# PRAKTIKUM SISTEM BASIS DATA

# Modul 7: SQL Aggregate

**Deskripsi:**

Modul ini berisi materi mengenai bagaimana mendapatkan perhitungan agregasi menggunakan SQL. Penggunaan sintaks agregasi juga seringkali mengharuskan proses pengelompokan *(grouping)* dari record-record di tabel. Mahasiswa akan diajak untuk mahir menghitung agregasi baik untuk data yang harus dikelompokkan maupun tidak.

**Tujuan Praktikum:**

1. Mahasiswa dapat menjelaskan berbagai macam operasi agregasi yang bisa dilakukan antar terhadap suatu tabel dengan komposisi kolom tertentu.
2. Mahasiswa dapat menuliskan sintaks grouping dengan benar.
3. Mahasiswa dapat menerapkan operasi agregasi yang tepat untuk kebutuhan analisis data.

**Alat dan Bahan:**

1. Modul Praktikum
2. Komputer
3. MySQL/XAMPP MariaDB
4. Moodle

**Waktu: 150 menit**

**Materi dan Pembagian Waktu:**

| No. | Materi | Waktu |
|---|---|---|
| 1. | Pretest | 10 menit |
| 2. | Materi SQL Aggregation | 50 menit |
| 3. | Mencoba latihan | 45 menit |
| 4. | Posttest | 45 menit |

**Materi Praktikum**

Bagian ini akan membahas fungsi agregasi kelompok. Menurut definisi KBBI, agregasi adalah pengumpulan sejumlah benda yang terpisah-pisah menjadi satu. Dalam hal ini, akan ada dua proses yaitu pengumpulannya (grouping dengan klausa GROUP BY dan kondisi HAVING) dan penghitungan hasil pengumpulannya yang umumnya ditempatkan di klausa SELECT. Masing-masing proses ini dapat berdiri sendiri tanpa melibatkan proses lainnya.

Perhatikan contoh berikut:

*Tabel 1. Tabel Dosen*

| *nip* | *namaPegawai* | *namaProdi* | *gajiTahunan* |
|---|---|---|---|
| 10101 | Sriyanto | Informatika | 66000000 |
| 12121 | Wahyu | Akuntansi | 75000000 |
| 15151 | Mozart | Sistem Informasi | 68000000 |
| 22222 | Einstein | Desain Produk | 60000000 |
| 32343 | Spielberg | Arsitektur | 62000000 |
| 33456 | Lucas | Kedokteran | 95000000 |
| 44565 | Katz | Biologi | 79000000 |
| 58583 | Charles | Informatika | 83000000 |
| 76543 | Simon | Biologi | 85000000 |
| 76766 | Peter | Sistem Informasi | 75000000 |
| 83821 | Bagong | Manajemen | 66000000 |
| 98345 | Karyono | Pendidikan Bahasa | 76000000 |

Jika kita memperhatikan Tabel 1 (tabel Dosen) di atas, kita mungkin menemui pertanyaan berikut:

1) Berapakah rata-rata gaji dosen?
2) Berapa rata-rata gaji pegawai masing-masing prodi?
3) Berapa orang yang gajinya Rp 66.000.000,-?
4) Berapa total gaji yang harus disiapkan perusahaan per prodi?
5) Siapa yang memiliki gaji terendah dan siapa yang tertinggi?

Pertanyaan tersebut sangat umum dijumpai terlebih ketika sistem basis data sudah memasuki tahap yang lebih tinggi, antara lain untuk keperluan analisis data. Kita tidak bisa mendapatkan jawaban hanya dengan menggunakan sintaks SQL yang telah kita pelajari sebelumnya. Mungkin untuk data kecil bisa dihitung secara manual. Namun untuk data yang besar atau raksasa, sangat tidak efisien jika harus menghitung secara manual. Untuk menjawab pertanyaan tersebut kita perlu mempelajari tentang *SQL group-aggregate function.*

Berikut ini fungsi agregasi yang didukung MySQL:

