SQL for data science

...

Outline

- Announcement
- Guest speaker
- Why are we learning SQL
- Typical Data Science Workflow
- Why PostgreSQL
  - Install PostgreSQL & pgAdmin4
  - Open datasets
  - Table creation
  - Data import & export
- Can’t we just use Python instead of SQL
- SQL for data science

Announcement

Revision of plan after students' feedback

- Assignments will be due **next week** Sunday.
- In-class activities are related to individual assignments

Guest speaker

Intro

Current Job

Current project

Why are we learning SQL

SQL is widely used.

- Google BigQuery
- AWS Redshift
- Snowflake
- Apache Spark
- PostgreSQL

Typical Data Science Workflow

Retrieve data from data warehouse using SQL

Put that data into Python dataframe

- Machine learning
- Visualization

Why PostgreSQL?

Open source

Accessible

Local installation

SQL standard compliant

- ANSI (American National Science Institute)
- ISO (International Organization for Standardization)

Can’t we just use Python (Pandas) instead of SQL?

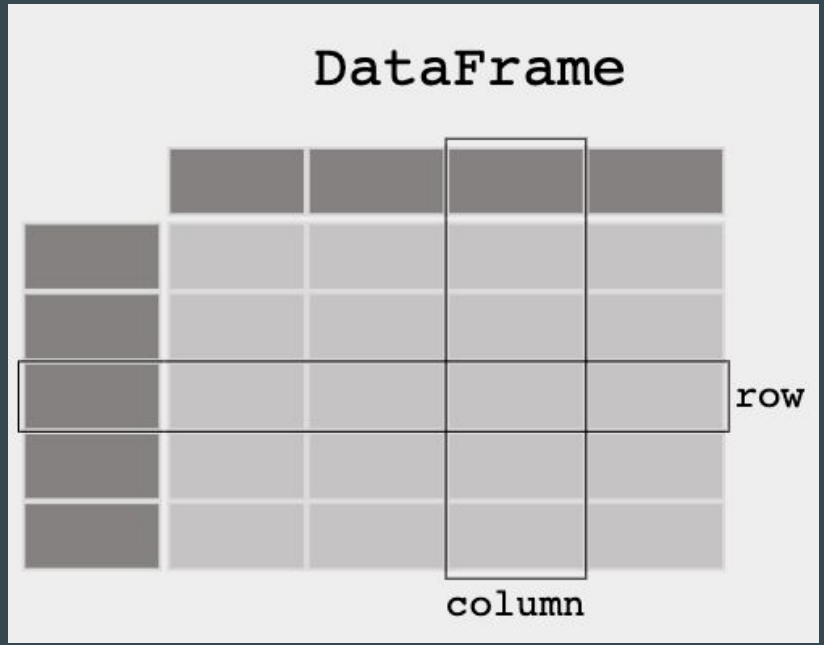
pandas: Python package for data manipulation & analysis.

You can do everything

Limitation in data science context

Demo:

Pandas



Learning Activities

Install PostgreSQL & pgAdmin4

Open datasets

Table creation

Data import & export

Outline

Querying data

Filtering data

1. Querying data

Select

Order by

Select distinct

SELECT

You can query data from tables by using the SELECT statement

```
SELECT
    select_list
FROM
    table_name;
```

Above is the basic form of the select statement that retrieves data from a single table

Select statement in more detail!

If you specify a list of columns, you need to place a comma (,) between two columns to separate them

```
SELECT list_1, list_2, list_3
FROM table_name;
```

Select data from all columns of the table

You can use an asterisk (*)

```
SELECT *
FROM table_name;
```

Specify the name of the table

```
SELECT *

FROM table_name;
```

When is not a good time to use the asterisk (*)

it is not a good practice to use the asterisk (*) in the SELECT statement **when you embed SQL statements in the application code like Python, Java, Node.js, or PHP** due to the following reasons:

