**Universidad de Costa Rica**

Escuela de Ciencias de la Computación e Informática

**CI-0141**

Bases de Datos Avanzadas



**Proyecto Final**

**Profesor:**

Dr. Luis Gustavo Esquivel Quirós

**Estudiantes:**

Marco Piedra Venegas - A64397

Alejandro Jiménez Corea - B84032

Daniel Artavia - B70771

Gabriel Molina Bulgarelli - C14826

I Semestre

2024

**Introducción**

En este proyecto se realizan tareas básicas de analítica de datos mediante DuckDB (<https://duckdb.org/>), un motor de base de datos con características NoSQL, sobre archivos en formato Parquet (<https://parquet.apache.org/>). Asimismo, se compara rendimiento con respecto a PostgreSQL (<https://www.postgresql.org/>), un motor de base de datos objeto-relacional.

El objetivo general del proyecto consiste en evidenciar la flexibilidad de un acercamiento NoSQL para procesamiento de datos semiestructurados, cuyo volumen supere los recursos de memoria.

**Descripción del problema**

DuckDB se describe como un motor de base de datos para analítica, que puede funcionar dentro de procesos (embedded). De este modo, se puede integrar dentro de otros programas (DuckDB Foundation, 2024).

Parquet es un formato columnar especializado para almacenamiento y acceso rápido de datos. Utiliza compresión de alto rendimiento y esquemas de codificación para grandes volúmenes (Apache Parquet, 2024).

PostgreSQL es un motor de base de datos objeto-relacional de código abierto, que se apega al estándar SQL y permite extensiones, tales como datos de información geográfica. Se originó en 1986 en la Universidad de California en Berkeley (The PostgreSQL Global Development Group, 2024).

**Descripción de datos**

El conjunto de datos por analizar para el proyecto consiste en los viajes en plataformas de transporte, recopilados mensualmente por la Comisión de Taxis y Limosinas de la Ciudad de Nueva York, y disponibles de forma pública (City of New York, 2024). Maneja 62 archivos en formato Parquet, que suman 26 GB. La media de tamaño es 415.7 MB, con una desviación estándar de 97.4 MB. El rango de tiempo es desde febrero de 2019 hasta marzo de 2024 (Anexos 1 y 2).

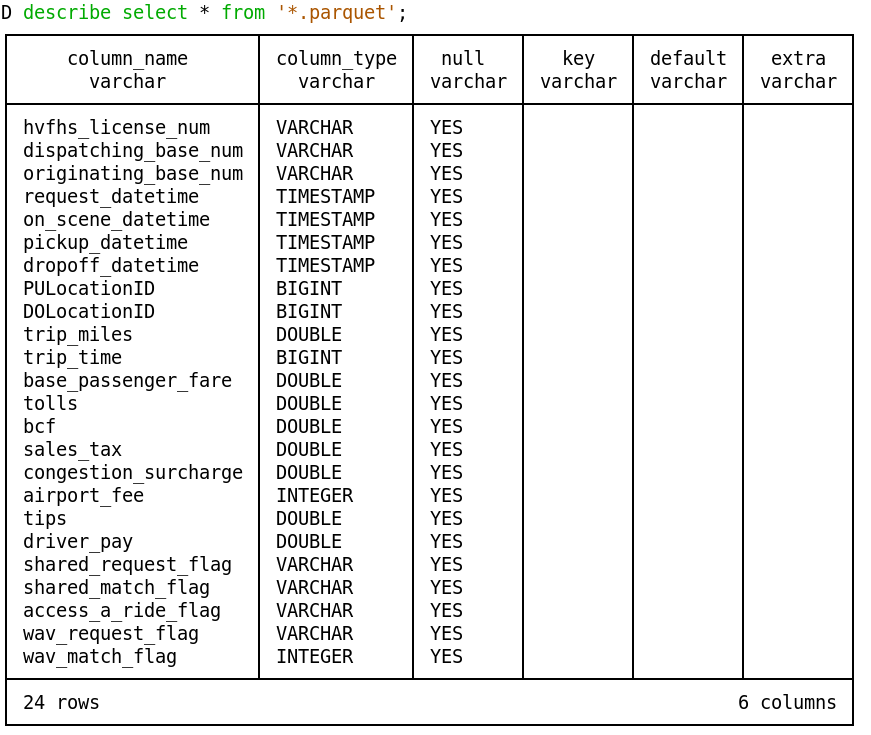
Asimismo, se consideran solo los viajes en la categoría llamada “High Volume For-Hire Vehicle”, que incluyen a las plataformas de transporte que generan más de 10000 viajes diarios en la ciudad de Nueva York. Se incluyen 4 proveedores: Uber, Lyft, Juno, y Via. Los datos de Juno terminan en noviembre de 2019, debido a que en ese mes dejó de operar (Bellon, 2019).

Cada archivo Parquet incluye campos como el código de proveedor, fecha y hora de inicio y fin de viaje, longitud del viaje en millas, tiempo del viaje en segundos, tarifa base para el pasajero, monto recibido por el conductor, propina, e impuestos.

**Modelado de Datos**

El modelo de datos por utilizar es columnar, representado por el procesamiento de archivos en formato Parquet. Se clasifica como un modelo NoSQL.

Se realiza procesamiento de una tabla con 24 columnas, listadas a continuación:



En el análisis de datos solo se utilizan las siguientes columnas:

* trip\_miles: distancia de un viaje en millas
* trip\_time: tiempo de un viaje en segundos
* base\_passenger\_fare: tarifa base de pago, en un viaje, para quien solicita el transporte
* driver\_pay: tarifa base de ingresos, en un viaje, para quien ofrece el transporte
* PULocationID: identificador de la zona de la ciudad de Nueva York donde inicia un viaje

**Escenarios de análisis**

En este proyecto, se realiza la comparación de rendimiento de DuckDB y PostgreSQL en cuanto a las funciones de agregación estadística. En particular, regr\_slope y regr\_intercept para obtener la pendiente de la línea de regresión y su intersección con el eje *y*, respectivamente.

Se desea realizar un análisis comparativo entre dos diferentes modelos de bases de datos utilizando una consulta SQL que aplica funciones de regresión lineal para analizar la relación entre tarifas de pasajeros, pagos a conductores y las características del viaje, como la distancia y el tiempo. Este análisis se basa en datos almacenados en archivos Parquet.

Se aplican estas operaciones para obtener regresión lineal de la ganancia base de conductor (driver\_pay) y cobro base al pasajero (base\_passenger\_fare), con respecto a longitud de viaje (trip\_miles) y tiempo de viaje (trip\_time).

