Introduction to Database

Maulana Hafidz Ismail

09 Oktober 2025

Contents

1 Introduction to Databases

1.1 Pengertian Data

- Data adalah fakta dan angka tentang sesuatu, seperti nama, usia, dan informasi pembelian.
- Data sangat penting bagi individu dan organisasi untuk pengambilan keputusan dan analisis.

1.2 Pengertian Basis Data/Database

- Basis data atau database adalah penyimpanan elektronik yang mengorganisir data secara sistematis, membuatnya lebih mudah dikelola dan aman.
- Contoh penggunaan basis data termasuk bank yang menyimpan data pelanggan dan rumah sakit yang menyimpan data pasien.

1.3 Organisasi Data dalam Basis Data/Database

- Data dalam basis data diorganisir dalam bentuk entitas yang mirip dengan tabel, dengan atribut yang menjadi kolom dan setiap baris mewakili instansi dari entitas tersebut.
- Terdapat berbagai jenis basis data, termasuk basis data relasional, basis data berorientasi objek, basis data graf, dan basis data dokumen.

1.4 Hubungan Data dalam Basis Data

- Data dalam basis data tidak dapat berdiri sendiri; harus ada hubungan dengan data lain untuk menghasilkan informasi yang bermakna.
- Contoh hubungan data dapat dilihat antara tabel pelanggan dan tabel pesanan, di mana nomor pesanan dicocokkan dengan ID pelanggan.

1.5 Struktur Tabel Pelanggan

- Tabel pelanggan memiliki kolom seperti `Customer ID`, `FirstName`, `LastName`, dan `Email`, yang disebut sebagai `field`.
- Setiap baris dalam tabel mewakili satu entitas pelanggan, dan setiap entitas harus dapat diidentifikasi secara unik menggunakan `primary key`, yaitu `Customer ID`.

1.6 Struktur Tabel Pesanan

- Tabel pesanan juga memiliki `field` dan `record`, dengan `Order ID` sebagai `primary key` dan `Customer ID` sebagai `foreign key`.
- `Foreign key` menghubungkan tabel pesanan dengan tabel pelanggan, memungkinkan pengambilan data yang bermakna dari kedua tabel tersebut.

1.7 Pengumpulan dan Penyajian Data

- Data dikumpulkan dari berbagai sumber seperti pesanan pelanggan dan umpan balik pengguna untuk meningkatkan layanan.
- Penyajian data yang baik membantu orang memahami informasi dengan lebih baik, sering kali menggunakan grafik untuk visualisasi.

1.8 Jenis Grafik Umum

- Bar Chart: Menampilkan data kategorikal dengan batang, menunjukkan perubahan nilai dari waktu ke waktu, seperti penjualan buku selama beberapa tahun.
- Bubble Chart: Menggambarkan perbandingan nilai dengan ukuran gelembung, seperti populasi negara, di mana gelembung yang lebih besar menunjukkan populasi yang lebih besar.
- Line Chart: Menyajikan data sebagai titik yang dihubungkan dengan garis, sering digunakan untuk menunjukkan tren, seperti harga emas dari waktu ke waktu.
- Pie Chart: Menunjukkan proporsi data dalam bentuk irisan, seperti preferensi olahraga siswa dalam kelas.

1.9 Memilih Grafik yang Tepat

- Pemilihan grafik tergantung pada audiens, tujuan presentasi, dan jenis data yang ada.
- Penting untuk memilih grafik yang menarik dan sesuai untuk menyampaikan pesan yang diinginkan.

1.10 Jenis Basis Data

- Basis data relasional memiliki keterbatasan dalam menyimpan data tidak terstruktur, sehingga muncul basis data NoSQL yang menawarkan struktur fleksibel.
- Basis data NoSQL dapat menyimpan data dalam berbagai format, termasuk basis data dokumen, basis data nilai kunci, dan basis data graf.

1.11 Big Data

- `Big data` adalah data kompleks yang dapat meningkat volumenya seiring waktu, berasal dari platform media sosial, situs belanja online, dan perangkat IoT.
- `Big data` mencakup data terstruktur, semi-terstruktur, dan tidak terstruktur, memberikan wawasan unik untuk pengambilan keputusan yang lebih baik di berbagai industri.

1.12 Basis Data Cloud

- Organisasi beralih ke basis data cloud untuk menghindari kesulitan dalam mengelola server fisik, dengan contoh layanan penyimpanan cloud seperti Dropbox dan iCloud.
- Basis data cloud memungkinkan penyimpanan data yang lebih terjangkau dan efisien.

1.13 Sejarah Basis Data

- Dimulai pada tahun 1960-an dengan komputerisasi basis data, yang membuat penyimpanan data lebih efisien.

1.14 Perkembangan Basis Data

- 1970-an hingga 1990-an: `Flat files`, `hierarchical`, dan `network databases`.
- 1980-an hingga sekarang: `Relational databases`.
- 1990-an hingga sekarang: `Object-oriented`, `object-relational`, dan `NoSQL databases`.

1.15 Tipe-Tipe Basis Data Klasik

- Flat Files: Menyimpan data dalam satu file atau tabel, dengan setiap baris mewakili satu catatan.
- Hierarchical Databases: Mengatur data dalam struktur hierarkis, dengan hubungan satu-ke-banyak (`One-to-many`).
- Network Databases: Memungkinkan hubungan banyak-ke-banyak (`Many-to-many`) dengan struktur mirip graf.

1.16 Relational Databases

- Dikenalkan pada tahun 1980-an, menyimpan data dalam tabel dengan kolom sebagai atribut dan baris sebagai catatan.
- Menggunakan `primary key` dan `foreign key` untuk menghubungkan data antar tabel.

1.17 NoSQL Databases

- Muncul untuk menangani data tidak terstruktur dan memberikan kecepatan serta fleksibilitas lebih.
- Tipe-tipe NoSQL termasuk `document databases`, `key-value databases`, `wide-column databases`, dan `graph databases`.

1.18 Bacaan Bermanfaat

- `https://www.oracle.com/uk/database/what-is-database/`
- `https://www.javatpoint.com/types-of-databases`
- `https://www.ibm.com/cloud/learn/relational-databases`
- `https://www.tutorialspoint.com/Types-of-databases`
- `http://graphdatamodeling.com/GraphDataModeling/History.html`

1.19 Structured Query Language (SQL)

- SQL adalah bahasa standar yang digunakan untuk berinteraksi dengan semua jenis basis data, terutama basis data relasional.
- Contoh basis data relasional yang dapat berinteraksi dengan SQL termasuk `MySQL`, `PostgreSQL`, `Oracle`, dan `Microsoft SQL Server`.

1.20 CRUD Operations (Singkatan)

- Interaksi dengan basis data melibatkan operasi dasar yang dikenal sebagai `CRUD`: `Create`, `Read`, `Update`, dan `Delete`.
- Operasi ini penting untuk manipulasi data dalam basis data.

1.21 Database Management System (DBMS)

- `DBMS` bertanggung jawab untuk menginterpretasikan dan mengeksekusi instruksi SQL yang diberikan.
- Sebagai web developer, Anda akan menggunakan `DBMS` untuk menjalankan semua instruksi SQL pada basis data.

1.22 CRUD Operations (Penjelasan)

- `CRUD` adalah singkatan dari `Create`, `Read`, `Update`, dan `Delete`, yang merupakan operasi dasar dalam pengelolaan data di basis data.
- Operasi ini mencakup penambahan data, pembacaan data, pembaruan data yang sudah ada, dan penghapusan data.

1.23 Subbahasa SQL

- SQL dibagi menjadi beberapa subbahasa, termasuk `DDL` (`Data Definition Language`), `DML` (`Data Manipulation Language`), `DQL` (`Data Query Language`), dan `DCL` (`Data Control Language`).
- `DDL` digunakan untuk mendefinisikan struktur basis data, seperti membuat dan mengubah tabel, sedangkan `DML` digunakan untuk memanipulasi data, seperti menambah, memperbarui, dan menghapus data.

1.24 Perintah SQL (Berdasarkan Subbahasa)

- Perintah DDL mencakup `CREATE` untuk membuat objek basis data, `ALTER` untuk mengubah struktur objek, dan `DROP` untuk menghapus objek.
- DML menggunakan perintah `INSERT INTO` yang kemudian diikuti `VALUES` untuk menambahkan data/instansi, `UPDATE` untuk memperbarui data, dan `DELETE` untuk menghapus data.

