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 exploran las capacidades de DuckDB, un motor de base de datos embebido optimizado para el análisis de datos. Aunque técnicamente es un motor relacional, DuckDB incluye características NoSQL que permiten mayor flexibilidad y eficiencia. De particular interés para este trabajo es su motor de ejecución de consultas vectorizado y columnar. Este motor realiza operaciones utilizando un vector con los datos de las columnas, en lugar de procesar secuencialmente todas las tuplas de las tablas, como lo hacen PostgreSQL, MySQL u otros sistemas tradicionales (DuckDB Foundation, 2024).

Además, DuckDB tiene soporte nativo para la lectura de archivos externos como CSV o Parquet. Este último 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). DuckDB aprovecha estas características, permitiendo cargar datos directamente desde estos archivos y ejecutar consultas analíticas con los datos. Esto facilita el análisis de datos sin requerir un uso intensivo de la memoria.

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.

Descripción del problema

El análisis de grandes volúmenes de datos presenta un desafío constante en la gestión de bases de datos. Tradicionalmente, se han utilizado motores de bases de datos relacionales como PostgreSQL para manejar y procesar estos datos. No obstante, el crecimiento exponencial de los datos y la necesidad de realizar análisis más complejos y rápidos exigen evaluar alternativas que ofrezcan mejor rendimiento y eficiencia.

En este estudio, se propone comparar el rendimiento de DuckDB con PostgreSQL. La comparación se centrará en medir el tiempo de ejecución de dos consultas específicas, diseñadas para realizar un análisis exploratorio de un conjunto de datos grande y complejo. Estas consultas representarán operaciones comunes en análisis de datos, como agregación, filtrado y combinación de datos.

Descripción de datos

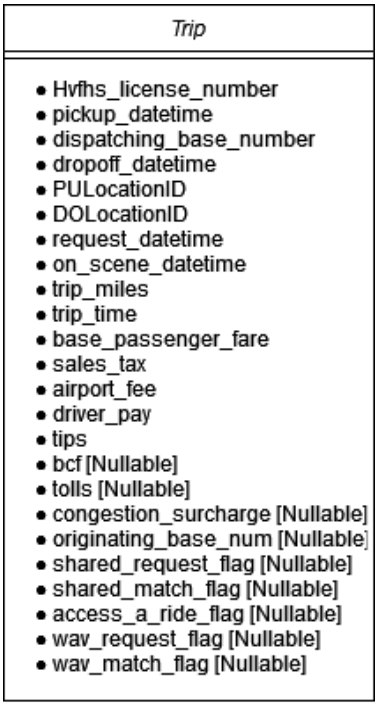
El conjunto de datos a analizar para el proyecto consiste en los viajes en plataformas de transporte, recopilados mensualmente por la Comisión de Taxis y Limosinas de la ciudad de Nueva York, y disponibles de forma pública (Taxi & Limousine Commission, 2024). Maneja 62 archivos en formato Parquet, que suman 25.2 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 7).

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 10,000 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, ya 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 Conceptual

El uso de modelos conceptuales y lógicos en bases de datos NoSQL, aunque no es común en la práctica debido a su naturaleza schemaless, es esencial para comprender y visualizar los datos antes de la implementación física. Según West (2011), la modelación conceptual proporciona una comprensión profunda de los datos y su estructura, lo que facilita tanto la implementación como el mantenimiento del sistema. West argumenta que un buen modelado conceptual actúa como un mapa que guía a los desarrolladores y administradores de bases de datos en la organización y uso eficiente de los datos. Sin un modelo claro, el riesgo de inconsistencias, duplicaciones y problemas de rendimiento aumenta significativamente. Una representación básica del esquema a utilizar con los datos de la *Comisión de Taxis y Limusinas de Nueva York* (2024), como menciona la estructura de West (2011) se muestra en la Figura 1.

