# Database - Epita - intermediate

Sept 3

# Presentations

Alexis Perrier
Data scientist - teacher
Started with mysql in 2000
Teaching : M6, Gustave Eiffel, Openclassrooms, Ynov, …

https://www.linkedin.com/in/alexisperrier/

# Course Logistics

- 10 sessions

- hands-on activities
- real datasets
- readings
- exit tickets
-
- postgreSQL

- evaluation
  - tbd : project or exam
  - small quizzes

- chatGPT, claude.ai, copilot when needed

# Discord

All course content, will be on the discord channel

- course resources
- questions - answers
- office hours

Join the **#Epitadb** channel

https://discord.gg/FQtE7GuFrz

# Github

The github repo will be updated all throughout the course

https://github.com/SkatAI/epitadb (WIP)

# Course goals

What you will be able to do:

- design efficient and scalable databases
- write lightning fast SQL queries
- setup, secure, administer, optimize a database

*(which in turn should bring good health, fortune and happiness )*

# Course Curriculum

- **Advanced Database Design**: understanding of **normalization** and **denormalization** concepts to optimize database structure

- **SQL Query Optimization**: write **high-performance SQL queries**,
  - execution plan analysis,
  - index usage,
  - optimization of joins and subqueries.

- **Index Creation and Management**:
  - Understand the importance of **indexes** in improving database performance
  - learn to create and manage different types of indexes (B-tree, hash, etc.).

# Course Curriculum

- **Views and Stored Functions**:
  - create views to simplify complex queries
  - use stored functions to **encapsulate business logic** within the database - **PLSQL**

- **Transactions and Concurrency Control**: ensure data integrity in multi-user environments.

- **Database Security**: Introduce database security concepts,
  - **access control**,
  - roles and permissions,
  - best practices for protecting sensitive data.

- **Maintenance and Monitoring**: Implement preventive maintenance techniques and monitoring to ensure database availability and performance.

# Course Curriculum

- **Triggers and procedures**
- **CTEs, Window functions**
- **Cloud**: how to setup a db on GCP / Azure / AWS and what are the common cloud services;
- Google BigQuery
- …

# Course Curriculum

By the end of this course, you will be able to:

1. **Design** optimized relational databases for various applications.
2. Write efficient and high-**performance** SQL queries.
3. Create and manage indexes to improve query **performance**.
4. Use views and stored functions to simplify and **optimize data processing**.
5. Manage **transactions** and implement **concurrency** control mechanisms.
6. Apply **security** measures to protect data.
7. Ensure regular **maintenance** and monitoring of databases to prevent performance issues.

# Datasets

1. Trees of Paris from Paris open data
2. Airdb : flights, passengers, airports
3. Ademe: building energy efficiency

# Questionnaire

what's your db level ?

https://docs.google.com/forms/d/1NrPqJK3iwZYpJYY2zKEwqr8Hb7bDpwbi4gw3xZSHMQ4

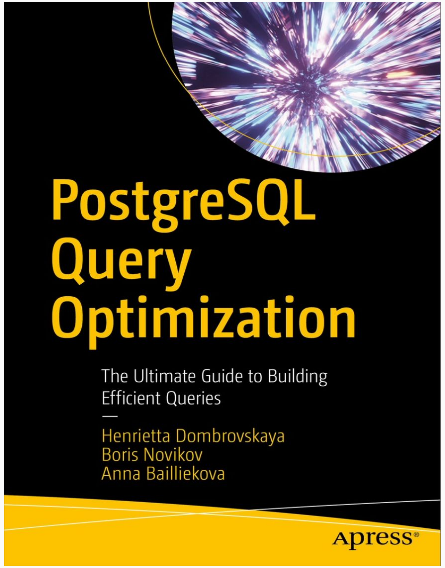
# Today

# Today

Goals

- context : understand why relational database and why **postgresQL**
- **install** postgres on local
- use **psql**
- **create** a database and **load** open data : trees of Paris
- have a working **pgadmin** console
- review common SQL queries
- derive more complex info with **Common Table Expressions** (CTE)