1.25 Pengambilan Data dan Kontrol Akses

- DQL menggunakan perintah `SELECT` untuk mengambil data dari satu atau beberapa tabel dengan kriteria filter yang ditentukan.
- DCL mengontrol akses ke basis data dengan perintah `GRANT` dan `REVOKE` untuk memberikan atau mencabut hak akses pengguna.

1.26 Keuntungan Menggunakan SQL

- SQL memerlukan sedikit keterampilan pemrograman, hanya menggunakan sekumpulan kata kunci untuk melakukan operasi dasar seperti menambah, membuat, memperbarui, dan menghapus data.
- SQL bersifat interaktif, memungkinkan pengembang untuk menulis kueri kompleks dalam waktu singkat.

1.27 Karakteristik SQL

- SQL adalah bahasa standar yang dapat digunakan dengan semua basis data relasional, seperti MySQL, dan memiliki banyak dukungan serta informasi yang tersedia.
- SQL bersifat portabel, artinya kode yang ditulis dapat dijalankan di berbagai perangkat dan sistem operasi tanpa masalah.

1.28 Fungsi SQL dalam Manajemen Basis Data

- SQL mencakup semua aspek manajemen basis data, termasuk pembuatan basis data, manipulasi data, dan pengelolaan keamanan basis data melalui berbagai subset seperti DDL, DML, DQL, dan DCL.
- SQL memungkinkan pengguna untuk memproses data dalam jumlah besar dengan cepat dan efisien.

1.29 Pengenalan SQL dan Pembuatan Database

- SQL digunakan untuk berinteraksi dengan database, termasuk membuat database dan tabel.
- Perintah untuk membuat database adalah `CREATE DATABASE database_name`, diikuti dengan perintah untuk membuat tabel menggunakan `CREATE TABLE table_name`.

1.30 Pembuatan Tabel dan Penentuan Field

- Saat membuat tabel, kita dapat langsung menentukan `field` atau `atribut` yang akan disimpan, termasuk `tipe data` untuk setiap kolom.

```
CREATE TABLE student (
  ID INT,
  first_name VARCHAR(50),
  last_name VARCHAR(50),
  date_of_birth DATE
);
```

Listing 1: SQL: Membuat Entitas dengan Atributnya

1.31 Manipulasi Data dalam Tabel

- Untuk menambahkan data ke tabel, digunakan perintah `INSERT INTO table_name (column1, column2, ...)` dan `VALUES (value1, value2, ...)`. Perintah pertama menentukan atribut entitasnya dan perintah kedua membuat instansi.

- Untuk memperbarui data, digunakan perintah `UPDATE table_name SET column1 = value1 WHERE condition`.
- Untuk menghapus data, digunakan perintah `DELETE FROM table_name WHERE condition`.
- `condition` biasanya adalah `Primary Key` dari instansi yang ditunjuk, ini dimanfaatkan sebagai `identifier`-nya.

1.32 Pembacaan Data dari Tabel

- Untuk membaca data, digunakan perintah `SELECT`, diikuti dengan kolom yang ingin diambil dan nama tabel.
- Contoh: `SELECT first_name, last_name FROM student WHERE ID = 1;` untuk mendapatkan nama mahasiswa dengan ID tertentu.

1.33 Kategorisasi Perintah SQL dan Contoh Syntax

1.33.1 Data Definition Language (DDL)

CREATE Command

- Tujuan: Membuat database atau tabel.

```
CREATE TABLE table_name (
  column_name1 datatype(size),
  column_name2 datatype(size),
  column_name3 datatype(size)
);
```

Listing 2: Sintaks CREATE Command

DROP Command

- Tujuan: Menghapus database atau tabel.

```
DROP TABLE table_name;
```

Listing 3: Sintaks DROP Command

ALTER Command

- Tujuan: Mengubah struktur tabel.

```
ALTER TABLE table_name ADD (column_name datatype(size));
```

Listing 4: Sintaks ALTER Command untuk Menambah Kolom

```
ALTER TABLE table_name ADD primary key (column_name);
```

Listing 5: Sintaks ALTER Command untuk Menambah Primary Key

TRUNCATE Command

- Tujuan: Menghapus semua catatan dari tabel tanpa menghapus tabel itu sendiri.

```
TRUNCATE TABLE table_name;
```

Listing 6: Sintaks TRUNCATE Command

COMMENT Command

- Tujuan: Menambahkan komentar untuk menjelaskan pernyataan SQL.

```
-- Ini adalah komentar
```

Listing 7: Sintaks COMMENT Command

1.33.2 Data Manipulation Language (DML)

INSERT Command

- Tujuan: Menambahkan data ke tabel yang ada.

```
INSERT INTO table_name (column1, column2, column3) VALUES (
    value1, value2, value3);
```

Listing 8: Sintaks INSERT Command

UPDATE Command

- Tujuan: Memperbarui data dalam tabel.

```
UPDATE table_name SET column1 = value1, column2 = value2
    WHERE condition;
```

Listing 9: Sintaks UPDATE Command

DELETE Command

- Tujuan: Menghapus data dari tabel.

```
DELETE FROM table_name WHERE condition;
```

Listing 10: Sintaks DELETE Command

1.33.3 Data Control Language (DCL)

GRANT Command

- Tujuan: Memberikan hak akses kepada pengguna.

```
GRANT privilege ON object TO user;
```

Listing 11: Sintaks GRANT Command

REVOKE Command

- Tujuan: Menghapus hak akses dari pengguna.

```
REVOKE privilege ON object FROM user;
```

Listing 12: Sintaks REVOKE Command

1.33.4 Transaction Control Language (TCL)

COMMIT Command

- Tujuan: Menyimpan semua perubahan yang telah dilakukan.

```
COMMIT;
```

Listing 13: Sintaks COMMIT Command

ROLLBACK Command

- Tujuan: Mengembalikan database ke keadaan terakhir yang disimpan.

```
ROLLBACK;
```

Listing 14: Sintaks ROLLBACK Command

1.34 Pengenalan Tabel dalam Basis Data

- Tabel terdiri dari baris dan kolom yang menyimpan data.
- Setiap tabel mewakili entitas dan berfungsi sebagai relasi dalam basis data.

1.35 Struktur Tabel

- Tabel adalah objek dasar dalam basis data relasional yang menyimpan data dalam bentuk baris dan kolom.
- Kolom (atau atribut) punya tampak vertikal, dan memiliki nama unik dan tipe data yang menentukan jenis nilai yang dapat disimpan.
- Setiap kolom memiliki nama yang mendeskripsikan data yang disimpan, seperti `FirstName`, `LastName`, dan `ProductID`.
- Baris (atau record) punya tampak horizontal, yaitu kombinasi dari kolom yang menyimpan data untuk instansi tertentu. Bisa dipahami sebagai satu instansi dari entitas tersebut.

1.36 Tipe Data dan Domain

- Setiap kolom dalam tabel memiliki tipe data yang ditentukan oleh SQL, yang mendefinisikan jenis nilai yang dapat disimpan.
- Tipe data kolom mencakup `integer`, `karakter`, `tanggal`, dan `waktu`, yang bervariasi tergantung pada sistem basis data.
- Contoh tipe data termasuk:
  - Tipe data numerik/integer (`INT`, `TINYINT`, `BIGINT`, `FLOAT`, `REAL`)
  - Tipe data tanggal dan waktu (`DATE`, `TIME`, `DATETIME`)
  - Tipe data karakter (`CHAR`, `VARCHAR`)
  - Tipe data biner (`BINARY`, `VARBINARY`)
  - Tipe data lain (`CLOB`, `BLOB`)
- Domain adalah set nilai yang sah untuk atribut, memastikan bahwa nilai yang dimasukkan sesuai dengan tipe data yang ditentukan.

1.37 Primary Key & Foreign Key (Struktur Tabel)

- Setiap record dalam tabel diidentifikasi secara unik oleh `Primary Key`, yang merupakan kolom dengan nilai unik. Unik disini berarti tidak ada yang sama.
- `Primary Key` dapat berupa satu kolom atau kombinasi beberapa kolom jika tidak ada kolom tunggal yang memiliki nilai unik.
- `Foreign Key` adalah kolom yang menghubungkan tabel satu dengan yang lain, memungkinkan hubungan antar tabel.

1.38 Batasan Integritas (Integrity Constraint)

- Tabel harus mematuhi aturan atau batasan yang dikenal sebagai batasan integritas (`Constraint`), yang meliputi:
  - Batasan kunci: Memastikan tabel memiliki kolom berisi nilai unik yang dapat digunakan untuk mengambil data (`Primary Key`).
  - Batasan domain: Aturan untuk nilai yang dapat disimpan dalam kolom tertentu.
  - Batasan integritas referensial: Memastikan bahwa nilai kolom yang dirujuk ada dalam tabel lain.