Figura 1. *Diagrama simplificado de viajes HVFHS en la Ciudad de Nueva York*



*Nota. La etiqueta **[Nullable]** Indica que el campo puede tener valores nulos, es decir, no es obligatorio que contenga un valor para cada registro.*

En este contexto y de acuerdo al brindado por la *Comisión de Taxis y Limusinas de Nueva York* (2024), se utilizará un modelo de datos columnar representado mediante el procesamiento de archivos en formato Parquet, que se clasifica como un modelo NoSQL. El esquema de datos solamente incluye los archivos .parquet de registros de viajes de alto volumen (High-Volume For-Hire Services o *HVFHS*). El modelo tiene como objetivo organizar los datos de manera eficiente para su análisis y uso en aplicaciones relacionadas con la gestión de servicios de transporte en la ciudad de Nueva York. Al utilizar un modelo conceptual claro y simplificado, se asegura que los datos sean consistentes y accesibles, mejorando así la calidad del análisis y la toma de decisiones en el ámbito del transporte urbano. El esquema incluye una única tabla que mantiene la relación entre todos los viajes realizados con varios atributos que pueden ser definidos como nulos al no ser agregados.

Atributos y Descripciones

1. **Hvfhs_license_num (*String*):** Número de licencia *TLC* de la base o negocio *HVFHS*. Este campo identifica de manera única a cada base o negocio que despacha los viajes.
2. **dispatching_base_num (*String*):** Número de licencia de base *TLC* de la base que despachó el viaje. Este campo es crucial para identificar la base que organizó y envió el viaje.
3. **pickup_datetime (*Timestamp*):** Fecha y hora de recogida del viaje. Indica el momento exacto en que el pasajero fue recogido.
4. **dropoff_datetime (*Timestamp*):** Fecha y hora de finalización del viaje. Indica el momento exacto en que el pasajero fue dejado en su destino.
5. **PULocationID (*Integer*):** Zona de taxi *TLC* en la que comenzó el viaje. Este campo utiliza un identificador numérico para las zonas geográficas definidas por la *TLC*.
6. **DOLocationID (*Integer*):** Zona de taxi *TLC* en la que terminó el viaje. Similar a PULocationID, este campo identifica la zona de destino del viaje.
7. **request_datetime (*Timestamp*):** Fecha y hora en que el pasajero solicitó ser recogido. Proporciona la marca temporal de cuándo se hizo la solicitud.
8. **on_scene_datetime (*Timestamp*):** Fecha y hora en que el conductor llegó al lugar de recogida. Este atributo es opcional y se aplica solo a vehículos accesibles.
9. **trip_miles (*Float*):** Millas totales del viaje de pasajeros. Mide la distancia recorrida durante el viaje.
10. **trip_time (*Integer*):** Tiempo total en segundos del viaje de pasajeros. Mide la duración del viaje desde la recogida hasta la entrega del pasajero.
11. **base_passenger_fare (*Float*):** Tarifa base del pasajero antes de peajes, propinas, impuestos y tarifas. Representa el costo inicial del viaje.
12. **sales_tax (*Float*):** Cantidad total recaudada en el viaje para el impuesto sobre ventas del estado de Nueva York. Incluye el impuesto aplicable al viaje.
13. **airport_fee (Float):** Tarifa fija de $2.50 para tanto la recogida como la entrega en los aeropuertos de LaGuardia, Newark y John F. Kennedy. Es un cargo específico para viajes desde y hacia estos aeropuertos.
14. **driver_pay (*Float*):** Pago total al conductor, excluyendo peajes y propinas y neto de comisiones, recargos o impuestos. Indica la compensación neta del conductor.