Se seleccionaron estas operaciones para comparación, ya que su ejecución no es trivial en un volumen de datos considerable. Asimismo, se encuentran disponibles en ambos motores de bases de datos y tienen la misma semántica. De este modo, se comparan ejecuciones equivalentes entre ambos motores.

En el experimento, se toman los tiempos de ejecución, al habilitar las opciones correspondientes con .timer on en DuckDB y \timing en PostgreSQL. Se realizan 3 corridas por cada combinación de factores: base de datos (2 niveles: DuckDB y PostgreSQL) y consulta (A y B, listadas en anexos 4 y 5).

El experimento se realiza en una máquina Dell XPS 13 (modelo 9315), con 12 CPUs Intel Core i5-1230U y 8 GB de RAM DDR5, en Debian GNU/Linux 12.6 (Bookworm). Cada ejecución se realiza “en frío”, i.e., se detiene y se vuelve a iniciar el proceso. En el caso de DuckDB, se realiza salida de la línea de comandos (el ejecutable único inicia el motor de base de datos y la interfaz de usuario). En PostgreSQL, se detiene y se inicia el servicio.

**Resultados**

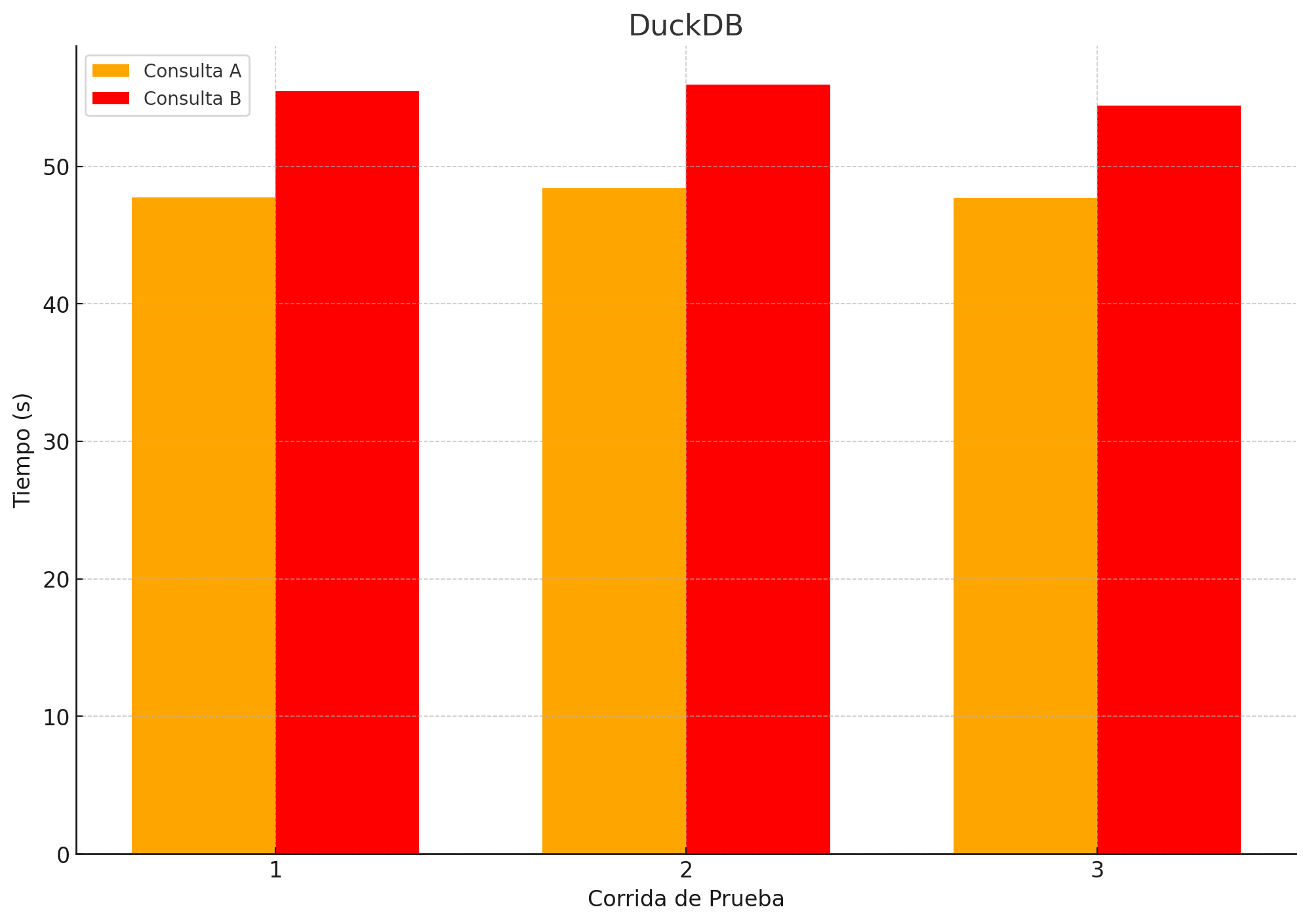
En cuanto a tiempos de ejecución de las consultas (anexos 4 y 5) en los motores de bases de datos se observa que, en promedio, en DuckDB la consulta A tarda 47.937 segundos y la consulta B tarda 55.271 segundos. Mientras tanto, en promedio, en PostgreSQL la consulta A tarda 81.109 segundos y la consulta B tarda 172.682 segundos (Anexo 8 y gráficos).