1.39 Struktur Logis dan Fisik

- Struktur Logis (`Logical Database Structure`): Diwakili oleh diagram yang dikenal sebagai `Entity Relationship Diagram (ERD)`, yang menunjukkan bagaimana tabel akan diimplementasikan secara visual. Hanya terdapat nama entitas, atribut, dan relasi dari setiap objek.
- Struktur Fisik (`Physical Database Structure`): Di mana entitas diimplementasikan sebagai tabel, dengan hubungan yang ditetapkan menggunakan `foreign key`. Disini tabel sudah berisi `records`.

1.40 Jenis Hubungan

- Hubungan antar entitas dapat berupa:
  - One-to-One
  - One-to-Many
  - Many-to-Many

2 Create, Read, Update, and Delete (CRUD) Operations

2.1 Tipe Data dalam Basis Data

- Tipe data menentukan jenis data yang dapat diterima oleh setiap kolom dalam tabel.
- Tipe data yang umum digunakan adalah `numerik`, `string`, dan `tanggal/waktu`.

2.2 Tipe Data Numerik

- Tipe data numerik memungkinkan kolom menyimpan data dalam bentuk angka.
- Dua tipe data numerik yang paling umum adalah `INTEGER` (bilangan bulat) dan `DECIMAL` (bilangan desimal).
- Tipe data `DECIMAL` bisa ditentukan presisinya yaitu dengan diberi argumen seperti ini `price DECIMAL(10,2)`. Maka dapat menyimpan data dengan jumlah 10 digit angka dan 2 darinya dibelakang koma seperti 123,53 atau 12345678,92.
- Tipe data `DECIMAL` juga diketahui sebagai tipe data `NUMERIC`.

2.3 Contoh Penggunaan Tipe Data Numerik

- Kolom kuantitas produk didefinisikan sebagai tipe data `integer`, hanya menerima angka bulat.
- Kolom total harga didefinisikan sebagai tipe data `decimal`, dapat menyimpan angka dengan nilai pecahan.

2.4 Variasi Tipe Data Numerik

- Dalam sistem manajemen basis data seperti MySQL, terdapat berbagai jenis integer dan decimal dengan batasan nilai minimum dan maksimum.
- Misalnya, `TINYINT` untuk bilangan bulat kecil (maksimum 255) dan `INT` untuk bilangan bulat besar (maksimum lebih dari empat miliar).

2.5 Tipe Data String dalam Basis Data

- Tipe data string digunakan untuk menyimpan data yang terdiri dari karakter, termasuk huruf, angka, dan simbol.
- Penting untuk memilih tipe data yang tepat untuk menjaga integritas data dalam tabel.

2.6 CHAR vs. VARCHAR

- `CHAR`:
  - Memiliki panjang tetap yang ditentukan saat kolom didefinisikan.
  - Contoh: `CHAR(50)` akan selalu menggunakan 50 karakter di dalam memory, meskipun hanya ada sedikit karakter yang dimasukkan.
- `VARCHAR`:
  - Memiliki panjang variabel, hanya menggunakan ruang yang diperlukan untuk menyimpan data.
  - Contoh: `VARCHAR(50)` akan menyimpan hingga 50 karakter, tetapi hanya menggunakan ruang memori sesuai dengan jumlah karakter yang dimasukkan.

2.7 Contoh Penggunaan Tipe Data String

- Dalam tabel mahasiswa, kolom nama mahasiswa dapat didefinisikan sebagai `VARCHAR` untuk mengakomodasi panjang nama yang bervariasi.
- Kolom `username` dapat didefinisikan sebagai `CHAR` jika panjang `username` selalu sama.

2.8 Tipe Data String Lainnya

- `TINYTEXT`, `TEXT`, `MEDIUMTEXT`, dan `LONGTEXT` adalah tipe data string lainnya yang digunakan untuk menyimpan teks dengan panjang yang berbeda, dari kurang dari 255 karakter hingga 4 gigabyte.

2.9 Pengertian Constraints dalam Basis Data

- Constraints digunakan untuk membatasi jenis data yang dapat disimpan dalam tabel, memastikan bahwa data yang dimasukkan akurat dan dapat diandalkan.
- Jika ada pelanggaran antara constraint dan operasi data, operasi tersebut akan dibatalkan.

2.10 NOT NULL Constraint

- Digunakan untuk memastikan bahwa kolom tidak dapat memiliki nilai kosong (`null`).
- Contoh: Dalam tabel pelanggan, kolom ID pelanggan dan nama pelanggan harus selalu diisi. Jika tidak ada data yang dimasukkan, pembuatan catatan baru akan dibatalkan.

2.11 DEFAULT Constraint

- Menetapkan nilai `default` untuk kolom jika tidak ada nilai yang dimasukkan.
- Contoh: Dalam tabel pemain sepak bola, kolom kota dapat memiliki nilai `default` "Barcelona". Jika tidak ada nilai yang dimasukkan untuk kolom kota, maka secara otomatis akan diisi dengan "Barcelona".

2.12 CHECK Constraint

- `CHECK` constraint memungkinkan kita untuk menetapkan kondisi yang harus dipenuhi oleh nilai yang dimasukkan ke dalam kolom tersebut.
- Misalnya, jika kita ingin kolom `status` hanya dapat memiliki salah satu dari tiga nilai: "Dikirim", "Selesai", atau "Dibatalkan", kita bisa mendefinisikan `CHECK` constraint seperti ini:

```
CREATE TABLE cart_order (
order_id INT PRIMARY KEY,
customer_id INT,
product_id INT,
quantity INT,
order_date DATE,
status VARCHAR(100) CHECK (status IN ('Dikirim', 'Selesai',
    'Dibatalkan'))
);
```

Listing 15: Contoh Sintaks CHECK Constraint

- Dalam contoh di atas, `CHECK` constraint memastikan bahwa setiap nilai yang dimasukkan ke dalam kolom `status` harus salah satu dari nilai yang ditentukan. Jika ada nilai lain yang dimasukkan, maka database akan menolak perintah tersebut dan memberikan pesan kesalahan.

2.13 Implementasi NOT NULL & DEFAULT dalam SQL

- `NOT NULL` dan `DEFAULT` dapat didefinisikan saat membuat tabel menggunakan pernyataan SQL.

```
CREATE TABLE Player (name VARCHAR (50) NOT NULL, city
    VARCHAR (30) DEFAULT "Barcelona");
```

Listing 16: Contoh Sintaks NOT NULL dan DEFAULT

2.14 Perintah CREATE untuk membuat Database

- Perintah `CREATE` digunakan untuk membuat database baru.

```
CREATE DATABASE nama_database;
```

Listing 17: Sintaks CREATE DATABASE

2.15 Perintah DROP untuk menghapus Database

- Perintah `DROP` digunakan untuk menghapus database yang sudah ada.

```
DROP DATABASE nama_database;
```

Listing 18: Sintaks DROP DATABASE

2.16 Poin Penting pada CREATE dan DROP

- Pastikan untuk menggunakan perintah `DROP` dengan hati-hati, karena menghapus database akan menghapus semua tabel dan data di dalamnya secara permanen.
- Sebelum membuat database, penting untuk memiliki pemahaman yang jelas tentang tujuan dan struktur data yang akan disimpan.

2.17 Sintaks CREATE TABLE SQL

- Untuk membuat tabel, gunakan perintah `CREATE TABLE` diikuti dengan nama tabel.
- Setelah nama tabel, tambahkan tanda kurung untuk mendefinisikan kolom dan tipe data yang diperlukan.

```
CREATE TABLE customers (name VARCHAR (75), address VARCHAR
    (75), phone VARCHAR (15));
```

Listing 19: Sintaks CREATE TABLE

2.18 Pengenalan Perintah ALTER TABLE

- Perintah `ALTER TABLE` digunakan untuk mengubah struktur tabel yang sudah ada dalam basis data.
- Sintaks dasar dimulai dengan `ALTER TABLE` diikuti dengan nama tabel yang ingin diubah.

2.19 Menambahkan Kolom Baru pada Tabel

- Untuk menambahkan kolom baru, gunakan kata kunci `ADD` diikuti dengan nama kolom dan tipe data.
- Contoh: `ALTER TABLE students ADD age INT;` untuk menambahkan kolom "age" dengan tipe data `integer`.

2.20 Menghapus Kolom pada Tabel

- Untuk menghapus kolom, gunakan kata kunci `DROP COLUMN` diikuti dengan nama kolom yang ingin dihapus.
- Contoh: `ALTER TABLE students DROP COLUMN nationality;` untuk menghapus kolom "nationality".

