

## Dokumentasi Belajar SQL (MySQL)

Tanggal mulai: 20 Januari 2026 – 25 Januari 2026

Source: <https://youtu.be/4ptA6e7N0xo?si=-sC3j-yI07Pcb7xi>

## Database

### Database

Langkah pertama dalam menyimpan informasi secara elektronik menggunakan SQL adalah membuat database. Pada bagian ini, kita akan belajar bagaimana **membuat (Create)**, **memilih (Select)**, **menghapus (Drop)**, dan **mengganti nama (Rename)** database dengan contoh.

- **CREATE Database** → Membuat database baru.
- **DROP Database** → Menghapus database yang sudah ada.
- **RENAME Database** → Mengubah nama database. *tidak bisa di mysql*
- **SELECT Database** → Memilih database yang akan digunakan.

Contoh Penggunaan:

The screenshot shows the MySQL Workbench interface. On the left is a 'Query Grid' window displaying the results of a 'show databases;' query, which lists several databases including 'information\_schema', 'mysql', 'performance\_schema', 'sys', and 'test'. On the right is a 'Command Prompt - mysql' window showing the MySQL monitor. The user has entered 'mysql -u root -p' and is prompted for a password. The MySQL version is 9.5.0. The monitor displays standard MySQL startup messages and help instructions. A syntax error occurs when the user tries to run 'show database;' again, resulting in an 'ERROR 1064' message.

```
Query 1 × Command Prompt - mysql × + ▾
File Edit View Insert Object Properties Help | Limit to 1000 rows | + | ▾
1 • show databases;
2

C:\Users\mrafl>mysql -u root -p
Enter password: *****
Welcome to the MySQL monitor. Commands end with ; or \
Your MySQL connection id is 22
Server version: 9.5.0 MySQL Community Server - GPL

Copyright (c) 2000, 2025, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation
affiliates. Other names may be trademarks of their resp
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the c

mysql> show database;
ERROR 1064 (42000): You have an error in your SQL syntax
for the right syntax to use near 'database' at line 1
mysql> show databases;
+-----+
| Database |
+-----+
| information_schema |
| mysql |
| performance_schema |
| sys |
+-----+
4 rows in set (0.024 sec)

mysql> |
```

Code	Fungsi	Hasilnya
Create database test;	Membuat database nama "test"	Database nama "test" terbuat.
Drop database test;	Untuk menghapus database "test"	Database "test" terhapus.

Create database if not exists test;	Untuk mengecek apakah database dengan nama "test" udah ada atau belum	Jika database nama "test" udah ada maka diberikan pesan eror "database exists".
Use test;	Untuk menggunakan database tertentu, seperti contoh database "test"	Database yang digunakan akan terbold menandakan sedang digunakan.
Alter database test read only = 1 (true) / 0 (false);	Untuk mengubah database "test" hanya read only. Tujuannya agar menghindari hal jika tidak sengaja melakukan <b>hapus</b> database gitu.	Membuat database "test" hanya read only.
Ctrl + /	Untuk command	-- use test;" command di MySQL.

## Data types

### Kategori Data Types dalam MySQL

- Numerik (Number)
- String (Character & Text)
- Tanggal & Waktu (Date & Time)
- Data Lainnya (JSON, BLOB, dll.)

### Numerik

Tipe	Keterangan	Contoh Nilai
INT	Bilangan bulat ( $\pm 2$ miliar)	123, -45
TINYINT	Bilangan kecil (0–255 atau -128–127)	0, 100
BIGINT	Bilangan bulat sangat besar	9223372036854775807
DECIMAL(x,y)	Angka presisi tinggi (misal uang)	DECIMAL(10,2) → 12345.67
FLOAT	Angka desimal presisi sedang	3.14, -0.01
DOUBLE	Angka desimal presisi lebih tinggi	1234567.890123
BOOLEAN / BOOL	Alias dari TINYINT(1)	0 (false), 1 (true)

## String

Tipe*	Keterangan	Contoh
CHAR(n)	Teks tetap panjang n karakter	'ID01' (fixed)
VARCHAR(n)	Teks fleksibel hingga n karakter	'halo dunia'
TEXT	Teks panjang (hingga 65.535 karakter)	Artikel, komentar
TINYTEXT, MEDIUMTEXT, LONGTEXT	Ukuran teks yang lebih besar	
ENUM	Pilihan terbatas (set seperti dropdown)	ENUM('aktif', 'non aktif')

## Tanggal & Waktu

Tipe	Keterangan	Contoh Nilai
DATE	Tanggal saja	'2025-04-19'
TIME	Waktu saja	'13:45:00'
DATETIME	Gabungan tanggal & waktu	'2025-04-19 13:45:00'
TIMESTAMP	Mirip DATETIME, disimpan sebagai Unix time, auto-updateable	'2025-04-19 13:45:00'
YEAR	Hanya tahun (4 digit)	'2025'

## Lain lain

Tipe	Keterangan
BLOB	Binary Large Object (file, gambar, dsb)
JSON	Menyimpan data dalam format JSON
SET	Menyimpan satu atau lebih nilai dari daftar tetap

# Table

## Table

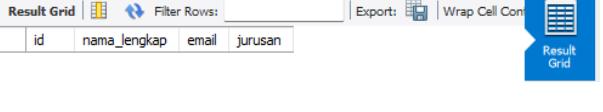
Pilar utama dari setiap database SQL adalah **tabel**. Secara dasar, struktur tabel sangat mirip dengan **spreadsheet**, di mana data disimpan dalam format grid yang terorganisir dengan baik.

Dalam bagian ini, kita akan belajar bagaimana **membuat (Create)**, **menghapus (Drop)**, **menghapus semua data (Truncate)**, **menyalin (Copy)**, **mengubah (Alter)**, dan **menggunakan tabel sementara (Temp Table)** dalam SQL.

- **CREATE TABLE** → Membuat tabel baru.
- **DROP TABLE** → Menghapus tabel dari database.
- **RENAME TABLE** → Mengubah nama tabel.
- **TRUNCATE TABLE** → Menghapus semua data dalam tabel tanpa menghapus strukturnya.
- **COPY TABLE** → Menyalin tabel beserta datanya.
- **TEMP TABLE** → Membuat tabel sementara yang hanya berlaku dalam sesi tertentu.
- **ALTER TABLE** → Mengubah struktur tabel, seperti menambah atau menghapus kolom.

Contoh penggunaan:

Code	Fungsi	Hasilnya
<pre>Query 1 x 1 • create table mahasiswa ( 2   id int, 3   nama varchar(20), 4   jurusan varchar(30) 5 ) 6</pre>	Membuat tabel “mahasiswa” dimana mencakup id, nama, dan jurusan.	Tabel mahasiswa terbuat.
<pre>7 • select * from mahasiswa</pre> 	Ini untuk melihat isi dari tabel “mahasiswa”, disitu kosong karena belum diisi data.	Memperlihatkan isi tabel “mahasiswa”.
Drop table mahasiswa;	Untuk menghapus table “mahasiswa”.	Table “mahasiswa” terhapus.
Truncate table mahasiswa;	Untuk menghapus isi data “mahasiswa”.	Isi data “mahasiswa” terhapus.
Create table mahasiswa_copy like mahasiswa;	Untuk membuat tabel “mahasiswa_copy” dimana hanya menyalin struktur dari tabel “mahasiswa”.	Tabel “mahasiswa_copy” terbuat dengan struktur tabel yang sama seperti tabel “mahasiswa”.
Create table mahasiswa_copy_2 as select * from mahasiswa;	Dimana membuat tabel “mahasiswa_copy_2”	Tabel “mahasiswa_copy_2” terbuat dengan

	" dimana struktur dan semua isi sama seperti tabel "mahasiswa".	struktur dan isi tabel yang sama seperti tabel "mahasiswa".
<pre> 9 • alter table mahasiswa add email varchar(50); 10 • select * from mahasiswa; 11 </pre> 	Untuk melakukan penambahan struktur baru dalam tabel "mahasiswa" misal disini tuh nambah "email".	Struktur tabel mahasiswa tertambah kolom "email".
Alter table mahasiswa drop column email;	Untuk melakukan penghapusan kolom tertentu pada tabel "mahasiswa".	Kolom "email" akan terhapus.
<pre> --  13 • alter table mahasiswa change nama nama_lengkap varchar(20); 14 • select * from mahasiswa; </pre> 	Untuk mengubah kolom "nama" ke "nama_lengkap".	Kolom "nama" berubah menjadi "nama_lengkap".
Alter table mahasiswa modify nama_lengkap varchar(150);	Untuk memodifikasi kolom tipe data.	Nama tabel tetap sama namun tipe data yang berubah.
Alter table mahasiswa modify jurusan varchar(30) first;	Untuk memindahkan kolom "jurusan" itu ke kolom pertama.	Struktur tabel: Jurusan Id Nama_lengkap
Alter table mahasiswa modify nama_lengkap varchar(100) after jurusan;	Untuk memindahkan kolom "nama_lengkap" itu ke kolom kedua setelah "jurusan".	Struktur tabel: Jurusan Nama_lengkap Id
rename table mahasiswa_copy to Rafli;	Untuk mengubah nama tabel "mahasiswa_copy" ke tabel nama "Rafli".	Dari tabel nama "mahasiswa_copy" ke tabel "Rafli".

<pre> 19 • Create temporary table temp_mahasiswa ( 20   Id int, 21   Nama varchar(10) 22 ); 23 24 • select * from temp_mahasiswa; 25 </pre> <p><b>Result Grid</b>   Filter Rows: <input type="text"/> Export:  Wrap Cell Content </p> <table border="1"> <thead> <tr> <th>Id</th> <th>Nama</th> </tr> </thead> <tbody> <tr> <td></td> <td></td> </tr> </tbody> </table>	Id	Nama			Untuk membuat tabel yang hanya sementara saja, dimana jika session sudah selesai (close) tabelnya hilang.	Terbuat tabel “temp_mahasiswa” yang hanya sementara.
Id	Nama					

## Insert rows

**Insert rows**

Perintah INSERT di MySQL digunakan untuk menambahkan baris (record) baru ke dalam sebuah tabel.

Sintaks : INSERT INTO nama\_tabel (kolom1, kolom2, ...) VALUES (nilai1, nilai2, ...);

**Note :**

- Pastikan data yang dimasukkan sesuai dengan tipe data kolomnya.
- Gunakan NULL untuk kolom AUTO\_INCREMENT atau kolom yang boleh kosong.
- Selalu gunakan nama kolom agar lebih aman saat struktur tabel berubah.

Cara penggunaan:

Code	Fungsi	Hasilnya												
<pre> --  34 • insert into siswa(id, nama, jurusan, nilai) values 35   (1,'Rafli','informatika',85), 36   (2,'Arifah','PSG',96); 37 38 • select * from siswa; </pre> <p><b>Result Grid</b>   Filter Rows: <input type="text"/> Export:  Wrap Cell Content </p> <table border="1"> <thead> <tr> <th>id</th> <th>nama</th> <th>jurusan</th> <th>nilai</th> </tr> </thead> <tbody> <tr> <td>1</td> <td>Rafli</td> <td>informatika</td> <td>85</td> </tr> <tr> <td>2</td> <td>Arifah</td> <td>PSG</td> <td>96</td> </tr> </tbody> </table>	id	nama	jurusan	nilai	1	Rafli	informatika	85	2	Arifah	PSG	96	Menambahkan data pada tabel “siswa”.	Data terisi pada tabel “siswa”.
id	nama	jurusan	nilai											
1	Rafli	informatika	85											
2	Arifah	PSG	96											
insert into siswa(id, nama, jurusan, nilai) values (3,'Budi','fisika',null);	Menambahkan data baru.	Data baru tertambah.												

