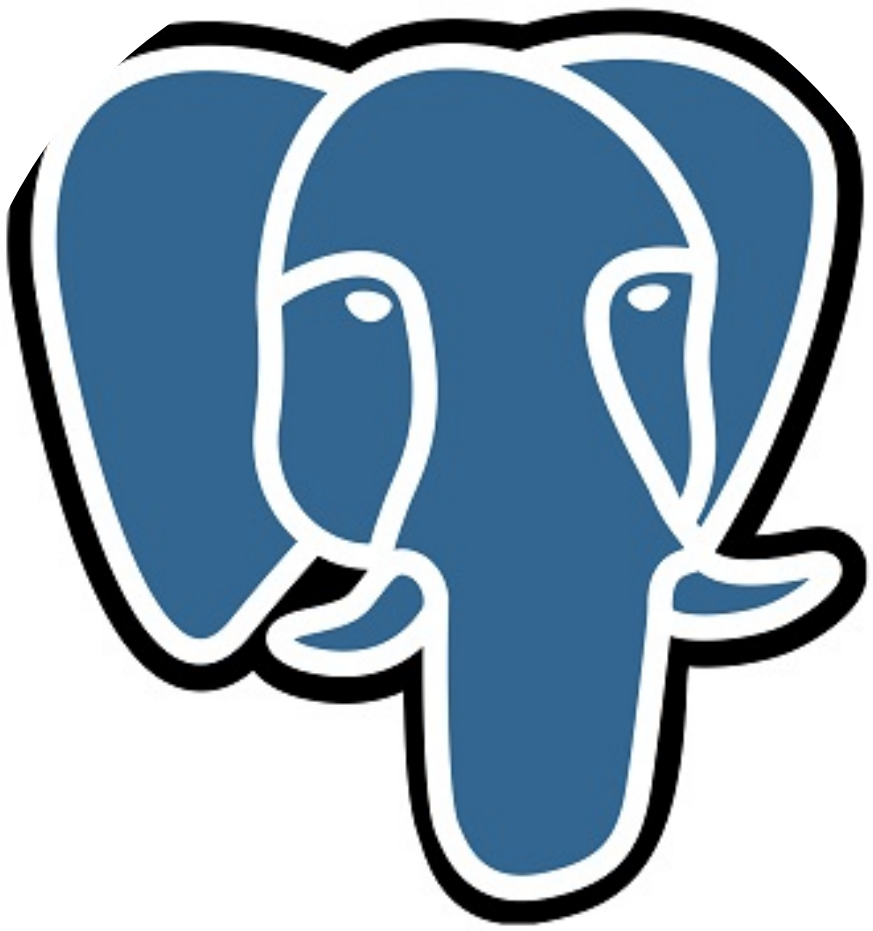

SQL Basics Part II - GROUP BY Statements

24.10.2019

SQL Basics

MIN MAX AVG SUM (Aggregate Functions)

★Aggregate functions basically take a lot of rows of data and aggregate/combine it into a single value.

★COUNT( ) was also an aggregate function.

★Aggregate functions are used a lot with the GROUP BY statement but for now we are going to use them without the GROUP BY statement.

Example:

```
SELECT AVG(amount) FROM payment_table;
```

★It will select average value of the amount column in the payment table.

```
SELECT ROUND(AVG(amount),2) FROM payment_table;
```

★ROUND function basically rounds the result, the second parameter specifies the number of decimals we want to round to.

```
SELECT MIN(amount) FROM payment_table;
```

★It will select the minimum value of the amount column.

```
SELECT MAX(amount) FROM payment_table;
```

★It will select the maximum value of the amount column.

```
SELECT SUM(amount) FROM payment_table;
```

★It will basically add all the elements of the amount column.

GROUP BY Statement

★GROUP BY clause might be a bit confusing at the beginning.

★It is one of the most useful tools in SQL.

★The **GROUP BY** clause divides the rows returned from the SELECT statement into groups.
★For each group, we can apply an aggregate function. We don't have to call an aggregate function in order to apply GROUP BY but most commonly it is used with aggregate functions.

