Clase 212 — Data warehouses: BigQuery, Snowflake, DuckDB

Parte: 5 — Ingeniería de Datos · Fuente: Reis & Housley Fundamentals of Data Engineering (O'Reilly, 2022) cap. 6 + 9 + docs oficiales. Duración estimada: 80 min.

Clase 212 — Data warehouses: BigQuery, Snowflake, DuckDB

> *Parte: 5 — Ingeniería de Datos · Fuente: Reis & Housley Fundamentals of Data Engineering (O'Reilly, 2022) cap. 6 + 9 + docs oficiales. Duración estimada: 80 min.*

Objetivo

Consultar y operar 3 data warehouses modernos desde Python: BigQuery (GCP, serverless, separación compute/storage), Snowflake (multi-cloud, virtual warehouses, time travel), DuckDB (embedded, OLAP local, no requiere server). Decidir cuál usar según escala, presupuesto y latencia.

Resultados de aprendizaje

Al finalizar, el estudiante podrá:

- Conectar a BigQuery (google-cloud-bigquery), Snowflake (snowflake-connector-python), DuckDB (duckdb) y ejecutar queries.
- Diseñar tablas con particionado (BigQuery: PARTITION BY date_field) y clustering (BigQuery, Snowflake) para reducir costos y latencia.
- Usar COPY INTO (Snowflake) y LOAD DATA (BigQuery) para bulk ingest desde S3/GCS.
- Aprovechar time travel (SELECT ... AT(TIMESTAMP => ...) Snowflake) para auditar / recuperar.
- Decidir DW: BigQuery (serverless, GCP-first), Snowflake (multi-cloud, separation, sharing), DuckDB (local/embedded), Redshift (legacy AWS).

Temas

#	Tema	Por qué importa
1	Compute/storage separation	Por qué los DW modernos son baratos: solo
2	Particionado vs clustering	Reducir bytes leídos.
3	BigQuery: SQL standard + UDF + ML	El más simple para empezar.
4	Snowflake: virtual warehouses, time travel	Features únicos.
5	DuckDB: el DW que entra en pip install	Análisis local, ETL liviano, embed en app.
6	Costo: cómo NO gastar miles	SELECT * sin partition filter = bankruptcy

Definiciones y características

- Data warehouse: DB OLAP optimizada para queries analíticas (agregados sobre millones/billones de filas). Diferente de OLTP (Postgres, MySQL) optimizada para transacciones.
- Compute/storage separation: storage en blob (S3/GCS/Azure) — barato y elástico. Compute (virtual warehouse / slot) se enciende para queries. Pagás storage barato + compute por uso.
- Particionado: la tabla está dividida por una columna (típicamente date). Query con WHERE date='2024-01-15' lee solo esa partición → menos bytes → menos $.
- Clustering (BigQuery) / Cluster keys (Snowflake): orden físico dentro de cada partición según N columnas. Mejora queries con WHERE col_clustered.
- Slot / Virtual Warehouse: unidad de compute que paralela tu query. BigQuery: slots compartidos o

reservados. Snowflake: tamaño de VW (XS/S/M/L/XL...) en $/hora.
- Time travel (Snowflake): consultar el estado pasado de una tabla (SELECT ... AT(TIMESTAMP => '2026-06-15 00:00:00')). Retención 1-90 días según tier.
- Data sharing (Snowflake): compartir tablas con otra cuenta Snowflake sin copiar data (live read).
- DuckDB: DB OLAP embedded (como SQLite pero columnar + vectorizada). No requiere server. Lee/escribe Parquet/CSV/Arrow nativo. Ideal para dev local + ETL liviano + análisis ad-hoc.

Dataset / recursos

- BigQuery: bigquery-public-data.new_york_taxi_trips.tlc_yellow_trips_2018 (GB-scale, gratis para query con free tier).
- DuckDB: local con parquets de NYC Taxi.
- Snowflake: trial 30 días con $400 de crédito.
- Librerías: duckdb>=1.0, google-cloud-bigquery, snowflake-connector-python.

Ejercicios

1. DuckDB local: con = duckdb.connect("warehouse.duckdb"). Cargá un parquet con CREATE TABLE trips AS SELECT FROM 'trips.parquet'. Hacé SELECT COUNT(), AVG(fare) FROM trips. Compará tiempo vs Polars (Clase 211).
2. BigQuery query: from google.cloud import bigquery; client = bigquery.Client(project="..."). client.query("SELECT borough, COUNT(*) FROM bigquery-public-data.new_york_taxi_trips.tlc_yellow_trips_2018 GROUP BY borough"). Crítico: usar LIMIT en exploración, no escanear 100 GB sin querer.
3. Particionado en BQ: crear tabla mi_proyecto.dataset.trips_partitioned con PARTITION BY DATE(pickup_datetime). Query con WHERE DATE(pickup_datetime) = '2018-01-15' → mostrá "bytes processed" con/sin partition filter.
4. Snowflake time travel: CREATE TABLE x AS SELECT .... Insertá data. DELETE FROM x WHERE .... SELECT * FROM x AT(OFFSET => -60) (60s atrás) — los datos vuelven.
5. DuckDB queryando S3 directo: con.execute("SELECT FROM 's3://bucket/path/.parquet' LIMIT 10") sin descargar nada — DuckDB lee remoto con HTTPFS.

