High-Volume For-Hire Services

Análisis de rendimiento entre DuckDB y PostgreSQL

Bases de Datos Avanzadas

Dr. Luis Gustavo Esquivel Quirós

Colaboradores

Marco Piedra Venegas - A64397

Alejandro Jiménez Corea - B84032

Daniel Artavia - B70771

Gabriel Molina Bulgarelli - C14826

Introducción

Descripción del problema

El manejo de grandes volúmenes de datos presenta un reto constante en la gestión de bases de datos. Tradicionalmente, motores como PostgreSQL han sido utilizados para este propósito.

Sin embargo, el crecimiento exponencial de datos y la demanda de análisis más complejos y rápidos requieren alternativas más eficientes y de mejor rendimiento, dando lugar a motores de bases de datos como DuckDB.

Introducción

Objetivo general

- Evaluar la eficacia y el rendimiento de DuckDB, un motor de base de datos embebido optimizado para análisis de datos, en comparación con PostgreSQL, un motor de base de datos relacional tradicional, para determinar su viabilidad en el análisis de grandes volúmenes de datos.

Para determinar la eficacia de estos motores, se plantea medir el tiempo de ejecución de dos consultas específicas diseñadas para análisis exploratorio de datos grandes y complejos, enfocándose en funciones de *Agregación, Filtrado* y *Combinación de datos.*

Introducción

Descripción de los datos

Viajes en plataformas de transporte en Nueva York brindados por la Comisión de Taxis y Limusinas. Son ***"High Volume For-Hire Vehicle",*** que incluye Uber, Lyft, Juno (hasta noviembre de 2019) y Via, con más de 10,000 viajes diarios.

- Abarcan desde ***febrero*** de ***2019*** hasta ***marzo*** de ***2024***.
- Incluye ***62 archivos Parquet***, totalizando ***25.2 GB***.

Los datos incluyen el código del proveedor, fechas y horas de inicio y fin del viaje, longitud en millas, tiempo en segundos, tarifa base, monto recibido, propina e impuestos.

Modelado de los datos

Uso de diagramas conceptuales

No es común en bases de datos NoSQL utilizar modelos conceptuales debido a su naturaleza ***schemaless***, pero estos son esenciales para visualizar y entender los datos antes de su implementación física.

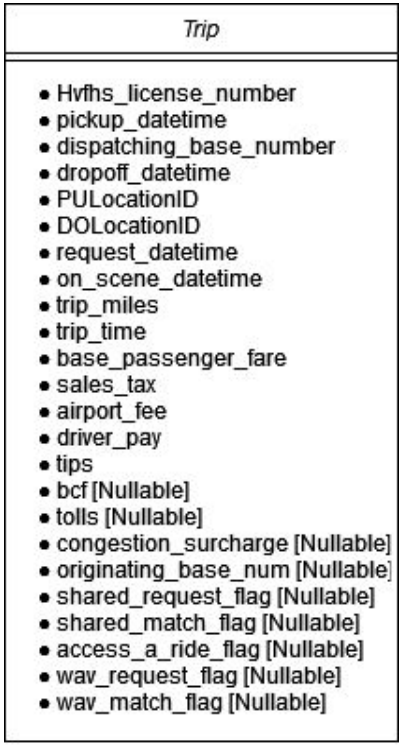
"Los modelos conceptuales ayudan a identificar las relaciones y dependencias entre los datos, lo que es crucial para diseñar consultas eficientes y estructuras de almacenamiento que soporten grandes volúmenes de datos y altas tasas de acceso." - Li y Manoharan

El modelo de datos de HVFHS es columnar representado mediante el procesamiento de archivos en formato Parquet, clasificado como un modelo NoSQL.

