DuckDB を使用したデータ可視化技術

自己紹介

- nk
- 業務:製造会社でExcel,Sharepoint,csv→データ前処理、PowerBIでデータ可視化、アプリケーション作成などなど
- 最近DuckDBの記事投稿を主にしています。
- Tableauなども最近さわってます。(先日なんとかDATASaberになれました。)

DuckDB とは 🦆

- Sqlite をデータ分析用に改造したRDBMS...と思いきや???
- 基本Sqliteと同じようにローカルで使用できる(一部拡張機能インストールにWeb接続必要)
- 正直sqliteのように純粋なデータベースとして使うことはあんまりない。高速なデータ処理に使用。(pandasのようなやつ)
- 様々なデータソース(AWS S3,parquet,iceberg,http)と数行でimport/exportできる守備範囲の広さ
- DuckDB-wasm・DuckDB UIデータ可視化、Streamlit連携、ベクトルデータベース???🤯

よくわからないまま色々作ってみよう!

...という感じで色々作っているので紹介します。

例 1: AWS Lambda + DuckDB → S3(Parquet)

AWS LambdaでS3上のParquetファイルを処理

```
import duckdb
def lambda_handler(event, context):
    con = duckdb.connect(database=':memory:')
    con.execute("SET home_directory='/tmp';")
    con.execute("INSTALL httpfs;")
    con.execute("LOAD httpfs;")
    test = con.sql(r'''
    SELECT * FROM read_csv("s3://your-bucket/your-file.csv")''').df()
    print(test.head())
```

料金試算(間違いがあるかも...)

項目	詳細	計算式	料金
Lambda 実行料金	実行時間: 30,000 秒 メモリ割り当て: 0.5 GB	30,000 × 0.5 GB × $0.0000166667 / GB-秒	$0.25
S3 リクエスト料金	GET リクエスト数: 3,000	3,000 × ($0.0004 / 1,000)	$0.0012
合計料金（Lambda + S3）			$0.25/月

最大の使用メモリは200MB、実行時間はだいたい6秒(数行の場合)

例 2: AWS Lambda → S3 Tables(Iceberg)


```
import json
import duckdb
def lambda_handler(event, context):
    # 1
    con = duckdb.connect(database=':memory:')
    con.execute("SET home_directory='/tmp';")
    # 2
    con.sql("FORCE INSTALL aws FROM core_nightly")
    con.sql("FORCE INSTALL httpfs FROM core_nightly")
    con.sql("FORCE INSTALL iceberg FROM core_nightly")
    con.sql("LOAD aws")
    con.sql("LOAD httpfs")
    con.sql("LOAD iceberg")
```

悩み事

- 結果的にはicebergのテーブル表示までできました。
- Lambda上で使用するために毎回リポジトリインストールやCREATE SECRETの実行をやるのはちょっと問題...
- このLambda上のクエリで15秒くらいかかる...

例 3:列数の異なる csv を duckdb で読む

列数の異なるcsvファイルとは🤔

```
6, 54
58,81,62
75,84,64,21,55
20,71,55,32
```

こういうやつです。機器から吐き出されるcsvファイルで一行目より下位の行の方が多かったり…🫠

実行結果

column0 int64	column1 int64	column2 int64	column3 int64	column4 int64
6	54	NULL	NULL	NULL
58	81	62	NULL	NULL
75	84	64	21	55
20	71	55	32	NULL

勝手に最大列数までnullで補完してからしかも最大列数分のヘッダーまでつけてくれます。

例 4: AWS CUR 可視化(DuckDB UI)

duck>

例 5:Streamlit との連携

DuckDBでWeb上にあるデータ(csv,gzファイル)に接続→Streamlitに渡す。

例 6:DuckDB-wasm で csv parquet 変換

htmlファイル一枚にまとまるのでこれを渡すだけで良い。


csv

例 7:ベクトルデータベースとして使用


```
import duckdb
conn = duckdb.connect(database=':memory:',
    config={
            "enable_external_access": "false",
            "autoinstall_known_extensions": "false",
            "autoload_known_extensions": "false"
        }
)
embedding_function = ... # langchainの使用など...省略
vector_store = DuckDB(conn, embedding_function)
vector_store.add_texts(['text1', 'text2'])
result = vector_store.similarity_search('text1')
```

まとめ

- さわってると色々な分野のツールを作れる!
- (実務ではやっていないですが)でもデータエンジニアからの目線だと...?
- 色々なことが逆に出来すぎてしまう。
- セキュリティ面:勝手にデータを抜いて加工できてしまう。
- データ基盤と連携していない野良ツールが出来てしまう。
- データの民主化→無秩序化につながる?

このあたりを実務でデータエンジニアをされている方にお聞きできればと思います。

ご清聴ありがとうございました。

参考記事一覧

- [例 1: AWS Lambda + DuckDB → S3(Parquet)](https://zenn.dev/amana/articles/7651ec03bb6c3e)
- [例 2: AWS Lambda + DuckDB → S3 Tables(Iceberg)](https://zenn.dev/amana/articles/4a68a4d381cfe5)
- [例 3:pandasで読めない列数の異なるcsvをduckdbで読む](https://zenn.dev/amana/articles/61599eeb1365f4)
- [例 4: S3上のAWS CURのファイル内容を可視化(DuckDB UI)](https://zenn.dev/amana/articles/623cf9deeec5e9)
- [例 5:Streamlitとの連携](https://zenn.dev/amana/articles/a5236d47630380)