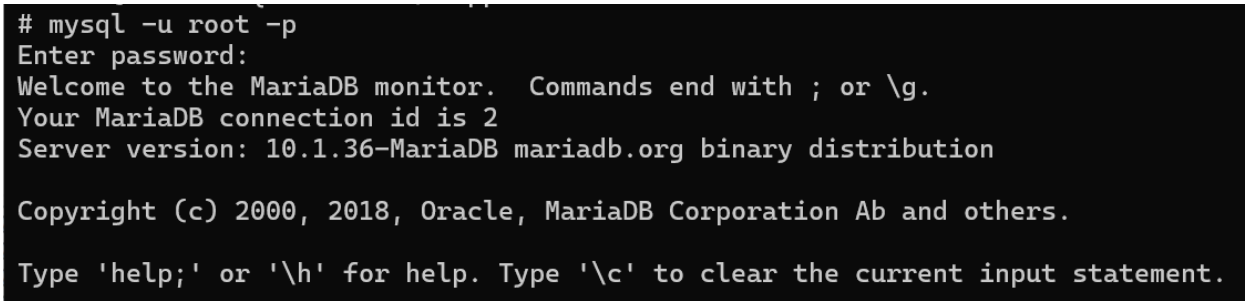
HAVING DAN GROUP BY
Create database dan Tabel Pegawai

Praktek kali ini adalah membuat sebuah database perusahaan dengan tabel pegawai didalamnya. Untuk merealisasikannya, kita perlu membuat database untuk menampung tabel pegawai kita. Masuk ke aplikasi XAMPP, klik start pada tombol MySQL. Setelah menyala, klik tombol shell untuk masuk ke command promp.

Langkah-langkah pembuatan database:

1. Ketik `mysql -u root -p` dengan demikian kita sudah masuk ke dalam server lokal kita



2. Buat database dengan nama company_namaku dalam praktek kali ini saya membuatnya dengan nama company_condrado

```
CREATE DATABASE company_condrado;
```

penjelasan:

- `CREATE DATABASE`: Ini adalah perintah SQL yang digunakan untuk membuat database baru.
- `company_condrado`: Ini adalah nama dari database yang akan dibuat.

untuk melihat hasil dari database yang dibuat bisa dengan menggunakan perintah `show databases;`

```
MariaDB [(none)]> SHOW DATABASES;
+----------------------+
| Database             |
+----------------------+
| aisyah               |
| akun                 |
| arsip                |
| belajar_database     |
| company_condrado     |
| db_cuti              |
| event                |
| icha_siskom          |
| information_schema   |
| inventar-it          |
| iqbal_siskom         |
| kelompok_2           |
| mysql                |
| performance_schema   |
| perpustakaan         |
| phpmyadmin           |
| proyek1              |
| rental_mobil_condradp |
| rental_mobil_ryan    |
| ryan                 |
| sekolah_adrian       |
| sekolah_condrado     |
| sekolah_fathir       |
| sekolah_paisal       |
| test                 |
| tugasibuanti         |
+----------------------+
26 rows in set (0.00 sec)
```

3. Jika sudah memastikan bahwa database berhasil dibuat, sekarang adalah menggunakan database tersebut untuk dimasukan tabel didalmnnya, menggunakan perintah `use` lalu diikuti nama database yang akan digunakan.

```
USE company_condrado;
```

Penjelasan:

- `USE`: Ini adalah perintah SQL yang digunakan untuk memilih database yang akan digunakan.
- `company_condrado`: Ini adalah nama dari database yang akan dipilih.

Jika berhasil maka hasilnya akan sebagai berikut:

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

4. Sekarang kita akan membuat tabel di dalam database `company_condrado` ini dengan menggunakan perintah `create table` diikuti oleh nama tabel yang ingin kita buat

```
CREATE TABLE pegawai (
    NIP INT PRIMARY KEY,
    NDep VARCHAR(100) NOT NULL,
    NBlk VARCHAR(100),
    JK ENUM('P', 'L') NOT NULL,
    Alamat TEXT NOT NULL,
    telp VARCHAR(15) NOT NULL,
    jabatan ENUM('Sales', 'Manajer', 'Staff'),
    Gaji BIGINT NOT NULL,
    NoCab VARCHAR(10) NOT NULL
);
```

Penjelasan:

- `CREATE TABLE pegawai`: Bagian ini mendefinisikan bahwa kita akan membuat tabel baru bernama `pegawai`.
- `NIP INT PRIMARY KEY`: Kolom `NIP` bertipe data `INT` dan berfungsi sebagai kunci utama (PRIMARY KEY) dari tabel. Ini berarti setiap nilai dalam kolom ini harus unik dan tidak boleh kosong (NULL).
- `NDep VARCHAR(100) NOT NULL`: Kolom `NDep` bertipe data `VARCHAR` dengan panjang maksimum 100 karakter dan tidak boleh kosong (NOT NULL).
- `NBlk VARCHAR(100)`: Kolom `NBlk` bertipe data `VARCHAR` dengan panjang maksimum 100 karakter. Kolom ini bisa bernilai NULL karena tidak ada penanda `NOT NULL`.
- `JK ENUM('P', 'L') NOT NULL`: Kolom `JK` bertipe data `ENUM` dengan pilihan nilai 'P' (perempuan) atau 'L' (laki-laki) dan tidak boleh kosong (NOT NULL).
- `Alamat TEXT NOT NULL`: Kolom `Alamat` bertipe data `TEXT` dan tidak boleh kosong (NOT NULL).
- `telp VARCHAR(15) NOT NULL`: Kolom `telp` bertipe data `VARCHAR` dengan panjang maksimum 15 karakter dan tidak boleh kosong (NOT NULL).
- `jabatan ENUM('Sales', 'Manajer', 'Staff')`: Kolom `jabatan` bertipe data `ENUM` dengan pilihan nilai 'Sales', 'Manajer', atau 'Staff'. Kolom ini bisa bernilai NULL karena tidak ada penanda `NOT NULL`.
- `Gaji BIGINT NOT NULL`: Kolom `Gaji` bertipe data `BIGINT` dan tidak boleh kosong (NOT NULL).

- `NoCab VARCHAR(10) NOT NULL`: Kolom `NoCab` bertipe data `VARCHAR` dengan panjang maksimum 10 karakter dan tidak boleh kosong (`NOT NULL`).

Hasil:

```
MariaDB [company_condrado]> CREATE TABLE pegawai (
    ->     NIP INT PRIMARY KEY,
    ->     NDep VARCHAR(100) NOT NULL,
    ->     NBlk VARCHAR(100),
    ->     JK ENUM('P', 'L') NOT NULL,
    ->     Alamat TEXT NOT NULL,
    ->     telp VARCHAR(15) NOT NULL,
    ->     jabatan ENUM('Sales', 'Manajer', 'Staff'),
    ->     Gaji BIGINT NOT NULL,
    ->     NoCab VARCHAR(10) NOT NULL
    -> );
Query OK, 0 rows affected (0.04 sec)
```

5. Untuk melihat tabel yang telah dibuat sudah terdaftar didalam database, bisa gunakan perintah `show tables`. Jika tabel yang dibuat sudah berhasil maka akan terdata di database tersebut.

```
MariaDB [company_condrado]> show tables;
+-----------------------------+
| Tables_in_company_condrado |
+-----------------------------+
| pegawai                     |
+-----------------------------+
1 row in set (0.01 sec)
```

6. Setelah memastikan bahwa tabel berhasil dibuat maka sekarang kita akan melihat struktur dari tabel yang sudah kita buat menggunakan perintah `desc table` kemudian diikuti oleh nama tabel yang tadi dibuat.

```
DESC table pegawai;
```

Penjelasan

- `DESC`: Singkatan dari `DESCRIBE`, perintah ini digunakan untuk mendapatkan informasi tentang struktur sebuah tabel dalam database.
- `pegawai`: Nama tabel yang ingin Kita lihat strukturnya.
  Hasil:

```
MariaDB [company_condrado]> desc pegawai;
+---------+---------------------------------+------+-----+---------+-------+
| Field   | Type                            | Null | Key | Default | Extra |
+---------+---------------------------------+------+-----+---------+-------+
| NIP     | int(11)                         | NO   | PRI | NULL    |       |
| NDep    | varchar(100)                    | NO   |     | NULL    |       |
| NBlk    | varchar(100)                    | YES  |     | NULL    |       |
| JK      | enum('P','L')                   | NO   |     | NULL    |       |
| Alamat  | text                            | NO   |     | NULL    |       |
| telp    | varchar(15)                     | NO   |     | NULL    |       |
| jabatan | enum('Sales','Manajer','Staff') | YES  |     | NULL    |       |
| Gaji    | bigint(20)                      | NO   |     | NULL    |       |
| NoCab   | varchar(10)                     | NO   |     | NULL    |       |
+---------+---------------------------------+------+-----+---------+-------+
9 rows in set (0.04 sec)
```