## Select

### SELECT di MYSQL

Perintah SELECT digunakan untuk mengambil (menampilkan) data dari satu atau lebih tabel dalam database.

Sintaks : SELECT kolom1, kolom2, ... FROM nama\_tabel;

Untuk menampilkan semua kolom: SELECT \* FROM nama\_tabel;

Cara penggunaan:

Code	Fungsi	Hasilnya
<pre>39 40 • select nama, nilai from siswa;</pre>  <p>The screenshot shows the MySQL Workbench interface. In the query editor, line 40 contains the SQL command 'select nama, nilai from siswa;'. Below it, the results grid displays two rows: Rafli with a value of 85 and Arifah with a value of 96. The results grid has buttons for 'Result Grid' and 'Form'.</p>	Select data bagian tertentu saja.	Menampilkan data tertentu saja

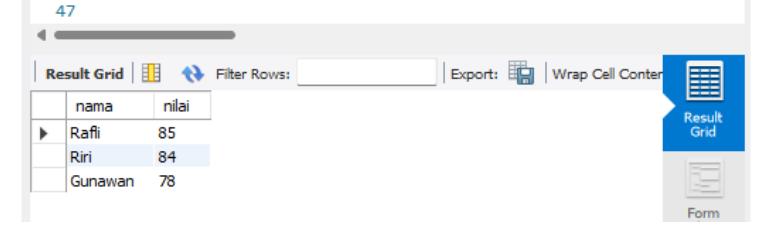
### SELECT dengan WHERE (Filter Data)

Kita bisa melakukan select data berdasarkan kondisi atau where kolom tertentu.

Contoh jika kita ingin mengambil data dari table siswa, dengan catatan data yang diambil adalah siswa yang mempunyai nilai diatas 90

Sintaks : SELECT \* FROM siswa WHERE nilai > 90;

Cara penggunaan:

Code	Fungsi	Hasilnya
<pre>46 • select nama, nilai from siswa where nilai &lt; 90; 47</pre>  <p>The screenshot shows the MySQL Workbench interface. In the query editor, line 46 contains the SQL command 'select nama, nilai from siswa where nilai &lt; 90;'. Below it, the results grid displays three rows: Rafli (nilai 85), Riri (nilai 84), and Gunawan (nilai 78). The results grid has buttons for 'Result Grid' and 'Form'.</p>	Select data siswa nama dan nilai ditampilkan dengan kondisi nilai < 90.	Menampilkan data dengan kondisi tertentu.

## Update

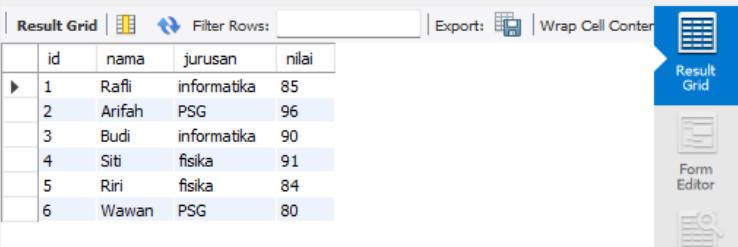
### UPDATE (Mengubah Data)

Perintah UPDATE digunakan untuk mengubah nilai pada satu atau beberapa kolom di baris tertentu dalam tabel.

UPDATE nama\_tabel SET kolom1 = nilai1, kolom2 = nilai2, ... WHERE kondisi

Note : Selalu gunakan WHERE untuk mencegah update semua baris secara tidak sengaja.

Cara penggunaan:

Code	Fungsi	Hasilnya
<pre>48 •      set SQL_SAFE_UPDATES=0; 49 50 •      update siswa set nilai = 80, nama = 'Wawan' where id=6; 51 52 •      select * from siswa; 53</pre> 	Mengubah isi data tertentu	Menampilkan update data dengan tertentu.

## Primary key

### PRIMARY KEY

**PRIMARY KEY** adalah **kolom** (atau gabungan beberapa kolom) dalam tabel database yang:

- **Unik** (tidak boleh ada nilai yang sama)
- **Tidak boleh kosong** (NOT NULL)
- Digunakan untuk **mengidentifikasi setiap baris** secara **spesifik**.

## PRIMARY KEY

### Aturan PRIMARY KEY:

- Nilai **harus unik** (tidak boleh ada duplikat).
- Nilai **tidak boleh NULL**.
- Dalam satu tabel, hanya boleh ada **satu PRIMARY KEY**.
- PRIMARY KEY bisa terdiri dari:
  - **Satu kolom** → (contoh: `id`)
  - **Gabungan beberapa kolom** → disebut **Composite Primary Key**.

## PRIMARY KEY

```
CREATE TABLE siswa (
    id INT PRIMARY KEY,
    nama VARCHAR(100)
);
```

```
CREATE TABLE siswa (
    id INT,
    nama VARCHAR(100),
    PRIMARY KEY (id)
);
```

```
CREATE TABLE siswa (
    id INT,
    nama VARCHAR(30),
    jurusan VARCHAR(10),
    nilai INT,
    PRIMARY KEY (id, nama)
);
```

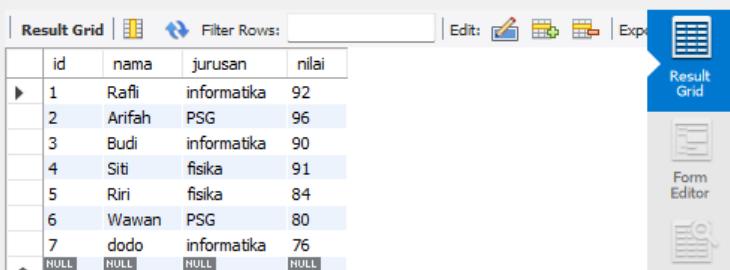
Artinya: kombinasi id dan nama harus unik.

## PRIMARY KEY

```
ALTER TABLE siswa
ADD PRIMARY KEY (id);
```

```
ALTER TABLE siswa
MODIFY COLUMN id INT NOT NULL AUTO_INCREMENT,
ADD PRIMARY KEY (id);
```

Cara penggunaan:

Code	Fungsi	Hasilnya
<pre>-- 54 • alter table siswa modify id int auto_increment, 55     add primary key (id); 56 57 • update siswa set nilai = 92 where id=1;</pre> 	Mengubah id menjadi primary key, efeknya Ketika melakukan update sudah tidak ada pesan ERROR.	Langsung bisa mengubah isi data tanpa pesan ERROR.

## Delete

### DELETE (Menghapus Data)

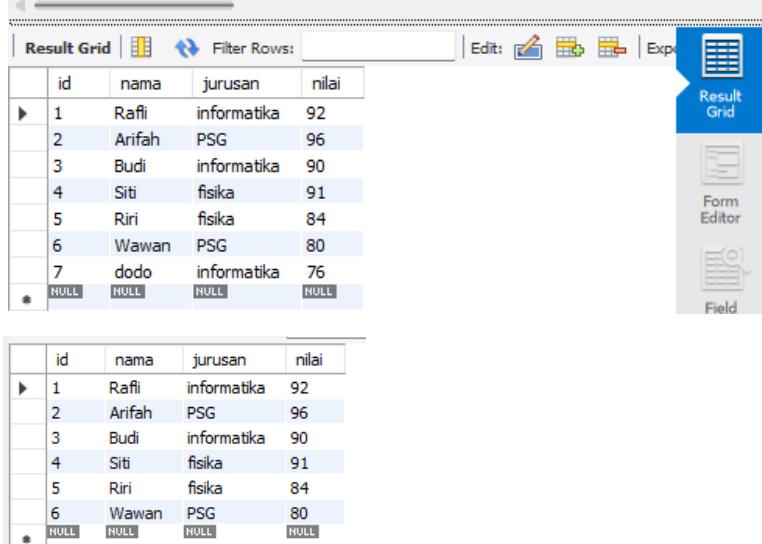
Perintah DELETE digunakan untuk menghapus baris dari sebuah tabel berdasarkan kondisi tertentu.

Sintaks Umum : DELETE FROM nama\_tabel WHERE kondisi;

Note : Tanpa WHERE, semua data di tabel bisa terhapus!

C

Cara penggunaan:

Code	Fungsi	Hasilnya																																													
<pre>62 • delete from siswa where id=7; 63 64 • select * from siswa; 65</pre>  <table border="1"><thead><tr><th></th><th>id</th><th>nama</th><th>jurusan</th><th>nilai</th></tr></thead><tbody><tr><td>▶</td><td>1</td><td>Rafli</td><td>informatika</td><td>92</td></tr><tr><td></td><td>2</td><td>Arifah</td><td>PSG</td><td>96</td></tr><tr><td></td><td>3</td><td>Budi</td><td>informatika</td><td>90</td></tr><tr><td></td><td>4</td><td>Siti</td><td>fisika</td><td>91</td></tr><tr><td></td><td>5</td><td>Riri</td><td>fisika</td><td>84</td></tr><tr><td></td><td>6</td><td>Wawan</td><td>PSG</td><td>80</td></tr><tr><td>*</td><td>7</td><td>dodo</td><td>informatika</td><td>76</td></tr><tr><td></td><td>* NULL</td><td>NULL</td><td>NULL</td><td>NULL</td></tr></tbody></table>		id	nama	jurusan	nilai	▶	1	Rafli	informatika	92		2	Arifah	PSG	96		3	Budi	informatika	90		4	Siti	fisika	91		5	Riri	fisika	84		6	Wawan	PSG	80	*	7	dodo	informatika	76		* NULL	NULL	NULL	NULL	Untuk menghapus baris tertentu.	Data yang dipilih akan terhapus.
	id	nama	jurusan	nilai																																											
▶	1	Rafli	informatika	92																																											
	2	Arifah	PSG	96																																											
	3	Budi	informatika	90																																											
	4	Siti	fisika	91																																											
	5	Riri	fisika	84																																											
	6	Wawan	PSG	80																																											
*	7	dodo	informatika	76																																											
	* NULL	NULL	NULL	NULL																																											

## Aliases

### Aliases

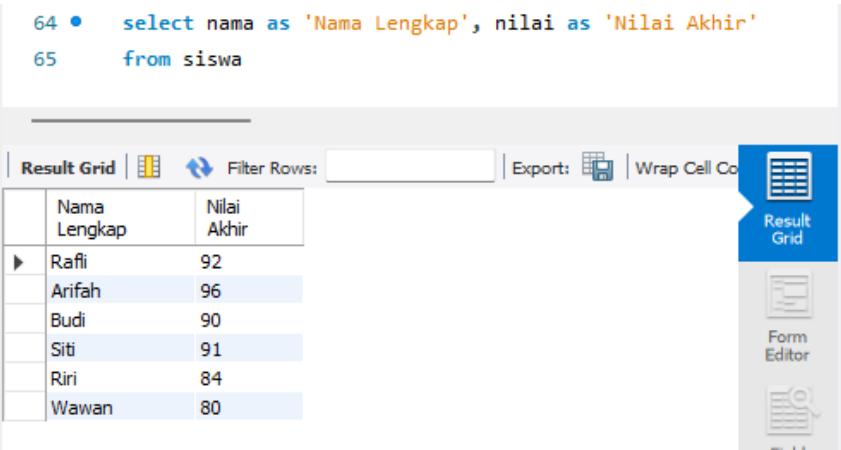
Alias dalam MySQL adalah **nama lain** (sementara) yang diberikan untuk kolom atau tabel, untuk membuat hasil query lebih mudah dibaca atau untuk mempersingkat nama yang panjang.

Alias dibuat menggunakan kata kunci **AS**, atau bahkan bisa tanpa AS (langsung beri nama setelah kolom atau tabel).

