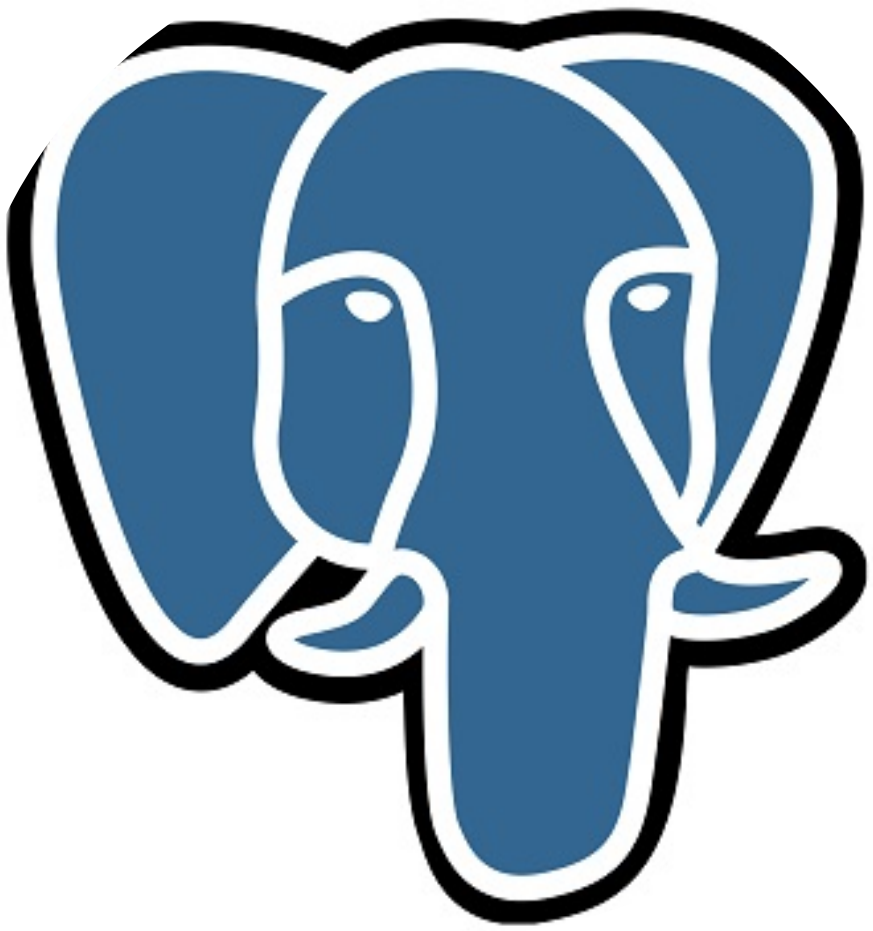

SQL Basics Part I - Statement Fundamentals

24.10.2019

SQL Basics

Intro

DEFINITION

Databases are systems that allow users to store and organize data. They are useful when dealing with large amounts of data.

SPREADSHEETS (such as Excel) VS DATABASES

Spreadsheets:

- good at one-time analysis.
- useful when quickly need to chart something out
- they work with reasonable data set sizes
- ability for untrained people to work with data

Databases:

- Good for data integrity. You don't want to have someone be able to just click on a cell and change the data, databases don't allow for that.

- Can handle massive amounts of data.
- Can quickly combine different datasets.
- Good for automating steps for re-use.
- Can support data for websites and applications.

FROM Excel to DATABASES

- The tabs referencing various sheets in Excel corresponds to the Tables in databases.
- Rows and columns in Excel correspond rows and columns in databases as well.

DATABASE PLATFORM OPTIONS

★There are a lot of platforms such as PostgreSQL, MySQL, Microsoft Access, SQLite etc. Here we will focus on PostgreSQL because it is free, open-source, widely used on the internet, multi-platform, easy to install.

★PostgreSQL is a good choice to learn how to use SQL and SQL learned in here can be applied to a variety of databases or SQL based software. Most of the commands will be general SQL.

