

Business Intelligence (BI)

DuckDB

Réalisé par :








ESSAFI Amina

OUZZIKI Dounya

Encadré par:

Mr.NAJDI Lotfi

Les axes :

-  **Introduction**
-  **Définition DuckDB**
-  **Caractéristiques principales**
-  **Cas d'usage**
-  **Avantages et limites**
-  **Application pratique**
-  **Conclusion**

Introduction

Imaginez ceci : vous êtes data scientist, et vous devez analyser **2,3 millions de transactions commerciales** en une fraction de seconde, directement sur **votre ordinateur portable**, sans serveur ni configuration complexe.

=> Positionnement de DuckDB dans l'écosystème actuel:

Dans l'écosystème des données, les outils traditionnels montrent leurs limites quand on travaille avec ce qu'on appelle les “**données moyennes**” (ou *medium data*) :

- **Trop volumineuses pour Excel, Pandas ,**
- **Pas assez massives** pour justifier l'utilisation de solutions Big Data comme **Spark, Hadoop,**
- Il comble une **niche ignorée** par les outils traditionnels : performance, simplicité.

Qu'est-ce que DuckDB ?

DuckDB est un **SGBD analytique (OLAP) embarqué** (il fonctionne localement, sans serveur).

Sa philosophie s'inspire de SQLite, mais orientée vers l'analyse de données à grande échelle :

→ La rencontre entre **"SQLite et l'analytique moderne"**.

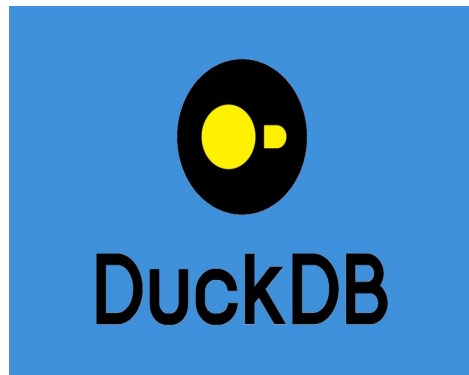
Il se positionne ainsi comme

- Un moteur SQL haute performance,
- Optimisé pour l'analyse locale ,
- Intégré dans les outils de data science.

=> Schéma de positionnement :

Dans le paysage des bases de données, DuckDB occupe la case du **OLAP local**, là où SQLite est utilisé pour l'OLTP local, et des solutions comme Snowflake ou BigQuery dominant le OLAP distribué.

	Local	Distribué
OLTP (transactionnel)	SQLite	PostgreSQL, MySQL
OLAP (analytique)	DuckDB	Snowflake, BigQuery



Caractéristiques principales

Intégration universelle :

- APIs natives Python, Java, Node.js
- Lecture directe CSV, JSON
- Compatible avec les notebooks Jupyter

Caractéristiques

Exécution intelligente :

- Vectorisation des requêtes.
- Parallélisme multi-cœurs transparent.
- Optimiseur de requêtes moderne.

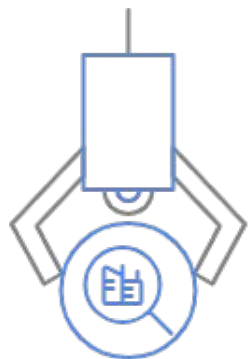
Stockage optimisé :

- Architecture columnar
- Compression native des données.
- Index adaptatifs automatiques.