# Resources



PostgreSQL Query Optimization

The Ultimate Guide to Building Efficient Queries

Henrietta Dombrovskaya
Boris Novikov
Anna Bailliekova

Apress

PostgresQL Query Optimization

**Postgres tutorials**

- beginner https://www.youtube.com/playlist?list=PLk1kxccoEnNEtwGZW-3KAcAlhI_Guwh8x
- advanced https://www.youtube.com/playlist?list=PLk1kxccoEnNHIAR2ggnzIkOc7jxqI-_w2
- https://www.geeksforgeeks.org/postgresql-tutorial/
- https://www.postgresql.org/docs/
- https://www.w3schools.com/postgresql/index.php

# Databases & postgresQL

# Why databases ?

organize complex structured data

Spotify example

- users, subscriptions
- devices
- songs
- artists, albums
- playlists
- plays
- etc ...

Social Network example

- users, subscriptions
- followers
- engagement
- posts
- media
- etc ...

Corporate example

- employees,
- salaries
- departments
- holidays
- etc ...

# Why database ?

Discussion : Why use a database instead of just … files like json, or excel / csv ?

Take a few minutes to read these articles

This one has good links: https://medium.com/nerd-for-tech/sql-is-one-of-the-first-things-you-should-learn-as-a-data-business-analyst-a42d1f3cfc11

https://www.geeksforgeeks.org/reasons-why-you-should-learn-sql/

# Types of databases

There are other types of databases than relational databases :

- Vector database for LLMs
  - text is transformed as a vector
  - db is optimized to retrieve similar vectors

- NoSQL database (MongoDB)
  - Data structure is dynamic
  - One data structure
  - Graph databases, key values (json), …

- postgres strikes back:
  - pgvector extension for vectors
  - JSONB data type for key values
  - many extensions, amazing query planner

# PostgresQL rules the world

In terms of relational databases compare

- postgreSQL
- mysql / MariaDB
- SQLite

https://opensource.com/article/19/1/open-source-databases

# PostgresQL and friends

**Extensions :** add-ons that enhance the functionality of a PostgreSQL database.

- provide new types of indexes, data types, procedural languages, or additional functions, thereby extending the core capabilities of PostgreSQL.

Examples include

- **PostGIS** for geographic data,
- **pg_trgm** for text search,
- **hstore** for key-value storage.

https://www.postgresql.org/download/products/6-postgresql-extensions/

# How the guardian moved from MongoDB to postgres

Why and how did the Guardian move from MongoDB to Postgres

1. Take a few minutes to read the article

https://www.theguardian.com/info/2018/nov/30/bye-bye-mongo-hello-postgres

2. if you have any, write down your questions

# How the guardian moved from MongoDB to postgres

**Reasons for Moving from MongoDB to PostgreSQL:**

1. **Operational Challenges**: The Guardian faced significant issues with MongoDB's OpsManager, including time-consuming upgrades, lack of effective support during outages, and the need for extensive custom scripting and management.
2. **Cost and Efficiency**: The high cost of MongoDB's support contract combined with the ongoing operational burden led them to seek a more manageable and cost-effective solution.
3. **Feature Limitations**: Alternatives like DynamoDB were considered but lacked essential features like encryption at rest, which Postgres on AWS RDS provided.

**Migration Process:**

1. **Parallel APIs**: They created a new API using PostgreSQL and ran it in parallel with the old MongoDB API to ensure a smooth transition.
2. **Data Migration**: Content was migrated using a script that compared and validated data between the two databases.
3. **Proxy Usage**: A proxy was employed to replicate traffic to both databases, ensuring consistency and allowing for real-time testing.
4. **Gradual Switchover**: The team gradually shifted traffic to the new Postgres API, eventually decommissioning MongoDB without causing downtime.

https://www.theguardian.com/info/2018/nov/30/bye-bye-mongo-hello-postgres

# Install postgres & pgadmin

# Now the fun part

Install **PostgreSQL 16** on Local

https://www.postgresql.org/download/
https://www.enterprisedb.com/downloads/postgres-postgresql-downloads