1. Database performance

2. Application performance

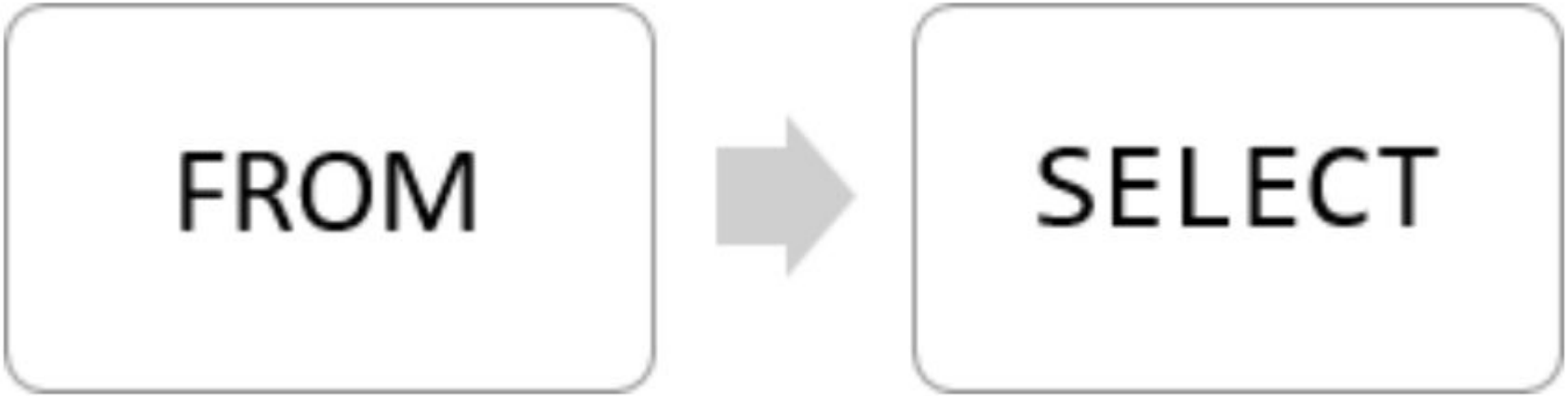
Concatenating between columns

```
SELECT
        first_name || ‘ ‘ || last_name,
        email
FROM
         Customer;
```

In this example, we used the **concatenation operator ||** to concatenate the first name, space, and last name of every customer.

ORDER OF EVALUATION

PostgreSQL evaluates the from clause before the select clause in the select statement



1. Querying data

Select

Order by

Select distinct

ORDER BY

When you query data from a table, the SELECT statement returns rows in an unspecified order. **To sort the rows of the result set, you use the ORDER BY clause** in the SELECT statement.

```
SELECT
        select_list
FROM
        table_name
ORDER BY
        sort_expression1 [ASC | DESC],
        ...
        sort_expressionN [ASC | DESC];
```

ORDER BY

When you query data from a table, the SELECT statement returns rows in an unspecified order. **To sort the rows of the result set, you use the ORDER BY clause** in the SELECT statement.

```
SELECT
        select_list
FROM
        table_name
ORDER BY
        sort_expression1 [ASC | DESC],
        ...
        sort_expressionN [ASC | DESC];
```

EXAMPLE

The following query uses the ORDER BY clause to sort customers by their first names in ascending order:

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

EXAMPLE

The following statement selects the first name and last name from the customer table and sorts the rows by the first name in ascending order and last name in descending order:

```
SELECT
        first_name,
        last_name
FROM
        customer
ORDER BY
        first_name ASC,
        last_name DESC;
```

ORDER OF EVALUATION

PostgreSQL evaluates the clauses in the SELECT statement in the following order: FROM, SELECT, and ORDER BY

1. Querying data

Select

Order by

Select distinct

SELECT DISTINCT

The DISTINCT clause is used in the SELECT statement to remove duplicate rows from a result set.

```
SELECT
    DISTINCT column1
FROM
    table_name;
```

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

Outline

Querying data

Filtering data

Filtering data

WHERE

LIMIT

IN

BETWEEN

LIKE

IS NULL

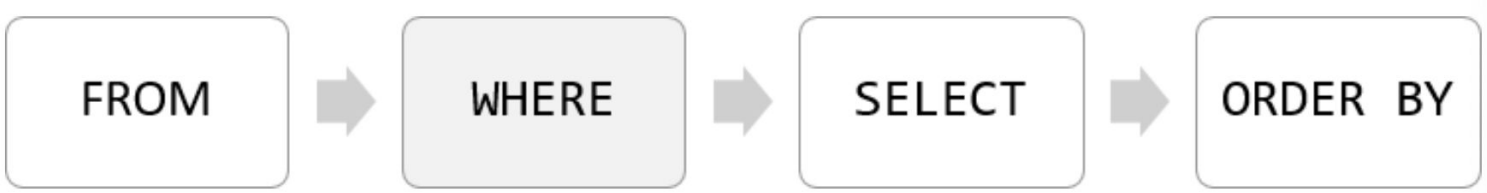
WHERE

The SELECT statement returns all rows from one or more columns in a table. To **select rows that satisfy a specified condition**, you use a **WHERE** clause.

```
SELECT select_list
FROM table_name
WHERE condition
ORDER BY sort_expression
```

ORDER OF EVALUATION

PostgreSQL evaluates the WHERE clause after the FROM clause and before the SELECT and ORDER BY clause:



WHERE clause conditions

Operator	Description
=	Equal
>	Greater than
<	Less than
>=	Greater than or equal
<=	Less than or equal
<> or !=	Not equal
AND	Logical operator AND
OR	Logical operator OR
IN	Return true if a value matches any value in a list
BETWEEN	Return true if a value is between a range of values
LIKE	Return true if a value matches a pattern
IS NULL	Return true if a value is NULL
NOT	Negate the result of other operators

