
SQL - Aggregations

Contents

- **Aggregation Functions**
  - `COUNT - SUM - MIN - MAX - AVG`
  - `GROUP BY`
  - `HAVING`
- **Functions**
  - `DISTINCT`
  - `DATE_TRUNC`
  - `DATE_PART`
- **Flow Control**
  - `CASE`
  - `NULLIF`

Aggregations

- We know how to prepare the data, now we can analyze the data.
- As a Data Scientist, you will try to understand the data by summarizing it and finding high-level patterns.
- SQL will help you with this task using ***Aggregation functions***
  - `COUNT` – How many rows are in a particular column
  - `SUM` – Add all values in a particular column
  - `MIN/MAX` – Lowest and highest values in a particular column
  - `AVG` – Average values in a particular column

There are a lot! Go to this link to check them

Aggregations - COUNT

- `COUNT(column)` counts the number of non-NULL values over a table or column
- Using COUNT, can you tell how many NULL values are in a column?

```
SELECT COUNT(*)
FROM address
```

```
SELECT COUNT(address_id)
FROM address
```

```
SELECT COUNT(address2)
FROM address
```

Aggregations - `SUM`

- `SUM(column)` - Returns the sum of all values in column
- Only works on numerical data (unlike `COUNT`)
- Ignores NULL values (treats them as 0)

```
SELECT SUM(replacement_cost)
FROM film
```

Aggregations - `MIN/MAX`

- `MIN(column)` - Returns the MINimum value in column
  - It returns the lowest number, earliest date, or first character from the alphabet
- `MAX(column)` - Returns the MAXimum value in column
  - It returns the highest number, latest date, or last character from the alphabet

```
SELECT MIN(replacement_cost), MAX(replacement_cost)
FROM film
```

Aggregations - AVG

- `AVG(column)` - Returns the AVeraGe of all values in column
  - Ignores `NULL`s in the numerator and denominator
  - Only works with numerical values

```
SELECT AVG(replacement_cost)
FROM film
```

Aggregations - Practicals (Part I)

Go to the portal and complete the first practical:

Basic Aggregation

Aggregations - `GROUP BY`

- Sometimes we don't want to find the aggregate value of a whole column, but for smaller groups in the table.
- Imagine you want to find the best customers, those who have rented more than 30 movies.
- If we try to look at the whole table, and count each customer, it will take forever
- We can use `GROUP BY` to divide the rows of a dataset into multiple groups based on some sort of key
- An aggregate function is then applied to all the rows

Aggregations - `GROUP BY`

- The syntax of a `GROUP BY` query is usually:

```
SELECT {column}, {aggregation}
FROM {table)
GROUP BY {column}
```

- When using aggregates, any column in `SELECT` which is not an aggregate must also be specified in the `GROUP BY` statement. (It doesn't have to be the other way around)
- Note that the aggregation is not mandatory.

Aggregations - GROUP BY

- Imagine you want to find the best customers, those who have rented more than 30 movies.

```
SELECT customer_id, COUNT(*)
FROM rental
GROUP BY customer_id
ORDER BY 1;
```

ORDER BY 1?? What do you think it means?

Aggregations - GROUP BY

- You can use `GROUP BY` with different columns:

```
SELECT customer_id, COUNT(*)
FROM rental
GROUP BY customer_id. inventory_id;
```

Aggregations - GROUP BY

- We wanted to check the customers with more than 30 rentals:

```
SELECT customer_id, COUNT(*) AS c
FROM rental
GROUP BY customer_id
/*What condition should I add?*/;
```

Aggregations - HAVING

- When `WHERE` tried to evaluate that column, the `GROUP BY` has not generated the groups yet.
- For these cases, we use `HAVING`, which is specifically designed for `GROUP BY` queries

```
SELECT {column}, {aggregation}
FROM {table}
GROUP BY {column}
HAVING {aggregation_condition}
```

Aggregations - HAVING

- We wanted to check the customers with more than 30 rentals:

```
SELECT customer_id, COUNT(*)
FROM rental
GROUP BY customer_id
HAVING COUNT(*) > 30
```

Aggregations - Practicals (Part II)

Go to the portal and complete the second practical:
Using Aggregated Functions

You could use other SQL commands for this practical,
but use aggregations to get some practice!

Aggregations - Useful Functions

- SQL offers useful functions for data analysis
- These functions can be used along aggregations
- For example, count the *unique* values of a column, or count the rentals *per day*
- Some common functions are:
  - `DISTINCT`
  - `DATE_TRUNC`
  - `DATE_PART`

Functions - DISTINCT

- DISTINCT returns the unique instances over a column
- Used in the SELECT statement.
- Multiple columns evaluates unique combinations

```
SELECT DISTINCT rental_rate
FROM film
```

```
SELECT DISTINCT rental_rate, rating
FROM film
```

Can you think of a way to count the unique values?

Functions - DATE_TRUNC

- Try the following query:

```
SELECT payment_date, COUNT(*)
FROM payment
GROUP BY payment_date
```

- The length of the output is (almost) the same as the original table

WHY??

Functions - DATE_TRUNC

- The reason is the date format: `YYYY-MM-DD HH:MM:SS`
- It's very unlikely that two operations take place at the same second
- We can use `DATE_TRUNC` to TRUNCate part of the date:

```
SELECT DATE_TRUNC({field}, {column});
```

Where `field` is the precision to truncate

```
SELECT DATE_TRUNC('day', payment_date) AS day, COUNT(*)
FROM payment
GROUP BY 1
ORDER BY 1;
```

You can check the available fields in this link

Functions - DATE_PART

- DATE_PART function retrieves subfields such as year (YEAR), month (MONTH), or hour (HOUR) from date/time values
- We can also extract Day Of the Week (DOW), Day Of the Year (DOY), or even millennium (MILLENNIUM)

```
SELECT DATE_PART('{field}', {column});
```

Where `field` is the date/time element

```
SELECT DATE_PART('DOW', payment_date) AS day, COUNT(*)
FROM payment
GROUP BY 1;
```

Go to the this link to know more about date functions

Aggregations - Practicals (Part III)

Go to the portal and complete the third practical:
Dates in SQL

Flow Control - `CASE`

- `CASE` creates a new column based on the conditions we declare
- It has to include the keywords `WHEN`, `THEN`, and `END`, and optionally ELSE
- It usually goes in the `SELECT` statement
- Pythonic way to see it:

```
if condition_1:
    return value_1
elif condition_2:
    return value_2
else:
    return value_3
```

Flow Control - CASE

```
SELECT title, release_year, rental_rate,
CASE
     WHEN rental_rate > 0 AND rental_rate < 2.99 THEN 'discount'
     WHEN rental_rate >= 2.99 AND rental_rate < 4.99 THEN 'regular'
     ELSE 'premium'
END AS quality
FROM film
```

Aggregations - Practicals (Part IV)

Go to the portal and complete the fourth practical:
Additional Functions in SQL