*Tabel 2. Fungsi Aggregasi di MySQL*

| Nama | Deskripsi |
|---|---|
| `AVG()` | Mengembalikan nilai rata-rata |
| `COUNT()` | Mengembalikan pencacahan dari jumlah row sesuai kondisi tertentu |
| `COUNT(DISTINCT)` | Mengembalikan pencacahan dari nilai-nilai yang berbeda (nilai yang sama tidak dihitung) |
| `GROUP_CONCAT()` | Mengembalikan string yang digabungkan |
| `MAX()` | Mengembalikan nilai maksimal |
| `MIN()` | Mengembalikan nilai minimal |
| `STD()` | Mengembalikan nilai standar deviasi dari populasi |
| `STDDEV()` | |
| `STDDEV_POP()` | |
| `STDDEV_SAMP()` | Mengembalikan nilai standar deviasi dari sampel |
| `SUM()` | Mengembalikan jumlah |
| `VAR_POP()` | Mengembalikan nilai standar variansi dari populasi |
| `VARIANCE()` | |
| `VAR_SAMP()` | Mengembalikan nilai variansi sampel |

1. **AVG([DISTINCT] *expr*)**

   Fungsi ini akan mengembalikan nilai rata-rata dari kolom atau ekspresi yang tercantum pada *expr.* Opsi DISTINCT bisa digunakan untuk membuat baris-baris yang memiliki nilai yang sama hanya dihitung satu kali. Jika tidak ada baris yang ditemukan, AVG() mengembalikan NULL.

   Contoh: *"Tampilkan rata-rata gaji tahunan para dosen"*

```
SELECT AVG(gajiTahunan)
FROM dosen;
```

```
+-------------------+
| AVG(gajiTahunan)  |
+-------------------+
| 74166666.66666667 |
+-------------------+
```

Gambar 1. Hasil query

Contoh penggunaan dengan grouping: *"Tampilkan rata-rata gaji tahunan para dosen per prodi"*

```
SELECT namaProdi, AVG(gajiTahunan)
FROM dosen
GROUP BY namaProdi;
```

```
+-------------------+-------------------+
| namaProdi         | AVG(gajiTahunan) |
+-------------------+-------------------+
| Akuntansi         |          75000000 |
| Arsitektur        |          62000000 |
| Biologi           |          82000000 |
| Desain Produk     |          60000000 |
| Informatika       |          74500000 |
| Kedokteran        |          95000000 |
| Manajemen         |          66000000 |
| Pendidikan Bahasa |          76000000 |
| Sistem Informasi  |          71500000 |
+-------------------+-------------------+
9 rows in set (0.002 sec)
```

Gambar 2. Hasil query

Saat sebuah query menggunakan grouping, adalah penting untuk memastikan bahwa satu-satunya atribut yang muncul di klausa SELECT yang tidak diagregasikan adalah atribut yang dituliskan di klausa GROUP BY. Dengan kata lain, tiap atribut yang tidak muncul di klausa GROUP BY dapat muncul pada klausa SELECT sebagai atribut yang diagregasi. Jika tidak diagregasi, maka query akan dianggap "*erroneous*". Contoh berikut adalah query yang *erroneous* karena *nip* tidak muncul di klausa GROUP BY, dan kemunculannya di klausa SELECT tidak diagregasi.

```
SELECT namaProdi, nip, AVG(gajiTahunan)
FROM dosen
GROUP BY namaProdi;
```

```
+------------------+-------+-----------------+
| namaProdi        | nip   | AVG(gajiTahunan) |
+------------------+-------+-----------------+
| Akuntansi        | 12121 |        75000000 |
| Arsitektur       | 32343 |        62000000 |
| Biologi          | 44565 |        82000000 |
| Desain Produk    | 22222 |        60000000 |
| Informatika      | 10101 |        74500000 |
| Kedokteran       | 33456 |        95000000 |
| Manajemen        | 83821 |        66000000 |
| Pendidikan Bahasa | 98345 |        76000000 |
| Sistem Informasi | 15151 |        71500000 |
+------------------+-------+-----------------+
9 rows in set (0.000 sec)
```

Gambar 3. Hasil query

Perhatikan bahwa HAVING adalah klausa untuk memberikan batasan kondisi terhadap hasil grouping. Penggunaan HAVING sangat mirip dengan WHERE, namun WHERE tidak bisa memberi batasan terhadap hasil grouping. Berikut adalah contoh kemampuan HAVING yang mirip dengan WHERE:
*"Berapa rata-rata gaji tahunan untuk setiap prodi Arsitektur dan Desain Produk?"*

```
SELECT namaProdi, AVG(gajiTahunan)
FROM dosen
GROUP BY namaProdi
HAVING namaProdi IN ("Arsitektur", "Desain Produk");
```

menghasilkan output sama dengan query berikut:

