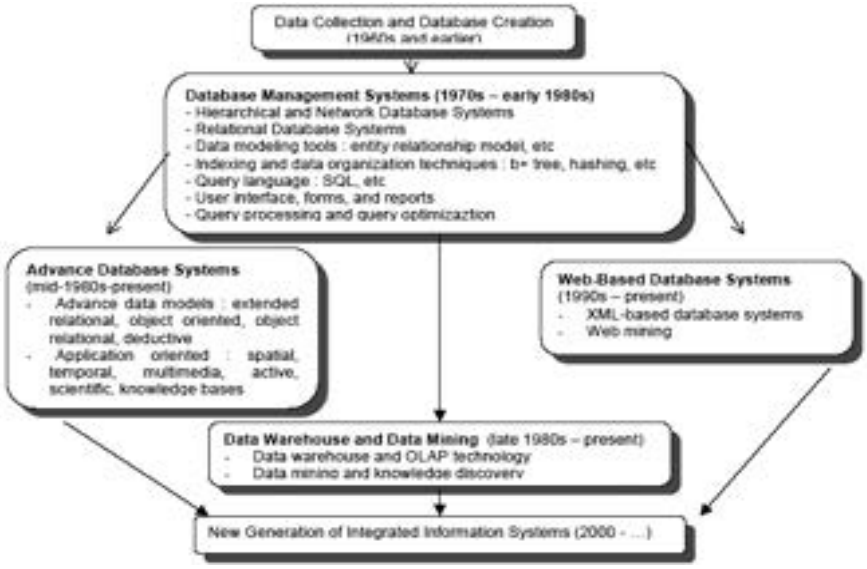
Kelebihan relational dalam basis data adalah:

a. Deskripsi sederhana dan mudah dalam penggunaannya.
b. Operasi data seperti *insert*, *delete*, *update* lebih baik.
c. Penerapannya dapat keberbagai bahasa pemrograman.
d. Dapat mengoptimalkan biaya operasional.
e. Keamanan lebih terjamin.

Terdapat sejumlah karakteristik dalam *file* sebagai berikut:

a. Tidak terdapat baris ganda.
b. Urutan pembuat baris tidak dibatasi.
c. Semua unsur kolom diberi nama unik dan letaknya tidak dibatasi.
d. Semua unsur kolom adalah nilai tunggal dan jenis sama.

Berikut disajikan bentuk umum, evolusi perkembangan teknologi basis data.



Gambar 2.9 Perkembangan Basis Data Hingga Saat ini

5. **Basis Data Berelasi Objek,** Pengolahan data berelasi objek (ORDBMS (*Object Relational Database Management System*)). Kelemahan RDBMS dilengkapi OODBMS. OODBMS mengandung sifat *encapsulation*, *inheritance*, *polymorphism*. OODBMS lebih mendukung ORDBMS (*Object Relational Database Management System*) pada data kompleks. Perbandingan OODBMS dengan ORDBMS ditunjukkan dalam tabel berikut 2.1

Tabel 2.1 Perbandingan OODBMS dengan ORDBMS

Pemodelan Data		
Model	**OODBMS**	**ORDBMS**
Identitas Objek	Ada	Ada dengan tipe REF (Refferences)
Pengkapsulan	Ada, tapi tidak dapat digunakan untuk queri	Ada dengan UDT (User Difined Types)
Penurunan	Ada	Ada (dipisahkan antara UDT dan Table)

Polymorphism	Ada, sebagai object oriented pada model bahasa pemrograman	Ada
Objek Kompleks	Ada	Ada, dengan UDT
Relasi	Ada	Sangat Mendukung untuk mendefinisikan batasan referensi integritas

6. **Data Warehouse,** Penyimpanan data dalam sebuah gudang informasidan bersifat non relasional, sehingga sering disebut sebagai basis data berorientasi pada sebuah subyek data, bersifat terpadu, waktu bervariasi dan bersifat permanen. Adapun ciri-cirinya adalah :
   a. Berorientasi pada subyek data artinya mengatur semua subyek terpusat terhadap konten data bukan pada *tools* yang dipakai.
   b. Terpadu, bahwa penggunaan data bersama yang mungkin dapat menyebabkan menjadi tidak konsisten.
   c. Variasi waktu data dalam gudang sesuai dan akurat selama dalam batas waktu tertentu.
   d. Permanen, data tidak ada perubahan secara langsung

   Keuntungan Warehousing adalah:
   a. Hasil berkapasitas besar dan dan lebih Kompetitif.
   b. Dapat Meningkatkan produktivitas.

   Jenis basis data berdasarkan penggunaannya:
   a. Pemrosesan transaksi secara online (*Online Transaction Processing (*OLTP)). Merupakan data yang sering diremajakan yang mencerminkan sifat aplikasi yang dinamis.
   b. Terdapat query *Decision Support System* (DSS). Merupakan Analisa basis data yang bersifat relatif statis.

   Perbedaan data Operasional dan DSS ditunjukkan dalam tabel 2.2

Tabel 2.2 Perbedaan Data Operasional dengan DSS

Data Operasional	Data DSS
Berorientasi pada aplikasi: Data digunakan untuk proses bisnis. Sebagai contoh: Sistem perbankan denga file terpisah yang sudah dalam bentuk normal untuk setiap proses bisnis	Berorientasi pada subyek: Data digunakan untuk subyek bisnis, misal informasi nasabah. Data dalam bentuk denormalisasi dimana sebuah record dapat meliputi keseluruhan proses bisnis
Data terperinci	Data Ringkas
Struktur statik	Struktur dinamik

Target operator komputer	Target pengambil keputusan pada seluruh tingkatan
Volatile (data dapat diubah)	Non volatile (data tidak dapat diubah setelah dimasukkan)
Kebutuhan data selalu diketahui sebelum rancangan sistem	Kebutuhan data sama sekali tidak diketahui sebelum rancangan sistem
Mengikuti siklus hidup pengembangan klasik dimana iterasi rancangan diselesaikan melalui normalisasi data dan memeriksa kebutuhan pemakai	Siklus hidup pengembangan sama sekali berbeda, dimana pemakai menggunakan aplikasi struktur data yang ada dan membuat rancagan siap untuk di analisis
Performansi penting karena jumlah pemakai konkuren sebagai besar dalam mengakses data	Masalah performansi lebih longgar karena jumlah pemakai jauh lebih sedikit dalam mengakses data sehingga tidak ada masalah konkuren yang perlu di perhatikan.
Penggerak transaksi (transaction driven)	Pengeerak analisis (analysis driven)
Data harus selalu tersedia untuk pemakai akhir (back up dan recovery harus terencana dengan baik)	Tidak mempunyai tingkat kebutuhan ketersediaan data yang sama (perencanaan back up dan recovery lebih longgar)
Mencerminkan situasi mutakhir	Mencerminkan nilai historis

7. **Data Mart,** Merupakan sebuah modul. Misalnya *data mart* publikasi buku, *data mart* sumber daya manusia sebuah perusahaan, dan lain sebagainya.
8. **Data Mining,** Eksplorasi hubungan data terjadi di luar kendali pengguna. *Data mining* menggambar suatu hubungan dan penyajian yang jelas. Contoh : Bank menawarkan kemudahan transaksii terhadap sejumlah nasabahnya. Tata kelola pemberian bunga bank dalam bentuk penawaran dalam kegiatan tertentu. Cara pengolahan dalam data mining adalah:
   a. Berdasarkan Verifikasi.
   b. Berdasarkan Penemuan.
   c. Perpaduan Pemeriksaan Data Mining dan dengan sejumlah kajian.

   Beberapa tool yang banyak beredar di pasaran adalah : Lminer, RapidMiner, Tanaga, dan lain sebagainya
9. **Web Database,** Dokumen yang tersimpan dalam Web dimana yang menjadi unsur utama dinyatakan dalam beberapa tipe. Salah satunya *hypertext* dimana sebuah aplikasi berbentuk *Hypertext Mark Up Language* (HTML). Konten dalam HTML terdiri dari sejumlah naskah berbentuk alphabetik dengan format

khusus dengan *link* ke sejumlah konten lain dengan menggunakan suatu protokol. Protokol itu berupa *Hypertext Transfer Protocol* (HTTP) dengan alamat khusus yang disebut *Uniform Resourcer Locator* (URL). Situs sebagai sebuah media untuk Web-DBMS Integrity. Beberapa acuan aplikasi basis data berbasis web adalah :

a. Kemampuan akses dan aturan yang lebih nyaman.
b. Konektivitas vendor independen boleh memilih DBMS saat ini.
c. Kemampuan interface didukung dengan web browser dan web *server*.
d. Konektivitas memberikan keuntungan untuk semua bentuk DBMS.
e. *Open*-arsitektur dengan berbagai teknologi.
f. Beban koneksi menjadi pertimbangan penting baik dalam pengurangan biaya operasional dan pemeliharaan aplikasi yang mendukung HTTP.
g. Sesi dan aplikasi otentikasi terdefenisi.

10. ***Database Language,*** Cara berinteraksi denga menggunakan bahasa khusus yang terdiri dari 2 jenis perintah dalam mengolah basis data antara lain:
    a. ***Data Definition Language* (DDL),** DDL berfungsi mengatur sejumlah *file* untuk membantu *Database Administrator* (DBA) membuat entitas, attribut dan relasi tertata dengan baik dan akan tersimpan dalam Kamus Data (*Data Dictionary*).
    b. ***Data Manipulation Language* (DML),** DML berfungsi menata data secara prosedural atau non prosedural terkait dalam pengambilan data berupa:
        1) Pemasukan atau penambahan data baru
        2) Penghapusan data
        3) Pengubahan atau modifikasi data

    Dalam mengakses data ada 2 hal yang diperhatikan antara lain:
    a. Instruksi harus sesuai dengan prinsip kerja DBMS baik dalam menyimpan atau pengambilan data dimana DML dibuat secara non prosudural.
    b. Instruksi-instruksi yang dihasilkan secara internal merujuk ke DBMS dalam tata kelola data. Pemrograman basis data secara konvensional yang sering digunakan berupa Pascal, C, Cobol, Fortran dan lain sebagainya, dengan pendekatan prosedural.

2.3 Arsitektur Basis Data

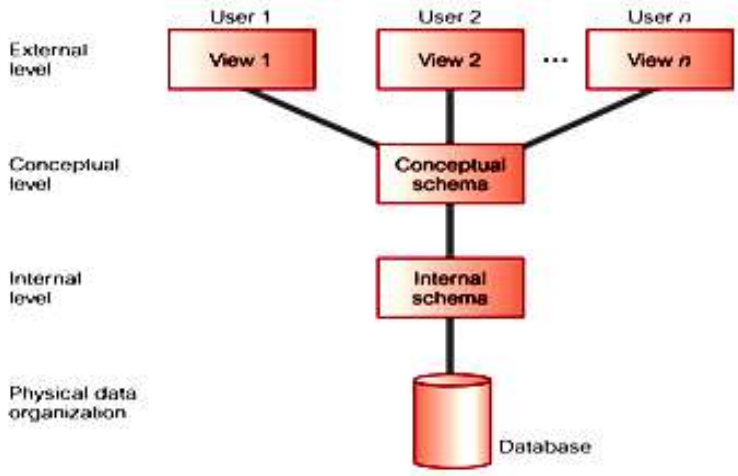
Bentuk disain sistem basis data harus memperhatikan skema struktur mendekripsikan proses simpan dan pemeliharaan sehingga tidak membingungkan pemakai. Menurut ANSI-SPARK ada 3 tingkatan arsitektur yaitu:

1. ***Tingkatan Eksternal (External Level),*** Berdasarkan pandangan pengguna yang menjelaskan bagian basis data sesuai sehingga pengguna dapat menggambarkan

pola data dalam bentuk yang sudah dikenal terkait dengan unsur entitas, atribut dan hubungannya.

2. ***Tingkatan Konseptual (Conseptual Level),*** Pola data dan hubungannya akan tersimpan. Dalam tingkatan konseptual ini mencakup:
   a. Setiap entitas dengan atribut dan relasinya terdefenisi dengan jelas.
   b. Batasan data konsisten.
   c. Informasi semantik berkaitan satu sama lainnya dan tidak perlu mempertimbangkan besar penyimpanan.
3. ***Tingkatan Internal (Internal Level),*** Proses pembuatan basis data dalam media penyimpanan secara fisik berkaitan dengan tempat penyimpanan (*physical storage)*. Aspek penting dalam tingkatan ini adalah:
   a. Indeks dan alokasi ruang penyimpanan, penyimpan dan penempatan *record*, Contoh : *sequential*, *relative* atau *index sequential.*
   b. Pemampatan (*compression*) dan teknik enkripsi (*encryption*).

Berikut arstektur basis data menurut ANSI-SPARK ditunjukkan dalam gambar 2.10



Gambar 2.10 Arsitektur Basis Data Dengan ANSI-SPARK

Aspek penting dalam Tingkatan Eksternal / View Level antara lain:

1. Pandangan user fokus pada pola basis data yang ada.
2. Segmen data sesuai dengan pemakai tertentu, ditunjukkan pada gambar 2.11.

Aspek dalam tingkat Konseptual / Logical Level antara lain:

1. Pandangan komunitas pada basis data.
2. Terdeskripsi unsur yang tersedia dalam basis data.
3. Terdefenisi struktur logika dari keseluruhan basis data.

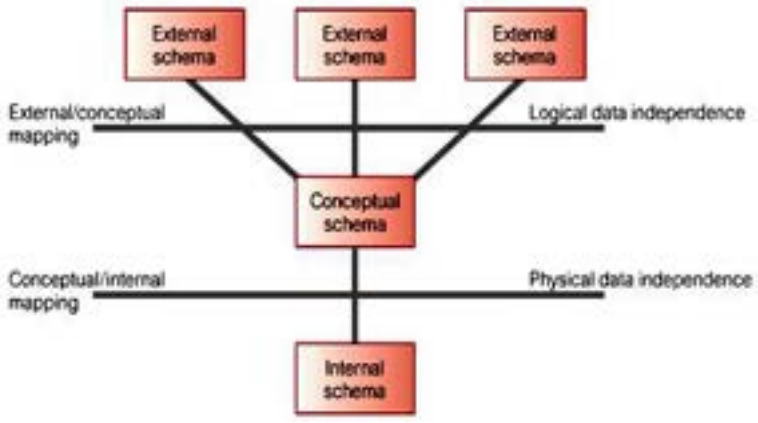
Aspek dalam tingkat Internal / Physical Level antara lain:

1. Bentuk fisik basis data pada media penyimpanan.
2. Gambaran proses penyimpanan data dalam database.

Hubungan arsitektur DBMS menjelaskan ketersediaan dan kebebasan data (*data independence*) artinya bahwa dalam tingkatan ini perubahan dari jenjang tinggi ke rendah atau sebaliknya tidak berpengaruh dalam:

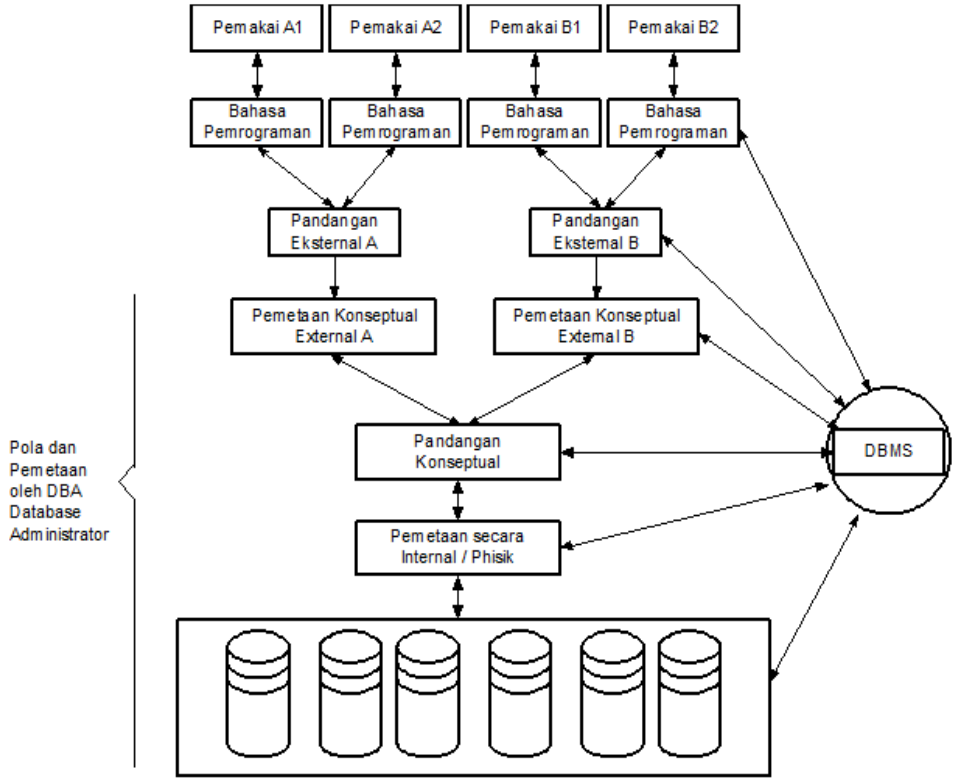
1. Pembatasan kebabasan data logika *(Logical Data Independence).*
2. Perubahan ketahanan skema pertukaran antara eksternal dan konseptual dijamin, misalnya saat penambahan atau penghapusan entitas, atribut dan relasi baru.
3. Perubahan kebebasan data fisik *(Physical Data Independence)* antara eksternal dan konseptual dijamin.
4. Ketahanan skema secara konseptual dalam hal pertukaran pada skema internal, misalnya struktur penyimpanan *file* yang berbeda dengan alat penyimpanan berbeda, termasuk modifikasi *hashing* terhadap skema konseptual (eksternal).

Berikut diperlihatkan bentuk arsitektur Kebabasan Data (Data Independent) dalam gambar 2.11.



Gambar 2.11 Arsitektur Data Independent

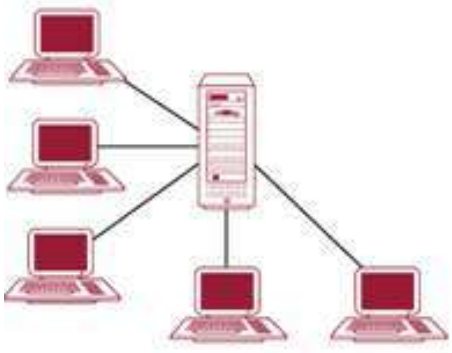
Secara teknis implementasinya *data independent* [ada konseptual dan fisik dalam basis data diperlihatkan dalam gambar 2.12



Gambar 2.12 Arsitektur Implementasi Data Independent

Beberapa variasi arsitektur *multi user* adalah:

1. *Teleprocessing*, ciri-cirinya adalah:
   a. Arsitektur secara tradisional.
   b. *Single mainframe* dengan sebuah terminal terpasang.



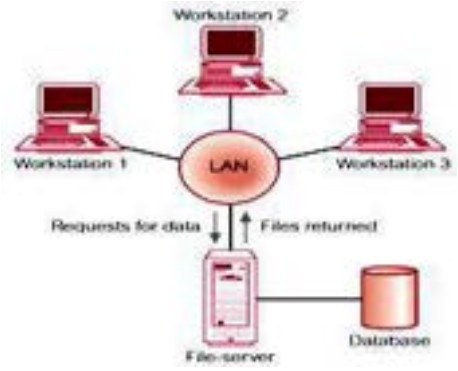
Gambar 2.13 Arsitektur Teleprocessing

2. *File*-Server, dengan ciri-ciri sebagai berikut:

a. *File*-server terhubung dengan sejumlah stasion melalui jaringan.
b. Basis data terseimpan pada *file-server.*
c. Aplikasi dan DBMS tersimpan dalam masing-masing workstation.

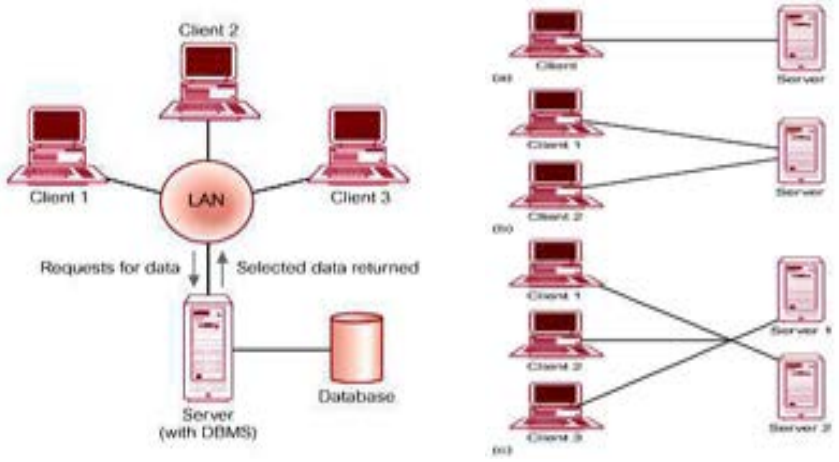
Kelemahan *File*-Server, ciri-cirinya adalah:

a. Lalu lintas jaringan yang signifikan.
b. Setiap *workstation* memiliki duplikasi DBMS.
c. Keterpaduan dan konkurensi lebih kompleks.



Gambar 2.14 Arsitektur *File* Server

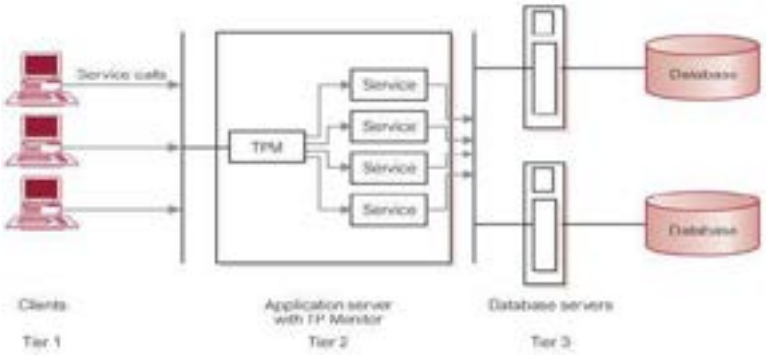
3. *Client-Server*, ciri-cirinya adalah:
   a. Basis data tersimpan dalam *Server.*
   b. *Client* sewaktu-wantu dapat mengakses basis data.



Gambar 2.15 Arsitektur *File* Server

4. Monitor Proses Tanssaksi *(Transaction Processing Monitors)*
   a. Transformasi data antara *Clients* dan *Servers* secara langsung oleh OLTP.

b. Perubahan posisi dari tingkatan menengah sampai tiga tingkat dalam sistem *client-server.*



Gambar 2.16 Arsitektur Transaction Monitors

Bab 3
Database Management System

3.1 Pendahuluan

Data merupakan salah satu hal yang sangat berharga bagi Perusahaan, UMKM, Rumah Sakit, Sekolah, Perguruan Tinggi, bahkan di Pemerintahan, bayangkan jika Perusahaan, Rumah Sakit, Sekolah, Perguruan Tinggi, UMKM, bahkan Pemerintahan memiliki data yang tidak teratur dan tidak terkontrol maka saat data tersebut dibutuhkan Perusahaan akan mengalami kesulitan mendapatkan data. Ingat, ditahun 2000-an saat berobat ke Puskesmas. Setiap kali berobat petugas selalu bertanya: *Apakah sudah pernah ke Puskesmas?*, maka petugas akan membuatkan keanggotaan baru di Puskesmas tersebut dalam bentuk **Kartu Keanggotaan** dan **Kartu Rekam Medik**. Kartu keanggotaan puskesmas biasanya berukuran 8 cm x 5 cm sedangkan Kartu Rekam Medik biasanya berukuran 15 cm x 21 cm seperti terlihat pada gambar 3.1 dan gambar 3.2 dibawah:


DINAS KESEHATAN KOTA TERJAUH

PUSKESMAS SURYO MENGGOLO

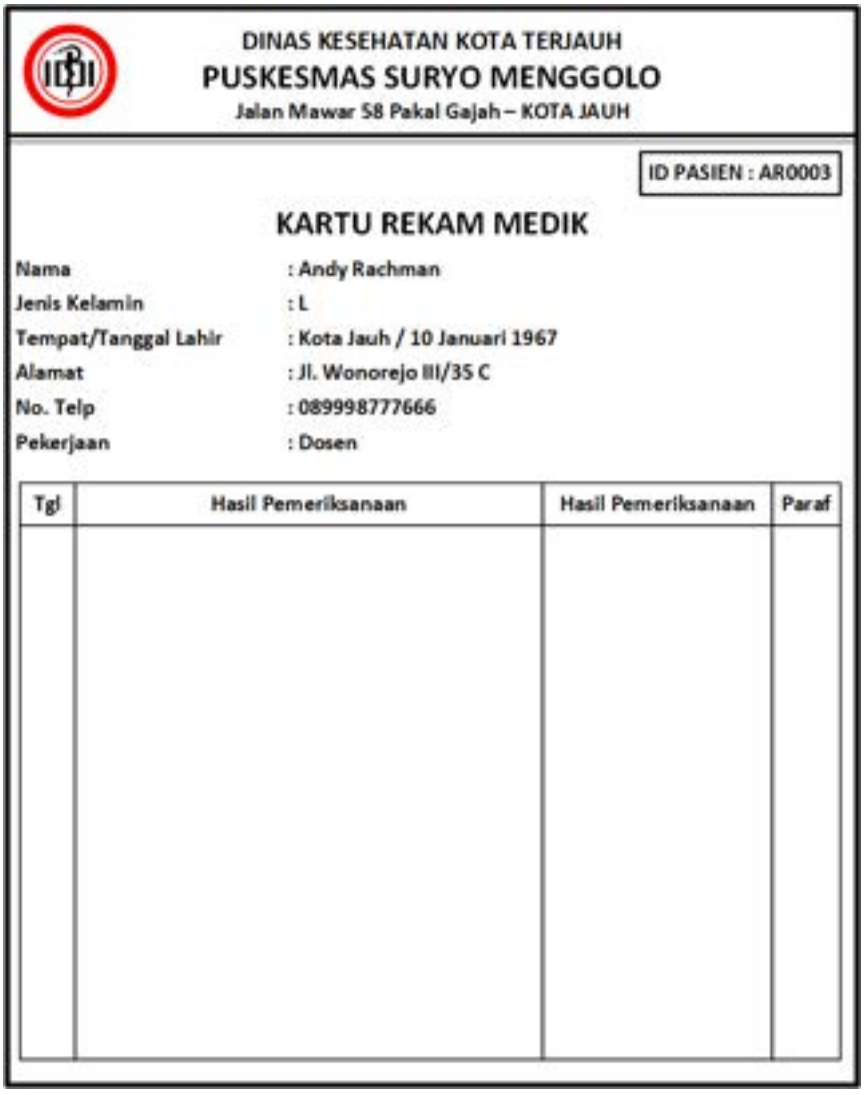
Jalan Mawar 58 Pakal Gajah – KOTA JAUH

KARTU TANDA BEROBAT

NAMA : Andy Rachman
UMUR : 20 Tahun
ALAMAT : Jl. Wonorejo III/35 C
ID KARTU : AR0003

"TIAP BEROBAT KARTU HARUS DIBAWA"

Gambar 3.1 Kartu Berobat Puskesmas Suryo Menggolo



DINAS KESEHATAN KOTA TERJAUH
PUSKESMAS SURYO MENGGOLO
Jalan Mawar 58 Pakal Gajah – KOTA JAUH

ID PASIEN : AR0003

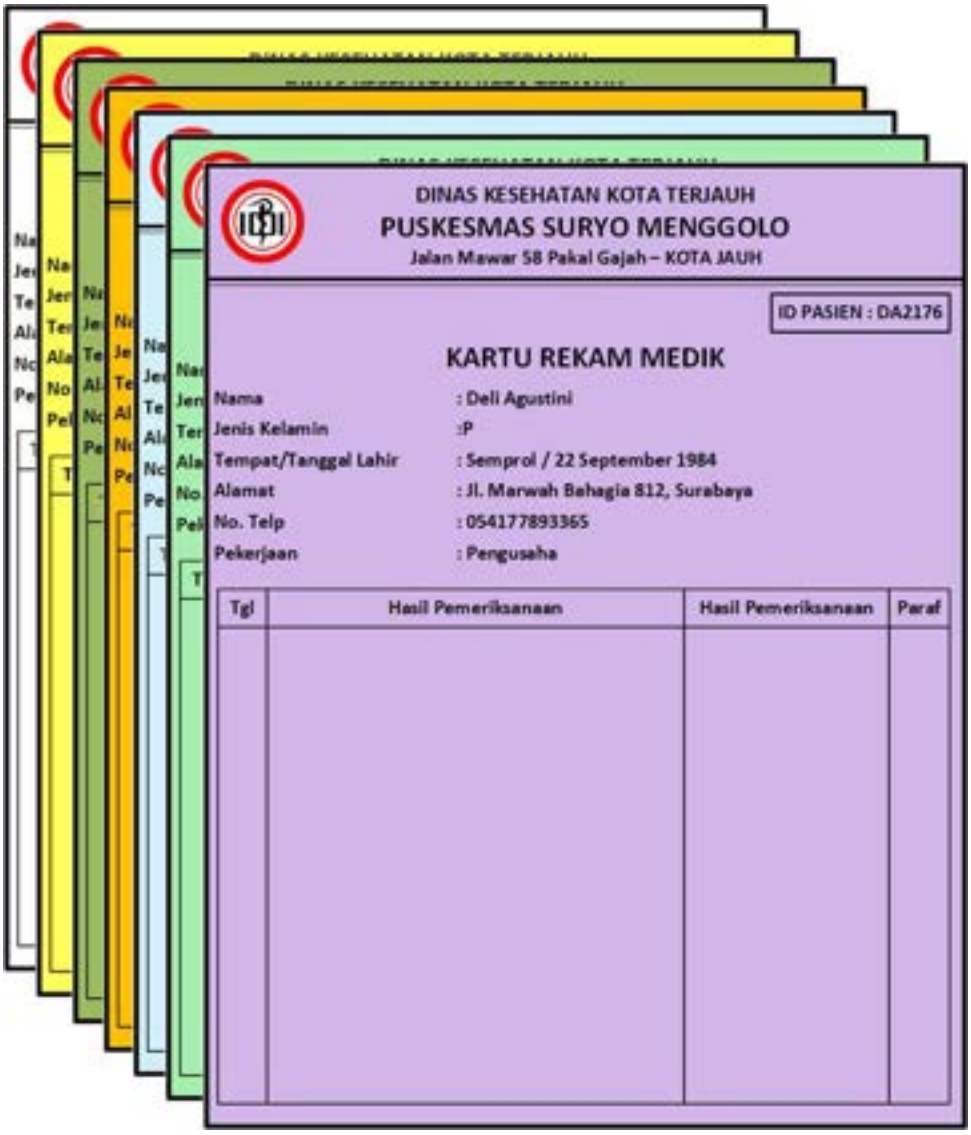
KARTU REKAM MEDIK

Nama : Andy Rachman
Jenis Kelamin : L
Tempat/Tanggal Lahir : Kota Jauh / 10 Januari 1967
Alamat : Jl. Wonorejo III/35 C
No. Telp : 089998777666
Pekerjaan : Dosen

Tgl	Hasil Pemeriksanaan	Hasil Pemeriksanaan	Paraf

Gambar 3.2 Kartu Rekam Medik Puskesmas Suryo Menggolo

Dengan adanya Kartu Berobat Pasien dan Kartu Rekam Medik Puskesmas berarti memiliki data Pasien serta data kondisi kesehatan pasien. Data kondisi kesehatan pasien dipantau oleh puskesmas pada Kartu Rekam Medik. Kondisi ini sudah sangat bagus untuk penyimpanan data, tetapi tengoklah kembali pada saat kita berobat 3 - 4 bulan setelahnya, apa yang terjadi? Pertanyaan yang sama akan ditanyakan oleh Petugas Puskesmas adalah *Apakah sudah pernah kesini*? *Atas nama siapa*? *Kartu berobat apakah dibawa*? Petugas Puskesmas akan mencari data pasien. Satu puskesmas bisa memiliki lebih dari 500 pasien bahkan lebih dengan kombinasi kode pasien yang beragam seperti terlihat pada gambar 3.3, Satu pasien diwakili oleh satu warna.

