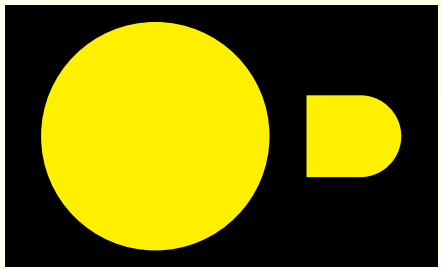
DuckDB

Uma revolução pra quem trabalha com dados

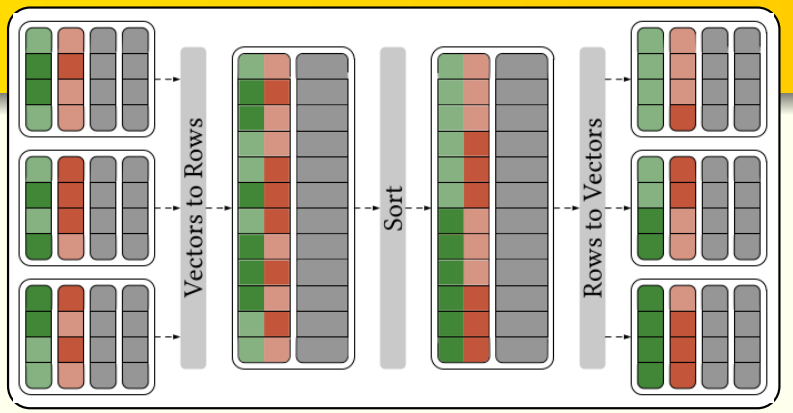
Danilo J. S. Bellini
@danilobellini

2024-07-06



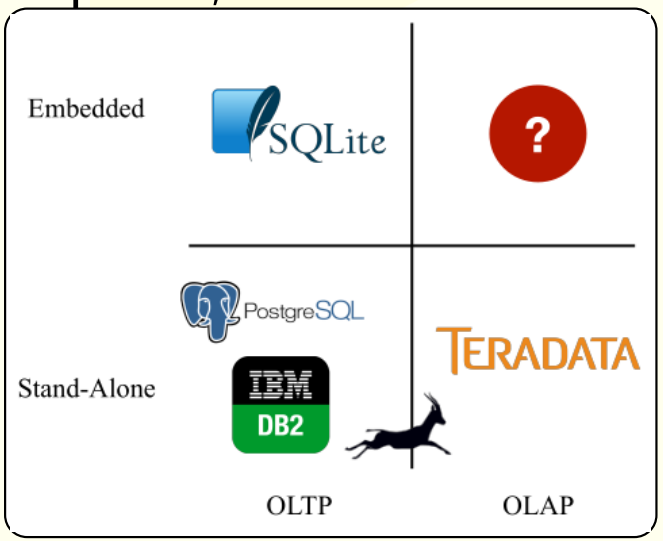
Python Sudeste 2024 @ UFSCar/SP

RDBMS? OLAP vs OLTP... ACID?



O que é o DuckDB?

- **Banco de dados** relacional (RDBMS)
- OLAP (*OnLine* ***Analytical*** *Processing*)
  - Otimizado para análise de grandes volumes de dados
  - Organização interna por "colunas"
- Pode processar carga maior que a memória
- Suporta transações ACID (Atômicas, Consistentes, Isoladas e Duráveis), embora não seja otimizado para OLTP
- Muitas possibilidades de integração (tipos de arquivos, locais de origem, outros bancos de dados)
- Suporta **SQL** (e um SQL amigável próprio)
- ***In-process*** (roda no processo *host*)
- Portável (Linux, Windows, macOS, WASM)
- Software livre/aberto (licença MIT)



Sobre os ombros de gigantes!

Inspirado em muitas pesquisas acadêmicas, surgiu como parte de pesquisas realizadas na Holanda. O "*may not be obvious at first unless you're Dutch*" do PEP-20 (Zen do Python) se encaixa aqui?