Déploiement zéro-friction :

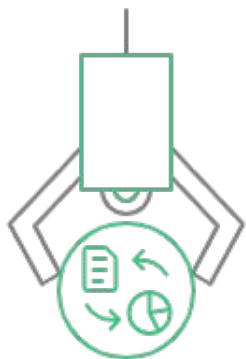
- Installation en une ligne de commande.
- Pas de configuration serveur.
- Base de données fichier unique (duckdb)

Cas d'usage :



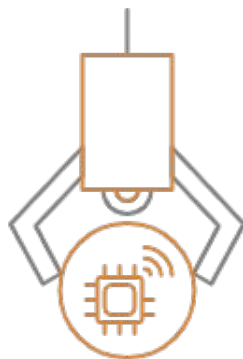
Exploration de données locale

Analyser des millions de lignes de manière indépendante



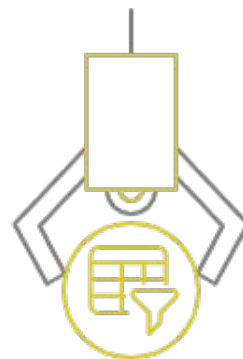
Prototypage BI rapide

Tester des modèles de données avant déploiement



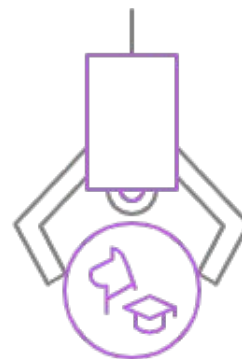
Analyse en périphérie

Léger et intégré pour une analyse locale



ETL local

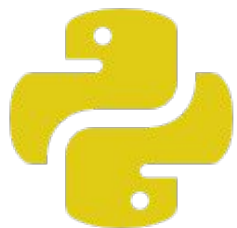
Transformer de gros fichiers sans infrastructure



Formation et recherche

Enseigner facilement l'analytique SQL avancée

Avantages



Écosystème Python/R

S'intègre
naturellement dans
les workflows de
data science.



Évolutivité douce

Grandit avec les
besoins sans refonte
d'architecture.



Polyvalence des formats

Analyse directe sans
ETL préalable
(schema-on-read).



Simplicité opérationnelle

Démarrage
instantané,
maintenance nulle
requis pour les
opérations.



Performance remarquable

Rivalise avec des
solutions
d'entreprise même
sur un ordinateur
portable.

Limites

Limitation single-machine

Pas de distribution
native (no native
clustering)

Maturité relative

L'écosystème est
moins riche que
PostgreSQL ou
MySQL.

Support entreprise

Moins d'intégrateurs
et de consultants
spécialisés sont
disponibles.

Concurrence limitée

Optimisé pour la
lecture, écriture
concurrente
restreinte.

Exemple pratique : Analyse de données avec DuckDB

Exemple d'analyse de données avec DuckDB, une base de données optimisée pour de grandes requêtes SQL. Il utilise le dataset "E-commerce Sales Data 2024", contenant des informations sur les clients, les produits et leurs interactions, afin d'étudier le comportement des acheteurs et les ventes à grande échelle.

L'objectif :

- Charger le dataset dans un DataFrame Pandas.
- Analyse Business avec DuckDB
- Mesure de performance

❖ Importation des bibliothèques et Chargement du dataset

```
import duckdb
import time
import pandas as pd

# Connexion à une base locale
con = duckdb.connect('ecommerce.duckdb')
```

❖ Chargement des fichiers CSV dans des vues

```
con.execute("""
CREATE OR REPLACE VIEW sales AS
SELECT * FROM read_csv_auto('E-commerce_2024.csv')
""")

df_products=con.execute("""
CREATE OR REPLACE VIEW products AS
SELECT * FROM read_csv_auto('product_details.csv')
""")

con.execute("""
CREATE OR REPLACE VIEW customers AS
SELECT * FROM read_csv_auto('customer_details.csv')
""")
```

❖ Requêtes d'analyse

Le code exécute quatre requêtes SQL distinctes pour analyser les données.

