Database Structure and Management with MySQL

Maulana Hafidz Ismail

10 Oktober 2025

Contents

1 Intro to MySQL

1.1 Pengantar WHERE Clause

- `WHERE clause` digunakan dalam pernyataan SQL untuk **menentukan kondisi atau aturan dalam memfilter data**.
- Sintaks dasar: `SELECT <kolom> FROM <tabel> WHERE <kondisi>`.

1.2 Kolasi (Collation)

- **Kolasi** menentukan bagaimana `string` dibandingkan dan diurutkan.
- Beberapa kolasi bersifat `case-sensitive` (membedakan huruf besar dan kecil), sementara yang lain bersifat `case-insensitive` (tidak membedakan).
- **Default Kolasi** adalah kolasi yang otomatis diterapkan pada `database`, tabel, atau kolom jika tidak ditentukan secara eksplisit, yang dapat bervariasi tergantung pengaturan `server MySQL` (contoh: `utf8mb4_general_ci` adalah `case-insensitive`).

1.3 Operator Logika (Logical Operators)

1.3.1 AND Operator

- Digunakan untuk memfilter data dengan memastikan bahwa **semua kondisi** yang digabungkan harus bernilai `true`.

```
SELECT * FROM customer_purchases
WHERE purchases > 2000 AND location = 'Gila␣County';
```

Listing 1: Contoh Penggunaan AND

1.3.2 OR Operator

- Memungkinkan `record` untuk dimasukkan jika **salah satu dari kondisi** yang digabungkan bernilai `true`.

```
SELECT * FROM customer_purchases
WHERE location = 'Gila␣County' OR location = 'Santa␣Cruz␣
    County';
```

Listing 2: Contoh Penggunaan OR

1.3.3 NOT Operator

- Membalikkan hasil, memilih `record` hanya jika kondisi yang ditentukan **tidak bernilai true**.

```
SELECT * FROM customer_purchases
WHERE NOT (location = 'Gila County' OR location = 'Santa
    Cruz County');
```

Listing 3: Contoh Penggunaan NOT

1.4 Operator Khusus dalam WHERE Clause

1.4.1 Operator IN

- Memungkinkan Anda untuk menentukan **beberapa nilai** dalam klausa `WHERE`. Berfungsi sebagai singkatan untuk beberapa kondisi `OR`.
- Dapat juga menggunakan `NOT IN` untuk hasil yang berlawanan.

```
SELECT * FROM customer_purchases
WHERE purchases IN (2000, 2500, 3000)
AND location IN ('Gila County', 'Santa Cruz County');
```

Listing 4: Contoh Penggunaan IN

1.4.2 Operator BETWEEN

- Digunakan untuk memilih nilai dalam **rentang tertentu** (angka, teks, atau tanggal).
- **Inklusi** berlaku: nilai batas rentang (contoh: 10 dan 20 pada `BETWEEN 10 AND 20`) akan masuk ke dalam set hasil.

```
SELECT * FROM customer_purchases
WHERE purchases BETWEEN 1000 AND 2000;
```

Listing 5: Contoh Penggunaan BETWEEN

1.4.3 Operator LIKE

- Digunakan untuk memfilter data berdasarkan **pencocokan pola**.
- **Wildcard**: `%` (nol, satu, atau beberapa karakter) dan `_` (satu karakter tunggal).

```sql
SELECT * FROM customer_purchases
WHERE location LIKE 'g__%'; -- Dimulai 'g' dan minimal 3
    karakter
```

Listing 6: Contoh Penggunaan LIKE

—

1.5 Alias dalam SQL

- **Alias** digunakan untuk memberikan **nama sementara** pada kolom dan tabel.
- **Situasi Penggunaan**:
    1. `Renaming`: Mengganti nama tabel atau kolom yang terlalu panjang/teknis.
    2. `Concatenation`: Menggabungkan `output` dari beberapa kolom menjadi satu.
    3. `Multiple Tables`: Mempermudah `query` saat berhadapan dengan banyak tabel (menggunakan notasi titik).

1.5.1 Sintaks Penggunaan Alias

```sql
SELECT original_column_name AS alias_name
FROM table_name;

SELECT CONCAT(column1, '␣', column2) AS new_column_name
FROM table_name;
```

Listing 7: Renaming Kolom dan Concatenation

```sql
SELECT x.column1, y.column2
FROM table1 AS x, table2 AS y
WHERE x.condition = y.condition;
```

Listing 8: Querying Multiple Tables dengan Alias

—

1.6 JOIN dalam MySQL

- **JOIN** menghubungkan catatan data antara satu atau beberapa tabel berdasarkan kolom yang umum di antara mereka.