15. **tips (*Float*):** Cantidad total de propinas recibidas del pasajero. Este campo registra la gratificación voluntaria dada al conductor.
16. Una cierta cantidad de atributos no se encuentran en todas las tuplas y tienen valores ya sea nulos o inexistentes. Estos son los siguientes:
    a. **bcf (*Float*):** Cantidad total recaudada en el viaje para el Fondo de Autos Negros (Black Car Fund). Este fondo es específico para ciertos tipos de servicios de transporte.
    b. **tolls (*Float*):** Cantidad total de todos los peajes pagados en el viaje. Registra los costos adicionales por peajes durante el trayecto.
    c. **congestion_surcharge (*Float*):** Cantidad total recaudada en el viaje por el recargo de congestión del estado de Nueva York. Aplica a áreas con alta densidad de tráfico.
    d. **originating_base_num (*String*):** Número de base que recibió la solicitud original del viaje. Identifica la base que inicialmente recibió la solicitud del pasajero.
    e. **shared_request_flag (*String*):** Indica si el pasajero aceptó un viaje compartido, independientemente de si fueron emparejados (Y/N). Este campo ayuda a identificar la disposición del pasajero a compartir el viaje.
    f. **shared_match_flag (*String*):** Indica si el pasajero compartió el vehículo con otro pasajero que reservó por separado en algún momento durante el viaje (Y/N). Este campo confirma si se realizó el viaje compartido.
    g. **access_a_ride_flag (*String*):** Indica si el viaje fue administrado en nombre de la Autoridad de Transporte Metropolitano (MTA) (Y/N).
    h. **wav_request_flag (*String*):** Indica si el pasajero solicitó un vehículo accesible para sillas de ruedas (WAV) (Y/N). Es un atributo opcional.
    i. **wav_match_flag (*String*):** Indica si el viaje ocurrió en un vehículo accesible para sillas de ruedas (WAV) (Y/N).

Además, autores como Li y Manoharan (2013) han destacado la importancia de los modelos conceptuales para optimizar el rendimiento y la escalabilidad en bases de datos NoSQL. Estos modelos 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.

Escenarios de análisis

En este proyecto, se compara el rendimiento de DuckDB y PostgreSQL en funciones de agregación estadística mediante dos tipos de consultas denominadas A y B.

La consulta A aplica las funciones *regr_slope* y *regr_intercept* para obtener la pendiente de la línea de regresión y su intersección con el eje y, respectivamente. La consulta B realiza una agregación de promedios de variables como el tiempo de viaje, la distancia de viaje, la tarifa de pasajero y la tarifa de conductor, agrupados por borough (una división administrativa similar a un distrito) de la ciudad de Nueva York. Para este propósito, se utiliza una tabla

adicional proporcionada por la Comisión de Taxis y Limosinas de la ciudad de Nueva York, mediante una operación de join.

El análisis se basa en datos almacenados en archivos Parquet y aplica funciones de regresión lineal para analizar la relación entre tarifas de pasajeros, pagos a conductores y características del viaje, como la distancia y el tiempo. Estas operaciones fueron seleccionadas debido a su complejidad con volúmenes de datos considerables y porque están disponibles en ambos motores de bases de datos con la misma semántica, permitiendo así comparaciones equivalentes. 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 la zona de la ciudad de Nueva York donde inicia un viaje.

Para medir los tiempos de ejecución, se habilitaron las opciones *.timer on* en DuckDB y *\timing* en PostgreSQL. Se realizaron tres corridas por cada combinación de factores: base de datos (DuckDB y PostgreSQL) y consulta (A y B, listadas en los anexos 3 y 4). Además, se efectuó un análisis exploratorio para asegurar la integridad de los datos utilizando la función describe de DuckDB (Figura 2).

El experimento se lleva a cabo en una máquina Dell XPS 13 (modelo 9315), con 12 CPUs Intel Core i5-1230U y 8 GB de RAM DDR5, utilizando Debian GNU/Linux 12.6 (Bookworm). Cada ejecución se realiza “en frío”, es decir, se detiene y se reinicia el proceso. En el caso de DuckDB, se sale 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 reinicia el servicio.