Chaque requête utilise `connect.execute()` pour exécuter une requête SQL, et `fetchdf()` pour récupérer les résultats sous forme de DataFrame Pandas

```
print("• Interactions par type :")
df1 = con.execute("""
SELECT "Interaction type", COUNT(*) AS total
FROM sales
GROUP BY "Interaction type"
ORDER BY total DESC
""").fetchdf()
print(df1)
```

```

# B. Top 10 des produits les plus achetés
print("\n • Top 10 produits les plus achetés :")
df2 = con.execute("""
SELECT p."Product Name", COUNT(*) AS nb_ventes
FROM sales s
JOIN products p ON s."user id" = p."Uniqe Id"
WHERE s."Interaction type" = 'Purchase'
GROUP BY p."Product Name"
ORDER BY nb_ventes DESC
LIMIT 10
""").fetchdf()

# C. Répartition des achats par genre
print("\n • Répartition des achats par genre :")
df3 = con.execute("""
SELECT c.Gender, ROUND(AVG(c.Age), 1) AS age_moyen, COUNT(*) AS nb_achats
FROM customers c
GROUP BY c.Gender
ORDER BY nb_achats DESC
""").fetchdf()

# D. Revenus par catégorie de produit
print("\n • Revenus par catégorie :")
df4 = con.execute("""
SELECT p.Category, SUM(c."Purchase Amount (USD)") AS revenus
FROM customers c
JOIN products p ON c."Item Purchased" = p."Product Name"
GROUP BY p.Category
ORDER BY revenus DESC
""").fetchdf()

```



Affichage des résultats

```
• Interactions par type :  
Interaction type  total  
0             like   1145  
1             view    871  
2          purchase    855  
3             None    423  
  
• Top 10 produits les plus achetés :  
Empty DataFrame  
Columns: [Product Name, nb_ventes]  
Index: []  
  
• Répartition des achats par genre :  
Gender  age_moyen  nb_achats  
0    Male      44.1      2652  
1  Female      44.0      1248  
  
• Revenus par catégorie :  
Empty DataFrame  
Columns: [Category, revenus]
```


❖ Mesure de performance

Mesure le temps d'exécution
d'une requête SQL simple
(comptage des interactions par
type) avec DuckDB.

Mesure le temps d'exécution de
la même opération

(comptage des interactions par
type) avec Pandas.

```
#Mesure de performance
start = time.time()
con.execute("""
SELECT "Interaction type", COUNT(*)
FROM sales GROUP BY "Interaction type"
""").fetchall()
end = time.time()
print(f"Temps d'exécution DuckDB : {round(end - start, 4)} secondes")

#Comparaison avec Pandas
start = time.time()
df = pd.read_csv('E-commerce_2024.csv')
df.groupby("Interaction type").size()
end = time.time()
print(f"Temps d'exécution Pandas : {round(end - start, 4)} secondes")
```

La comparaison porte sur le **temps d'exécution d'une même opération** :

- Compter le nombre d'interactions par type, en utilisant **DuckDB** et **Pandas**.
 1. D'abord, DuckDB exécute une requête SQL directement sur le fichier CSV, et on mesure le temps nécessaire.
 2. Ensuite, Pandas lit le même fichier et effectue un regroupement par type d'interaction.

Les résultats montrent que:

- DuckDB prend **0,0188 secondes**,
- Pandas est légèrement plus rapide avec **0,0126 secondes**.

Cela montre que **les deux outils sont très performants**, même avec des fichiers volumineux.

DuckDB reste très compétitif, pour des analyses SQL complexes.

Conclusion :

En résumé, DuckDB démocratiser l'analytique haute performance en la rendant accessible à tous, sans infrastructure complexe.

Sa vision est :

- Transformer chaque développeur en analyste de données autonome,
- Chaque ordinateur portable en un véritable mini-entrepôt analytique.

Son impact sur l'écosystème BI est significatif, en

- Réduisant la complexité,
- Accélérant les cycles d'expérimentation ,
- Ouvrant l'accès aux données à un public plus large.

À l'avenir, DuckDB poursuit son évolution avec

- Le support croissant des **formats cloud** comme Delta Lake ou Iceberg,
- Amélioration du **streaming** (real-time streaming),
- Des **intégrations natives avec l'IA et le machine learning**, affirmant ainsi sa place dans les outils data de demain.

Merci pour votre attention