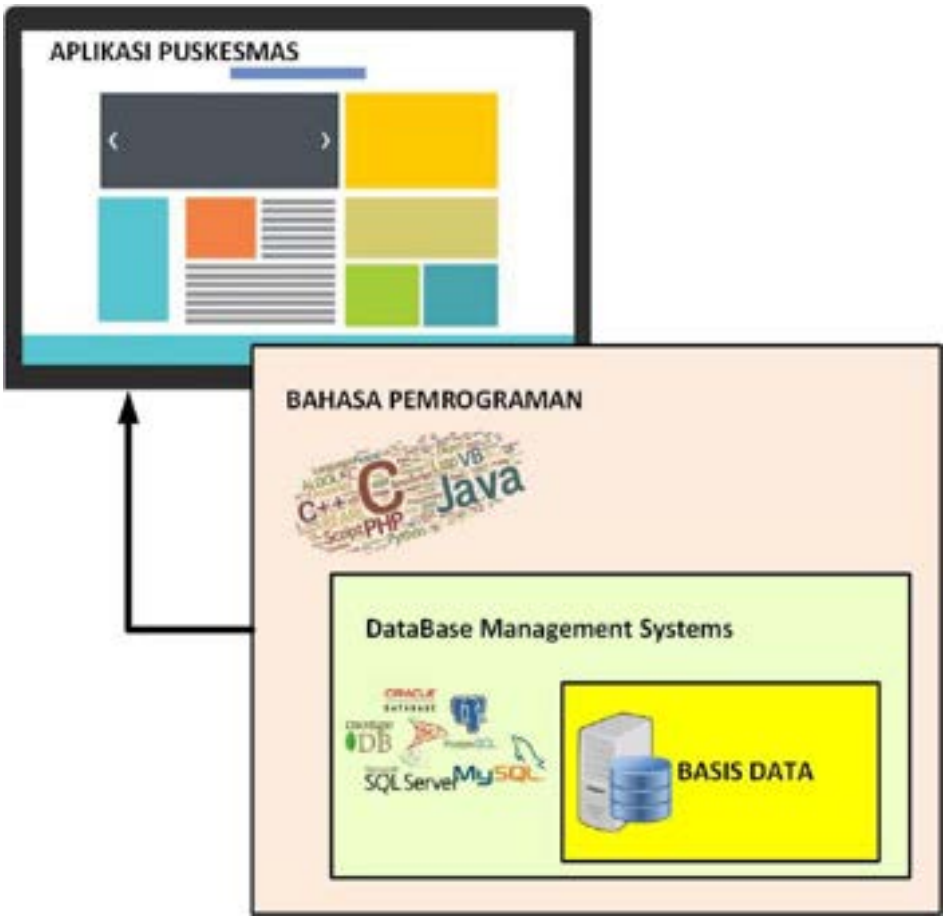


Gambar 3.3 Kartu Rekam Medik Pasien Puskesmas Secara Keseluruhan

Nah disinilah permasalahan tersebut akhirnya muncul, dimana dengan jumlah pasien yang sangat banyak dengan data yang sangat besar dan kebutuhan waktu yang sempit seringkali Petugas Puskesmas membuatkan Kartu Rekam Medik yang baru bagi pasiennya dan seringkali kondisi ini selalu terjadi bagi pasien yang sama, yang berarti satu pasien bisa memiliki lebih dari 1 Kartu Rekam Medik. Kondisi seperti ini akan menyebabkan kesalahan yang fatal saat Kartu Rekam Medik yang ada dibuat manual dan akan memberikan analisis kondisi pasien yang berbeda untuk satu dokter karena Kartu Rekam Mediknya berbeda mesikipun nama pasiennya sama, lihat gambar 3.4 berikut ini.

Gambar 3.4 Kartu Rekam Medik Dengan Data Pasien Sama

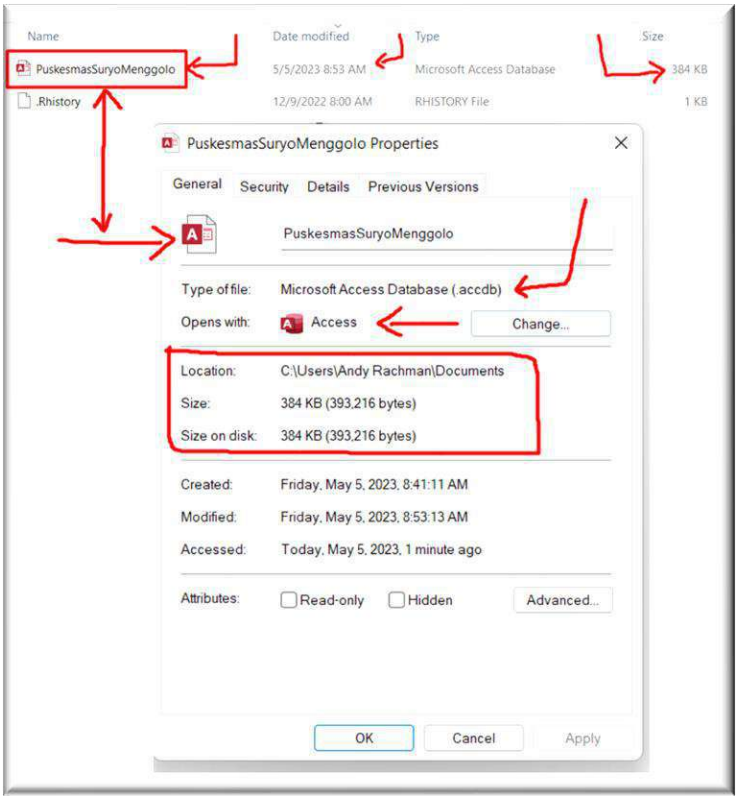
Kondisi seperti tersebut diatas (gambar 3.4) didalam teori basis data disebut dengan redundansi atau duplikat atau sama atau berulang. Jadi jika terdapat data atau dokumen yang sama persis satu dengan lainnya maka dapat dikatakan bahwa data itu adalah redundansi atau sama. Untuk mengatasi hal tersebut diperlukan suatu alat bantu agar tidak terjadi redundansi data. Alat bantu tersebut adalah aplikasi. Jika pada kondisi diatas makan Aplikasi yang dimaksud adalah Aplikasi Puskesmas. Sebuah Aplikasi Puskesmas secara arsitektur terbagi menjadi 3 bagian, yaitu **Aplikasi itu sendiri**, **Bahasa Pemrograman** yang membangun aplikasi tersebut, dan yang terakhir adalah **Basis Data atau *DataBase***. Adapun arsitektur dalam pengembangan aplikasi seperti terlihat pada gambar 3.5 berikut ini.



Gambar 3.5 Arsitektur Pengembangan Aplikasi

3.2 Pengenalan DBMS

Setiap aplikasi yang dikembangkan oleh seorang *developer* pasti menggunakan bahasa pemrograman, entah itu C++, Java, PHP, .NET, dan lain-lain. Pada saat pengembangan aplikasi developer juga memerlukan sebuah basis data yang digunakan dalam menyimpan data. Nah basis data ini memerlukan perangkat bantu yang dinamakan DataBase Management Systems (DBMS). DBMS yang digunakan dapat beragam, mulai dari MySQL, SQL Server, Mongo DB, Microsoft Access, Paradok, PostgreSQL, dan masih banyak lagi. Masing-masing DBMS menghasilkan sebuah file database dengan ekstensi atau tipe tertentu, misalnya Microsoft Access, setiap file *database* yang dibuat memiliki tipe data ***.accdb**. *.accdb menunjukkan bahwa file *database* tersebut diciptakan oleh Microsoft Access dimana *accdb singkatan dari **Microsoft Access Database** seperti terlihat pada gambar 3.6 berikut ini.



Gambar 3.6 Penciptaan Database Puskesmas dengan DBMS Microsoft Access

DBMS merupakan sebuah sistem perangkat lunak yang memungkinkan pengguna untuk melakukan definisi, penciptaan, perawatan, dan akses kontrol pada sebuah database (basis data). DBMS berinteraksi dengan program aplikasi pengguna dan basis data. Sebuah DBMS menyediakan fitur-fitur sebagai berikut (Connolly & Begg, 2015):

1. Memungkinkan pengguna dalam melakukan pendefinisan basis data dengan menggunakan **Data Definition Language (DDL)**. DDL memungkinkan pengguna menentukan Tipe dan Struktur Data serta batasan pada data yang akan disimpan dalam sebuah basis data.
2. Memungkinkan pengguna dalam memasukkan, menghapus, mengambil, dan memperbaruhi data dari basis data. Kondisi ini dilakukan dengan menggunakan Bahasa Manipulasi Data atau **Data Manipulation Language (DML)**.
3. Memungkin penyediaan akses kontrol ke basis data, misalnya:
   a. Sistem Keamanan (*Security System*) dalam pencegahan pengguna yang tidak memiliki hak akses agar tidak dapat mengakses basis data.
   b. Sistem Integritas (*Integrity System*) dalam menjaga konsistensi data yang disimpan pada basis data.

c. Sistem Kontrol Konkurensi (*Concurrency Control System*) memungkinan pengaksesan bersama dari basis data.
d. Sistem Kontrol Pemulihan (*Recovery Control Systems*) memungkinan pemulihan basis data ke keadaan konsisten sebelumnya setelah kegagalan perangkat keras atau perangkat lunak
e. Katalog yang dapat diakses pengguna (*User-Accessible Catalog*) berisi deskripsi dari data pada suatu basis data.

Kelebihan Sistem Manajemen Basis Data bila dibandingkan dengan cara tradisional antara lain (Bal Gupta & Mittal, 2017):

1. **Terkendalinya Terjadinya Redundansi/Duplikasi**: Pada sistem tradisional, setiap aplikasi yang dikembangkan pasti memiliki datanya sendiri-sendiri. Kondisi tersebut akan menyebabkan duplikasi data. Terjadinya duplikasi akan menyebabkan pemakaian ruang penyimpanan yang besar, oleh karena itu, keberadaan DBMS akan sangat membantu mengontrol terjadinya duplikasi data. Hal ini dapat dilihat pada gambar 4 dimana terjadinya duplikasi data rekam medik pasien Puskesmas Suryo Menggolo.
2. **Konsistensi Data**: Pada penyimpanan data secara tradisional pada saat perbaikan data pada suatu file menyebabkan terjadinya data yang tidak akurat karena file berbeda bisa jadi berisi informasi yang berbeda dari item data yang sama pada waktu tertentu**.** Hal ini menyebabkan informasi yang salah pada pengguna, pada sistem basis data hal ini dapat diatas dengan pengendalian atau mengontrol terjadinya redundansi.
3. **Idependensi Data Program**: File sistem tradisional bergantung pada data yang berarti pengorganisasian file bergantung pada aplikasi tertentu. Pada sistem basis data menyediakan independensi antara sistem file dan aplikasi sehingga pada saat perubahan data pada sistem basis data tidak harus melakukan perubahan pada program aplikasi.
4. **Berbagi Data**: Pada sistem basis data, data dikontrol secara terpusat dan dapat dibagikan oleh semua pengguna yang memiliki hak akses.
5. **Pemberlakuan Standar**: Pada sistem manajemen basis data, data diatur oleh Administrator basis data / *Database Administrator* (**DBA**), hal ini dilakukan untuk memastikan bahwa data yang dikirimkan sesuai dengan standar atau aturan yang ada, misalnya data mahasiswa, data dosen, data penjualan, dan lain-lainnya.
6. **Peningkatan Intgritas Data**: Yang dimaksud dengan integritas data adalah bahwa data yang ada dalam sebuah basis data akurat dan konsisten. Dengan kontrol yang terpusat memungkinkan proses pemeriksaan yang memadai menyediakan integritas data.

7. **Peningkatan Keamanan**: Keamanan basis data (*database security*) berarti perlindungan data pada basis data terhadap pengguna yang tidak berwenang menggunakannya. Administrator basis data / *Database Administrator* (**DBA**) memastikan bahwa prosedur pengaksesan harus ditaati termasuk diantaranya skema autentifikasi.
8. **Akses Data Yang Efisien**: Sistem basis data melakukan proses penyimpanan data dengan meggnunakan teknik-teknik yang canggih sehingga data dapat disimpan secara efisien.
9. **Penyeimbang Persyaratan Bertentangan**: Administrator basis data / *Database Administrator* (**DBA**) bertanggungjawab penuh pada berjalannya sistem sehingga DBA dapat mengambil keputusan atau tindakan yang bertentangan dengan persyaratan pengguna dan aplikasi.
10. **Peningkatan Fungsi Cadangan dan Pemulihan**: Dengan adanya backup dan recovery subsystem (bagian cadangan dan pemulihan), sistem basis data menyediakan fasilitas pemulihan sistem dari kerusakan/kegagalan perangkat keras dan perangkat lunak.
11. **Meminimalkan Program Perawatan**: Dalam sistem basis data tradisional pengaksesan deskripsi data dan logika dilakukan secara manual yang artinya perubahan pada format data atau metode akses menyebabkan modifikasi program yang ada, tetapi dengan adanya sistem basis data hal ini dapat dikurangi atau direduksi dengan adanya independensi data dan program aplikasi.
12. **Kualitas Data Tinggi**: Kualitas data pada sebuah sistem basis data sangat tinggi bila dibandingkan dengan sistem tradisional hal ini disebabkan karena pada sebuah sistem basis data memiliki alat bantu dan teknik-teknik khusus yang dapat diimplementasikan, misalnya teknik pencarian data, teknik pemrosesan data dan lain-lainnya.
13. **Aksesibilitas dan Daya Tanggap Data Yang Baik**: Pada bagian ini, sistem basis data mempunyai fitur penggunaan bahasa kueri atau *query language*. Dengan fitur ini, sistem basis data dapat melayani permintaan pengguna secara langsung tanpa harus menulis program aplikasi. Fungsi ini dapat dimungkinkan karena kemampuan integritas data pada sistem basis data.
14. **Kendali Serentak**: Kendali Serentak atau concurrency control yang dimaksud adalah kemampuan sistem basis data dalam pengelolaan akses basis data oleh banyak pengguna. Hal ini juga memungkinkan pencegahan kehilangan informasi atau kehilangan integritas.
15. **Skala Ekonomis**: Dalam sistem basis data pengorganisasian file dilakukan secara terpusat, dengan teknik ini beberapa program aplikasi yang dikembangkan dapat mengurangi biaya dibandingkan dengan sistem

tradisional, hal ini karena biaya pengoperasian dan pengelolaan basis data hanya dilakukan pada tempat sehinga lebih ekonomis.

16. **Peningkatan Produktifitas Programmer**: Keberadaan sistem basis data sangat membantu programmer atau pengembang, hal ini dikarenakan sistem basis data memberikan banyak fitur atau fungsi yang dapat dimanfaatkan oleh programmer untuk melayani kebutuhkan pengguna sehingga produktifitas programmer jauh lebih baik dan meningkat.

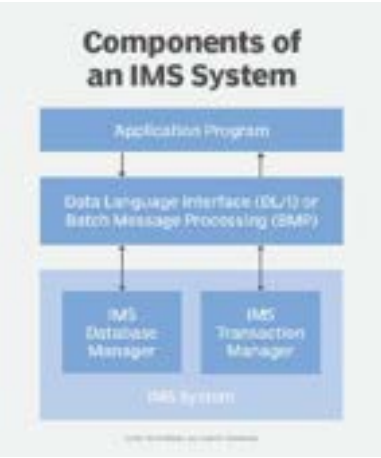
3.3 Sejarah Perkembangan DBMS

Dari permasalahan diatas, dapat diambil kesimpulan bahwa sebuah data sangat penting, apalagi data tersebut berjumlah sangat besar, dan beragam. Untuk itu diperlukan teknik untuk melakukan proses pengaturan, proses temu kembali informasi dan pengolahan data dengan tepat sehingga data tersebut dapat disampaikan sebuah informasi yang bermanfaat bagi pengguna informasi. Pada tahun 1960, Charles Bachman mengembangkan suatu penyimpanan data yang terintegrasi berbasis pada model data jaringan (geeksforgeeks.org, 2022). Dipertengahan tahun 1960, IBM menciptakan Sistem Manajemen Basis Data/DataBase Management Systems (**DBMS**) yang diberikan nama Information Management Systems (**IMS**). IMS pertama kali digunakan pada IBM 2740 pada tahun 1968. Saat ini IMS digunakan oleh IBM pada IBM System/360 (en.wikipedia.com, 2023).

Gambar 3.7 IBM Z2740 (A) dan IBM System/360 (B)

IMS ini dikembangkan oleh IBM untuk mendukung Badan Penerbangan dan Antariksa Nasional yaitu Program Luar Angkasa Apollo. Di tahun 1970 banyak perusahaan yang menggunakan IMS diantaranya perusahaan manufaktur, asuransi, dan perusahaan ritel. Pada tahun 1980 – 1990 sebagian Bank dan Lembaga Keuangan juga menggunakan IMS. Ditahun 2000 an 90% perusahaan dunia menggunakan IMS. IMS sendiri terdiri dari dua komponen utama, yaitu IMS Database Manager (IMS DB) dan

IMS Transaction Manager (IMS TS). Pengelolaan data pada IMS DB berbentuk hirarki, dimana setiap level bergantung pada data di level berikutnya yang lebih tinggi (Gills, 2022).



Gambar 3.8 Komponen System pada IBM IMS

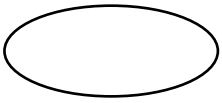
Bab 4
Entity Relationship Model dan Relational Database Model

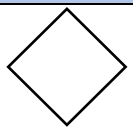
4.1 Entity Relationship Diagram (ER-Diagram)

Entity Relationship Model (ER-Model) dikembangkan untuk memfasilitasi perancangan basis data untuk menggambarkan hubungan antar data secara menyeluruh yang mewakili struktur logis dari database (Silberschatz, Korth, & Sudharshan, 2019). ER-Model merupakan model data konseptual tingkat tinggi yang didasari pada paradigma bahwasannya dunia nyata merupakan sekumpulan dari beberapa objek dasar (entitas) dan relasi antar objek data tersebut(Fikry, 2019).

ER-Model membantu dalam melakukan analisis persyaratan data secara sistematis untuk menghasilkan perancangan basis data yang baik dengan pendekatan top-down untuk perancangan basis data dimulai dari mengidentifikasi data penting dalam hal ini disebut entitas, atribut yang menyimpan informasi detail tentang suatu entitas, hubungan antar data sampai dengan batasan-batasannya yang harus direpresentasikan dalam model (Connolly & Begg, 2016). ER-Model adalah teknik penting yang harus dikuasai oleh setiap perancang basis data. Simbol dasar yang digunakan pada Entity Relationship Model adalah:

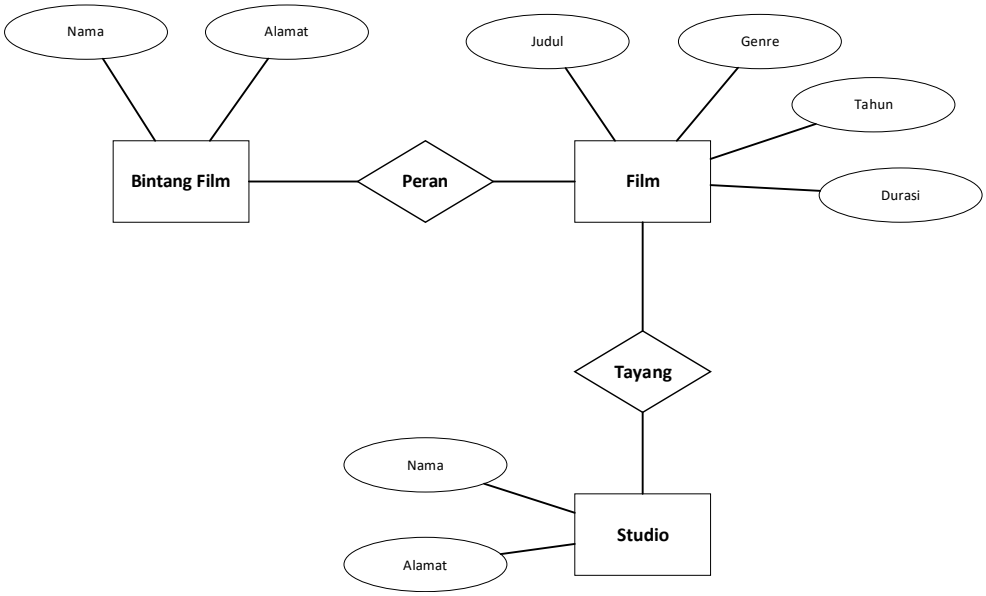
Tabel 4.1 Simbol Entity Relationship Model

Simbol	Keterangan
	**Entitas**, kumpulan dari objek di dunia nyata yang dapat diidentifikasi secara unik.
	**Atribut**, bagian dari entitas/relasi/tabel yang berisi penjelasan rinci dari sebuah entitas/relasi/tabel.

Simbol	Keterangan
	**Relasi**, Suatu hubungan antara satu entitas dengan entitas lainnya.
	Link, garis sebagai penhubung antara entitas dengan relasi atau antara entitas dengan atributnya.

Pada ER-Model, struktur data direpresentasikan dalam bentuk grafik yang disebut Entity Relationship Diagram (ERD) yang menggunakan tiga tipe element, yaitu Entity, Attributes dan Relationship (Garcia-Molina, Ullman, & Widom, 2008). Entity Relationship Diagram dapat mengekspresikan struktur logis dari keseluruhan basis data secara grafis. Entity Relationship Diagram ditampilkan dengan cara yang simpel dan jelas serta sebagian besar menggambarkan secara luas penggunaan ER-Model(Silberschatz et al., 2019).

Gambar 4.1 merupakan Entity Relationship Diagram yang menggambarkan basis data sederhana tentang film, terdiri dari tiga entity set yaitu Film, Bintang dan Studio.



Gambar 4.1 Entity Relationship Diagram untuk Database Film

Set entitas Film memiliki empat atribut kami yang biasa: judul, genre, tahun, dan durasi. Dua entitas lainnya menetapkan Bintang dan Studio kebetulan memiliki dua atribut yang sama: nama dan alamat, masing-masing dengan arti yang jelas. Terlihat juga dua hubungan dalam diagram, yaitu:

1. **Peran**, adalah hubungan antara entitas film dengan bintang film yang memiliki makna dalam sebuah film diperankan oleh satu atau beberapa bintang film, sebaliknya seorang bintang film memerankan satu atau beberapa film.
2. **Tayang**, adalah hubungan antara entitas film dengan studio yang memiliki makna sebuah film ditayangkan di satu atau beberapa studio, sebaliknya satu studio dapat menayangkan satu atau beberapa film.

4.1.1 Entity Set

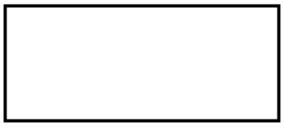
Objek dasar yang mewakili ER-Model adalah entitas, yang merupakan objek di dunia nyata dengan eksistensi independen (real) atau objek dengan eksistensi konseptual (abstrak) (Connolly & Begg, 2016) seperti yang ditunjukkan pada gambar 4.2. Entitas dapat berupa objek dengan keberadaan fisik seperti mobil, rumah, karyawan atau dapat berupa objek dengan keberadaan konseptual seperti perusahaan, pekerjaan atau universitas (Elmasri & Navathe, 2010). Entitas semacam objek abstrak dan kumpulan entitas serupa membentuk kumpulan entitas/entity set.

Entitas Eksistensi Fisik	Entitas Eksistensi Konseptual
Karyawan	Perusahaan
Rumah	Pekerjaan
Mobil	Perpustakaan
Supplier	Universitas
Sparepart	

Gambar 4.2 Contoh entitas dengan eksistensi fisik atau konseptual

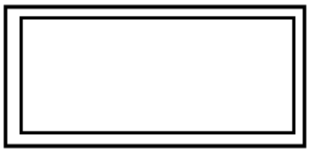
Entity set adalah sekumpulan entitas dengan tipe data dan atribut yang serupa, misalnya kumpulan orang yang menjadi instruktur di perguruan tinggi tertentu dapat disebut sebagai himpunan entitas instruktur. Dalam pemodelan data, entitas yang dipilih dalam abstrak tanpa mengacu ke himpunan entitas individu tertentu. Entitas terbagi menjadi terbagi menjadi dua, yaitu:

1. Strong Entity atau Entitas Kuat, adalah entitas yang independen, dapat berdiri sendiri, dan tidak bergantung pada entitas lainnya(Gupta & Mittal, 2017). Entitas kuat memiliki karakteristik atribut yang unik disebut kunci utama yaitu sebuah atribut tunggal atau gabungan atribut-atribut yang secara unik dapat digunakan untuk membedakan dari entitas yang lain. Entitas kuat pada model Entity Relationship Diagram (ERD) digambarkan dengan simbol persegi panjang.



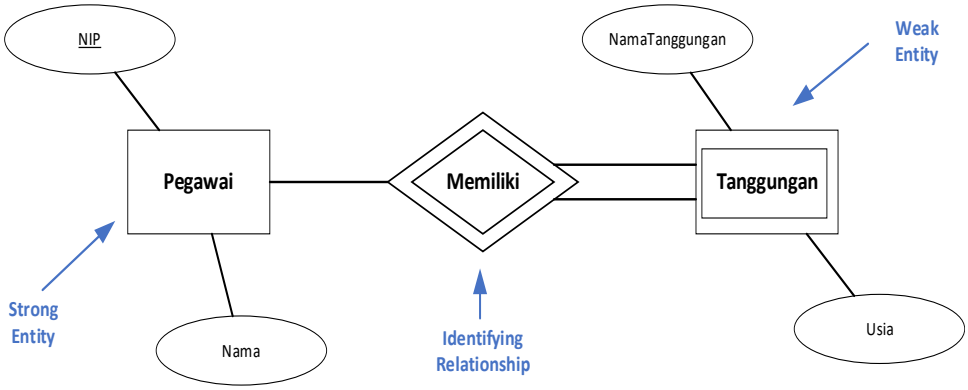
Gambar 4.3 Simbol Entitas Kuat (Strong Entity)

2. Weak Entity atau Entitas Lemah, adalah entitas yang tidak dapat berdiri sendiri karena keberadaanya bergantung penuh terhadap entitas kuat(Gupta & Mittal, 2017). Entitas lemah dengan menghubungkan entitas tertentu dari tipe entitas yang lain ditambah atribut dari entitas lemah. Entitas lemah tidak memiliki kunci utama melainkan kunci tamu yang terbentuk dari kunci utama entitas kuat. Entitas lemah pada model Entity Relationship Diagram (ERD) digambarkan dengan simbol persegi panjang bertumpuk.



Gambar 4.4 Simbol Entitas Lemah (Weak Entity)

Gambar 4.5 menggambarkan penggunaan entitas kuat dan entitas lemah dalam ER-Diagram.



Gambar 4.5 Contoh penggunaan entitas kuat dan entitas lemah dalam pemodelan ERD

Pada gambar 4.5 digambarkan terdapat satu entitas kuat yaitu pegawai dan satu entitas lemah yaitu Tanggungan, kedua entitas tersebut dihubungkan dengan relasi tayang sebagai identifying relationship. Pegawai disebut sebagai entitas kuat karena dapat berdiri sendiri atau tidak bergantung terhadap entitas lain, sedangkan tanggungan disebut entitas lemah karena keberadannya bergantung penuh terhadap pegawai. Artinya setiap pegawai bisa saja memiliki tanggungan bisa juga tidak, namun tanggungan akan muncul jika ada pegawai yang sudah memiliki tanggungan.

Setiap entitas yang lemah harus dikaitkan dengan entitas yang mengidentifikasi atau entitas kuat, himpunan entitas yang lemah dikatakan keberadaannya tergantung pada himpunan entitas yang kuat. Set entitas yang kuat dikatakan memiliki set entitas lemah yang diidentifikasinya. Hubungan yang mengaitkan set entitas lemah dengan set entitas pengenal disebut identifying relationship.

4.1.2 Atribut

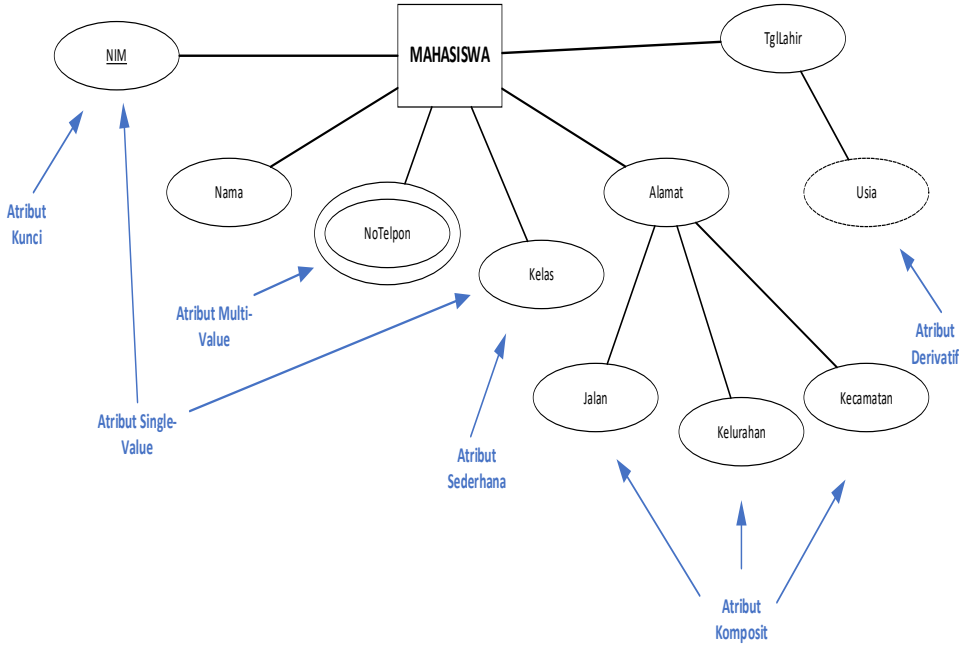
Pada setiap entitas diwakili oleh seperangkat atribut yang merupakan sifat deskriptif yang dimiliki oleh setiap anggota himpunan entitas. Penentuan atribut untuk entity set menyatakan bahwa basis data menyimpan informasi yang sama terkait setiap entitas dalam entity set. Tetapi setiap entitas bisa saja memiliki nilai sendiri untuk setiap atribut (Silberschatz et al., 2019).