Jika alias mengandung **spasi**, gunakan tanda kutip atau backticks

```
SELECT nama AS 'Nama Siswa', nilai AS 'Nilai Akhir'
FROM siswa;
```

Cara penggunaan:

Code	Fungsi	Hasilnya
<pre>64 •     select nama as 'Nama Lengkap', nilai as 'Nilai Akhir' 65      from siswa</pre> 	Untuk mengubah kolom tertentu.	Hasilnya berubah.

## SQL Operator

### SQL Operator

**SQL Operators** adalah simbol atau kata kunci dalam SQL yang digunakan untuk melakukan berbagai operasi pada data dalam database. Operator ini memungkinkan kita untuk membandingkan nilai, menggabungkan kondisi, mencari pola, dan banyak lagi.

Di MySQL, operator-operator ini sangat penting saat menulis query untuk **memfilter**, **mengelompokkan**, **menggabungkan**, atau **mengolah** data.

#### 1. AND Operator

**Fungsi:** Menggabungkan dua kondisi atau lebih dan hanya mengembalikan hasil yang memenuhi semua kondisi.

Contoh jika kita ingin melihat data siswa yang berjurusan 'Fisika' dengan nilai diatas 90

```
SELECT * FROM siswa  
WHERE jurusan = 'Fisika' AND nilai > 90;
```

## 2. OR Operator

**Fungsi:** Menggabungkan dua kondisi atau lebih dan mengembalikan hasil jika salah satu kondisi terpenuhi.

Contoh jika kita ingin melihat data siswa yang berjurusan 'Fisika' atau siswa yang bernilai diatas 90

```
SELECT * FROM siswa  
WHERE jurusan = 'Fisika' OR nilai > 90;
```

## 3. NOT Operator

**Fungsi:** Membalikkan hasil dari kondisi (menghasilkan TRUE jika kondisi bernilai FALSE, dan sebaliknya).

Contoh jika kita ingin melihat semua data siswa yang tidak berjurusan 'Fisika'

```
SELECT * FROM siswa  
WHERE NOT jurusan = 'Fisika';
```

```
SELECT * FROM siswa  
WHERE jurusan <> 'Fisika';
```

```
SELECT * FROM siswa  
WHERE jurusan != 'Fisika';
```

## 4. LIKE Operator

**Fungsi:** Digunakan untuk pencarian berdasarkan pola pada string.

Wildcard yang sering digunakan:

- % = mewakili nol atau lebih karakter.
- \_ = mewakili tepat satu karakter.

Cari siswa yang **nama-nya** mengandung kata "an" di mana saja.

```
SELECT * FROM siswa  
WHERE nama LIKE '%an%';
```

Cari semua siswa yang **nama-nya** lima huruf dan huruf kedua-nya "a".

```
SELECT * FROM siswa  
WHERE nama LIKE '_a___';
```

## 5. IN Operator

Operator IN digunakan untuk menentukan apakah suatu nilai terdapat dalam daftar nilai.

```
SELECT * FROM siswa  
WHERE jurusan IN ('Fisika', 'Matematika');
```

## 6. IS NULL Operator

Operator IS NULL digunakan untuk mengecek apakah kolom memiliki nilai kosong (NULL).

```
SELECT * FROM siswa  
WHERE nilai IS NULL;
```

## 7. BETWEEN Operator

**Fungsi:** Memilih nilai dalam rentang tertentu (inklusif: termasuk nilai batas bawah dan atas).

Contoh : Tampilkan data semua siswa yang memiliki nilai di rentang 87 - 100

```
SELECT * FROM siswa  
WHERE nilai BETWEEN 87 AND 100;
```

## 8. UNION dan UNION ALL Operator

**Fungsi:** Menggabungkan hasil dari dua atau lebih query SELECT dan menghilangkan duplikat secara otomatis.

```
SELECT nama FROM siswa  
UNION  
SELECT nama FROM mahasiswa;
```

**Fungsi:** Sama seperti UNION, tetapi **tidak** menghapus data duplikat.

```
SELECT nama FROM siswa  
UNION ALL  
SELECT nama FROM mahasiswa;
```

## Order By

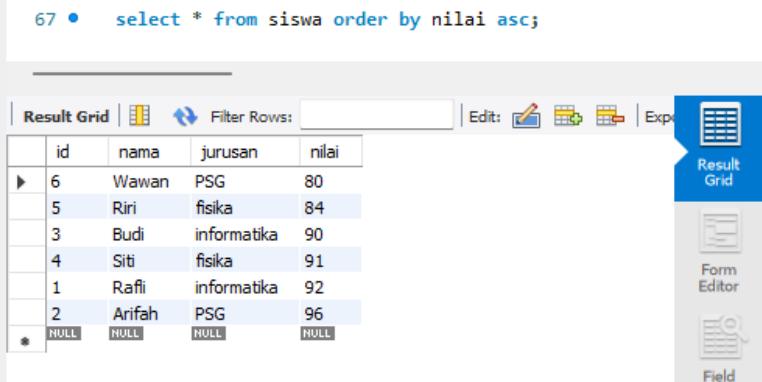
### ORDER BY

ORDER BY digunakan dalam perintah SELECT untuk **mengurutkan hasil query** berdasarkan satu atau beberapa kolom.

```
SELECT nama_kolom1, nama_kolom2, ...
FROM nama_tabel
ORDER BY nama_kolom [ASC|DESC];
```

- ASC (Ascending) = Dari Kecil ke Besar
- DESC (Descending) = Dari Besar ke Kecil

Cara penggunaan:

Code	Fungsi	Hasilnya
<pre>67 • select * from siswa order by nilai asc;</pre> 	Untuk mengurutkan nilai dari kecil ke besar (ascending) atau dari besar ke kecil (descending).	Akan mengurutkan nilainya sesuai dengan nilai kecil ke terbesar.

## Limit

### LIMIT

LIMIT digunakan untuk **membatasi jumlah baris** (row) yang ditampilkan oleh hasil query.

```
SELECT kolom1, kolom2, ...
FROM nama_tabel
LIMIT jumlah_baris;
```

Cara penggunaan:

Code	Fungsi	Hasilnya
<pre>69 • select * from siswa limit 2;</pre>	Untuk menampilkan data dengan menentukan jumlah baris tertentu.	Akan menampilkan data dengan jumlah baris tertentu
<pre>71 • select * from siswa order by nilai asc limit 3;</pre>	Untuk menampilkan data dengan menggunakan order by kondisi tertentu.	Menampilkan data dengan jumlah baris yang di limitkan contohnya 3 baris saja dari nilai terkecil ke terbesar.

## Offset

### OFFSET

OFFSET digunakan bersama LIMIT untuk **melewati sejumlah baris pertama** dari hasil query sebelum menampilkan data.

```
SELECT * FROM nama_tabel  
LIMIT jumlah_baris OFFSET offset;
```

```
SELECT * FROM nama_tabel  
LIMIT offset, jumlah_baris;
```

Cara penggunaan:

Code	Fungsi	Hasilnya
<pre>73 • select * from siswa limit 2 offset 4;</pre>	Untuk melakukan limit data yang ditampilkan dan offset data keberapa.	Akan menampilkan data sesuai dengan yang ingin ditampilkan.

<pre>73 • select * from siswa limit 4,2;</pre>	<p>Fungsinya sama saja namun penempatan offsite nya di depan koma dan baru barusnya.</p>	<p>Akan menampilkan data sesuai dengan yang ingin di tampilkan.</p>
--	--	---

## Select Distinct

### Select Distinct

- SELECT DISTINCT digunakan untuk **menghilangkan duplikat** dari hasil query dan hanya menampilkan nilai yang **unik/satu kali saja** dari kolom tertentu.

Contoh kita ingin menampilkan semua jurusan yang ada pada tabel siswa

```
SELECT DISTINCT jurusan
FROM siswa;
```

Cara penggunaan:

Code	Fungsi	Hasilnya
<pre>73 • select distinct jurusan from siswa;</pre>	<p>Untuk melihat data tanpa duplikat.</p>	<p>Menampilkan seluruh data yang duplikat akan di anggap satu aja.</p>

## Aggregate Function

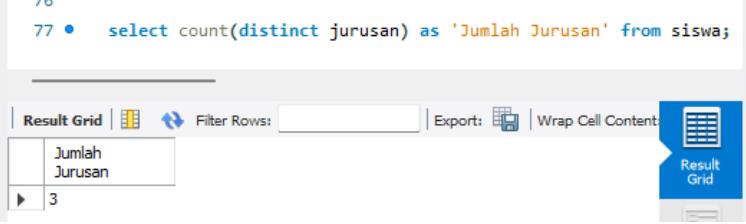
### Aggregate Function

Aggregate function adalah **fungsi yang melakukan perhitungan terhadap sekumpulan baris**, dan **mengembalikan satu nilai hasil**. Fungsi ini sangat berguna saat digabung dengan GROUP BY atau untuk menganalisis data.

## Beberapa aggregate function di mysql

Fungsi	Keterangan
COUNT()	Menghitung jumlah baris
SUM()	Menjumlahkan nilai
AVG()	Menghitung rata-rata
MIN()	Mencari nilai terkecil
MAX()	Mencari nilai terbesar

Cara penggunaan:

Code	Fungsi	Hasilnya
<pre>75 •  select count(*) as 'Jumlah data' from siswa; 76 77 •  select count(distinct jurusan) as 'Jumlah Jurusan' from siswa;</pre> 	Untuk melakukan perhitungan.	Akan menampilkan total suatu nilai data

## Group by

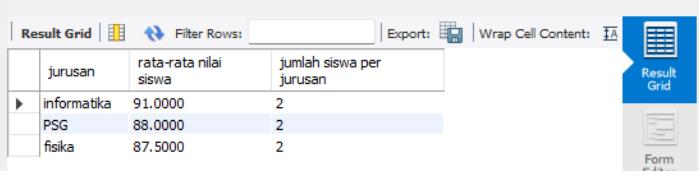
### Group by

- GROUP BY digunakan untuk **mengelompokkan data berdasarkan nilai kolom tertentu**, lalu **melakukan operasi agregat** (seperti SUM( ), AVG( ), COUNT( ), dll.) pada setiap grup.

Sintaks Dasar :

```
SELECT kolom, fungsi_agregat(kolom)
FROM nama_tabel
GROUP BY kolom;
```

Cara penggunaan:

Code	Fungsi	Hasilnya
<pre>79 •     select jurusan, avg(nilai) as 'rata-rata nilai siswa' 80      from siswa group by jurusan; 81 82 •     select jurusan, avg(nilai) as 'rata-rata nilai siswa', 83      count(*) as 'jumlah siswa per jurusan' 84      from siswa group by jurusan;</pre> 	Untuk melakukan pengelompokan data berdasarkan nilai kolom tertentu.	Menampilkan hasil sesuai dengan agregat yang diperintahkan

## Having

### Having

HAVING digunakan untuk menyaring data setelah proses agregasi dan (GROUP BY) selesai.

Berbeda dengan WHERE yang menyaring sebelum data dikelompokkan.

### Having Sintaks

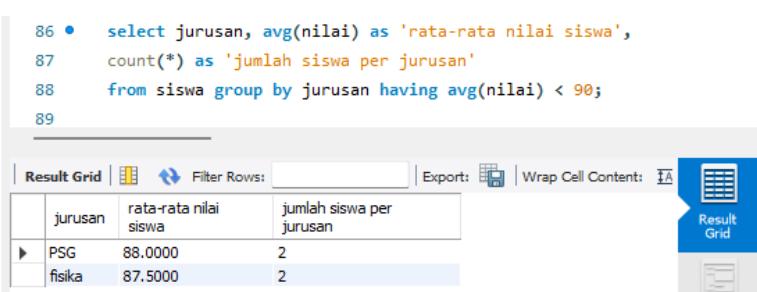
```
•
SELECT kolom1, fungsi_agregat(kolom2)
FROM nama_tabel
GROUP BY kolom1
HAVING kondisi_agregat;
```

```
SELECT kolom1, fungsi_agregat(kolom2)
FROM nama_tabel
WHERE kondisi_awal
GROUP BY kolom1
HAVING kondisi_agregat;
```

Penjelasan sintaks :

- SELECT → memilih kolom yang ingin ditampilkan.
- fungsi\_agregat → seperti SUM(), COUNT(), AVG(), MIN(), MAX().
- WHERE (opsional) → menyaring baris sebelum pengelompokan.
- GROUP BY → mengelompokkan data berdasarkan satu kolom atau lebih.
- HAVING → menyaring hasil kelompok berdasarkan fungsi agregat.