Modelado de los datos

Diagrama conceptual de HVFHS en Nueva York

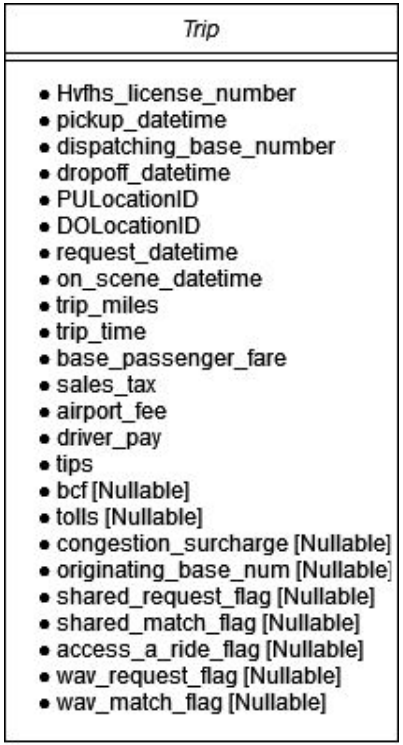
- **Hvfhs_license_num (String):** Número de licencia TLC de la base o negocio HVFHS, identificando de manera única a cada base o negocio que despacha los viajes.
- **dispatching_base_num (String):** Número de licencia de base TLC de la base que despachó el viaje, crucial para identificar la base que organizó y envió el viaje.
- **pickup_datetime (Timestamp):** Fecha y hora de recogida del viaje.
- **dropoff_datetime (Timestamp):** Fecha y hora de finalización del viaje.
- **PULocationID (Integer):** Zona de taxi TLC donde comenzó el viaje, utilizando un identificador numérico para las zonas geográficas definidas por la TLC.



Modelado de los datos

Diagrama conceptual de HVFHS en Nueva York

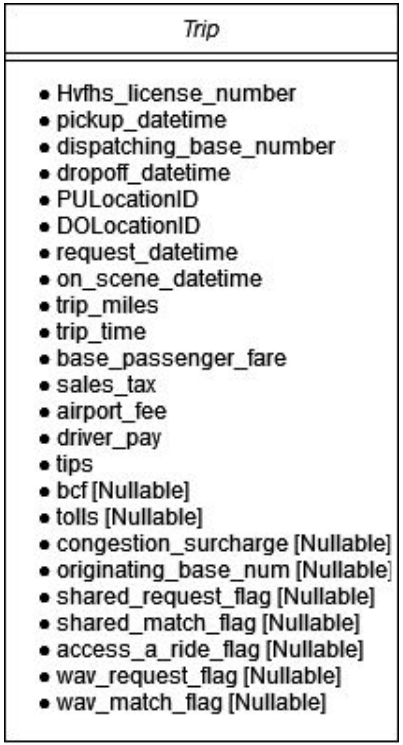
- **DOLocationID (Integer):** Zona de taxi TLC donde terminó el viaje, similar a PULocationID, identificando la zona de destino.
- **request_datetime (Timestamp):** Fecha y hora en que el pasajero solicitó ser recogido, proporcionando la marca temporal de cuándo se hizo la solicitud.
- **on_scene_datetime (Timestamp):** Fecha y hora en que el conductor llegó al lugar de recogida, opcional y aplicable solo a vehículos accesibles.
- **trip_miles (Float):** Millas totales del viaje de pasajeros, midiendo la distancia recorrida.



Modelado de los datos

Diagrama conceptual de HVFHS en Nueva York

- **trip_time (Integer):** Tiempo total en segundos del viaje de pasajeros.
- **base_passenger_fare (Float):** Tarifa base del pasajero representando el costo inicial del viaje.
- **sales_tax (Float):** Cantidad total recaudada para el impuesto sobre ventas del estado de Nueva York, incluyendo el impuesto aplicable al viaje.
- **airport_fee (Float):** Tarifa fija de $2.50 en aeropuertos específicos.
- **driver_pay (Float):** Pago total al conductor, excluyendo peajes y propinas, neto de comisiones, recargos o impuestos.
- **tips (Float):** Cantidad total de propinas recibidas del pasajero.