Setiap atribut dikaitkan dengan sekumpulan nilai yang disebut domain. Domain mendefinisikan nilai potensial yang mungkin dimiliki atribut dan mirip dengan konsep domain dalam model relasional (Connolly & Begg, 2016). Domain dari nama atribut lebih sulit untuk didefinisikan, karena terdiri dari semua nama yang memungkinkan. Bisa berupa string karakter, bukan hanya terdiri dari huruf dan juga tanda hubung atau karakter khusus lainnya. Model data yang dikembangkan sepenuhnya mencakup domain dari setiap atribut dalam ER-Model.

Terdapat beberapa jenis atribut yang digunakan dalam pemodelan data menggunakan Entity Relationship Diagram (Connolly & Begg, 2016), antara lain:

1. **Atribut Sederhana**, adalah atribut yang hanya memiliki satu atribut dan tidak dapat dibagi menjadi atribut yang lebih kecil. Atribut sederhana sering juga disebut sebagai atribut atom. Contoh dari atribut sederhana adalah NIM dan Kelas dari entitas mahasiswa. Pada ERD atribut sederhana digambarkan dengan simbol elips biasa.
2. **Atribut Single-Value**, adalah atribut yang hanya dapat memiliki satu nilai didalamnya. Contoh dari atribut Single-Value adalah NIM dan Kelas dari Entitas mahasiswa. Sama halnya dengan atribut sederhana, atribut ini digambarkan dengn simbol elips biasa pada ERD.
3. **Atribut Multi-Value**, adalah atribut yang dapat menampung lebih dari satu nilai dalam satu baris data/record. Contoh dari atribut Multi-Value yaitu Gelar, No Telpon dan email dari entitas mahasiswa. Atribut ini menggunakan simbol elips bertumpuk atau dengan garis ganda untuk menggambarkan menggunakan ERD.
4. **Atribut Derivatif**, adalah atribut yang nilainya dihasilkan dari atribut yang lain yang dapat melibatkan asosiasi atribut daru berbagai jenis entitas. Contoh atribut derivatif adalah Usia dimana nilai dari atribut tersebut merupakan hasil perhitungan antara tanggal lahir dan tanggal hari ini, sehingga keberadaan atribut usia bergantung terhadap atribut tanggal lahir. Atribut ini digambarkan dengan simbol elips yang bergaris putus-putus.
5. **Atribut Composite**, adalah atribut yang dapat dibagi menjadi beberapa komponen yang lebih kecil dengan keberadaan independen masing-masing. Contoh dari atribut composite adalah atribut Nama atau Alamat. Atribut nama

tersebut dapat dibagi menjadi atribut nama depan dan nama belakang, Sedangkan alamat dapat dibagi menjadi atribut nama jalan, nomor, kota dan provinsi. Simbol yamh digunakan untuk menggambarkan atribut komposit dalam ERD yaitu elips biasa yang memiliki sub dibawahnya. Gambar 4.6 menunjukkan simbol beberapa jenis atribut yang digunakan pada ERD.



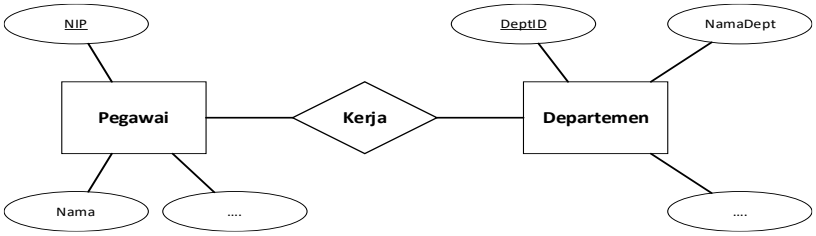
Gambar 4.6 Contoh jenis atribut dalam pemodelan ERD

4.1.3 Relationship

Relationship merupakan asosiasi/hubungan di antara dua atau lebih entity set (Gupta & Mittal, 2017). Relationship set adalah sekumpulan relasi antar entitas dengan tipe yang sama (Silberschatz et al., 2019). Relationship Types merupakan sekumpulan entitas yang memiliki hubungan dan memiliki arti antara entitas yang ada (Fikry, 2019).

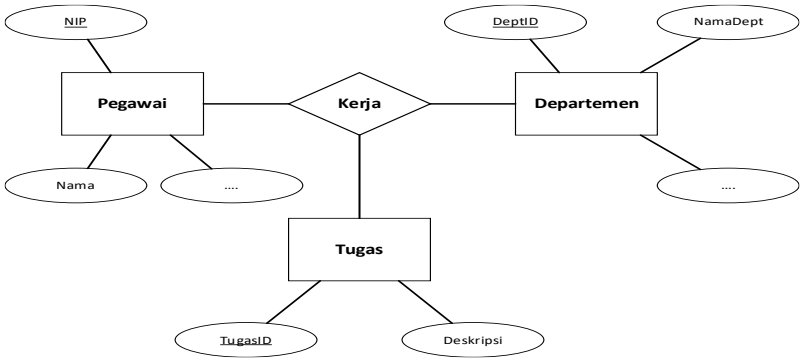
Derajat relationship merupakan jumlah himpunan entitas yang berpartisipasi dalam relationship set. Terdapat beberapa jenis derajat relationship, yaitu:

1. **Binary Relationship Set**, relasi/hubungan yang hanya melibatkan dua entitas dengan satu relasi.



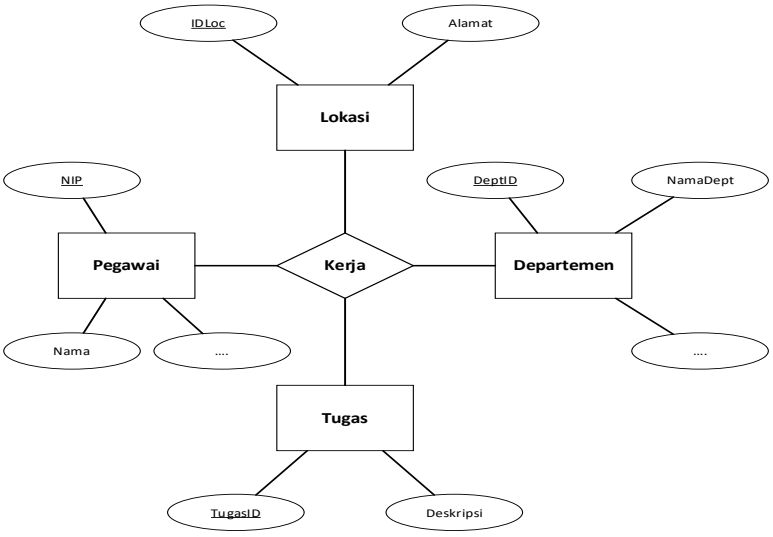
Gambar 4.7 Binary Relationship Set

2. **Ternary Relationship Set**, relasi/hubungan yang melibatkan tiga entitas dengan satu relasi.



Gambar 4.8 Ternary Relationship Set

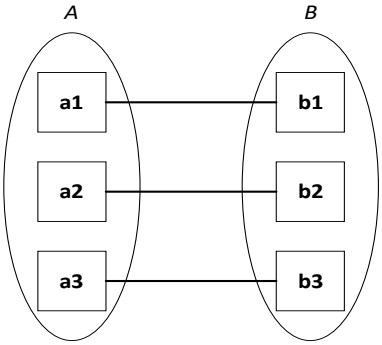
3. **N-Ary Relationship Set**, relasi/hubungan yang melibatkan lebih dari tiga atau n entitas dengan satu relasi.



Gambar 4.9 N-Ary Relationship Set

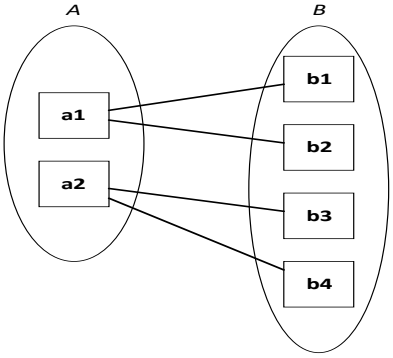
Dalam menggambarkan relasi terdapat aturan penting adalah menentukan jumlah entitas yang dapat direlasikan dengan entitas lainnya melalui relationship set yang disebut mapping cardinalities atau rasio kardinalitas (Silberschatz et al., 2019). Rasio kardinalitas paling berguna dalam menggambarkan binary relationship set, meskipun itu dapat berkontribusi pada deskripsi himpunan hubungan yang melibatkan lebih dari dua himpunan entitas. Terdapat tiga jenis rasio kardinalitas yang digunakan, yaitu:

1. Relasi satu ke satu / one to one relationship, Entitas di A dikaitkan dengan paling banyak satu entitas di B, dan entitas di B dikaitkan dengan paling banyak satu entitas di A.



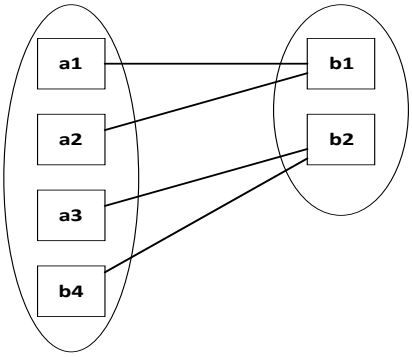
Gambar 4.10 Relasi satu ke satu

2. Relasi satu ke banyak / one to many relationship, Entitas dalam A dikaitkan dengan jumlah entitas apa pun (nol atau lebih) di B. Entitas di B, bagaimanapun, dapat dikaitkan dengan paling banyak satu entitas di A.



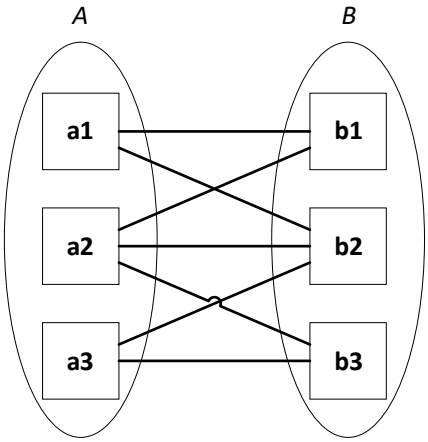
Gambar 4.11 Relasi satu ke banyak

3. Relasi banyak ke satu / many to one relationship, Entitas dalam A dikaitkan dengan paling banyak satu entitas di B. Entitas dalam B bagaimanapun dapat dikaitkan dengan angka apa pun (nol atau lebih) entitas di A.



Gambar 4.12 Relasi banyak ke satu

4. Relasi banyak ke banyak / many to many relationship, Entitas dalam A dikaitkan dengan jumlah entitas apa pun (nol atau lebih) di B, dan entitas di B dikaitkan dengan jumlah entitas apa pun (nol atau lebih) di A.



Gambar 4.13 Relasi banyak ke banyak

4.2 Relational Database Model

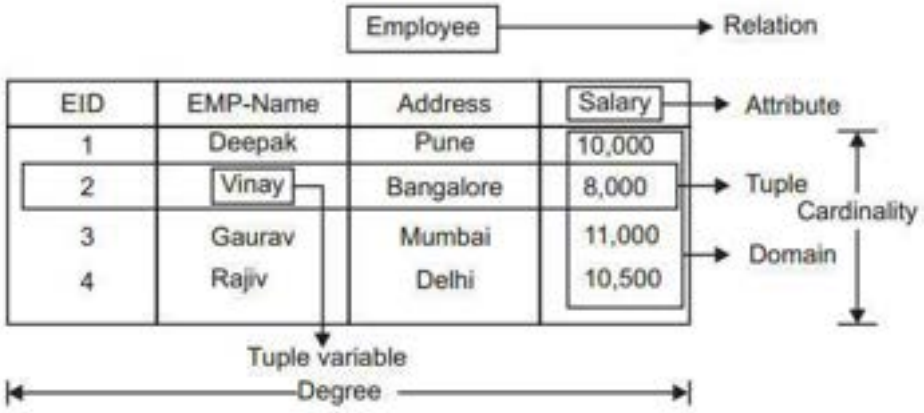
Model basis data Relasional termasuk model data yang umum digunakan, karena model ini sangat mudah dan sederhana untuk dirancang dan diimplementasikan pada tingkat logis. Tabel yang berbeda dalam database dapat dirancang menggunakan atribut dan nilai data yang sesuai dengan sangat mudah dimana Semua hubungan dirancang dengan cara tabular, yang membantu pengguna untuk berkonsentrasi pada pandangan logis database daripada detail internal yang kompleks tentang bagaimana data disimpan (Gupta & Mittal, 2017). Model relational menggunakan tabel sederhana yang terdiri dari baris dan kolom sebagai pengganti struktur pohon dan jaringan untuk menyederhanakan pandangan pengguna terhadap basis data.

Model relasional mewakili database sebagai kumpulan relasi. Secara informal, setiap relasi menyerupai tabel nilai atau dikenal dengan flat file of record. Disebut flat file karena setiap record memiliki struktur linear atau flat yang sederhana (Elmasri & Navathe, 2010). Beberapa keuntungan dari relational model, yaitu:

1. **Sederhana,** Model basis data Relasional sangat mudah dan sederhana untuk dirancang dan diimplementasikan pada tingkat logis. Tabel yang berbeda dalam database dapat dirancang menggunakan atribut dan nilai data yang sesuai dengan sangat mudah. Semua hubungan dirancang dengan cara tabular, yang membantu pengguna untuk berkonsentrasi pada pandangan logis database daripada detail internal yang kompleks tentang bagaimana data disimpan.
2. **Fleksibel,** relasional basis data menyediakan fleksibilitas yang memungkinkan perubahan pada struktur basis data dapat dengan mudah diakomodasi.
3. **Independensi Data**, Karena data berada dalam tabel, struktur database dapat diubah tanpa harus mengubah aplikasi apa pun yang didasarkan pada struktur. Jika Anda menggunakan database non relasional, Anda mungkin harus memodifikasi aplikasi yang akan mengakses informasi ini dengan menyertakan pointer ke data baru. Tetapi dengan basis data relasional, informasi tersebut segera dapat diakses karena secara otomatis terkait dengan data lain berdasarkan posisinya dalam tabel.
4. **Independensi Struktural**, Basis data relasional hanya berkaitan dengan data dan bukan dengan struktur, yang meningkatkan kinerja. Oleh karena itu waktu pemrosesan dan ruang penyimpanan relatif besar dalam database relasional tetapi pengguna tidak diharuskan untuk mengetahui detail desain struktur. Flxibility struktural dari database relasional memungkinkan kombinasi data untuk diambil yang tidak pernah diantisipasi pada saat database awalnya dirancang.
5. **Kemampuan Kueri,** Ini memungkinkan bahasa kueri tingkat tinggi yaitu, SQL (Structure Query Language) yang menghindari navigasi database yang kompleks. Dalam model ini kueri didasarkan pada hubungan logis dan memproses kueri tersebut tidak memerlukan jalur akses yang telah ditentukan sebelumnya di antara data yaitu, pointer.
6. **Teknologi Matang,** Model relasional berguna untuk mewakili sebagian besar objek dunia nyata dan hubungan di antara mereka. Implementasi hubungan sangat mudah melalui penggunaan kunci.
7. Kemampuan untuk dengan mudah memanfaatkan teknologi perangkat keras baru yang memudahkan pengguna.

Terdapat beberapa istilah yang digunakan dalam model data relational (Gupta & Mittal, 2017), yaitu:

1. Relasi, merupakan sebuah entity set dalam ER-Model yang terdiri dari beberapa baris dan beberapa kolom.
2. Tupel, satu baris pada sebuah relasi/entitas atau sering disebut record.
3. Atribut, satu kolom pada sebuah relasi/entitas atau sering disebut field.
4. Domain, kumpulan nilai yang terdapat pada satu atribut/field.
5. Degree, jumlah atribut/field dalam sebuah relasi.
6. Kardinalitas, jumlah baris/record dalam sebuah relasi.



Gambar 4.14 Istilah dalam model data relational

4.2.1 Relational Key

Dalam sebuah relation terdapat juga beberapa jenis kunci yang sering disebut dengan relational key. Relational key merupakan atribut/field yang dijadikan sebagai kunci atau key yang dapat mengidentifikasikan sebuah entitas/relasi secara unik. Terdapat lima jenis relational key, yaitu:

1. Super Key (Kunci Super), Satu atau lebih atribut yang secara unik mengidentifikasikan sebuah record dalam sebuah entitas/relasi. Untuk menentukan super key dapat dilakukan dengan memilih atribut yang bersifat unik untuk mewakili satu record/tupel.
   Contoh Super Key dari relasi pegawai: NIP, Email, NoHP atau kombinasi dari beberapa atribut dalam sebuah relasi.
2. Candidate Key (Kunci Kandidat), Satu atau lebih atribut yang secara unik dan spesifik mengidentifikasikan sebuah record dalam sebuah entitas/relasi. Menentukan candidate key dapat dilakukan dengan memilih dari beberapa atribut super key yang memiliki sifat unik dan spesifik.
   Contoh Candidate Key dari relasi pegawai: Nip dan Email.
3. Primary Key (Kunci Utama), Satu atribut yang secara unik, spesifik serta dapat mewakili satu record dalam sebuah entitas/relasi. Menentukan Kunci utama

dapat dilakukan dengan memilih dari atribut candidate key yang atributnya tidak memiliki nilai yang sama.

Contoh Primary Key dari relasi pegawai: NIP (nilai dari NIP tidak akan dimiliki oleh lebih dari satu pegawai).

4. Alternate Key (Kunci Alternatif), Satu atau lebih atribut dari candidate key yang tidak dipilih atau dijadikan sebagai kunci utama.
   Contoh Alternate Key pada relasi pegawai: Email.
5. Foreign Key (Kunci Tamu), Satu atribut dengan domain yang sama dan menjadi kunci utama pada sebuah entitas/relasi tetapi pada entitas/relasi lain sebagai atribut biasa.

4.2.2 Integrity Constrains

Dalam sistem basis data terdapat aturan atau batasan yang harus diikuti untuk menjaga data tetap konsisten, akurat dan stabil yang dikenal sebagai Integrity Constrains. Aturan yang terdapat pada integrity constrains antara lain:

1. **Entity Integrity Rule,** aturan yang diterapkan pada sebuah entitas yang menyatakan bahwa kunci utama atau bagiannya dalam sebuah relasi tidak boleh bernilai kosong (null). Sebagai contoh atribut A dalam sebuah relasi R menjadi kunci utama, maka atribut A tidak boleh bernilai kosong.
2. **Referential Integrity Rule,** aturan yang mengatur hubungan antara kunci utama untuk menjaga konsistensi data antara tabel yang saling ber-relasi. Atau dapat dijelaskan juga bahwa Kunci tamu dapat berupa null atau hanya dapat memiliki nilai yang ada dalam kunci utama yang ber-relasi dengannya. Misalkan A menjadi atribut dalam relasi R1, yang juga merupakan kunci utama dalam relasi R2, maka nilai A dalam R1 adalah null atau sama dengan dalam relasi R2.
3. **Domain Constrains,** aturan ini diterapkan pada setiap nilai atribut. Dengan mengikuti batasan ini dapat menjaga konsistensi dalam basisdata. Aturan ini termasuk tipe data (bilangan bulat, varchar, char, format waktu, format tanggal, dll.), Ukuran variabel, cek (seperti nilai bukan nol, dll.) dll.
4. **Key Constrains,** Pada sebuah relasi R, jika atribut A merupakan kunci utama maka A harus memiliki nilai yang unik atau dapat disebut juga bahwa atribut kunci utama harus memiliki nilai yang unik.
5. **Tuple Uniqueness Constrains,** Pada sebuah relasi R, seluruh tupel pada relasi R harus memiliki nilai nyata, dengan kata lain tupel yang sama dalam sebuah relasi tidak perbolehkan.

Bab 5
Normalisasi

5.1 Tujuan Normalisasi

Normalisasi adalah sebuah teknik untuk menghasilkan sekumpulan relasi dengan sifat-sifat yang diinginkan, berdasarkan kebutuhan data dari sebuah perusahaan. Tujuan normalisasi adalah untuk mengidentifikasi sekumpulan relasi yang sesuai yang mendukung kebutuhan data suatu perusahaan. Basis Data yang baik adalah basis data yang tidak memiliki redudansi data, sehingga proses normalisasi dapat dilakukan untuk meminimalisasi redudansi data.

5.2 Redundansi Data dan Update Anomali

Tujuan utama dari desain database relasional adalah mengelompokkan atribut ke dalam relasi untuk meminimalkan redundansi data. Jika tujuan ini tercapai, manfaat potensial dari database yang diimplementasikan adalah sebagai berikut:

1. *Update* data yang disimpan dalam database dicapai dengan jumlah operasi yang minimal, sehingga mengurangi peluang terjadinya ketidakkonsistenan data dalam database;
2. Pengurangan ruang penyimpanan file yang dibutuhkan oleh relasi dasar sehingga meminimalkan biaya.

Basis data relasional juga bergantung pada keberadaan redundansi data dalam jumlah tertentu. Redundansi ini berupa salinan *primary key* (atau *candidate key*) yang bertindak sebagai *foreign key* dalam hubungan terkait untuk memungkinkan pemodelan relasi antar data.

Pada bagian ini kami mengilustrasikan masalah yang terkait dengan redundansi data yang tidak diinginkan dengan membandingkan relasi Karyawan dan Cabang yang ditunjukkan pada Tabel 5.1 dan Tabel 5.2 dengan relasi CabangKaryawan yang ditunjukkan pada Tabel 5.3. Relasi CabangKaryawan adalah format alternatif dari relasi Karyawan dan Cabang. Relasi memiliki bentuk sebagai berikut:

Karyawan (NoKaryawan, NamaKaryawan, Posisi, Gaji, NoCabang)
Cabang (NoCabang, AlamatCabang)
CabangKaryawan (NoKaryawan, NamaKaryawan, Posisi, Gaji, NoCabang, AlamatCabang)

**Primary key* untuk setiap relasi digarisbawahi.

Tabel 5.1 Karyawan

NoKaryawan	NamaKaryawan	Posisi	Gaji	NoCabang
SL21	Biggest Rock	Manager	30000	B006
SL37	Anton Lee	Assistant	12000	B004
SG14	John White	Supervisor	18000	B004
SA9	Sarah Dubin	Assistant	9000	B007
SG5	Yulia Jr	Manager	24000	B004
SL41	Bagus Priambodo	Assistant	9000	B006

Tabel 5.2 Cabang

NoCabang	AlamatCabang
B006	Pacitan, Jawa Timur
B007	Tangerang, Banten
B004	Bandung, Jawa Barat

Tabel 5.3 CabangKaryawan

NoKaryawan	NamaKaryawan	Posisi	Gaji	NoCabang	AlamatCabang
SL21	Biggest Rock	Manager	30000	B006	Pacitan, Jawa Timur
SL37	Anton Lee	Assistant	12000	B004	Bandung, Jawa Barat
SG14	John White	Supervisor	18000	B004	Bandung, Jawa Barat
SA9	Sarah Dubin	Assistant	9000	B007	Tangerang, Banten
SG5	Yulia Jr	Manager	24000	B004	Bandung, Jawa Barat
SL41	Bagus Priambodo	Assistant	9000	B006	Pacitan, Jawa Timur

Dalam relasi CabangKaryawan terdapat data redundan detail dari cabang berulang setiap karyawan di cabang tersebut. Sebaliknya, detail cabang hanya muncul satu kali untuk setiap cabang dalam relasi cabang, dan hanya nomor cabang (NoCabang) yang berulang dalam relasi Karyawan untuk menunjukkan setiap Karyawan. Relasi yang memiliki redundan data memiliki masalah yang disebut **update anomali**. Adapun update anomali dapat diklasifikasikan menjadi anomali *insertion, deletion,* atau *modification*.

1. Anomali Insertion

Ada dua jenis anomali *insertion*, yang diilustrasikan menggunakan relasi CabangKaryawan yang ditunjukkan pada Tabel 5.3:

a. Untuk memasukkan detail karyawan baru ke dalam relasi CabangKaryawan, kita harus menyertakan detail cabang di mana karyawan akan ditempatkan.

Misalnya, untuk menyisipkan detail karyawan baru yang berlokasi di nomor cabang B007, kita harus memasukkan detail nomor cabang B007 yang benar sehingga detail cabang konsisten dengan nilai cabang B007 di tuple lain dari relasi CabangKaryawan.

b. Untuk memasukkan detail cabang baru yang saat ini tidak memiliki karyawan ke dalam relasi CabangKaryawan, Anda perlu memasukkan *null* ke dalam atribut karyawan. Akan tetapi hal ini tidak diperbolehkan karena akan melanggar integritas entitas dikarenakan NoKaryawan adalah primary key untuk relasi CabangKaryawan. Oleh karena itu, kita tidak dapat memasukkan tuple untuk cabang baru ke dalam relasi CabangKaryawan dengan null untuk NoKaryawan. Rancangan relasi yang ditunjukkan pada Tabel 5.1 dan Tabel 5.2 menghindari masalah ini, karena detail cabang dimasukkan ke dalam relasi Cabang secara terpisah dari detail Karyawan. Detail dari Karyawan yang pada akhirnya berada di cabang tersebut kemudian dimasukkan ke dalam relasi Karyawan.

2. Anomali *Deletion*

Jika kita menghapus sebuah *tuple* dari relasi CabangKaryawan yang merepresentasikan anggota karyawan yang terletak di sebuah cabang, detail tentang cabang tersebut juga hilang dari database. Sebagai contoh, jika kita menghapus *tuple* untuk NoKaryawan SA9 (Sarah Dubin) dari relasi CabangKaryawan, detail terkait nomor cabang B007 akan hilang dari database. Rancangan relasi pada Tabel 5.1 dan Tabel 5.2 menghindari masalah ini, karena tupel cabang disimpan secara terpisah dari tupel karyawan dan hanya atribut NoCabang yang menghubungkan kedua relasi. Jika kita menghapus tuple untuk nomor staf SA9 dari relasi Karyawan, detail cabang nomor B007 tetap tidak terpengaruh dalam relasi Cabang

3. Anomali *Modification*

Jika kita ingin mengubah nilai pada atribut cabang tertentu dalam relasi CabangKaryawan (misalnya, alamat cabang nomor B004), maka kita harus memperbarui tupel semua karyawan yang terletak di cabang itu. Jika modifikasi ini tidak dilakukan pada semua tupel yang sesuai dari relasi CabangKaryawan, database akan menjadi tidak konsisten. Dalam contoh ini, nomor cabang B004 mungkin tampak memiliki alamat yang berbeda di tupel karyawan yang berbeda.

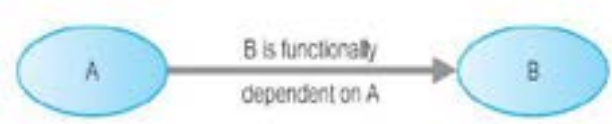
Contoh sebelumnya mengilustrasikan bahwa relasi Karyawan dan Cabang pada Tabel 5.1 dan Tabel 5.2 memiliki sifat yang lebih diinginkan daripada relasi CabangKaryawan pada Tabel 5.3. Hal ini menunjukkan bahwa meskipun relasi CabangKaryawan menjadi persoalan terhadap update anomali, kita dapat menghindari anomali ini dengan mendekomposisi relasi asli ke dalam relasi Karyawan dan Cabang.

5.3 Ketergantungan Fungsional

Ketergantungan fungsional merupakan konsep penting terkait dengan normalisasi yang menggambarkan hubungan antar atribut. Pada bagian ini akan dijelaskan mengenai ketergantungan fungsional dan kemudian fokus pada karakteristik khusus dari ketergantungan fungsional yang berguna untuk normalisasi.

5.3.1 Karakteristik dari Ketergantungan Fungsional

Untuk diskusi tentang ketergantungan fungsional, asumsikan bahwa skema relasional memiliki atribut (A, B, C, . . . , Z) dan database dijelaskan oleh relasi universal tunggal yang disebut R = (A, B, C, . . . , Z). Asumsi ini berarti bahwa setiap atribut dalam database memiliki nama yang unik.



Gambar 5.1 diagram ketergantungan fungsional.

Ketergantungan fungsional menggambarkan hubungan antar atribut dalam suatu relasi. Dengan mempertimbangkan relasi antara atribut A dan B, di mana atribut B secara fungsional bergantung pada atribut A. Jika kita mengetahui nilai A dan kita memeriksa relasi yang memiliki ketergantungan ini, kita hanya menemukan satu nilai B di semua tupel yang diberikan nilai A. Ketergantungan antara atribut A dan B dapat direpresentasikan seperti Gambar 5.1.

Namun, ada jenis ketergantungan fungsional tambahan yang disebut ketergantungan transitif, karena keberadaannya dalam suatu relasi berpotensi menyebabkan jenis *update* anomali. Ketergantungan transitif adalah Suatu kondisi di mana A, B, dan C adalah atribut dari suatu relasi sehingga jika A ® B dan B ® C, maka C bergantung secara transitif pada A melalui B (asalkan A tidak bergantung secara fungsional pada B atau C). Contoh ketergantungan transitif diberikan dalam Contoh 5.1.

Contoh 5.1 Contoh dari Ketergantungan transitive functional

Pertimbangkan ketergantungan fungsional berikut dalam relasi CabangKaryawan yang ditunjukkan pada Tabel 5.3:

NoKaryawan ® NamaKaryawan, Posisi, Gaji, NoCabang, AlamatCabang

NoCabang ® AlamatCabang

Ketergantungan transitif NoCabang ® AlamatCabang ada di NoKaryawan melalui NoCabang. Dengan kata lain, atribut NoKaryawan secara fungsional menentukan AlamatCabang melalui atribut NoCabang, dan NoCabang maupun AlamatCabang tidak secara fungsional menentukan NoKaryawan.

5.3.2 Ketergantungan Fungsional

Mengidentifikasi semua dependensi fungsional antara satu set atribut seharusnya cukup sederhana jika arti dari setiap atribut dan hubungan antar atribut dipahami dengan baik. Jenis informasi ini dapat disediakan oleh perusahaan dalam bentuk diskusi dengan pengguna dan/atau dokumentasi yang sesuai, seperti spesifikasi kebutuhan pengguna. Namun, jika pengguna tidak tersedia untuk konsultasi dan/atau dokumentasi tidak lengkap, maka—bergantung pada aplikasi database—perancang database mungkin perlu menggunakan akal sehat dan/atau pengalaman mereka untuk menyediakan informasi yang hilang. Contoh 5.2 mengilustrasikan betapa mudahnya untuk mengidentifikasi ketergantungan fungsional antara atribut dari suatu relasi ketika tujuan dari setiap atribut dan relasi atribut dipahami dengan baik.