Windows :
https://www.postgresqltutorial.com/postgresql-getting-started/install-postgresql/

if you run into problems write it down in the doc
https://docs.google.com/document/d/1mX9-5-PeN0QD7OwsRSvTkJ32iwHqoKC7mtHQ8Aw5Cbk/edit?usp=sharing

# start, stop, check, connect

make sure you know how to

- start and stop postgres
- check that postgres is running
- connect with psql in the terminal
- list users with \du : you should see 2 users
  - postgres
  - your name

# start, stop, check, connect : on mac

- start
    - brew services start postgresql@16
- stop
    - brew services stop postgresql@16
- check that postgres is running
    - launchctl list | grep postgres
- connection with psql in the terminal
    - psql -U postgres

# start, stop, check, connect : on windows

- start
  - …
- stop
  - …
- check that postgres is running
  - …
- connection with psql in the terminal
  - …

# psql and command prompts

# psql specific prompts

- connect on local as postgres user with psql
- try these prompts
- figure out what they return

```
# \d
# \dt
# \dn
# \df
# \du
# \q
```

```
# \d table_name
```

https://commandprompt.com/education/postgresql-basic-psql-commands/
https://tomcam.github.io/postgres/

# psql specific prompts

connect with

```
psql -h 35.238.75.182 -U epita -d airdb
```

with password

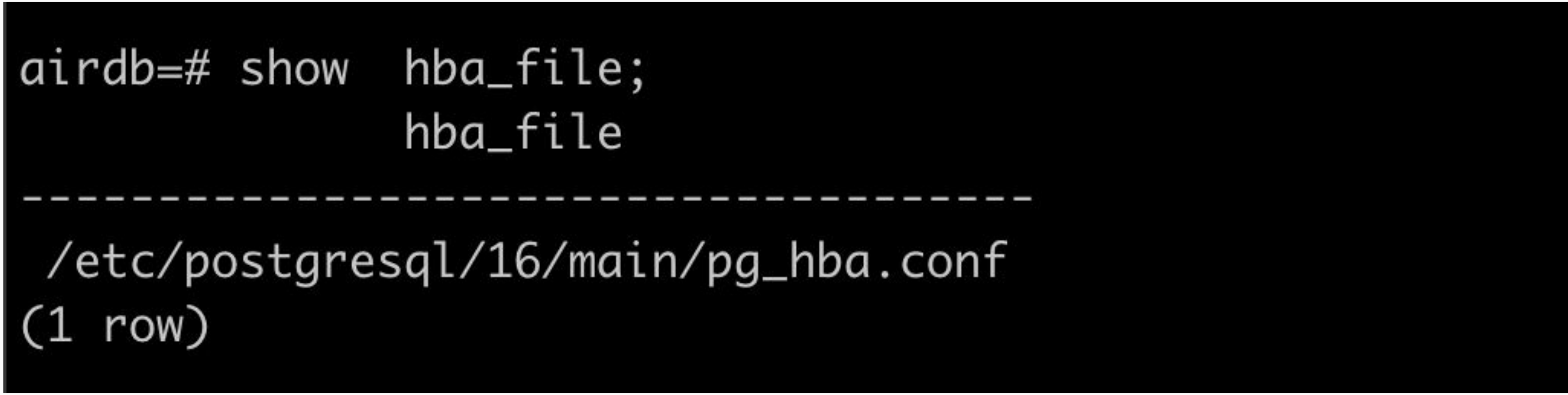
epita_2024

Let's go through https://tomcam.github.io/postgres/ on the airdb database

# postgres configuration files

There are 2 configuration files for a postgres server

- postgresql.conf : manages how the server operates
- pg_hba.conf : manages who can connect and how they authenticate



```
airdb=# show  hba_file;
              hba_file
------------------------------------
 /etc/postgresql/16/main/pg_hba.conf
(1 row)
```

# postgres configuration files

General server configuration

This file controls most of the global settings for the PostgreSQL server. It includes:

- Resource allocation (memory, CPU)
- Default storage locations
- Replication settings
- Client connection defaults
- Query planner settings
- Logging and statistics
- Autovacuum settings
- Client/server communication parameters
- Locale and formatting
- Error handling

