

SQL numeric functions and aggregation

Aggregation and ordering in SQL

Aggregation and ordering

Aggregation and ordering help us to **summarise** and **arrange** data so that we can examine it from different viewpoints.

Aggregation refers to the process of **summarising or combining multiple rows of data** into a single value or set of values.

Ordering refers to **sorting the result set** of a SQL query based on one or more columns.

Aggregation clauses

An aggregate function in SQL **returns one value after calculating multiple values** of a column. We **use aggregate functions** with the **GROUP BY** and **HAVING** clauses and can use **ORDER BY** to sort the results.

GROUP BY

The **GROUP BY** clause is used with **aggregate functions** to **group the result set** by one or more columns.

HAVING

The **HAVING** clause is used to **filter the result set** based on an **aggregate function**.

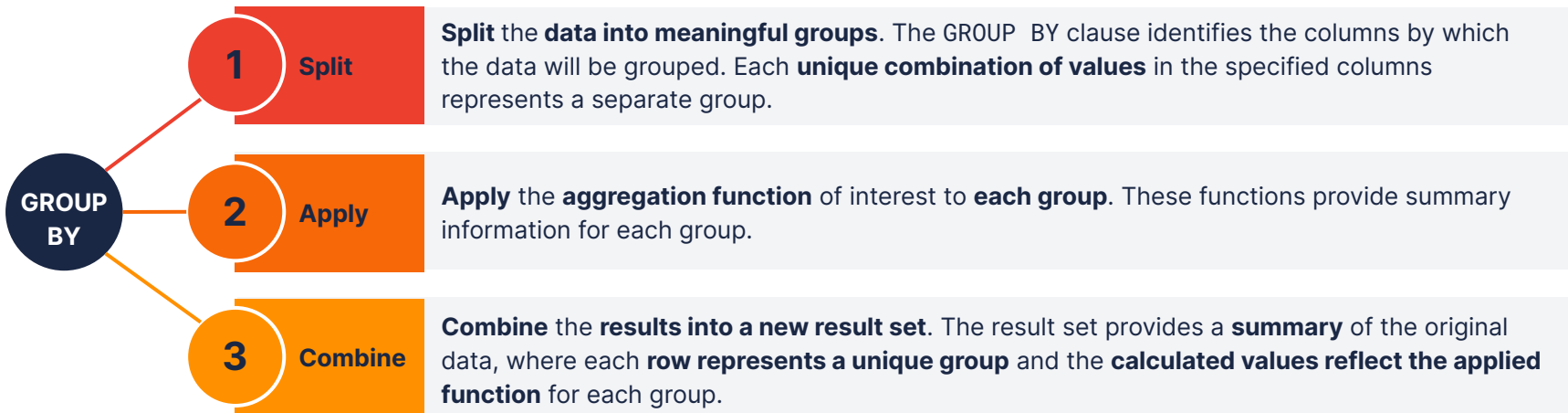
ORDER BY

The **ORDER BY** clause can be used with **aggregate functions** to **sort the result set** based on the calculated values of the aggregates.

GROUP BY

The **GROUP BY** clause allows us to **divide a table into distinct sets** based on one or more columns. This division is based on a shared attribute within the data.

The **GROUP BY** clause follows a **split-apply-combine** paradigm, which helps in understanding its purpose and functionality.



GROUP BY Syntax

Syntax

```
SELECT
    column1,
    FUNCTION(columnN)
FROM
    table_name
GROUP BY
    column1;
```

The query **retrieves specific columns** from a table and **applies an aggregate function** to one of the columns. It then **groups the data** based on one or more columns specified in the GROUP BY clause.

Note: If we select **multiple columns** we must include those **same columns** in the **GROUP BY** clause or we will get an error.

Example

```
SELECT
    clinic_id,
    AVG(age)
FROM
    Patient
GROUP BY
    clinic_id;
```

- 1 Split:** Group records from the **Patient** table based on their **clinic ID**, creating distinct groups of patients belonging to each clinic.
- 2 Apply:** Calculate the **average age of the patients within each clinic group**, by applying the **AVG** aggregation function to the age column.
- 3 Combine:** Retrieve the **clinic ID** and **corresponding average age** as the result, providing the average age for each clinic based on the grouped patients.