- Kolom `NIP` menggunaan tipe data `int` dan `primary key` dikarenakan kolom ini merupakan data unique, dimana data inilah yang membedakan antara satu pegawai dengan pegawai lainnya, dan alasan digunakannya `int` sebagai tipe datanya adalah karena isi dari datanya nantinya berupa angka. Karena kolom ini adalah `primary` maka kolom ini otomatis `not null`
- Kolom `NDep` merupakan kolom untuk mengisi nama depan dari pegawai. Karena isi dari kolom ini adalah karakter, maka kolom ini menggunakan tipe data `varchar`. Kolom ini juga menggunakan constrain `NOT NULL` agar mencegah kolom kosong karean pegawai diwajibkan memasukan minimal nama depan mereka.
- Kolom `NBlk` merupakan kolom untuk menampung data nama belakang dari pegawai. Karena isi dari kolom ini nantinya adalah karakter, maka kolom ini menggunakan tipe data `varchar`. Kolom ini tidak diatur dengan constrain `not null` karena tiap pegawai tidak semua memiliki nama belakang sehingga data kolom bisa kosong.
- Kolom `JK` merupakan kolom untuk memilih jenis kelamin dari pegawai. Data yang bisa akan dimasukannya nantinya hanya bisa dua yaitu Laki-laki (L) dan Perempuan (P), maka tipe data yang tepat untuk kolom ini adalah `ENUM` dengan value `('P', 'L')`. Nantinya pegawai hanya boleh memilih antara pilihan `P` atau `L` untuk mengidentifikasi jenis kelamin mereka. Karena data ini akan memilih jenis kelamin, maka kolom tidak boleh kosong, maka digunakanlah constrain `NOT NULL`.
- Kolom `Alamat` merupakan kolom untuk memasukan alamat pegawai. Karena nantinya data yang dimasukkan adalah alamat, maka data tentunya akan memiliki value panjang sehingga tipe data `varchar` tidak akan mencukupi valuenya. Oleh karena itu digunakanlah tipe data `TEXT` sebagai alternatifnya. Alamat sendiri tidak boleh kosong oleh karena itu digunakanlah constrain `NOT NULL`.
- Kolom `telp` merupakan kolom untuk memasukan nomor telepon pegawai. Nomor telepon sendiri memang merupakan angka, namun nomor telepon yang akan dimasukkan akan menggunakan simbol garis hubung (-) untuk memudahkan pembacaan nomor. karena garis hubung merupakan karakter, maka digunakanlah tipe data `varchar` sebagai alternatifnya.

Nomor telepon tidak boleh kosong sehingga digunakanlah constrain `Not Null`.

- Kolom `Jabatan` merupakan kolom untuk memasukan jabatan pegawai. Dalam perusahaan ini hanya terdapat 3 jabatan yang boleh dipilih yaitu 'Manajer', 'Sales', 'Staf'. Karena itu kita kembali menggunakan tipe data `ENUM` dengan value jabatan tersebut, agar pegawai hanya dapat memilih ketiga jabatan tersebut. Karena salah satu data dari pegawai tidak memasukan jabatan mereka, maka kolom ini menggunakan constrain `Null` agar kolomnya bisa kosong.
- Kolom `Gaji` merupakan kolom untuk memasukan gaji pegawai. Sama seperti alamat tadi, nilai gaji yang akan dimasukkan tidak akan mencukupi jika harus menggunakan tipe data `int`. Oleh karena itu, digunakanlah `bigint` sebagai alternatif. Karena isi dari kolom Gaji tidak boleh kosong, maka digunakanlah constrain `NOT NULL`.
- Kolom `NoCab` merupakan kolom untuk memasukan nomor cabang. Karena isi datanya merupakan gabungan antara huruf dan angka, maka digunakanlah tipe data `varchar`. Kolom tidak boleh kosong, maka digunakanlah constrain `not null`.

7. Setelah memastikan struktur tabel sudah tepat, saatnya memasukan data pada tabel yang telah dibuat. Untuk datanya menggunakan refrensi sebagai berikut:

Tabel 1.1 Tabel pegawai.

NIP	NDep	NBlk	JK	Alamat	Telp	Jabatan	Gaji	NoCab
10107	Emya	Salsalina	P	Jl. Suci 78 Bandung	022-555768	Manajer	5.250.000	C101
10246	Dian	Anggraini	P	Jl. Mawar 5 Semarang	024-555102	*Sales*	2.750.000	C103
10324	Martin	Susanto	L	Jl. Bima 51 Jakarta	021-555785	Staf	1.750.000	C102
10252	Antoni	Irawan	L	Jl. A. Yani 15 Jakarta	021-555888	Manajer	5.750.000	C102
10176	Diah	Wahyuni	P	Jl. Maluku 56 Bandung	022-555934	*Sales*	2.500.000	C101
10314	Ayu	Rahmadani	P	Jl. Malaka 342 Jakarta	021-555098	*Sales*	1.950.000	C102
10307	Erik	Andrian	L	Jl. Manggis 5 Semarang	024-555236	Manajer	6.250.000	C103
10415	Susan	Sumantri	P	Jl. Pahlawan 24 Surabaya	031-555120		2.650.000	C104
10407	Rio	Gunawan	L	Jl. Melati 356 Surabaya	031-555231	Staf	1.725.000	C104

Untuk merealisaskannya kita akan memasukannya dengan perintah `INSERT INTO` kemudiaan diikuti dengan nama tabel dan kemudiaan perintah `VALUES` diikuti

oleh data-datanya.

```
INSERT INTO pegawai (NIP, NDep, NBlk, JK, Alamat, telp, jabatan, Gaji, NoCab)
VALUES (10107, 'Emya', 'Salsalina', 'P', 'Jl. Suci 78 Bandung', '022-555768',
'Manajer', 5250000, 'C101'),
(10246, 'Dian', 'Anggraini', 'P', 'Jl. Mawar 5 Semarang', '024-555102',
'Sales', 2750000, 'C103'),
(10324, 'Martin', 'Susanto', 'L', 'Jl. Bima 51 Jakarta', '021-555888',
'Manajer', 1750000, 'C102'),
(10252, 'Antoni', 'Irawan', 'L', 'Jl. A. Yani 51 Jakarta', '021-555888',
'Manajer', 5750000, 'C102'),
(10176, 'Diah', 'Wahyuni', 'P', 'Jl. Maluku 56 Bandung', '022-555934', 'Sales',
2500000, 'C101'),
(10314, 'Ayu', 'Rahmadani', 'P', 'Jl. Malaka 342 Jakarta', '021-555098',
'Sales', 1950000, 'C102'),
(10307, 'Erik', 'Andrian', 'L', 'Jl. Manggis 5 Semarang', '024-555236',
'Manajer', 6250000, 'C103'),
(10415, 'Susan', 'Sumantri', 'P', 'Jl. Pahlawan 24 Surabaya', '031-555120',
'Manajer', 2650000, 'C104'),
(10407, 'Rio', 'Gunawan', 'L', 'Jl. Melati 356 Surabaya', '031-555231',
'Staff', 1725000, 'C104');
```

Penjelasan:

- `INSERT INTO pegawai (NIP, NDep, NBlk, JK, Alamat, telp, jabatan, Gaji, NoCab)`: Bagian ini mendefinisikan bahwa kita akan memasukkan data ke dalam tabel `pegawai`. Kolom-kolom yang akan diisi adalah `NIP`, `NDep`, `NBlk`, `JK`, `Alamat`, `telp`, `jabatan`, `Gaji`, dan `NoCab`.
- `VALUES`: Kata kunci ini menandakan bahwa nilai-nilai yang akan dimasukkan ke dalam kolom-kolom yang telah disebutkan di atas akan mengikuti setelahnya.
- `(10107, 'Emya', 'Salsalina', 'P', 'Jl. Suci 78 Bandung', '022-555768', 'Manajer', 5250000, 'C101')`: Ini adalah nilai-nilai untuk baris pertama yang akan dimasukkan ke dalam tabel `pegawai`.
  - `10107`: Nilai untuk kolom `NIP`
  - `'Emya'`: Nilai untuk kolom `NDep`
  - `'Salsalina'`: Nilai untuk kolom `NBlk`
  - `'P'`: Nilai untuk kolom `JK`
  - `'Jl. Suci 78 Bandung'`: Nilai untuk kolom `Alamat`
  - `'022-555768'`: Nilai untuk kolom `telp`
  - `'Manajer'`: Nilai untuk kolom `jabatan`
  - `5250000`: Nilai untuk kolom `Gaji`
  - `'C101'`: Nilai untuk kolom `NoCab`
- Setiap baris berikutnya dipisahkan oleh tanda koma `,`, dan memiliki format yang sama dengan baris pertama, dengan nilai-nilai yang sesuai untuk setiap kolom.