Cara penggunaan:

Code	Fungsi	Hasilnya									
<pre>86 •     select jurusan, avg(nilai) as 'rata-rata nilai siswa', 87      count(*) as 'jumlah siswa per jurusan' 88      from siswa group by jurusan having avg(nilai) &lt; 90; 89</pre>  <table border="1"><thead><tr><th>jurusan</th><th>rata-rata nilai siswa</th><th>jumlah siswa per jurusan</th></tr></thead><tbody><tr><td>PSG</td><td>88.0000</td><td>2</td></tr><tr><td>fisika</td><td>87.5000</td><td>2</td></tr></tbody></table>	jurusan	rata-rata nilai siswa	jumlah siswa per jurusan	PSG	88.0000	2	fisika	87.5000	2	Sama seperti where namun ini tuh setelah proses agregasi.	Menampilkan filter sesuai dengan having.
jurusan	rata-rata nilai siswa	jumlah siswa per jurusan									
PSG	88.0000	2									
fisika	87.5000	2									

## Data Constraints

### Data Constraints dalam MySQL

**Data Constraints** adalah aturan atau kondisi yang diterapkan pada data di dalam tabel. Tujuan utama dari constraints ini adalah untuk **menjaga kualitas, konsistensi, dan integritas data** dalam basis data, serta mencegah terjadinya kesalahan saat memasukkan atau memodifikasi data.

### Jenis Data Constraints dalam MYSQL

- NOT NULL
- DEFAULT
- UNIQUE
- CHECK
- PRIMARY KEY, COMPOSITE KEY,
- ALTERNATE KEY
- FOREIGN KEY

### NOT NULL Constraint

**Fungsi:** Mencegah kolom memiliki nilai NULL.

**Kapan Digunakan:** Jika sebuah kolom wajib diisi, misalnya nama atau email.

### DEFAULT Constraint

**Fungsi:** Memberikan nilai **bawaan (default)** ke kolom jika tidak diisi secara eksplisit saat insert.

## UNIQUE Constraint

**Fungsi:** Memastikan bahwa nilai dalam kolom tidak ada yang **duplicat**.

**Contoh Kasus:** Kolom email yang harus unik untuk setiap pengguna.

## CHECK Constraint

**Fungsi:** Membatasi nilai yang diperbolehkan dalam kolom sesuai dengan kondisi tertentu.

## Contoh

```
CREATE TABLE user (
    id INT AUTO_INCREMENT PRIMARY KEY,
    -- NOT NULL: nama tidak boleh kosong
    name VARCHAR(100) NOT NULL,
    -- UNIQUE: email harus unik
    email VARCHAR(100) NOT NULL UNIQUE,
    -- DEFAULT: status default adalah 'active'
    status VARCHAR(20) NOT NULL DEFAULT 'active',
    -- CHECK: umur minimal 18
    age INT CHECK (age >= 18)
);
```

Cara penggunaan:

Code	Fungsi	Hasilnya
<pre>90 • ④ create table users( 91     id int auto_increment primary key, 92     name varchar(100) not null, 93     email varchar(100) not null unique, 94     status varchar(20) default 'active', 95     age int check(age &gt;= 18) 96 ); 97 98 -- ini yang benar: 99 • insert into users(name, email, status, age) values 100 ('budi','budi@gmail.com', 'inactive', 22); 101 102 -- ini yang akan error 103 • insert into users(name, email, status, age) values 104 ('null','budi@gmail.com', 'inactive', 19);</pre>	Jadi code yang create tabel itu merupakan constraint (aturan) yang dibuat untuk tabel.  Ketika kita memasukkan sebuah data dan tidak sesuai dengan constraint pada tabel maka akan muncul error	Akan terjadi error pada saat memasukkan data Ketika data yang dimasukkan tidak sesuai dengan constraint yang kita buat awal pada pembuatan tabel.

## Tabel Relationship

### Table relationship

Dalam basis data relasional, data disimpan dalam bentuk tabel. Untuk menghubungkan antar tabel agar bisa saling berelasi dan tidak duplikasi data, kita menggunakan **relasi tabel** (table relationships).

Untuk **menghubungkan tabel** dalam database relasional (seperti MySQL, PostgreSQL, SQLite, dsb), kita menggunakan **foreign key** sebagai penghubung antar tabel.

### Apa Itu Foreign Key?

Foreign key adalah **kolom dalam satu** tabel yang merujuk ke **primary key** di tabel lain. Ini adalah cara untuk menciptakan relasi antar tabel.

### Menghubungkan table

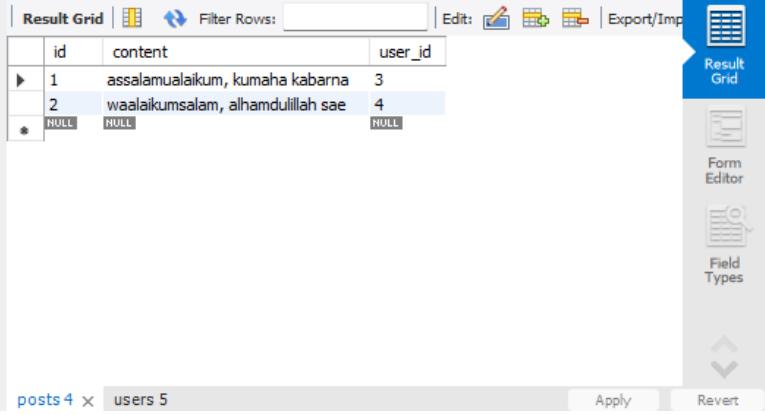
Jadi sekarang kita akan coba menghubungkan 2 table contoh table posts (children) akan kita hubungkan dengan table users (parents)

## Menghubungkan table

```
CREATE TABLE user (
    id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(100) NOT NULL,
    email VARCHAR(100) NOT NULL UNIQUE,
    status VARCHAR(20) NOT NULL DEFAULT 'active',
    age INT CHECK (age >= 18)
);
```

```
CREATE TABLE posts (
    id INT PRIMARY KEY,
    content TEXT,
    user_id INT,
    FOREIGN KEY (user_id) REFERENCES users(id)
);
```

Cara penggunaan:

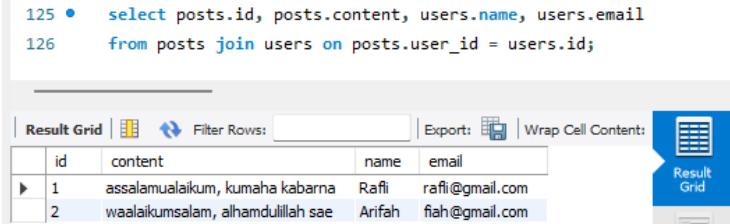
Code	Fungsi	Hasilnya
<pre>111 • Ⓜ create table posts( 112     id int auto_increment primary key, 113     content text not null, 114     user_id int, 115     foreign key (user_id) references users(id) 116 ); 117 118 •   insert into posts(content, user_id) values 119     ('assalamualaikum, kumaha kabarna', 3), 120     ('waalaikumsalam, alhamdulillah sae', 4); 121 122 •   select * from posts; 123 •   select * from users;</pre> 	Ini melakukan pembuatan tabel children (posts) dimana dibuat terhubung dengan tabel parents (users).	Tabel children (posts) terbuat dan terhubung dengan tabel parents (users).

## Mengambil data gabungan ( join )

Setelah table terhubung dan kita telah melakukan penambahan data atau insert. Sekarang kita bisa menggunakan join untuk mengambil data gabungan dari posts dan users tadi, sehingga sewaktu kita melakukan select pada data posts kita bisa tahu data asli user yang membuat post tersebut

```
SELECT posts.id, posts.content, users.name  
FROM posts  
JOIN users ON posts.user_id = users.id;
```

Cara penggunaan:

Code	Fungsi	Hasilnya
<pre>125 •   select posts.id, posts.content, users.name, users.email 126     from posts join users on posts.user_id = users.id;</pre> 	Melakukan penggabungan kolom dari tabel users dan posts.	Menghasilkan tabel baru yang dimana kolom berasal dari tabel children (posts) dan tabel parents (users)

## Menghapus atau mengubah foreign key

Terkadang kita salah dalam memberikan foreign key ke column pada tabel yang kita buat.

Tentunya kita dapat mengubah atau menghapus foreign key tersebut.

Setelah kita membuat foreign key, foreign key tersebut akan mendapatkan nama constraint atau label. Lalu kita bisa drop label tersebut

Kita bisa check nama dari foreign key dengan sintaks

```
SHOW CREATE TABLE posts;
```

## Menghapus atau mengubah foreign key

Setelah mengetahui nama (misal: `posts_ibfk_1`) foreign key kita bisa hapus dengan sintaks.

```
ALTER TABLE posts DROP FOREIGN KEY posts_ibfk_1;
```

Untuk menambahkan kembali foreign key ke tabel yang sudah ada kita dapat menggunakan sintaks.

```
ALTER TABLE posts
ADD CONSTRAINT fk_user_id
FOREIGN KEY (user_id) REFERENCES users(id);
```

## Memberikan nama constraint di awal

```
CREATE TABLE posts (
    id INT PRIMARY KEY, -- Kolom id sebagai primary key (unik, tidak null)
    content TEXT, -- Kolom untuk isi post
    user_id INT, -- Kolom foreign key yang menghubungkan ke users(id)
    CONSTRAINT fk_user_id -- Memberi nama constraint foreign key (opsional tapi
    FOREIGN KEY (user_id) -- Foreign key merujuk ke kolom id di tabel users
    REFERENCES users(id)
);
```

Keuntungan memberi nama constraint pada foreign key adalah mempermudah menghapus dan juga mengubah constraint nanti

Cara penggunaan:

Code	Fungsi	Hasilnya
<pre>128 •  show create table posts; 129 130 •  alter table posts drop foreign key posts_ibfk_1;</pre> 	Untuk melakukan penghapusan foreign key pada tabel posts.	Foreign key terhapus pada tabel posts.