2.21 Mengubah Atribut Kolom

- Untuk mengubah atribut kolom, gunakan kata kunci `MODIFY` diikuti dengan nama kolom dan tipe data baru.
- Contoh: `ALTER TABLE students MODIFY country VARCHAR(100);` untuk mengubah batas karakter kolom "country" menjadi 100.

2.22 Pengenalan Sintaksis INSERT INTO

- Untuk menulis pernyataan `INSERT`, mulai dengan klausa `INSERT INTO`, diikuti dengan nama tabel dan daftar kolom dalam tanda kurung yang dipisahkan oleh koma.
- Gunakan kata kunci `VALUES` dan tuliskan daftar nilai dalam tanda kurung. Setiap nilai harus sesuai dengan kolom yang dituju dan memiliki tipe data yang sama.

```
INSERT INTO customers (name, address, phone)
VALUES ("Maulana", "Gisting", "08123875123"),
       ("Hafizh", "Batam", "08127364123");
```

Listing 20: Perintah INSERT INTO dengan Satu dan Banyak Baris

2.23 Menambahkan Beberapa Baris Data

- Anda dapat menambahkan beberapa baris data sekaligus dengan menuliskan klausa `INSERT INTO` dan nama tabel, diikuti dengan kata kunci `VALUES` dan beberapa pasangan tanda kurung yang dipisahkan oleh koma untuk setiap baris data.

2.24 Penggunaan Pernyataan SQL SELECT

- Pernyataan dasar SQL `SELECT` ditulis dengan kata kunci `SELECT`, diikuti dengan nama kolom yang ingin diambil, kata kunci `FROM`, dan nama tabel. Contoh: `SELECT name FROM players;`
- Pernyataan ini mengembalikan hasil yang menampilkan semua nama pemain dari tabel yang ditentukan.

2.25 Mengambil Data dari Beberapa Kolom

- Untuk mengambil data dari beberapa kolom, gunakan koma untuk memisahkan nama kolom. Contoh: `SELECT name, level FROM players;`
- Hasilnya akan menampilkan data dari kolom nama dan level dalam tabel pemain.

2.26 Mengambil Semua Data dari Tabel

- Ada dua cara untuk mengambil semua data dari semua kolom dalam tabel:
  1. Menyebutkan semua nama kolom: `SELECT ID, name, age, level FROM players;`
  2. Menggunakan tanda asterisk sebagai singkatan: `SELECT * FROM players;`
- Keduanya akan mengembalikan semua informasi yang tersedia dalam tabel pemain.

2.27 Pengantar Pernyataan INSERT INTO SELECT

- Pernyataan `INSERT INTO SELECT` digunakan untuk mengambil data dari kolom dalam tabel sumber dan menempatkan hasilnya ke dalam kolom dalam tabel target.
- Contoh penggunaan: Mengambil data dari kolom C di tabel sumber dan menempatkannya di kolom B di tabel target.

2.28 Sintaks Dasar INSERT INTO SELECT

- Mulailah dengan pernyataan `INSERT INTO`, diikuti dengan nama tabel target dan nama kolom yang ingin diisi.
- Gunakan kata kunci `SELECT` untuk menyebutkan kolom yang ingin diambil dari tabel sumber, diikuti dengan kata kunci `FROM` dan nama tabel sumber.

```
INSERT INTO tableName1 (column1, column2)
SELECT columnToBeCopied FROM tableName2
```

Listing 21: Sintaks INSERT INTO SELECT

2.29 Perintah UPDATE

Sintaks untuk Memperbarui Satu Record

- Gunakan perintah `UPDATE` diikuti dengan nama tabel, kemudian `SET` untuk menentukan kolom yang akan diperbarui, dan akhiri dengan `WHERE` untuk menentukan record yang dimaksud.

```
UPDATE student_table SET home_address = 'alamat_baru',
    contact_number = 'nomor_baru' WHERE ID = 3;
```

Listing 22: Contoh UPDATE Satu Record

Sintaks untuk Memperbarui Banyak Record

- Untuk memperbarui informasi untuk beberapa siswa sekaligus, gunakan sintaks yang mirip tetapi sesuaikan `WHERE` untuk mencakup kondisi yang lebih luas.

```
UPDATE student_table
SET college_address = 'Harper␣Building'
WHERE department = 'Engineering';
```

Listing 23: Contoh UPDATE Banyak Record

Memperbarui Beberapa Kolom

- Anda juga dapat memperbarui beberapa kolom dalam satu pernyataan `UPDATE`.

```
UPDATE student_table
SET home_address = 'alamat_baru', college_address = 'Harper␣
    Building'
WHERE department = 'Engineering';
```

Listing 24: Contoh UPDATE Beberapa Kolom

2.30 Menghapus Record dari Tabel (DELETE)

- Menghapus Satu Catatan: Gunakan pernyataan `DELETE FROM` diikuti dengan nama tabel dan klausa `WHERE` untuk menentukan catatan yang akan dihapus. Contoh: `DELETE FROM student WHERE last_name = 'Miller';`.
- Menghapus Beberapa Catatan: Gunakan pernyataan yang sama tetapi dengan kondisi yang lebih luas. Contoh: `DELETE FROM student WHERE department = 'Engineering';`.
- Menghapus Semua Catatan: Gunakan pernyataan `DELETE FROM` tanpa klausa `WHERE`. Contoh: `DELETE FROM student;`.
- Penghapusan yang lebih efisien adalah dengan perintah `TRUNCATE table_name`. Ini akan menghapus seluruh record dari table dan tetap menyisakan tablenya, namun ini tidak dapat dibatalkan dan `inkremental` akan memulai dari awal.

2.31 Melihat Database Aktif

- Untuk melihat database yang aktif atau yang sedang digunakan bisa menggunakan perintah `SELECT DATABASE()`.

3 SQL Operators, Sorting, and Filtering Data

3.1 Pengantar Operator Aritmatika dalam SQL

- Operator adalah kata atau karakter yang digunakan untuk melakukan berbagai aktivitas dalam basis data, mirip dengan kata sambung dalam kalimat.
- Penting untuk memahami operator karena mereka memungkinkan manipulasi data untuk berbagai tujuan, seperti menghitung hari cuti karyawan atau membandingkan pencapaian karyawan.

3.2 Jenis Operator Aritmatika

- Operator penjumlahan (`+`), pengurangan (`-`), perkalian (`*`), pembagian (`/`), dan modulus (`%`) digunakan untuk melakukan operasi matematika dalam SQL.

3.3 Contoh Penggunaan Operator Aritmatika Sederhana

- Penjumlahan: `SELECT 10 + 15;` menghasilkan 25.
- Pengurangan: `SELECT 5 - 5;` menghasilkan 0.
- Perkalian: `SELECT 5 * 5;` menghasilkan 25.
- Pembagian: `SELECT 5 / 5;` menghasilkan 1.
- Modulus: `SELECT 100 % 10;` menghasilkan 0.

3.4 Penggunaan Operator dalam SQL

- Untuk melakukan operasi, gunakan perintah `SELECT` diikuti oleh operand dan operator. Ini berperan seperti kata kunci `return` dalam programming.
- Contoh sintaks: `SELECT operand1 operator operand2;` di mana `operand` adalah angka yang ingin dihitung.

3.5 Operator Aritmatika dalam Klausa SELECT dan WHERE

- Operator aritmatika digunakan untuk melakukan operasi matematika pada data numerik dalam tabel saat menulis kueri SQL `SELECT`.
- Operator ini dapat digunakan dalam klausa `SELECT` dan `WHERE`.