Hasil:

```
MariaDB [company_condrado]> INSERT INTO pegawai (NIP, NDep, NBlk, JK, Alamat, telp, jabatan, Gaji, NoCab) VALUES
    -> (10107, 'Emya', 'Salsalina', 'P', 'Jl. Suci 78 Bandung', '022-555768', 'Manajer', 5250000, 'C101'),
    -> (10246, 'Dian', 'Anggraini', 'P', 'Jl. Mawar 5 Semarang', '024-555102', 'Sales', 2750000, 'C103'),
    -> (10324, 'Martin', 'Susanto', 'L', 'Jl. Bima 51 Jakarta', '021-555888', 'Manajer', 1750000, 'C102'),
    -> (10252, 'Antoni', 'Irawan', 'L', 'Jl. A. Yani 51 Jakarta', '021-555888', 'Manajer', 5750000, 'C102'),
    -> (10176, 'Diah', 'Wahyuni', 'P', 'Jl. Maluku 56 Bandung', '022-555934', 'Sales', 2500000, 'C101'),
    -> (10314, 'Ayu', 'Rahmadani', 'P', 'Jl. Malaka 342 Jakarta', '021-555098', 'Sales', 1950000, 'C102'),
    -> (10307, 'Erik', 'Andrian', 'L', 'Jl. Manggis 5 Semarang', '024-555236', 'Manajer', 6250000, 'C103'),
    -> (10415, 'Susan', 'Sumantri', 'P', 'Jl. Pahlawan 24 Surabaya', '031-555120', 'Manajer', 2650000, 'C104'),
    -> (10407, 'Rio', 'Gunawan', 'L', 'Jl. Melati 356 Surabaya', '031-555231', 'Staff', 1725000, 'C104');
Query OK, 9 rows affected (0.01 sec)
Records: 9  Duplicates: 0  Warnings: 0
```

8. Sekarang setelah memasukan data didalam tabel, saatnya untuk melihat tabel dengan data yang sudah dimasukkan didalamnya dengan perintah `SELECT * FROM` kemudian diikuti oleh nama tabelnya.

```
SELECT*FROM pegawai;
```

Penjelasan:

- `SELECT`: Ini adalah kata kunci yang digunakan untuk mengambil data dari database.
- `*`: Tanda bintang (`*`) adalah wildcard yang menunjukkan bahwa semua kolom dalam tabel harus disertakan dalam hasil kueri.
- `FROM`: Kata kunci ini digunakan untuk menunjukkan tabel dari mana data akan diambil.
- `pegawai`: Nama tabel dari mana data akan diambil.

Hasil:

```
MariaDB [company_condrado]> SELECT*FROM pegawai;
+-------+--------+-----------+----+--------------------------+------------+---------+---------+-------+
| NIP   | NDep   | NBlk      | JK | Alamat                   | telp       | jabatan | Gaji    | NoCab |
+-------+--------+-----------+----+--------------------------+------------+---------+---------+-------+
| 10107 | Emya   | Salsalina | P  | Jl. Suci 78 Bandung      | 022-555768 | Manajer | 5250000 | C101  |
| 10176 | Diah   | Wahyuni   | P  | Jl. Maluku 56 Bandung    | 022-555934 | Sales   | 2500000 | C101  |
| 10246 | Dian   | Anggraini | P  | Jl. Mawar 5 Semarang     | 024-555102 | Sales   | 2750000 | C103  |
| 10252 | Antoni | Irawan    | L  | Jl. A. Yani 51 Jakarta   | 021-555888 | Manajer | 5750000 | C102  |
| 10307 | Erik   | Andrian   | L  | Jl. Manggis 5 Semarang   | 024-555236 | Manajer | 6250000 | C103  |
| 10314 | Ayu    | Rahmadani | P  | Jl. Malaka 342 Jakarta   | 021-555098 | Sales   | 1950000 | C102  |
| 10324 | Martin | Susanto   | L  | Jl. Bima 51 Jakarta      | 021-555888 | Manajer | 1750000 | C102  |
| 10407 | Rio    | Gunawan   | L  | Jl. Melati 356 Surabaya  | 031-555231 | Staff   | 1725000 | C104  |
| 10415 | Susan  | Sumantri  | P  | Jl. Pahlawan 24 Surabaya | 031-555120 | Manajer | 2650000 | C104  |
+-------+--------+-----------+----+--------------------------+------------+---------+---------+-------+
9 rows in set (0.01 sec)
```

Menyeleksi Data Berdasarkan Nilai

Untuk menampilkan data atau menyeleksi data berdasarkan nilai tertentu kita dapat mennggunakan perintah `select count` dan untuk membuat visual tabel baru gunakan perintah `as nama_tampilan_tabel`.

Contoh:

```
SELECT COUNT(NIP) AS JumlahPegawai, COUNT(jabatan) AS JumlahJabatan
```

Hasil:

```
MariaDB [company_condrado]> SELECT COUNT(NIP) AS JumlahPegawai, COUNT(jabatan) AS JumlahJabatan
    -> FROM pegawai;
+--------------+--------------+
| JumlahPegawai | JumlahJabatan |
+--------------+--------------+
|             9 |             9 |
+--------------+--------------+
1 row in set (0.00 sec)
```

Penjelasan:

- `SELECT COUNT(NIP) AS JumlahPegawai`:
  - Fungsi `COUNT(NIP)` menghitung data di mana kolom `NIP` tidak bernilai `NULL`.
  - Hasil dari perhitungan ini diberi tabel baru (alias) sebagai `JumlahPegawai`, yang berarti jumlah pegawai yang memiliki NIP.
- `COUNT(jabatan) AS JumlahJabatan`:
  - Fungsi `COUNT(jabatan)` menghitung data di mana kolom `jabatan` tidak bernilai `NULL`.
  - Hasilnya diberi alias sebagai `JumlahJabatan`, yang berarti jumlah jabatan yang terdaftar di dalam tabel.

Menyeleksi Data berdasarkan nilai dan Kriteria tertentu

Untuk menampilkan data atau menyeleksi data berdasarkan nilai dan kriteria tertentu kita dapat mennggunakan perintah `select count` dan untuk kriterianya dalam kasus ini kita gunakan `where kriteria` dan untuk membuat visual tabel baru gunakan perintah `as nama_tampilan_tabel`.

Contoh:

```
SELECT COUNT(NIP) AS JumlahPegawai
FROM pegawai
WHERE NoCab = 'C102';
```

Hasil:

```
MariaDB [company_condrado]> SELECT COUNT(NIP) AS JumlahPegawai
    -> FROM pegawai
    -> WHERE NoCab = 'C102';
+---------------+
| JumlahPegawai |
+---------------+
|             3 |
+---------------+
1 row in set (0.00 sec)
```

Penjelasan:

- `SELECT COUNT(NIP) AS JumlahPegawai`:
  - `SELECT` digunakan untuk memilih kolom atau data yang akan ditampilkan dari hasil query.
  - `COUNT(NIP)` menghitung jumlah baris di mana kolom `NIP` tidak bernilai `NULL`. Ini berarti menghitung jumlah pegawai yang memiliki NIP di dalam hasil filter (yaitu yang bekerja di cabang tertentu).
  - Hasil dari `COUNT(NIP)` diberi alias sebagai `JumlahPegawai`, sehingga hasilnya akan tampil dengan nama kolom `JumlahPegawai`.
- `FROM pegawai`:
  - `FROM` menentukan tabel dari mana data akan diambil. Dalam hal ini, data diambil dari tabel `pegawai`.
- `WHERE NoCab = 'C102'`:
  - `WHERE` digunakan untuk memfilter data berdasarkan kondisi tertentu.
  - `NoCab = 'C102'` adalah kondisi yang memfilter hanya baris-baris yang memiliki nilai `'C102'` di kolom `NoCab`. Dengan kata lain, hanya pegawai yang berada di cabang dengan kode `'C102'` yang akan dihitung.

