Clase 034 — Parquet, Arrow, PyArrow, DuckDB

Parte: 0 — Prerrequisitos · Fuente: docs Apache Arrow + DuckDB + Wes McKinney blogs.

Duración estimada: 75 min.

Clase 034 — Parquet, Arrow, PyArrow, DuckDB

> *Parte: 0 — Prerrequisitos · Fuente: docs Apache Arrow + DuckDB + Wes McKinney blogs. Duración estimada: 75 min.*

Objetivo

Conocer el stack columnar moderno que reemplaza al CSV para datos serios: Parquet (formato en disco), Arrow (formato in-memory), PyArrow (la implementación Python), y DuckDB (SQL embebido sobre Parquet/Arrow). Saber por qué el ecosistema entero (Polars, pandas 2.x, Spark, BigQuery, DataFusion) convergió a este stack.

Resultados de aprendizaje

Al finalizar, el estudiante podrá:

- Leer y escribir Parquet con pandas, polars y pyarrow.
- Aplicar column pruning (leer solo columnas necesarias) y predicate pushdown (leer solo filas necesarias).
- Manejar particionado por columna (year=2024/month=03/) para queries eficientes.
- Hacer queries SQL con DuckDB directamente sobre Parquet sin cargarlo a RAM.
- Reconocer ventajas de Arrow: zero-copy entre librerías (Polars ↔ pandas ↔ Spark).

Temas

- CSV: limitaciones (sin tipos, fila por fila, sin compresión).
- Parquet: columnar, comprimido (snappy/zstd), tipos preservados, metadata por chunk.
- Arrow: formato in-memory zero-copy.
- PyArrow: API Python para ambos.
- DuckDB: "SQLite para analytics", consulta Parquet directo.
- Particionado tipo Hive.

Definiciones y características

- Parquet: formato columnar en disco. Standard de facto para data engineering.
- Arrow: formato columnar in-memory standardizado. Zero-copy entre lenguajes.
- PyArrow: implementación Python de Arrow (pa.Table, pa.parquet.read_table).
- Column pruning: leer solo las columnas requeridas → ahorra I/O y RAM.
- Predicate pushdown: aplicar filtros en el read level → no carga lo que descartás.
- DuckDB: in-process SQL OLAP DB. pip install duckdb y ya.
- Hive partitioning: directorios clave=valor/; el lector deduce particiones.

Dataset / recursos

- NYC Taxi Parquet (Cloudfront público).
- Librerías: pyarrow, duckdb, polars, pandas.

Ejercicios

1. CSV → Parquet: leer un CSV grande con pandas, escribir Parquet con df.to_parquet('file.parquet', compression='zstd'). Comparar tamaño en disco (típicamente 3-10× menor).
2. Column pruning: leer SOLO 2 columnas de un Parquet de 50 columnas con pq.read_table('f.parquet', columns=['a', 'b']). Comparar tiempo vs leer todo.
3. DuckDB sobre Parquet: duckdb.sql("SELECT date, AVG(amount) FROM 'taxi/*.parquet' WHERE amount > 10 GROUP BY date").df(). Sin cargar nada explícitamente.
4. Particionado: escribir df.to_parquet('out/', partition_cols=['year', 'month']). Inspeccionar estructura de directorios.
5. Arrow zero-copy: arrow_table = polars_df.to_arrow(); pandas_df = arrow_table.to_pandas(). Verificar que es rápido (no copia memoria).

Homework verificable

Sobre NYC Taxi (varios meses Parquet, ~5 GB total):

1. Query con DuckDB: total tip_amount por mes para passenger_count >= 2. Sin descargar todos los archivos a memoria.
2. Reportar tiempo y RAM peak (psutil o resource).
3. Comparar contra pandas leyendo todo y filtrando.

Criterio de aceptación: DuckDB completa el query usando < 2 GB RAM mientras pandas explota (OOM o muy lento).

Errores comunes

Síntoma / mensaje	Causa y cómo arreglar
Parquet sin compresión → archivos enormes	Fix: compression='zstd' (default moderno)
read_parquet carga todo aunque solo necesi	Sin column pruning. Fix: columns=['a', 'b'
DuckDB no encuentra archivos	Path glob mal. Fix: 'data/.parquet', 'data
Particionado no detectado	Estructura no Hive. Fix: directorios key=v
Mixing PyArrow Table con pandas DataFrame	Confusión. Fix: decidir un formato; conver

Preguntas frecuentes

¿Parquet o CSV en 2026?

Parquet, salvo para interchange manual con herramientas no técnicas (Excel). 3-10× más chico, 10-100× más rápido de leer, tipos preservados.

¿DuckDB o Spark?

DuckDB para single-machine, datasets < TB. Spark para cluster, > TB. La frontera se mueve: DuckDB maneja hasta cientos de GB cómodamente.

¿Arrow es solo formato o también compute?

Ambos. Arrow Compute (pyarrow.compute) tiene operaciones vectorizadas. Polars usa el motor de Arrow + propio (rust).

¿Por qué zstd?

Compresión mejor que snappy a costo similar de CPU. Es el default en formats modernos (Polars write_parquet default).

¿pandas 2.x soporta Arrow nativo?

Sí — pd.read_csv(..., dtype_backend='pyarrow') usa Arrow types internamente. Más rápido, menos RAM.

Referencias

- Apache Arrow project
- Apache Parquet docs
- DuckDB docs
- McKinney (2017 blog), Apache Arrow and the "10 Things I Hate About pandas".

Siguiente clase

Clase 035 — Matplotlib: anatomía figura/axes

Apéndice: notebook (primer bloque)

Stack columnar moderno: generamos un dataset sintético, lo escribimos en CSV vs Parquet (varias compresiones), inspeccionamos metadata, y consultamos con DuckDB sin cargar a RAM.

```
import time, tempfile, shutil
from pathlib import Path
import numpy as np
import pandas as pd

try:
    import pyarrow as pa
    import pyarrow.parquet as pq
    import duckdb
except ImportError as e:
    raise ImportError('Instalá: pip install pyarrow duckdb pandas') from e

print(f'pyarrow {pa.__version__} | duckdb {duckdb.__version__} | pandas {pd.__version__}')
```

Archivos complementarios

- notebook.ipynb