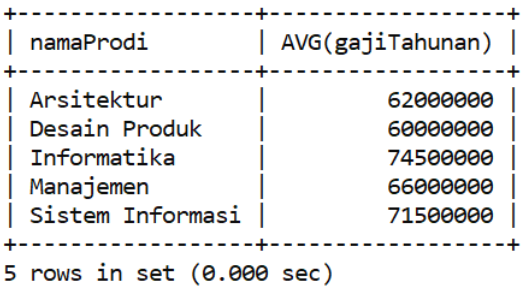
```
SELECT namaProdi, AVG(gajiTahunan)
FROM dosen
WHERE namaProdi IN ("Arsitektur", "Desain Produk")
GROUP BY namaProdi;
```

```
+---------------+-----------------+
| namaProdi     | AVG(gajiTahunan) |
+---------------+-----------------+
| Arsitektur    |        62000000 |
| Desain Produk |        60000000 |
+---------------+-----------------+
```

Gambar 4. Hasil query

Sedangkan berikut ini adalah contoh yang bisa dilakukan menggunakan HAVING dan tidak bisa dilakukan menggunakan WHERE:
*"Berapa rata-rata gaji tahunan untuk setiap prodi yang memiliki rata-rata gaji tahunan di bawah Rp 75 juta?"*

```
SELECT namaProdi, AVG(gajiTahunan)
FROM dosen
GROUP BY namaProdi
HAVING AVG(gajiTahunan) < 75000000;
```



```
+------------------+------------------+
| namaProdi        | AVG(gajiTahunan) |
+------------------+------------------+
| Arsitektur       |         62000000 |
| Desain Produk    |         60000000 |
| Informatika      |         74500000 |
| Manajemen        |         66000000 |
| Sistem Informasi |         71500000 |
+------------------+------------------+
5 rows in set (0.000 sec)
```

Gambar 5. Hasil query

Setiap atribut yang dituliskan di klausa HAVING tanpa diagregasikan, harus muncul di klausa GROUP BY. Jika tidak maka query akan dianggap *erroneous.*

Berikut ini urutan eksekusi query:

1) Seperti query tanpa agregasi, klausa FROM adalah klausa yang pertama dieksekusi untuk mengambil sebuah relasi/tabel.
2) Jika klausa WHERE muncul, predikat di klausa WHERE diterapkan pada relasi hasil pengambilan klausa FROM.
3) Tuple-tuple yang memenuhi predikat WHERE kemudian dikelompokkan berdasarkan klausa GROUP BY jika ada. Jika GROUP BY tidak ada, seluruh set tuple yang memenuhi predikat WHERE diperlakukan sebagai satu kelompok.
4) Klausa HAVING, jika ada, kemudian diterapkan untuk setiap kelompok. Kelompok yang tidak memenuhi predikat HAVING kemudian dihilangkan.
5) Klausa SELECT menggunakan kelompok yang tersisa untuk memunculkan tuple hasil dari query, menerapkan fungsi agregasi untuk mendapatkan

**2. COUNT(*expr*)**

Fungsi ini akan mengembalikan cacah jumlah dari nilai non-NULL dari *expr* atau baris yang dikembalikan oleh statemen SELECT. Hasilnya bertipe data BIGINT.
Contoh: *"Berapa jumlah dosen yang bergaji tahunan Rp 75 juta?" (2 orang)*

```
SELECT COUNT(*)
FROM dosen
WHERE gajiTahunan = 75000000;
```

Contoh penggunaan COUNT dengan GROUP BY: *"Berapa jumlah dosen per prodi?"*

```
SELECT namaProdi, COUNT(*) AS jumlahDosen
FROM dosen
GROUP BY namaProdi;
```

```
+-------------------+-------------+
| namaProdi         | jumlahDosen |
+-------------------+-------------+
| Akuntansi         |           1 |
| Arsitektur        |           1 |
| Biologi           |           2 |
| Desain Produk     |           1 |
| Informatika       |           2 |
| Kedokteran        |           1 |
| Manajemen         |           1 |
| Pendidikan Bahasa |           1 |
| Sistem Informasi  |           2 |
+-------------------+-------------+
9 rows in set (0.000 sec)
```

Gambar 6. Hasil query

3. **COUNT(DISTINCT *expr*)**
   Opsi DISTINCT bisa digunakan untuk membuat baris-baris yang memiliki nilai yang sama hanya dihitung satu kali.