1. **INNER JOIN**: Mengembalikan catatan data yang **memiliki nilai yang cocok** di kedua tabel yang digabungkan.

2. **LEFT JOIN**: Mengembalikan **semua catatan dari tabel kiri** dan catatan yang cocok dari tabel kanan. Jika tidak cocok, nilai `NULL` akan dimasukkan untuk kolom dari tabel kanan.
3. **RIGHT JOIN**: Mengembalikan **semua catatan dari tabel kanan** dan catatan yang cocok dari tabel kiri. Jika tidak cocok, nilai `NULL` akan dimasukkan untuk kolom dari tabel kiri.
4. **SELF JOIN**: Menggabungkan tabel dengan dirinya sendiri untuk mendapatkan informasi spesifik yang ada dalam tabel yang sama.

```
SELECT Customers.FullName, Bookings.BookingID
FROM Customers INNER JOIN Bookings
ON Customers.CustomerID = Bookings.CustomerID;
```

Listing 9: Contoh INNER JOIN

—

1.7 Operator UNION

- **UNION** digunakan untuk **menggabungkan hasil dari beberapa pernyataan SELECT** dalam satu `query`.

1.7.1 Sintaks dan Praktik Terbaik

```
SELECT column1, column2, ... FROM table1 WHERE condition
UNION
SELECT column1, column2, ... FROM table2 WHERE condition;
```

Listing 10: Sintaks Dasar UNION

- **Syarat**: Setiap pernyataan `SELECT` harus memiliki **jumlah kolom yang sama**, **tipe data yang serupa**, dan **urutan yang sama**.
- **Default**: Operator `UNION` hanya mengembalikan **nilai yang unik**.
- **Duplikat**: Untuk mengembalikan semua nilai, termasuk yang duplikat, gunakan `UNION ALL`.

—

1.8 GROUP BY dan Fungsi Agregat

1.8.1 GROUP BY Clause

- **GROUP BY** digunakan untuk **mengelompokkan baris** dalam tabel berdasarkan kolom tertentu menjadi baris ringkasan (`subgrup`).
- Satu baris hasil mewakili beberapa baris yang dikelompokkan berdasarkan nilai yang sama.

1.8.2 Fungsi Agregat (Aggregate Functions)

Fungsi ini sering digunakan dengan `GROUP BY` untuk melakukan perhitungan dan mengembalikan satu nilai untuk setiap `subgrup`:

- `SUM()`: Menjumlahkan nilai kolom.
- `AVG()`: Menghitung rata-rata nilai kolom.
- `MAX()`: Mengembalikan nilai maksimum.
- `MIN()`: Mengembalikan nilai minimum.
- `COUNT()`: Menghitung jumlah kemunculan nilai.

```
SELECT Department, COUNT(OrderID)
FROM orders
GROUP BY Department;
```

Listing 11: Contoh Penggunaan GROUP BY

—

1.9 HAVING Clause

- **HAVING clause** digunakan untuk menentukan **kondisi filter pada data yang telah dikelompokkan** oleh `GROUP BY clause`.
- **Perbedaan WHERE vs. HAVING**:
  - `WHERE` memfilter data **sebelum** pengelompokan.
  - `HAVING` memfilter data **setelah** pengelompokan (pada hasil agregasi).
- Jika `HAVING` digunakan tanpa `GROUP BY`, fungsinya mirip dengan `WHERE`.

```
SELECT Department, SUM(OrderTotal) AS TotalSales
FROM Orders
GROUP BY Department
HAVING SUM(OrderTotal) > 2275; -- Hanya departemen dengan
    TotalSales > 2275
```

Listing 12: Contoh Penggunaan HAVING

2 Updating Databases and Working with Views

2.1 Pengantar REPLACE Command

- `REPLACE command` digunakan untuk **menyisipkan atau memperbarui data** dalam tabel.
- **Cara Kerja**: Memeriksa kunci primer atau unik. Jika ditemukan kunci duplikat, `record` yang ada akan **dihapus dan diganti** dengan yang baru. Jika tidak ada yang cocok, berfungsi seperti `INSERT`.
- **Tujuan**: Mengganti `record` yang ada tanpa menghasilkan kesalahan duplikat (`duplicate entry`).