Figura 2. *Exploración de esquema de datos mediante DuckDB*

```
D describe select * from '*.parquet';
```

column_name varchar	column_type varchar	null varchar	key varchar	default varchar	extra varchar
hvfhs_license_num	VARCHAR	YES			
dispatching_base_num	VARCHAR	YES			
originating_base_num	VARCHAR	YES			
request_datetime	TIMESTAMP	YES			
on_scene_datetime	TIMESTAMP	YES			
pickup_datetime	TIMESTAMP	YES			
dropoff_datetime	TIMESTAMP	YES			
PULocationID	BIGINT	YES			
DOLocationID	BIGINT	YES			
trip_miles	DOUBLE	YES			
trip_time	BIGINT	YES			
base_passenger_fare	DOUBLE	YES			
tolls	DOUBLE	YES			
bcf	DOUBLE	YES			
sales_tax	DOUBLE	YES			
congestion_surcharge	DOUBLE	YES			
airport_fee	INTEGER	YES			
tips	DOUBLE	YES			
driver_pay	DOUBLE	YES			
shared_request_flag	VARCHAR	YES			
shared_match_flag	VARCHAR	YES			
access_a_ride_flag	VARCHAR	YES			
wav_request_flag	VARCHAR	YES			
wav_match_flag	INTEGER	YES			
24 rows					6 columns

Resultados

Tabla 1. *Tiempos de Ejecución Promedio de Consultas en DuckDB y PostgreSQL*

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	

Tal como se puede observar en la tabla 1 y en la figura 3, se compararon los tiempos de ejecución de las consultas A y B en los motores de bases de datos DuckDB, mientras que en la figura 4 los de PostgreSQL. Los resultados obtenidos muestran que, en promedio, la consulta A en DuckDB tarda 47.937 segundos, mientras que la consulta B tarda 55.271 segundos. Por otro lado, en PostgreSQL, la consulta A tarda 81.109 segundos y la consulta B 172.682 segundos.

Figura 3. *Tiempos de Ejecución de Consultas en DuckDB*

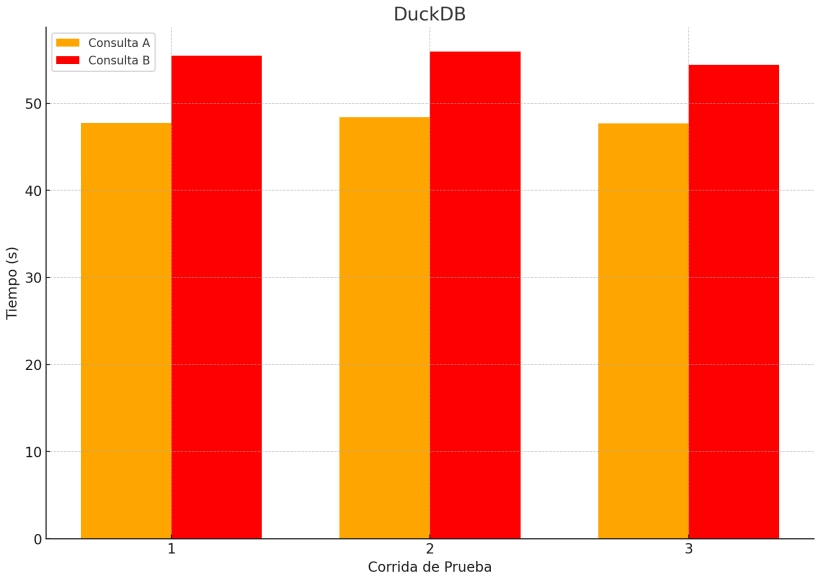
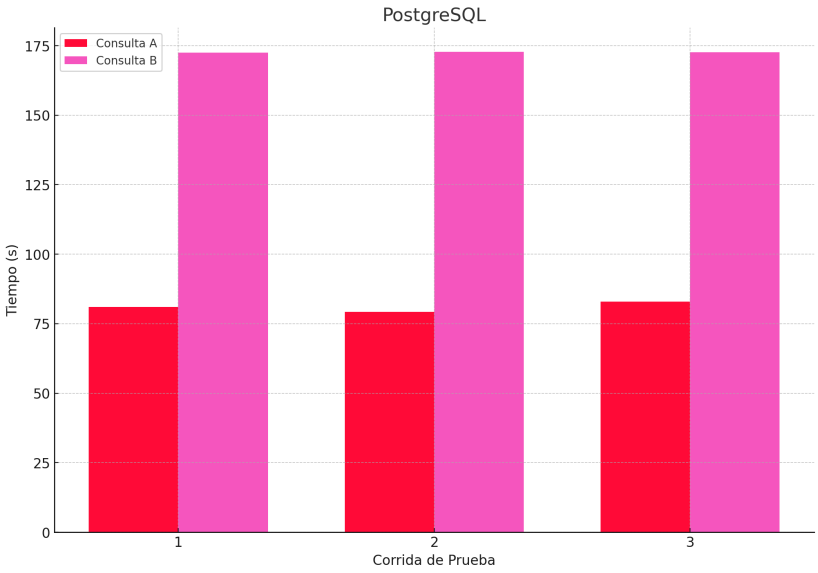