Count dan Group By

`GROUP BY` adalah kode dalam SQL yang digunakan untuk mengelompokkan baris-baris hasil query berdasarkan satu atau lebih kolom.

Query:

```sql
SELECT NoCab, COUNT(NIP) AS Jumlah_Pegawai
FROM pegawai
GROUP BY NOCab
```

Hasil:

```
MariaDB [company_condrado]> SELECT NoCab, COUNT(NIP) AS Jumlah_Pegawai
    -> FROM pegawai
    -> GROUP BY NoCab;
+-------+----------------+
| NoCab | Jumlah_Pegawai |
+-------+----------------+
| C101  |              2 |
| C102  |              3 |
| C103  |              2 |
| C104  |              2 |
+-------+----------------+
4 rows in set (0.01 sec)
```

Penjelasan:

- `SELECT NoCab, COUNT(NIP) AS Jumlah_Pegawai`:
  - `SELECT` digunakan untuk memilih kolom yang ingin ditampilkan dalam hasil query.
  - `NoCab`: Kolom ini menunjukkan kode cabang. Kolom ini akan ditampilkan sebagai hasil dan menjadi dasar pengelompokan data.
  - `COUNT(NIP) AS Jumlah_Pegawai`: Fungsi `COUNT(NIP)` menghitung jumlah baris di mana kolom `NIP` tidak bernilai `NULL` untuk setiap cabang. Hasilnya diberi alias sebagai `Jumlah_Pegawai`, yang berarti jumlah pegawai di masing-masing cabang.
- `FROM pegawai`:
  - `FROM` menentukan tabel dari mana data akan diambil. Dalam hal ini, data diambil dari tabel `pegawai`.
- `GROUP BY NoCab`:
  - `GROUP BY` digunakan untuk mengelompokkan baris-baris hasil query berdasarkan kolom `NoCab`.
  - Setiap kelompok berisi semua baris dengan nilai `NoCab` yang sama.
  - Setelah dikelompokkan berdasarkan `NoCab`, fungsi `COUNT(NIP)` akan menghitung jumlah pegawai (`NIP`) di setiap cabang (`NoCab`).

GROUP BY DAN HAVING COUNT

`GROUP BY` dan `HAVING` adalah dua klausa yang sering digunakan bersama dalam SQL untuk mengelompokkan data dan kemudian memfilter hasil berdasarkan kondisi tertentu, terutama ketika menggunakan fungsi agregat seperti `COUNT`.

Query:

```sql
SELECT NoCab, COUNT(NIP) AS Jumlah_Pegawai
FROM pegawai
GROUP BY NoCab HAVING COUNT(NIP) ⩾ 3;
```

Hasil:

```
MariaDB [company_condrado]> SELECT NoCab, COUNT(NIP) AS Jumlah_Pegawai
    -> FROM pegawai
    -> GROUP BY NoCab HAVING COUNT(NIP) >= 3;
+-------+----------------+
| NoCab | Jumlah_Pegawai |
+-------+----------------+
| C102  |              3 |
+-------+----------------+
1 row in set (0.00 sec)
```

Penjelasan:

- `SELECT NoCab, COUNT(NIP) AS Jumlah_Pegawai`:
  - `SELECT`: Digunakan untuk memilih kolom yang ingin ditampilkan dalam hasil query.
  - `NoCab`: Kolom yang menunjukkan kode cabang, yang akan ditampilkan dalam hasil.
  - `COUNT(NIP) AS Jumlah_Pegawai`: Fungsi `COUNT(NIP)` menghitung jumlah baris di mana kolom `NIP` tidak bernilai `NULL` dalam setiap grup `NoCab`. Hasilnya diberi alias sebagai `Jumlah_Pegawai`, yang berarti jumlah pegawai di masing-masing cabang.
- `FROM pegawai`:
  - `FROM`: Menentukan tabel dari mana data akan diambil. Dalam hal ini, data diambil dari tabel `pegawai`.
- `GROUP BY NoCab`:
  - `GROUP BY`: Mengelompokkan data berdasarkan nilai `NoCab`, sehingga setiap kelompok terdiri dari semua baris yang memiliki nilai `NoCab` yang sama.
  - Setelah data dikelompokkan, fungsi `COUNT(NIP)` akan menghitung jumlah pegawai (`NIP`) di setiap kelompok `NoCab`.
- `HAVING COUNT(NIP) ⩾ 3`:
  - `HAVING`: Digunakan untuk memfilter hasil setelah data dikelompokkan.
  - `COUNT(NIP) ⩾ 3`: Kondisi ini memfilter kelompok-kelompok yang telah dibentuk oleh `GROUP BY` sehingga hanya kelompok yang memiliki `COUNT(NIP)` (jumlah pegawai) lebih besar atau sama dengan 3 yang akan ditampilkan dalam hasil akhir.

SELECT SUM

`SUM` adalah fungsi agregat dalam SQL yang digunakan untuk menjumlahkan nilai-nilai numerik dalam suatu kolom. Fungsi ini menghitung total dari semua nilai yang dipilih, baik dalam seluruh tabel atau dalam kelompok tertentu jika digunakan bersama dengan `GROUP BY`.

Query:

```
SELECT SUM(Gaji) AS Total_Gaji
FROM pegawai;
```

Hasil:

```
MariaDB [company_condrado]> SELECT SUM(Gaji) AS Total_Gaji
    -> FROM pegawai;
+------------+
| Total_Gaji |
+------------+
|   30575000 |
+------------+
1 row in set (0.00 sec)
```

Penjelasan:

- `SELECT SUM(Gaji) AS Total_Gaji`:
  - `SELECT`: Digunakan untuk memilih kolom atau data yang ingin ditampilkan dari hasil query.
  - `SUM(Gaji)`: Fungsi agregat `SUM()` menjumlahkan semua nilai dalam kolom `Gaji`. Ini berarti bahwa total dari semua gaji pegawai akan dihitung.
  - `AS Total_Gaji`: Memberi alias pada hasil perhitungan `SUM(Gaji)`, sehingga hasilnya akan ditampilkan dengan nama kolom `Total_Gaji`
- `FROM pegawai`:
  - `FROM`: Menentukan tabel dari mana data akan diambil. Dalam hal ini, data diambil dari tabel `pegawai`.

Lanjutan Select Sum

Query:

```
SELECT SUM(Gaji) AS Gaji_Manajer
FROM pegawai
WHERE jabatan = 'Manajer';
```

Hasil:

```
MariaDB [company_condrado]> SELECT SUM(Gaji) AS Gaji_Manajer
    -> FROM pegawai
    -> WHERE jabatan = 'Manajer';
+--------------+
| Gaji_Manajer |
+--------------+
|     21650000 |
+--------------+
1 row in set (0.00 sec)
```

Penjelasan:

- `SELECT SUM(Gaji) AS Gaji_Manajer`:
  - `SELECT`: Digunakan untuk memilih kolom atau data yang ingin ditampilkan dari hasil query.
  - `SUM(Gaji)`: Fungsi agregat `SUM()` menjumlahkan semua nilai dalam kolom `Gaji`. Dalam hal ini, hanya gaji pegawai yang memenuhi kondisi tertentu (yaitu, pegawai yang memiliki jabatan "Manajer") yang akan dijumlahkan.
  - `AS Gaji_Manajer`: Memberi alias pada hasil perhitungan `SUM(Gaji)`, sehingga hasilnya akan ditampilkan dengan nama kolom `Gaji_Manajer`.
- `FROM pegawai`:
  - `FROM`: Menentukan tabel dari mana data akan diambil. Dalam hal ini, data diambil dari tabel `pegawai`.
- `WHERE jabatan = 'Manajer'`:
  - `WHERE`: Digunakan untuk memfilter data berdasarkan kondisi tertentu.
  - `jabatan = 'Manajer'`: Kondisi ini memastikan bahwa hanya baris-baris yang memiliki nilai "Manajer" pada kolom `jabatan` yang akan disertakan dalam perhitungan `SUM(Gaji)`.