Contoh 5.2 Mengidentifikasi ketergantungan fungsional untuk relasi CabangKaryawan

Kita mulai dengan memeriksa semantik atribut dalam relasi CabangKaryawan yang ditunjukkan pada Tabel 5.3. Pada contoh ini diasumsikan bahwa posisi dan cabang menentukan gaji anggota karyawan. Dari relasi tersebut dapat diidentifikasikan ketergantungan fungsional sebagai berikut:

NoKaryawan ® NamaKaryawan, Posisi, Gaji, NoCabang, AlamatCabang
NoCabang ® AlamatCabang
AlamatCabang ® NoCabang
NoCabang, Posisi ® Gaji
AlamatCabang, Posisi ® Gaji

Ada 5 ketergantungan fungsional dalam relasi CabangKaryawan dengan NoKaryawan, NoCabang, AlamatCabang, (NoCabang, Posisi), dan (AlamatCabang, Posisi) sebagai determinan. Untuk setiap ketergantungan fungsional, semua atribut di sisi kanan secara fungsional bergantung pada determinan di sisi kiri.

5.4 Mengidentifikasi Primary Key Menggunakan Functional Dependency

Tujuan utama mengidentifikasi ketergantungan fungsional untuk suatu relasi adalah untuk menentukan *integrity constraints* yang harus dipegang pada suatu relasi. *Integrity constraints* yang penting untuk dipertimbangkan terlebih dahulu adalah identifikasi *candidate key*, salah satunya dipilih menjadi *primary key* untuk relasi. Berikut ini merupakan contoh untuk mendidentifikasi primary key untuk relasi tertentu.

Contoh 5.3 Mengidentifikasi primary key untuk relasi CabangKaryawan

Dalam contoh 5.2 dijelaskan identifikasi 5 ketergantungan fungsional untuk relasi CabangKaryawan yang ditunjukkan pada Tabel 5.3. Determinan untuk ketergantungan fungsional ini adalah NoKaryawan, NoCabang, AlamatCabang, (NoCabang, Posisi), dan (AlamatCabang, Posisi).

Untuk mengidentifikasi *candidate key* untuk relasi CabangKaryawan, kita harus mengidentifikasi atribut yang secara unik mengidentifikasi setiap tupel dalam relasi ini. Jika suatu relasi memiliki lebih dari satu *candidate key*, maka akan diindetifikasi *candidate key* yang bertindak sebagai *primary key* untuk relasi tersebut. Semua atribut yang bukan merupakan bagian dari *primary key* (atribut *non-primary-key*) harus bergantung secara fungsional pada *key* tersebut.

Satu-satunya *candidate key* dari relasi CabangKaryawan, dan karena itu *primary key*, adalah NoKaryawan, karena semua atribut lain dari relasi secara fungsional bergantung pada NoKaryawan. Meskipun NoCabang, AlamatCabang, (NoCabang, Posisi), dan (*bAddress, position*) adalah determinan dalam relasi ini, mereka bukanlah *candidate key* untuk relasi tersebut.

Pada bagian ini telah dibahas mengenai ketergantungan ketergantungan fungsional yang paling berguna dalam mengidentifikasi kendala-kendala yang penting pada suatu relasi dan bagaimana ketergantungan tersebut dapat digunakan untuk mengidentifikasi *primary key* (atau *candidate key*) untuk relasi tertentu. Adapun konsep ketergantungan fungsional dan key (*primary key* atau *candidate key*) adalah inti dari proses normalisasi.

5.5 Proses Normalisasi

Normalisasi adalah suatu teknik untuk menganalisis hubungan berdasarkan *primary key* (atau *candidate key*) dan ketergantungan fungsionalnya. Teknik tersebut melibatkan serangkaian aturan yang dapat digunakan untuk menguji relasi individu sehingga database dapat dinormalisasi pada tingkatan apapun. Ketika persyaratan tidak terpenuhi, relasi yang melanggar persyaratan harus didekomposisi menjadi relasi yang secara individual memenuhi persyaratan normalisasi.

Normalisasi terdiri dari beberapa langkah, dan setiap langkah sesuai dengan bentuk normal tertentu dan memiliki sifat masing-masing. Untuk model data relasional, hanya *First Normal Form* (1NF) yang penting dalam membuat relasi; semua bentuk normal selanjutnya adalah opsional. Akan tetapi untuk menghindari *update* anomali, maka disarankan agar melanjutkan setidaknya sampai *Third Normal Form* (3NF).

Pada pembahasan kali ini kita akan mengekstraksi informasi tentang atribut dari formulir yang pertama kali diubah menjadi format tabel, yang dideskripsikan sebagai *Unnormalized Form* (UNF). Untuk menyederhanakan deskripsi normalisasi, kami mengasumsikan bahwa sekumpulan ketergantungan fungsional diberikan untuk setiap

relasi dalam contoh yang dikerjakan dan bahwa setiap relasi memiliki *primary key* yang ditunjuk. Dengan kata lain, penting bahwa arti dari atribut dan hubungannya dipahami dengan baik sebelum memulai proses normalisasi. Informasi ini sangat mendasar untuk normalisasi dan digunakan untuk menguji apakah suatu relasi berada dalam bentuk normal tertentu.

5.5.1 Bentuk Normal Pertama (1NF)

Bentuk Tidak Normal (UNF)
Suatu keadaan dimana sebuah tabel memiliki satu atau lebih data yang berulang.

Bentuk Normal Pertama (1NF)
Suatu relasi dimana atribut dari relasi tersebut hanya memiliki nilai tunggal dan tidak ada pengulangan grup atribut dalam baris.

Kita akan memulai proses normalisasi dengan terlebih dahulu mentransfer data dari sumbernya ke dalam format tabel yang terdiri dari baris dan kolom. Dalam format ini, tabel dalam bentuk yang tidak dinormalisasi dan disebut sebagai tabel yang tidak dinormalisasi. Untuk mengubah tabel yang tidak dinormalisasi menjadi Bentuk Normal Pertama, maka perlu untuk mengidentifikasi dan menghapus *repeating group* di dalam tabel. *Repeating group* adalah atribut, atau sekumpulan atribut, di dalam tabel yang muncul dengan banyak nilai untuk satu kemunculan *key attribute*. Didalam konteks ini, istilah "*key*" mengacu pada atribut yang secara unik mengidentifikasi setiap baris dalam tabel yang tidak dinormalisasi.

Tabel 5.4 Bentuk Tabel Tidak Normal dari RentalKlien

NoKli	NoProp	NamaKli	AlmtProp	TglPinj	TglSlsi	HrgSw	NoPmlk	NPmlk
CR76	PG4	Joko Gunawan	Medan, Indonesia	1-7-12	31-8-13	350	CO40	Tina Novi
	PG16		Denpasar, Indonesia	1-9-13	1-9-14	450	CO93	Tony Shaw
CR56	PG4	Alicia May	Medan, Indonesia	1-9-11	10-6-12	350	CO40	Tina Novi
	PG36		Jambi, Indonesia	10-10-12	1-12-13	375	CO93	Tony Shaw
	PG16		Denpasar, Indonesia	1-11-14	10-8-15	450	CO93	Tony Shaw

Penulis mengidentifikasi atribut *key* untuk tabel RentalKlien yang tidak dinormalisasi sebagai NoKli. Selanjutnya, Penulis mengidentifikasi *repeating group* dalam tabel yang tidak dinormalisasi sebagai detail sewa properti. Struktur *repeating group* = (NoProp, AlmtProp, TglPinj, TglSlsi, HrgSw, NoPmlk, NPmlk).

Tabel 5.5 Bentuk Normal Pertama dari RentalKlien

NoKli	NoProp	NamaKli	AlmtProp	TglPinj	TglSlsi	HrgSw	NoPmlk	NPmlk
CR76	PG4	Joko Gunawan	Medan, Indonesia	1-7-12	31-8-13	350	CO40	Tina Novi
CR76	PG16	Joko Gunawan	Denpasar, Indonesia	1-9-13	1-9-14	450	CO93	Tony Shaw
CR56	PG4	Alicia May	Medan, Indonesia	1-9-11	10-6-12	350	CO40	Tina Novi
CR56	PG36	Alicia May	Jambi, Indonesia	10-10-12	1-12-13	375	CO93	Tony Shaw
CR56	PG16	Alicia May	Denpasar, Indonesia	1-11-14	10-8-15	450	CO93	Tony Shaw

Untuk mengubah tabel yang tidak dinormalisasi menjadi 1NF, maka diharuskan untuk menghilangkan *repeating group*. Dengan pendekatan pertama, kita menghilangkan *repeating group* dengan memasukkan data klien yang sesuai ke setiap baris. Bentuk relasi normal pertama yang dihasilkan RentalKlien ditunjukkan pada Tabel 5.5.

5.5.2 Bentuk Normal Kedua (2NF)

Bentuk Normal Kedua (2NF) didasarkan pada konsep ketergantungan fungsional penuh. Bentuk normal kedua berlaku untuk relasi dengan *composite key*, yaitu relasi dengan *primary key* yang terdiri dari dua atau lebih atribut. Relasi dengan *single-attribute primary key* secara otomatis setidaknya berada dalam 2NF. Relasi yang tidak ada dalam 2NF mungkin mengalami *update* anomali. Misalnya, kita ingin mengubah nomor sewa properti PG4. Kita harus memperbarui dua tupel dalam relasi RentalKlien pada Tabel 5.5. Jika hanya satu tupel yang diperbarui dengan sewa baru, hal ini akan mengakibatkan ketidakkonsistenan dalam database.

Bentuk Normal Kedua (2NF)

Relasi yang berada dalam bentuk normal pertama dan setiap atribut *non-primary-key* sepenuhnya bergantung secara fungsional pada primary key.

Normalisasi relasi 1NF ke 2NF melibatkan penghilangan ketergantungan parsial. Aturan normalisasi kedua (2NF) dapat dikatakan bahwa sebuah relasi dalam bentuk normal pertama dan setiap atribut bukan primary key yang tergantung secara fungsional kepada primary key. Pada contoh 5.4 merupakan proses konversi relasi 1NF ke 2NF.

Contoh 5.4 Bentuk Normal Kedua (2NF)

Berikut ini merupakan ketergantungan fungsional dari relasi RentalKlien:

fd1 NoKli, NoProp ® TglPinj, TglSlsi (Primary key)
fd2 NoKli ® NamaKli (Ketergantungan Parsial)
fd3 NoProp ® AlmtProp, HrgSw, NoPmlk, NPmlk (Ketergantungan Parsial)
fd4 NoPmlk ® NPmlk (Ketergantungan Transitif)
fd5 NoKli, TglPinj ® NoProp, AlmtProp, TglSlsi,
HrgSw, NoPmlk, NPmlk (Candidate key)
fd6 NoProp, TglPinj ® NoKli, NamaKli, TglSlsi (Candidate key)

Tabel 5.6 Relasi Bentuk Normal Kedua yang diturunkan dari relasi RentalKlien

NoKli	NamaKli
CR76	Joko Gunawan
CR56	Alicia May

NoKli	NoProp	TglPinj	TglSlsi
CR76	PG4	1-7-12	31-8-13
CR76	PG16	1-9-13	1-9-14
CR56	PG4	1-9-11	10-6-12
CR56	PG36	10-10-12	1-12-13
CR56	PG16	1-11-14	10-8-15

NoProp	AlmtProp	HrgSw	NoPmlk	NPmlk
PG4	Medan, Indonesia	350	CO40	Tina Novi
PG16	Denpasar, Indonesia	450	CO93	Tony Shaw
PG4	Medan, Indonesia	350	CO40	Tina Novi
PG36	Jambi, Indonesia	375	CO93	Tony Shaw
PG16	Denpasar, Indonesia	450	CO93	Tony Shaw

Penulis melanjutkan proses normalisasi relasi RentalKlien dengan menggunakan ketergantungan fungsional. Penulis mulai dengan menguji apakah relasi RentalKlien berada dalam 2NF dengan mengidentifikasi adanya ketergantungan parsial pada *primary key*. Pada atribut klien (NamaKli) tergantung secara parsial pada *primary key*, dengan kata lain, hanya pada atribut NoKli (diwakili sebagai fd2). Atribut properti (AlmtProp, HrgSw, NoPmlk, NPmlk) sebagian bergantung pada *primary key*, yaitu, hanya pada atribut NoProp (diwakili sebagai fd3). Atribut sewa properti (TglPinj dan TglSlsi) bergantung secara penuh pada seluruh *primary key*; yaitu atribut NoKli dan NoProp (diwakili sebagai fd1).

Identifikasi ketergantungan dependensi dalam relasi RentalKlien menunjukkan bahwa relasi tersebut bukan dalam 2NF. Untuk mengubah relasi RentalKlien menjadi 2NF memerlukan pembuatan relasi baru sehingga atribut *non-primary-key* dihapus bersama dengan salinan bagian dari *primary key* di mana mereka sepenuhnya bergantung secara fungsional. Proses ini akan menghasilkan tiga relasi baru yang disebut Klien, Rental, dan PemilikProperti, seperti yang ditunjukkan pada Tabel 5.6. Ketiga relasi ini berada dalam bentuk normal kedua, karena setiap atribut *non-primary-key* secara

fungsional bergantung penuh pada relasi *primary key*. Adapun bentuk relasinya sebagai berikut:

Klien (NoKli, NamaKli)
Rental (NoKli, NoProp, TglPinj, TglSlsi)
PemilikProperti (NoProp, AlmtProp, HrgSw, NoPmlk, NPmlk)

5.5.3 Bentuk Normal Ketiga (3NF)

Meskipun relasi 2NF memiliki redundansi yang lebih sedikit daripada relasi 1NF, relasi tersebut mungkin masih mengalami anomali pembaruan. Sebagai contoh, jika kita ingin mengupdate nama pemilik, seperti Tony Shaw (NoPmlk CO93), kita harus mengupdate dua tupel pada relasi PemilikProperti pada Tabel 5.6. Jika kita hanya memperbarui satu tupel, maka akan terjadi ketidakkonsistenan data. Oleh karena itu kita perlu menghilangkan ketergantungan tersebut dengan melanjutkan proses normalisasi ke bentuk normal ketiga.

Bentuk Normal Ketiga (3NF)
Relasi yang berada dalam bentuk normal pertama dan kedua dan di mana tidak ada atribut *non-primary-key* yang bergantung secara transitif pada *primary key*.

Normalisasi relasi 2NF ke 3NF melibatkan penghapusan ketergantungan transitif. Jika ada ketergantungan transitif, kami menghapus atribut yang bergantung secara transitif dari relasi dengan menempatkan atribut dalam relasi baru bersama dengan salinan determinan. Berikut ini merupakan contoh konversi relasi 2NF menjadi relasi 3NF.

Contoh 5.5 Third Normal Form (3NF)
Berikut ini merupakan ketergantungan fungsional untuk relasi Klien, Rental, dan PemilikProperti yang didapatkan dari Contoh 5.4.

Klien
fd2 NoKli ® NamaKli (Primary key)
Rental
fd1 NoKli, NoProp ® TglPinj, TglSlsi (Primary key)
fd5' NoKli, TglPinj ® NoProp, TglSlsi (Candidate key)
fd6' NoProp, TglPinj ® NoKli, TglSlsi (Candidate key)
PemilikProperti
fd3 NoProp ® AlmtProp, HrgSw, NoPmlk, NPmlk (Primary key)
fd4 NoPmlk ® NPmlk (Transitive dependency)

Semua atribut *non-primary-key* dalam relasi Klien dan Rental secara fungsional hanya bergantung pada *primary key*. Relasi Klien dan Sewa tidak memiliki ketergantungan transitif.

Semua atribut *non-primary-key* dalam relasi PemilikProperti secara fungsional

bergantung pada *primary key*, kecuali NPmlk, yang bergantung secara transitif pada NoPmlk (direpresentasikan sebagai fd4). Untuk mengubah relasi PemilikProperti menjadi 3NF, pertama-tama kita harus menghilangkan ketergantungan transitif ini dengan membuat dua relasi baru yang disebut PropertiDisewakan dan Pemilik.

PropertiDisewakan (NoProp, AlmtProp, HrgSw, NPmlk)

Pemilik (NoPmlk, NPmlk)

Relasi PropertiDisewakan dan Pemilik berada dalam 3NF, karena tidak ada ketergantungan transitif pada primary key. Relasi RentalKlien diubah oleh proses normalisasi menjadi empat relasi dalam 3NF. Berikut ini merupakan relasi 3NF yang dihasilkan:

Client (NoKli, NamaKli)

Rental (NoKli, NoProp, TglPinj, TglSlsi)

PropertiDisewakan (NoProp, AlmtProp, HrgSw, NPmlk)

Pemilik (NoPmlk, NPmlk)

Bab 6
Bahasa Basis Data

6.1 Data Definition Language

Data Definition Language (DDL) (Turner, 2020) digunakan untuk mendefinisikan database baru, tabel data baru, menghapus database, menghapus tabel data, dan mengubah struktur tabel data dengan kata kunci berikut; CREATE, ALTER, DROP, dan TRUNCATE (Ramakrishnan & Gehrke, 2014). Dalam bab 6 ini, akan dibahas secara praktis tentang Data Definition Language (DDL) SQL.

6.1.1 Perintah CREATE

Perintah CREATE akan digunakan untuk membuat beberapa kegiatan, diantaranya Create Database, Create Table.

1. **Membuat Database**

   Berikut bahasa sintaks yang digunakan untuk membuata database, seperti dibawah ini.

   CREATE DATABASE nama_database;

   Sebagai contoh membuat database untuk latihan didalam perangkat lunak sistem manajemen database MariaDB.

   CREATE DATABASE dbbab6;

   Berikut penerapan pada command line atau pada shell windows, yang sangat mudah untuk anda gunakan.

```
MariaDB [(none)]> CREATE DATABASE dbbab6;
Query OK, 1 row affected (0.002 sec)
```

2. **Membuat Tabel**

   Berikut bahasa sintaks yang digunakan untuk membuata tabele, seperti dibawah ini.

   CREATE TABLE nama_tabel
   (
   tabel_kolom-1 tipe_data,

```
    tabel_kolom-2 tipe_data,
    tabel_kolom-3 tipe_data,
    tabel_kolom-4 tipe_data,
    tabel_kolom-N tipe_data
);
```

Sebelum anda membuat tabel, harus anda perhatikan adalah mengaktifkan database yang sudah dibuat dengan perintah USE diikuti dengan nama_database yang telah anda buat.

```
MariaDB [(none)]> USE dbbab6;
Database changed
```

Sebagai contoh membuat tabel untuk latihan didalam perangkat lunak sistem manajemen database MariaDB. Untuk latihan pertama membuat tabel dengan nama Karyawan, dengan struktur seperti Tabel 6.1 dibawah ini.

Tabel 6.1 Sturktur Tabel Karyawan

Nama Field / Nama Kolom	Tipe Data	Ukuran	Kunci Field
NoKaryawan	VarChar	6	Primary
NamaKaryawan	VarChar	50	
AlamatKaryawan	Text		
NomorTelepon	VarChar	15	
EmailKaryawan	VarChar	25	

Berikut perintah SQL dalam penerapan untuk membuat tabel

```
CREATE TABLE karyawan
(
    NoKaryawan varchar(6) PRIMARY KEY,
    NamaKaryawan varchar(50),
    AlamatKaryawan text,
    NomorTelepon varchar(15),
    EmailKaryawan varchar(25)
);
```

Setiap tabel harus memiliki field kunci untuk mengidentifikasi secara unik setiap catatan tabel. Key field adalah field dalam record tabel data yang berisi data unik dan memisahkan record itu dari semua record lain dalam database. Key field yang biasa digunakan dalam pembuatan database, yaitu Primary Key dan Foreign Key.

Atribut Primary Key dipastikan diletakan dibelakang field/kolom NoKaryawan dikarenakan sebagai field yang unik. Akan tetapi Primary Key itu dapat diletakan

pada field mana saja yang akan dijadikan kunci field. Atribut kolom lainnya adalah "NOT NULL" yang memastikan bahwa nilai NULL tidak diterima ke dalam kolom, dan "Foreign Key" yang memastikan record terkait di tabel lain tidak salah dalam penggabungan. Berikut penerapan pada command line atau pada shell windows.

```
MariaDB [dbbab6]> CREATE TABLE karyawan
    -> (
    -> NoKaryawan varchar(6) PRIMARY KEY,
    -> NamaKaryawan varchar(50),
    -> AlamatKaryawan text,
    -> NomorTelepon varchar(15),
    -> EmailKaryawan varchar(25)
    -> );
Query OK, 0 rows affected (0.052 sec)
```

Titik koma wajib di akhir pernyataan digunakan untuk memproses setiap perintah sebelumnya. Dalam contoh ini, string VARCHAR digunakan untuk menentukan tipe data dan ukuran field. Tipe data lainnya dapat berupa DATE, NUMBER, TEXT atau INTEGER.

3. Mengubah Tabel

Untuk melakukan mengubah struktur tabel yang telah dibuat dapat gunakan perintah ALTER. Beberapa hal yang dapat dilakukan oleh perintah ini yaitu: menambah field (Add), mengganti nama field (Change) ataupun mengubah nama field (Rename), dan menghapus field (Drop). Dengan contoh ALTER TABLE nama_tabel ADD nama_kolom datatype.

a. Tambahkan Kolom ke Tabel

Untuk menambahkan kolom ke tabel, menggunakan sintaks ALTER TABLE ADD:

ALTER TABLE table_name
ADD
new_column_name field_definition
[FIRST | AFTER nama_kolom]

Dalam sintaks ini:

1) table_name untuk tentukan nama tabel yang ingin ditambahkan kolom atau kolom baru setelah kata kunci ALTER TABLE.
2) new_field_name untuk tentukan nama field baru.
3) kolom _definition untuk tentukan tipe data, ukuran maksimum, dan batasan field dari field baru.
4) FIRST | AFTER kolom _name tentukan posisi kolom baru dalam tabel. Anda dapat menambahkan kolom setelah kolom yang ada (nama_kolom ATER)

atau sebagai kolom pertama (FIRST). Jika Anda menghilangkan klausa ini, field ditambahkan di akhir daftar kolom tabel.

Contoh berikut menggunakan pernyataan ALTER TABLE ADD untuk menambahkan kolom di akhir tabel karyawan:

ALTER TABLE karyawan ADD KotaLahir VARCHAR(50) AFTER NamaKaryawan;

Pernyataan ini menunjukkan daftar kolom dari tabel karyawan:

DESCRIBE karyawan;

```
MariaDB [dbbab6]> DESCRIBE karyawan;
+----------------+-------------+------+-----+---------+-------+
| Field          | Type        | Null | Key | Default | Extra |
+----------------+-------------+------+-----+---------+-------+
| NoKaryawan     | varchar(6)  | NO   | PRI | NULL    |       |
| NamaKaryawan   | varchar(50) | YES  |     | NULL    |       |
| KotaLahir      | varchar(50) | YES  |     | NULL    |       |
| AlamatKaryawan | text        | YES  |     | NULL    |       |
| NomorTelepon   | varchar(15) | YES  |     | NULL    |       |
| EmailKaryawan  | varchar(25) | YES  |     | NULL    |       |
+----------------+-------------+------+-----+---------+-------+
6 rows in set (0.018 sec)
```

Gambar 6.1 Describe Tabel Karyawan

b. Ubah field tabel

Untuk mengubah kolom pada tabel, menggunakan sintaks ALTER TABLE RENAME:

ALTER TABLE old_table_name RENAME new_table_name;

Penjelasan sintaks ini:

1) ALTER TABLE adalah perintah dasar untuk mengubah struktur tabel old_table_name adalah Nama dari tabel yang akan diubah kolomnya.
2) RENAME adalah perintah untuk mengganti nama tabel nama_tabel_lama adalah nama tabel lama yang ingin diganti nama_tabel_baru adalah Nama tabel baru yang berjumlah maximal 50 karakter, bisa

Sebagai contoh, diberikan perintah berikut:

ALTER TABLE karyawan RENAME TO tbkaryawan;

```
MariaDB [dbbab6]> ALTER TABLE karyawan RENAME TO tbkaryawan;
Query OK, 0 rows affected (0.025 sec)

MariaDB [dbbab6]> SHOW TABLES;
+------------------+
| Tables_in_dbbab6 |
+------------------+
| authors          |
| buku             |
| jobs             |
| pekerja          |
| penerbit         |
| penjualan        |
| tbkaryawan       |
| tmppenerbit      |
+------------------+
8 rows in set (0.001 sec)
```

Gambar 6.2 Nama Tabel Karyawan berubah nama

Mengubah nama field atau nama kolom dalam suatu tabel, maka sintaks penulisannya dengan menggunakan perintah CHANGE, seperti berikut:

ALTER TABLE table_name RENAME COLUMN old_column_name TO new_column_name;

Penjelasan sintaks ini:

1) ALTER TABLE adalah perintah dasar untuk mengubah struktur tabel nama_tabel adalah Nama dari tabel yang akan diubah nama kolom nya.
2) CHANGE adalah perintah untuk mengganti nama kolom nama_kolom_lama adalah nama kolom lama yang ingin diganti nama_kolom_baru adalah Nama kolom baru yang berjumlah maximal 50 karakter, bisa.

Contoh berikut menggunakan pernyataan ALTER TABLE CHANGE untuk mengubah nama kolom pada tabel karyawan:

ALTER TABLE tbkaryawan CHANGE NomorTelepon NomorMobile VarChar(15) NOT NULL;

Pernyataan ini menunjukan mengubah field NomorTelepon menjadi NomorMobile dari tabel tbkaryawan:

```
MariaDB [dbbab6]> desc tbkaryawan;
+----------------+-------------+------+-----+---------+-------+
| Field          | Type        | Null | Key | Default | Extra |
+----------------+-------------+------+-----+---------+-------+
| NoKaryawan     | varchar(6)  | NO   | PRI | NULL    |       |
| NamaDepan      | varchar(50) | YES  |     | NULL    |       |
| NamaAkhir      | varchar(50) | NO   |     | NULL    |       |
| KotaLahir      | varchar(50) | YES  |     | NULL    |       |
| AlamatKaryawan | text        | YES  |     | NULL    |       |
| NomorMobile    | varchar(15) | NO   |     | NULL    |       |
| EmailKaryawan  | varchar(25) | YES  |     | NULL    |       |
+----------------+-------------+------+-----+---------+-------+
7 rows in set (0.019 sec)
```

Gambar 6.3 Perubahan Tabel NomorTelepon menjadi NomorMobile

c. Menambahkan Foreign Key

Untuk menabahkan foreign key pada field yang dianggap kunci tamu pada tabel, menggunakan sintaks:

ALTER TABLE table-child ADD FOREIGN KEY (field-key) REFERENCES table-parent(field-key);

Sebagai Contoh

Penjelasan sintaks ini:

1) ALTER TABLE adalah perintah dasar untuk mengubah struktur tabel nama_child adalah Nama dari tabel yang akan ditambahkan foreign-key.
2) ADD FOREIGN KEY adalah perintah untuk menambahkan foreign-key pada key-field yang dianggap sebagai kunci tamu pada table-child.
3) REFERENCES adalah perintah untuk menentukan mereferensikan dari table-parent mana yang akan dihubungkan, dan pada field-key yang sebagai PRIMARY KEY.

Contoh berikut menggunakan pernyataan ADD FOREIGN KEY untuk menambahkan nama field menjadi foreign-key pada tabel isi_penjualan:

ALTER TABLE isi_penjualan ADD FOREIGN KEY (kd_buku) REFERENCES buku(kd_buku);

Pernyataan ini menunjukan menambahkan dan mengubah status field kd_buku menjadi foreign-key pada tabel isipenjualan:

4. Menghapus Tabel

a. Perintah DROP

Perintah drop digunakan untuk menghapus objek seperti tabel, indeks, atau tampilan. Pernyataan DROP tidak dapat dibatalkan, jadi setelah objek

dihancurkan, tidak ada cara untuk memulihkannya. Sintaks pernyataan drop adalah:

DROP nama objek tipe objek;

Sebagai contoh:

DROP TABLE tbKaryawan;

Dalam contoh ini, kita menghapus tabel tbKaryawan.

b. Perintah TRUNCATE

Seperti DROP, pernyataan TRUNCATE digunakan untuk menghapus semua record dari tabel dengan cepat. Namun, tidak seperti DROP, yang benar-benar menghancurkan tabel, TRUNCATE mempertahankan struktur lengkapnya untuk digunakan kembali nanti.

Sintaks pernyataan truncate adalah:

TRUNCATE TABLE table_name;

Sebagai contoh:

TRUNCATE TABLE tbKaryawan;

Dalam contoh ini, kami menandai semua record pada tabel tbKaryawan untuk dealokasi, sehingga dianggap kosong untuk digunakan kembali.

6.1.2 Latihan Pembuatan Tabel

Sebelum kita melanjutkan pembahasan ini, alangkah baiknya anda buat terlebih dulu membuat beberapa tabel seperti berikut ini.

Tabel 6.2 Sturktur Tabel Penjualan

Nama Field	Tipe Data	Ukuran	NULL	Kunci Field
no_inv	Char	6	NO	Primary key
tgl_inv	Date	3	NO	
nm_pembeli	VarChar	50	NO	
ttl_harga	Int	8	NO	

Tabel 6.3 Sturktur Tabel Isi_Penjualan

Nama Field / Nama Kolom	Tipe Data	Ukuran	NULL	Kunci Field
no_inv	Char	6	NO	Foreign Key
kd_buku	Char	6	NO	Foreign Key
qty	Int	3	NO	
jml_harga	Int	8	NO	

Tabel 6.4 Sturktur Tabel Buku

Nama Field / Nama Kolom	Tipe Data	Ukuran	Kunci Field
kd_buku	Char	6	Primary Key
jdl_buku	VarChar	100	
stok_buku	int	10	
tipe_buku	Varchar	10	
royalty	Varchar	5	
harga_buku	int	6	

6.2 Data Manipulation Language

Data manipulation language (DML) digunakan untuk query dan memodifikasi data database. Dalam bab ini, kami akan menjelaskan cara menggunakan SELECT, INSERT, UPDATE, dan DELETE, didefinisikan di bawah ini.

1. INSERT digunakan untuk menyisipkan data ke dalam tabel.
2. SELECT digunakan untuk mengkueri data dalam database.
3. UPDATE digunakan untuk memperbarui data dalam tabel.
4. DELETE digunakan untuk menghapus data dari tabel.

Dalam pernyataan DML terdapat hal-hal yang perlu anda perhatikan:

Setiap klausa dalam pernyataan harus dimulai pada baris baru. Awal setiap klausa harus sejajar dengan awal klausa lainnya. Jika klausa memiliki beberapa bagian, klausa tersebut akan muncul pada baris terpisah dan diindentasi di bawah awal klausa untuk menunjukkan hubungan.

1. Upper case letters digunakan untuk mewakili kata-kata yang dilindungi undang-undang.
2. Lower case letters digunakan untuk mewakili kata-kata yang dilindungi undang-undang.