GROUP BY query illustration

Patient table

Patient

Patient_id	Clinic_id	Age
1	1	54
2	3	22
3	1	51
4	2	30
5	2	36
6	3	20
7	1	67
8	1	74
9	2	61
10	3	46

1. Split

Patient_id	Clinic_id	Age
1	1	54
3	1	51
7	1	67
8	1	74

Patient_id	Clinic_id	Age
4	2	30
5	2	36
9	2	61

Patient_id	Clinic_id	Age
2	3	22
6	3	20
10	3	46

2. Apply (AVG)

Clinic_id	AVG(Age)
1	61.5

Clinic_id	AVG(Age)
2	42.333

Clinic_id	AVG(Age)
3	29.333

3. Combine

Clinic_id	AVG(Age)
1	61.5
2	42.333
3	29.333

HAVING

The **HAVING** clause is similar to the WHERE clause and used to **filter** the result set. The key difference is that **HAVING** can be used with **aggregate functions**.

Syntax

```
SELECT
    column1,
    FUNCTION(columnN)
FROM table_name
GROUP BY column1
HAVING condition;
```

Now we have added a **HAVING** clause, so the results are **filtered** using the **specified condition**.

Example

```
SELECT
    clinic_id,
    AVG(age)
FROM Patient
GROUP BY clinic_id
HAVING AVG(age) > 30;
```

The addition of the **HAVING** clause means that the results grid will only display the **average age** for each clinic where **the average age is greater than 30**.



Note: WHERE and HAVING can be used in the **same query** but the **where clause** only works on **individual rows**, not on aggregated data. In general, we use **WHERE prior to GROUP BY** as a pre-filter and **HAVING after GROUP BY** as a post-filter.

GROUP BY with HAVING query illustration

1. Split

Patient_id	Clinic_id	Age
1	1	54
3	1	51
7	1	67
8	1	74

Patient_id	Clinic_id	Age
4	2	30
5	2	36
9	2	61

Patient_id	Clinic_id	Age
2	3	22
6	3	20
10	3	46

2. Apply (AVG)

Clinic_id	AVG(Age)
1	61.5

Clinic_id	AVG(Age)
2	42.333

Clinic_id	AVG(Age)
3	29.333

HAVING AVG(age)>30;

3. Combine

Clinic_id	AVG(Age)
1	61.5
2	42.333

ORDER BY

ORDER BY can be used to **sort** the result set based on the **calculated values of the aggregations** (in ascending or descending order).

Syntax

```
SELECT
    column1,
    FUNCTION(columnN)
FROM table_name
GROUP BY column1
ORDER BY FUNCTION(columnN);
```

By using an **ORDER BY** clause, the results are **arranged according to the aggregated field** in either ascending or descending order.

Example

```
SELECT
    clinic_id,
    AVG(age)
FROM Patient
GROUP BY clinic_id
ORDER BY AVG(age) ASC;
```

The addition of the **ORDER BY** clause **orders the result set in ascending** order based on the **average age** at each clinic. (We use **DESC** for **descending order**. If we left out **ASC** it still sorts in **ascending order by default**.)



Note: We can still have the **HAVING** and **WHERE** clauses, but they must come **before** the **ORDER BY** clause.

GROUP BY with ORDER BY query illustration

1. Split

Patient_id	Clinic_id	Age
1	1	54
3	1	51
7	1	67
8	1	74

Patient_id	Clinic_id	Age
4	2	30
5	2	36
9	2	61

Patient_id	Clinic_id	Age
2	3	22
6	3	20
10	3	46

2. Apply (AVG)

Clinic_id	AVG(Age)
1	61.5

Clinic_id	AVG(Age)
2	42.333

Clinic_id	AVG(Age)
3	29.333

ORDER BY AVG(age) ASC;

3. Combine

Clinic_id	AVG(Age)
3	29.333
2	42.333
1	61.5