★SQL stands for **Structured Query Language**. `It is basically a programming language used to communicate with our database.`

SQL Statement Fundamentals

1. SELECT STATEMENT

★One of the most common tasks is query data from tables by using the SELECT statement.

★Select statement has many clauses that you can combine to from a powerful query.

★Let's start with a basic form of the SELECT statement to query data from a table:

Syntax:

```
SELECT column1,column2,... FROM table_name;
```

★It will select the list of specified columns from the specified table
★If we want to query data from all column, we can use an asterisk (*) as the shorthand for all columns.

```
SELECT column1,column2,...
FROM table_name;
```

★This syntax (another line for every keyword) will work and it can make the code easier to read.

★ SQL language is **case - insensitive**. It means if we use SELECT or select the effect will be the same.
★ However by convention, we will use SQL keywords in uppercase to make the code easier to read.

Example:

```
SELECT * FROM actor;
```

★ "actor" is a table stored in the database we are working on.
★ We've basically selected all the columns of the actor table.

```
SELECT first_name,last_name FROM actor;
```

★ We've selected specific columns of the actor table.

---

2. SELECT DISTINCT STATEMENT

★ In a table, a column may contain many duplicate values; and sometimes we only want to list the different (distinct) values.
★ The **SELECT DISTINCT** keyword can be used to return only distinct values.

Syntax:

```
SELECT DISTINC column1 FROM table_name;
```

★ It will only return the unique elements of the specified column from the specified table.

---

3. SELECT WHERE STATEMENT

★ What if we want to query just particular rows from a table?
★ The **WHERE** clause in the **SELECT** statement can be used to return particular rows of specified columns.

Syntax:

```
SELECT column1,column2
FROM table_name
WHERE conditions;
```

★ The conditions are used to filter the rows returned.

★ We use operators to construct the conditions.

Basic Operators:

`=`	Equal
`>`	Greater than
`<`	Less than
`>=`	Greater than or equal
`<=`	Less than or equal
`!=`	Not equal
`AND`	Logical AND operator
`OR`	Logical OR operator

Example:

★ If we want to select all customers whose first name is Jamie:

```
SELECT last_name,first_name
FROM customer_table
WHERE first_name = 'Jamie';
```

★ If we want to select the email addresses of all customers whose first name is Jamie and last name is Rice:

```
SELECT email
FROM customer_table
WHERE first_name = 'Jamie' AND last_name = 'Rice';
```

★ If we want to select who paid rental with amount is either less than 1$ or greater than 8$:

```
SELECT customer_id, amount, payment_date
FROM payment_table
WHERE amount <= 1 OR amount >= 8;
```

---

4. COUNT FUNCTION

★ The **COUNT** function returns the number of input rows that match a specific condition of a query.

Syntax:

★ Let's see how many rows do we have in our table:

```
SELECT COUNT(*) FROM table;
```

★ Count rows of a specific column:

```
SELECT COUNT(column) FROM table;
```

★ Finally we can use COUNT with DISTINCT to count only the distinct elements of a specified column.

```
SELECT COUNT(DISTINC column) FROM table;
```

★ Note that COUNT( ) function does not consider NULL values in the column.

5. LIMIT STATEMENT

★ The **LIMIT** allows us to limit the number of rows we get back after a query.

★ Useful when wanting to get all columns but not all rows.

★ Goes at the end of a query

Syntax:

★ Let's say I want to see the first 10 rows of the customer table. Like df.head() function in pandas.

```
SELECT * FROM customer LIMIT 10;
```

---

6. ORDER BY STATEMENT

★ When we query data from a table, PostgreSQL returns the rows in the order that they were inserted into the table.

★ In order to sort the result set, we use **ORDER BY** clause in the SELECT statement.

★ So the ORDER BY clause allows you to sort the rows returned from the SELECT statement in ascending or descending order based on criteria specified.