6.2.1 Pernyataan INSERT

Pernyataan INSERT, memungkinkan pengguna untuk memasukan data kedalam tabel, perintah yang digunakan untuk maksud tersebut adalah:

INSERT INTO nama_tabel [(nama_field-1, nama_field-2, ...)]
VALUES (‘value-1‘, 'value-2', ...);
Sebagai contoh:

```
MariaDB [dbbab6]> INSERT INTO karyawan(NoKaryawan, NamaDepan,
NamaAkhir, KotaLahir, AlamatKaryawan, NomorMobile, EmailKaryawan)
VALUE ('230304','Novita','Ginting','Medan','Kayumanis
Bogor','081298127634','novitabg@gmail.com');
```

Query OK, 1 row affected (0.004 sec)
Berikutnya masukan 5 buah record seperti data di kedalam tabel Karyawan

NoKaryawan	NamaDepan	NamaAkhir	KotaLahir	AlamatKaryawan	NomorMobile	EmailKaryawan
230301	Ilham	Hadiansyah	Jakarta	Pengasinan Depok	085158847086	ilham@email.com
230302	Davina	Putri	Bandung	Sawangan Depok	083803811761	davina@yyahoo.co.id
230303	Irvan	Ardiansyah	Jakarta	Depok	083875357554	irvan@ggmail.com
230304	Novita	Ginting	Medan	Kayumanis Bogor	081298127634	novitabg@gmail.com
230305	Rinno	Kertadisastra	Depok	Kebagusan	081514482727	rinno@gmail.com

Gambar 6.4 Data Karyawan

Setelah anda memasukan record untuk table karyawan, anda diminta untuk memasukan beberapa record untuk tabel buku, penjualan dan isi_penjualan, seperti tabel dibawah ini.

kd_buku	jdl_buku	stok_buku	tipe_buku	royalty	harga_buku
B01	Bumi Manusia	30	novel	15%	75000
B02	Dilan: Dia Adalah Dilanku Tahun 1990	25	Novel	10%	30000
B03	Tujuh Kelana	15	Novel	15%	85000
B04	Gadis Kretek	30	Novel	15%	60000
B05	Mukaddimah Ibnu Khaldun	40	Ilmu Pengetahuan	10%	265000
B06	Il Principe (Sang Pangeran)	35	Novel	10%	36000

Gambar 6.5 Data Buku

no_inv	tgl_inv	nm_pembeli	ttl_harga
INV001	2023-06-05	Ibu Istianti Elyana	105000
INV002	2023-06-05	Rani Irma Handayani	145000
INV003	2023-06-06	Kristiana	265000
INV004	2023-06-06	Irvan Y. Ardiansyah	66000
INV005	2023-06-07	Anggraeni Putri	111000
INV006	2023-06-07	Rani Handayani	325000

Gambar 6.6 Data Penjualan

no_inv	kd_buku	qyt
INV001	B01	1
INV001	B02	1
INV002	B03	1
INV002	B04	1
INV003	B05	1
INV004	B02	1
INV004	B06	1
INV005	B01	1
INV005	B06	1
INV006	B04	1
INV006	B05	1

Gambar 6.7 Data Isi_Penjualan

6.2.2 Pernyataan SELECT

Pernyataan SELECT, atau perintah SELECT, memungkinkan pengguna untuk mengekstrak data dari tabel, berdasarkan kriteria tertentu. Ini diproses sesuai dengan urutan berikut:

SELECT DISTINCT item(s)
FROM table(s)
WHERE predicate
GROUP BY field(s)
ORDER BY fields

Kita dapat menggunakan pernyataan SELECT untuk menampilkan nomor telepon karyawan dari tabel Karyawan sebagai berikut:

SELECT NamaDepan, NamaAkhir, NomorMobile
FROM Karyawan ORDER BY NamaAkhir;

Perintah tersebut akan menampilkan nama depan, nama belakang, dan nomor mobile karyawan dari tabel Karyawan dengan mengurutkan data berdasarkan NamaAkhir, hasil tersebut terlihat pada gambar 6.8.

```
MariaDB [dbbab6]> SELECT NamaDepan, NamaAkhir, NomorMobile
    -> FROM Karyawan ORDER BY NamaAkhir;
+-----------+---------------+--------------+
| NamaDepan | NamaAkhir     | NomorMobile  |
+-----------+---------------+--------------+
| Irvan     | Ardiansyah    | 083875357554 |
| Novita    | Ginting       | 081298127634 |
| Ilham     | Hadiansyah    | 085158847086 |
| Rinno     | Kertadisastra | 081514482727 |
| Davina    | Putri         | 083803811761 |
+-----------+---------------+--------------+
5 rows in set (0.010 sec)
```

Gambar 6.8 Tabel Karyawan

6.2.3 SELECT statement with WHERE criteria

Terkadang Anda mungkin ingin fokus pada sebagian tabel Penerbit, seperti hanya penerbit yang berada di Yogyakarta. Dalam situasi ini, Anda akan menggunakan pernyataan SELECT dengan kriteria WHERE, yaitu, WHERE PenerbitKota = ' Yogyakarta '. Berikut adalah contoh menggunakan pernyataan di atas.

```
MariaDB [dbbab6]> SELECT * FROM Penerbit WHERE PenerbitKota = 'Yogyakarta';
+------------+--------------+--------------+------------------+----------------+
| idpenerbit | PenerbitNama | PenerbitKota | PenerbitProvinsi | PenerbitNegara |
+------------+--------------+--------------+------------------+----------------+
|          1 | Deepublish   | Yogyakarta   | DI Yogyakarta    | Indonesia      |
+------------+--------------+--------------+------------------+----------------+
1 row in set (0.000 sec)
```

Gambar 6.9 Menampilkan Record PenerbitKota dengan Kriteria

1. **Menampilkan Total Harga diatas 120000.**

```
MariaDB [dbbab6]> SELECT no_inv, ttl_harga FROM Penjualan WHERE ttl_harga > 120000;
+--------+-----------+
| no_inv | ttl_harga |
+--------+-----------+
| INV002 |    145000 |
| INV003 |    265000 |
| INV006 |    325000 |
+--------+-----------+
3 rows in set (0.000 sec)
```

Gambar 6.10 Menampilkan Record No. Invoic e dan total harga

Dilihat contoh Gambar 6.10 menggambarkan cara membatasi pemilihan rekaman dengan kriteria WHERE. Masing-masing contoh ini memberikan hasil yang total harga lebih dari 120000.

2. **Kriteria Menggunakan Operator Logika.**

Contoh berikut menunjukkan cara memilih record menggunakan PenerbitProvinsi= sebagai bagian dari pernyataan WHERE.

Berikut adalah contoh menggunakan pernyataan di atas.

```
MariaDB [dbbab6]> SELECT * FROM penerbit
    -> WHERE PenerbitProvinsi = "Jawa Barat" OR
    -> PenerbitProvinsi = "sumatra selatan" OR
    -> PenerbitProvinsi = "Jakarta";
+------------+-------------------------+--------------+------------------+----------------+
| idpenerbit | PenerbitNama            | PenerbitKota | PenerbitProvinsi | PenerbitNegara |
+------------+-------------------------+--------------+------------------+----------------+
|          6 | Grafindo Media Tama     | Bandung      | Jawa Barat       | Indonesia      |
|          7 | Bening Media Publishing | Palembang    | Sumatra Selatan  | indonesia      |
+------------+-------------------------+--------------+------------------+----------------+
2 rows in set (0.000 sec)
```

Gambar 6.11 Menampilkan Penerbit dalam Provinsi

Dilihat contoh Gambar 6.11 menggambarkan cara membatasi pemilihan penerbit dengan kriteria WHERE. Kriteria yang digukan lebih dari satu dan masing-masing kriteria tersebut dengan operator logika OR.

3. **Menggunakan Wildcard dalam Klausa LIKE**

Kata kunci LIKE memilih baris yang berisi bidang yang cocok dengan bagian string karakter tertentu. LIKE digunakan dengan char, varchar, teks, datetime, dan data smalldatetime. Wildcard memungkinkan pengguna untuk mencocokkan bidang yang

berisi huruf-huruf tertentu. Tabel 6.5 menunjukkan empat cara untuk menentukan wildcard dalam pernyataan SELECT dalam format ekspres reguler.

Table 6.5 Simbol dalam Wildcard

Simbol	Deskrispsi
%	Setiap string dari nol atau lebih karakter
_	Setiap karakter tunggal
[ ]	Setiap karakter tunggal dalam rentang yang ditentukan (misalnya, [a-f]) atau set (misalnya, [abcdef])
[^]	Setiap karakter tunggal yang tidak berada dalam rentang yang ditentukan (misalnya, [^a – f]) atau set (misalnya, [^abcdef])

Dalam contoh berikut penggunaan, LIKE '%syah' menelusuri semua nama belakang yang dimulai dengan huruf "syah".

SELECT NamaAkhir **FROM** Karyawan
WHERE NamaAkhir Like "%syah";

Berikut adalah contoh menggunakan pernyataan di atas.

```
MariaDB [dbbab6]> SELECT NamaAkhir FROM Karyawan
    -> WHERE NamaAkhir Like "%syah";
+------------+
| NamaAkhir  |
+------------+
| Hadiansyah |
| Ardiansyah |
+------------+
2 rows in set (0.016 sec)
```

Gambar 6.12 Hasil menampilkan Nama Akhir memiliki kata “syah“

Misalnya kita mencari mengandung kata 'an', maka simbol '%' mengapit kata 'an' seperti LIKE '%an%' menelusuri semua nama akhir yang mengandung kata 'an'.

SELECT NamaAkhir **FROM** Karyawan
WHERE NamaAkhir Like "%syah";

Berikut adalah contoh menggunakan pernyataan di atas.

```
MariaDB [dbbab6]> SELECT NamaAkhir FROM Karyawan
    -> WHERE NamaAkhir Like "%syah";
+------------+
| NamaAkhir  |
+------------+
| Hadiansyah |
| Ardiansyah |
+------------+
2 rows in set (0.001 sec)
```

Gambar 6.13 Hasil menampilkan Nama Akhir memiliki kata "an"

Gambar 6.13 Hasil menampilkan Nama Akhir memiliki kata "an" ada didepan ataupun dibelakang dan ditengah kata Nama Akhir.

4. SELECT statement with ORDER BY clause

Anda menggunakan klausa ORDER BY untuk mengurutkan rekaman dalam daftar yang dihasilkan. Gunakan ASC untuk mengurutkan hasil dalam urutan naik dan DESC untuk mengurutkan hasil dalam urutan menurun.

Misalnya, dengan ASC:

SELECT NoKaryawan, NamaDepan, NamaAkhir **FROM** karyawan
ORDER BY NamaDepan ASC;

Berikut adalah contoh menggunakan pernyataan di atas.

```
MariaDB [dbbab6]> SELECT NoKaryawan, NamaDepan, NamaAkhir FROM karyawan
    -> ORDER BY NamaDepan ASC;
+------------+-----------+---------------+
| NoKaryawan | NamaDepan | NamaAkhir     |
+------------+-----------+---------------+
| 230302     | Davina    | Putri         |
| 230301     | Ilham     | Hadiansyah    |
| 230303     | Irvan     | Ardiansyah    |
| 230304     | Novita    | Ginting       |
| 230305     | Rinno     | Kertadisastra |
+------------+-----------+---------------+
5 rows in set (0.011 sec)
```

Gambar 6.14 Menampilkan Hasil Query Pengurutan Ascending Nama Depan

Juga dengan DESC:

SELECT NoKaryawan, NamaDepan, NamaAkhir FROM karyawan
ORDER BY NamaDepan DESC;

Berikut adalah contoh menggunakan pernyataan di atas.

```
MariaDB [dbbab6]> SELECT NoKaryawan, NamaDepan, NamaAkhir FROM karyawan
    -> ORDER BY NamaDepan DESC;
+------------+-----------+---------------+
| NoKaryawan | NamaDepan | NamaAkhir     |
+------------+-----------+---------------+
| 230305     | Rinno     | Kertadisastra |
| 230304     | Novita    | Ginting       |
| 230303     | Irvan     | Ardiansyah    |
| 230301     | Ilham     | Hadiansyah    |
| 230302     | Davina    | Putri         |
+------------+-----------+---------------+
5 rows in set (0.001 sec)
```

Gambar 6.15 Menampilkan Hasil Query Pengurutan Descending Nama Depan

5. SELECT statement with GROUP BY clause

Klausa GROUP BY digunakan untuk membuat satu baris output per setiap grup dan menghasilkan nilai ringkasan untuk kolom yang dipilih, seperti yang ditunjukkan di bawah ini.

SELECT tipe_buku **FROM** BUKU
GROUP BY tipe_buku;

Berikut adalah contoh menggunakan pernyataan di atas.

```
MariaDB [dbbab6]> SELECT tipe_buku FROM BUKU
    -> GROUP BY tipe_buku;
+------------------+
| tipe_buku        |
+------------------+
| Ilmu Pengetahuan |
| novel            |
+------------------+
2 rows in set (0.009 sec)
```

Gambar 6.16 Menampilkan Hasil Query Group berdasarkan Tipe Buku

SELECT tipe_buku **AS** "Tipe Buku", **MIN**(harga_buku) AS "Minimum harga" **FROM** Buku **WHERE** royalty > '10%' **GROUP BY** tipe_buku;

Berikut adalah contoh menggunakan pernyataan di atas.

```
MariaDB [dbbab6]> SELECT tipe_buku AS "Tipe Buku", MIN(harga_buku) AS "Minimum harga"
    -> FROM Buku WHERE royalty > '10%' GROUP BY tipe_buku;
+-----------+--------------+
| Tipe Buku | Minimum harga |
+-----------+--------------+
| novel     |        60000 |
+-----------+--------------+
1 row in set (0.009 sec)
```

Gambar 6.17 Menampilkan Hasil Query Group berdasarkan Tipe Buku

Contoh berikut dengan ketentuan bahwa jika pernyataan SELECT menyertakan kriteria WHERE di mana harga buku not null,

SELECT tipe_buku, harga_buku **FROM** buku **WHERE** harga_buku is not null;

Berikut adalah contoh menggunakan pernyataan di atas.

```
MariaDB [dbbab6]> SELECT tipe_buku, harga_buku FROM buku WHERE harga_buku is not null;
+------------------+------------+
| tipe_buku        | harga_buku |
+------------------+------------+
| novel            |      75000 |
| Novel            |      30000 |
| Novel            |      85000 |
| Novel            |      60000 |
| Ilmu Pengetahuan |     265000 |
| Novel            |      36000 |
+------------------+------------+
6 rows in set (0.001 sec)
```

Gambar 6.18 Menampilkan Hasil tipe buku dan harga buku tanpa Group By

Maka pernyataan dengan klausa GROUP BY akan terlihat seperti ini:

SELECT tipe_buku **AS** "Tipe Buku", MIN(harga_buku) **AS** "Minimum Harga" **FROM** buku **WHERE** harga_buku is not null **GROUP BY** tipe_buku;

Berikut adalah contoh menggunakan pernyataan di atas.

```
MariaDB [dbbab6]> SELECT tipe_buku AS "Tipe Buku", MIN(harga_buku) AS "Minimum Harga"
    -> FROM buku WHERE harga_buku is not null GROUP BY tipe_buku;
+------------------+---------------+
| Tipe Buku        | Minimum Harga |
+------------------+---------------+
| Ilmu Pengetahuan |        265000 |
| novel            |         30000 |
+------------------+---------------+
2 rows in set (0.001 sec)
```

Gambar 6.19 Menampilkan Hasil tipe buku dan harga buku dengan Group By

Gambar 6.19 Menampilkan Hasil tipe buku dan harga buku dengan Group By dimana harga buku menampilkan harga yang paling rendah.

6. Menggunakan COUNT dengan GROUP BY

Kita dapat menggunakan COUNT untuk menghitung berapa banyak item dalam satu wadah. Namun, jika kita ingin menghitung item yang berbeda ke dalam kelompok yang terpisah, seperti kelereng dengan berbagai warna, maka kita akan menggunakan fungsi COUNT dengan perintah GROUP BY.

Pernyataan SELECT di bawah ini mengilustrasikan cara menghitung grup data menggunakan fungsi COUNT dengan klausa GROUP BY.

SELECT COUNT(*) **FROM** buku **GROUP BY** tipe_buku;

```
MariaDB [dbbab6]> SELECT COUNT(*) FROM buku GROUP BY tipe_buku;
+----------+
| COUNT(*) |
+----------+
|        1 |
|        5 |
+----------+
2 rows in set (0.003 sec)
```

Gambar 6.20 Menampilkan Hasil Jumlah Tipe Buku

7. Using AVG and SUM with GROUP BY

Kita dapat menggunakan fungsi AVG untuk memberi kita rata-rata grup mana pun, dan SUM untuk memberikan total. Contoh berikut menggunakan fungsi AVG dengan GROUP BY pada tipe buku. Perintah yang dapat digunakan untuk maksud tersebut seperti dibawah ini.

SELECT AVG(stok_buku) **FROM** buku **GROUP BY** tipe_buku;

```
MariaDB [dbbab6]> SELECT AVG(stok_buku) FROM buku GROUP BY tipe_buku;
+----------------+
| AVG(stok_buku) |
+----------------+
|        40.0000 |
|        27.0000 |
+----------------+
2 rows in set (0.003 sec)
```

Gambar 6.21 Menampilkan Hasil Rata-Rata Stok Buku

Contoh berikut menggunakan fungsi SUM dengan GROUP BY pada tipe buku. Perintah yang dapat digunakan untuk maksud tersebut seperti dibawh ini.

SELECT SUM(stok_buku) **FROM** buku **GROUP BY** tipe_buku;

```
MariaDB [dbbab6]> SELECT SUM(stok_buku) FROM buku GROUP BY tipe_buku;
+----------------+
| SUM(stok_buku) |
+----------------+
|             40 |
|            135 |
+----------------+
2 rows in set (0.002 sec)
```

Gambar 6.22 Menampilkan Hasil Jumlah Stok Buku

8. Pembatasan Baris dengan HAVING

Klausa HAVING dapat digunakan untuk membatasi baris. Ini mirip dengan kondisi WHERE kecuali HAVING dapat mencakup fungsi agregat; di mana tidak dapat melakukan ini. Klausa HAVING berperilaku seperti klausa WHERE, tetapi berlaku untuk grup.

SELECT NamaDepan AS "Nama Depan“, KotaLahir as “Kota Kelahiran“
FROM Karyawan GROUP BY NamaDepan, AlamatKaryawan
HAVING KotaLahir <> “Jakarta“;

Perintah yang dapat digunakan untuk maksud tersebut seperti dibawh ini.

```
MariaDB [dbbab6]> SELECT NamaDepan AS "Nama Depan", KotaLahir as "Kota Kelahiran"
    -> FROM Karyawan  GROUP BY NamaDepan, AlamatKaryawan
    -> HAVING KotaLahir <> "Jakarta";
+------------+----------------+
| Nama Depan | Kota Kelahiran |
+------------+----------------+
| Davina     | Bandung        |
| Novita     | Medan          |
| Rinno      | Depok          |
+------------+----------------+
3 rows in set (0.039 sec)
```

Gambar 6.23 Menampilkan Hasil Kota Lahir Selain Jakarta

Bab 7
MySQL

7.1 Sejarah MySQL

MySQL adalah sistem manajemen basis data relasional (RDBMS) yang sangat populer. Pengembangan MySQL dimulai pada tahun 1994 oleh Michael Widenius dan David Axmark di Swedia. Nama MySQL sendiri berasal dari nama anak Michael, My, dan singkatan SQL (Structured Query Language).

Pada awalnya, MySQL adalah proyek open-source yang dikembangkan sebagai alternatif yang lebih terjangkau untuk sistem manajemen basis data yang ada saat itu. Versi awal MySQL dirilis pada tahun 1995, dan proyek ini terus berkembang secara aktif sejak saat itu. Pada tahun 2000, MySQL AB didirikan sebagai perusahaan komersial untuk memasarkan dan mendukung MySQL. Perusahaan ini didirikan oleh Michael Widenius, David Axmark, dan Allan Larsson. MySQL AB kemudian menjadi penyedia utama MySQL dan mengembangkan versi komersial MySQL yang disebut MySQL Enterprise Edition MySQL (Teknik & Diponegoro, 2020).

Pada tahun 2008, Sun Microsystems membeli MySQL AB dengan harga sekitar 1 miliar dolar AS. Sun Microsystems kemudian diakuisisi oleh Oracle Corporation pada tahun 2010, sehingga membuat Oracle menjadi pemilik dan pengeksekusi utama MySQL. MySQL terus berkembang dan mengalami pembaruan secara teratur. Fitur-fitur baru dan peningkatan kinerja diperkenalkan dalam setiap rilis baru. MySQL juga mendukung berbagai bahasa pemrograman, termasuk PHP, Python, Java, dan banyak lagi. MySQL juga telah memainkan peran penting dalam perkembangan teknologi web. MySQL sering digunakan sebagai basis data back-end untuk aplikasi web dan sistem manajemen konten (CMS) seperti WordPress, Drupal, dan Joomla. Pada akhir tahun 2020, MySQL merupakan salah satu RDBMS yang paling populer dan banyak digunakan di dunia, dengan komunitas pengguna yang besar dan aktif. MySQL terus menjadi pilihan yang populer bagi pengembang dan organisasi untuk kebutuhan manajemen basis data relasional mereka.

7.2 Database MySQL

Sistem perangkat lunak yang digunakan untuk mengelola, menyimpan, mengorganisir, dan mengakses data dalam basis data disebut dengan DBMS. DBMS adalah singkatan dari "Database Management System" atau sistem manajemen basis data dalam bahasa Indonesia. DBMS adalah perangkat lunak yang digunakan untuk mengelola dan mengatur basis data. Basis data adalah kumpulan data terorganisir yang disimpan dalam sistem komputer, dan DBMS bertanggung jawab untuk menyediakan metode penyimpanan, pengambilan, pembaruan, dan penghapusan data dalam basis data. Dengan DBMS, user akan lebih mudah mengontrol dan memanipulasi data yang ada (Solichin, 2015). Contoh DBMS populer termasuk MySQL, Oracle Database, Microsoft SQL Server, PostgreSQL, dan MongoDB. Setiap DBMS memiliki fitur dan kelebihan masing-masing yang dapat disesuaikan dengan kebutuhan aplikasi atau organisasi yang menggunakan basis data.

Salah satu jenis DBMS yang mendukung adanya relationship atau hubungan antar tabel disebut dengan RDBMS (Achmad Solichin, 2016). MySQL adalah sistem manajemen basis data relasional (RDBMS) yang sangat populer. Salah satu perangkat lunak open-source yang digunakan untuk mengelola basis data relasional. MySQL menyediakan lingkungan yang aman dan handal untuk menyimpan, mengakses, dan mengelola data dalam berbagai aplikasi. MySQL digunakan secara luas dalam berbagai aplikasi, termasuk situs web, aplikasi bisnis, sistem manajemen konten (CMS), dan banyak lagi. Popularitasnya yang tinggi, dukungan komunitas yang kuat, dan kinerja yang handal membuat MySQL menjadi salah satu pilihan utama untuk kebutuhan manajemen basis data relasional (Dewa Putu Yudhi Ardiana, 2022).

Kinerja yang tinggi dan penggunaannya yang dapat digunakan oleh banyak pengguna sekaligus menjadikan MySQL sebagai open-source yang populer dan memberikan dukungan secara komprehensif pada kebutuhan pengembangan aplikasi membuat Mysql banyak diminati (Enterprise, 2017). MySQL memiliki sejumlah fitur yang membuatnya menonjol di antara RDBMS lainnya. Beberapa fitur utama MySQL adalah:

1. **Skalabilitas:** MySQL dapat digunakan untuk membangun aplikasi dari skala kecil hingga besar dengan jutaan baris data. Ini mendukung pertumbuhan dan peningkatan kebutuhan basis data seiring waktu.
2. **Kinerja yang Cepat:** MySQL dirancang untuk memberikan kinerja yang tinggi. Dengan pengoptimalan internal dan penggunaan indeks yang efisien, MySQL mampu menangani beban kerja yang intensif secara efisien.

3. **Multi-platform:** MySQL mendukung berbagai platform seperti Windows, Linux, macOS, dan berbagai sistem operasi lainnya. Ini memungkinkan pengguna untuk mengimplementasikan MySQL pada berbagai lingkungan.
4. **Keamanan:** MySQL menyediakan berbagai fitur keamanan untuk melindungi data, termasuk akses pengguna yang terkendali, enkripsi data, dan perlindungan terhadap serangan keamanan.
5. **Replikasi dan Kepiawaian Tinggi:** MySQL mendukung fitur replikasi yang memungkinkan replikasi data antara server MySQL yang berbeda. Ini membantu meningkatkan ketersediaan sistem dan memastikan kepiawaian tinggi.
6. **Komunitas Besar dan Dukungan:** MySQL memiliki komunitas pengguna yang besar dan aktif di seluruh dunia. Ini berarti ada banyak sumber daya, dokumentasi, dan dukungan yang tersedia bagi pengguna MySQL.
7. **Kompatibilitas dengan Standar Industri:** MySQL mengikuti standar industri dalam hal bahasa SQL, sehingga memudahkan penggunaan dan migrasi basis data antara platform yang berbeda.

Meskipun MySQL merupakan sistem manajemen basis data yang populer dan andal, ada beberapa kelemahan yang perlu dipertimbangkan:

1. **Keterbatasan Skalabilitas:** MySQL memiliki batasan dalam hal skalabilitas vertikal (scaling up) karena tergantung pada kekuatan satu server tunggal. Meskipun dapat diatasi dengan meng-upgrade hardware server, ini tidak praktis untuk mengatasi kebutuhan skalabilitas yang sangat besar. Meskipun MySQL juga mendukung replikasi dan klasterisasi untuk skalabilitas horizontal (scaling out), mengkonfigurasi dan mengelola lingkungan yang terdistribusi dapat menjadi rumit.
2. **Kinerja pada Beban Tinggi:** Saat menghadapi beban tinggi, kinerja MySQL dapat terpengaruh. Jika terdapat banyak permintaan yang kompleks atau transaksi yang berat, waktu respons mungkin meningkat dan kinerja keseluruhan dapat menurun. Penting untuk mengoptimalkan kueri, mengindeks tabel dengan baik, dan mempertimbangkan caching untuk mengurangi beban pada sistem.
3. **Kurangnya Fitur Lengkap:** Meskipun MySQL telah berkembang dengan pesat dan menambahkan banyak fitur baru, masih ada beberapa fitur yang tidak sepenuhnya didukung. Beberapa fitur tingkat lanjut seperti trigger yang kompleks, prosedur penyimpanan, atau pemrosesan antrian pesan mungkin tidak sekuat implementasi pada sistem manajemen basis data lainnya.
4. **Skema yang Tidak Fleksibel:** MySQL menggunakan pendekatan skema yang cukup kaku, yang berarti struktur tabel harus ditentukan sebelumnya. Jika terjadi perubahan skema yang signifikan, seperti menambahkan atau menghapus

kolom, dapat membutuhkan perubahan skema secara manual yang berpotensi mempengaruhi aplikasi yang bergantung pada basis data.

5. **Keamanan yang Terbatas:** Meskipun MySQL menyediakan beberapa fitur keamanan seperti otentikasi pengguna dan izin akses, keamanan secara umum dianggap sebagai kelemahan. Pada tingkat paling dasar, MySQL tidak menyediakan enkripsi data secara default. Perlindungan keamanan tambahan harus diimplementasikan secara manual untuk melindungi data yang disimpan dalam basis data.
6. **Tidak Dapat Memproses Data Semi-Struktur:** MySQL lebih baik digunakan untuk data yang terstruktur dengan skema yang didefinisikan dengan baik. Jika Anda perlu mengelola data semi-struktur atau tidak terstruktur, seperti JSON atau XML, MySQL mungkin kurang efektif. Meskipun MySQL memiliki beberapa fitur untuk mengelola data semi-struktur, sistem manajemen basis data lain yang dirancang khusus untuk kasus penggunaan ini mungkin lebih cocok.

Perlu dicatat bahwa beberapa kelemahan yang disebutkan di atas dapat diatasi dengan menggunakan solusi alternatif, menggunakan konfigurasi yang tepat, atau menggabungkan MySQL dengan alat atau teknologi lainnya. Selain itu, banyak perusahaan dan organisasi menggunakan MySQL dengan sukses dan mampu mengatasi kelemahan-kelemahan tersebut dengan pendekatan yang tepat.

7.3 Instalasi MySQL

MySQL Community Server adalah versi open-source dari sistem manajemen basis data MySQL. Ini adalah versi yang paling umum digunakan dan tersedia secara gratis untuk digunakan oleh pengguna dan pengembang. Selain itu, Database administrator atau programmer juga dapat menggunakan mysql pada bagian dari paket aplikasi Amp (Apache-MySql-Php). MySQL adalah salah satu komponen inti dalam paket XAMPP. XAMPP adalah sebuah paket perangkat lunak yang dikembangkan oleh Apache Friends, yang berfungsi sebagai platform pengembangan web yang mudah digunakan. XAMPP memiliki komponen-komponen utama seperti Apache (web server), MySQL (database server), PHP, dan Perl. Dalam konteks XAMPP, MySQL digunakan sebagai server basis data untuk pengembangan dan pengujian aplikasi web lokal.

MySql mendukung berbagai macam sistem operasi, namun karena sebagian besar menggunakan Windows, maka buku ini akan menjelaskan cara instalasi MySQL di Windows. Untuk menggunakan mysql secara sendiri (stand alone), beberapa tahapan perlu dilakukan antara lain (Pratama, 2017):

1. Masuk ke halaman download mysql melalui link http://dev.mysql.com/downloads/mysql/ untuk mendownload aplikasi MySQL. Untuk instalasi aplikasi, terdapat 2 jenis file pada MySQL Installer, yaitu file

dengan ukuran 16 MB dan berukuran 313 MB. Versi online installer yakni Versi 16 MB, yang akan mengunduh seluruh file MySQL, namun saat proses instalasi dilakukan harus terhubung dengan internet. Untuk versi offline installer yakni versi 313 MB, tidak perlu terhubung dengan internet ketika proses instalasi. Oleh karena itu, silahkan menekan tombol download pada bagian bawah.

Gambar 7.1 *MySQL Installer for Windows*

2. Selanjutnya akan ditemukan file instalasi MySQL berupa mysql-installer-web-community-8.0.32.0 setelah proses download file MySQL selesai. MySQL terus memperbaharui versinya sehingga kemungkinan versi yang anda download akan berbeda, namun kurang lebih tahapan instalasinya tetap sama.
3. MySQL akan menampilkan form login atau membuat akun di mysql.com pada halaman "*Begin your download*". Lanjutkan dengan mengisi form registrasi jika berminat, sebaliknya lanjutkan dengan menekan link "No thanks, just start my download" jika tidak berminat.

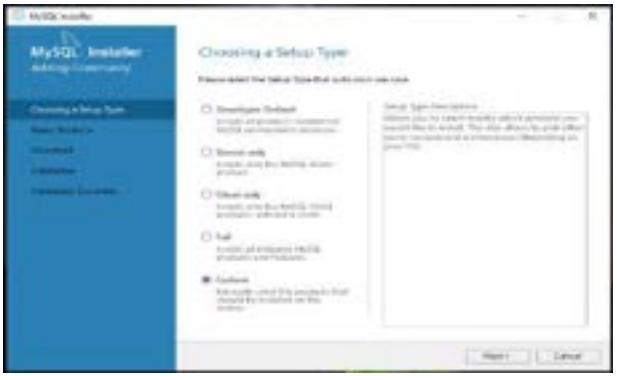
Gambar 7.2 *Begin Your Download*

4. Lakukan klik 2x pada file mysql-installer-community-8.0.32.0.msi. Jendela konfirmasi hak akses administrator akan tampil beberapa kali, jika anda menggunakan Windows 7, 8 atau Windows 10 dan silahkan klik Yes.