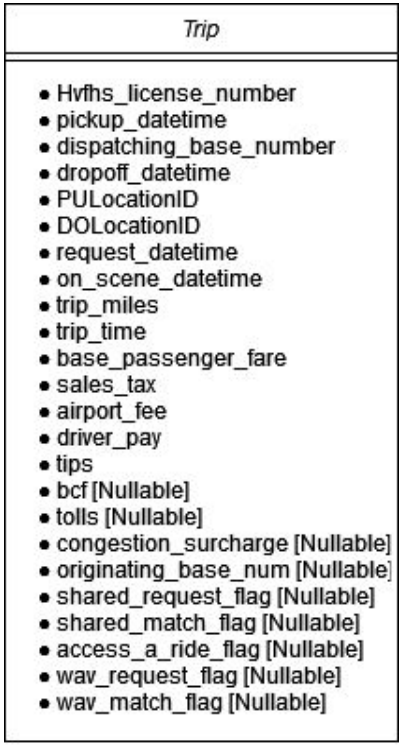


Modelado de los datos

Diagrama conceptual de HVFHS en Nueva York

Atributos opcionales - *[Nullable]*

- **bcf (Float):** Cantidad recaudada para el Black Car Fund.
- **tolls (Float):** Cantidad de todos los peajes pagados en el viaje.
- **congestion_surcharge (Float):** Cantidad total recaudada por el recargo de congestión del estado de Nueva York.
- **originating_base_num (String):** Número de base que recibió la solicitud original del viaje.
- **shared_request_flag (String):** Indica si el pasajero aceptó un viaje compartido (Y/N).

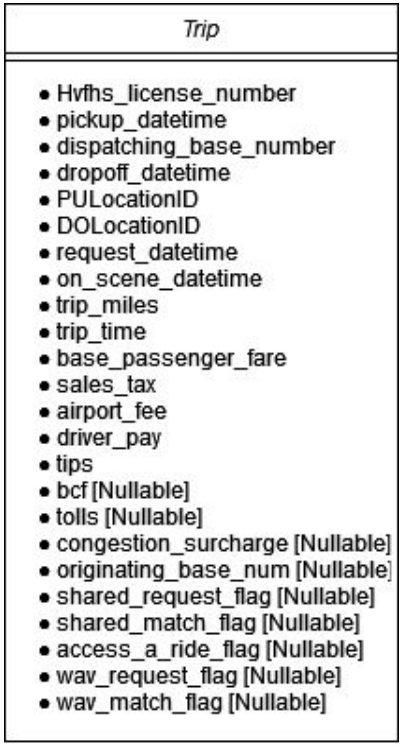


Modelado de los datos

Diagrama conceptual de HVFHS en Nueva York

Atributos opcionales - *[Nullable]*

- **shared_match_flag (String):** Indica si el pasajero compartió el vehículo con otro pasajero (Y/N).
- **access_a_ride_flag (String):** Indica si el viaje fue administrado en nombre de la MTA (Y/N).
- **wav_request_flag (String):** Indica si el pasajero solicitó un vehículo accesible para sillas de ruedas (Y/N).
- **wav_match_flag (String):** Indica si el viaje ocurrió en un vehículo accesible para sillas de ruedas (Y/N).



Escenarios de análisis

Descripción de consultas realizadas

Se compara el rendimiento de DuckDB y PostgreSQL con funciones de agregación estadística mediante dos tipos de consultas denominadas A y B.

- ***Consulta A:*** Utiliza las funciones *regr_slope* y *regr_intercept* para calcular la pendiente y la intersección de la línea de regresión con el eje y.
- ***Consulta B:*** Agrupa y promedia variables como el tiempo y la distancia de viaje, así como las tarifas de pasajeros y conductores, por borough en Nueva York. Esto se logra usando una tabla adicional de la Comisión de Taxis y Limosinas mediante una operación de join.

Escenarios de análisis

Descripción de columnas consultadas

Se habilitaron opciones de temporización en DuckDB y PostgreSQL y se realizaron tres corridas por cada combinación de base de datos y consulta.

Las columnas seleccionadas para este estudio son:

- ***trip_miles***: distancia de un viaje en millas.
- ***trip_time***: tiempo de un viaje en segundos.
- ***base_passenger_fare***: tarifa base pagada por el pasajero.
- ***driver_pay***: tarifa base de ingresos del conductor.
- ***PULocationID***: identificador de zona de la ciudad de Nueva York donde inicia un viaje.