<pre>132 • alter table posts add constraint fk_users_id 133      foreign key (user_id) references users(id);</pre> <p>Form Editor   Navigate: ⟲ ⟳ ⏴ ⏵ ⏶ ⏷  </p> <p>posts Table:</p> <pre>CREATE TABLE `posts` (   `id` int NOT NULL AUTO_INCREMENT,   `content` text NOT NULL,   `user_id` int DEFAULT NULL,   PRIMARY KEY (`id`),   KEY `fk_users_id` (`user_id`),   CONSTRAINT `fk_users_id` FOREIGN KEY (`user_id`)     REFERENCES `users` (`id`) ) ENGINE=InnoDB AUTO_INCREMENT=3 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci</pre> <p>Create Table: ⌂</p>	<p>Untuk menambahkan kembali foreign key dari tabel posts.</p>	<p>Foreign key berhasil ditambahkan pada tabel posts.</p>
---	--	---

## Foreign key option

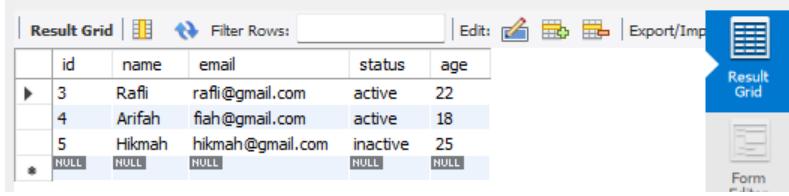
- Ketika kamu membuat sebuah foreign key di MySQL, kamu bisa menentukan **aksi** yang terjadi jika data diubah atau dihapus di tabel referensi. Aksi ini ditentukan dengan opsi berikut:

```
ON DELETE <action>
ON UPDATE <action>
```

### Opsi-opsi yang tersedia:

Opsi	Penjelasan
CASCADE	Jika data diubah atau dihapus di tabel referensi, maka data terkait ikut diubah atau dihapus.
SET NULL	Jika data diubah atau dihapus di tabel referensi, maka foreign key di-set ke NULL.
SET DEFAULT	Jika data diubah atau dihapus, nilai foreign key di-set ke nilai default (jarang digunakan di MySQL).
RESTRICT	Melarang penghapusan atau pengubahan jika ada data yang berelasi. (default di MySQL)
NO ACTION	Sama seperti RESTRICT, tapi periksa dilakukan setelah semua constraint dicek. Di MySQL, NO ACTION = RESTRICT.

Cara penggunaan:

Code	Fungsi	Hasilnya
<pre>132 • alter table posts drop foreign key fk_users_id; 133 134 • alter table posts add constraint fk_users_id 135     foreign key (user_id) 136     references users(id) 137     on delete cascade 138     on update cascade; 139 140 • delete from users where id = 1;</pre> 	Untuk melakukan aksi yang terjadi jika data di ubah atau dihapus. Di tentukan dari action (opsi).	Aksi jika data di update atau delete akan berpengaruh dari action (opsi) yang ditentukan sebelumnya.

## Jenis-jenis tabel Relationship

### Jenis-Jenis Table Relationship di MySQL

Dalam database relasional seperti MySQL, **relationship** antar tabel digunakan untuk menghubungkan data di berbagai tabel. Ada **tiga jenis utama**:

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

### One-to-One (1:1)

Setiap baris di Tabel A **hanya** berelasi dengan **satu baris** di Tabel B, dan sebaliknya.

**Contoh:**

- Setiap user hanya punya satu profile.

## One-to-One (1:1)

```
CREATE TABLE users (
    id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(100) NOT NULL,
    email VARCHAR(100) NOT NULL UNIQUE,
    status VARCHAR(20) DEFAULT 'active',
    age INT,
    CHECK (age >= 18)
);
```

```
CREATE TABLE profiles (
    id INT AUTO_INCREMENT PRIMARY KEY,
    user_id INT NOT NULL UNIQUE,
    bio TEXT,
    CONSTRAINT fk_user_id FOREIGN KEY (user_id)
        REFERENCES users(id)
        ON DELETE CASCADE
);
```

## One-to-Many (1:N)

Satu baris di Tabel A bisa berelasi dengan **banyak baris** di Tabel B, tetapi satu baris di Tabel B hanya milik **satu baris** di Tabel A.

### 💡 Contoh:

- Satu users bisa menulis banyak posts.
- Satu customer bisa punya banyak orders.

```
create table profiles (
    id int auto_increment primary key,
    bio text,
    user_id int,
    constraint fk_user_id foreign key (user_id) references
    users(id)
);
```

Kalau hubungan many to many pada pembuatan tabel baru tidak usah menggunakan **UNIQUE** seperti yang dilakukan pada one to one.

## Many-to-Many (N:N)

Satu baris di Tabel A bisa berelasi dengan **banyak baris** di Tabel B, dan sebaliknya.

### 💡 Contoh:

- Satu student bisa mengambil banyak courses.
- Satu course bisa diikuti oleh banyak students.

## Many-to-Many (N:N)

```
CREATE TABLE students (
    id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(100)
);

CREATE TABLE courses (
    id INT AUTO_INCREMENT PRIMARY KEY,
    title VARCHAR(100)
);

CREATE TABLE course_student (
    student_id INT,
    course_id INT,
    PRIMARY KEY (student_id, course_id),
    FOREIGN KEY (student_id) REFERENCES students(id)
        ON DELETE CASCADE,
    FOREIGN KEY (course_id) REFERENCES courses(id)
        ON DELETE CASCADE
);
```

Cara penggunaan:

Code	Fungsi	Hasilnya																					
<pre>192      -- student as s 193      -- course as c 194      -- course_student as cs 195 •   select s.name, c.title from student as s 196      join course_student as cs on s.id = cs.student_id 197      join courses as c on cs.course_id = c.id; 198</pre>  <table border="1"><thead><tr><th></th><th>name</th><th>title</th></tr></thead><tbody><tr><td>▶</td><td>Rafli</td><td>matematika</td></tr><tr><td></td><td>Rafli</td><td> fisika</td></tr><tr><td></td><td>Arifah</td><td>bahasa</td></tr><tr><td></td><td>suri</td><td>matematika</td></tr><tr><td></td><td>suri</td><td> fisika</td></tr><tr><td></td><td>suri</td><td>bahasa</td></tr></tbody></table>		name	title	▶	Rafli	matematika		Rafli	fisika		Arifah	bahasa		suri	matematika		suri	fisika		suri	bahasa	Untuk melakukan join many to many dengan menggunakan metode join sederhana.	3 tabel saling terhubung dengan menggunakan metode join sederhana
	name	title																					
▶	Rafli	matematika																					
	Rafli	fisika																					
	Arifah	bahasa																					
	suri	matematika																					
	suri	fisika																					
	suri	bahasa																					

## Join

JOIN

JOIN adalah perintah dalam SQL untuk menggabungkan data dari dua atau lebih tabel berdasarkan kolom yang memiliki relasi (biasanya foreign key dan primary key).

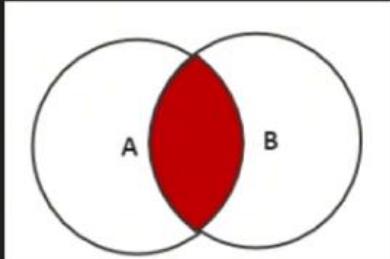
Tadi sewaktu di materi table relationship kita sudah sempat membahas join secara sederhana dan mencobakan langsung penggunaannya dari relasi tabel kita.

## Jenis jenis JOIN

- Inner Join
- Left Join
- Left Outer Join
- Right Join
- Right Outer Join
- Full Join

### Inner JOIN

**Menampilkan:** Hanya data yang cocok di kedua tabel.



```
SELECT <columns>
FROM TableA
INNER JOIN TableB ON A.Key = B.Key;
```

Contoh Penggunaan:

```
199    -- inner join
200 •  select p.id, p.content, u.name, u.email
201    from posts as p
202    inner join users as u
203    on p.user_id = u.id;
204
```

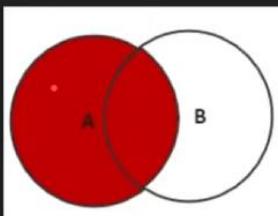
Result Grid | Filter Rows:  Export Wrap Cell Content:

Result Grid

	id	content	name	email
▶	2	waalaikumsalam, alhamdulillah sae	Arifah	fiah@gmail.com
	3	assalamualaikum, kumaha kabarna	Rafli	raffi@gmail.com

## Left JOIN

- Menampilkan semua data dari **Table A**.
- Jika ada kecocokan di **Table B**, data dari B ditampilkan.
- Jika tidak cocok, nilai dari B akan menjadi NULL.



```
SELECT <columns>
FROM TableA
LEFT JOIN TableB ON A.Key = B.Key;
```

Contoh penggunaan:

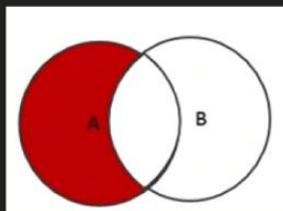
```
205    -- left join
206 •  select * from posts
207      left join users on posts.user_id = users.id;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content: | Result Grid

	id	content	user_id	id	name	email
▶	2	waalaikumsalam, alhamdulillah sae	4	4	Arifah	fiah@gmail.com
	3	assalamualaikum, kumaha kabarna	3	3	Rafli	raffi@gmail.com

## Left Outer JOIN

- Menampilkan **hanya** data dari Table A **yang tidak punya pasangan** di Table B.



```
SELECT <columns>
FROM TableA
LEFT JOIN TableB ON A.Key = B.Key
WHERE B.Key IS NULL;
```

Contoh penggunaan:

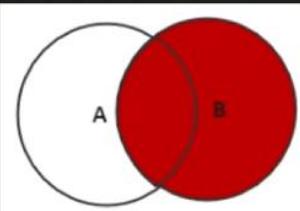
```
209    -- left outer join
210 •  select * from posts left join
211      users on posts.user_id = users.id where users.id is null;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content: | Result Grid

	id	content	user_id	id	name	email	status	age
▶	2	waalaikumsalam, alhamdulillah sae	4	4	Arifah	fiah@gmail.com	Active	25

## Right JOIN

- Kebalikan dari LEFT JOIN.
- Menampilkan semua data dari **Table B**.
- Jika ada kecocokan di A, akan ditampilkan.
- Jika tidak, nilai dari A menjadi NULL.



```
• SELECT <columns>
  FROM TableA
  RIGHT JOIN TableB ON A.Key = B.Key;
```

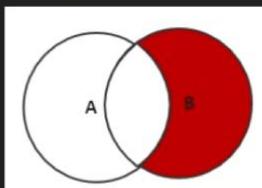
Contoh penggunaan:

```
213    -- right join
214 •  select * from posts
215    right join users on posts.user_id = users.id;
216
```

Result Grid		Filter Rows:	Export:	Wrap Cell Content:					
		id	content	user_id	id	name	email	status	age
	▶	NULL	NULL	NULL	1	Alice	alice@gmail.com	active	25
		NULL	NULL	NULL	2	Kana	kana@gmail.com	active	22
3		assalamualaikum, kumaha kabarna		3	3	Rafli	rafl@gmail.com	active	22
2		waalaikumsalam, alhamdulillah sae		4	4	Arifah	fiah@gmail.com	active	18
		NULL	NULL	NULL	5	Hikmah	hikmah@gmail.com	inactive	25
		NULL	NULL	NULL	6	Budi	budi@gmail.com	active	25

## Right Outer JOIN

- Menampilkan **hanya** data dari Table B yang **tidak punya pasangan** di Table A.



```
• SELECT <columns>
  FROM TableA
  RIGHT JOIN TableB ON A.Key = B.Key
  WHERE A.Key IS NULL;
```

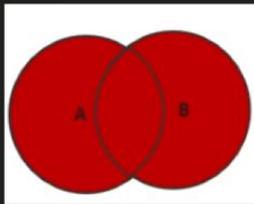
Contoh penggunaan:

```
217    -- right outer join
218 •  select * from posts right join
219    users on posts.user_id = users.id where posts.user_id is null;
```

Result Grid		Filter Rows:	Export:	Wrap Cell Content:	Result Grid	Form Editor			
		id	content	user_id	id	name	email	status	age
	▶	NULL	NULL	NULL	1	Alice	alice@gmail.com	active	25
		NULL	NULL	NULL	2	Kana	kana@gmail.com	active	22
		NULL	NULL	NULL	5	Hikmah	hikmah@gmail.com	inactive	25
		NULL	NULL	NULL	6	Budi	budi@gmail.com	active	25