Gabungan Sum dan Group By

Query:

```sql
SELECT NoCab, SUM(Gaji) AS TotalGaji
FROM pegawai
GROUP BY NoCab
```

Hasil:

```
MariaDB [company_condrado]> SELECT NoCab, SUM(Gaji) AS TotalGaji
    -> FROM pegawai
    -> GROUP BY NoCab;
+-------+-----------+
| NoCab | TotalGaji |
+-------+-----------+
| C101  |   7750000 |
| C102  |   9450000 |
| C103  |   9000000 |
| C104  |   4375000 |
+-------+-----------+
4 rows in set (0.00 sec)
```

Penjelasan:

1. `SELECT NoCab, SUM(Gaji) AS TotalGaji`:
   - `SELECT`: Digunakan untuk memilih kolom yang ingin ditampilkan dalam hasil query.
   - `NoCab`: Kolom yang menunjukkan kode cabang. Kolom ini akan ditampilkan sebagai hasil dan digunakan untuk pengelompokan data.
   - `SUM(Gaji)`: Fungsi agregat `SUM()` menjumlahkan semua nilai dalam kolom `Gaji` untuk setiap cabang. Ini berarti bahwa total gaji dari semua pegawai di setiap cabang akan dihitung.
   - `AS TotalGaji`: Memberi alias pada hasil perhitungan `SUM(Gaji)`, sehingga hasilnya akan ditampilkan dengan nama kolom `TotalGaji`.
2. `FROM pegawai`:
   - `FROM`: Menentukan tabel dari mana data akan diambil. Dalam hal ini, data diambil dari tabel `pegawai`.
3. `GROUP BY NoCab`:
   - `GROUP BY`: Mengelompokkan data berdasarkan nilai `NoCab`, sehingga setiap kelompok terdiri dari semua baris yang memiliki nilai `NoCab` yang sama.
   - Setelah data dikelompokkan, fungsi `SUM(Gaji)` akan menghitung total gaji untuk setiap kelompok `NoCab`.

ss

Query:

```sql
SELECT NoCab, SUM(Gaji) AS TotalGaji
FROM pegawai
GROUP BY NoCab HAVING SUM(Gaji) ⩾ 8000000;
```

Hasil:

```
MariaDB [company_condrado]> SELECT NoCab, SUM(Gaji) AS TotalGaji
    -> FROM pegawai
    -> GROUP BY NoCab HAVING SUM(Gaji) >= 8000000;
+-------+-----------+
| NoCab | TotalGaji |
+-------+-----------+
| C102  |   9450000 |
| C103  |   9000000 |
+-------+-----------+
2 rows in set (0.00 sec)
```

Penjelasan:

- `SELECT NoCab, SUM(Gaji) AS TotalGaji`:
  - `SELECT`: Digunakan untuk memilih kolom atau data yang ingin ditampilkan dalam hasil query.
  - `NoCab`: Kolom yang menunjukkan kode cabang. Kolom ini akan ditampilkan dalam hasil query dan digunakan untuk pengelompokan data.
  - `SUM(Gaji)`: Fungsi agregat `SUM()` menjumlahkan semua nilai dalam kolom `Gaji` untuk setiap cabang. Ini berarti bahwa total gaji dari semua pegawai di setiap cabang akan dihitung.
  - `AS TotalGaji`: Memberi alias pada hasil perhitungan `SUM(Gaji)`, sehingga hasilnya akan ditampilkan dengan nama kolom `TotalGaji`.
- `FROM pegawai`:
  - `FROM`: Menentukan tabel dari mana data akan diambil. Dalam hal ini, data diambil dari tabel `pegawai`.
- `GROUP BY NoCab`:
  - `GROUP BY`: Mengelompokkan data berdasarkan nilai `NoCab`. Setiap grup terdiri dari baris-baris yang memiliki nilai `NoCab` yang sama. Setelah data dikelompokkan, fungsi `SUM(Gaji)` akan menghitung total gaji untuk setiap kelompok `NoCab`.
- `HAVING SUM(Gaji) ⩾ 8000000`:
  - `HAVING`: Digunakan untuk memfilter hasil setelah data dikelompokkan.
  - `SUM(Gaji) ⩾ 8000000`: Kondisi ini memfilter kelompok-kelompok yang telah dibentuk oleh `GROUP BY` sehingga hanya kelompok yang memiliki `SUM(Gaji)` (total gaji) lebih besar atau sama dengan 8.000.000 yang akan ditampilkan dalam hasil akhir.

Select AVG

`AVG` adalah fungsi agregat dalam SQL yang digunakan untuk menghitung rata-rata nilai dari suatu kolom numerik. Fungsi ini menambahkan semua nilai

dalam kolom yang dipilih dan membaginya dengan jumlah baris yang ada (yang nilainya tidak null) untuk menghasilkan rata-rata.

Query:

```
SELECT AVG(Gaji) AS Rata_rata
FROM pegawai;
```

Hasil:

```
MariaDB [company_condrado]> SELECT AVG(Gaji) AS Rata_rata
    -> FROM pegawai;
+--------------+
| Rata_rata    |
+--------------+
| 3397222.2222 |
+--------------+
1 row in set (0.00 sec)
```

Penjelasan:

- `SELECT AVG(Gaji) AS Rata_rata`:
  - `SELECT`: Digunakan untuk memilih kolom atau hasil perhitungan yang ingin ditampilkan dari query.
  - `AVG(Gaji)`: Fungsi agregat `AVG()` menghitung rata-rata nilai dari kolom `Gaji`. Ini berarti bahwa nilai-nilai gaji dari semua pegawai akan dijumlahkan dan kemudian dibagi dengan jumlah pegawai untuk mendapatkan rata-ratanya.
  - `AS Rata_rata`: Memberi alias pada hasil perhitungan `AVG(Gaji)`, sehingga hasilnya akan ditampilkan dengan nama kolom `Rata_rata`. Ini mempermudah pembacaan hasil query.
- `FROM pegawai`:
  - `FROM`: Menentukan tabel dari mana data akan diambil. Dalam hal ini, data diambil dari tabel `pegawai`.

Lanjutan Select AVG

Query:

```
SELECT AVG(Gaji) AS Rata_rataMGr
FROM pegawai
WHERE jabatan = 'Manajer';
```

Hasil:

```
MariaDB [company_condrado]> SELECT AVG(Gaji) AS Rata_rataMgr
    -> FROM pegawai
    -> WHERE jabatan = 'Manajer';
+--------------+
| Rata_rataMgr |
+--------------+
| 4330000.0000 |
+--------------+
1 row in set (0.00 sec)
```

Penjelasan:

- `SELECT AVG(Gaji) AS Rata_rataMGr`:
  - `SELECT`: Digunakan untuk memilih kolom atau hasil perhitungan yang ingin ditampilkan dari query.
  - `AVG(Gaji)`: Fungsi agregat `AVG()` menghitung rata-rata nilai dari kolom `Gaji`. Dalam hal ini, hanya gaji dari pegawai yang memiliki jabatan "Manajer" yang akan dihitung rata-ratanya.
  - `AS Rata_rataMGr`: Memberi alias pada hasil perhitungan `AVG(Gaji)`, sehingga hasilnya akan ditampilkan dengan nama kolom `Rata_rataMGr`. Ini mempermudah pembacaan hasil query, yang menunjukkan bahwa rata-rata gaji ini khusus untuk manajer.
- `FROM pegawai`:
  - `FROM`: Menentukan tabel dari mana data akan diambil. Dalam hal ini, data diambil dari tabel `pegawai`.
- `WHERE jabatan = 'Manajer'`:
  - `WHERE`: Digunakan untuk memfilter data yang akan diproses oleh query.
  - `jabatan = 'Manajer'`: Kondisi ini memastikan bahwa hanya baris-baris yang memiliki nilai "Manajer" pada kolom `jabatan` yang akan disertakan dalam perhitungan rata-rata gaji.