4. **GROUP_CONCAT(*expr*)**
   Sesuai namanya, fungsi ini akan menggabungkan nilai-nilai dari suatu kolom hasil agregasi menjadi satu sel saja.
   Contoh:

```
SELECT namaProdi, GROUP_CONCAT(gajiTahunan)
FROM dosen
GROUP BY namaProdi;
```

```
+-------------------+---------------------------+
| namaProdi         | GROUP_CONCAT(gajiTahunan) |
+-------------------+---------------------------+
| Akuntansi         | 75000000                  |
| Arsitektur        | 62000000                  |
| Biologi           | 79000000,85000000         |
| Desain Produk     | 60000000                  |
| Informatika       | 83000000,66000000         |
| Kedokteran        | 95000000                  |
| Manajemen         | 66000000                  |
| Pendidikan Bahasa | 76000000                  |
| Sistem Informasi  | 68000000,75000000         |
+-------------------+---------------------------+
9 rows in set (0.020 sec)
```

Gambar 7. Hasil query

5. **MAX (*expr*)**
   Fungsi ini akan menampilkan nilai terbesar dari populasi.
   Untuk menampilkan siapa yang mendapat gaji terbesar adalah dengan subquery berikut:

```
SELECT namaPegawai
FROM dosen
WHERE gajiTahunan = (SELECT MAX(gajiTahunan) FROM dosen);
```

Bukan dengan query berikut:

```
SELECT namaPegawai, MAX(gajiTahunan)
FROM dosen
```

6. **MIN(*expr*)**
   Fungsi ini akan menampilkan nilai terkecil dari populasi. Penggunaannya sama seperti MAX(expr).

7. **STD (*expr*)**
   Fungsi ini akan menampilkan nilai standar deviasi dari populasi.
   Contoh:

```
SELECT STD(gajiTahunan) FROM dosen;
```

8. **SUM (*expr*)**
   Fungsi ini akan menampilkan hasil penjumlahan total dari populasi.
   Contoh:

```
SELECT SUM(gajiTahunan) FROM dosen;
```

   Contoh berikut menjawab pertanyaan: *"Berapa total gaji yang harus disiapkan perusahaan per prodi?"*

```
SELECT namaProdi, SUM(gajiTahunan)
FROM dosen
GROUP BY namaProdi;
```

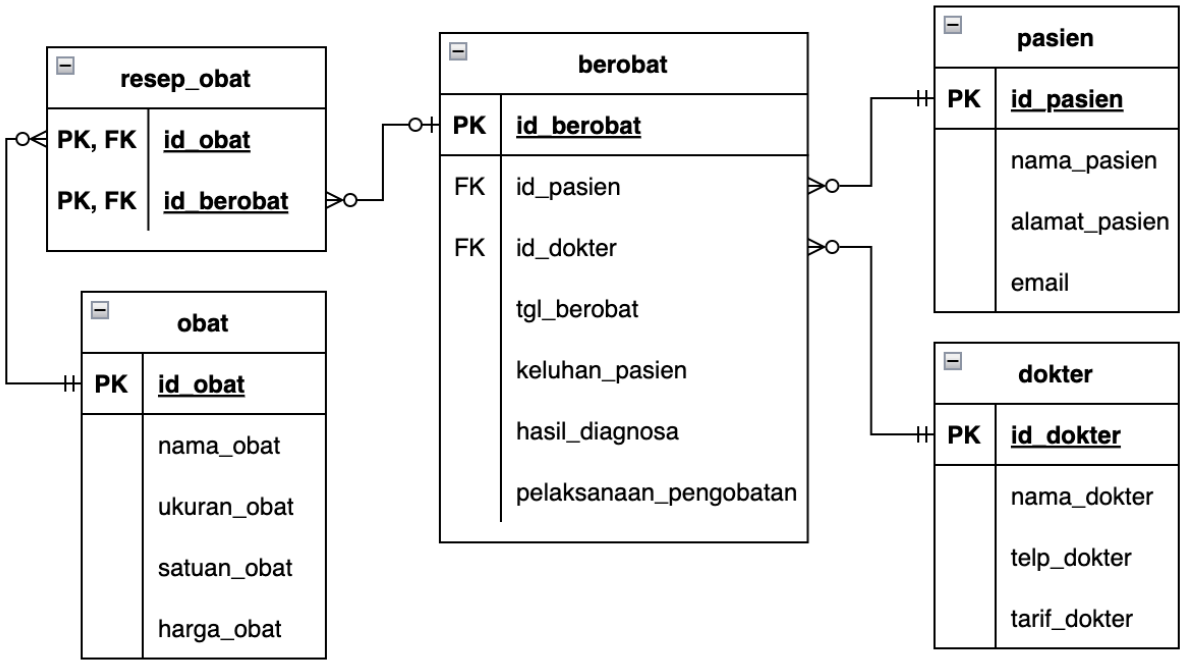
9. **VARIANCE (*expr*)**
   Fungsi ini akan menampilkan nilai variance dari populasi.
   Contoh:

```
SELECT VARIANCE(gajiTahunan)
FROM dosen;
```