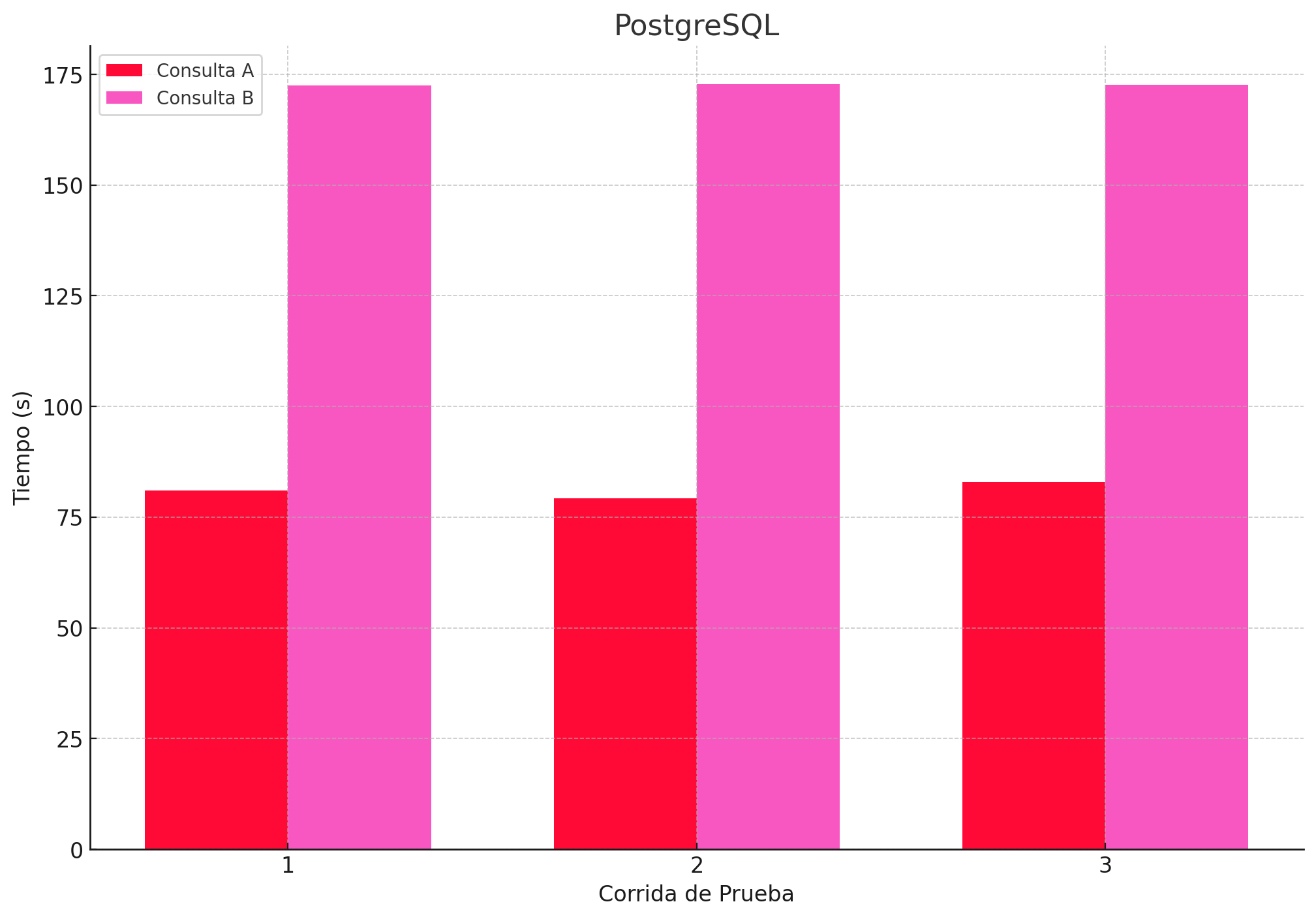
De esta forma, DuckDB presenta una mejora de rendimiento de 1.153 en la consulta A, así como una mejora rendimiento de 2.129 en la consulta B. Cabe destacar que DuckDB accede a los archivos de forma directa, mientras que PostgreSQL requirió cargar datos a una tabla desde los archivos.

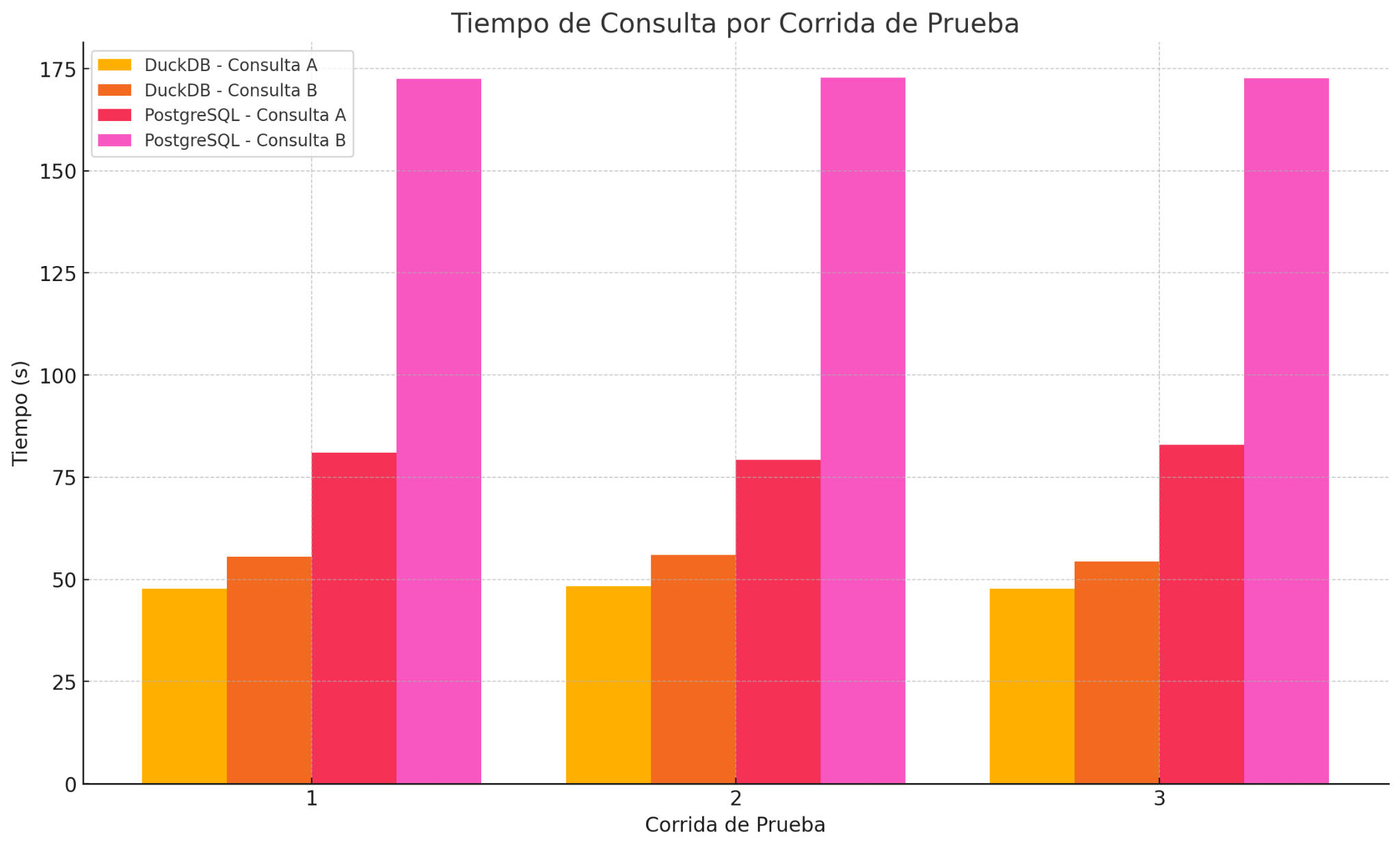
Una interpretación de la mejora de rendimiento con DuckDB, con respecto a PostgreSQL, es el procesamiento nativo de archivos en formato Parquet. El formato columnar y la compresión de datos de Parquet permite reducir tiempos de acceso a datos en disco.