## FULL JOIN

- **Menampilkan:** Semua data dari kedua tabel, cocok maupun tidak cocok
- Karena MySQL tidak mendukung **FULL JOIN secara langsung**, ini disimulasikan dengan gabungan **LEFT JOIN** dan **RIGHT JOIN menggunakan UNION**.



```
SELECT <columns>
FROM TableA
LEFT JOIN TableB ON A.Key = B.Key
UNION
SELECT <columns>
FROM TableA
RIGHT JOIN TableB ON A.Key = B.Key;
```

Contoh penggunaan:

```
221 -- full join
222 • select * from posts
223 left join users on posts.user_id = users.id
224 union
225 select * from posts
226 right join users on posts.user_id = users.id;
```

Result Grid							
	id	content	user_id	id	name	email	status
▶	2	waalaikumsalam, alhamdulillah sae	4	4	Arifah	fiah@gmail.com	active
	3	assalamualaikum, kumaha kabarna	3	3	Rafli	rafl@gmail.com	active
	NULL	NULL	NULL	1	Alice	alice@gmail.com	active
	NULL	NULL	NULL	2	Kana	kana@gmail.com	active
	NULL	NULL	NULL	5	Hikmah	hikmah@gmail.com	inactive
	NULL	NULL	NULL	6	Budi	budi@gmail.com	active

## Function

### Function di MySql

**Fungsi-fungsi SQL (SQL Functions)** memberikan cara yang **efisien dan fleksibel** untuk menganalisis data. Dengan memanfaatkan fungsi-fungsi ini di dalam *query* (perintah SQL), kamu dapat:

- **Memperdalam pemahaman terhadap data**, misalnya dengan menghitung selisih tanggal, manipulasi string untuk menemukan pola tertentu.
- **Meningkatkan akurasi analisis**, karena fungsi-fungsi tersebut bisa membersihkan, mengubah, atau menyaring data sebelum diproses lebih lanjut.
- **Mengubah data mentah (raw data) menjadi informasi yang bermanfaat**, yang bisa digunakan untuk pengambilan keputusan (*actionable knowledge*).

## Function di MySql

- Date Time Function
- String Function
- Numeric Function
- JSON Function
- Conversion Function
- Datetype Function

### Date time Function

Digunakan untuk memanipulasi dan mengambil informasi dari data bertipe **DATE**, **DATETIME**, dan **TIMESTAMP**.

<https://dev.mysql.com/doc/refman/8.4/en/date-and-time-functions.html>

Link: <https://dev.mysql.com/doc/refman/8.4/en/date-and-time-functions.html>

Contoh penggunaan:

```
228 • select curdate();
229 • select now();
230 • select year(curdate());
231
232 • select * from siswa;
233
234 • alter table siswa
235     add tanggal_masuk date;
236
237 • insert into siswa(nama, jurusan, nilai, tanggal_masuk) values
238     ('Ruru','bahasa korea', 89, '2026-01-01');
239
240 • select *, year(tanggal_masuk) as angkatan from siswa;
```

Result Grid						
	id	nama	jurusan	nilai	tanggal_masuk	angkatan
▶	1	Rafli	informatika	92	NULL	NULL
	2	Arifah	PSG	96	NULL	NULL
	3	Budi	informatika	90	NULL	NULL
	4	Siti	fisika	91	NULL	NULL
	5	Riri	fisika	84	NULL	NULL
	6	Wawan	PSG	80	NULL	NULL
	8	Ruru	bahasa korea	89	2026-01-01	2026

Result Grid  
Form Editor

## String Function

Berfungsi untuk memanipulasi data bertipe **VARCHAR**, **TEXT**, dll.

Reference : <https://dev.mysql.com/doc/refman/8.4/en/string-functions.html>

Link: <https://dev.mysql.com/doc/refman/8.4/en/string-functions.html>

Contoh penggunaan:

```
242 • select lower("raFLi ADHaN") as nama_saya;
243 • select upper("raFLi ADHaN") as nama_saya;
244 • select ltrim("      raFLi ADHaN") as nama_saya;
245 • select substring_index('raFLi ADHaN', ' ', 1) as first_name;
246 • select substring_index('raFLi ADHaN', ' ', -1) as last_name;
247
248 • select upper(nome) as Nama_Kapital from siswa;
249 • select ltrim(nome) as Nama_Kapital from siswa;
```

The screenshot shows the MySQL Workbench interface with the 'Result Grid' tab selected. The results table contains the following data:

Nama_Kapital
Rafli
Arifah
Budi
Siti
Riri
Wawan
Ruru

## Numeric Function

Digunakan untuk operasi numerik seperti pembulatan, logika matematika, dll.

Reference : <https://dev.mysql.com/doc/refman/8.4/en/numeric-functions.html>

Link: <https://dev.mysql.com/doc/refman/8.4/en/numeric-functions.html>

Contoh penggunaan:

```
251 • select pi();
252 • select *, pow(nilai,2) as nilai_pangkat_2 from siswa;
253 • select *, (nilai/100) as nilai_skala_1 from siswa;
254
```

The screenshot shows the MySQL Workbench interface with the 'Result Grid' tab selected. The results table contains the following data:

	id	nama	jurusan	nilai	tanggal_masuk	nilai_pangkat_2
▶	1	Rafli	informatika	92	NULL	8464
2	Arifah	PSG		96	NULL	9216
3	Budi	informatika		90	NULL	8100
4	Siti	fisika		91	NULL	8281
5	Riri	fisika		84	NULL	7056
6	Wawan	PSG		80	NULL	6400
8	Ruru	bahasa korea		89	2026-01-01	7921

## JSON Function

MySQL mendukung **tipe data JSON** untuk menyimpan dan mengelola data semi-terstruktur. Fungsi JSON di MySQL digunakan untuk **membaca, memodifikasi, dan mengekstrak** informasi dari kolom atau nilai bertipe JSON.

Sebelum materi dimulai, kita bahas sedikit tentang jenis data json di mysql:

- JSON Array contoh : [js', 'css', 'php']
- JSON Object: {  
    "nama": "Andi",  
    "usia": 25,  
    "kota": "Bandung" }

Link: <https://dev.mysql.com/doc/refman/8.4/en/json-function-reference.html>

Contoh penggunaan:

The screenshot shows three examples of MySQL queries using JSON functions:

- Query 255: `select json_array('css','html','js');` results in a Result Grid showing `["css", "html", "js"]`.
- Query 256: `select json_object('nama', 'rango', 'umur', 18);` results in a Result Grid showing `{"nama": "rango", "umur": 18}`.
- Query 258: `select * from siswa;`  
Query 259:  
Query 260: `alter table siswa add skilss json;`  
Query 261: `alter table siswa add alamat json;`
- Query 262: `select * from siswa;` results in a Result Grid showing student data with columns: id, nama, jurusan, nilai, tanggal\_masuk, skilss, alamat. The data includes rows for Rafli, Arifah, Budi, Siti, Riri, Wawan, and Ruru.

```

263 •     insert into siswa(nama, jurusan, nilai, skilss, alamat) values
264     (
265         'trisnaldi', 'matematika', 93, '["js", "css", "html"]',
266         {
267             "jalan": "JL. Diponegoro No.23",
268             "kota": "Medan",
269             "provinsi": "Sumatera Utara"
270         }
271     );

```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content: |

	id	nama	jurusan	nilai	tanggal_masuk	skilss	alamat
▶	1	Rafli	informatika	92	NULL	NULL	NULL
2	Arifah	PSG		96	NULL	NULL	NULL
3	Budi	informatika	90	NULL	NULL	NULL	NULL
4	Siti	fisika	91	NULL	NULL	NULL	NULL
5	Riri	fisika	84	NULL	NULL	NULL	NULL
6	Wawan	PSG		80	NULL	NULL	NULL
8	Ruru	bahasa korea	89	2026-01-01	NULL	NULL	NULL
9	trisnaldi	matematika	93	NULL	["js", "css", "html"]	{"kota": "Medan", "jalan": "JL. Diponegoro No.23", "provinsi": "Sumatera Utara"}	NULL
*	NULL	NULL	NULL	NULL	NULL	NULL	NULL

```

273 •     update siswa set alamat = json_set(alamat, '$.kota', 'Bandung')
274     where id = 9;

```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content: |

	id	nama	jurusan	nilai	tanggal_masuk	skilss	alamat
▶	1	Rafli	informatika	92	NULL	NULL	NULL
2	Arifah	PSG		96	NULL	NULL	NULL
3	Budi	informatika	90	NULL	NULL	NULL	NULL
4	Siti	fisika	91	NULL	NULL	NULL	NULL
5	Riri	fisika	84	NULL	NULL	NULL	NULL
6	Wawan	PSG		80	NULL	NULL	NULL
8	Ruru	bahasa korea	89	2026-01-01	NULL	NULL	NULL
9	trisnaldi	matematika	93	NULL	["js", "css", "html"]	{"kota": "Bandung", "jalan": "JL. Diponegoro No.23", "provinsi": "Sumatera Utara"}	NULL
*	NULL	NULL	NULL	NULL	NULL	NULL	NULL

**Json\_replace** itu hanya mengganti Sedangkan kalau **json\_set** itu bisa menambahkan.

```

276 •     select nama, json_keys(alamat) from siswa;

```

Result Grid | Filter Rows: | Export: |

	nama	json_keys(alamat)
▶	Rafli	NULL
	Arifah	NULL
	Budi	NULL
	Siti	NULL
	Riri	NULL
	Wawan	NULL
	Ruru	NULL
	trisnaldi	["kota", "jalan", "provinsi", "post_code"]

Untuk mengecek yang ada json\_key nya

```

278 •     update siswa set alamat = json_remove(alamat, '$.post_code')
279     where id = 9;

```

Perintah untuk menghapus key value json

281 • select \*, json\_extract(alamat, '\$.kota') as kota from siswa;

Result Grid | Filter Rows: Export: Wrap Cell Content:

	id	nama	jurusan	nilai	tanggal_masuk	skilss	alamat	kota
▶	1	Rafli	informatika	92	NULL	NULL	NULL	NULL
	2	Arifah	PSG	96	NULL	NULL	NULL	NULL
	3	Budi	informatika	90	NULL	NULL	NULL	NULL
	4	Siti	fisika	91	NULL	NULL	NULL	NULL
	5	Riri	fisika	84	NULL	NULL	NULL	NULL
	6	Wawan	PSG	80	NULL	NULL	NULL	NULL
	8	Ruru	bahasa korea	89	2026-01-01	NULL	NULL	NULL
	9	trisnaldi	matematika	93	NULL	["js", "css", "html"]	{"kota": "Bandung", "jalan": "Jl. Diponegoro No... "}	Bandung

284 • select \*, json\_contains(skilss, '["js"]') as bisa\_javascript  
285 from siswa;

Result Grid | Filter Rows: Export: Wrap Cell Content:

	id	nama	jurusan	nilai	tanggal_masuk	skilss	alamat	bisa_javascript
▶	1	Rafli	informatika	92	NULL	NULL	NULL	NULL
	2	Arifah	PSG	96	NULL	NULL	NULL	NULL
	3	Budi	informatika	90	NULL	NULL	NULL	NULL
	4	Siti	fisika	91	NULL	NULL	NULL	NULL
	5	Riri	fisika	84	NULL	NULL	NULL	NULL
	6	Wawan	PSG	80	NULL	NULL	NULL	NULL
	8	Ruru	bahasa korea	89	2026-01-01	NULL	NULL	NULL
	9	trisnaldi	matematika	93	NULL	["js", "css", "html"]	{"kota": "Bandung", "jalan": "Jl. Diponegoro No... "}	1

## Flow Control Function

### Flow Control Function

Flow Control Functions digunakan untuk mengatur alur logika dalam query SQL, mirip seperti perintah if, switch, atau else di bahasa pemrograman.