3.5.1 Penjumlahan (+)

- Dalam `SELECT`: Menambahkan nilai dari dua kolom.

```
SELECT salary + allowance FROM employee;
```

Listing 25: Penjumlahan dalam SELECT

- Dalam `WHERE`: Mengambil data karyawan dengan total gaji tertentu.

```
SELECT * FROM employee WHERE salary + allowance = 25000;
```

Listing 26: Penjumlahan dalam WHERE

3.5.2 Pengurangan (-)

- Dalam `SELECT`: Mengurangi nilai satu kolom dari kolom lainnya.

```
SELECT salary - tax FROM employee;
```

Listing 27: Pengurangan dalam SELECT

- Dalam `WHERE`: Mengambil data karyawan yang gajinya setelah pajak mencapai jumlah tertentu.

```
SELECT * FROM employee WHERE salary - tax = 50000;
```

Listing 28: Pengurangan dalam WHERE

3.5.3 Perkalian (*)

- Dalam `SELECT`: Mengalikan nilai dari dua kolom.

```
SELECT tax * 2 FROM employee;
```

Listing 29: Perkalian dalam SELECT

- Dalam `WHERE`: Mengambil data karyawan yang pajaknya setelah digandakan mencapai jumlah tertentu.

```
SELECT * FROM employee WHERE tax * 2 = 4000;
```

Listing 30: Perkalian dalam WHERE

3.5.4 Pembagian (/)

- Dalam `SELECT`: Membagi nilai satu kolom dengan kolom lainnya.

```
SELECT allowance / salary * 100 FROM employee;
```

Listing 31: Pembagian dalam SELECT

- Dalam `WHERE`: Mengambil data karyawan yang mendapatkan `allowance` minimal persentase tertentu.

```
SELECT * FROM employee WHERE allowance / salary * 100 >= 5;
```

Listing 32: Pembagian dalam WHERE

3.5.5 Modulus (%)

- Dalam `SELECT`: Mendapatkan sisa dari pembagian.

```
SELECT hours % 2 FROM employee;
```

Listing 33: Modulus dalam SELECT

- Dalam `WHERE`: Mengambil data karyawan yang bekerja dalam jumlah jam genap.

```
SELECT * FROM employee WHERE hours % 2 = 0;
```

Listing 34: Modulus dalam WHERE

3.6 Pengantar Operator Perbandingan

- Operator perbandingan digunakan untuk membandingkan dua nilai atau ekspresi, dengan hasil yang bisa benar atau salah.
- Operator ini membantu dalam memfilter data untuk menyertakan atau mengecualikan data tertentu.

3.7 Jenis Operator Perbandingan

- Sama dengan (`=`): Digunakan untuk menemukan data yang sama.
- Kurang dari: (`<`) Digunakan untuk menemukan data yang kurang dari nilai tertentu.
- Kurang dari atau sama dengan: (`<=`) Digunakan untuk menemukan data yang kurang dari atau sama dengan nilai tertentu.

- Lebih dari (`>`): Digunakan untuk menemukan data yang lebih dari nilai tertentu.
- Lebih dari atau sama dengan (`>=`): Digunakan untuk menemukan data yang lebih dari atau sama dengan nilai tertentu.
- Tidak sama dengan: (`<>` atau `!=`) Digunakan untuk menemukan data yang tidak sama dengan nilai tertentu.

3.8 Contoh Penggunaan Operator Perbandingan

- Operator Kesetaraan: Mengambil data karyawan dengan ID tertentu menggunakan `SELECT * FROM employee WHERE employee_id = 1;`.
- Operator Ketidaksamaan: Mengambil data karyawan dengan gaji yang tidak sama dengan 24000 menggunakan `SELECT * FROM employee WHERE salary <> 24000;`.
- Operator Lebih Besar Dari: Mengambil data karyawan dengan gaji lebih dari 50000 menggunakan `SELECT * FROM employee WHERE salary > 50000;`.
- Operator Lebih Besar Dari atau Sama Dengan: Mengambil data karyawan dengan pajak 1000 atau lebih menggunakan `SELECT * FROM employee WHERE tax >= 1000;`.
- Operator Kurang Dari: Mengambil data karyawan dengan tunjangan kurang dari 2500 menggunakan `SELECT * FROM employee WHERE allowance < 2500;`.
- Operator Kurang Dari atau Sama Dengan: Mengambil data karyawan yang bekerja kurang dari atau sama dengan 10 jam menggunakan `SELECT * FROM employee WHERE hours <= 10;`.

3.9 Klausa ORDER BY dalam SQL

- Tujuan: Klausa `ORDER BY` digunakan untuk mengurutkan data dalam tabel berdasarkan satu atau lebih kolom.
- Data dapat diurutkan dalam urutan menaik (`ASC`) atau menurun (`DESC`).
- Data yang terurut membantu dalam pengambilan keputusan bisnis yang lebih akurat dan efisien.

3.10 Sintaks Dasar ORDER BY

```sql
SELECT column1, column2, ...
FROM table_name
ORDER BY column_name [ASC|DESC];
```

Listing 35: Sintaks Dasar ORDER BY

- `ASC` (default) untuk urutan menaik, `DESC` untuk urutan menurun.
- Klausa `ORDER BY` digunakan untuk mengurutkan hasil dari kueri SQL `SELECT`.

3.11 Contoh Penggunaan ORDER BY

- Mengurutkan Berdasarkan Satu Kolom: Mengurutkan data berdasarkan kolom `nationality`:

```sql
SELECT ID, first_name, last_name, nationality
FROM student_table
ORDER BY nationality ASC;
```

Listing 36: ORDER BY Satu Kolom

- Mengurutkan Berdasarkan Beberapa Kolom: Mengurutkan data berdasarkan `nationality` dan `date_of_birth`:

```sql
SELECT ID, first_name, last_name, date_of_birth, nationality
FROM student_table
ORDER BY nationality ASC, date_of_birth DESC;
```

Listing 37: ORDER BY Beberapa Kolom

3.12 Catatan Penting ORDER BY

- Jika kolom memiliki tipe data numerik, data akan diurutkan dalam urutan numerik.
- Jika kolom memiliki tipe data teks, data akan diurutkan dalam urutan alfabetis.
- Anda dapat menggunakan tanda bintang (*) untuk memilih semua kolom:

```sql
SELECT *
FROM student_table
ORDER BY nationality ASC;
```

Listing 38: ORDER BY dengan Tanda Bintang

3.13 Pengertian WHERE Clause

- Fungsi: `WHERE clause` digunakan untuk memfilter data dalam query SQL, sehingga hanya catatan yang memenuhi kondisi tertentu yang akan diambil.

3.14 Sintaks Dasar WHERE Clause

```
SELECT column1, column2 FROM table_name WHERE condition;
```

Listing 39: Sintaks Dasar WHERE

3.15 Operator dalam WHERE Clause

Operator Perbandingan Tambahan

- `Greater than` (`!>`): Memeriksa apakah nilai di sebelah kiri tidak lebih besar.
- `Less than` (`!<`): Memeriksa apakah nilai di sebelah kiri tidak lebih kecil.

Operator Lain

- `BETWEEN`: Memfilter catatan dalam rentang nilai tertentu.

```
SELECT * FROM student_table WHERE DOB BETWEEN '2010-01-01'
    AND '2010-05-30';
```

Listing 40: Contoh Operator BETWEEN

- `LIKE`: Menentukan pola dalam kriteria filter. Misalnya, untuk mencari nama yang dimulai dengan "Sc":

```
SELECT * FROM student_table WHERE faculty LIKE 'Sc%';
```

Listing 41: Contoh Operator LIKE

- `IN`: Menentukan beberapa nilai yang mungkin untuk kolom. Misalnya, untuk mencari mahasiswa dari USA atau UK:

```
SELECT * FROM student_table WHERE country IN ('USA', 'UK');
```

Listing 42: Contoh Operator IN

- `ALL`: Digunakan untuk membandingkan satu nilai dengan semua nilai dalam himpunan nilai lainnya. Mengembalikan `true` hanya jika semua nilai dalam subkueri memenuhi kondisi.

```sql
SELECT EmployeeName
FROM employees
WHERE AnnualSalary > ALL (SELECT AnnualSalary FROM employees
     WHERE Department = 'Finance');

-- Bonus: Dapat membandingkan dua tabel tanpa harus
    menggunakan JOIN
SELECT EmployeeID
FROM employee_orders
WHERE HandlingCost > ALL (SELECT OrderTotal * 0.2 FROM
    orders WHERE orders.OrderID = employee_orders.OrderID);
```

Listing 43: Contoh Operator ALL

- `AND`: Memungkinkan adanya beberapa kondisi dalam klausa `WHERE` dari pernyataan SQL.
- `ANY`: Digunakan untuk membandingkan suatu nilai dengan nilai yang berlaku dalam daftar sesuai dengan kondisi. Mengembalikan `true` jika setidaknya satu nilai dalam subkueri memenuhi kondisi. Di beberapa kasus, `ANY` memberikan hasil yang sama seperti `JOIN`.

```sql
SELECT EmployeeName
FROM employees
WHERE AnnualSalary > ANY (SELECT AnnualSalary FROM employees
     WHERE Department = 'Marketing');

-- Memberikan hasil yang sama seperti JOIN
SELECT EmployeeName
FROM employees
WHERE EmployeeID = ANY (SELECT EmployeeID FROM
    employee_orders WHERE Status = 'Completed');
```

Listing 44: Contoh Operator ANY

- `EXISTS`: Digunakan untuk mencari keberadaan baris dalam tabel tertentu yang memenuhi kriteria tertentu.
- `NOT`: Membalikkan arti dari operator logika yang digunakan bersamanya, seperti `NOT EXISTS`, `NOT BETWEEN`, `NOT IN`, dll.
- `OR`: Digunakan untuk menggabungkan beberapa kondisi dalam klausa `WHERE` dari pernyataan SQL.
- `IS NULL`: Digunakan untuk membandingkan suatu nilai dengan nilai `NULL`.