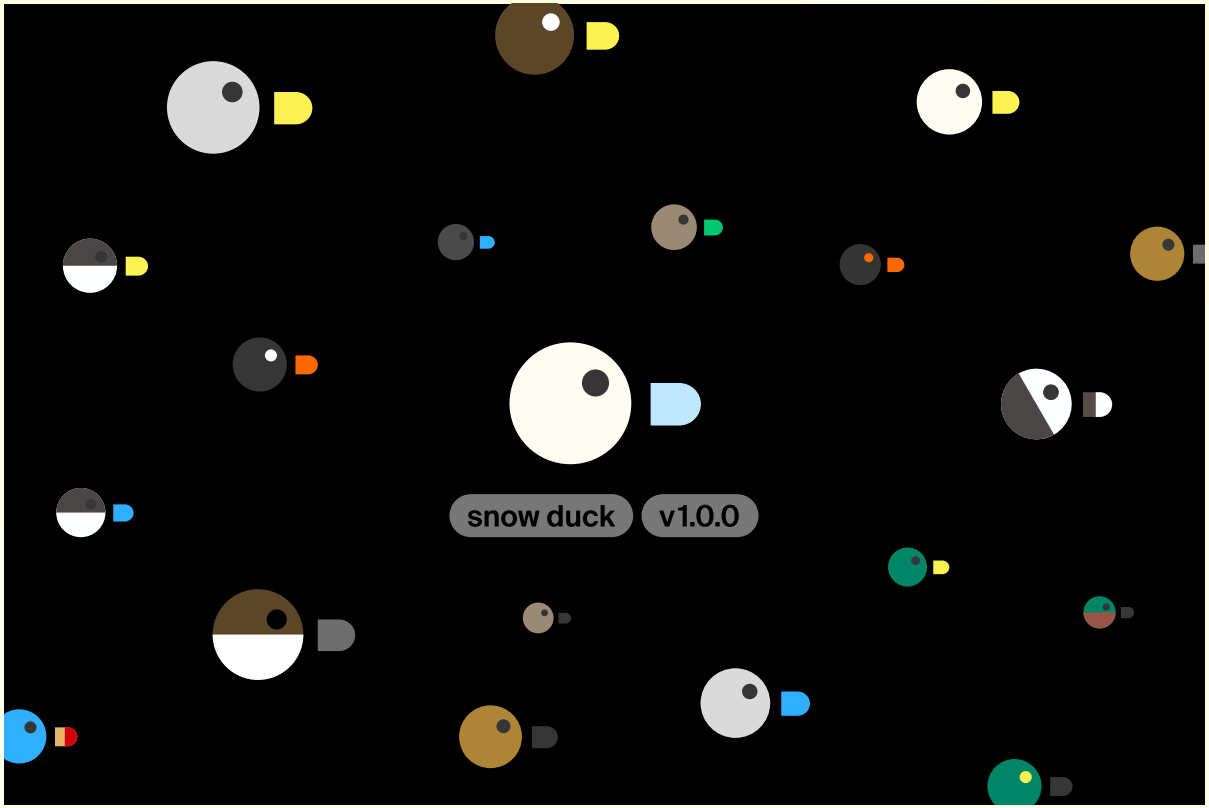
Elementos do **SQLite** (RDBMS mais usado no mundo) que foram trazidos para o DuckDB:

- Não possuir dependências externas (exceto p/ extensões)
- Rodar no mesmo processo *host*
- Shell/REPL (linha de comando) e testes de lógica SQL
- Habilidade em salvar arquivos com extensão "`.db`" ou manter em memória com o nome "`:memory:`"

O parser SQL é um fork do parser do **PostgreSQL**, e diversos elementos de sintaxe são os mesmos, como "`::TIPO`" para conversões de tipos, "`~`" e "`!~`" para expressões regulares (POSIX), sintaxe para uso de colunas JSON e até mesmo funções.