- CASE
- IF()
- IFNULL()
- NULLIF()

### CASE

Memilih nilai berdasarkan **berbagai kondisi**.

```
CASE
    WHEN kondisi1 THEN hasil1
    WHEN kondisi2 THEN hasil2
    ...
    ELSE hasil_default
END
```

Contoh penggunaan:

```
289 •     select *,
290     case
291         when nilai >= 90 then 'A'
292         when nilai >= 85 then 'B'
293         when nilai >= 80 then 'C'
294         else 'D'
295     end as grade
296 from siswa;
```

Result Grid							
	id	nama	jurusan	nilai	tanggal_masuk	skills	alamat
▶	1	Rafli	informatika	92	NULL	NULL	NULL
	2	Arifah	PSG	96	NULL	NULL	NULL
	3	Budi	informatika	90	NULL	NULL	NULL
	4	Siti	fisika	91	NULL	NULL	NULL
	5	Riri	fisika	84	NULL	NULL	NULL
	6	Wawan	PSG	80	NULL	NULL	NULL
	8	Ruru	bahasa korea	89	2026-01-01	NULL	NULL
	9	trisnaldi	matematika	93	NULL	["js", "css", "html"]	{"kota": "Bandung", "jalan": "JL. Diponegoro No... A

## IF(expr1, expr2, expr3)

Jika expr1 benar (TRUE), kembalikan expr2, jika tidak, kembalikan expr3.

```
IF(kondisi, nilai_jika_true, nilai_jika_false)
```

Contoh penggunaan:

```
289 •     select *,
290     case
291         when nilai >= 90 then 'A'
292         when nilai >= 85 then 'B'
293         when nilai >= 80 then 'C'
294         else 'D'
295     end as grade,
296     if (nilai >= 80, 'Lulus', 'Tidak Lulus') as passed
297 from siswa;
```

Result Grid									
	id	nama	jurusan	nilai	tanggal_masuk	skills	alamat	grade	passed
▶	1	Rafli	informatika	92	NULL	NULL	NULL	A	Lulus
	2	Arifah	PSG	96	NULL	NULL	NULL	A	Lulus
	3	Budi	informatika	90	NULL	NULL	NULL	A	Lulus
	4	Siti	fisika	91	NULL	NULL	NULL	A	Lulus
	5	Riri	fisika	84	NULL	NULL	NULL	C	Lulus
	6	Wawan	PSG	80	NULL	NULL	NULL	C	Lulus
	8	Ruru	bahasa korea	89	2026-01-01	NULL	NULL	B	Lulus
	9	trisnaldi	matematika	93	NULL	["js", "css", "html"]	{"kota": "Bandung", "jalan": "JL. Diponegoro No... A	A	Lulus

## IFNULL(expr1, expr2)

Kembalikan expr1 jika bukan NULL, jika expr1 NULL maka kembalikan expr2.

```
IFNULL(nilai1, nilai_pengganti)
```

Contoh penggunaan:

```
289 •     select *,  
290     case  
291         when nilai >= 90 then 'A'  
292         when nilai >= 85 then 'B'  
293         when nilai >= 80 then 'C'  
294         else 'D'  
295     end as grade,  
296     if (nilai >= 80, 'Lulus', 'Tidak Lulus') as passed,  
297     ifnull(nilai, 0) as nilai_baru  
298     from siswa;
```

Result Grid							Filter Rows:	Export:	Wrap Cell Content:	
	id	nama	jurusan	nilai	tanggal_masuk	sklls	alamat	grade	passed	nilai_baru
▶	1	Rafli	informatika	92	NULL	NULL	NULL	A	Lulus	92
2	Arifah	PSG	96	NULL	NULL	NULL	NULL	A	Lulus	96
3	Budi	informatika	90	NULL	NULL	NULL	NULL	A	Lulus	90
4	Siti	fisika	91	NULL	NULL	NULL	NULL	A	Lulus	91
5	Riri	fisika	84	NULL	NULL	NULL	NULL	C	Lulus	84
6	Wawan	PSG	80	NULL	NULL	NULL	NULL	C	Lulus	80
7	Ruru	bahasa korea	89	2026-01-01	NULL	NULL	NULL	B	Lulus	89
8	trisnaldi	matematika	93	NULL	[{"js": "css", "html": "js"}]	{"kota": "Bandung", "jalan": "JL. Diponegoro No...}	A	Lulus	93	

## NULLIF(expr1, expr2) – Return NULL if Equal

Kembalikan NULL jika expr1 = expr2, jika tidak, kembalikan expr1.

```
NULLIF(expr1, expr2)
```

Contoh penggunaan:

```
300 •     select nullif(9,9);
```

Result Grid		Filter Rows:
	nullif(9,9)	
▶	NULL	

```
301 •     select nullif(10,9);
```

Result Grid		Filter Rows:
	nullif(10,9)	
▶	10	

## Indexes

### Indexes

**Index (indeks)** di MySQL adalah struktur data khusus yang digunakan oleh database untuk meningkatkan kecepatan pencarian data dalam tabel. Tanpa index, MySQL harus melakukan **full table scan** (membaca semua baris satu per satu) untuk menemukan data yang sesuai.

Analogi: Index seperti daftar isi di buku. Tanpa index, kita harus membaca seluruh buku untuk menemukan bab tertentu.

### Kenapa menggunakan index

#### ↳ Tujuan dan Manfaat Penggunaan Index:

- Meningkatkan performa query **SELECT** — terutama pada tabel besar.
- Mempercepat operasi **JOIN**, **ORDER BY**, **GROUP BY**.
- Mengoptimalkan pencarian berdasarkan kolom tertentu.
- Mengurangi waktu eksekusi query.

#### ⚠️ Tapi hati-hati:

- Index membuat **insert/update/delete** jadi **sedikit lebih lambat** karena MySQL harus memperbarui struktur index.
- Terlalu banyak index dapat mengganggu performa keseluruhan.

### Tips Penggunaan Index

1. Gunakan index pada kolom yang sering dipakai dalam **WHERE**, **JOIN**, **ORDER BY**, dan **GROUP BY**.
2. Hindari index pada kolom dengan data yang hampir semuanya unik (**contohnya kolom ID sudah otomatis PRIMARY KEY**).
3. Hindari indexing pada kolom yang sering diubah atau memiliki tipe data besar.

### Membuat index

```
CREATE TABLE produk (
    id INT PRIMARY KEY,
    nama_produk VARCHAR(100),
    kategori VARCHAR(50),
    harga INT,
    INDEX index_name (kategori)
);
```

Contoh penggunaan:

```
305 • create table produk(
306     id int auto_increment primary key,
307     nama varchar(30),
308     kategori varchar(30),
309     harga int,
310     index kateogry_index(kategori),
311     index nama_index(nama),
312
313     -- nama, nama dan kategori, nama dan kategori dan harga
314     -- not impact ke kategori dan harga, kategori, harga
315     index nama_kategori_harga(nama, kategori, harga)
316 );
317
318 • show create table produk;

320 • select * from produk where kategori = 'a';
321 • select * from produk where nama = 'a';
322 • select * from produk where nama = 'a' and kategori = 'a';
323 • select * from produk where nama = 'a' and kategori = 'a' and harga=1;
324
```

## Menambah dan menghapus index

```
ALTER TABLE produk
DROP INDEX index_name;

ALTER TABLE produk
ADD INDEX index_name (kolom);
```

Contoh penggunaan:

```
325 • alter table produk drop index nama_index;
326
327 • alter table produk add index harga_index(harga);
```

## Kesimpulan

Tanpa Index	Dengan Index
Full table scan	Akses langsung ke data
Lambat untuk pencarian	Cepat untuk pencarian
Efisien untuk data kecil	Wajib untuk data besar

## Subquery

### Subquery

Subquery adalah query SQL yang **ditempatkan di dalam query lain**. Subquery disebut juga **inner query**, sedangkan query luarnya disebut **outer query**.

Biasanya digunakan untuk:

- Mengambil data berdasarkan hasil dari query lain
- Digunakan dalam WHERE, FROM, atau SELECT

### Subquery dalam where

Digunakan untuk membandingkan nilai terhadap hasil subquery.

```
select * from siswa  
where nilai > (select avg(nilai) from siswa);
```

Contoh penggunaan:

```
329 • select * from siswa  
330     where nilai > (select avg(nilai) from siswa);
```

	id	nama	jurusan	nilai	tanggal_masuk	skilss	alamat
▶	1	Rafli	informatika	92	NULL	NULL	NULL
	2	Arifah	PGSD	96	NULL	NULL	NULL
	3	Budi	informatika	90	NULL	NULL	NULL
	4	Siti	fisika	91	NULL	NULL	NULL
*	9	trishnaldi	matematika	93	NULL	["js", "css", "html"]	{"kota": "Bandung", "jal": NULL}
	NULL	NULL	NULL	NULL	NULL	NULL	NULL

### Subquery dalam from ( Derived Table )

Subquery ini menghasilkan tabel sementara.

```
select name, content from (select posts.id ,  
posts.content, users.name , users.email  
from posts join users on posts.user_id = users.id) as sub;
```

Contoh penggunaan:

```
332 •   select name, content from
333     (select p.id, p.content, u.name, u.email
334      from posts as p join users as u on p.user_id = u.id)
335      as derived_table;
```

Result Grid		Filter Rows:	Export:	Wrap Cell Content:
	name	content		
▶	Arifah	waalaikumsalam, alhamdulillah sae		
	Rafli	assalamualaikum, kumaha kabarna		

## Subquery dalam SELECT

Digunakan untuk menghitung atau mengambil nilai tambahan.

```
select nama,jurusan,nilai,
(select avg(nilai) from siswa) as 'rata-rata siswa' from siswa;
```

Contoh penggunaan:

```
337 •   select nama, jurusan, nilai,
338       (select avg(nilai) from siswa) as 'Rata rata nilai'
339       from siswa;
```

Result Grid		Filter Rows:	Export:	Wrap Cell Content:
	nama	jurusan	nilai	Rata rata nilai
▶	Rafli	informatika	92	89.3750
	Arifah	PGSD	96	89.3750
	Budi	informatika	90	89.3750
	Siti	fisika	91	89.3750
	Riri	fisika	84	89.3750
	Wawan	PGSD	80	89.3750
	Ruru	bahasa korea	89	89.3750
	trisnaldi	matematika	93	89.3750

## View

### View

**View** adalah **tabel virtual** yang isinya berasal dari hasil query (SELECT) terhadap satu atau lebih tabel yang ada di database.

- View tidak menyimpan data secara fisik, hanya menyimpan query-nya.
- Digunakan untuk menyederhanakan query yang kompleks, membatasi akses kolom/baris tertentu, dan meningkatkan keamanan.

