Clase 214 — Formatos columnares: Parquet, Avro

Parte: 5 — Ingeniería de Datos · Fuente: docs Parquet 2.x + Avro spec + Reis & Housley cap. 6. Duración estimada: 70 min.

Clase 214 — Formatos columnares: Parquet, Avro

> *Parte: 5 — Ingeniería de Datos · Fuente: docs Parquet 2.x + Avro spec + Reis & Housley cap. 6. Duración estimada: 70 min.*

Objetivo

Elegir formato de almacenamiento según el patrón de lectura: Parquet (columnar, OLAP, queries analíticas), Avro (row-based, OLTP/streaming, schema evolution), ORC (columnar, Hive ecosystem). Entender por qué Parquet es 5-100× más rápido que CSV para queries analíticas, y por qué Avro domina en Kafka.

Resultados de aprendizaje

Al finalizar, el estudiante podrá:

- Convertir CSV → Parquet con pandas.to_parquet / polars.write_parquet / pyarrow.
- Diferenciar row-based (CSV, JSON, Avro) de columnar (Parquet, ORC) y elegir según query pattern.
- Aprovechar column pruning + predicate pushdown + row group pruning de Parquet (Polars/DuckDB/Spark lo hacen automático).
- Aplicar compresión (snappy default, zstd mejor ratio, gzip mejor compat) y entender trade-off CPU vs tamaño.
- Definir un schema Avro y usarlo en Kafka con Schema Registry (concept).

Temas

#	Tema	Por qué importa
1	Row vs columnar	OLTP vs OLAP en el storage.
2	Parquet anatomy: file > row group > column	Lo que permite predicate pushdown.
3	Compresión: snappy, zstd, gzip, lz4	Trade-off CPU vs tamaño.
4	Dictionary encoding, RLE	Por qué Parquet con strings repetidos pesa
5	Avro: schema-first, row-based, compact bin	Por qué domina en Kafka.
6	Schema evolution: forward/backward/full co	Cuándo se rompe; cómo manejarlo.

Definiciones y características

- Row-based (CSV, JSON, Avro): cada fila es un bloque contiguo. Eficiente para SELECT * de pocas filas; ineficiente para agregados sobre 1 columna de 1M filas.
- Columnar (Parquet, ORC): cada columna es un bloque contiguo. Eficiente para SELECT col1, col2 FROM tbl WHERE col1 > X — lee solo col1 y col2.
- Row group (Parquet): conjunto de filas (default ~128 MB). Stats (min/max/null_count) por columna por row group — habilitan predicate pushdown sin abrir el row group.
- Column chunk: la parte de una columna dentro de un row group. Físicamente contigua → vectorizable.
- Page: subdivisión de column chunk (~1 MB). Unidad de compresión + read.
- Predicate pushdown: si filtrás WHERE col > 100 y el row group tiene max(col)=50, se skipea sin leer.
- Dictionary encoding: si una columna tiene ≤2^16 valores distintos, Parquet guarda un diccionario +

índices. Strings repetidos: huge win.
- RLE (Run-Length Encoding): si los valores se repiten (AAAA BBBB), guarda (A,4)(B,4). Bueno para columnas con poca varianza.
- Avro: formato row-based binario con schema embedded o en Schema Registry. Diseñado para streaming + schema evolution.
- Schema evolution: agregar/quitar campos sin romper consumers viejos. Avro lo hace bien (default values, alias). Parquet también soporta pero limitado.

Dataset / recursos

- Dataset: NYC Taxi (CSV ~2 GB/mes) — convertir a Parquet (~150 MB/mes).
- Librerías: pyarrow>=15, polars, duckdb, fastavro (Avro).

Ejercicios

1. CSV → Parquet: descargá NYC Taxi 1 mes en CSV. Cargá con pandas, escribí Parquet snappy. Comparar tamaños (CSV vs Parquet) y tiempo de query (COUNT WHERE borough='Manhattan').
2. Compresión benchmark: mismo dataset, escribir con snappy, zstd, gzip, lz4. Reportar tamaño + tiempo de lectura. (Hint: zstd suele ganar en ratio; snappy en speed).
3. Predicate pushdown: en DuckDB, EXPLAIN ANALYZE SELECT FROM parquet WHERE pickup_date='2024-01-15'. Compará "rows read" vs SELECT FROM parquet sin WHERE.
4. Row groups y stats: con pyarrow.parquet.ParquetFile(path).metadata, inspeccioná min/max/null_count por columna por row group.
5. Avro schema + roundtrip: definí schema Avro para evento de click. Serializá 1000 eventos con fastavro, deserializá. Compará tamaño con JSON puro.