3.16 Contoh Penggunaan WHERE Clause Praktis

- Mengambil Data Berdasarkan Fakultas: `SELECT * FROM student_table WHERE faculty = 'engineering';`
- Mengambil Data Berdasarkan Usia: `SELECT * FROM student_table WHERE age >= 18;`
- Mengambil Data dengan Beberapa Kriteria: `SELECT * FROM student_table WHERE faculty = 'engineering' AND age < 25;`

3.17 Catatan Penting WHERE Clause

- Penggunaan Tanda Kutip: Untuk nilai teks, selalu gunakan tanda kutip tunggal (`' '`) di sekitar nilai.
- Kombinasi Operator: Anda dapat menggabungkan beberapa kondisi menggunakan operator logika seperti `AND` dan `OR` untuk memperluas filter.

3.18 Pengertian SELECT DISTINCT

- Pernyataan `SELECT DISTINCT` digunakan untuk mengembalikan nilai yang unik dari kolom dalam tabel, tanpa adanya duplikat.
- Contoh: Jika Anda ingin mengetahui negara asal mahasiswa, Anda dapat menggunakan `SELECT DISTINCT` untuk mendapatkan daftar negara tanpa duplikat.

3.19 Penggunaan SELECT DISTINCT dalam Kueri

- Untuk menggunakan `SELECT DISTINCT`, Anda menulis pernyataan `SELECT` diikuti dengan kata `DISTINCT` dan nama kolom yang ingin Anda ambil, kemudian diakhiri dengan nama tabel.
- Contoh sintaks: `SELECT DISTINCT country FROM students;` akan mengembalikan daftar negara yang unik dari tabel mahasiswa.

3.20 Menggunakan DISTINCT dengan Fungsi Agregat SQL

- `DISTINCT` dapat digunakan dengan fungsi agregat seperti `COUNT` untuk menghitung jumlah nilai unik, fungsi lain seperti `AVG`, `MAX`.
- Contoh: `SELECT COUNT(DISTINCT country)` untuk menghitung jumlah negara unik dalam tabel pelanggan.

3.21 Poin Penting tentang SELECT DISTINCT

- Jika hanya satu kolom yang diberikan, query akan mengembalikan nilai unik untuk kolom tersebut.
- Jika lebih dari satu kolom yang diberikan, query akan mengambil kombinasi unik untuk kolom-kolom tersebut.
- `NULL` dianggap sebagai nilai unik oleh `DISTINCT`.

4 Database Design

4.1 Definisi Skema Basis Data

- Skema adalah organisasi atau pengelompokan informasi dan hubungan di antara mereka. Bisa dipahami sebagai Blueprint dari database.
- Dalam konteks basis data MySQL, skema berarti koleksi struktur data atau desain abstrak tentang bagaimana data disimpan
- Proses perancangan skema basis data dikenal sebagai pemodelan data dan dilakukan oleh perancang basis data

4.2 Perbedaan Definisi Skema di Berbagai Sistem Basis Data

- SQL Server: Skema adalah kumpulan komponen seperti tabel, kolom, tipe data, dan kunci.
- PostgreSQL: Skema adalah namespace untuk objek basis data seperti tampilan, indeks, dan fungsi.
- Oracle: Setiap pengguna memiliki satu skema yang dinamai sesuai dengan pengguna tersebut.

4.3 Tiga Jenis Skema Basis Data

- Skema Konseptual atau Logis (Logical)
  - Menjelaskan struktur keseluruhan basis data untuk semua pengguna, termasuk entitas, atribut, dan hubungan antar entitas.
  - Biasanya diwakili dengan Diagram Hubungan Entitas (ER-D).
- Skema Internal atau Fisik (Physical)
  - Menggambarkan penyimpanan fisik data dalam basis data, termasuk bagaimana data disimpan di disk dalam bentuk tabel, kolom, dan catatan. Lebih dekat dengan kode.
  - Menyediakan detail tentang penyimpanan dan jalur akses data.
- Skema Eksternal atau Tampilan (View)
  - Menjelaskan bagaimana pengguna eksternal dapat melihat data, hanya mencakup bagian yang relevan bagi pengguna tertentu.
  - Ada banyak skema eksternal untuk satu basis data, tergantung pada kebutuhan pengguna.

4.4 Keuntungan Skema Basis Data

- Menyediakan pengelompokan logis untuk objek basis data.
- Mempermudah akses dan manipulasi objek basis data.
- Meningkatkan keamanan basis data dengan memberikan izin berdasarkan hak akses pengguna.
- Memungkinkan transfer kepemilikan skema dan objeknya antara pengguna dan skema lain.
- Membantu insinyur basis data mengorganisir data ke dalam tabel yang terdefinisi dengan baik.
- Memudahkan pengembangan aplikasi dengan memberikan pemahaman tentang bagaimana data harus disimpan.
- Mencegah kebutuhan untuk rekayasa ulang model data di masa depan, yang dapat menghemat waktu dan biaya.

4.5 Kunci Utama dan Kunci Asing

- Kunci utama (primary key) digunakan untuk mengidentifikasi setiap baris dalam tabel secara unik.
- Kunci asing (foreign key) digunakan untuk menghubungkan tabel yang berbeda, memastikan integritas referensial antara tabel.

4.6 Membangun Skema Basis Data

- Skema basis data adalah langkah pertama dalam desain basis data, penting untuk memastikan struktur yang solid sebelum melanjutkan.
- Skema terdiri dari objek skema seperti tabel, kolom, dan hubungan, serta tipe data, kunci utama, dan kunci asing.

4.7 Jenis Hubungan/Relasi Antara Tabel

- Sau ke Banyak (One-to-Many)
  - Satu catatan di satu tabel dapat terhubung dengan banyak catatan di tabel lain. Contohnya, seorang siswa dapat terdaftar di beberapa kursus.
- Satu ke Satu (One-to-One)
  - Satu catatan di satu tabel terhubung dengan satu catatan di tabel lain. Misalnya, setiap kepala departemen memiliki satu lokasi di kampus.

- Banyak ke Banyak (Many-to-Many)
  - Satu catatan di satu tabel dapat terhubung dengan banyak catatan di tabel lain dan sebaliknya. Contohnya, seorang siswa dapat terlibat dalam beberapa proyek penelitian yang diawasi oleh beberapa staf.

4.8 Diagram Hubungan Entitas (ER-Diagram)

- Menggunakan ERD (Entity Relationship Diagram) untuk menggambarkan hubungan antara entitas, seperti siswa dan kursus, dengan simbol dan bentuk yang sesuai.
- Menunjukkan bagaimana kunci primer dan kunci asing digunakan untuk menjaga integritas referensial antara tabel.

4.9 Konsep Dasar Model Relasional

- Data: Basis data terdiri dari kumpulan tabel yang saling terkait, di mana setiap tabel menyimpan data dalam bentuk baris dan kolom.
- Hubungan: Tabel-tabel dalam basis data dapat saling berhubungan melalui kunci primer dan kunci asing.
- Keterbatasan: Setiap tabel harus memenuhi kondisi tertentu yang dikenal sebagai batasan integritas relasional.

4.10 Elemen Utama dalam Model Relasional

- Relasi: Merupakan tabel yang menyimpan data, di mana setiap baris disebut sebagai tuple dan setiap kolom sebagai atribut.
- Kolom: Menyimpan data dalam bentuk atribut, dengan setiap kolom memiliki tipe data tertentu (misalnya, numerik, teks, atau tanggal).
- Domain: Kumpulan nilai yang diperbolehkan untuk setiap kolom, tergantung pada tipe data kolom tersebut.

4.11 Keterbatasan (Constraint)

- Key Constraints: Setiap baris dalam tabel harus memiliki atribut yang dapat secara unik mengidentifikasi baris tersebut. Tidak boleh NULL.
- Domain Constraints: Memastikan bahwa nilai yang dimasukkan ke dalam kolom sesuai dengan tipe data yang ditentukan.
- Referential Integrity Constraints: Mengatur hubungan antar tabel melalui kunci asing, yang harus merujuk pada kunci primer di tabel lain.