Syntax:

```
SELECT column1,column2
FROM table1
ORDER BY column1 ASC/DESC;
```

Example:

★ If we want to see names and last names of customers sorted by the first name in ascending order

```
SELECT first_name,last_name
FROM customer_table
ORDER BY first_name ASC;
```

★ If we want to see first names ordered by last names. This property will work only for PostgreSQL for others we need to select the columns that we want to sort by.

```
SELECT first_name
FROM customer_table
ORDER BY last_name ASC;
```

★ If we want to sort by names and then last names. It will first sort by the names and if the first names are same these rows will be sorted by last names.

```
SELECT *
FROM customer_table
ORDER BY first_name ASC, last_name DESC;
```

Challenges:

★ Get the customer IDs for the top 10 highest payment amounts from payment_table:

```
SELECT cutomer_id
FROM payment_table
ORDER BY amount DESC
LIMIT 10;
```

★ Get the movie titles with the film ids 1-5:

```
SELECT title,film_id
FROM film_table
ORDER BY film_id ASC
LIMIT 5;
```

---------- or:

```
SELECT title
FROM film_table
WHERE film_id >= 1 AND film_id <= 5
```

---

7. BETWEEN OPERATOR

★ We use the **BETWEEN** operator instead of writing `>= AND <=` operators.
★ We can also use the **NOT BETWEEN** operator.

Syntax:

```
WHERE value BETWEEN low AND high;
```

The above statement is the same as:

```
WHERE value >= low AND value <= high;
```

Example:

★ If we want to select the customer ids whose payment values are between 8$ and 9$ (>=8$ and <=9$)

```
SELECT first_name
FROM payment_table
WHERE amount BETWEEN 8 AND 9;
```

★ If we want to select the payment ids whose payment amounts are not between specific dates.

```
SELECT payment_id
FROM payment_table
WHERE payment_date NOT BETWEEN '2007-02-07' AND '2007-02-15';
```

---

8. IN STATEMENT

★ We use the **IN** operator with the **WHERE** clause to return the elements which match with any specified list of values.
★ We can also use the **NOT IN** operator.

Syntax:

```
WHERE value IN (value1,value2,...)
```

The above statement is the same as:

```
WHERE value = value1 OR value = value2
```

★ List of values can be a result set of a **SELECT** statement

---

Syntax:

```
WHERE value IN (SELECT value FROM table_name)
```

Example:

★ If we want to select the elements with customer_id = 1 OR customer_id = 2.

```
SELECT customer_id
FROM rental_table
WHERE customer_id IN (1,2)
ORDER BY return_date DESC;
```

---

9. LIKE STATEMENT

★ Suppose we want to find the customers whose name begins with Jen. We can use **LIKE** statement.
★ **LIKE** is case sensitive **ILIKE** is case insensitive
★ Also we can use **NOT LIKE** statement as well.

Syntax:

```
WHERE first_name LIKE 'Jen%'
```

★% means pattern (any string). So we are looking for the elements begins with Jen and continues with any kind of string value.
★ _ means single character. So if we say 'Jen_' then we are looking for names start with Jen hand has only one more char at the end.
★ Note that we can use statements as **'%y'** to specify names ends with y or **'%er%'** to specify names contains er.

Example:

★ If we want to select the names starts with Jen:

```
SELECT first_name, last_name
FROM customer_table
WHERE first_name LIKE 'Jen%'
```

---

10. PRACTICE

Challenges:

★ How many payment transactions were greater than $5.00?

```
SELECT COUNT(amount)
FROM payment_table
WHERE amount>5;
```

★ How many actors have a first name that starts with the letter P?

```
SELECT COUNT(*)
FROM actor_table
WHERE first_name LIKE 'P%';
```

★ How many unique districts are our customers from?

```
SELECT COUNT(DISTINCT district)
FROM address;
```

★ Retrieve the list of names for those distinct districts from the previous challenge.

```
SELECT DISTINCT district
FROM address;
```

★ How many films have a rating of R and a replacement cost between $5 and $15?

```
SELECT COUNT(*)
FROM film
WHERE rating='R' AND replacement_cost BETWEEN 5 AND 15;
```

★ How many films have the word Truman somewhere in the title?

```
SELECT COUNT(*)
FROM film
WHERE title LIKE '%Truman%';
```

More blogs