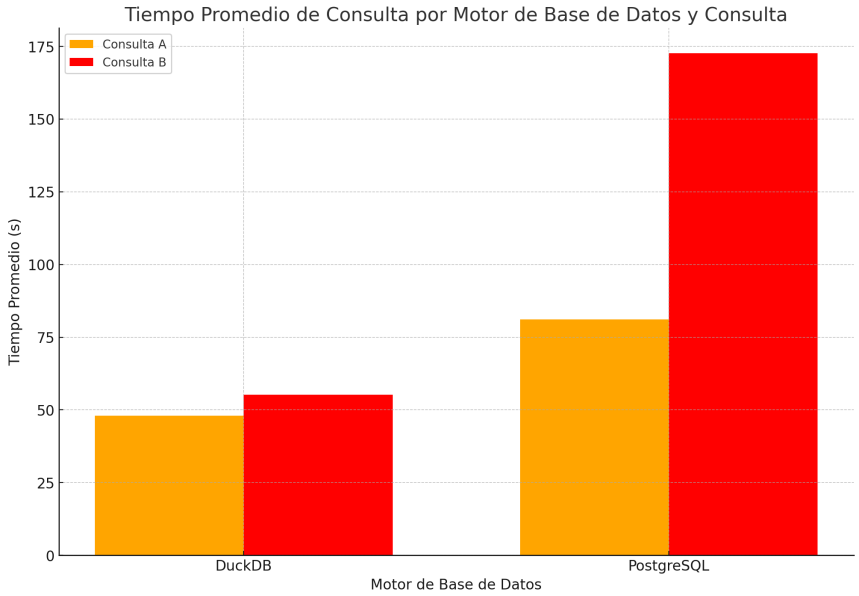
Figura 4. *Tiempos de Ejecución de Consultas en PostgreSQL*



Estos resultados indican que DuckDB presenta una mejora significativa en el rendimiento en comparación con PostgreSQL. Se puede observar en la figura 4 que DuckDB es

aproximadamente 1.69 veces más rápido que PostgreSQL en la ejecución de la consulta A, y 3.12 veces más rápido en la ejecución de la consulta B.

Figura 4. *Tiempos de Ejecución Promedio de Consultas en DuckDB y PostgreSQL*



La mejora en el rendimiento de DuckDB se puede atribuir a varias razones técnicas.En primer lugar, DuckDB accede a los archivos de forma directa y procesa nativamente los archivos en formato Parquet. El formato columnar y la compresión de datos de Parquet permiten reducir significativamente los tiempos de acceso a los datos en disco. En contraste, PostgreSQL requiere cargar los datos en una tabla desde los archivos, lo cual añade un paso adicional y tiempo de procesamiento. Por otra parte, DuckDB utiliza una implementación de vectorización de consultas del motor de consultas X100 en MonetDB, tal como se describe en el trabajo de Boncz et al. (2005). La vectorización de consultas aprovecha la homogeneidad de datos dentro de una columna (por ejemplo, un mismo tipo de dato, como double) en el modelo columnar de bases de datos. Esto permite aplicar optimizaciones, como las operaciones SIMD del procesador, que serían limitadas en el modelo tabular utilizado por PostgreSQL.