Homework verificable

Notebook + reporte:

1. Benchmark sobre 1 año de NYC Taxi:
- CSV (raw), Parquet snappy, Parquet zstd, Avro snappy, JSON gzipped.
- Reportar para cada uno: tamaño en disco, tiempo de query COUNT *, tiempo de SELECT col WHERE filter.
1. Demonstración de schema evolution: tabla Avro con schema v1 (5 campos), agregar campo opcional → v2. Consumer viejo (con v1) lee data de v2 sin romperse.
2. Justificación de elección: para 3 casos de uso (analytics warehouse, Kafka stream, archive a largo plazo), recomendar formato y compresión.
3. Cálculo de costos: pasar de CSV a Parquet en S3, asumir 100 GB/mes generados. ¿$ ahorrados en S3 + transferencia?

Criterio de aceptación: el alumno demuestra con números (no opinión) por qué Parquet es ~5-50× más rápido para queries analíticas, y produce esquema Avro funcional con evolución backward-compatible.

Errores comunes

Síntoma / mensaje	Causa y cómo arreglar
Parquet más grande que CSV	(1) Schema con muchas columnas vacías. (2)

Lectura Parquet lenta — más lenta que CSV	Estás haciendo SELECT * (no aprovecha colu
ArrowInvalid: Schema mismatch al concatena	Dos parquets con types ligeramente distint
Avro consumer rompe al evolucionar schema	Cambio no-compatible (renombrar campo sin
partitionBy('user_id') en Parquet crea 1M	Particionado high-cardinality. Fix: partic
zstd no se lee en sistema legacy	Algunos viejos solo soportan snappy/gzip.

Preguntas frecuentes

¿Parquet o ORC?

Parquet: dominante fuera del ecosistema Hadoop puro. ORC: nativo del ecosistema Hive/Hortonworks (algo declining). Para 2026 greenfield: Parquet sin pensar.

¿Parquet o Delta Lake / Iceberg / Hudi?

Parquet es solo formato de archivo. Delta Lake, Iceberg, Hudi son "table formats" sobre Parquet que agregan: ACID transactions, time travel, schema evolution rica, MERGE/UPSERT. Para data lakes serios: una de estos sobre Parquet. Para análisis ad-hoc: Parquet pelado alcanza.

¿CSV alguna vez tiene sentido?

(1) Interoperabilidad humana (Excel, miras con less). (2) Tamaño chico (<1 MB) — no vale la pena la complejidad. (3) Output de un proceso para humano. Para todo lo demás: Parquet o JSON.

¿Avro o Protobuf en Kafka?

Ambos serializan binario con schema. Avro: schema embeddable o Registry, mejor evolution semantics. Protobuf: más rápido, mejor ecosystem gRPC. Confluent Kafka favorece Avro (Schema Registry built-in); gRPC users favorecen Protobuf.

¿Cuánto comprime zstd vs snappy?

Aprox: snappy reduce 50-60%, zstd reduce 65-75%. Zstd CPU cost ~2-3× snappy en compresión, ~1× en decompresión. Para almacenamiento long-term: zstd. Para alta throughput: snappy.

¿Cómo manejo Parquet con timezone-aware datetimes?

Parquet 2.x soporta timestamp(unit=us, tz='UTC'). Polars y PyArrow lo respetan. Cuidado: pandas <2.0 a veces hace conversión silenciosa. Estandarizar en UTC siempre.

Referencias

- Parquet docs — file format spec.
- Avro spec — schema language.
- PyArrow Parquet API.
- Apache Iceberg — table format moderno.
- CSV is dead, long live Parquet (mejor para 2026).

Siguiente clase

Clase 215 — Modelado dimensional (star/snowflake schemas)

Apéndice: notebook (primer bloque)

Requiere: pip install pyarrow polars duckdb fastavro.

```
import pyarrow as pa, pyarrow.parquet as pq, polars as pl, pandas as pd, numpy as np, time, json
from pathlib import Path
import tempfile, shutil

WORK = Path(tempfile.gettempdir()) / 'parquet_bench'
if WORK.exists(): shutil.rmtree(WORK)
WORK.mkdir()

rng = np.random.default_rng(42)
N = 2_000_000
df = pl.DataFrame({
    'zone_id': rng.integers(0, 100, N),
    'fare':    rng.uniform(5, 100, N),
    'tip':     rng.uniform(0, 20, N),
    'pickup_date': pl.date(2024, 1, 1) + pl.duration(days=pl.Series(rng.integers(0, 90, N))),
    'borough': rng.choice(['Manhattan', 'Brooklyn', 'Queens', 'Bronx'], N),
    'note':    rng.choice(['short ride', 'long ride', 'airport', 'rush hour', None], N),
})
print(f'dataset: {N:,} rows, {len(df.columns)} cols')
```

Archivos complementarios

- notebook.ipynb