Homework verificable

Proyecto con:

1. Pipeline que extrae datos de una API → carga a un DW (elegir 1: BQ free tier, DuckDB, Snowflake trial) → corre 5 queries analíticas.
2. Comparativa de costo: misma query escaneando (a) tabla sin particionar (5 GB), (b) tabla particionada con filter (50 MB). Reportar $ estimado.
3. Tabla con clustering / cluster keys sobre 2 columnas que mejoran una query frecuente. Mostrar mejora de performance.
4. Setup de service account con permisos mínimos (BQ Data Viewer, no Admin).
5. README explicando cuál DW elegirías para 3 escenarios: startup 5 dev, empresa multi-cloud, análisis personal local.

Criterio de aceptación: el alumno demuestra entender el modelo de costo (bytes scanned × $/TB) y propone arquitectura razonable para cada escenario.

Errores comunes

Síntoma / mensaje	Causa y cómo arreglar
Factura BigQuery $500 inesperados	Alguien hizo SELECT sin partition filter
Query Snowflake lenta — escalar el warehou	Posible, pero antes: revisar el plan (EXPL
DuckDB OOM en query agregada	Default memory limit (80% RAM). Fix: SET m
Permission denied BigQuery	El service account no tiene rol. Fix: en G
Snowflake Statement timeout	Tu VW es chico. Fix: subir tamaño temporal
JOIN lento entre dos tablas grandes	Falta cluster key compartida. Fix: cluster
Particionado por columna high-cardinality	BQ permite max 4000 particiones. Fix: part

Preguntas frecuentes

¿BQ vs Snowflake vs Redshift vs Databricks SQL?

- BigQuery: serverless puro, ideal en GCP, modelo "pay per query".
- Snowflake: separación compute/storage explícita, time travel, data sharing nativo. Multi-cloud.
- Redshift: el viejo (2012). AWS-native. RA3 instances son competitivos pero menos elegantes.
- Databricks SQL (sobre lakehouse): si ya tenés Spark en Databricks, integrado.

Para empezar: BigQuery (GCP) o Snowflake (multi-cloud).

¿DuckDB como warehouse de producción?

Para una sola máquina y team chico: sí (hasta cientos de GB). Para múltiples usuarios escribiendo concurrente y escala TB+: no — usá Snowflake/BQ. MotherDuck ofrece DuckDB managed cloud con escalado.

¿Cómo evito quemar $500 sin querer en BQ?

(1) Project Settings → Quotas → Query usage limit per day. (2) Linter pre-commit que rechaza SELECT * sin partition. (3) Crear materialized views para queries repetidas — pagás 1 vez. (4) dry_run=True antes de ejecutar.

¿DELETE en un DW columnar es caro?

Sí. Las tablas columnares no están optimizadas para DELETE individual. Patrón: marcar como deleted_at (soft delete) o reescribir partition (INSERT OVERWRITE). Snowflake/BQ lo manejan razonablemente; Redshift sufre.

¿Cómo cargo 1 TB a un warehouse?

(1) Subí archivos a blob storage (S3/GCS). (2) Usá comando bulk del DW: LOAD DATA (BQ), COPY INTO (Snowflake), COPY (Redshift). NUNCA INSERT ... VALUES por row.

¿Costo storage es relevante?

En 2026: ~$23/TB/mes en BQ/Snowflake. Para 10 TB = $230/mes. Para 1 PB: $23K/mes — empieza a importar. Para datasets enormes: lifecycle policies (mover a Glacier/Coldline después de N días).

Referencias

- Reis & Housley Fundamentals of Data Engineering (O'Reilly, 2022) cap. 6 (storage) y 9 (queries).
- BigQuery docs — empezar por Quickstarts.
- Snowflake docs — Quickstarts + Cost optimization.
- DuckDB docs + MotherDuck (DuckDB managed).
- Designing Data-Intensive Applications (Kleppmann, 2017) — fundamentos OLAP.

Siguiente clase

Clase 213 — Streaming intro: Kafka, Kinesis

Apéndice: notebook (primer bloque)

DuckDB se corre local sin credenciales. BQ y Snowflake requieren cuenta — los mostramos como código de referencia.

```
import duckdb, pandas as pd, tempfile, time
from pathlib import Path
WORK = Path(tempfile.gettempdir()) / 'dw_demo'
WORK.mkdir(exist_ok=True)
print('DuckDB version:', duckdb.__version__)
```

Archivos complementarios

- notebook.ipynb