Escenarios de análisis

Descripción de resultados

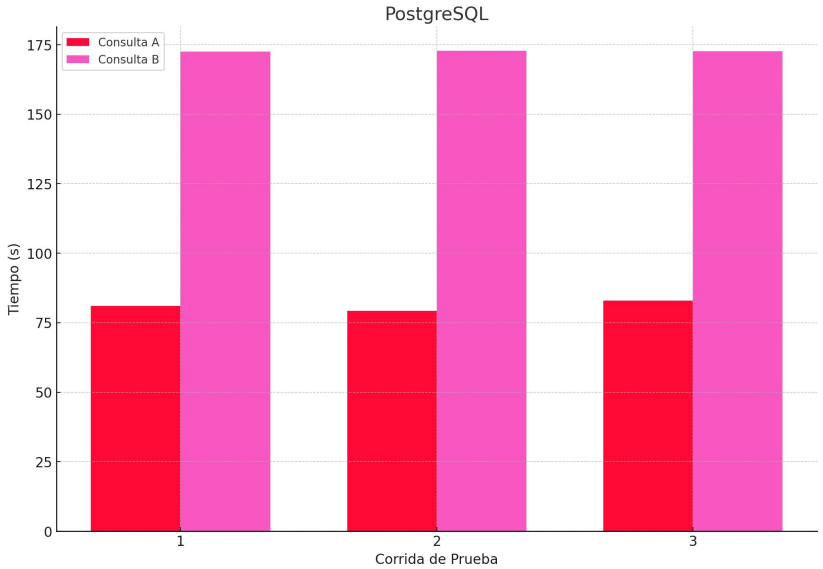
El experimento se llevó a cabo en una máquina:

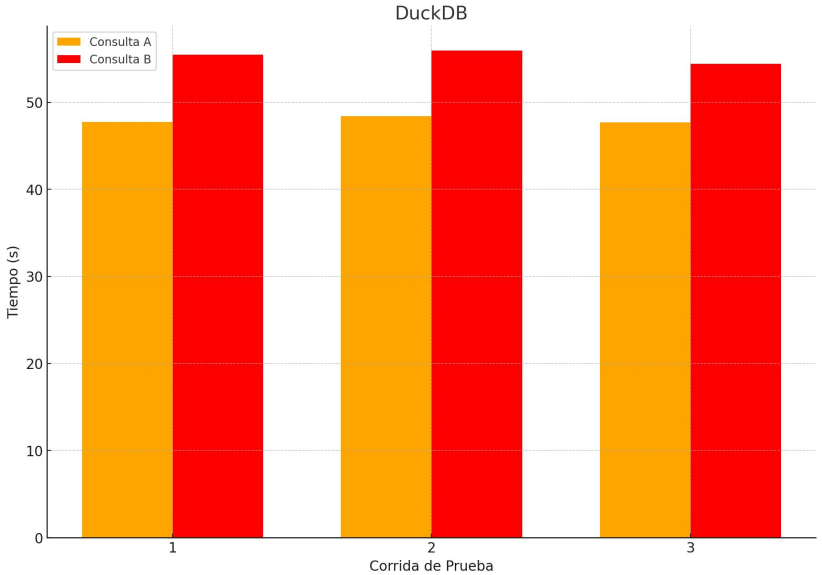
- Dell XPS 13 (modelo 9315)
- Con 12 CPUs Intel Core i5-1230U
- 8 GB de RAM DDR5,
- Utilizando Debian GNU/Linux 12.6 (Bookworm).