2.1.1 Sintaks REPLACE Command

```
REPLACE INTO table_name (column1, column2, ...)
VALUES (value1, value2, ...);

REPLACE INTO table_name
SET column1 = value1, column2 = value2, ...;
```

Listing 13: Sintaks REPLACE Command

—

2.2 Constraints (Batasan)

- **Constraints** digunakan untuk memastikan bahwa tabel hanya menerima **data yang valid**.

2.2.1 Tipe-Tipe Constraints

1. **Key Constraints**: Menerapkan aturan pada jenis kunci (*misalnya*, `PRIMARY KEY` harus unik dan `NOT NULL`).
2. **Domain Constraints**: Mengatur nilai yang dapat disimpan dalam kolom tertentu (*misalnya*, menggunakan `CHECK` untuk membatasi jumlah tamu).
3. **Referential Integrity Constraints**: Aturan untuk kunci referensial (`FOREIGN KEY`), memastikan nilai di tabel referensi selalu ada di tabel yang direferensikan.

2.2.2 Opsi Cascade

Opsi ini digunakan untuk menjaga integritas data secara otomatis pada `FOREIGN KEY`:

- `ON DELETE CASCADE`: Menghapus baris terkait di tabel referensi jika baris di tabel utama dihapus.

- `ON UPDATE CASCADE`: Memperbarui baris terkait jika nilai `PRIMARY KEY` di tabel utama diperbarui.

—

2.3 ALTER Statement

- `ALTER statement` digunakan untuk **menambah, menghapus, dan memodifikasi kolom** serta `constraints` dalam tabel yang sudah ada.

2.3.1 Perintah Umum dalam ALTER TABLE

```
ALTER TABLE table_name MODIFY COLUMN column_name datatype;
    -- Memodifikasi kolom
ALTER TABLE table_name ADD column_name datatype;        --
    Menambah kolom baru
ALTER TABLE table_name DROP COLUMN column_name;        --
    Menghapus kolom
ALTER TABLE table_name RENAME TO new_table_name;       --
    Mengganti nama tabel
ALTER TABLE table_name CHANGE from_column to_column datatype
    ; -- Mengganti nama kolom/field
```

Listing 14: Perintah ALTER TABLE

—

2.4 Menyalin Tabel

- **Menyalin Data dan Struktur** (`CREATE TABLE ... AS SELECT`): Menyalin data dan mendefinisikan struktur berdasarkan hasil `SELECT`.

```
CREATE TABLE ClientsTest AS
SELECT * FROM clients;

-- Menyalin tabel antar database
CREATE TABLE testDB.ClientsTest AS
SELECT * FROM LuckyShrub.clients;
```

Listing 15: Menyalin Tabel dengan Data

- **Menyalin Struktur** (`CREATE TABLE ... LIKE`): Menyalin hanya struktur dan semua `constraints` (kecuali `FOREIGN KEY`) dari tabel sumber. Data harus dimasukkan secara terpisah.

```
CREATE TABLE ClientsTest3 LIKE clients;
INSERT INTO ClientTest3 AS SELECT * FROM clients;
```

Listing 16: Menyalin Struktur Tabel dan Data

—

2.5 Subquery

- **Subquery** adalah `query` yang berada **di dalam query lain** (`child query` di dalam `parent query`).
- **Eksekusi**: Subquery dieksekusi terlebih dahulu, dan hasilnya digunakan oleh `query` luar.
- **Bentuk Hasil**: Dapat mengembalikan `nilai tunggal`, satu baris, satu kolom, atau beberapa baris.
- **Operator Perbandingan Kompleks**: Dapat digunakan dengan `multiple row subqueries`:
  - `ANY/SOME`: Mengembalikan data jika nilai memenuhi kondisi **salah satu** nilai yang dikembalikan `subquery`.
  - `ALL`: Mengembalikan data jika nilai memenuhi kondisi **semua** nilai yang dikembalikan `subquery`.