AVG dan Group By

Query:

```sql
SELECT NoCab, AVG(Gaji) AS RataGaji
FROM pegawai
GROUP BY NoCAb
```

Hasil:

```
MariaDB [company_condrado]> SELECT NoCab, AVG(Gaji) AS RataGaji
    -> FROM pegawai
    -> GROUP BY NoCab;
+-------+--------------+
| NoCab | RataGaji     |
+-------+--------------+
| C101  | 3875000.0000 |
| C102  | 3150000.0000 |
| C103  | 4500000.0000 |
| C104  | 2187500.0000 |
+-------+--------------+
4 rows in set (0.01 sec)
```

Penjelasan:

- `SELECT NoCab, AVG(Gaji) AS RataGaji`:
  - `SELECT`: Digunakan untuk memilih kolom atau hasil perhitungan yang ingin ditampilkan dari query.
  - `NoCab`: Kolom yang menunjukkan kode cabang. Kolom ini akan ditampilkan dalam hasil query dan digunakan untuk pengelompokan data.
  - `AVG(Gaji)`: Fungsi agregat `AVG()` menghitung rata-rata nilai dari kolom `Gaji` untuk setiap cabang. Ini berarti bahwa rata-rata gaji dari semua pegawai di setiap cabang akan dihitung.
  - `AS RataGaji`: Memberi alias pada hasil perhitungan `AVG(Gaji)`, sehingga hasilnya akan ditampilkan dengan nama kolom `RataGaji`. Ini mempermudah pembacaan hasil query.
- `FROM pegawai`:
  - `FROM`: Menentukan tabel dari mana data akan diambil. Dalam hal ini, data diambil dari tabel `pegawai`.
- `GROUP BY NoCab`:
  - `GROUP BY`: Mengelompokkan data berdasarkan nilai `NoCab`. Setiap grup terdiri dari baris-baris yang memiliki nilai `NoCab` yang sama. Setelah data dikelompokkan, fungsi `AVG(Gaji)` akan menghitung rata-rata gaji untuk setiap kelompok `NoCab`.

Lanjutan AVG, Having, Group By

Query:

```sql
SELECT NoCab, AVG(Gaji) AS RataGaji
FROM pegawai
GROUP BY NoCab HAVING NoCab = 'C101' OR NoCab = 'C102';
```

Hasil:

```
MariaDB [company_condrado]> SELECT NoCab, AVG(Gaji) AS RataGaji
    -> FROM pegawai
    -> GROUP BY NoCab HAVING NoCab = 'C101' OR NoCab = 'C102';
+-------+--------------+
| NoCab | RataGaji     |
+-------+--------------+
| C101  | 3875000.0000 |
| C102  | 3150000.0000 |
+-------+--------------+
2 rows in set (0.00 sec)
```

Penjelasan:

- `SELECT NoCab, AVG(Gaji) AS RataGaji`:
  - `SELECT`: Digunakan untuk memilih kolom atau hasil perhitungan yang ingin ditampilkan dari query.
  - `NoCab`: Kolom yang menunjukkan kode cabang. Kolom ini akan ditampilkan dalam hasil query dan digunakan untuk pengelompokan data.
  - `AVG(Gaji)`: Fungsi agregat `AVG()` menghitung rata-rata nilai dari kolom `Gaji` untuk setiap cabang. Ini berarti bahwa rata-rata gaji dari semua pegawai di setiap cabang yang dipilih akan dihitung.
  - `AS RataGaji`: Memberi alias pada hasil perhitungan `AVG(Gaji)`, sehingga hasilnya akan ditampilkan dengan nama kolom `RataGaji`. Ini mempermudah pembacaan hasil query.
- `FROM pegawai`:
  - `FROM`: Menentukan tabel dari mana data akan diambil. Dalam hal ini, data diambil dari tabel `pegawai`.
- `GROUP BY NoCab`:
  - `GROUP BY`: Mengelompokkan data berdasarkan nilai `NoCab`. Setiap grup terdiri dari baris-baris yang memiliki nilai `NoCab` yang sama. Setelah data dikelompokkan, fungsi `AVG(Gaji)` akan menghitung rata-rata gaji untuk setiap kelompok `NoCab`.
- `HAVING NoCab = 'C101' OR NoCab = 'C102'`:
  - `HAVING`: Digunakan untuk memfilter hasil setelah data dikelompokkan.
  - `NoCab = 'C101' OR NoCab = 'C102'`: Kondisi ini memastikan bahwa hanya kelompok cabang `C101` atau `C102` yang akan ditampilkan dalam hasil. `HAVING` digunakan untuk memfilter kelompok hasil dari `GROUP BY`.

MAX DAN MIN

Dalam SQL, fungsi `MAX` dan `MIN` adalah fungsi agregat yang digunakan untuk menghitung nilai maksimum dan minimum dari sebuah kolom dalam dataset.

Query:

```
SELECT MAX(Gaji) AS GajiTerbesar, MIN(Gaji) AS GajiTerkecil
FROM pegawai;
```

Hasil:

```
MariaDB [company_condrado]> SELECT MAX(Gaji) AS GajiTerbesar, MIN(Gaji) AS GajiTerkecil
    -> FROM pegawai;
+--------------+--------------+
| GajiTerbesar | GajiTerkecil |
+--------------+--------------+
|      6250000 |      1725000 |
+--------------+--------------+
1 row in set (0.00 sec)
```

Penjelasan:

- `SELECT MAX(Gaji) AS GajiTerbesar, MIN(Gaji) AS GajiTerkecil`:
  - `SELECT`: Digunakan untuk memilih kolom atau hasil perhitungan yang ingin ditampilkan dari query.
  - `MAX(Gaji)`: Fungsi agregat `MAX()` digunakan untuk mencari nilai maksimum dari kolom `Gaji`, yaitu gaji terbesar di antara semua pegawai.
  - `AS GajiTerbesar`: Memberi alias pada hasil perhitungan `MAX(Gaji)`, sehingga hasilnya akan ditampilkan dengan nama kolom `GajiTerbesar`.
  - `MIN(Gaji)`: Fungsi agregat `MIN()` digunakan untuk mencari nilai minimum dari kolom `Gaji`, yaitu gaji terkecil di antara semua pegawai.
  - `AS GajiTerkecil`: Memberi alias pada hasil perhitungan `MIN(Gaji)`, sehingga hasilnya akan ditampilkan dengan nama kolom `GajiTerkecil`.
- `FROM pegawai`:
  - `FROM`: Menentukan tabel dari mana data akan diambil. Dalam hal ini, data diambil dari tabel `pegawai`.

LANJUTA MAX DAN MIN

Query:

```
SELECT MAX(Gaji) AS GajiTerbesar, MIN(Gaji) AS GajiTerkecil
FROM pegawai
WHERE jabatan = 'Manajer';
```

Hasil:

```
MariaDB [company_condrado]> SELECT MAX(Gaji) AS GajiTerbesar, MIN(Gaji) AS GajiTerkecil
    -> FROM pegawai
    -> WHERE jabatan = 'Manajer';
+--------------+--------------+
| GajiTerbesar | GajiTerkecil |
+--------------+--------------+
|      6250000 |      1750000 |
+--------------+--------------+
1 row in set (0.00 sec)
```

Penjelasan:

- `SELECT MAX(Gaji) AS GajiTerbesar, MIN(Gaji) AS GajiTerkecil`:
  - `SELECT`: Digunakan untuk memilih kolom atau hasil perhitungan yang ingin ditampilkan dari query.
  - `MAX(Gaji)`: Fungsi agregat `MAX()` digunakan untuk mencari nilai maksimum dari kolom `Gaji`, yaitu gaji terbesar di antara semua pegawai dengan jabatan "Manajer".
  - `AS GajiTerbesar`: Memberi alias pada hasil perhitungan `MAX(Gaji)`, sehingga hasilnya akan ditampilkan dengan nama kolom `GajiTerbesar`.
  - `MIN(Gaji)`: Fungsi agregat `MIN()` digunakan untuk mencari nilai minimum dari kolom `Gaji`, yaitu gaji terkecil di antara semua pegawai dengan jabatan "Manajer".
  - `AS GajiTerkecil`: Memberi alias pada hasil perhitungan `MIN(Gaji)`, sehingga hasilnya akan ditampilkan dengan nama kolom `GajiTerkecil`.
- `FROM pegawai`:
  - `FROM`: Menentukan tabel dari mana data akan diambil. Dalam hal ini, data diambil dari tabel `pegawai`.
- `WHERE jabatan = 'Manajer'`:
  - `WHERE`: Digunakan untuk memfilter data yang akan diproses oleh query.
  - `jabatan = 'Manajer'`: Kondisi ini memastikan bahwa hanya baris-baris yang memiliki nilai "Manajer" pada kolom `jabatan` yang akan disertakan dalam perhitungan gaji terbesar dan terkecil.