No obstante, es importante señalar que DuckDB no cuenta con facilidades avanzadas de recuperación de bases de datos, como una implementación del algoritmo ARIES. Por lo tanto, su uso está más orientado al análisis de datos interactivo y al uso embebido dentro de otras aplicaciones, similar al motor de base de datos SQLite (SQLite, 2024).

DuckDB demuestra ser una opción altamente eficiente para la ejecución de consultas, especialmente en escenarios donde el acceso y procesamiento de archivos en formato Parquet es esencial. Sin embargo, es necesario considerar sus limitaciones en cuanto a las facilidades avanzadas de recuperación de bases de datos.

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/

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 eBooks*. https://doi.org/10.1016/c2009-0-30508-5

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

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. 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 3. 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 4. 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 5. Resultados de consultas en DuckDB

Consulta A

slope_passenger_mi… double	intercept_passenge… double	slope_passenger_time double	intercept_passenge… double	…	intercept_driver_m… double	slope_driver_time double	intercept_driver_t… double
2.8721327258002245	7.385506926815001	0.01842244871883288	0.45863906387253905	…	5.674310009919955	0.015749820695916444	-0.7331341673942156
1 rows							8 columns (7 shown)

Consulta B

nyc_borough varchar	avg_passenger double	avg_driver double	avg_miles double	avg_time double
Unknown	16.584720750101873	13.760941704035835	4.47813697513249	915.4113330615572
Bronx	16.830887688827886	14.387078783307208	4.302425895592714	961.8999149975191
Staten Island	18.868895272556667	15.783697065528049	5.300918792979251	945.9410640556766
Brooklyn	19.108697121647577	15.512255502742297	4.206310959430512	1096.0952368070498
N/A	22.915405733593897	18.379297819544934	6.777490171233477	1142.2823328581737
Manhattan	23.330579257384432	18.117832457088333	4.788125558932168	1184.8889437267476
Queens	23.648157104599235	19.3558556446227	6.32917457250853	1216.5134582592054
EWR	48.29858326474578	11.099713580246918	19.57026282578876	2811.0533333333333

Anexo 6. Resultados de consultas en PostgreSQL

Consulta A

```
 slope_passenger_miles | intercept_passenger_miles | slope_passenger_time | intercept_passenger_time | slope_driver_miles | intercept_driver_miles |  slope_driver_time  | int
ercept_driver_time
-----------------------+---------------------------+----------------------+--------------------------+--------------------+------------------------+---------------------+----
------------------
    2.8721327258146885 |         7.385506923002682 | 0.018422448719017354 |       0.4586390599512873 | 2.3558808670587212 |      5.674310010363294 | 0.01574982069595115 |   -
0.7331341669795817
(1 row)
```

Consulta B

nyc_borough	avg_passenger_fare	avg_driver	avg_miles	avg_time
Unknown	16.5847207501019	13.760941704035835	4.478136975132487	915.4113330615572
Bronx	16.830887689258027	14.387078783469008	4.302425895591926	961.8999149975191
Staten Island	18.868895272463483	15.78369706546883	5.300918792979318	945.9410640556766
Brooklyn	19.108697121929993	15.512255503906616	4.206310959429094	1096.0952368070498
N/A	22.915405733593488	18.379297819544796	6.77749017123349	1142.2823328581737
Manhattan	23.330579258043425	18.11783245681816	4.788125558929057	1184.8889437267476
Queens	23.648157104428382	19.355855645506203	6.3291745725107535	1216.5134582592054
EWR	48.29858326474749	11.09971358024691	19.57026282578872	2811.0533333333333

(8 rows)

Anexo 7. 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