Sebagai tambahan, nilai NULL merepotkan pemrosesan operator agregasi. Contoh jika ada beberapa tuple di tabel *dosen* yang memiliki nilai NULL pada gajinya. Ketika kita menjalankan SUM terhadap *gajiTahunan,* apakah dosen yang gajinya NULL disamakan dengan Rp 0,-? Apakah hasil SUM = NULL? Maka secara standar, seluruh fungsi agregasi kecuali COUNT(*) mengabaikan nilai NULL.

**Latihan Unguided:**

Kita lanjutkan model DB dari minggu lalu, silakan ambil file "SQL Modul 7.sql", pada drive atau klik disini. Kemudian import ke dalam database kalian.




1. Tampilkan informasi dokter yang memiliki tarif paling besar.
2. Tampilkan informasi dokter yang memiliki tarif paling kecil.
3. Tampilkan jumlah jenis obat yang dimiliki oleh klinik, berdasarkan satuan obatnya.

| satuan_obat | jumlah |
|---|---|
| mg | 4 |
| ml | 3 |
| g | 2 |

4. Tampilkan rata-rata harga obat pada setiap satuan obat

| satuan_obat | Rata-rata Harga Obat |
|---|---|
| mg | 249.4895 |
| ml | 42.74033333333333 |
| g | 4665.2975 |

5. Hitunglah ada berapa jenis satuan obat yang dimiliki oleh klinik

| Jumlah jenis Obat |
|---|
| 3 |

6. Tampilkan jumnlah obat yang diresepkan pada tiap pengobatan

| id_berobat | Jumlah Obat |
|---|---|
| 1 | 2 |
| 2 | 1 |

7. Tampilkan id_berobat beserta nama obatnya yang ditampilkan dalam 1 kolom

| id_berobat | GROUP_CONCAT(nama_obat) |
|---|---|
| 1 | Simvastatin,Sanmol |
| 2 | Neo Kaolana Sirup |

8. Tampilkan obat yang harus dibeli oleh pasien

| nama_pasien | Obat yang harus dibeli |
|---|---|
| rika | Neo Kaolana Sirup |
| roger | Simvastatin,Sanmol |

9. Tampilkan obat yang telah diresepkan dokter

| Nama Dokter | Obat yang pernah diresepkan |
|---|---|
| Lukas Chrisantyo | Neo Kaolana Sirup |
| Rosa Delima | Simvastatin,Sanmol |

10. Tampilkan semua nama dokter dan berapa kali melakukan periksa. Urutkan berdasarkan nama dokter.

| Nama Dokter | Jumlah Periksa |
|---|---|
| Agata Filiana | 0 |
| Hasta Mahendra | 0 |
| Jane Yolanda | 0 |
| Lala Rahayu | 0 |
| Lukas Chrisantyo | 2 |
| Marwata Nainggolan | 0 |
| Nila Anggia | 2 |
| Prayoga Adriansyah | 0 |
| Rosa Delima | 1 |
| Septi Farida | 0 |
| Yahya Mahendra | 0 |
| Zulfa Utami | 0 |

11. Tampilkan daftar pasien yang telah ditangani oleh dokter

| Nama Dokter | Jumlah Periksa |
|---|---|
| Lukas Chrisantyo | rika,rupawan |
| Nila Anggia | roger,rika |
| Rosa Delima | roger |

12. Tampilkan total pembayaran yang harus dibayar pasien (jumlah harga obat dan tarif dokter)

| id_berobat | nama_pasien | Total Pembayaran |
|---|---|---|
| 1 | roger | 202448 |
| 2 | rika | 218621 |
| 3 | rupawan | 200000 |
| 4 | roger | 150000 |
| 5 | rika | 150000 |

**REFERENSI:**


Silberschatz, A., Korth, H. & Sudarshan, S. (2020). *Database System Concepts,* McGrawHill.

MySQL Group By Functions - https://dev.mysql.com/doc/refman/5.7/en/group-by-functions.html

Lynn Beighley. (2007). *Head First SQL*, 2007, O'Reilly. ISBN: 978-0-596-52684-9