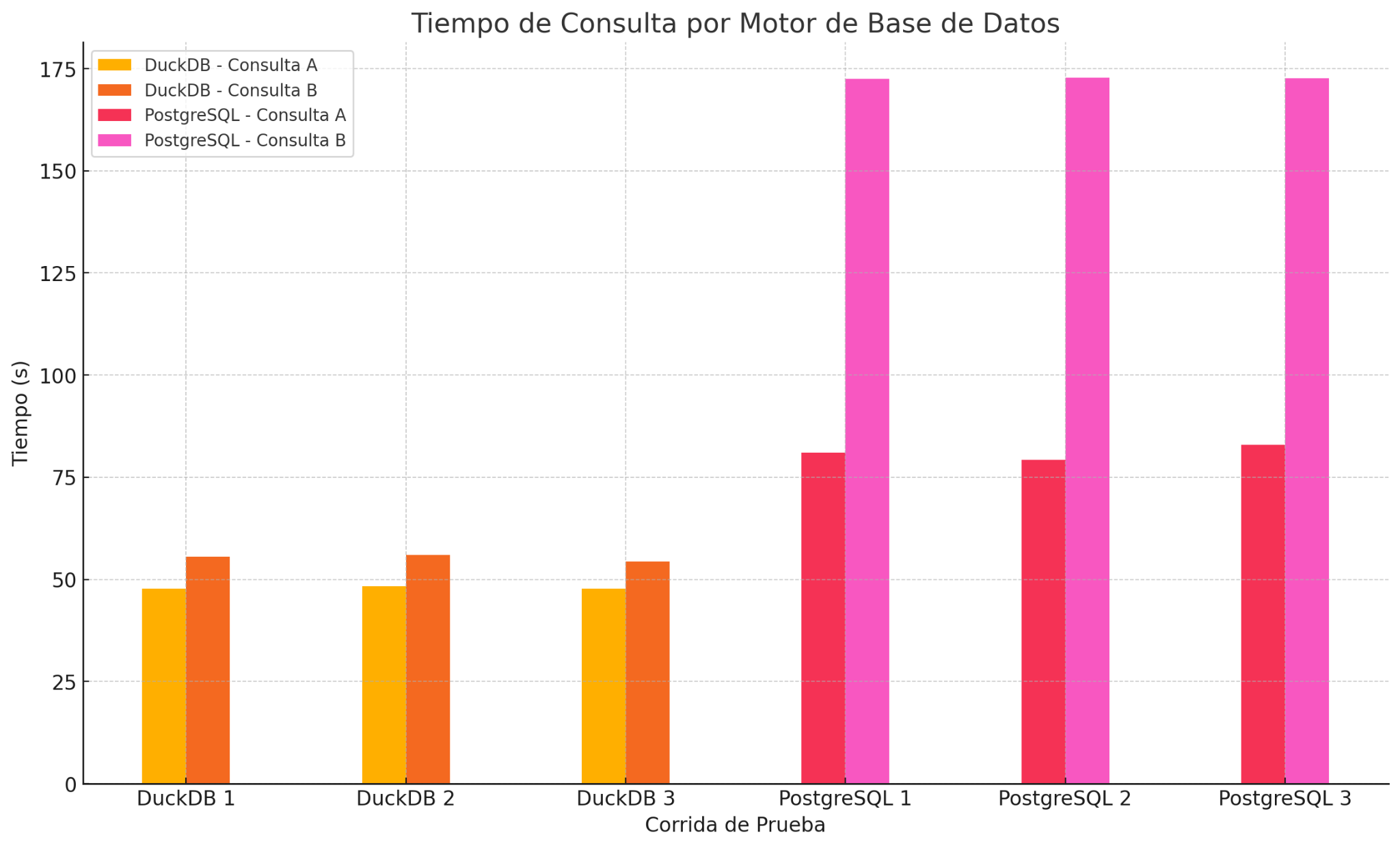
Asimismo, DuckDB utiliza una implementación de vectorización de consultas del motor de consultas X100 en MonetDB (Boncz et al., 2005). Debido a la homogeneidad de datos dentro de una columna (i.e., un mismo tipo de dato, como *double*) en el modelo columnar de bases de datos, es posible aplicar optimizaciones (e.g., operaciones SIMD del procesador) que serían limitadas en el modelo tabular.