4.12 Tipe Hubungan dalam Model Relasional

- One-to-one (1:1): Setiap record di satu tabel berhubungan dengan satu record di tabel lain.
- One-to-many (1:N): Satu record di tabel pertama dapat berhubungan dengan banyak record di tabel kedua.
- Many-to-many (N:N): Banyak record di satu tabel dapat berhubungan dengan banyak record di tabel lain, biasanya dipecah menjadi dua hubungan satu-ke-banyak dengan tbel penghubung.

4.13 Definisi Lain

- Degree adalah angka dari jumlah kolom atribut pada sebuah tabel. Jika di suatu tabel terdapat kolom ID dan nama maka tabel tersebut memiliki 2 degree.
- Cadinality merujuk seberapa banyak record yang ada di dalam suatu tabel. Jika kita punya 100 siswa di tabel siswa maka tabel tersebut punya seratus cardinality.

4.14 Kunci Primer (Primary Key)

- Kunci primer atau Primary Key adalah atribut yang unik untuk setiap baris dalam tabel dan tidak boleh memiliki nilai null. Contohnya, dalam tabel siswa, ID siswa dapat digunakan sebagai kunci primer untuk mengidentifikasi siswa secara unik.
- Jika ada beberapa kandidat kunci yang dapat digunakan, satu akan dipilih sebagai kunci primer, sementara yang lainnya menjadi kunci sekunder.

4.15 Kunci Komposit (Composite Primary Key)

- Kunci komposit atau Composite Primary Key digunakan ketika tidak ada kolom tunggal yang memiliki nilai unik. Ini adalah kombinasi dari dua atau lebih atribut yang bersama-sama membentuk nilai unik untuk setiap baris.
- Contoh penggunaan kunci komposit adalah dalam tabel pengiriman, di mana kombinasi ID pelanggan dan kode proyek dapat digunakan untuk mengidentifikasi pengiriman secara unik.

4.16 Konsep Foreign Key

- Foreign key adalah kolom atau kombinasi kolom yang digunakan untuk menghubungkan dua tabel dalam basis data relasional, menciptakan referensi silang antara mereka.

- Dalam basis data toko online, tabel Customer dan tabel Order dihubungkan melalui foreign key (customer ID) di tabel Order.

4.17 Contoh Sintaks FOREIGN KEY

```
CREATE TABLE Bookings (
    BookingID INT NOT NULL PRIMARY KEY,
    BookingDate DATE NOT NULL,
    TableNumber INT NOT NULL,
    NumberOfGuests INT NOT NULL CHECK (NumberOfGuests <= 8),
    CustomerID INT NOT NULL,
    FOREIGN KEY (CustomerID) REFERENCES Customers (
        CustomerID)
        ON DELETE CASCADE
        ON UPDATE CASCADE
        ON DELETE SET NULL
);
```

Listing 45: Contoh syntax Foreign Key

- ON DELETE CASCADE dan ON UPDATE CASCADE menjaga integritas relasi parent–child tanpa harus menulis query manual, dan dua hal ini optional. Maka deklarasi foreign key bisa hanya dengan sintaks FOREIGN KEY (CustomerID) REFERENCES Customers (CustomerID).

4.18 Hubungan Tabel

- Tabel parent (Customer) dapat ada tanpa child (Order), tetapi child tidak dapat ada tanpa parent.

4.19 Tabel Sebagai Parent dan Child

- Tabel dapat berfungsi sebagai parent dan child secara bersamaan, seperti dalam contoh tabel `Employee` di mana karyawan dapat menjadi manajer (parent) dan juga memiliki manajer (child).

4.20 Konsep Kunci dalam Basis Data

- Relational Database: Kumpulan data yang dikelola dalam sistem manajemen basis data seperti Oracle atau MySQL, memungkinkan pengambilan data dengan spesifikasi tabel dan kolom yang diinginkan.

- Kunci Utama (Primary Key): Atribut yang secara unik mengidentifikasi setiap baris dalam tabel. Kunci utama harus dipilih dari kunci kandidat yang memiliki nilai unik di setiap baris. Contoh: `Vehicle ID` dalam tabel kendaraan.

4.21 Identifikasi Kunci Kandidat

- Dalam tabel kendaraan pada video materi, terdapat tiga kunci kandidat: Vehicle ID, Nomor Plat Mobil, dan Nomor Telepon Pemilik.
- Meskipun Owner ID memiliki nilai unik, tidak disarankan sebagai kunci utama karena seorang pemilik dapat memiliki lebih dari satu kendaraan, sehingga tidak memenuhi syarat keunikan.

4.22 Pemilihan Kunci Utama yang Tepat

- Vehicle Plate Number: Dapat berubah jika pemilik mengganti nomor plat, sehingga tidak ideal sebagai kunci utama.
- Owner Phone Number: Juga tidak stabil karena dapat berubah, yang dapat menyebabkan kebingungan dalam database.
- Vehicle ID: Merupakan pilihan terbaik karena unik dan tidak diharapkan berubah.

4.23 Pembuatan Tabel dalam SQL

- Membuat Database: Gunakan perintah `CREATE DATABASE automobile;` untuk membuat database baru.
- Membuat Tabel: Gunakan perintah `CREATE TABLE` untuk mendefinisikan tabel kendaraan dengan kolom yang sesuai, termasuk mendeklarasikan `Vehicle ID` sebagai kunci utama.

```
CREATE TABLE vehicle(
    vehicleID varchar(10),
    ownerID varchar(10),
    plateNumber varchar(10),
    phoneNumber INT,
    PRIMARY KEY (vehicleID)
);
```

Listing 46: Contoh perintah untuk membuat tabel kendaraan

4.24 Mengubah Struktur Tabel

- Untuk mendefinisikan `Owner ID` sebagai kunci asing, gunakan perintah `ALTER TABLE`:

```
ALTER TABLE vehicle ADD FOREIGN KEY (ownerID) REFERENCES
    owner(ownerID);
```

4.25 Keyword CONSTRAINT sebelum deklarasi ATURAN

- Dengan menambahkan `CONSTRAINT` diikuti dengan nama constraint sebelum keyword `PRIMARY KEY`, Anda menetapkan nama aturan keunikan dan integritas untuk kolom yang ditentukan sebagai kunci utama. Ini memastikan bahwa kolom tersebut tidak hanya unik, tetapi juga tidak boleh memiliki nilai NULL.
- Hal yang sama (Penamaan `CONSTRAINT`) bisa dilakukan untuk aturan lain seperti `UNIQUE`, `CHECK`, `FOREIGN KEY`, dll.

```
-- PRIMARY KEY
CREATE TABLE Mahasiswa (
  ID INT,
  Nama VARCHAR(100),
  Email VARCHAR(100),
  CONSTRAINT PK_Mahasiswa PRIMARY KEY (ID)
);

-- UNIQUE
CREATE TABLE Users (
    UserID INT PRIMARY KEY,
    Email  VARCHAR(255),
    CONSTRAINT UQ_Users_Email UNIQUE (Email)
);

-- CHECK
CREATE TABLE Products (
    ProductID INT PRIMARY KEY,
    Price DECIMAL(10,2) NOT NULL,
    CONSTRAINT CK_Products_Price CHECK (Price > 0)
);

-- FOREIGN KEY
CREATE TABLE Orders (
    OrderID INT PRIMARY KEY,
    CustomerID INT,
    CONSTRAINT FK_Orders_Customers
        FOREIGN KEY (CustomerID) REFERENCES Customers(
            CustomerID)
);
```

Listing 47: Sintaks Penamaan CONSTRAINT

4.26 Entitas dalam Basis Data

- Entitas adalah objek yang memiliki karakteristik yang dapat diidentifikasi. Contoh entitas termasuk pelanggan, produk, atau pesanan.

- Dalam tabel pengiriman untuk toko e-commerce, nama tabel 'deliveries' adalah entitas, dan kolom-kolom seperti ID, nama, dan status pengiriman adalah atribut yang terkait.

4.27 Jenis Atribut dalam Basis Data

- Simple Attribute: Tidak dapat dibagi lebih lanjut. Contoh: nilai grade.
- Composite Attribute: Dapat dibagi menjadi sub-atribut. Contoh: nama lengkap yang terdiri dari nama depan dan nama belakang.
- Single Value Attribute: Hanya menyimpan satu nilai. Contoh: tanggal lahir.
- Multi Value Attribute: Dapat menyimpan beberapa nilai. Contoh: kolom email yang menyimpan lebih dari satu alamat email (sebaiknya dihindari dalam basis data relasional).
- Derived Attribute: Nilai yang dihitung dari atribut lain. Contoh: usia yang dihitung dari tanggal lahir.
- Key Attribute: Menyimpan nilai unik untuk mengidentifikasi catatan. Contoh: ID siswa yang unik.