5. Lisensi MySQL akan tampil setelah proses persiapan install selesai, sekilas anda akan terlihat bahwa lisensi *GNU GENERAL PUBLIC LICENSE* digunakan oleh MySQL. Ceklist "*I accept the terms in the License Agreement*" lalu klik next.

Gambar 7.3 Jendela *License Agreement*

6. Terdapat beberapa pilihan tipe instalasi pada jendela "*Choosing a Setup Type*". Mulai dari untuk developer default, server only, client only, full hingga custom. Karena dalam tahap pengembangan program akan menggunakan MySQL, maka pilihan yang paling pas adalah "Developer Default". Namun opsi ini akan memasang banyak aplikasi, hingga 10 komponen serta memakan ruang harddisk lebih dari 1GB. Yang kita perlukan saat ini hanyalah MySQL Server. Jika memilih Server Only, maka kita tidak dapat mengubah folder instalasi MySQL. Oleh karena itu pilih Custom dilanjutkan dengan mengklik tombol Next.



Gambar 7.4 Jendela *Choosing a Setup Type*

7. Aplikasi dan fitur apa saja yang ingin diinstall akan ditampilkan pada jendela "*Select Products and Features*". Dapat dilihat bahwa file instalasi MySQL membundel banyak komponen selain MySQL Server, antara lain: Aplikasi

admin untuk excel, visual studio, connector MySQL, dan juga file dokumentasi. Hal ini menyebabkan file installer MySQL begitu besar. Klik tanda tambah (+) disamping MySQL Server, hingga anda menemukan MySQL Server 8.0. Tekan tanda panah hijau untuk memindahkan MySQL 8.0.32.0–X64 ke jendela "Product/Features to Be Installed". Klik pilihan MySQL Server 8.0.32–X64 pada jendela sebelah kanan, sehingga muncul pilihan "*Advanced Option*" di pojok kanan bawah.

Gambar 7.5 Jendela *Select Products and Features*

Jendela baru untuk mengubah lokasi instalasi MySQL akan keluar seperti yang ditunjukkan pada gambar 7.6 ketika mengklik *Advanced Option*. Jika anda langsung menekan tombol next, maka program MySQL akan diinstallasi pada folder: C:\Program Files\MySQL\MySQL Server 8.0, dan folder data di C:\ProgramData\MySQL\MySQL Server 8.0. Walaupun tidak menjadi kendala, tapi direkomendasikan untuk mengubahnya ke folder lain agar mudah diakses. Folder MySQL Server 8.0 dan data tidak harus ada terlebih dahulu karena secara otomatis akan dilakukan oleh MySQL. Akhiri dengan menekan tombol OK, kemudian menekan Next.

Gambar 7.6 Jendela *Advanced Option*

8. Klik tombol execute seperti pada gambar 7.7 untuk mendownload seluruh produk MySQL Server 8.0.32,

Gambar 7.7 Jendela *download package MySQL Server 8.0.32*

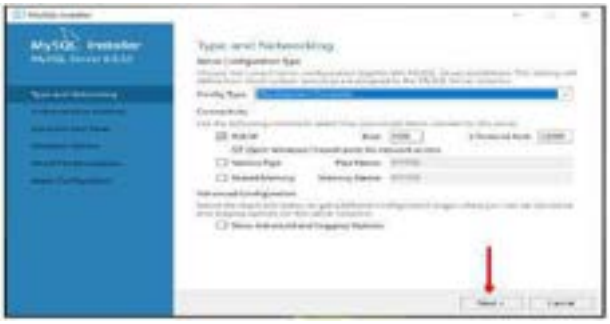
9. Pada Jendela Installation akan tampil produk dan fitur apa saja yang ingin diinstal. Lanjutkan dengan menekan tombol Execute, untuk memulai Proses Instalasi MySQL Server. Proses instalasi akan membutuhkan waktu beberapa lama, dan klik tombol Next terus-menerus sampai masuk ke bagian konfigurasi awal MySQL.

Gambar 7.8 Jendela *Installation MySQL Server 8.0.32*

7.4 Konfigurasi Awal MySQL

MySQL akan masuk ke menu konfigurasi awal MySQL, setelah proses instalasi. Untuk proses konfigurasi dilakukan tahapan-tahapan berikut:

1. Pada jendela "*Type and Networking*" gunakan pengaturan default pada opsi Development Computer dan port 3306. Selanjutnya klik Next



Gambar 7.9 Jendela *Installation MySQL Server 8.0.32*

2. Biarkan pilihan default pada jendela "*Authentication Method*" bagian "*Use Strong Password Encryption for Authentication*", kemudian klik Next.

Gambar 7.10 Jendela *Authentication Method*

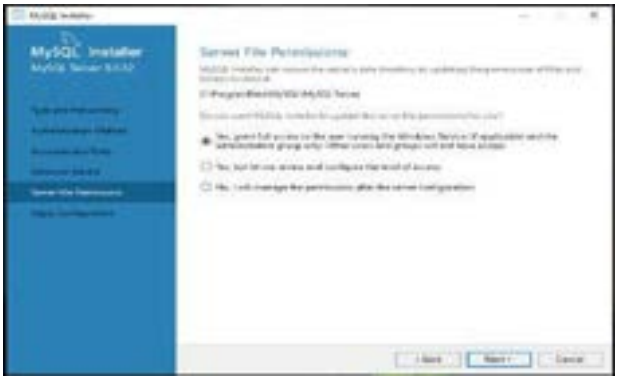
3. Pada jendela "*Account and Roles*", kita harus memasukkan password untuk user root. Dalam MySQL, "superuser" mengacu pada pengguna dengan hak istimewa tertinggi yang dikenal sebagai "root". Akun root memiliki akses penuh dan kontrol penuh atas server MySQL. Untuk kebutuhan belajar, boleh diset dengan password yang mudah diingat misalnya 123456 atau sesuai dengan keinginan, namun perlu diingat karena password tersebut nantinya akan menjadi kunci masuk pada penggunakan mysql.

Gambar 7.11 Jendela *Account and Roles*

4. Jika checkbox “Configure MySQL Server as Windows Service” dipilih pada jendela ”*Windows Service*”, maka akan diinstalasi sebagai “Windows Service”. Dengan artian, MySQL akan langsung aktif setiap Windows berjalan. Jika ingin menjalankan MySQL dari sumber lain seperti XAMPP maka MySQL tidak dapat berjalan bersamaan pada satu komputer karena berada pada port yang sama. Maka dari itu, hapus pilihan “Configure MySQL Server as Windows Service”. Biarkan pilihan default pada opsi Standard System Account, selanjutnya klik Next.

Gambar 7.12 *Windows Service*

5. MySQL dapat mengamankan direktori data server dengan memperbarui izin lokasi file dan folder di C:\ProgramData\MySQL\MySQL Server. Berikan akses penuh kepada pengguna pada jendela ”*Server File Permissions*” yang menjalankan layanan windows dan grup administrator saja agar pengguna dan grup lain tidak akan memiliki akses, selanjutnya klik Next.



Gambar 7.13 Jendela *Server File Permissions*

6. Klik execute pada jendela ”*Apply Configuration*” untuk menerapkan perubahan. Lanjutkan dengan men-klik finish setelah seluruh konfigurasi diterapkan.

Gambar 7.14 Jendela *Apply Configuration*

7. Klik next untuk konfigurasi produk, anda dapat membatalkan kapan saja jika hendak meninggalkan wizard tanpa mengkonfigurasi semua produk dengan menekan cancel.

Gambar 7.15 Jendela *"Product Configuration"*

8. Pada jendela "*Installation Complete*", klik finish untuk mengakhiri prosedur instalasi.

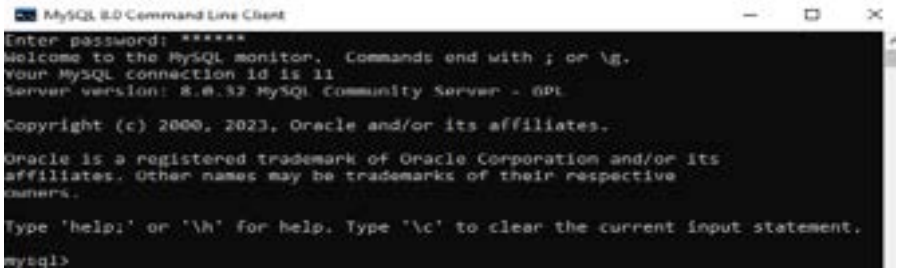
Gambar 7.16 Jendela *"Installation Complete"*

7.5 Memulai MySQL

Setelah melakukan instalasi MySQL, MySQL Server 8.0 secara otomatis juga telah berjalan di latar belakang. Pengguna dapat memulai MySQL dengan memilih menu Start, kemudian pilih MySQL dan klik MySQL 8.0 Command Line Client.

Gambar 7.17 *MySQL 8.0 Command Line*

Input password pada jendela DOS yang terbuka untuk user root sesuai dengan password yang dimasukkan pada proses instalasi, kemudian tekan enter. Jika tampil "Welcome to the MySQL monitor", berarti instalasi MySQL telah berhasil. Lanjutkan dengan membuat database menggunakan perintah SQL, gunakan tanda titik koma di akhir setiap perintah lalu tekan enter.



Gambar 7.18 *MySQL 8.0 Command Line Monitor*

Bab 8
Data Definition Language (DDL)

8.1 Definisi Data Definition Language

Subbahasa dalam Structure Query Language (SQL) terdiri dari beberapa bagian antara lain Data Definition Language (DDL), Data Manipulation Language (DML) dan Data Controlling Language (DCL)(Connolly & Begg, 2016). DDL digunakan untuk membentuk atau membuat struktur basis data, DML digunakan untuk melakukan manipulasi atau pengolahan data pada basis data dan DCL digunakan untuk melakukan kontrol data serta server basis data serta dapat juga digunakan untuk mengaudit penggunaan basis data sampai alokasi basis data kepada user. Istilah yang harus dipahami adalah dalam Structure Query Language (SQL) menggunakan istilah tabel, baris dan kolom sedangkan dalam relasional basis data menggunakan istilah relasi, tupel dan atribut(Elmasri & Navathe, 2010).

Pada bab ini akan dibahas secara rinci tentang Data Definition Language (DDL) yang digunakan dalam SQL. Perintah DDL digunakan untuk membuat, memodifikasi dan menghapus struktur basis data termasuk tabel. Selain itu perintah DDL memiliki efek langsung pada basis data serta merekam informasi pada data dictionary (kamus data)(Gupta & Mittal, 2017). DLL juga memungkinkan tidak hanya menspesifikasikan hanya satu himpunan relasi tetapi juga informasi tentang setiap relasi, termasuk(Silberschatz, Korth, & Sudharshan, 2019):

1. Skema setiap hubungan.
2. Jenis nilai yang terkait dengan setiap atribut.
3. Integrity Constrains.
4. Himpunan index yang dipelihara dalam setiap relasi.
5. Informasi keamanan dan otorisasi untuk setiap relasi.
6. Struktur penyimpanan fisik dari setiap relasi pada disk.

Hasil kompilasi perintah DDL adalah terbentuknya seperangkat tabel yang tersimpan dalam file khusus secara kolektif yang disebut dengan katalog sistem. Katalog

sistem mengintegrasikan metadata yang menggambarkan objek dalam basis data dan memudahkan objek tersebut diakses atau dimanipulasi(Connolly & Begg, 2016). Pada metadata berisi catatan, item data dan objek lain yang menarik bagi pengguna atau diperlukan oleh DBMS.

Terdapat tiga jenis perintah yang dapat digunakan pada Data Definition Language (DDL), yang ditunjukkan pada tabel 8.1.

Tabel 8.1 Perintah Dasar Data Definition Language (DDL)

Perintah	Fungsi
CREATE	Perintah yang digunakan untuk menciptakan struktur basis data seperti DATABASE, TABLE, INDEX, VIEW dan TRIGGER.
ALTER	Perintah yang digunakan untuk memodifikasi struktur tabel.
DROP	Perintah yang digunakan untuk menghapus struktur basis data seperti DATABASE, TABLE, INDEX, VIEW dan TRIGGER.
RENAME	Perintah yang digunakan untuk mengubah nama tabel.
TRUNCATE	Perintah yang digunakan untuk menghapus semua record pada database.

8.2 Perintah CREATE

Perintah SQL yang utama dalam Data Definition Language (DDL) adalah perintah CREATE yang dapat digunakan untuk membuat atau menciptakan basis data, tabel, index, view dan trigger (Elmasri & Navathe, 2010).

1. **CREATE DATABASE.**

   Perintah dalam DDL yang digunakan untuk membuat database baru. Format penulisan untuk perintah ini adalah sebagai berikut

   CREATE DATABASE NamaDatabase;

   Pada pembahasan tentang Data Definition Language (DDL) ini akan menggunakan contoh kasus pembuatan sistem basis data tentang rekam medis dengan nama RumahSakit_XYZ. Maka berdasarkan contoh kasus tersebut perintah yang digunakan untuk membuat basis data adalah sebagai berikut

   CREATE DATABASE RumahSakit_XYZ;

   Jika perintah tersebut berhasil dijalankan melalui aplikasi, maka akan muncul notifikasi **Query OK, 1 row affected** dan akan tercipta satu buah basis data dalam sistem pengelola basis data dengan nama RumahSakit_XYZ.

2. **CREATE TABLE.**

   Perintah CREATE TABLE digunakan untuk membuat tabel baru dengan memberikan nama dan menentukan atribut dan batasan awalnya. Atribut

ditentukan terlebih dahulu, dan setiap atribut diberi nama, tipe data untuk menentukan domain nilainya, dan batasan atribut apa pun, seperti NOT NULL. Kunci, Entity Integrity, dan Referential Integrity dapat ditentukan dalam pernyataan CREATE TABLE setelah atribut dideklarasikan, atau dapat ditambahkan nanti menggunakan perintah ALTER TABLE. Format penulisan untuk perintah ini adalah sebagai berikut:

```
CREATE TABLE NamaTabel(
  NamaField1 tipedata(panjang),
  NamaField2 tipedata(panjang),
  ...
  NamaFieldn tipedata(panjang),
  PRIMARY KEY (NamaField_key)
);
```

Contoh pembuatan tabel Pasien pada database RumahSakit_XYZ dengan struktur yang ditunjukkan pada tabel 8.2.

Tabel 8.2. Struktur tabel pasien.

Nama Field	TypeData	Panjang	Constrains
NoPasien	Varchar	10	PK, Not Null
NamaPasien	Varchar	30	Not Null
Jalan	Varchar	50	Not Null
Kel	Varchar	30	Not Null
Kec	Varchar	30	Not Null
Kota	Varchar	30	Not Null
KodePos	Varchar	5	Not Null
Telpon	Varchar	13	Not Null

Perintah DDL yang dijalankan untuk membuat tabel dengan struktur diatas adalah sebagai berikut:

```
CREATE TABLE Pasien (
  NoPasien VARCHAR(10) NOT NULL,
  Daftar_NoPend VARCHAR(20) NOT NULL,
  NamaPasien VARCHAR(30) NOT NULL,
  Jalan VARCHAR(50) NOT NULL,
  Kel VARCHAR(30) NOT NULL,
  Kec VARCHAR(30) NOT NULL,
  Kota VARCHAR(30) NOT NULL,
  KodePos VARCHAR(5) NOT NULL,
  Telpon VARCHAR(13) NOT NULL,
  PRIMARY KEY(NoPasien)
);
```

3. CREATE INDEX.

Indeks pada atribut relasi adalah struktur data yang memungkinkan sistem database untuk menemukan tupel tersebut dalam relasi yang memiliki nilai tertentu untuk atribut tersebut secara efisien, tanpa memindai semua tupel relasi (Silberschatz et al., 2019). Format penulisan perintah pembuatan INDEX pada tabel adalah sebagai berikut:

CREATE INDEX NamaIndex ON NamaTabel (NamaField);

Jika ingin mendeklarasikan bahwa kunci pencarian adalah kunci kandidat, dapat menambahkan atribut UNIQUE ke INDEX yang dibuat dengan format penulisan sebagai berikut:

CREATE UNIQUE INDEX NamaIndex ON NamaTabel (NamaField);

Contoh pembuatan INDEX pada tabel Pasien field Telpon dengan nama PasienIdx.

CREATE INDEX PasienIdx ON Pasien (Telpon);

Hasil kompilasi perintah tersebut akan membentuk index pada field telpon pada tabel pasien seperti yang ditunjukkan pada gambar 8.1.

	Field	Type		Null	Key	Default	
☐	NoPasien	varchar(10)	11B	NO	PRI	(NULL)	OK
☐	NamaPasien	varchar(30)	11B	NO		(NULL)	OK
☐	Jalan	varchar(50)	11B	NO		(NULL)	OK
☐	Kel	varchar(30)	11B	NO		(NULL)	OK
☐	Kec	varchar(30)	11B	NO		(NULL)	OK
☐	Kota	varchar(30)	11B	NO		(NULL)	OK
☐	KodePos	varchar(5)	10B	NO		(NULL)	OK
☐	Telpon	varchar(13)	11B	NO	MUL	(NULL)	OK

Gambar 8.1 Hasil pembuatan Index pada tabel Pasien

4. CREATE VIEW.

View merupakan salah satu perintah yang terdapat dalam SQL yang digunakan untuk membuat tabel virtual berisi atribut dan nilai yang berasal dari kueri data satu tabel atau lebih(Garcia-Molina, Ullman, & Widom, 2008). Data yang terdapat dalam View selalu up-to-date seiring dengan perubahan yang dilakukan terhadap tabel sumber yang terlibat dalam view(Elmasri & Navathe, 2010). Format penulisan perintah pembuatan VIEW pada SQL adalah sebagai berikut:

CREATE VIEW NamaView AS KueriSelect;

Contoh pembuatan VIEW pada database RumahSakit_XYZ dengan nama pasienjkt.

```
CREATE VIEW pasienjkt AS
  SELECT nopasien, namapasien, CONCAT(jalan,
  kel, kec, kota, kodepos) as Alamat, telpon
```

```
FROM pasien WHERE kota='Jakarta';
```

5. **CREATE TRIGGER.**

Trigger merupakan salah satu fungsi yang terdapat pada SQL yang dapat digunakan untuk membuat sistem mengeksekusi perintah secara otomatis sebagai timbal balik dari modifikasi basis data (Silberschatz et al., 2019). Pada umumnya perintah Trigger akan dieksekusi sebelum atau sesudah perintah INSERT, UPDATE, DELETE dijalankan. Trigger juga merupakan mekanisme yang berguna untuk memperingatkan pengguna atau untuk memulai tugas tertentu secara otomatis ketika kondisi tertentu terpenuhi. Format penulisan perintah dalam pembuatan TRIGGER pada database adalah sebagai berikut:

```
CREATE TRIGGER nama_trigger
{BEFORE | AFTER} {INSERT | UPDATE| DELETE }
  ON nama_table
  FOR EACH ROW
BEGIN
  PERINTAH SQL
END;
```

Contoh pembuatan TRIGGER pada basis data Rumahsakit_XYZ dengan nama history_pasien.

```
CREATE TRIGGER history_pasien
  BEFORE UPDATE
  ON pasien
  FOR EACH ROW
BEGIN
  INSERT INTO logpasien
  set nopasien = OLD.nopasien,
  telpon_lama=old.telpon,
  telpon_baru=new.telpon,
  waktu = NOW();
END;
```

8.3 Perintah ALTER

Perintah Alter pada DDL digunakan untuk melakukan modifikasi atau perubahan struktur tabel yang telah diciptakan, tindakan ubah struktur tabel tersebut termasuk menambahkan atau menghapus kolom (atribut), mengubah definisi kolom, dan menambahkan atau menghapus tabel constrains (Elmasri & Navathe, 2010). Format penulisan perintah ALTER pada SQL adalah sebagai berikut:

1. **Menambahkan kolom (atribut)**

Format yang digunakan untuk menambahkan kolom dengan perintah ALTER pada SQL yaitu:

```
ALTER TABLE table_name
  ADD [COLUMN] [IF NOT EXISTS]
  column_name column_definition
[FIRST | AFTER column_name]
```

Contoh menambahkan kolom pada tabel pasien dengan nama tanggal_lahir dan tipe data date.

```
ALTER TABLE pasien ADD COLUMN IF NOT EXIST
tanggal_lahir date;
```

2. Menghapus kolom (atribut)

Format yang digunakan untuk menghapus kolom dengan perintah ALTER pada SQL yaitu:

```
ALTER TABLE table_name
  DROP [COLUMN] [IF EXISTS] column_name
```

Contoh menghapus kolom/atribut tanggal_lahir dari tabel pasien.

```
ALTER TABLE pasien
  DROP COLUMN tanggal_lahir;
```

3. Modifikasi kolom (atribut)

Format yang digunakan untuk modifikasi kolom dengan perintah ALTER pada SQL yaitu:

```
ALTER TABLE table_name
  MODIFY [COLUMN] column_name column_definition
[FIRST | AFTER column_name]
```

Contoh perintah modifikasi kolom/atribut telpon dari tabel pasien, yaitu mengubah tipe data menjadi CHAR(15) dan memindahkan posisi ke setelah atribut nama_pasien.

```
ALTER TABLE pasien MODIFY telpon char(15)
  AFTER nama_pasien;
```

8.4 Perintah DROP

Perintah DROP pada SQL merupakan perintah yang digunakan untuk menghapus DATABASE, TABLE, INDEX, VIEW dan TRIGGER. Perintah DROP juga dapat digunakan untuk menghapus jenis elemen skema bernama lainnya, seperti constrain atau domain (Elmasri & Navathe, 2010).

1. Menghapus DATABASE.

Format penulisan perintah untuk menghapus database yaitu

DROP DATABASE nama_database;

Berikut adalah contoh perintah untuk menghapus database Rumahsakit_XYZ dengan menggunakan DROP.

DROP DATABASE Rumahsakit_XYZ;

2. **Menghapus TABLE.**
   Format penulisan perintah untuk menghapus tabel yaitu

   DROP TABLE nama_table;

   Berikut adalah contoh perintah untuk menghapus tabel pasien dengan menggunakan DROP.

   DROP TABLE pasien;

3. **Menghapus INDEX.**
   Format penulisan perintah untuk menghapus index pada tabel yaitu

   DROP INDEX nama_index ON nama_table;

   Berikut adalah contoh perintah untuk menghapus index pasienidx dengan menggunakan DROP.

   DROP INDEX PasienIdx ON pasien;

4. **Menghapus VIEW.**
   Format penulisan perintah untuk menghapus view yaitu

   DROP VIEW nama_view;

   Berikut adalah contoh perintah untuk menghapus view pasienjkt dari tabel database Rumahsakit_XYZ dengan menggunakan DROP.

   DROP VIEW pasienjkt;

5. **Menghapus TRIGGER.**
   Format penulisan perintah untuk menghapus trigger yaitu

   DROP TRIGGER nama_trigger;

   Berikut adalah contoh perintah untuk menghapus trigger history_pasien pada database Rumahsakit_XYZ dengan menggunakan DROP.

   DROP TRIGGER history_pasien;

8.5 Perintah RENAME

Perintah RENAME dalam SQL digunakan untuk melakukan perubahan nama pada Tabel, dapat dilakukan pada satu atau lebih tabel. Untuk menjalankan perintah RENAME pada tabel, user harus memiliki hak akses istimewa ALTER dan DROP untuk

tabel asli serta CREATE dan INSERT untuk tabel baru. Format penulisan perintah untuk RENAME yaitu:

```
RENAME TABLE nama_table_lama TO nama_table_baru;
                    atau
RENAME TABLE nama_table_lama1 TO nama_table_baru1,
  nama_table_lama2 TO nama_table_baru2,
  nama_table_lama3 TO nama_table_baru3;
```

Contoh penulisan perintah mengubah nama tabel pasien menjadi data_pasien pada database Rumahsakit_XYZ dengan menggunakan perintah RENAME.

```
RENAME TABLE pasien TO data_pasien;
```

8.6 Perintah TRUNCATE

TRUNCATE merupakan salah satu perintah DDL pada SQL yang digunakan untuk menghapus atau mengosongkan data yang terdapat pada sebuah tabel(Gupta & Mittal, 2017). Pada dasarnya cara kerja TRUNCATE mirip dengan perintah DELETE, yang membedakan adalah perintah TRUNCATE lebih sederhana atau lebih singkat dibandingkan dengan perintah DELETE. Beberapa karakteristik dari perintah TRUNCATE, yaitu:

1. Tidak dapat menggunakan clausa WHERE pada statemen.
2. Perintah yang dieksekusi akan menghapus data pada tabel secara keseluruhan.
3. Data yang telah dihapus tidak bisa di rollback.
4. Waktu eksekusi perintah lebih cepat.

Format penulisan perintah TRUNCATE adalah:

```
TRUNCATE TABLE nama_tabel;
```

Contohnya menghapus seluruh data yang terdapat pada tabel pasien adalah:

```
TRUNCATE TABLE pasien;
```

Pada perintah TRUNCATE jika dieksekusi maka akan mereset sequence yang terdapat di tabel tersebut jika salah satu kolom/atributnya menggunakan constrain Auto Increment(AI).

Bab 9
Data Manipulation Language

9.1 Jenis Perintah Data Manipulation Language

Data manipulation language (DML) merupakan bagian dari SQL yang digunakan untuk melakukan manipulasi data pada tabel database relasional (Hariyanto, 2004). Operasi manipulasi data secara garis besar meliputi empat fungsi, yaitu: pemilihan data, penyisipan data, pembaharuan data, dan penghapusan data (Coronel & Morris, 2017; Gupta & Mittal, 2017; Hariyanto, 2004; Hoffer et al., 2011). Jenis perintah dasar DML ditunjukkan pada Tabel 9.1.

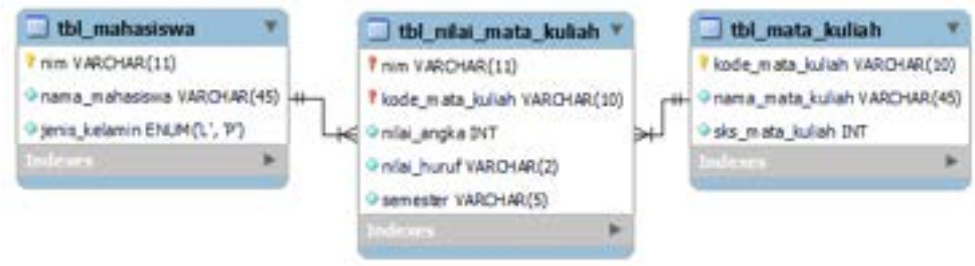
Tabel 9.1 Jenis Perintah Dasar DML

Perintah	Fungsi
SELECT	Perintah untuk memilih data dari satu atau beberapa tabel. Perintah ini dapat dikombinasikan dengan klausa tertentu untuk menghasilkan data yang sesuai dengan keinginan.
INSERT	Perintah untuk menyisipkan satu atau beberapa data baru ke dalam tabel.
UPDATE	Perintah untuk memperbaharui satu atau beberapa data yang sesuai dengan kriteria. Penentuan kriteria data dapat dilakukan dengan menggunakan klausa WHERE yang sederhana ataupun kompleks.
DELETE	Perintah untuk menghapus satu atau beberapa data yang sesuai dengan kriteria. Penentuan kriteria data dapat dilakukan dengan menggunakan klausa WHERE yang sederhana ataupun kompleks.

Perintah-perintah DML dapat dituliskan dengan berbagai cara sesuai kebutuhan. Bagi seorang pengguna ahli atau *database developer*, perintah DML dapat dituliskan melalui *console* atau aplikasi dengan interface grafis. Hal ini biasanya dilakukan untuk

pengujian database atau ketika menangani *toubleshoot* database. Perintah DML juga bisa disisipkan pada berbagai bahasa pemrograman.

Pada buku ini akan digunakan sebuah contoh kasus database dengan nama db_akademik. Database db_akademik terdiri atas tiga buah tabel, yaitu: tbl_mahasiswa, tbl_mata_kuliah, dan tbl_nilai_mata_kuliah. Desain struktur database db_akademik ditunjukkan seperti Gambar 9.1.



Gambar 9.1 Desain Struktur Database Contoh Kasus

9.2 Perintah SELECT

Perintah SELECT berfungsi untuk memilih data dari satu atau beberapa tabel (Connolly & Begg, 2015; Hariyanto, 2004). Perintah SELECT dapat berisi sintaks untuk memilih kolom, memilih baris, menggabungkan beberapa tabel, mengelompokkan data, dan menerapkan beberapa perhitungan sederhana (Harrington, 2016). Perintah SELECT tidak akan melakukan perubahan apa pun pada data.

Perintah SELECT dapat terdiri atas dua klausa wajib dan beberapa klausa tambahan seperti pada Tabel 9.2.

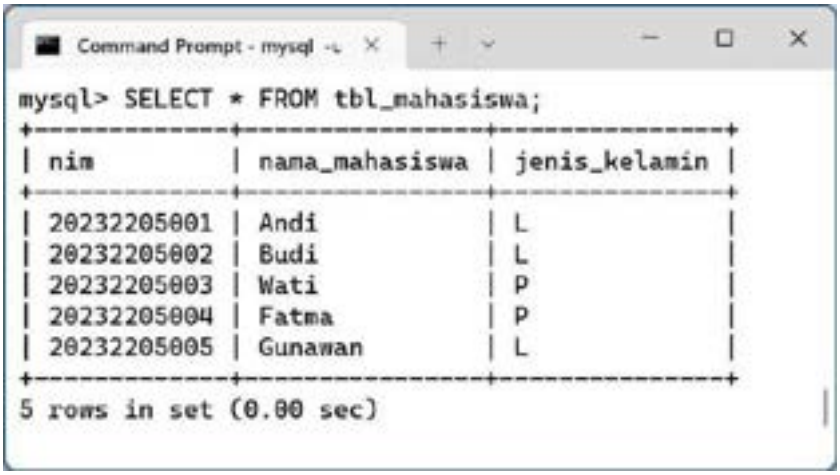
Tabel 9.2 Klausa Pada Perintah SELECT

Klausa	Keterangan
SELECT	Menentukan nama kolom yang akan ditampilkan pada keluaran. Nilai yang akan ditampilkan juga dapat diberikan fungsi agregat seperti COUNT, SUM, AVG, MIN, dan MAX.
FROM	Menentukan nama tabel yang digunakan.
WHERE	Melakukan penyaringan baris-baris data yang memenuhi kondisi tertentu.
GROUP BY	Membentuk kelompok-kelompok baris data dengan nilai kolom yang sama.
HAVING	Melakukan penyaringan kelompok-kelompok baris data yang memenuhi kondisi tertentu.
ORDER BY	Menentukan aturan pengurutan baris data yang ditampilkan pada keluran.