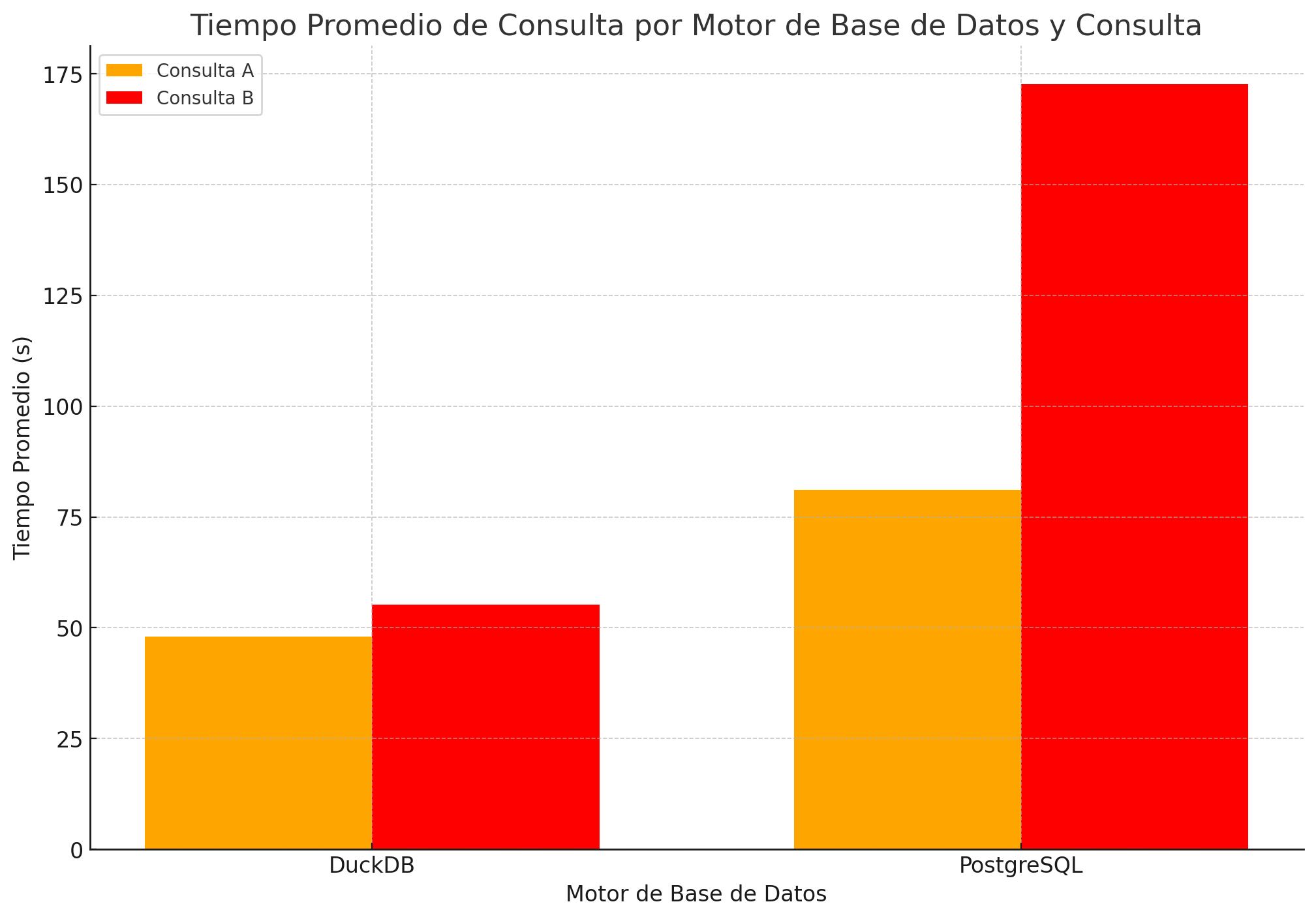
Por otro lado, es importante señalar que DuckDB no cuenta con facilidades avanzadas de recuperación de base de datos (e.g., una implementación del algoritmo ARIES). El caso de uso de DuckDB es análisis de datos interactivo y uso empotrado (embedded) dentro de otra aplicación, similar al motor de base de datos SQLite (SQLite, 2024).











**Referencias**

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/

City of New York (2024). *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

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

**Anexos**

**Anexo 1.** Listado de archivos del conjunto de datos, junto con tamaño en MB:

*490M fhvhv\_tripdata\_2019-02.parquet*

*583M fhvhv\_tripdata\_2019-03.parquet*

*534M fhvhv\_tripdata\_2019-04.parquet*

*545M fhvhv\_tripdata\_2019-05.parquet*

*512M fhvhv\_tripdata\_2019-06.parquet*

*493M fhvhv\_tripdata\_2019-07.parquet*

*479M fhvhv\_tripdata\_2019-08.parquet*

*493M fhvhv\_tripdata\_2019-09.parquet*

*524M fhvhv\_tripdata\_2019-10.parquet*

*536M fhvhv\_tripdata\_2019-11.parquet*

*550M fhvhv\_tripdata\_2019-12.parquet*

*507M fhvhv\_tripdata\_2020-01.parquet*

*533M fhvhv\_tripdata\_2020-02.parquet*

*331M fhvhv\_tripdata\_2020-03.parquet*

*110M fhvhv\_tripdata\_2020-04.parquet*

*154M fhvhv\_tripdata\_2020-05.parquet*

*189M fhvhv\_tripdata\_2020-06.parquet*

*249M fhvhv\_tripdata\_2020-07.parquet*

*277M fhvhv\_tripdata\_2020-08.parquet*

*301M fhvhv\_tripdata\_2020-09.parquet*

*330M fhvhv\_tripdata\_2020-10.parquet*

*288M fhvhv\_tripdata\_2020-11.parquet*

*290M fhvhv\_tripdata\_2020-12.parquet*

*295M fhvhv\_tripdata\_2021-01.parquet*

*289M fhvhv\_tripdata\_2021-02.parquet*

*352M fhvhv\_tripdata\_2021-03.parquet*

*352M fhvhv\_tripdata\_2021-04.parquet*

*370M fhvhv\_tripdata\_2021-05.parquet*

*376M fhvhv\_tripdata\_2021-06.parquet*

*378M fhvhv\_tripdata\_2021-07.parquet*

*365M fhvhv\_tripdata\_2021-08.parquet*

*376M fhvhv\_tripdata\_2021-09.parquet*

*411M fhvhv\_tripdata\_2021-10.parquet*

*393M fhvhv\_tripdata\_2021-11.parquet*

*392M fhvhv\_tripdata\_2021-12.parquet*

*358M fhvhv\_tripdata\_2022-01.parquet*

*389M fhvhv\_tripdata\_2022-02.parquet*

*450M fhvhv\_tripdata\_2022-03.parquet*

*435M fhvhv\_tripdata\_2022-04.parquet*

*447M fhvhv\_tripdata\_2022-05.parquet*

*437M fhvhv\_tripdata\_2022-06.parquet*

*424M fhvhv\_tripdata\_2022-07.parquet*

*417M fhvhv\_tripdata\_2022-08.parquet*

*437M fhvhv\_tripdata\_2022-09.parquet*

*473M fhvhv\_tripdata\_2022-10.parquet*

*443M fhvhv\_tripdata\_2022-11.parquet*

*481M fhvhv\_tripdata\_2022-12.parquet*

*452M fhvhv\_tripdata\_2023-01.parquet*

*438M fhvhv\_tripdata\_2023-02.parquet*

*499M fhvhv\_tripdata\_2023-03.parquet*

*470M fhvhv\_tripdata\_2023-04.parquet*

*490M fhvhv\_tripdata\_2023-05.parquet*

*476M fhvhv\_tripdata\_2023-06.parquet*

*470M fhvhv\_tripdata\_2023-07.parquet*

*430M fhvhv\_tripdata\_2023-08.parquet*

*456M fhvhv\_tripdata\_2023-09.parquet*

*464M fhvhv\_tripdata\_2023-10.parquet*

*444M fhvhv\_tripdata\_2023-11.parquet*

*468M fhvhv\_tripdata\_2023-12.parquet*

*451M fhvhv\_tripdata\_2024-01.parquet*

*442M fhvhv\_tripdata\_2024-02.parquet*

*485M fhvhv\_tripdata\_2024-03.parquet*

**Anexo 2.** Listado de enlaces a los archivos del conjunto de datos

*https://d37ci6vzurychx.cloudfront.net/trip-data/fhvhv\_tripdata\_2019-02.parquet*