MAX MIN DAN GROUP BY

Query:

```sql
SELECT NoCab, MAX(Gaji) AS GajiTerbesar, MIN(Gaji) AS GajiTerkecil
FROM pegawai
GROUP BY NoCab;
```

Hasil:

```
MariaDB [company_condrado]> SELECT NoCab, MAX(Gaji) AS GajiTerbesar, MIN(Gaji) AS GajiTerkecil
    -> FROM pegawai
    -> GROUP BY NoCab;\
+-------+--------------+--------------+
| NoCab | GajiTerbesar | GajiTerkecil |
+-------+--------------+--------------+
| C101  |      5250000 |      2500000 |
| C102  |      5750000 |      1750000 |
| C103  |      6250000 |      2750000 |
| C104  |      2650000 |      1725000 |
+-------+--------------+--------------+
4 rows in set (0.00 sec)
```

Penjelasan:

- `SELECT NoCab, MAX(Gaji) AS GajiTerbesar, MIN(Gaji) AS GajiTerkecil`:
  - `SELECT`: Digunakan untuk memilih kolom atau hasil perhitungan yang ingin ditampilkan dari query.
  - `NoCab`: Kolom yang menunjukkan kode cabang. Kolom ini akan ditampilkan dalam hasil query dan digunakan untuk pengelompokan data.
  - `MAX(Gaji)`: Fungsi agregat `MAX()` menghitung nilai maksimum dari kolom `Gaji` untuk setiap cabang. Ini berarti gaji terbesar di setiap cabang akan dihitung.
  - `AS GajiTerbesar`: Memberi alias pada hasil perhitungan `MAX(Gaji)`, sehingga hasilnya akan ditampilkan dengan nama kolom `GajiTerbesar`.
  - `MIN(Gaji)`: Fungsi agregat `MIN()` menghitung nilai minimum dari kolom `Gaji` untuk setiap cabang. Ini berarti gaji terkecil di setiap cabang akan dihitung.
  - `AS GajiTerkecil`: Memberi alias pada hasil perhitungan `MIN(Gaji)`, sehingga hasilnya akan ditampilkan dengan nama kolom `GajiTerkecil`.
- `FROM pegawai`:
  - `FROM`: Menentukan tabel dari mana data akan diambil. Dalam hal ini, data diambil dari tabel `pegawai`.
- `GROUP BY NoCab`:
  - `GROUP BY`: Mengelompokkan data berdasarkan nilai `NoCab`. Setiap grup terdiri dari baris-baris yang memiliki nilai `NoCab` yang sama. Fungsi agregat seperti `MAX()` dan `MIN()` kemudian diterapkan pada setiap grup.

LANJUTAN MAX MIN DAN GROUP BY

Query:

```sql
SELECT NoCab, MAX(Gaji) AS GajiTerbesar, MIN(Gaji) AS GajiTerkecil
FROM pegawai
GROUP BY NoCab HAVING COUNT(NIP) >=3;
```

Hasil:

```
MariaDB [company_condrado]> SELECT NoCab, MAX(Gaji) AS GajiTerbesar, MIN(Gaji) AS GajiTerkecil
    -> FROM pegawai
    -> GROUP BY NoCab HAVING COUNT(NIP) >=3;
+-------+--------------+--------------+
| NoCab | GajiTerbesar | GajiTerkecil |
+-------+--------------+--------------+
| C102  |      5750000 |      1750000 |
+-------+--------------+--------------+
1 row in set (0.00 sec)
```

Penjelasan:

- `SELECT NoCab, MAX(Gaji) AS GajiTerbesar, MIN(Gaji) AS GajiTerkecil`:
  - `SELECT`: Digunakan untuk memilih kolom dan fungsi agregat yang akan ditampilkan dalam hasil query.
  - `NoCab`: Kolom yang menunjukkan kode cabang. Ini adalah kolom yang digunakan untuk pengelompokan dan juga akan ditampilkan dalam hasil.
  - `MAX(Gaji)`: Fungsi agregat `MAX()` menghitung nilai maksimum dari kolom `Gaji` untuk setiap grup `NoCab`. Ini memberi gaji terbesar di setiap cabang.
  - `AS GajiTerbesar`: Memberi alias `GajiTerbesar` pada hasil dari `MAX(Gaji)`, sehingga nama kolom dalam hasil query adalah `GajiTerbesar`.
  - `MIN(Gaji)`: Fungsi agregat `MIN()` menghitung nilai minimum dari kolom `Gaji` untuk setiap grup `NoCab`. Ini memberi gaji terkecil di setiap cabang.
  - `AS GajiTerkecil`: Memberi alias `GajiTerkecil` pada hasil dari `MIN(Gaji)`, sehingga nama kolom dalam hasil query adalah `GajiTerkecil`.
- `FROM pegawai`:
  - `FROM`: Menentukan tabel dari mana data akan diambil. Dalam hal ini, data diambil dari tabel `pegawai`.
- `GROUP BY NoCab`:
  - `GROUP BY`: Mengelompokkan data berdasarkan kolom `NoCab`. Setiap grup terdiri dari baris-baris dengan nilai `NoCab` yang sama. Fungsi agregat seperti `MAX()` dan `MIN()` kemudian diterapkan pada setiap grup.
- `HAVING COUNT(NIP) ⩾ 3`:
  - `HAVING`: Memfilter hasil setelah pengelompokan yang dilakukan oleh `GROUP BY`. Berbeda dengan `WHERE`, yang memfilter baris sebelum pengelompokan, `HAVING` memfilter hasil dari `GROUP BY`.
  - `COUNT(NIP) ⩾ 3`: Menghitung jumlah pegawai dalam setiap grup `NoCab` dan memastikan hanya grup (cabang) yang memiliki setidaknya 3

pegawai yang ditampilkan dalam hasil. Fungsi `COUNT(NIP)` menghitung jumlah pegawai di setiap cabang.

LANJUTAN MAX MIN DAN AS

Query:

```
SELECT COUNT(NIP) AS JumlahPegawai, SUM(Gaji) AS TotalGaji,
AVG(Gaji) AS RataGaji, MAX(Gaji) AS GajiMaks, MIN(Gaji) AS GajiMin
FROM pegawai
```

Hasil:

```
MariaDB [company_condrado]> SELECT COUNT(NIP) AS JumlahPegawai, SUM(Gaji) AS TotalGaji,
    -> AVG(Gaji) AS RataGaji, MAX(Gaji) AS GajiMaks,MIN(Gaji) AS GajiMin
    -> FROM pegawai;
+---------------+-----------+--------------+----------+---------+
| JumlahPegawai | TotalGaji | RataGaji     | GajiMaks | GajiMin |
+---------------+-----------+--------------+----------+---------+
|             9 |  30575000 | 3397222.2222 |  6250000 | 1725000 |
+---------------+-----------+--------------+----------+---------+
1 row in set (0.00 sec)
```

Penjelasan:

- `SELECT`:
  - Kata kunci ini digunakan untuk menentukan kolom atau hasil perhitungan yang akan ditampilkan dalam hasil query.
- `COUNT(NIP) AS JumlahPegawai`:
  - `COUNT(NIP)`: Fungsi agregat `COUNT()` menghitung jumlah baris yang tidak bernilai `NULL` dalam kolom `NIP`. Karena `NIP` biasanya merupakan nomor identifikasi unik pegawai, fungsi ini menghitung jumlah total pegawai dalam tabel.
  - `AS JumlahPegawai`: Memberi alias `JumlahPegawai` pada hasil dari `COUNT(NIP)`, sehingga kolom ini akan muncul dengan nama `JumlahPegawai` dalam hasil query.
- `SUM(Gaji) AS TotalGaji`:
  - `SUM(Gaji)`: Fungsi agregat `SUM()` menjumlahkan semua nilai dalam kolom `Gaji`. Ini menghitung total gaji yang dibayarkan kepada semua pegawai yang ada dalam tabel.
  - `AS TotalGaji`: Memberi alias `TotalGaji` pada hasil dari `SUM(Gaji)`, sehingga kolom ini akan muncul dengan nama `TotalGaji` dalam hasil query.
- `AVG(Gaji) AS RataGaji`:
  - `AVG(Gaji)`: Fungsi agregat `AVG()` menghitung rata-rata nilai dalam kolom `Gaji`. Ini memberikan rata-rata gaji yang diterima oleh pegawai.