DuckDB v1.0.0 “Snow Duck” (Anas Nivis)

- Primeira versão estável, lançada em 2024-06-03!
- Quase 6 anos de DuckDB (primeiro commit em 2018-07-13)
- Licença MIT desde 2018-12-04
- Escrito em C++, disponível no PyPI (também possui integrações com outras ferramentas/linguagens, como o `dplyr` do R, mas Python possui mais recursos)



Cadê o SQL? *Show me the code*!

"Olá mundo!" no Shell local ou `https://shell.duckdb.org`:

```sql
SELECT 'Hello World' AS msg;  -- ";" é necessário no shell
```

Outros exemplos introdutórios:

```sql
-- DuckDB detecta (e converte) os tipos em muitos casos
VALUES (1, '2', 0, 1/4, true), (5, 'test', .25, 0.78, NULL);

-- Podemos converter tipos e aninhar queries/subqueries
SELECT x, y, y - x AS z FROM (
  SELECT
    col0::INT64 AS x,
    col1::INT64 AS y,  -- Trailing comma, é permitido!
  FROM (VALUES (1, 5), (3, 17), (4, 33), (7, 257))
);

-- Colunas podem ser do tipo lista, mas podemos "desaninhar"
SELECT range(15);
SELECT unnest(range(15)) AS idx;  -- Lista -> linhas
SELECT unnest([  -- List comprehension!
  format('{:2d} ** 2 -> {:03d}', x, x * x) FOR x IN range(15)
]) AS values;
```

Tabelas/views em memória; Janela; Arquivos CSV

```sql
-- Criando a partir de dados constantes
CREATE TABLE ir AS SELECT  -- Imposto de Renda CDB (e outros)
  col0 AS aliquota,
  col1 AS ate_dias_div_10,
FROM (VALUES (.225, 18), (.2, 36), (.175, 72), (.15, NULL));

-- Statements para "sondar" o que houve na criação
SHOW DATABASES;  -- Um único "catalog": a memória (memory)
SHOW TABLES;  -- Apenas a tabela que acaba de ser criada
FROM ir;  -- O "SELECT *" é implícito

-- Eis uma VIEW (aspas duplas em nomes de coluna c/ espaço)
CREATE OR REPLACE VIEW tabela_ir_completa AS SELECT
  format('{:.1f}%', aliquota * 100) AS "Alíquota CDB",
  ifnull(  -- Ou coalesce
    first(ate_dias_div_10) OVER (  -- Agregação por janela
      ORDER BY ate_dias_div_10
      ROWS BETWEEN 1 PRECEDING AND 1 PRECEDING
    ) * 10 + 1,
    0  -- Chamada de função não pode ter trailing comma
  ) || ' dias' AS "A partir de",  -- || concatena strings
  ifnull(ate_dias_div_10 * 10 || ' dias', 'Inf') AS "Até",
FROM ir;

-- Tabelas e views podem ser exportadas para CSV ou Parquet
FROM tabela_ir_completa;
COPY tabela_ir_completa TO 'tabela_ir_completa.csv';
FROM 'tabela_ir_completa.csv';  -- Lendo direto do CSV!
```

E no Python? HTTP?

```python
import duckdb

# Rodando um SQL sem parâmetros em duckdb.default_connection
duckdb.sql("SELECT 'Outro ' || 'hello world' AS msg")

# O ideal é criar uma conexão para isolar e facilitar testes
# (Use conn.sql(...).show() caso queira exibir em um script)
conn = duckdb.connect(":memory:")  # Podia ser um "arquivo.db"
conn.sql("SELECT {'mais': 1, 'hello': 'world'} AS struct_msg")

# Algo similar, mas com parâmetros, devolvendo uma tupla
conn.execute("SELECT $primeiro + $segundo", {
    "primeiro": 2,
    "segundo": 1.2,
}).fetchone()  # .fetchall() devolveria lista de tuplas

# Os tipos devolvidos são diferentes dependendo do método
#   .execute -> DuckDBPyConnection (conexão e cursor)
#   .sql -> DuckDBPyRelation (resultado de uma query)
# Mas ambos possuem fetchone/fetchall e outros que veremos

# O arquivo CSV não precisa estar na sua máquina!
conn.sql("FROM read_csv('https://t.ly/gdUfb')")
# Funciona com a URL resolvida do shortener? Sem o read_csv?

conn.sql("""-- Ao invés de fazer um download a cada query...
CREATE TABLE iris AS FROM read_csv('https://t.ly/gdUfb')
""")
```