*https://d37ci6vzurychx.cloudfront.net/trip-data/fhvhv\_tripdata\_2019-03.parquet*

*https://d37ci6vzurychx.cloudfront.net/trip-data/fhvhv\_tripdata\_2019-04.parquet*

*https://d37ci6vzurychx.cloudfront.net/trip-data/fhvhv\_tripdata\_2019-05.parquet*

*https://d37ci6vzurychx.cloudfront.net/trip-data/fhvhv\_tripdata\_2019-06.parquet*

*https://d37ci6vzurychx.cloudfront.net/trip-data/fhvhv\_tripdata\_2019-07.parquet*

*https://d37ci6vzurychx.cloudfront.net/trip-data/fhvhv\_tripdata\_2019-08.parquet*

*https://d37ci6vzurychx.cloudfront.net/trip-data/fhvhv\_tripdata\_2019-09.parquet*

*https://d37ci6vzurychx.cloudfront.net/trip-data/fhvhv\_tripdata\_2019-10.parquet*

*https://d37ci6vzurychx.cloudfront.net/trip-data/fhvhv\_tripdata\_2019-11.parquet*

*https://d37ci6vzurychx.cloudfront.net/trip-data/fhvhv\_tripdata\_2019-12.parquet*

*https://d37ci6vzurychx.cloudfront.net/trip-data/fhvhv\_tripdata\_2020-01.parquet*

*https://d37ci6vzurychx.cloudfront.net/trip-data/fhvhv\_tripdata\_2020-02.parquet*

*https://d37ci6vzurychx.cloudfront.net/trip-data/fhvhv\_tripdata\_2020-03.parquet*

*https://d37ci6vzurychx.cloudfront.net/trip-data/fhvhv\_tripdata\_2020-04.parquet*

*https://d37ci6vzurychx.cloudfront.net/trip-data/fhvhv\_tripdata\_2020-05.parquet*

*https://d37ci6vzurychx.cloudfront.net/trip-data/fhvhv\_tripdata\_2020-06.parquet*

*https://d37ci6vzurychx.cloudfront.net/trip-data/fhvhv\_tripdata\_2020-07.parquet*

*https://d37ci6vzurychx.cloudfront.net/trip-data/fhvhv\_tripdata\_2020-08.parquet*

*https://d37ci6vzurychx.cloudfront.net/trip-data/fhvhv\_tripdata\_2020-09.parquet*

*https://d37ci6vzurychx.cloudfront.net/trip-data/fhvhv\_tripdata\_2020-10.parquet*

*https://d37ci6vzurychx.cloudfront.net/trip-data/fhvhv\_tripdata\_2020-11.parquet*

*https://d37ci6vzurychx.cloudfront.net/trip-data/fhvhv\_tripdata\_2020-12.parquet*

*https://d37ci6vzurychx.cloudfront.net/trip-data/fhvhv\_tripdata\_2021-01.parquet*

*https://d37ci6vzurychx.cloudfront.net/trip-data/fhvhv\_tripdata\_2021-02.parquet*

*https://d37ci6vzurychx.cloudfront.net/trip-data/fhvhv\_tripdata\_2021-03.parquet*

*https://d37ci6vzurychx.cloudfront.net/trip-data/fhvhv\_tripdata\_2021-04.parquet*

*https://d37ci6vzurychx.cloudfront.net/trip-data/fhvhv\_tripdata\_2021-05.parquet*

*https://d37ci6vzurychx.cloudfront.net/trip-data/fhvhv\_tripdata\_2021-06.parquet*

*https://d37ci6vzurychx.cloudfront.net/trip-data/fhvhv\_tripdata\_2021-07.parquet*

*https://d37ci6vzurychx.cloudfront.net/trip-data/fhvhv\_tripdata\_2021-08.parquet*

*https://d37ci6vzurychx.cloudfront.net/trip-data/fhvhv\_tripdata\_2021-09.parquet*

*https://d37ci6vzurychx.cloudfront.net/trip-data/fhvhv\_tripdata\_2021-10.parquet*

*https://d37ci6vzurychx.cloudfront.net/trip-data/fhvhv\_tripdata\_2021-11.parquet*

*https://d37ci6vzurychx.cloudfront.net/trip-data/fhvhv\_tripdata\_2021-12.parquet*

*https://d37ci6vzurychx.cloudfront.net/trip-data/fhvhv\_tripdata\_2022-01.parquet*

*https://d37ci6vzurychx.cloudfront.net/trip-data/fhvhv\_tripdata\_2022-02.parquet*

*https://d37ci6vzurychx.cloudfront.net/trip-data/fhvhv\_tripdata\_2022-03.parquet*

*https://d37ci6vzurychx.cloudfront.net/trip-data/fhvhv\_tripdata\_2022-04.parquet*

*https://d37ci6vzurychx.cloudfront.net/trip-data/fhvhv\_tripdata\_2022-05.parquet*

*https://d37ci6vzurychx.cloudfront.net/trip-data/fhvhv\_tripdata\_2022-06.parquet*

*https://d37ci6vzurychx.cloudfront.net/trip-data/fhvhv\_tripdata\_2022-07.parquet*

*https://d37ci6vzurychx.cloudfront.net/trip-data/fhvhv\_tripdata\_2022-08.parquet*

*https://d37ci6vzurychx.cloudfront.net/trip-data/fhvhv\_tripdata\_2022-09.parquet*

*https://d37ci6vzurychx.cloudfront.net/trip-data/fhvhv\_tripdata\_2022-10.parquet*

*https://d37ci6vzurychx.cloudfront.net/trip-data/fhvhv\_tripdata\_2022-11.parquet*

*https://d37ci6vzurychx.cloudfront.net/trip-data/fhvhv\_tripdata\_2022-12.parquet*

*https://d37ci6vzurychx.cloudfront.net/trip-data/fhvhv\_tripdata\_2023-01.parquet*