2.5.1 Tipe-Tipe Subquery Berdasarkan Lokasi

- **Dalam klausa** `FROM`: Hasil `subquery` digunakan sebagai **tabel sementara** (`derived table`).
- **Dalam pernyataan** `INSERT/UPDATE/DELETE`: Digunakan dalam klausa `WHERE` atau sebagai sumber data (`INSERT`).

```
SELECT some_column, some_column
FROM (Subquery) AS alias;
```

Listing 17: Subquery dalam Klausa FROM

2.5.2 Operator EXISTS dan NOT EXISTS

- `EXISTS`: Menguji **keberadaan baris** dalam hasil yang dikembalikan oleh `subquery`. Mengembalikan `true` jika `subquery` menghasilkan **satu atau lebih** `record`.
- `NOT EXISTS`: Menguji **ketidakadaan hasil**. Mengembalikan `true` jika `subquery` **tidak mengembalikan** baris hasil.

```
SELECT employee_id, employee_name
FROM employees
WHERE EXISTS (
    SELECT *
    FROM departments
    WHERE departments.department_id = employees.
        department_id
```

```
);
```

Listing 18: Contoh EXISTS

—

2.6 Views (Tampilan)

- **Views** adalah **tabel virtual** yang dibuat dari satu atau beberapa tabel yang ada, tetapi **tidak menyimpan data sendiri**.
- **Fungsi**: Menyediakan akses data yang lebih sederhana, menyembunyikan data yang tidak ingin ditampilkan, menyederhanakan `query` kompleks, dan mengamankan data (`subset` data).
- **Sifat**: View dan tabel sumber terhubung. Operasi `UPDATE` pada `view` akan mengubah data pada tabel sumber.

2.6.1 Sintaks untuk Mengelola Views

```
CREATE VIEW view_name AS
    SELECT column1, column2
    FROM table_name
    WHERE condition;

ALTER VIEW nama_view AS
SELECT kolom1, kolom2
FROM tabel_asli
WHERE ...

RENAME TABLE old_view_name TO new_view_name;
DROP VIEW view_name;
```

Listing 19: Membuat dan Mengubah Views

3 Functions and MySQL Stored Procedures

3.1 Pengantar Fungsi MySQL

- **Fungsi** (`Function`) adalah potongan kode yang melakukan operasi dan **mengembalikan hasil**.
- Fungsi dapat menerima `parameter` atau `argumen` dan berguna untuk memanipulasi data.
- **Kategori Fungsi**: `Numeric`, `String`, `Date`, `Comparison`, dan `Control Flow`.

Table 1: Kategori dan Contoh Fungsi MySQL

Kategori	**Contoh Fungsi**
Numeric Aggregate	`SUM()`, `AVG()`, `MAX()`, `MIN()`, `COUNT()`
Numeric Math	`ROUND()`, `MOD()`, `CEIL()`, `FLOOR()`
String	`CONCAT()`, `SUBSTR()`, `UPPER()`, `LOWER()`, `LENGTH()`
Date	`CURRENT_DATE()`, `DATE_FORMAT()`, `DATEDIFF()`, `ADDDATE()`
Comparison	`GREATEST()`, `LEAST()`, `ISNULL()`, `COALESCE()`
Control Flow	`CASE`, `NULLIF()`, `IFNULL()`

3.2 Fungsi Numeric

3.2.1 Math Functions

- `ROUND(angka, tempat_desimal)`: Membulatkan angka ke tempat desimal tertentu.
- `MOD(angka, pembagi)`: Mengembalikan **sisa** dari pembagian.
- `CEIL(angka)`: Membulatkan **ke atas** ke integer terkecil yang tidak kurang dari nilai yang diberikan. (*Contoh*: `CEIL(3.1)` menghasilkan 4).
- `FLOOR(angka)`: Membulatkan **ke bawah** ke integer terbesar yang tidak lebih dari nilai yang diberikan.

```
SELECT ROUND(column_name, decimal_places) FROM table_name;
SELECT MOD(column_name, divisor) FROM table_name;
```

Listing 20: Contoh ROUND dan MOD

3.3 Fungsi String

Fungsi yang digunakan untuk memanipulasi nilai `string`:

- `CONCAT(s1, s2, ...)`: Menggabungkan beberapa string.
- `SUBSTRING(kolom, mulai, panjang)`: Mengekstrak segmen dari string (`start_index` mulai dari 1).
- `UPPER(kolom)/LOWER(kolom)`: Mengubah string menjadi huruf kapital/kecil.
- `FORMAT(angka, desimal)`: Memformat angka ke dalam format tertentu.
- `CHARINDEX(' ', FullName)/INSTR(kolom, " ")`: Mengembalikan posisi karakter/kata.
- `LENGTH(kolom)`: Mengembalikan panjang string.
- `SUBSTRING_INDEX(kolom, pemisah, hitungan)`: Memisahkan kata-kata berdasarkan pemisah.

```
SELECT SUBSTRING(column_name, 1, 5) FROM table_name;

-- Mengambil kata pertama (sebelum pemisah pertama)
SELECT SUBSTRING_INDEX(FullName, '␣', 1) AS FirstName
FROM Customers;
```

Listing 21: Contoh SUBSTRING dan SUBSTRING_INDEX

3.4 Fungsi Date

Fungsi untuk mengekstrak dan memformat nilai waktu dan tanggal.