Klausa wajib adalah SELECT dan FROM, sedangkan selebihnya bersifat opsional (Connolly & Begg, 2015; Elmasri & Navathe, 2016; Hariyanto, 2004; Hoffer et al., 2011). Perintah SELECT memiliki banyak variasi untuk menghasilkan keluaran yang diinginkan. Jika diinginkan menampilkan semua data dari tabel tbl_mahasiswa (sesuai struktur database pada Gambar 9.1), maka dituliskan perintah SELECT sebagai berikut:

SELECT * FROM tbl_mahasiswa;

Hasil eksekusi perintah SELECT tersebut diperlihatkan seperti pada Gambar 9.2 Berikut:



Gambar 9.2 Contoh Hasil Eksekusi Perintah SELECT Sederhana

Tanda * pada perintah SELECT di atas menunjukkan bahwa semua kolom dari tabel tbl_mahasiswa akan ditampilkan pada keluaran. Jika diinginkan hanya menampilkan kolom tertentu saja, misal nim dan nama_mahasiswa, maka perintah SELECT dapat ditulis sebagai berikut:

SELECT nim, nama_mahasiswa FROM tbl_mahasiswa;

Nama kolom pada keluaran juga dapat diganti menggunakan nama alias AS seperti pada perintah SELECT berikut ini:

SELECT nim AS nomor_induk_mahasiswa, nama_mahasiswa AS nama_lengkap_mahasiswa FROM tbl_mahasiswa;

Pada kondisi tertentu, diinginkan menampilkan data dengan kriteria tertentu. Hal ini berbeda dengan perintah SELECT sebelumnya yang menampilkan seluruh data. Perintah SELECT dapat dikombinasikan dengan klausa WHERE untuk melakukan penyaringan baris data yang memenuhi kondisi tertentu.

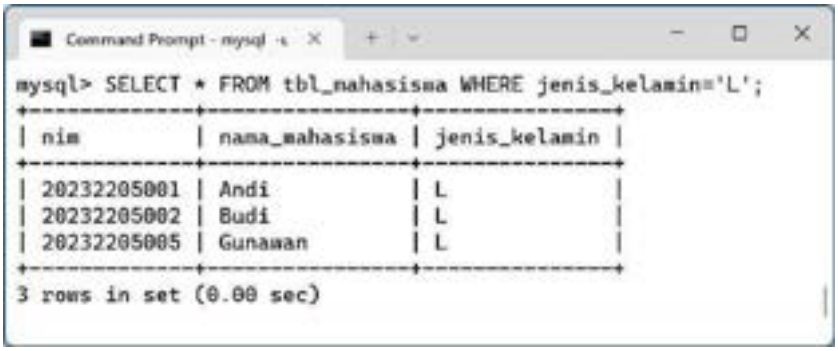
Perintah SELECT memungkinkan penggunaan penghubung logika AND, OR, dan NOT pada klausa WHERE untuk menggabungkan beberapa aturan penyaringan (Nugroho, 2004). Di samping itu kita juga bisa menggunakan berbagai jenis operator untuk menentukan kriteria data yang akan disaring. Pada Tabel 9.3 ditunjukkan jenis operator yang dapat digunakan pada klausa WHERE (Connolly & Begg, 2015; Gupta & Mittal, 2017).

Tabel 9.3 Jenis Operator Pada Penggunaan Klausa WHERE

Jenis Operator	Keterangan
Comparison	Melakukan penyaringan data dengan melakukan perbandingan nilai pada sebuah kolom. Contoh operator *comparison,* yaitu: sama dengan (=), lebih kecil (<), lebih besar (>), lebih kecil atau sama dengan (<=), lebih besar atau sama dengan (>=), dan tidak sama dengan (<>).
Range	Melakukan penyaringan data yang berada pada rentang tertentu. Contoh operator *range* adalah: BETWEEN.
Set Membership	Melakukan penyaringan data yang termasuk ke dalam keanggotaan tertentu. Contoh operator *set membership* adalah: IN dan NOT IN.
Pattern Match	Melakukan penyaringan data yang memenuhi pola tertentu. Contoh operator *pattern match* adalah: LIKE dan NOT LIKE.
Null	Melakukan penyaringan data yang memiliki nilai *null*. Contoh operator *null* adalah: IS NULL dan IS NOT NULL.

Jika kita ingin menampilkan data mahasiswa dari tbl_mahasiswa yang berjenis kelamin laki-laki, maka dapat digunakan perintah SELECT berikut:

SELECT * FROM tbl_mahasiswa WHERE jenis_kelamin='L';



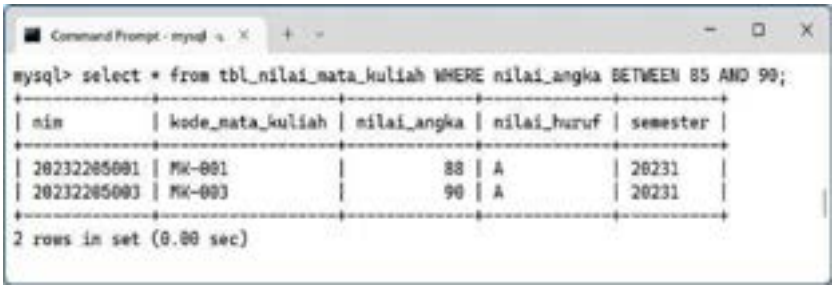
Gambar 9.3 Contoh Hasil Eksekusi Perintah SELECT dengan Klausa WHERE dan Operator Sama Dengan

Hasil eksekusi perintah SELECT tersebut diperlihatkan seperti pada Gambar 9.3. Pada Gambar 9.3 terlihat bahwa data mahasiswa yang ditampilkan hanya yang berjenis kelamin laki-laki (jenis_kelamin=L).

Operator BETWEEN digunakan untuk menyaring data yang memenuhi syarat rentang nilai tertentu. Jika kita ingin menampilkan data dari tabel tbl_nilai_mata_kuliah dengan nilai angka antara 85 dan 90 (sesuai struktur database pada Gambar 9.1), maka dituliskan perintah SELECT sebagai berikut:

SELECT * FROM tbl_nilai_mata_kuliah WHERE nilai_angka BETWEEN 85 AND 90;

Hasil eksekusi perintah SELECT tersebut diperlihatkan seperti pada Gambar 9.4. Pada Gambar 9.4 terlihat bahwa data nilai mata kuliah yang ditampilkan pada keluaran memiliki nilai angka paling rendah 85 dan paling tinggi 90.



Gambar 9.4 Contoh Hasil Eksekusi Perintah SELECT dengan Klausa WHERE dan Operator BETWEEN

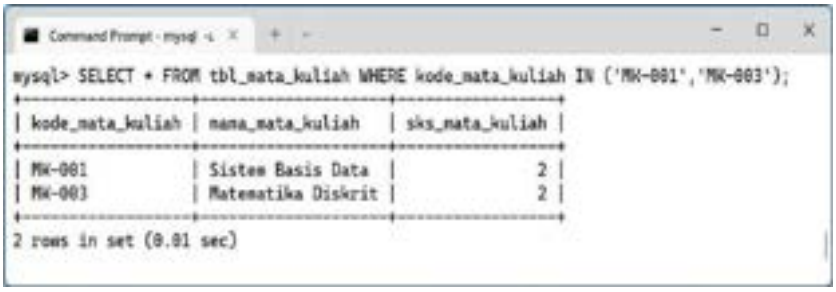
Operator IN dan NOT IN masing-masing akan memeriksa apakah sebuah data berada atau tidak berada di dalam keanggotaan yang ditentukan. Jika kita ingin menampilkan data dari tabel tbl_mata_kuliah (sesuai struktur database pada Gambar 9.1) di mana kode mata kuliah yang diizinkan tampil pada keluaran adalah MK-001 dan MK-003, maka dapat dituliskan perintah SELECT sebagai berikut:

SELECT * FROM tbl_ mata_kuliah WHERE kode_mata_kuliah IN ('MK-001','MK-003');

Hasil eksekusi perintah SELECT tersebut diperlihatkan seperti pada Gambar 9.5. Pada Gambar 9.5 terlihat bahwa data mata kuliah yang ditampilkan pada keluaran hanya yang memiliki kode mata kuliah MK-001 atau MK-003.

Penggunaan operator = seperti pada Gambar 9.3 akan menghasilkan data dengan nilai yang harus tepat sama dengan yang ditentukan. Jika kita ingin melakukan pencarian data mahasiswa pada tbl_mahasiswa berdasarkan nama mahasiswa, maka penggunaan operator = tidak efektif. Hal ini disebabkan karena kemungkinan sulit menuliskan nama

mahasiswa secara lengkap dan tepat. Oleh karena itu, SQL menyediakan operator LIKE yang dapat digunakan untuk mencari data berdasarkan pola tertentu.



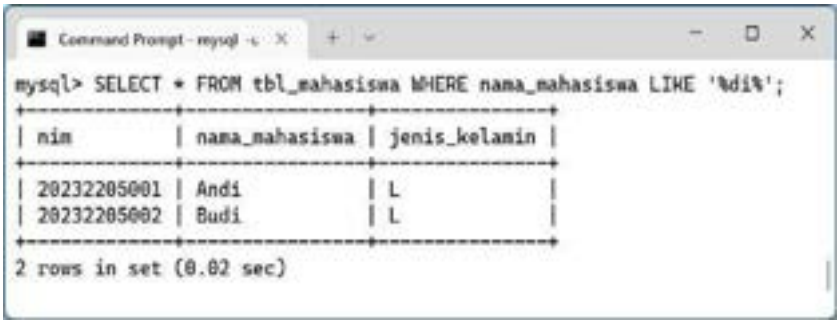
Gambar 9.5 Contoh Hasil Eksekusi Perintah SELECT dengan Klausa WHERE dan Operator IN

Operator LIKE memiliki simbol khusus untuk membentuk pola data yang diinginkan. Simbol % (persen) digunakan untuk merepresentasikan 0 atau lebih karakter sembarang. Simbol _ *(underscore)* digunakan untuk merepresentasikan satu karakter sembarang. Jika diinginkan hasil pencarian mengandung karakter tertentu, maka karakter tersebut dituliskan secara eksplisit pada operator LIKE (Gupta & Mittal, 2017; Nugroho, 2004).

Jika kita ingin menampilkan data mahasiswa dengan nama mahasiswa mengandung karakter "di", maka kita bisa menuliskan perintah SELECT sebagai berikut:

SELECT * FROM tbl_mahasiswa WHERE nama_mahasiswa LIKE '%di%';

Hasil eksekusi perintah SELECT tersebut diperlihatkan seperti pada Gambar 9.6. Pada Gambar 9.6 terlihat bahwa data mahasiswa yang ditampilkan pada keluaran memiliki nama mahasiswa yang mengandung karakter "di".



Gambar 9.6 Contoh Hasil Eksekusi Perintah SELECT dengan Klausa WHERE dan Operator LIKE

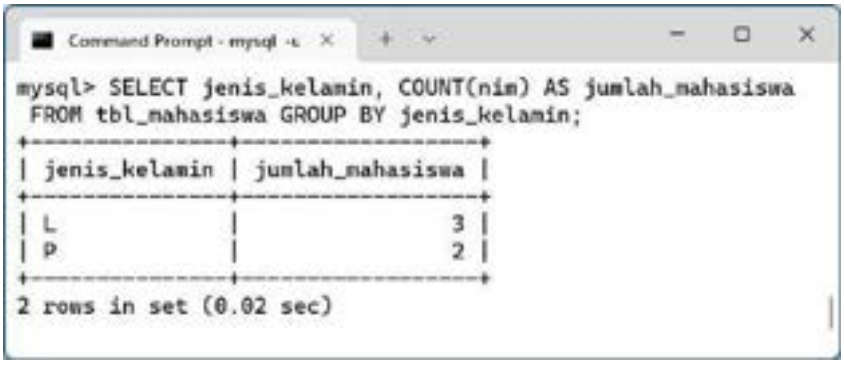
Operator IS NULL dan IS NOT NULL masing-masing akan menyaring data yang memiliki nilai null dan tidak null pada kolom yang ditentukan. Jika kita ingin menampilkan data mahasiswa dari tbl_mahasiswa di mana kolom jenis kelamin belum diisi, maka dituliskan perintah SELECT sebagai berikut:

SELECT * FROM tbl_mahasiswa WHERE jenis_kelamin IS NULL;

Data yang ditampilkan menggunakan perintah SELECT dapat dikelompokkan berdasarkan kesamaan nilai kolom tertentu. Klausa yang digunakan untuk mengelompokkan data adalah GROUP BY (Connolly & Begg, 2015). Klausa ini diikuti dengan nama kolom yang akan dijadikan sebagai dasar pengelompokan data. Jika kita ingin menampilkan rekapitulasi jumlah mahasiswa dari tabel tbl_mahasiswa berdasarkan jenis kelamin, maka dituliskan perintah SELECT sebagai berikut:

SELECT jenis_kelamin, COUNT(nim) AS jumlah_mahasiswa FROM tbl_mahasiswa GROUP BY jenis_kelamin;

Hasil eksekusi perintah SELECT tersebut diperlihatkan seperti pada Gambar 9.7. Pada Gambar 9.7 terlihat hasil rekapitulasi jumlah mahasiswa berdasarkan jenis kelamin. Kolom jumlah_mahasiswa menggunakan fungsi agregat COUNT yang berfungsi untuk mencacah jumlah data setiap kelompok.



Gambar 9.7 Contoh Hasil Eksekusi Perintah SELECT dengan Klausa GROUP BY

Penyaringan data berkelompok dapat dilakukan dengan menggunakan klausa HAVING. Jika kita ingin menampilkan kembali data mahasiswa seperti pada Gambar 9.7 tetapi dengan syarat jumlah mahasiswa lebih besar atau sama dengan 3, maka dituliskan perintah SELECT sebagai berikut:

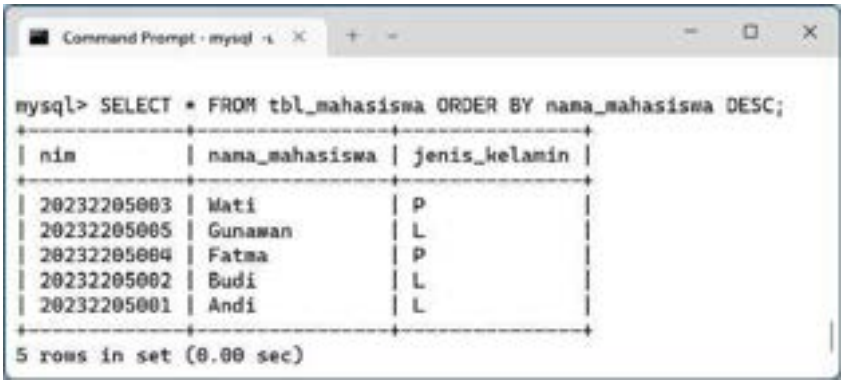
SELECT jenis_kelamin, COUNT(nim) AS jumlah_mahasiswa FROM tbl_mahasiswa GROUP BY jenis_kelamin HAVING jumlah_mahasiswa >= 3;

Data yang ditampilkan melalui perintah SELECT dapat diurutkan berdasarkan satu atau lebih kolom tertentu baik secara *ascending* maupun *descending*. Pengurutan data pada keluaran dapat dilakukan dengan menggunakan klausa ORDER BY diikuti dengan nama kolom yang menjadi dasar pengurutan. Secara *default*, metode pengurutan yang digunakan adalah *ascending* (Connolly & Begg, 2015; Hariyanto, 2004; Hoffer et al., 2011; Nugroho, 2004).

Jika kita ingin menampilkan data mahasiswa dari tabel tbl_mahasiswa dengan melakukan pengurutan berdasarkan nama mahasiswa secara *descending*, maka dituliskan perintah SELECT sebagai berikut:

SELECT * FROM tbl_mahasiswa ORDER BY nama_mahasiswa DESC;

Hasil eksekusi perintah SELECT tersebut diperlihatkan seperti pada Gambar 9.8. Pada Gambar 9.8 data mahasiswa ditampilkan dengan urutan nama mahasiswa secara *descending*.



Gambar 9.8 Contoh Hasil Eksekusi Perintah SELECT dengan Klausa ORDER BY

Perintah SELECT sebelumnya hanya menampilkan data dari sebuah tabel. Bahasa SQL memungkinkan kita menampilkan data yang berasal dari beberapa tabel. Klausa yanag digunakan untuk menampilkan data dari beberapa tabel adalah JOIN. Perintah JOIN terdiri atas CROSS, INNER, dan OUTER. INNER JOIN terdiri atas NATURAL JOIN, JOIN USING, dan JOIN ON. OUTER JOIN terdiri atas LEFT JOIN, RIGHT JOIN, dan FULL JOIN. (Coronel & Morris, 2017; Hoffer et al., 2011). Contoh perintah SELECT dengan CROSS JOIN adalah sebagai berikut:

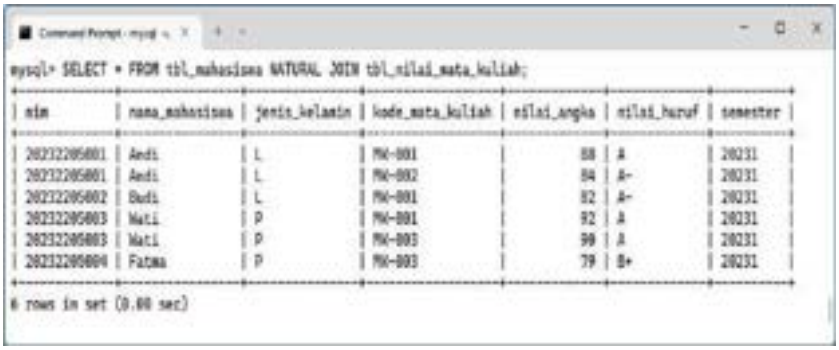
SELECT * FROM tbl_mahasiswa CROSS JOIN tbl_mata_kuliah

Perintah di atas akan menampilkan data dengan cara memasangkan semua data dari tabel tbl_mahasiswa dengan semua data pada tabel tbl_mata_kuliah *(cartesian product).* Hasil CROSS JOIN tidak mempertimbangkan kecocokan data antara kedua tabel.

Perintah SELECT dengan INNER JOIN akan menampilkan data yang cocok antara kedua tabel. Contoh perintah SELECT dengan INNER JOIN sebagai berikut:

SELECT * FROM tbl_mahasiswa NATURAL JOIN tbl_nilai_mata_kuliah

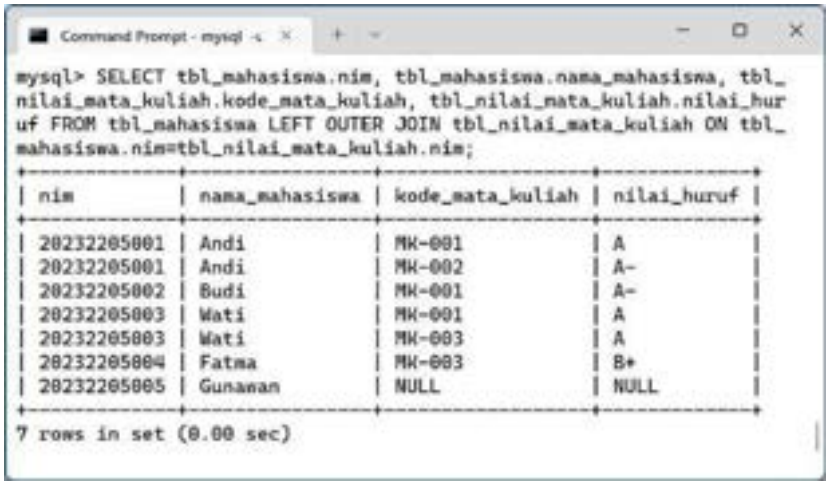
Perintah tersebut akan menampilkan data yang cocok antara tabel tbl_mahasiswa dan tabel tbl_nilai_mata_kuliah. Hasil perintah tersebut seperti ditampilkan pada Gambar 9.9.



Gambar 9.9 Contoh Hasil Eksekusi Perintah SELECT dengan Klausa NATURAL JOIN

Perintah SELECT dengan OUTER JOIN akan menampilkan semua data dari tabel pertama dengan semua data yang bersesuaian pada tabel lainnya. Perintah LEFT OUTER JOIN akan menampilkan semua data dari tabel sebelah kiri dengan semua data yang bersesuaian pada tabel sebelah kanan (Coronel & Morris, 2017; Hoffer et al., 2011). Contoh perintah SELECT dengan LEFT OUTER JOIN adalah sebagai berikut:

SELECT tbl_mahasiswa.nim, tbl_mahasiswa.nama_mahasiswa, tbl_nilai_mata_kuliah.kode_mata_kuliah, tbl_nilai_mata_kuliah.nilai_huruf FROM tbl_mahasiswa LEFT OUTER JOIN tbl_nilai_mata_kuliah ON tbl_mahasiswa.nim=tbl_nilai_mata_kuliah.nim;



```
Command Prompt - mysql -u
mysql> SELECT tbl_mahasiswa.nim, tbl_mahasiswa.nama_mahasiswa, tbl_
nilai_mata_kuliah.kode_mata_kuliah, tbl_nilai_mata_kuliah.nilai_hur
uf FROM tbl_mahasiswa LEFT OUTER JOIN tbl_nilai_mata_kuliah ON tbl_
mahasiswa.nim=tbl_nilai_mata_kuliah.nim;
+-------------+----------------+-----------------+--------------+
| nim         | nama_mahasiswa | kode_mata_kuliah | nilai_huruf |
+-------------+----------------+-----------------+--------------+
| 20232205001 | Andi           | MK-001          | A            |
| 20232205001 | Andi           | MK-002          | A-           |
| 20232205002 | Budi           | MK-001          | A-           |
| 20232205003 | Wati           | MK-001          | A            |
| 20232205003 | Wati           | MK-003          | A            |
| 20232205004 | Fatma          | MK-003          | B+           |
| 20232205005 | Gunawan        | NULL            | NULL         |
+-------------+----------------+-----------------+--------------+
7 rows in set (0.00 sec)
```

Gambar 9.10 Contoh Hasil Eksekusi Perintah SELECT dengan Klausa LEFT OUTER JOIN

Hasil eksekusi perintah SELECT tersebut diperlihatkan seperti pada Gambar 9.10. Pada Gambar 9.10 seluruh data dari tabel tbl_mahasiswa ditampilkan beserta data yang sesuai pada tabel tbl_nilai_mata_kuliah berdasarkan kolom nim. Jika sebuah data pada tabel tbl_mahasiswa tidak memiliki relasi pada tabel tbl_nilai_mata_kuliah, maka akan ditampilkan data null.

Perintah RIGHT OUTER JOIN merupakan kebalikan dari LEFT OUTER JOIN. Perintah ini akan menampilkan semua data dari tabel di sebelah kanan beserta dengan data yang sesuai pada tabel sebelah kiri. Sedangkan perintah FULL OUTER JOIN akan menampilkan data yang sesuai dari kedua tabel (Coronel & Morris, 2017; Hoffer et al., 2011).

9.3 Perintah INSERT

Perintah INSERT digunakan untuk menyisipkan baris data baru ke dalam sebuah tabel (Connolly & Begg, 2015; Coronel & Morris, 2017; Elmasri & Navathe, 2016). Bentuk umum perintah INSERT adalah sebagai berikut:

INSERT INTO nama_tabel [(list_nama_kolom)] VALUES (list_data);

Pada perintah INSERT, nama kolom yang akan disisipi data bersifat opsional. Apabila nama kolom tidak didefenisikan, maka semua kolom akan diisi data sesuai urutan pada tabel. Jika kita ingin menyisipkan data baru pada tabel tbl_mahasiswa (sesuai struktur database pada Gambar 9.1), maka dituliskan perintah INSERT sebagai berikut:

INSERT INTO tbl_mahasiswa VALUES (‘20232205123‘,‘Pasnur‘,‘L‘);

Perintah tersebut akan menyisipkan sebuah baris data baru ke dalam tabel tbl_mahasiswa dengan data nim='20232205123', nama_mahasiswa='Pasnur', dan jenis_kelamin='L'. Jika diinginkan hanya menginput kolom nim dan nama_mahasiswa saja, maka perintah tersebut menjadi:

INSERT INTO tbl_mahasiswa (nim, nama_mahasiswa) VALUES ('20232205123','Pasnur');

9.4 Perintah UPDATE

Perintah UPDATE digunakan untuk melakukan modifikasi pada satu atau beberapa data pada sebuah tabel. Jumlah data yang mengalami perubahan tergantung pada persyaratan yang dituliskan pada klausa WHERE (Connolly & Begg, 2015; Elmasri & Navathe, 2016). Bentuk umum perintah UPDATE adalah sebagai berikut:

UPDATE nama_tabel SET nama_kolom_1=data_kolom_1 [,nama_kolom_2=data_kolom_2 ...] [WHERE kondisi_persyaratan];

Pada perintah UPDATE kita dapat melakukan modifikasi pada satu atau beberapa kolom yang didefinisikan. Jumlah kolom yang dimodifikasi minimal satu. Penggunaan klausa WHERE berfungsi untuk mendefinisikan kriteria data yang akan dimodifikasi. Penggunaan klausa WHERE bersifat opsional, akan tetap jika tidak digunakan maka semua data pada tabel akan mengalami modifikasi. Contoh perintah UPDATE untuk modifikasi data yang telah disisipkan sebelumnya pada perintah INSERT adalah sebagai berikut:

UPDATE tbl_mahasiswa SET nim='20232205099', nama_mahasiswa='Aisyah' WHERE nim='20232205123';

9.5 Perintah DELETE

Perintah DELETE digunakan untuk menghapus satu atau beberapa data pada sebuah tabel. Jumlah data yang dihapus tergantung pada persyaratan yang dituliskan pada klausa WHERE (Connolly & Begg, 2015; Elmasri & Navathe, 2016). Bentuk umum perintah DELETE adalah sebagai berikut:

DELETE FROM nama_tabel [WHERE kondisi_persyaratan];

Penggunaan klausa WHERE pada perintah DELETE bersifat opsional, akan tetapi jika tidak digunakan maka semua data pada tabel akan dihapus. Contoh perintah DELETE untuk menghapus sebuah data pada tabel tbl_mahasiswa adalah sebagai berikut:

DELETE FROM tbl_mahasiswa WHERE nim='20232205099';

Bab 10
Fungsi Di MySQL

10.1 Pengertian Fungsi (Function)

Fungsi (*Function*) atau bisa disebut metode (*method*) adalah subprogram yang berisikan langkah-langkah perintah untuk mengerjakan suatu proses tertentu sesuai dengan tugas dari fungsi tersebut. Di dalam MySQL terdapat banyak fungsi yang sudah disediakan untuk tugas tertentu. Dengan menggunakan fungsi-fungsi tersebut maka akan memudahkan dalam mengelola basis data yang sudah dibangun. Di dalam MySQL ada dua jenis fungsi yaitu *Aggregate Function* dan *Scalar Function.*

10.2 Aggregate Function

Fungsi *aggregate* adalah fungsi untuk menghitung nilai yang diperoleh dari kolom pada tabel untuk menghasilkan nilai balik yang berupa nilai tunggal. Di dalam MySQL terdapat fungsi-fungsi *aggregate* yang sering digunakan adalah:

1. AVG()
2. COUNT()
3. SUM()
4. MAX()
5. MIN()

Fungsi-fungsi *aggregate* di atas sering dipakai dalam pengelolaan basis data. Di bawah ini akan dijelaskan penggunakan dan contoh kasus.

1. AVG()

avg() berfungsi untuk menghitung nilai rata-rata pada kolom atau field atau atribut. Terdapat tabel pembayaran bimbingan belajar di bawah ini:

```
MariaDB [bimbinganbelajar]> select * from pembayaran;
+-------------------+----------+-----------------+--------------------+--------------------+
| Kode_Pembayaran | status   | tgl_pembayaran | jumlah_pembayaran | Kode_Pendaftaran |
+-------------------+----------+-----------------+--------------------+--------------------+
| B001              | cash     | 2023-03-30     |             500000 | P01              |
| B002              | cash     | 2023-03-30     |             500000 | P02              |
| B003              | transfer | 2023-03-30     |            1000000 | P03              |
| B004              | transfer | 2023-03-30     |             700000 | P04              |
| B005              | transfer | 2023-03-30     |             500000 | P04              |
| B006              | cash     | 2023-03-30     |             200000 | P01              |
+-------------------+----------+-----------------+--------------------+--------------------+
6 rows in set (0.171 sec)
```

Contoh kasus 1: Berapa jumlah rata rata nominal pembayaran peserta bimbingan belajar.

Query 1 sebagai berikut:

```
SELECT AVG(JUMLAH_PEMBAYARAN)
FROM pembayaran;
```

Maka hasil query kasus 1 adalah:

```
+------------------------+
| AVG(JUMLAH_PEMBAYARAN) |
+------------------------+
|            566666.6667 |
+------------------------+
1 row in set (0.016 sec)
```

Contoh kasus 2: Berapa rata rata nominal pembayaran peserta bimbingan belajar dengan merubah judul kolom menjadi RATA-RATA.

Query 2 sebagai berikut:

```
SELECT AVG(jumlah_pembayaran) AS RATA-RATA
FROM pembayaran;
```

Maka hasil query kasus 2 adalah:

```
+-------------+
| RATA_RATA   |
+-------------+
| 566666.6667 |
+-------------+
1 row in set (0.007 sec)
```

Contoh kasus 3: Berapa rata rata nominal pembayaran peserta bimbingan belajar yang berstatus CASH.

Query 3 sebagai berikut:

```
SELECT AVG(jumlah_pembayaran) AS RATA-RATA
FROM pembayaran
WHERE status = "CASH";
```

Maka hasil query kasus 3 adalah:

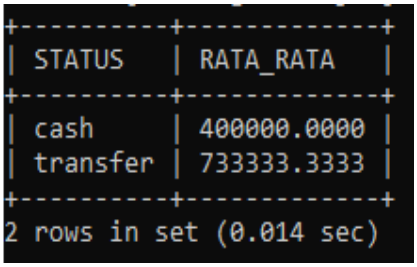
```
+-------------+
| RATA_RATA   |
+-------------+
| 400000.0000 |
+-------------+
1 row in set (0.015 sec)
```

Contoh kasus 4: Berapa rata rata nominal pembayaran peserta bimbingan belajar masing-masing status.

Query 4 sebagai berikut:

```
SELECT STATUS, AVG(jumlah_pembayaran) AS RATA-RATA
FROM pembayaran
GROUP BY STATUS;
```

Maka hasil query kasus 3 adalah:



```
+----------+-------------+
| STATUS   | RATA_RATA   |
+----------+-------------+
| cash     | 400000.0000 |
| transfer | 733333.3333 |
+----------+-------------+
2 rows in set (0.014 sec)
```

2. COUNT()

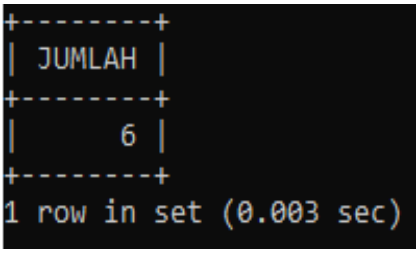
count() berfungsi untuk menghitung banyaknya baris atau record pada nilai suatu kolom atau field atau atribut.