Key points:
- Affects the overall behavior and performance of the PostgreSQL server
- Changes typically require a server restart to take effect
- Located in the data directory

Example settings:

```
max_connections = 100
shared_buffers = 128MB
log_destination = 'stderr'
```

# postgres configuration files

2. pg_hba.conf

Role: Client authentication control

This file controls how clients are allowed to connect to the server. "HBA" stands for "host-based authentication". It specifies:

- Which hosts can connect
- Which database they can connect to
- Which PostgreSQL user names they can use
- How clients are authenticated (password, ident, trust, etc.)

Key points:
- Controls access at a very granular level
- Changes can typically be loaded with a simple reload, not requiring a full restart
- Critical for security management
- Also located in the data directory

Example entries:

```
# TYPE  DATABASE        USER            ADDRESS                 METHOD
local   all             postgres                                peer
host    all             all             127.0.0.1/32            md5
host    production      app_user        192.168.1.0/24          scram-sha-256
```

# setup psql with .psqlrc

**.psqlrc** is a configuration file for the psql command-line interface in PostgreSQL.
It allows you to **customize your psql environment** and set default behaviors.

- Usually located in your home directory: `~/.psqlrc` on Unix-like systems
- On Windows: %APPDATA%\postgresql\psqlrc.conf

- Customizes the psql environment
- Sets default options and behaviors
- Runs commands automatically when psql starts

# setup psql with .psqlrc

- always timing the queries
- pager mode
- expanded mode

```
1.   Set default pager: \pset pager always
2.   Set line style: \pset linestyle unicode
3.   Set timing on: \timing
4.   Set expanded auto mode: \x auto
5.   Custom prompt:
     \set PROMPT1 '%[%033[1m%]%M %n@%/%R%[%033[0m%]%# '
6.   History settings:
     \set HISTSIZE 2000
     \set HISTCONTROL ignoredups
7.   Enable verbose error reports: \set VERBOSITY verbose
```

# Let's explore Paris data - practice 1

Choose a dataset from the Paris open data platform
https://opendata.paris.fr/pages/catalogue/
download the dataset
create the table
load the dataset into the table

see
https://docs.google.com/document/d/1_UIrC1C651sv7rRQ4nvn2DqhjOPfSw7fhobXexS1LwM/edit

# Go over basic sql querying

where, groupby, order, distinct, count(*), …
Union,
subqueries

write the queries for these questions

- how many
- list the distinct …
- find …

# Let's explore Paris - query

Once the data is in the database, explore the dataset

- top 6 most common tree names
- number of remarquable trees by arrondissement
- height and circumference outliers
- mean dimensions per stage

# pgAdmin

# Install pgAdmin

- install pgAdmin
- connect to the local server
- psql and query tool
- https://www.pgadmin.org/download/

# Common Table Expressions

# More complex queries with CTEs

define temporary result sets that can be referenced within another SQL statement

```
WITH cte_name (column_list) AS (
    CTE_query_definition
)
  statement;
```

https://www.geeksforgeeks.org/postgresql-cte/

# More complex queries with CTEs

find the average circumference and height of trees in each arrondissement and then filter for arrondissements where the average circumference is greater than 100.

find the most common tree species in each arrondissement and then filter to only show arrondissements where the most common species accounts for more than 30% of the total trees.

identifying the top 5 tallest trees in each arrondissement. We'll use a CTE to first rank the trees by height within each arrondissement and then filter to get only the top 5 tallest trees per arrondissement.

# More complex queries with CTEs

```
WITH arrondissement_stats AS (
   SELECT
      arrondissement,
      AVG(circumference) AS
avg_circumference,
      AVG(height) AS avg_height,
      COUNT(*) AS tree_count
   FROM
      trees
   GROUP BY
      arrondissement
)
SELECT
   arrondissement,
   avg_circumference,
   avg_height,
   tree_count
FROM
   arrondissement_stats
WHERE
   avg_circumference > 100
ORDER BY
   avg_circumference DESC;
```

find the average circumference and height of trees in each arrondissement
then filter for arrondissements where the average circumference is greater than 100.