4.28 Pentingnya Memilih Entitas dan Atribut

- Hanya pertimbangkan entitas dan atribut yang relevan untuk proyek Anda. Data yang tidak digunakan tidak perlu disimpan dalam sistem basis data.
- Memastikan bahwa data yang disimpan membantu pengguna menyelesaikan tugas dan aktivitas tertentu.

4.29 Entity Relationship Diagram (ER-D)

- ER-D (Entity Relationship Diagram) adalah alat untuk merepresentasikan dan mendokumentasikan model hubungan entitas dalam basis data.
- Membantu dalam merancang basis data dengan memastikan bahwa semua entitas, atribut, dan hubungan terdefinisi dengan baik.

4.30 Simbol dan Notasi dalam ER-D

- Representasi Entitas: Kotak dengan dua kompartemen (nama entitas di atas, atribut di bawah).
- Representasi Hubungan:
  - Garis lurus untuk hubungan 1:1.

  - Garis lurus dengan notasi kaki gagak untuk hubungan 1:N.
  - Garis lurus dengan notasi kaki gagak di kedua sisi untuk hubungan M:N.

4.31 Pengertian Normalisasi (Normalization)

- Normalisasi (`Normalization`) adalah proses yang digunakan untuk mengatur struktur tabel dalam basis data agar meminimalkan masalah yang sering muncul, seperti duplikasi data, kesulitan dalam memperbarui data, dan kompleksitas dalam kueri data.
- Proses ini bertujuan untuk menciptakan desain basis data yang efisien dan terorganisir.

4.32 Anomali dalam Basis Data

- Anomali Penyisipan (`Insert Anomaly`)
  - Terjadi ketika penambahan data baru memerlukan data tambahan yang tidak tersedia.
  - Contoh: Dalam tabel pendaftaran mahasiswa, jika ingin menambahkan kursus baru, Anda tidak dapat melakukannya tanpa mendaftarkan siswa baru terlebih dahulu, karena kolom ID siswa harus terisi.
- Anomali Pembaruan (`Update Anomaly`)
  - Terjadi ketika memperbarui satu catatan memerlukan pembaruan di beberapa tempat dalam tabel.
  - Contoh: Jika direktur departemen diganti, Anda harus memperbarui nama direktur di setiap catatan siswa yang terdaftar di departemen tersebut. Jika ada yang terlewat, informasi menjadi tidak konsisten.
- Anomali Penghapusan (`Deletion Anomaly`)
  - Terjadi ketika penghapusan satu catatan menyebabkan hilangnya data penting lainnya.
  - Contoh: Menghapus data seorang siswa yang terdaftar di beberapa kursus dapat menghapus informasi tentang departemen yang terkait dengan siswa tersebut, jika tidak dirancang dengan baik.

4.33 Solusi Mengatasi Anomali Melalui Normalisasi

- Untuk mengatasi masalah anomali, tabel yang tidak dinormalisasi perlu dipecah menjadi beberapa tabel dengan tujuan tunggal (Tabel Mahasiswa, Tabel Kursus, Tabel Departemen).
- Pemisahan ini membantu mengurangi duplikasi data dan mempermudah pemeliharaan serta pembaruan informasi.

- Dengan desain yang dinormalisasi, penulisan kueri SQL menjadi lebih sederhana dan efisien, memungkinkan pencarian, pengurutan, dan analisis data yang lebih mudah.

4.34 Tiga Bentuk Normal Form

4.34.1 Bentuk Normal Pertama (1NF)

- Definisi: Tabel harus memiliki nilai atomik, artinya setiap kolom hanya boleh berisi satu nilai (tidak boleh ada nilai ganda dalam satu kolom).
- 1NF fokus ke `atomicity` dan menghindari `repeating groups`.
- `Repeating group` yang melanggar 1NF terjadi ketika banyak field non-key yang memiliki record berulang. Maka ini harus dipecah sebagai entitas lain/tabel lain.
- Kalau ada field berulang di beberapa baris (misalnya ID sama) tapi kolom lain berbeda, itu tidak melanggar 1NF dan justru representasi relasi `one-to-many`. Namun `Primary Key` harus tetap dipastikan unik, bisa dengan `surrogate PK` atau `composite PK`.
- Contoh: Dalam tabel awal, kolom seperti nama pasien dan total biaya mungkin memiliki beberapa nilai dalam satu sel. Untuk memenuhi 1NF, kita harus memisahkan data pasien ke dalam tabel terpisah.

```
CREATE TABLE Patient (
    PatientID VARCHAR(10) NOT NULL,
    PatientName VARCHAR(50),
    SlotID VARCHAR(10) NOT NULL,
    TotalCost Decimal,
    CONSTRAINT PK_Patient PRIMARY KEY (PatientID, SlotID)
);
```

Listing 48: SQL untuk Membuat Tabel Pasien (Memenuhi 1NF)

4.34.2 Bentuk Normal Kedua (2NF)

- Definisi: Menghindari ketergantungan parsial, di mana atribut non-kunci bergantung hanya pada sebagian dari kunci komposit.
- Ketergantungan fungsional adalah hubungan antara dua atribut dalam tabel, di mana nilai unik dari satu kolom (kunci primer) menentukan nilai kolom lainnya.
- Ketergantungan parsial terjadi pada tabel dengan kunci primer komposit, di mana atribut non-kunci hanya bergantung pada sebagian dari nilai kunci primer.

Meningkatkan Tabel ke 2NF

- Untuk memenuhi kriteria 2NF, semua atribut non-kunci harus bergantung pada seluruh nilai kunci primer.
- Tabel yang tidak memenuhi kriteria ini harus dipecah menjadi beberapa tabel terpisah untuk menghilangkan ketergantungan parsial.
- Contoh: Dalam tabel pasien, nama pasien dan total biaya dapat ditentukan hanya dengan menggunakan ID pasien. Ini menunjukkan ketergantungan parsial. Untuk memperbaikinya, kita memisahkan tabel menjadi tabel pasien dan tabel janji.

```
CREATE TABLE Patient (
    PatientID VARCHAR(10) NOT NULL,
    PatientName VARCHAR(50),
    PRIMARY KEY (PatientID)
);
```

Listing 49: SQL untuk Membuat Tabel Pasien Baru (Memenuhi 2NF)

4.34.3 Bentuk Normal Ketiga (3NF)

- Definisi: Tidak ada ketergantungan transitif, di mana atribut non-kunci tidak boleh bergantung pada atribut non-kunci lainnya.
- Contoh: Dalam tabel operasi, jika kolom kode pos bergantung pada dewan, maka ini adalah ketergantungan transitif. Kita harus memisahkan tabel menjadi tabel lokasi dan tabel dewan.

```
CREATE TABLE Location (
    SurgeryNumber INT NOT NULL,
    Postcode VARCHAR(10),
    PRIMARY KEY (SurgeryNumber)
);
```

Listing 50: SQL untuk Membuat Tabel Lokasi (Memenuhi 3NF)

4.35 Ilustrasi Proses Normalisasi

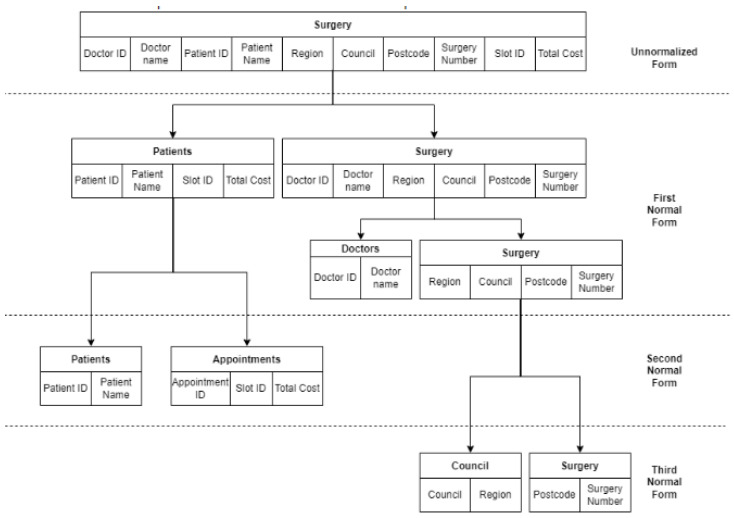

Figure 1: Proses Normalisasi dari NF1 sampai NF3