Contoh kasus 5: Berapa jumlah nota pembayaran peserta bimbingan belajar.

Query 5 sebagai berikut:

```
SELECT COUNT(*) AS JUMLAH
FROM pembayaran;
```

Maka hasil query kasus 5 adalah:



```
+--------+
| JUMLAH |
+--------+
|      6 |
+--------+
1 row in set (0.003 sec)
```

Contoh kasus 6: Berapa jumlah nota pembayaran peserta bimbingan belajar yang berstatus CASH.

Query 6 sebagai berikut:

```
SELECT COUNT(*) AS JUMLAH
FROM pembayaran;
WHERE status = “CASH“;
```

Maka hasil query kasus 6 adalah:

```
+--------+
| JUMLAH |
+--------+
|      3 |
+--------+
1 row in set (0.009 sec)
```

Contoh kasus 7: Berapa jumlah nota pembayaran peserta bimbingan belajar masing-masing status.

Query 7 sebagai berikut:

```
SELECT COUNT(*) AS JUMLAH
FROM pembayaran;
GROUP BY status;
```

Maka hasil query kasus 7 adalah:

```
+----------+--------+
| STATUS   | JUMLAH |
+----------+--------+
| cash     |      3 |
| transfer |      3 |
+----------+--------+
2 rows in set (0.007 sec)
```

3. SUM()

sum() berfungsi untuk menghitung jumlah atau total nilai pada kolom atau field atau atribut.

Contoh kasus 8: Berapa total nominal pembayaran peserta bimbingan belajar.

Query 8 sebagai berikut:

```
SELECT SUM(JUMLAH_PEMBAYARAN) AS JUMLAH_BAYAR
FROM pembayaran;
```

Maka hasil query kasus 8 adalah:

```
+--------------+
| JUMLAH_BAYAR |
+--------------+
|      3400000 |
+--------------+
1 row in set (0.002 sec)
```

Contoh kasus 9: Berapa total nominal pembayaran peserta bimbingan belajar yang berstatus TRANSFER.

Query 9 sebagai berikut:

```
SELECT SUM(JUMLAH_PEMBAYARAN) AS JUMLAH_BAYAR
FROM pembayaran;
```

WHERE status = TRANSFER;

Maka hasil query kasus 9 adalah:

```
+--------------+
| JUMLAH_BAYAR |
+--------------+
|      2200000 |
+--------------+
1 row in set (0.008 sec)
```

Contoh kasus 10: Berapa total nominal pembayaran peserta bimbingan belajar masing-masing status.

Query 10 sebagai berikut:

```
SELECT SUM(JUMLAH_PEMBAYARAN) AS JUMLAH_BAYAR
FROM pembayaran;
GROUP BY status;
```

Maka hasil query kasus 10 adalah:

```
+----------+--------------+
| STATUS   | JUMLAH_BAYAR |
+----------+--------------+
| cash     |      1200000 |
| transfer |      2200000 |
+----------+--------------+
2 rows in set (0.010 sec)
```

4. MAX()

max() berfungsi untuk mencari nilai tertinggi pada kolom atau field atau atribut.

Contoh kasus 11: Berapa nilai terbesar nominal pembayaran peserta bimbingan belajar.

Query 11 sebagai berikut:

```
SELECT MAX(JUMLAH_PEMBAYARAN) AS MAKSIMUM
FROM pembayaran;
```

Maka hasil query kasus 11 adalah:

```
+----------+
| MAKSIMUM |
+----------+
|  1000000 |
+----------+
1 row in set (0.004 sec)
```

Contoh kasus 12: Berapa nilai terbesar nominal pembayaran peserta bimbingan belajar yang bersatus CASH.

Query 12 sebagai berikut:

```
SELECT MAX(JUMLAH_PEMBAYARAN) AS MAKSIMUM
FROM pembayaran
WHERE status = "CASH";
```

Maka hasil query kasus 12 adalah:

```
+----------+
| MAKSIMUM |
+----------+
|   500000 |
+----------+
1 row in set (0.004 sec)
```

Contoh kasus 13: Berapa nilai terbesar nominal pembayaran peserta bimbingan belajar masing-masing status.

Query 13 sebagai berikut:

```
SELECT MAX(JUMLAH_PEMBAYARAN) AS MAKSIMUM
FROM pembayaran
GROUP BY status;
```

Maka hasil query kasus 13 adalah:

```
+----------+----------+
| STATUS   | MAKSIMUM |
+----------+----------+
| cash     |   500000 |
| transfer |  1000000 |
+----------+----------+
2 rows in set (0.004 sec)
```

5. MIN()

min() berfungsi untuk mencari nilai terendah atau terkecil pada kolom atau field atau atribut.

Contoh kasus 14: Berapa nilai terkecil nominal pembayaran peserta bimbingan belajar.

Query 14 sebagai berikut:

```
SELECT MIN(JUMLAH_PEMBAYARAN) AS MINIMUM
FROM pembayaran;
```

Maka hasil query kasus 14 adalah:

```
+---------+
| MINIMUM |
+---------+
|  200000 |
+---------+
1 row in set (0.011 sec)
```

Contoh kasus 15: Berapa nilai terkecil nominal pembayaran peserta bimbingan belajar yang berstatus TRANSFER.

Query 15 sebagai berikut:

```
SELECT MIN(JUMLAH_PEMBAYARAN) AS MINIMUM
FROM pembayaran
WHERE status = "TRANSFER";
```

Maka hasil query kasus 15 adalah:

```
+---------+
| MINIMUM |
+---------+
|  500000 |
+---------+
1 row in set (0.002 sec)
```

Contoh kasus 16: Berapa nilai terkecil nominal pembayaran peserta bimbingan belajar masing-masing status.

Query 16 sebagai berikut:

```
SELECT MIN(JUMLAH_PEMBAYARAN) AS MINIMUM
FROM pembayaran
GROUP BY status;
```

Maka hasil query kasus 16 adalah:

```
+----------+---------+
| STATUS   | MINIMUM |
+----------+---------+
| cash     |  200000 |
| transfer |  500000 |
+----------+---------+
2 rows in set (0.007 sec)
```

10.3 Scalar Function

Fungsi *scalar* adalah fungsi untuk menghasilkan nilai balik dari nilai input yang diberikan. Di dalam MySQL terdapat fungsi-fungsi *scalar* yang sering digunakan adalah:

1. UCASE()
2. LCASE()
3. MID()
4. LENGTH()
5. ROUND()
6. NOW()

7. CURDATE()
8. CURTIME()

Fungsi-fungsi *scalar* di atas sering dipakai dalam pengelolaan basis data. Di bawah ini akan dijelaskan penggunakan dan contoh kasus.

1. UCASE()

ucase() berfungsi untuk merubah tampilan data pada kolom menjadi huruf besar.

Contoh kasus 17: Mengubah data di kolom status menjadi huruf besar.

Query 17.A sebagai berikut:

```
SELECT UCASE(STATUS) AS STATUS
FROM pembayaran
```

Maka hasil query kasus 17.A adalah:

```
+----------+
| STATUS   |
+----------+
| CASH     |
| CASH     |
| TRANSFER |
| TRANSFER |
| TRANSFER |
| CASH     |
+----------+
6 rows in set (0.003 sec)
```

Query 17.B sebagai berikut:

```
SELECT KODE_PEMBAYARAN, UCASE(STATUS) AS STATUS
FROM pembayaran
```

Maka hasil query kasus 17.B adalah:

```
+-----------------+----------+
| KODE_PEMBAYARAN | STATUS   |
+-----------------+----------+
| B001            | CASH     |
| B002            | CASH     |
| B003            | TRANSFER |
| B004            | TRANSFER |
| B005            | TRANSFER |
| B006            | CASH     |
+-----------------+----------+
6 rows in set (0.002 sec)
```

2. LCASE()

lcase() berfungsi untuk merubah tampilan data pada kolom menjadi huruf kecil.

Contoh kasus 18: Mengubah data di kolom kode_pembayaran menjadi huruf kecil.

Query 18 sebagai berikut:

```
SELECT LCASE(KODE_PEMBAYARAN) AS KODE, STATUS
FROM pembayaran
```

Maka hasil query kasus 18 adalah:

```
+------+----------+
| KODE | STATUS   |
+------+----------+
| b001 | cash     |
| b002 | cash     |
| b003 | transfer |
| b004 | transfer |
| b005 | transfer |
| b006 | cash     |
+------+----------+
6 rows in set (0.008 sec)
```

3. MID()

mid() berfungsi untuk merubah tampilan data pada kolom dengan ketentuan digit tertentu (karakter ekstrak).

Contoh kasus 19: Menampilan data status pada digit 1 sampai digit 2 di tabel pembayaran.

Query 19 sebagai berikut:

```
SELECT MID(STATUS,1,2) AS STATUS
FROM pembayaran
```

Maka hasil query kasus 19 adalah:

```
+--------+
| STATUS |
+--------+
| ca     |
| ca     |
| tr     |
| tr     |
| tr     |
| ca     |
+--------+
6 rows in set (0.017 sec)
```

4. LENGTH()

length() berfungsi untuk menampilkan jumlah charakter data pada kolom.

Contoh kasus 20: Menampilan jumlah charakter data pada kolom status.

Query 20 sebagai berikut:

```
SELECT LENGTH (STATUS) AS STATUS
FROM pembayaran
```

Maka hasil query kasus 20 adalah:

```
+--------+
| STATUS |
+--------+
|      4 |
|      4 |
|      8 |
|      8 |
|      8 |
|      4 |
+--------+
6 rows in set (0.026 sec)
```

5. ROUND()

round() berfungsi untuk menampilkan data dengan ketentuan berapa digit dibelakang koma data pada kolom.

Tampilan dari Query:

```
SELECT AVG(JUMLAH PEMBAYARAN)
FROM pembayaran
```

```
+-------------------------+
| AVG(JUMLAH_PEMBAYARAN) |
+-------------------------+
|            566666.6667 |
+-------------------------+
1 row in set (0.014 sec)
```

Contoh kasus 21: Menampilan rata-rata nominal pembayaran dengan tampilan 0 digit dibelakang koma.

Query 21 sebagai berikut:

```
SELECT ROUND(AVG(JUMLAH_PEMBAYARAN),0)
FROM pembayaran
```

Maka hasil query kasus 21 adalah:

```
+-------------------------------+
| ROUND(AVG(JUMLAH_PEMBAYARAN),0) |
+-------------------------------+
|                        566667 |
+-------------------------------+
1 row in set (0.015 sec)
```

6. NOW()

now() berfungsi untuk menampilkan tanggal dan waktu sistem saat ini.

7. CURDATE()

curdate() berfungsi untuk menampilkan tanggal sistem saat ini.

8. CURTIME()

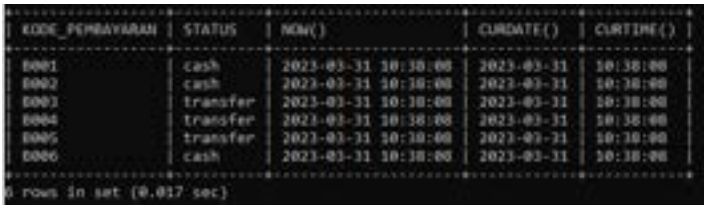
curtime() berfungsi untuk menampilkan waktu sistem saat ini.

Contoh kasus 22: Menampilan kode pembayaran, status, tanggal dan waktu, tanggal, serta waktu pembayaran pada tabel pembayaran.

Query 22 sebagai berikut:

```
SELECT KODE_PEMBAYARAN, STATUS, NOW(), CURDATE(), CURTIME()
FROM pembayaran
```

Maka hasil query kasus 22 adalah:



KODE_PEMBAYARAN	STATUS	NOW()	CURDATE()	CURTIME()
B001	cash	2023-03-31 10:38:00	2023-03-31	10:38:00
B002	cash	2023-03-31 10:38:00	2023-03-31	10:38:00
B003	transfer	2023-03-31 10:38:00	2023-03-31	10:38:00
B004	transfer	2023-03-31 10:38:00	2023-03-31	10:38:00
B005	transfer	2023-03-31 10:38:00	2023-03-31	10:38:00
B006	cash	2023-03-31 10:38:00	2023-03-31	10:38:00

6 rows in set (0.017 sec)

DAFTAR PUSTAKA

Sianipar, E. S., & Rahardjo, B. (2017). Basis Data. Informatika.

Purwanto, E., & Rokhman, F. A. (2017). Konsep Basis Data dan Implementasinya dengan MySQL. Andi.

Sutanto, E., & Sanjaya, R. (2016). Pemodelan dan Analisis Database Menggunakan Model Entity-Relationship pada Sistem Informasi Akademik. Jurnal Ilmiah Ilmu Komputer (JIKA), 10(2), 77-84.

Santoso, H. B., & Putra, D. A. (2015). Rancang Bangun Basis Data untuk Aplikasi Inventori pada Toko Sinar Jaya Komputer menggunakan MySQL. Jurnal RESTI (Rekayasa Sistem dan Teknologi Informasi), 1(2), 186-192.

Sari, D. P., & Indrayana, F. (2019). Penggunaan Database MySQL dalam Sistem Informasi Akademik pada Sekolah Dasar XYZ. Prosiding Seminar Nasional Informatika (SEMNASIF), 1-6.

Hidayat, A. A., & Yatim, A. (2018). Perancangan Database Menggunakan Model Relasional pada Sistem Informasi Penggajian Karyawan PT XYZ. Prosiding Seminar Nasional Teknik Elektro (SNTE), 1-5.

David B., Grimson J. (1992), Distributed Database System. First Edition. Mackays of Chatham PLC. , Great Britain 07458 ISBN 0-201-54400-8

A. J.Fabbri and A. R. Schwab, Practical Database Management,

Boston: PWS-KENT, 1992.

M Tamer O., Valduriez P. (1999), Princiles of Distributed Database

System. Third Edition. Prentice Hall, Upper Saddle River, New Jersey 07458

Adi Nugroho, ST., MMSI, (2004), Konsep Perancangan Sistem Basis Data, Andi Offset, Yogyakarta

Arbie. (2004). Manajemen Database dengan MySQL. Penerbit Andi: Yogyakarta

Kristanto, H. (2004). Konsep dan Perancangan Database. Penerbit Andi: Yogyakarta.

Kadir Abdul, (2008), Belajar database menggunakan MySQL, Andi Offset, Yogyakarta

Connoly, T., & Begg, C. (2010). Database System: A Practical Approach To Design, Implementation, and Management (Fifth). Boston: Pearson Education.

Roby, Yanto (2016). Manajemen Basis Data Menggunakan MySQL.

Yogyakarta: Deepublish.

Mita Agnitia Lestari, Muhamad Tabrani , Surtika Ayumida. (2018) Sistem Informasi Pengolahan Data Administrasi Kependudukan Pada Kantor Desa Pucung Karawang.

Bal Gupta, S., & Mittal, A. (2017). Introduction To Database Management System. Laxmi Publications (P) Ltd.

Connolly, T., & Begg, C. (2015). Database Systems—A Practical Approach to Design, Implementation, and Management Sixth Edition. Pearson.

en.wikipedia.com, en. wikipedia. com. (2023). IBM Information Management System. IBM Information Management System. https://en.wikipedia.org/wiki/IBM_Information_Management_System

geeksforgeeks.org, geeksforgeeks. org. (2022). History of DBMS. History of DBMS. https://www.geeksforgeeks.org/history-of-dbms/

Gills, A. S. (2022). IBM IMS (Information Management System). IBM IMS (Information Management System). https://www.techtarget.com/searchdatacenter/definition/IMS-Information-Management-System#:~:text=IBM%20IMS%20(Information%20Management%20System)%20is%20a%20database%20and%20transaction,introduced%20by%20IBM%20in%201968.

Khorshidi, R., & Hassani, A. (2013). Comparative analysis between TOPSIS and PSI methods of materials selection to achieve a desirable combination of strength and workability in Al/SiC composite. Materials and Design, 52(June), 999–1010. https://doi.org/10.1016/j.matdes.2013.06.011

mnactec.cat, mnactec. cat. (2022). Central Processing Unit (CPU) IBM System/360 model 30. Central Processing Unit (CPU) IBM System/360 Model 30. https://mnactec.cat/en/object-detail/14/computacio-electronica-i-telecomunicacions/central-processing-unit-cpu-ibm-system360-model-30

Papathanasiou, J., B, N. P., Bournaris, T., & Manos, B. (2016). A Decision Support System for Multiple Criteria Alternative Ranking Using TOPSIS and VIKOR : A Case Study on Social Sustainability in Agriculture. ICDSST, 2, 3–15. https://doi.org/10.1007/978-3-319-32877-5

sharktastica.co.uk, sharktastica. co. uk. (2022). IBM 2740 & 2741 Communications Terminal. IBM 2740 & 2741 Communications Terminal. sharktastica.co.uk

Yoon, K.P., & Hwang, C.-L. (1995). Multiple Attribute Decision Making: An Introduction. Sage University Paper Series on Quantative Applications in the Social Sciences, 47–53.

Connolly, T. M., & Begg, C. E. (2016). Database systems: A pragmatic approach. In M. Horton (Ed.), Database Systems: A Practical Approach to Design, Implementation, and Management (6th ed.). United States of America: Pearson.

Elmasri, R., & Navathe, S. B. (2010). Fundamentals Of Database System (6th ed.; M. Hirsch, ed.). Boston: Addison-Wesley.

Fikry, M. (2019). Buku Basis Data. In Eriyanto (Ed.), Unimal Press. Banda Aceh: Unimal Press.

Garcia-Molina, H., Ullman, J. D., & Widom, J. (2008). Database Systems: The Complete Book. In M. J. Horton (Ed.), Pearson (2nd ed.). United States of America: Pearson.

Gupta, S. B., & Mittal, A. (2017). An Introduction to Database Management Systems. In Library Hi Tech (2nd ed., Vol. 2). https://doi.org/10.1108/eb047557

Silberschatz, A., Korth, H. F., & Sudharshan, S. (2019). Database system concepts. In McGraw-Hill education (7th ed.). New York: McGraw Hill.

Connolly, Thomas and Begg, Carolyn. (2015). Database Systems: A Practical Approach to Design, Implementation, and Management. Sixth Edition. Pearson Education.

Achmad Solichin. (2016). Pemrograman Web dengan PHP dan MySQL. In Penerbit Budi Luhur.

Dewa Putu Yudhi Ardiana, D. (2022). Pemrograman Web. Indie Press.

Enterprise, J. (2017). Otodidak MySQL Untuk Pemula. PT ELEX MEDIA KOMPUTINDO.

Pratama, A. (2017). MySQL Uncover, Panduan Belajar MySQL dan MariaDB untuk Pemula. Bandung. Duniailkom.

Solichin, A. (2015). MySQL Dari Pemula Hingga Mahir. In Universitas Budi Luhur, Jakarta (Issue November). ACHMATIM.NET.

Teknik, F., & Diponegoro, U. (2020). Surat Persetujuan. 10000(April), 2020.

Connolly, T. M., & Begg, C. E. (2016). Database systems: A pragmatic approach. In M. Horton (Ed.), Database Systems: A Practical Approach to Design, Implementation, and Management (6th ed.). United States of America: Pearson.

Elmasri, R., & Navathe, S. B. (2010). Fundamentals Of Database System (6th ed.; M. Hirsch, ed.). Boston: Addison-Wesley.

Garcia-Molina, H., Ullman, J. D., & Widom, J. (2008). Database Systems: The Complete Book. In M. J. Horton (Ed.), Pearson (2nd ed.). United States of America: Pearson.

Gupta, S. B., & Mittal, A. (2017). An Introduction to Database Management Systems. In Library Hi Tech (2nd ed., Vol. 2). https://doi.org/10.1108/eb047557

Silberschatz, A., Korth, H. F., & Sudharshan, S. (2019). Database system concepts. In McGraw-Hill education (7th ed.). New York: McGraw Hill.

Connolly, T., & Begg, C. (2015). Database Systems: A Practical Approach to Design, Implementation, and Management (Six Editio). Pearson Education. https://doi.org/10.1007/978-1-4842-1191-5

Coronel, C., & Morris, S. (2017). Database Systems: Design, Implementation, and Management (12th ed.). Cengage Learning.

Elmasri, R., & Navathe, S. B. (2016). Fundamentals of Database System (Seventh). Pearson. https://doi.org/10.3139/9783446437845.011

Gupta, S. B., & Mittal, A. (2017). Introduction to Database Management Systems (Second Edi, Vol. 2, Issue 3). Laxmi Publication. https://doi.org/10.1108/eb047557

Hariyanto, B. (2004). Sistem Manajemen Basis Data: Pemodelan, Perancangan, dan Terapannya. Penerbit Informatika.

Harrington, J. L. (2016). Relational Database Design and Implementation. Morgan Kaufman.

Hoffer, J. A., Ramesh, V., & Topi, H. (2011). Modern Database Management (Tenth Edit). Pearson Education.

Nugroho, A. (2004). Konsep Pengembangan Sistem Basis Data. Penerbit Informatika.

Abdul Kadir (2013) Pengertian MySQL, Buku Pintar Programme Pemula PHP, Yogyakarta. Mediakom.

Firdaus Solihin (2009), Basis Data 2, Buku Ajar Teknik Informatika Universitas Trunojoyo.

Priyadi, Yudi (2014), Kolaborasi SQL & ERD Dalam Implementasi Database, Penerbit Andi Yogyakarta.

Raharjo, Budi (2011), membuat Database Menggunakan MySQl, Bandung Informatika.

Setiawan Dimas (2019), Belajar Penggunakan Function di MySQL, Kelas Programmer.

TENTANG PENULIS

J. Prayoga, lahir di Payakumbuh pada Tanggal 10 Juni 1990. Penulis menyelesaikan Pendidikan Strata 1 pada Tahun 2013 di Universitas Putra Indonesia " YPTK " Padang Program Studi Sistem Informasi, Magister (S2) bidang Ilmu Komputer pada Tahun 2016 juga di Universitas Putra Indonesia "YPTK" Padang Kosentrasi Sistem Informasi. Saat ini penulis aktif mengajar di Universitas Dharmawangsa Fakultas Teknik dan Ilmu Komputer sejak tahun 2016 sebagai dosen Tetap pada Program Studi D3 Sistem Informasi dan juga menjabat sebagai Ketua Program Sttudi D3 Sistem Informasi periode 2020 – 2024. Selain itu penulis juga Fokus melakukan penelitian di bidang Data Mining, Digital Marketing. Penulis juga aktif menjadi reviewer di berbagai Jurnal Nasional Akreditas (SINTA 5 – SINTA 6). Penulis juga anggota APTIKOM. Selain itu penulis juga merupakan seoarang Entrepreneur muda yang bergerak di bidang Wedding Organizer (WO)

Sinar Sinurat, lahir di Sirimbang Kecamatan Pangurusan Kabupaten Tapanuli Utara pada tanggal 17 Agustus 1967. Penulis menyelesaikan pendidikan Strata I pada tahun 1994 di Institut Sains dan Teknologi TD Pardede "ISTP" Medan, Magister (S2) bidang Ilmu Komputer pada tahun 2010 di Universitas Putra Indonesia ''YPTK'' Padang. Penulis aktif mengajar di STMIK Budi Darma sejak tahun 2005 (Sekarang Universitas Budi Darma) sebagai Dosen Tetap pada program studi Teknik Informatika.

Andy Rachman adalah dosen Jurusan Teknik Informatika – Institut Teknologi Adhi Tama Surabaya. Penulis sampai buku ini ditulis merupakan Ketua Peneliti dan Publikasi – ITATS. Lahir di Surabaya tanggal 22 Februari 1977. Saat in penulis aktif sebagai Reviewer dibeberapa Jurnal Nasional maupun Jurnal Internasional. Penulis juga merupakan Asesor Kompetensi BNSP bidang TIK. Penulis saat ini juga aktif diberbagai keanggotaan bidang ilmu, mulai dari ACM, IEEE, KODEPENA, bahkan saat ini penulis telah mendapatkan sertifikat kompetensi bidang Penulis Non Fiksi dari BNSP. Saat ini penulis telah memiliki 30 buku ber-ISBN dan mendapatkan penghargaan sebagai Dosen Penulis Buku Terproduktif 1 di Kampus ITATS Tahun 2022. Penulis juga memiliki 2 buah book

chapter Internasional. Bidang Penelitian pada Rekayasa Perangkat Lunak, Pembuatan Game dan Realitas Virtual, Interaksi Manusia dan Komputer, dan Sistem Operasi.

Irmawati Carolina, lahir di Jakarta pada tanggal 16 Juni 1975. Penulis menyelesaikan pendidikan Strata I pada tahun 1997 di Institut Sains dan Teknologi Nasional Jakarta, Magister (S2) bidang Ilmu Komputer pada tahun 2010 di STMIK Nusa Mandiri Jakarta, Program Profesi Insinyur tahun 2022 di Universitas Hasanudin Makasar. Penulis aktif serta mengajar di Universitas Bina Sarana Informatika sejak tahun 1998 sebagai Dosen Tetap pada program studi Sistem Informasi Akuntansi. Fokus penelitian yang dilakukan adalah bidang, Audit Sistem Informasi, Machine Learning, Tata Kelola TI. Selain itu penulis aktif pada beberapa organisasi.

Andi Irmayana, Lahir di Makassar tanggal 18 September 1985. Lulus sarjana tahun 2008 pada program studi Sistem Informasi STMIK Dipanegara Makassar. Menyelesaikan studi S2 pada jurusan Teknik Elektro konsentrasi Teknik Informastika pada tahun 2011 di Universitas Hasanuddin makassar. Menjadi dosen sejak tahun 2008 pada Universitas Dipa Makassar. Aktif dalam organisasi dosen icoris dan indoceiss. Aktif dalam kegiatan pengabdian masyarakat dan melakukan penelitian yang dipublikasi pada jurnal nasional dan internasional terakreditasi.

Adi Supriyatna, merupakan pria kelahiran Jakarta 17 Oktober 1985. Telah menyelesaikan pendidikan Strata Satu Program Studi Sistem Informasi pada tahun 2008 di Sekolah Tinggi Manajemen Informatika dan Komputer Nusa Mandiri Jakarta, kemudian menyelesaikan Program Magister (S2) program studi Ilmu Komputer pada tahun 2011 di Sekolah Tinggi Manajemen Informatika dan Komputer Nusa Mandiri Jakarta. Saat ini Penulis aktif mengajar di Universitas Bina Sarana Informatika sejak tahun 2007 sebagai Dosen Tetap pada Fakultas Teknik dan Informatika. Fokus penelitian yang dilakukan adalah bidang Sistem Informasi, Database, Data Warehouse, dan Data Mining. Penulis telah memiliki beberapa sertifikasi kompetensi dari BNSP antara lain Sertifikasi Skema Programmer, Sertifikasi Skema Database Administrator dan Sertifikasi Skema Associate Data Science serta sebagai Asesor Kompetensi pada LSP Universitas BSI. Penulis aktif menjadi sebagai penulis dan mitra bestari di berbagai Jurnal Nasional Akreditasi. Penulis juga

aktif sebagai pengurus dan anggota asosiasi profesi seperti Asosiasi Perguruan Tinggi Informatika dan Komputer (APTIKOM), Indonesian Computer, Electronics and Instrumentation Support Society (INDOCEISS).

Pasnur, lahir di Parepare pada tanggal 15 April 1980. Penulis menyelesaikan pendidikan Strata I (S1) di Jurusan Elektro Fakultas Teknik Universitas Hasanuddin Makassar pada tahun 2004. Penulis menyelesaikan pendidikan Magister (S2) di Jurusan Teknik Informatika Fakultas Teknologi Informasi Institut Teknologi Sepuluh Nopember Surabaya pada tahun 2015. Penulis aktif mengajar di Universitas Teknologi Akba Makassar (Unitama) sejak tahun 2009 sebagai Dosen Tetap pada program studi Teknik Informatika. Fokus penelitian yang dilakukan adalah bidang Natural Language Processing, Information Retrieval, Computer Vision, Image Processing, dan Data Mining. Penulis juga aktif menjadi reviewer pada beberapa Jurnal Nasional Terakreditasi (SINTA 2 – SINTA 4). Selain itu penulis merupakan Kepala UPT ICT Unitama dengan salah satu tugas adalah melakukan pengembangan sistem informasi manajemen terintegrasi.

Budanis Dwi Meilani, lahir di Banyuwangi 17 Mei 1972. Ketertarikan penulis pada bidang ilmu komputer terutama Informatika dimulai pada tahun 1998 silam. Penulis menyelesaikan Pendidikan S1 Teknik Informatika di Institut Teknologi Adhi Tama Surabaya (ITATS) dan berhasil lulus tahun 1998. Penulis kemudian melanjutkan Pendidikan S2 Teknik Informatika di Institut Teknologi 10 Nopember Surabaya (ITS) dan berhasil lulus tahun 2008.

Penulis memiliki kepakaran dibidang Algorithma & Pemrograman, Pemrograman Berorientasi Obyek, Struktur Data, Data Mining dan Sistem Pendukung Keputusan. Saat ini penulis berkarir sebagai dosen tetap di Institut Teknologi Adhi Tama Surabaya (ITATS) pada program studi Sistem Informasi. Untuk mewujudkan karir menjadi dosen maka penulis juga aktif dalam penelitian penelitian yang berhubungan dengan bidang kepakaran. Penelitian yang dilakukan penulis mendapatkan dana dari berbagai sumber diantaranya dana yang didapat dari mandiri, institusi swasta, institusi tempat bekerja dan Kemenristek DIKTI. Hasil dari penelitian sudah banyak yang penulis publikasikan berupa jurnal dan prosiding. Semoga kedepannya penulis bisa memberikan kontribusi poitif bagi bangsa dan negara.

Buku "Sistem Basis Data" adalah panduan komprehensif yang membawa pembaca ke dalam dunia yang menarik dan penting dari sistem basis data. Dalam buku ini, pembaca akan dibimbing melalui konsep dasar, prinsip, dan praktik yang terkait dengan manajemen basis data, yang menjadi tulang punggung dari sebagian besar aplikasi dan sistem informasi modern.

Buku ini dimulai dengan memperkenalkan konsep dasar seperti definisi basis data, perbedaan antara basis data relasional dan non-relasional, serta peran dan fungsi sistem basis data. Pembaca juga akan mempelajari tentang model data yang berbeda, seperti model hierarki, jaringan, dan relasional, serta bagaimana merancang basis data yang efisien dan sesuai dengan kebutuhan aplikasi. Selain itu, buku ini membahas bahasa query, yang memungkinkan pengguna untuk mengakses dan memanipulasi data dalam basis data. Pembaca akan mempelajari cara menggunakan bahasa query, seperti SQL (Structured Query Language), untuk melakukan pencarian, pembaruan, dan penghapusan data.

DITERBITKAN OLEH
CV. GRAHA MITRA EDUKASI
Jln Payanibung Ujung D
Dalu Sepuluh-B, Tanjung Morawa
Kab. Deli Serdang Sumatera Utara

ISBN 978-623-09-4150-4