Motor de base de datos	Consulta	Prueba	Tiempo (s)	Promedio (s)
DuckDB	A	1	47.717	47.937
		2	48.397	
		3	47.700	
	B	1	55.478	55.271
		2	55.947	
		3	54.390	
PostgreSQL	A	1	81.059	81.109
		2	79.334	
		3	82.933	
	B	1	172.537	172.682
		2	172.816	
		3	172.693	

Escenarios de análisis

Resultados de tiempos de ejecución por prueba



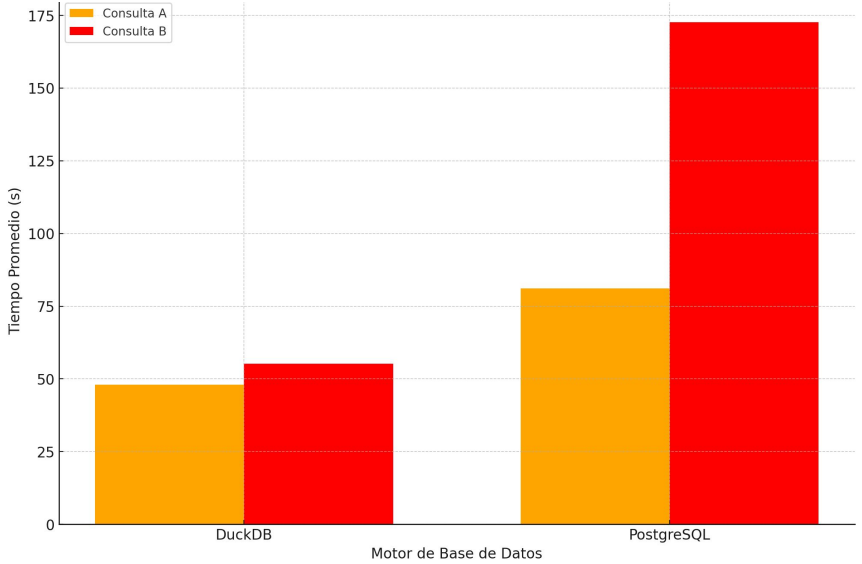


Escenarios de análisis

Descripción de resultados tiempos de ejecución por prueba

DuckDB supera significativamente a PostgreSQL en rendimiento.

- En la consulta A, DuckDB es **1.69 veces** más rápido.
- En la consulta B, DuckDB es **3.12 veces** más rápido.



Escenarios de análisis

Razones de un mejor rendimiento con DuckDB

- **Acceso Directo a Archivos:** DuckDB accede y procesa archivos Parquet directamente, aprovechando su formato columnar y compresión de datos para reducir tiempos de acceso.
- **Eliminación de Pasos Adicionales:** PostgreSQL requiere cargar datos en una tabla, añadiendo tiempo de procesamiento.
- **Vectorización de Consultas:** DuckDB utiliza la vectorización de consultas del motor X100 de MonetDB. Esto permite aplicar optimizaciones, como las operaciones SIMD del procesador, que serían limitadas en el modelo tabular utilizado por PostgreSQL.

Referencias del proyecto

Apache Parquet (2024). Overview. https://parquet.apache.org/docs/overview/

Boncz, P., Zukowski, M., & Nes, N. (2005). MonetDB/X100: Hyper-Pipelining Query Execution. In Proceedings of International conference on verly large data bases (VLDB) 2005. Very Large Data Base Endowment.

Bellon, T. (2019). Gett's Juno ends NYC ride-hailing services, citing regulation. Reuters. https://www.reuters.com/article/us-gett-juno-idUSKBN1XS2JZ/

Taxi & Limousine Commission (2024). New York City - TLC Trip Record Data. https://www.nyc.gov/site/tlc/about/tlc-trip-record-data.page

DuckDB Foundation (2024). Why DuckDB. https://duckdb.org/why_duckdb

West, M. (2011). Developing high quality data models. Elsevier. https://doi.org/10.1016/c2009-0-30508-5

Referencias del proyecto

Li, G., & Manoharan, S. (2013). A performance comparison of SQL and NoSQL databases. IEEE Pacific Rim Conference on Communications, Computers and Signal Processing (PACRIM), 15-19.

The PostgreSQL Global Development Group (2024). About PostgreSQL. https://www.postgresql.org/about/

EnterpriseDB (2024). Download PostgreSQL. https://www.enterprisedb.com/downloads/postgres-postgresql-downloads

SQLite (2024). About SQLite. https://www.sqlite.org/about.html