- `CURRENT_DATE()`: Mengembalikan tanggal saat ini (`YYYY-MM-DD`).
- `CURRENT_TIME()`: Mengembalikan waktu saat ini (`HH:MM:SS`).
- `DATEDIFF(tanggal1, tanggal2)`: Mengidentifikasi jumlah hari antara dua tanggal.
- `ADDDATE(tanggal, INTERVAL X UNIT)`: Menambahkan jumlah hari, bulan, atau tahun.
- `QUARTER(tanggal)`: Mengembalikan kuartal tahun (1-4).

```
-- Mengubah format tanggal
SELECT DATE_FORMAT('2023-01-01', '%D␣%W␣%M␣%Y');

-- Menambahkan 30 hari
SELECT ADDDATE('2022-01-01', INTERVAL 30 DAY) AS newDate;
```

Listing 22: Contoh DATE_FORMAT dan ADDDATE

3.5 Fungsi Comparison (Perbandingan)

Fungsi yang memungkinkan perbandingan nilai:

- `GREATEST(v1, v2, ...)`: Menemukan nilai tertinggi.
- `LEAST(v1, v2, ...)`: Menentukan nilai terendah.
- `ISNULL(kolom)`: Menguji apakah suatu nilai adalah `NULL`.
- `COALESCE(v1, v2, ...)`: Mengembalikan **nilai pertama yang tidak NULL** dari daftar argumen.

```
SELECT COALESCE(column_name, 'Default␣Value') AS result FROM
    your_table;
```

Listing 23: Contoh COALESCE

3.6 Control Flow Functions

Fungsi yang memungkinkan evaluasi kondisi, mirip dengan logika `if-then-else`.

3.6.1 CASE Function

Berfungsi mirip dengan pernyataan `if-then-else`:

```
SELECT column_name,
    CASE
        WHEN condition1 THEN result1
        WHEN condition2 THEN result2
        ELSE result
    END AS alias_name
FROM table_name;
```

Listing 24: Sintaks CASE Function

3.6.2 NULLIF dan IFNULL

- `NULLIF(e1, e2)`: Mengembalikan `NULL` jika `e1` sama dengan `e2`; jika tidak, mengembalikan `e1`.
- `IFNULL(e1, e2)`: Mengembalikan `e1` jika tidak `NULL`; jika `e1` adalah `NULL`, mengembalikan `e2`.

```
SELECT amount / NULLIF(quantity, 0) AS result FROM sales;
```

Listing 25: Contoh NULLIF untuk Menghindari Pembagian Nol

—

3.7 Stored Procedures

- **Stored Procedure**: Blok kode yang dapat **disimpan dalam database dan dipanggil** menggunakan perintah `CALL`.
- **Manfaat**: Kode menjadi lebih konsisten, dapat digunakan kembali, dan mempermudah pemeliharaan.

3.7.1 Mode Parameter

Parameter dapat menerima 3 jenis argumen:

1. **IN** (Default): Mengirimkan nilai ke prosedur. Hanya dapat dibaca.
2. **OUT**: Mengembalikan nilai dari prosedur ke pemanggil.
3. **INOUT**: Mengirimkan nilai ke prosedur dan mengembalikan nilai yang telah diubah.

3.7.2 Sintaks dan Pengelolaan

```
CREATE PROCEDURE GetProductsDetails()
    BEGIN
        SELECT * FROM products;
    END;

CALL GetProductsDetails();
DROP PROCEDURE GetProductsDetails;
```

Listing 26: Membuat Stored Procedure Dasar

```
-- Parameter IN
CREATE PROCEDURE GetLowestPricedProducts(IN lowest_price INT
    )
    BEGIN
        SELECT * FROM products WHERE price <= lowest_price;
    END;

-- Parameter OUT
CREATE PROCEDURE GetProductCount(OUT total_count INT)
  BEGIN
    SELECT COUNT(*) INTO total_count FROM products;
  END;
```

Listing 27: Stored Procedure dengan Parameter IN dan OUT