Instalação e configuração do ambiente; Exercício!

```
python -m venv venv  # Criação de um ambiente virtual
. venv/bin/activate  # Ou venv\Scripts\Activate.ps1 no Windows

pip install duckdb                    # Não possui dependências!
pip install fsspec                    # Acesso a arquivos em ZIP
pip install pandas polars pyarrow     # Integrações (dados)
pip install notebook jupysql          # Jupyter + plugin
pip install duckdb-engine             # Integração com SQLAlchemy

jupyter notebook  # Iniciar o servidor local p/ criar notebook
```

```
# Usar o JupySQL p/ salvar um CSV com as médias de todos os
# campos, separadas por tipo de iris (https://t.ly/gdUfb).

# Cria a conexão, como no slide anterior
import duckdb; conn = duckdb.connect(":memory:")

# "Mágicas" do JupySQL (apenas para notebook)
%load_ext sql
%sql conn --alias duckdb  # Ou o apelido que você quiser
%sql SELECT 2 ** 5  -- Exemplo (%%sql para bloco)
```

Opcional (DuckDB Shell local): instalar o "*Command line*" do https://duckdb.org/docs/installation ou https://repology.org/project/duckdb/versions

Integrações com o Python

```python
# DuckDB <-> Array do Numpy
import numpy as np
conn.sql("SELECT range(5, 17, 3)").fetchnumpy()
npdata = np.arange(35).reshape(5, -1)
conn.sql("FROM npdata")  # Pela variável local

# DuckDB <-> Pandas
import pandas as pd
conn.sql("SELECT * EXCLUDE column2 FROM npdata").df()
tbl = pd.DataFrame([{"a": a, "b": a ** 2} for a in range(5)])
conn.sql("FROM tbl")

# E se a variável não for local?
conn.register("dataset", tbl)  # Também há unregister
conn.sql("FROM dataset")

# DuckDB <-> Polars (exige pyarrow instalado)
import polars as pl  # COLUMNS aceita filtro por regex!
result = conn.sql("""
SELECT COLUMNS('.*[134]') FROM npdata
""").pl()
col3mod = result.select(pl.col("column3") * pl.col("column1"))
conn.sql("FROM col3mod")
```

Conectando ao PostgreSQL

```
docker run --rm -d \
  -p 5432:5432 \
  -e POSTGRES_USER=ducky \
  -e POSTGRES_PASSWORD=sanca \
  -e POSTGRES_DB=pyse \
  --name pgdb \
  postgres:16
```

```
conn.sql("""
ATTACH
'dbname=pyse user=ducky password=sanca host=127.0.0.1'
AS remotedb (TYPE POSTGRES)
""")
conn.sql("SHOW DATABASES")

conn.sql("CREATE TABLE remotedb.single (value JSON)")
conn.sql("""
INSERT INTO remotedb.single
SELECT DISTINCT {'type': name}::JSON
FROM memory.iris
""")

conn.sql("""  -- Leitura com o próprio PostgreSQL
FROM postgres_query('remotedb',
  'SELECT * FROM single'
)
""")
```

FIM!

https://github.com/
danilobellini/
slides-latex

Para rodar os testes dos slides:

```
python3.8 -m doctest slides.tex
```

Referências:

- https://duckdb.org/docs/
- https://duckdb.org/2024/06/03/announcing-duckdb-100.html
- https://duckdb.org/why_duckdb