*https://d37ci6vzurychx.cloudfront.net/trip-data/fhvhv\_tripdata\_2023-02.parquet*

*https://d37ci6vzurychx.cloudfront.net/trip-data/fhvhv\_tripdata\_2023-03.parquet*

*https://d37ci6vzurychx.cloudfront.net/trip-data/fhvhv\_tripdata\_2023-04.parquet*

*https://d37ci6vzurychx.cloudfront.net/trip-data/fhvhv\_tripdata\_2023-05.parquet*

*https://d37ci6vzurychx.cloudfront.net/trip-data/fhvhv\_tripdata\_2023-06.parquet*

*https://d37ci6vzurychx.cloudfront.net/trip-data/fhvhv\_tripdata\_2023-07.parquet*

*https://d37ci6vzurychx.cloudfront.net/trip-data/fhvhv\_tripdata\_2023-08.parquet*

*https://d37ci6vzurychx.cloudfront.net/trip-data/fhvhv\_tripdata\_2023-09.parquet*

*https://d37ci6vzurychx.cloudfront.net/trip-data/fhvhv\_tripdata\_2023-10.parquet*

*https://d37ci6vzurychx.cloudfront.net/trip-data/fhvhv\_tripdata\_2023-11.parquet*

*https://d37ci6vzurychx.cloudfront.net/trip-data/fhvhv\_tripdata\_2023-12.parquet*

*https://d37ci6vzurychx.cloudfront.net/trip-data/fhvhv\_tripdata\_2024-01.parquet*

*https://d37ci6vzurychx.cloudfront.net/trip-data/fhvhv\_tripdata\_2024-02.parquet*

*https://d37ci6vzurychx.cloudfront.net/trip-data/fhvhv\_tripdata\_2024-03.parquet*

**Anexo 3**. **Comandos de DuckDB para análisis exploratorio de datos**

.timer on

select count(\*) from '\*.parquet';

select \* from 'taxi\_zone\_lookup.csv';

select Borough from 'taxi\_zone\_lookup.csv';

describe select \* from '\*.parquet';

select avg(base\_passenger\_fare), avg(driver\_pay), avg(trip\_miles), avg(trip\_time) from '\*.parquet';

select Borough, avg(base\_passenger\_fare), avg(driver\_pay), avg(trip\_miles), avg(trip\_time) from read\_parquet('\*.parquet') join 'taxi\_zone\_lookup.csv' on LocationID = PULocationID group by Borough;

select Borough, avg(base\_passenger\_fare), avg(driver\_pay), avg(trip\_miles), avg(trip\_time) from read\_parquet('\*.parquet') join 'taxi\_zone\_lookup.csv' on LocationID = PULocationID group by Borough order by avg(base\_passenger\_fare);

select regr\_slope(base\_passenger\_fare, trip\_time), regr\_intercept(base\_passenger\_fare, trip\_time) from '\*.parquet';

select corr(base\_passenger\_fare, trip\_time), corr(base\_passenger\_fare, trip\_miles) from '\*.parquet';

select corr(base\_passenger\_fare, base\_passenger\_fare) from '\*.parquet';

select regr\_r2(base\_passenger\_fare, trip\_time), regr\_r2(base\_passenger\_fare, trip\_miles) from '\*.parquet';

select regr\_r2(driver\_pay, trip\_time), regr\_r2(driver\_pay, trip\_miles) from '\*.parquet';

select regr\_r2(driver\_pay, tolls), regr\_r2(base\_passenger\_fare, tolls) from '\*.parquet';

select regr\_r2(driver\_pay, tips), regr\_r2(base\_passenger\_fare, tips) from '\*.parquet';

select regr\_r2(driver\_pay, congestion\_surcharge), regr\_r2(base\_passenger\_fare, congestion\_surcharge) from '\*.parquet';

explain select count(\*) from '\*.parquet';

explain analyze select count(\*) from '\*.parquet';

**Anexo 4. Consultas en DuckDB**

***Consulta A***

SELECT

regr\_slope(base\_passenger\_fare, trip\_miles) AS slope\_passenger\_miles,

regr\_intercept(base\_passenger\_fare, trip\_miles) AS intercept\_passenger\_miles,

regr\_slope(base\_passenger\_fare, trip\_time) AS slope\_passenger\_time,

regr\_intercept(base\_passenger\_fare, trip\_time) AS intercept\_passenger\_time,

regr\_slope(driver\_pay, trip\_miles) AS slope\_driver\_miles,

regr\_intercept(driver\_pay, trip\_miles) AS intercept\_driver\_miles,

regr\_slope(driver\_pay, trip\_time) AS slope\_driver\_time,

regr\_intercept(driver\_pay, trip\_time) AS intercept\_driver\_time

FROM read\_parquet('\*.parquet');

***Consulta B***

SELECT Borough AS nyc\_borough,

AVG(base\_passenger\_fare) AS avg\_passenger\_fare,

AVG(driver\_pay) AS avg\_driver,

AVG(trip\_miles) AS avg\_miles,

AVG(trip\_time) AS avg\_time

FROM read\_parquet('\*.parquet')

JOIN 'taxi\_zone\_lookup.csv' ON LocationID = PULocationID

GROUP BY Borough

ORDER BY avg\_passenger;

**Anexo 5. Consultas en *PostgreSQL***

***Preparación e inserción de datos desde DuckDB hacia PostgreSQL***

attach 'dbname=postgres user=postgres host=127.0.0.1' as postgres\_db (type postgres);

create table postgres\_db.nyc\_tlc (

pulocationid int,

dolocationid int,

trip\_miles double,

trip\_time double,

base\_passenger\_fare double,

driver\_pay double

);

insert into postgres\_db.nyc\_tlc

select PULocationID, DOLocationID, trip\_miles, trip\_time, base\_passenger\_fare, driver\_pay from '\*.parquet';

***Consulta A***

SELECT

regr\_slope(base\_passenger\_fare, trip\_miles) AS slope\_passenger\_miles,

regr\_intercept(base\_passenger\_fare, trip\_miles) AS intercept\_passenger\_miles,

regr\_slope(base\_passenger\_fare, trip\_time) AS slope\_passenger\_time,

regr\_intercept(base\_passenger\_fare, trip\_time) AS intercept\_passenger\_time,

regr\_slope(driver\_pay, trip\_miles) AS slope\_driver\_miles,

regr\_intercept(driver\_pay, trip\_miles) AS intercept\_driver\_miles,

regr\_slope(driver\_pay, trip\_time) AS slope\_driver\_time,

regr\_intercept(driver\_pay, trip\_time) AS intercept\_driver\_time

FROM nyc\_tlc;