Using WHERE clause with the AND operator

```
SELECT
        last_name,
        first_name
FROM
        customer
WHERE
        first_name = 'Jamie' AND
        last_name = 'Rice';
```

Using WHERE clause with the AND operator

```
SELECT
        last_name,
        first_name
FROM
        customer
WHERE
        first_name = 'Jamie' AND
        last_name = 'Rice';
```

	last_name character varying (45)	first_name character varying (45)
1	Rice	Jamie

Using the WHERE clause with the OR operator

```
SELECT
        first_name,
        last_name
FROM
        customer
WHERE
        last_name = 'Rodriguez' OR
        first_name = 'Adam';
```

Using the WHERE clause with the OR operator

```
SELECT
        first_name,
        last_name
FROM
        customer
WHERE
        last_name = 'Rodriguez' OR
        first_name = 'Adam';
```

	first_name character varying (45)	last_name character varying (45)
1	Laura	Rodriguez
2	Adam	Gooch

Using WHERE clause with the IN operator

If you want to match a string with any string in a list, you can use the **IN** operator.

```
SELECT
        first_name,
        last_name
FROM
        customer
WHERE
        first_name IN ('Ann','Anne','Annie');
```

Using WHERE clause with the IN operator

If you want to match a string with any string in a list, you can use the **IN** operator.

```
SELECT
        first_name,
        last_name
FROM
        customer
WHERE
        first_name IN ('Ann','Anne','Annie');
```

	first_name character varying (45)	last_name character varying (45)
1	Ann	Evans
2	Anne	Powell
3	Annie	Russell

Using the WHERE clause with the LIKE operator

To find a string that matches a specified pattern, you use the **LIKE** operator

```
SELECT
        first_name,
        last_name
FROM
        customer
WHERE
        first_name LIKE 'Ann%'
```

Using the WHERE clause with the LIKE operator

To find a string that matches a specified pattern, you use the **LIKE** operator

```
SELECT
        first_name,
        last_name
FROM
        customer
WHERE
        first_name LIKE 'Ann%'
```

	first_name character varying (45)	last_name character varying (45)
1	Anna	Hill
2	Ann	Evans
3	Anne	Powell
4	Annie	Russell
5	Annette	Olson

Using the WHERE clause with the BETWEEN operator

The BETWEEN operator returns true if a value is in a range of values.

```
SELECT
        first_name,
        LENGTH(first_name) name_length
FROM
        customer
WHERE
        first_name LIKE 'A%' AND
        LENGTH(first_name) BETWEEN 3 AND 5
ORDER BY
        name_length;
```

	first_name character varying (45)	name_length integer
1	Amy	3
2	Ann	3
3	Ana	3
4	Andy	4
5	Anna	4
6	Anne	4
7	Alma	4
8	Adam	4
9	Alan	4
10	Alex	4
11	Angel	5
12	Agnes	5
13	Andre	5
14	Aaron	5
15	Allan	5
16	Allen	5
17	Alice	5
18	Alvin	5
19	Anita	5
20	Amber	5
21	April	5
22	Annie	5

Using the WHERE clause with the not equal operator (!= or <>)

Note that you can use the != operator and <> operator interchangeably because they are equivalent.

```
SELECT
        first_name,
        last_name
FROM
        customer
WHERE
        first_name LIKE 'Bra%' AND
        last_name <> 'Motley';
```

	first_name character varying (45)	last_name character varying (45)
1	Brandy	Graves
2	Brandon	Huey
3	Brad	Mccurdy

Filtering data

WHERE

LIMIT

IN

BETWEEN

LIKE

IS NULL

LIMIT clause

LIMIT is an optional clause of the SELECT statement that constrains the number of rows returned by the query.

```
SELECT select_list
FROM table_name
ORDER BY sort_expression
LIMIT row_count
```

LIMIT clause example

```
SELECT
        film_id,
        title,
        release_year
FROM
        film
ORDER BY
        film_id
LIMIT 5;
```

film
* film_id
title
description
release_year
language_id
rental_duration
rental_rate
length
replacement_cost
rating
last_update
special_features
fulltext

LIMIT clause example

```
SELECT
        film_id,
        title,
        release_year
FROM
        film
ORDER BY
        film_id
LIMIT 5;
```

	film_id integer	title character varying (255)	release_year integer
1	1	Academy Dinosaur	2006
2	2	Ace Goldfinger	2006
3	3	Adaptation Holes	2006
4	4	Affair Prejudice	2006
5	5	African Egg	2006

Filtering data

WHERE

LIMIT

IN → **Covered when we discussed the WHERE clause**

BETWEEN → **Covered when we discussed the WHERE clause**

LIKE → **Covered when we discussed the WHERE clause**

IS NULL → **Covered when we discussed the WHERE clause**