    - `AS RataGaji`: Memberi alias `RataGaji` pada hasil dari `AVG(Gaji)`, sehingga kolom ini akan muncul dengan nama `RataGaji` dalam hasil query.
- `MAX(Gaji) AS GajiMaks`:
    - `MAX(Gaji)`: Fungsi agregat `MAX()` mencari nilai maksimum dalam kolom `Gaji`. Ini memberikan informasi tentang gaji tertinggi yang diterima oleh seorang pegawai.
    - `AS GajiMaks`: Memberi alias `GajiMaks` pada hasil dari `MAX(Gaji)`, sehingga kolom ini akan muncul dengan nama `GajiMaks` dalam hasil query.
- `MIN(Gaji) AS GajiMin`:
    - `MIN(Gaji)`: Fungsi agregat `MIN()` mencari nilai minimum dalam kolom `Gaji`. Ini memberikan informasi tentang gaji terendah yang diterima oleh seorang pegawai.
    - `AS GajiMin`: Memberi alias `GajiMin` pada hasil dari `MIN(Gaji)`, sehingga kolom ini akan muncul dengan nama `GajiMin` dalam hasil query.
- `FROM pegawai`:
    - `FROM`: Menentukan tabel dari mana data akan diambil. Dalam hal ini, data diambil dari tabel `pegawai`.

LANJUTAN MAX MIN GROUP BY HAVING

Query:

```
SELECT COUNT(NIP) AS JumlahPegawai, SUM(Gaji) AS TotalGaji,
AVG(Gaji) AS RataGaji, MAX(Gaji) AS GajiMaks, MIN(Gaji) AS GajiMin
FROM pegawai
WHERE jabatan = `Staff` OR jabatan = 'Sales'
GROUP BY NoCab HAVING SUM(Gaji) ⩽ 26000000
```

Hasil:

```
MariaDB [company_condrado]> SELECT COUNT(NIP) AS JumlahPegawai, SUM(Gaji) AS TotalGaji,
    -> AVG(Gaji) AS RataGaji, MAX(Gaji) AS GajiMaks, MIN(Gaji) AS GajiMin
    -> FROM pegawai
    -> WHERE jabatan = 'Staff' OR jabatan = 'Sales'
    -> GROUP BY NoCab HAVING SUM(Gaji) <= 2600000;
+---------------+-----------+--------------+----------+---------+
| JumlahPegawai | TotalGaji | RataGaji     | GajiMaks | GajiMin |
+---------------+-----------+--------------+----------+---------+
|             1 |   2500000 | 2500000.0000 |  2500000 | 2500000 |
|             1 |   1950000 | 1950000.0000 |  1950000 | 1950000 |
|             1 |   1725000 | 1725000.0000 |  1725000 | 1725000 |
+---------------+-----------+--------------+----------+---------+
3 rows in set (0.01 sec)
```

Penjelasan:

- `SELECT`:
  - Kata kunci ini digunakan untuk menentukan kolom atau hasil perhitungan yang akan ditampilkan dalam hasil query.
- `COUNT(NIP) AS JumlahPegawai`:
  - `COUNT(NIP)`: Fungsi agregat `COUNT()` menghitung jumlah baris yang tidak bernilai `NULL` dalam kolom `NIP`. Karena `NIP` biasanya merupakan nomor identifikasi unik pegawai, fungsi ini menghitung jumlah total pegawai yang memiliki jabatan `Staff` atau `Sales` di setiap cabang (`NoCab`).
  - `AS JumlahPegawai`: Memberi alias `JumlahPegawai` pada hasil dari `COUNT(NIP)`, sehingga kolom ini akan muncul dengan nama `JumlahPegawai` dalam hasil query.
- `SUM(Gaji) AS TotalGaji`:
  - `SUM(Gaji)`: Fungsi agregat `SUM()` menjumlahkan semua nilai dalam kolom `Gaji` untuk pegawai yang memiliki jabatan `Staff` atau `Sales` di setiap cabang (`NoCab`). Ini menghitung total gaji yang dibayarkan kepada pegawai dalam jabatan tersebut di setiap cabang.
  - `AS TotalGaji`: Memberi alias `TotalGaji` pada hasil dari `SUM(Gaji)`, sehingga kolom ini akan muncul dengan nama `TotalGaji` dalam hasil query.
- `AVG(Gaji) AS RataGaji`:
  - `AVG(Gaji)`: Fungsi agregat `AVG()` menghitung rata-rata nilai dalam kolom `Gaji` untuk pegawai yang memiliki jabatan `Staff` atau `Sales` di setiap cabang (`NoCab`). Ini memberikan rata-rata gaji yang diterima oleh pegawai dalam jabatan tersebut di setiap cabang.
  - `AS RataGaji`: Memberi alias `RataGaji` pada hasil dari `AVG(Gaji)`, sehingga kolom ini akan muncul dengan nama `RataGaji` dalam hasil query.
- `MAX(Gaji) AS GajiMaks`:
  - `MAX(Gaji)`: Fungsi agregat `MAX()` mencari nilai maksimum dalam kolom `Gaji` untuk pegawai yang memiliki jabatan `Staff` atau `Sales` di setiap cabang (`NoCab`). Ini memberikan informasi tentang gaji tertinggi yang diterima oleh seorang pegawai dalam jabatan tersebut di setiap cabang.
  - `AS GajiMaks`: Memberi alias `GajiMaks` pada hasil dari `MAX(Gaji)`, sehingga kolom ini akan muncul dengan nama `GajiMaks` dalam hasil query.
- `MIN(Gaji) AS GajiMin`:
  - `MIN(Gaji)`: Fungsi agregat `MIN()` mencari nilai minimum dalam kolom `Gaji` untuk pegawai yang memiliki jabatan `Staff` atau `Sales` di setiap cabang (`NoCab`). Ini memberikan informasi tentang gaji terendah

yang diterima oleh seorang pegawai dalam jabatan tersebut di setiap cabang.
    - `AS GajiMin`: Memberi alias `GajiMin` pada hasil dari `MIN(Gaji)`, sehingga kolom ini akan muncul dengan nama `GajiMin` dalam hasil query.
- `FROM pegawai`:
    - `FROM`: Menentukan tabel dari mana data akan diambil. Dalam hal ini, data diambil dari tabel `pegawai`.
- `WHERE jabatan = 'Staff' OR jabatan = 'Sales'`:
    - `WHERE`: Digunakan untuk memfilter data sebelum pengelompokan. Hanya baris-baris yang memenuhi kondisi dalam `WHERE` yang akan diproses lebih lanjut.
    - `jabatan = 'Staff' OR jabatan = 'Sales'`: Kondisi ini memastikan hanya pegawai yang memiliki jabatan `Staff` atau `Sales` yang akan disertakan dalam perhitungan.
- `GROUP BY NoCab`:
    - `GROUP BY`: Mengelompokkan data berdasarkan kolom `NoCab`. Setiap grup terdiri dari baris-baris yang memiliki nilai `NoCab` yang sama. Fungsi agregat kemudian diterapkan pada setiap grup.
    - Setelah data difilter menggunakan `WHERE`, data yang tersisa dikelompokkan berdasarkan cabang (`NoCab`).
- `HAVING SUM(Gaji) ⩽ 26000000`:
    - `HAVING`: Digunakan untuk memfilter hasil setelah pengelompokan yang dilakukan oleh `GROUP BY`. Berbeda dengan `WHERE`, yang memfilter baris sebelum pengelompokan, `HAVING` memfilter hasil dari `GROUP BY`.
    - `SUM(Gaji) ⩽ 26000000`: Kondisi ini memastikan bahwa hanya cabang-cabang yang memiliki total gaji kurang dari atau sama dengan 26.000.000 yang akan ditampilkan dalam hasil.

No	Nama	Nilai	Tugas
1.	Condrado	3	Membuat Catatan
2.	Adrian	3	Membantu Membuat Catatan
3.	Fahri Ilham	3	Membantu membuat Catatan dan mengurus upload file ke drive
4.	Muh. Nabil Maulana	2	Penyemangat
5.	Yeremia Tasik	2	Membantu merevisi catatan