**CTE Definition (`arrondissement_stats`)**:

- This CTE calculates the average circumference (`avg_circumference`) and average height (`avg_height`) of trees for each arrondissement.
- It also counts the number of trees in each arrondissement (`tree_count`).
- The results are grouped by the `arrondissement`.

**Main Query**:

- The main query selects data from the CTE `arrondissement_stats`.
- It filters arrondissements where the average circumference is greater than 100.
- Finally, it orders the results by `avg_circumference` in descending order.

# More complex queries with CTEs

find the most common tree species in each arrondissement and then filter to only show arrondissements where the most common species accounts for more than 30% of the total trees.

use the following window function:

COUNT(*)::decimal / SUM(COUNT(*)) OVER (PARTITION BY arrondissement) AS species_percentage

```
WITH species_count AS (
   SELECT
      arrondissement,
      species,
      COUNT(*) AS species_count,
      COUNT(*)::decimal / SUM(COUNT(*)) OVER (PARTITION BY arrondissement) AS species_percentage
   FROM
      trees
   GROUP BY
      arrondissement, species
),
most_common_species AS (
   SELECT
      arrondissement,
      species,
      species_count,
      species_percentage
   FROM
      species_count
   WHERE
      species_percentage > 0.30
)
SELECT
   arrondissement,
   species,
   species_count,
   species_percentage
FROM
   most_common_species
ORDER BY
```

**CTE Definition (`species_count`)**:

- This CTE calculates the number of trees of each species within each arrondissement (`species_count`).
- It also calculates the percentage of trees of that species relative to the total number of trees arrondissement (`species_percentage`).
- The percentage is calculated using a `COUNT(*)::decimal` divided by the sum of `COUNT(` the partitioned rows for each arrondissement.

**CTE Definition (`most_common_species`)**:

- This CTE filters the results from `species_count` to only include species that account for m than 30% of the total trees in an arrondissement.

**Main Query**:

- The main query selects the filtered data from `most_common_species`.
- It orders the results by `arrondissement` and then by `species_percentage` in descend order.

# More complex queries with CTEs

identify the top 5 tallest trees in each arrondissement. We'll use a CTE to first rank the trees by height within each arrondissement and then filter to get only the top 5 tallest trees per arrondissement.

```
WITH tree_ranks AS (
   SELECT
      idbase,
      arrondissement,
      name,
      genre,
      species,
      height,
      ROW_NUMBER() OVER (PARTITION BY arrondissement ORDER BY height DESC) AS rank
   FROM
      trees
   WHERE
      height > 0
)
SELECT
   idbase,
   arrondissement,
   name,
   genre,
   species,
   height,
   rank
FROM
   tree_ranks
WHERE
   rank <= 5
ORDER BY
   arrondissement,
   rank;
```

**CTE Definition (`tree_ranks`)**:

- The CTE calculates the rank of each tree within its arrondissement based on its height.
- `ROW_NUMBER() OVER (PARTITION BY arrondissement ORDER BY height DESC)`:
  - `PARTITION BY arrondissement`: Divides the data into partitions by arrondissement.
  - `ORDER BY height DESC`: Orders the trees in each arrondissement by their height in descending order.
  - `ROW_NUMBER()`: Assigns a unique sequential integer to rows within each partition (arrondissement) base order specified.
- This CTE only includes trees with a positive height (`WHERE height > 0`).

**Main Query**:

- The main query selects all columns from the `tree_ranks` CTE.
- It filters to include only the top 5 tallest trees in each arrondissement (`WHERE rank <= 5`).
- The results are ordered by arrondissement and then by rank within each arrondissement.

That's all folks :)
Thank you

# Recap

What we saw today

- you're all setup with a local install of postgresQL
- Why use postgresQL
- loading data into a newly create db and tables
- simple querying & CTEs
-

# Exit ticket

That's all for today
I need your feedback to improve the course
Exit ticket
https://forms.gle/7yTmpP2jW1EHMhgE6

# Next time

How to organize the data in tables and columns
Normalization
Olap vs OLTP