***Consulta B***

SELECT Borough AS nyc\_borough,

AVG(base\_passenger\_fare) AS avg\_passenger\_fare,

AVG(driver\_pay) AS avg\_driver,

AVG(trip\_miles) AS avg\_miles,

AVG(trip\_time) AS avg\_time

FROM nyc\_tlc

JOIN zones ON locationid = PULocationID

GROUP BY Borough

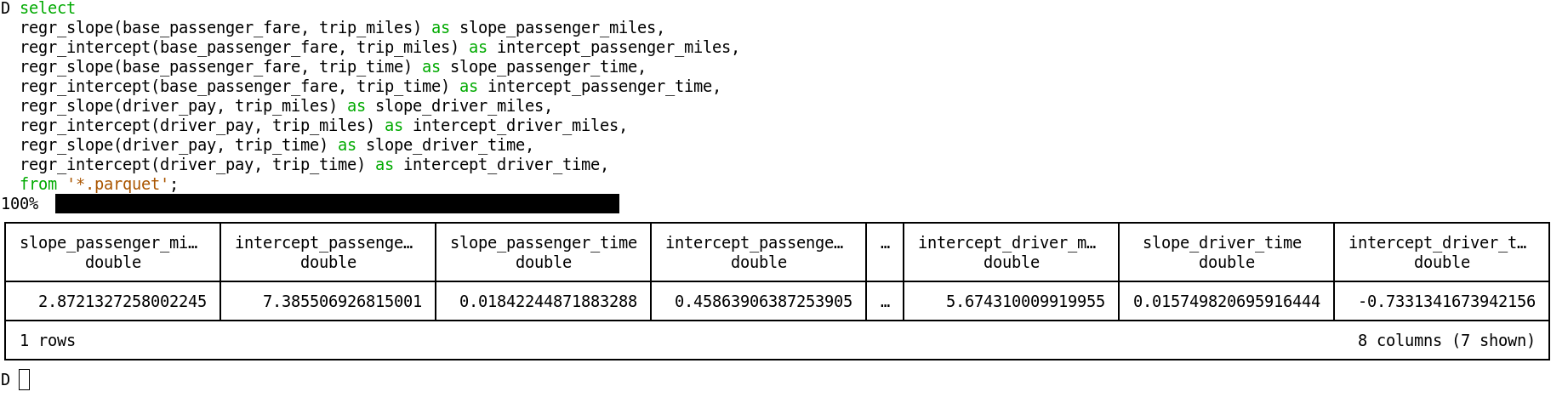
ORDER BY avg\_passenger\_fare;

**Anexo 6. Tiempos de ejecución de consultas**

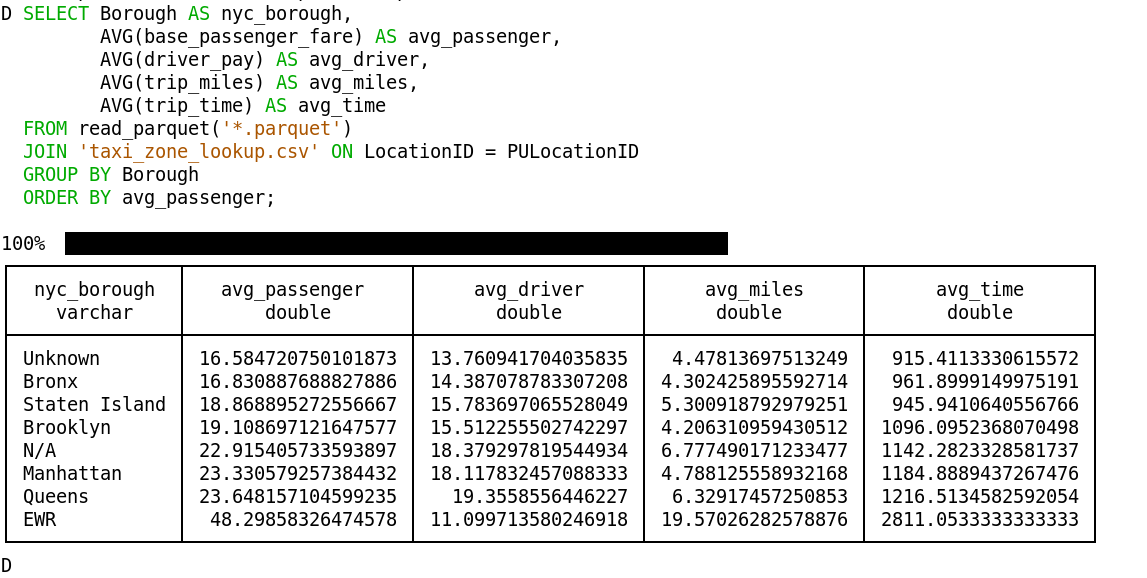
| **Corrida de prueba** | **Motor de base de datos** | **Consulta** | **Tiempo (s)** | **Promedio (s)** |
| --- | --- | --- | --- | --- |
| 1 | DuckDB | A | 47.717 | 47.937 |
| 2 | DuckDB | A | 48.397 |
| 3 | DuckDB | A | 47.700 |
| 1 | DuckDB | B | 55.478 | 55.271 |
| 2 | DuckDB | B | 55.947 |
| 3 |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

**Anexo 7. Resultados de consultas en DuckDB**

*Consulta A*

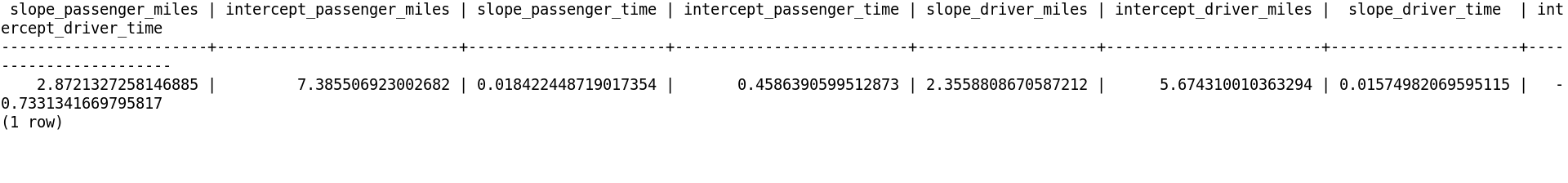


*Consulta B*



**Anexo 8. Resultados de consultas en PostgreSQL**

*Consulta A*



*Consulta B*