### Membuat view

```
CREATE VIEW nama_view AS
SELECT kolom1, kolom2
FROM nama_tabel
WHERE kondisi;
< - 10
```

Contoh penggunaan:

The screenshot shows the MySQL Workbench interface. On the left, the schema browser displays a database named 'test' with tables like course\_student, courses, mahasiswa, posts, produk, profiles, rafli, siswa, student, and users. A 'Views' folder contains a single view named 'siswa\_view'. The central pane shows the SQL code for creating the view:

```
342 •    create view siswa_view as
343      select *,
344          case
345              when nilai >= 90 then 'A'
346              when nilai >= 85 then 'B'
347              when nilai >= 80 then 'C'
348              else 'D'
349          end as grade,
350      if (nilai >= 80, 'Lulus', 'Tidak Lulus') as passed,
351      ifnull(nilai, 0) as nilai_baru
352      from siswa;
353
354      -- versi simple setelah menerapkan view
355 •    select * from siswa_view;
```

Below the code, the 'Result Grid' shows the data returned by the query:

	id	nama	jurusan	nilai	tanggal_masuk	skilss	alamat
▶	1	Rafli	informatika	92	NULL	NULL	NULL
	2	Arifah	PGSD	96	NULL	NULL	NULL
	3	Budi	informatika	90	NULL	NULL	NULL
	4	Siti	fisika	91	NULL	NULL	NULL
	5	Riri	fisika	87	NULL	NULL	NULL
	6	Wawan	PGSD	80	NULL	NULL	NULL
	8	Ruru	bahasa korea	89	2026-01-01	NULL	NULL
	9	trisnaldi	matematika	93	NULL	["js", "css", "html"]	{"kota": "B

The right sidebar of the results grid includes icons for 'Result Grid', 'Form Editor', and 'Field Types'.

```
355      -- kita bisa update view, tetapi yang
356      -- akan ke update adalah tabel aslinya
357 •  update siswa_view set nilai = 87 where id = 5;
```

	id	nama	jurusan	nilai	tanggal_masuk	skills	alamat
▶	1	Rafli	informatika	92	NULL	NULL	NULL
	2	Arifah	PGSD	96	NULL	NULL	NULL
	3	Budi	informatika	90	NULL	NULL	NULL
	4	Siti	fisika	91	NULL	NULL	NULL
	5	Riri	fisika	87	NULL	NULL	NULL
	6	Wawan	PGSD	80	NULL	NULL	NULL
	8	Ruru	bahasa korea	89	2026-01-01	NULL	{ "kota": "B
	9	trisnaldi	matematika	93	NULL	[ "js", "css", "html" ]	

\*Hati-hati: Perlu diingat kalau melakukan **update** pada view ini, itu berpengaruh ke data yang aslinya.

\*catatan: Ketika menggunakan 2 tabel itu gak bisa **update**.

## Mengubah dan menghapus view

```
CREATE OR REPLACE VIEW nama_view AS
SELECT ...
```

```
DROP VIEW nama_view;
```

Contoh penggunaan:

```

361 •  create or replace view siswa_view as
362      select nama, jurusan,
363          case
364              when nilai >= 90 then 'A'
365              when nilai >= 85 then 'B'
366              when nilai >= 80 then 'C'
367              else 'D'
368          end as grade,
369          if (nilai >= 80, 'Lulus', 'Tidak Lulus') as passed
370      from siswa;
371
372 •  select * from siswa_view;

```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

	nama	jurusan	grade	passed
▶	Rafli	informatika	A	Lulus
	Arifah	PGSD	A	Lulus
	Budi	informatika	A	Lulus
	Siti	fisika	A	Lulus
	Riri	fisika	B	Lulus
	Wawan	PGSD	C	Lulus
	Ruru	bahasa korea	B	Lulus
	trisnaldi	matematika	A	Lulus

## Kelebihan view

1. Menyederhanakan query kompleks.
2. Membatasi akses data sensitif (security).
3. Menyediakan antarmuka tetap untuk query meskipun struktur tabel berubah.
4. Memfasilitasi penggunaan kembali query.

## ~~Kelebihan view~~

1. Tidak semua view dapat di-update (tergantung kompleksitas query-nya).
2. Tidak menyimpan data (hanya query), sehingga tidak meningkatkan performa secara langsung.
3. Beberapa operasi (JOIN kompleks, agregasi) bisa membuat view tidak bisa di-update atau delete.

## User Management

## User Management di MySQL

User Management adalah proses untuk mengelola siapa yang dapat mengakses MySQL server, termasuk pengaturan:

- Nama pengguna (user)
  - Hak akses (privileges)
  - Pengaturan keamanan (host, password)

## Tujuan User Management

- Menjamin keamanan database
  - Memastikan setiap user hanya dapat mengakses data yang diperlukan
  - Mendukung multi-user environment

## Struktur User di MySQL

MySQL biasanya menyimpan informasi user dalam tabel sistem mysql.user, atau database mysql dan table user

```
show databases;  
use mysql;  
show tables;  
SELECT * FROM user;
```

## Contoh penggunaan:

```
374 •      show databases;  
375 •      use mysql;  
376 •      show tables;  
377 •      select * from user;
```

## Membuat user

```
CREATE USER 'nama_user'@'host' IDENTIFIED BY 'password';
```

```
CREATE USER 'budi'@'localhost' IDENTIFIED BY '123456';
```

```
CREATE USER 'andi'@'%' IDENTIFIED BY 'rahasia123';
```

Host digunakan untuk menentukan dari mana user tersebut bisa mengakses MySQL. Biasanya 'localhost' jika akses hanya dari server lokal, atau '%' untuk akses dari IP mana pun.

## Catatan Keamanan

Penggunaan '%' sebaiknya dilakukan **dengan hati-hati**, terutama di server yang dapat diakses publik, karena berisiko keamanan lebih tinggi.

Disarankan juga untuk:

- Mengatur **firewall** agar hanya IP tertentu yang bisa mengakses MySQL.

Contoh penggunaan:

```
379 • create user 'andi'@'localhost' identified by '123456';
```

Result Grid							
	Host	User	Select_priv	Insert_priv	Update_priv	Delete_priv	O
▶	localhost	andi	N	N	N	N	N
	localhost	mysql.infoschema	Y	N	N	N	N
	localhost	mysql.session	N	N	N	N	N
	localhost	mysql.sys	N	N	N	N	N
*	localhost	root	Y	Y	Y	Y	Y
*	NULL	NULL	NULL	NULL	NULL	NULL	NULL

```
Command Prompt - mysql x + v
Microsoft Windows [Version 10.0.26100.7623]
(c) Microsoft Corporation. All rights reserved.

C:\Users\mrafl>mysql -u andi -p
Enter password: *****
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 15
Server version: 9.5.0 MySQL Community Server - GPL

Copyright (c) 2000, 2025, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> |
```

## Membuat, menghapus dan update user

```
CREATE USER 'budi'@'localhost' IDENTIFIED BY '123456';

ALTER USER 'budi'@'localhost' IDENTIFIED BY 'budi123';

DROP USER 'budi'@'localhost';
```

Contoh penggunaan:

```
alter user 'andi'@'localhost' identified by 'andi123';
```

Host	User	Select_priv	Insert_priv	Update_priv	Delete_priv	Grant_priv
localhost	mysql.infoschema	Y	N	N	N	N
localhost	mysql.session	N	N	N	N	N
localhost	mysql.sys	N	N	N	N	N
localhost	root	Y	Y	Y	Y	Y
*	NULL	NULL	NULL	NULL	NULL	NULL

## Hak akses ( Privileges )

Di MySQL, **privileges** atau hak akses digunakan untuk mengontrol apa saja yang boleh dilakukan oleh pengguna terhadap database, tabel, atau objek tertentu. Penggunaan hak akses yang tepat sangat penting untuk menjaga keamanan dan integritas data.

## Hak akses ( Privileges )

Privilege	Deskripsi
ALL	Semua hak akses (kecuali GRANT OPTION)
SELECT	Membaca data dari tabel
INSERT	Menambahkan data ke tabel
UPDATE	Mengubah data dalam tabel
DELETE	Menghapus data dari tabel
CREATE	Membuat database atau tabel baru
DROP	Menghapus database atau tabel
ALTER	Mengubah struktur tabel
INDEX	Membuat atau menghapus index
GRANT OPTION	Memberi hak akses ke user lain

# Hak akses ( Privileges )

## Sintaks dasar

```
GRANT hak_akses ON database.tabel TO 'nama_user'@'host';
```

```
GRANT ALL PRIVILEGES ON db_toko.* TO 'budi'@'localhost';
```

```
GRANT SELECT, INSERT ON db_toko.produk TO 'budi'@'localhost';
```

```
GRANT ALL PRIVILEGES ON *.* TO 'admin'@'%';
```

```
GRANT SELECT ON db_toko.* TO 'eko'@'localhost' WITH GRANT OPTION;
```

```
REVOKE SELECT, INSERT ON db_toko.produk FROM 'budi'@'localhost';
```

Contoh penggunaan:

```
385 • grant select on test.siswa to 'andi'@'localhost';
```

```
C:\Users\mrafl>mysql -u andi -p
Enter password: *****
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 17
Server version: 9.5.0 MySQL Community Server - GPL

Copyright (c) 2000, 2025, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> use test
Database changed
mysql> select * from siswa;
+----+-----+-----+-----+-----+
| id | nama      | jurusan    | nilai | tanggal_masuk | skilss           | alamat
+----+-----+-----+-----+-----+
| 1  | Rafli     | informatika | 92   | NULL          | NULL            | NULL
| 2  | Arifah    | PGSD       | 96   | NULL          | NULL            | NULL
+----+-----+-----+-----+-----+
```

Pada perintah diatas memperbolehkan user 'andi' untuk melakukan use pada database test dan dapat melakukan select.

```
mysql> delete from siswa where id=8
      ->;
ERROR 1142 (42000): DELETE command denied to user 'andi'@'localhost' for table 'siswa'
mysql> |
```

Ketika melakukan delete tidak dapat di karenakan pada perintah sebelumnya hanya memperbolehkan select saja.

```
385 • grant select, delete on test.siswa to 'andi'@'localhost';
```

Perintah harus berubah atau di tambahkan seperti ini, jadi Ketika melakukan delete pada user andi sudah dapat terhapus.

```
mysql> delete from siswa where id=8
      ->;
ERROR 1142 (42000): DELETE command denied to user 'andi'@'localhost' for table 'siswa'
mysql> delete from siswa where id=8;
Query OK, 1 row affected (0.075 sec)

mysql> select * from siswa;
+----+-----+-----+-----+-----+-----+
| id | nama | jurusan | nilai | tanggal_masuk | skilss | alamat |
+----+-----+-----+-----+-----+-----+
| 1 | Rafli | informatika | 92 | NULL | NULL | NULL |
| 2 | Arifah | PGSD | 96 | NULL | NULL | NULL |
| 3 | Budi | informatika | 90 | NULL | NULL | NULL |
| 4 | Siti | fisika | 91 | NULL | NULL | NULL |
| 5 | Riri | fisika | 87 | NULL | NULL | NULL |
| 6 | Wawan | PGSD | 80 | NULL | NULL | NULL |
| 9 | trisnaldi | matematika | 93 | NULL | ["js", "css", "html"] | {"kota": "Bandung", "jalan": "JL. Diponegoro No.23", "provinsi": "Sumatera Utara"} |

```

387 • `grant all privileges on test.* to 'andi'@'localhost';`

```
C:\Users\mrafl>mysql -u andi -p
Enter password: *****
Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 18
Server version: 9.5.0 MySQL Community Server - GPL

Copyright (c) 2000, 2025, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> use test;
Database changed
mysql> select * from posts;
+----+-----+-----+
| id | content | user_id |
+----+-----+-----+
| 2 | waalaikumsalam, alhamdulillah sae | 4 |
| 3 | assalamualaikum, kumaha kabarna | 3 |
+----+-----+-----+
2 rows in set (0.014 sec)
```

389 ✘ `revoke all privileges on test.siswa from 'andi'@'localhost';`

## Transaction

### Transaction

**Transaksi (Transaction)** adalah sekumpulan perintah SQL yang dieksekusi sebagai satu kesatuan logis. Transaksi memastikan bahwa database tetap dalam kondisi **konsisten** meskipun terjadi error atau kegagalan sistem.

Note : Transaksi hanya bekerja di **engine yang mendukung transaksi**, seperti **InnoDB**.

Contoh penggunaan:

```
393 •    start transaction;  
394  
395 •    update siswa set nilai = 100 where id=3;  
396 •    select * from siswa;  
397  
398 •    commit;  
399 •    rollback;
```

Perintah commit itu berarti database sudah final data akan berubah.

Rollback itu Ketika salah melakukan menghapus/delete/update, bisa di rollback.