Example:

```
SELECT customer_id
FROM payment
GROUP BY customer_id;
```

★It will group the results (whole customer_id column) by the customer_id.
★It just acts like we've used the "SELECT DISTINCT customer_id FROM payment". We can say without aggregate functions **GROUP BY** clause works very similar to the **DISTINCT** clause.
★Yani customer_id'si aynı olanları gruplar ve tek bir row'da yazdırır. Böylece aslında sadece unique customer_id'lerden oluşan bir result column elde ederiz.
★Distinct'ten önemli bir farkı şu: gruplar tek satır olarak görünse de o tek satır içinde tüm grup elemanlarını tutuyor. Yani biz sum dediğimizde o id'ye sahip tüm customer'lar üzerinde bir sum yapmış oluyoruz.

```
SELECT customer_id, SUM(amount)
FROM payment
```

```
GROUP BY customer_id;
```

★ It will return the distinct customer_ids and corresponding total payment amounts for each customer_id. First groups by customer_id and then sums the payment values for each group.

```
SELECT customer_id, SUM(amount)
FROM payment
GROUP BY customer_id
ORDER BY SUM (amount)  DESC;
```

★After returning the customer_ids and total payments, now we sorted the result by total payments.

```
SELECT staff_id, COUNT(*)
FROM payment
GROUP BY staff_id;
```

★Returns grouped staff_id's and number of elements for each group.

★ GROUP BY dediğimiz column'u mutlaka SELECT'in yanına da yazdığımıza dikkat et PostgreSQL için bu şart değil ama diğer database management systems için şart.

GROUP BY - PRACTICE

Challenges:

★We have 2 types of staff members with staff_id = 1 and staff_id = 2. We want to give a bonus to the staff member that handled the most payments.
★How many payments did each staff member handle? And how much was the total amount processed by each staff member?

```
SELECT staff_id, COUNT(*), SUM(amount)
FROM payment
GROUP BY staff_id;
```

★Corporate headquarters is auditing our store. They want to know the average replacement cost of movies by rating.
★For example, R rated movies have an average replacement cost of $20.23

```
SELECT rating, AVG(replacement_cost)
FROM film
GROUP BY rating;
```

★We want to send coupons to the 5 customers who have spent the most amount of money.
★Get me the customer ids of the top 5 spenders.

```
SELECT customer_id, SUM(amount)
FROM payment
GROUP BY customer_id
ORDER BY SUM(amount) DESC
LIMIT 5;
```

HAVING Clause

★We often use **HAVING** clause in conjunction with the **GROUP BY** clause to filter group rows that do not satisfy a specified condition.

★It is basically like the **WHERE** statement except that we are using it with the **GROUP BY** statement.

★ The **HAVING** clause sets the condition for group rows created by the GROUP BY clause after the GROUP BY clause applies while the **WHERE** clause sets the condition for individual rows before GROUP BY clause applies.
★ Yani **WHERE** ile önce condition check ediliyor ve o condition'ı sağlayan satırlar alınıyor. **HAVING** ile ise önce GROUP BY grupluyor en son HAVING ile condition check ediliyor.

Example:

```
SELECT customer_id, SUM(amount)
FROM payment
GROUP BY customer_id;
```

★We've already know that, the above code will group by customer_id and sum all the amounts paid for each customer id.

```
SELECT customer_id, SUM(amount)
FROM payment
GROUP BY customer_id
HAVING SUM(amount) > 200;
```

★So first it is grouping and returns 2 rows customer_ids and corresponding total amounts for each customer. Then having statement returns a part of this resultant table which satisfies the given condition.

```
SELECT store_id, COUNT(customer_id)
FROM customer
GROUP BY store_id
HAVING COUNT(customer_id) > 300;
```

★First group by store_id returns 2 columns store_id and the corresponding number of customers for each store. Later filtering out the rows have customers < 300.

★ Let's use both WHERE and HAVING to better understand the difference.

Example:

```
SELECT rating, rental_rate
FROM film
WHERE rating IN('R','G','PG');
```

★This will select the rating and rental_rate columns only if the rating is 'R' or 'G' or 'PG'

```
SELECT rating, AVG(rental_rate)
FROM film
WHERE rating IN('R','G','PG')
GROUP BY rating;
```

★So at first, it does the filtering by **WHERE** statement then it does the grouping by ratings.

```
SELECT rating, AVG(rental_rate)
FROM film
WHERE rating IN('R','G','PG')
GROUP BY rating
HAVING AVG(rental_rate) < 3;
```

★After the grouping, we will obtain 2 columns: ratings and corresponding average rental_rate price for each rating group.
★ Finally, we can apply another filter to this final dataframe by using the **HAVING** clause.

HAVING - PRACTICE

Challenges:

★We want to know what customers are eligible for our platinum credit card. The requirements are that the customer has at least a total of 40 transaction payments.
★What customers (by customer_id) are eligible for the credit card?

```
SELECT customer_id, COUNT(*)
FROM payment
GROUP BY customer_id
HAVING COUNT ( * )  >=  40 ;
```

★When grouped by rating, what movie ratings have an average rental duration of more than 5 days?

```
SELECT rating, AVG(rental_duration)
FROM film
GROUP BY rating
HAVING AVG(rental_duration) > 5  ;
```

More blogs