A close up of a sign

Description automatically generated

**Escuela de Ingeniería en Computación**

**IC-4302 – Bases de Datos II**

**Proyecto 2**

Red Social de Viajes

**Estudiantes:**

Esteban Josué Solano Araya – 2021579468

Pablo Mesén Alvarado – 2023071259

Daniel Zeas Brown – 2023147474

**Profesor:**

Keneth Obando Rodríguez

**22 de noviembre de 2024**

**II Semestre 2024**

Tabla de Contenidos

[1. Introducción 3](#_Toc183202498)

[2. Estructura del Proyecto 3](#_Toc183202499)

[3. Cómo se cargaron los datos en Neo4j 3](#_Toc183202500)

[4. Consultas utilizadas en Neo4j 6](#_Toc183202501)

[4.1 Extracción de datos 6](#_Toc183202502)

[5. Análisis con Spark 6](#_Toc183202503)

[5.1 Importación de librerías y Conexión a PostgreSQL 6](#_Toc183202504)

[5.2 Conexión a Neo4j e Inicialización de un cursor 7](#_Toc183202505)

[5.3 Creación de la sesión de Spark 8](#_Toc183202506)

[5.4 Función para ejecutar consultas Cypher 8](#_Toc183202507)

[5.5 Ejecución de consulta Cypher 8](#_Toc183202508)

[5.6 GTPC (Gasto Total por Cliente) 9](#_Toc183202509)

[5.7 PMC (Productos Más Comprados) 11](#_Toc183202510)

[5.8 PGPC (Promedio General Por Cliente) 12](#_Toc183202511)

[5.9 FCPC (Frecuencia de Compras Por Cliente) 13](#_Toc183202512)

[6. Resultados 15](#_Toc183202513)

[6.1 Gasto Total por Cliente (GTPC) / total\_spent\_per\_customer 15](#_Toc183202514)

[6.2 Productos más Comprados (PMC) / product\_purchase\_count 15](#_Toc183202515)

[6.3 Promedio de Gasto por Cliente (PGPC) / average\_spent\_per\_customer 16](#_Toc183202516)

[6.4 Frecuencia de Compra por Cliente / transaction\_count\_per\_customer 16](#_Toc183202517)

1. Introducción

Este proyecto analiza transacciones simuladas utilizando servicios integrados como Neo4j, Spark y PostgreSQL. Los datos se cargan inicialmente en Neo4j, se procesan mediante Spark para generar métricas clave, y se almacenan en PostgreSQL para facilitar consultas adicionales.

2. Estructura del Proyecto

***A screenshot of a computer

Description automatically generated***

3. Cómo se cargaron los datos en Neo4j

1. **Lectura del archivo CSV**  
   Utilizamos la biblioteca pandas para leer el archivo Transactions.csv, seleccionando las columnas relevantes:

* transaction\_id
* product\_id
* customer\_id
* transaction\_date
* standard\_cost

Eliminamos filas con valores faltantes en las columnas clave para garantizar que los datos cargados sean consistentes.

1. **Limpieza de datos**

* La columna standard\_cost contenía valores formateados con caracteres como ‘$’ y ‘,’.   
  Estos se limpiaron y se transformaron en valores numéricos (float) para asegurar la compatibilidad con Neo4j.

1. **Creación de nodos y relaciones en Neo4j**  
   Los datos fueron cargados en Neo4j mediante transacciones Cypher, creando:

* Nodos para clientes (Customer).
* Nodos para productos (Product).
* Relación entre ambos nodos con las transacciones (TRANSACTION).

Cada transacción incluye propiedades como transaction\_id, transaction\_date y standard\_cost.

Código Utilizado para cargar datos en Neo4j junto a explicación en cada punto.

A screen shot of a computer code

Description automatically generated

4. Consultas utilizadas en Neo4j

4.1 Extracción de datos

Los datos fueron extraídos desde Neo4j con la siguiente consulta Cypher:

A black background with colorful text

Description automatically generated

Esta consulta generó una tabla con tres columnas: customer\_id, product\_id, y standard\_cost. Los datos se procesaron con Python usando el driver oficial de Neo4j:

A computer screen with colorful text

Description automatically generated

5. Análisis con Spark

5.1 Importación de librerías y Conexión a PostgreSQL

* pyspark.sql.SparkSession: Se usa para inicializar una sesión de Spark, que permite realizar análisis de datos distribuidos.
* neo4j.GraphDatabase: Facilita la conexión con una base de datos Neo4j y la ejecución de consultas Cypher.
* pandas: Ayuda a manipular y transformar los datos en estructuras tabulares.
* psycopg2: Proporciona una interfaz para conectarse y realizar consultas con PostgreSQL.

La función psycopg2.connect establece la conexión con una base de datos PostgreSQL utilizando:

* dbname: Nombre de la base de datos.
* user y password: Credenciales de usuario.
* hosty port: Dirección y puerto donde está corriendo el servidor.

A black screen with purple text

Description automatically generated

5.2 Conexión a Neo4j e Inicialización de un cursor

Conexión a Neo4j:

* + uri: Define la dirección del servidor Neo4j utilizando el protocolo bolt://.
  + GraphDatabase.driver: Inicializa un controlador para conectarse al servidor.
  + auth: Recibe un tuple con el nombre de usuario y contraseña para autenticar.

Inicialización de un cursor:

* + conn.cursor crea un cursor que se utiliza para ejecutar comandos SQL en PostgreSQL.

A computer screen with white text

Description automatically generated

5.3 Creación de la sesión de Spark

SparkSession.builder configura la aplicación Spark con:

* appName: El nombre de la aplicación Spark.
* master: Define el modo de ejecución como local.

A computer screen with blue text

Description automatically generated

5.4 Función para ejecutar consultas Cypher

La función recibe una consulta Cypher (query) como parámetro y ejecuta esta consulta en el servidor Neo4j.

* driver.session(): Abre una sesión temporal con Neo4j.
* session.run(query): Ejecuta la consulta Cypher.
* list(): Convierte los resultados en una lista para su manejo posterior.

A black background with white text

Description automatically generated

5.5 Ejecución de consulta Cypher

La consulta busca nodos de tipo Customer y Product conectados por relaciones TRANSACTION. Devuelve tres atributos:

* + - c.customer\_id: Identificador del cliente.
    - p.product\_id: Identificador del producto.
    - t.standard\_cost: Costo de la transacción.

Ejecución de la consulta:

* + Usa la función execute\_cypher\_query para enviar la consulta al servidor Neo4j y obtener los resultados en query\_output.

A computer screen with blue text

Description automatically generated

5.6 GTPC (Gasto Total por Cliente)

**Preparación de los datos**:

* Itera sobre los registros de query\_output (resultado de Neo4j).
* Extrae y organiza los valores en un diccionario con las claves customer\_id, product\_id y standard\_cost.
* Los diccionarios se almacenan en la lista GTPC\_data.

**Conversión a DataFrame:**

* Pandas: Convierte GTPC\_data en un DataFrame (estructura tabular).
* Spark: Transforma el DataFrame de pandas en un DataFrame compatible con Spark.

**Análisis con Spark:**

* Agrupa los datos por customer\_id.
* Calcula la suma de standard\_cost para cada cliente.
* Renombra la columna de la suma como total\_cost.
* Ordena los resultados por customer\_id.

A screen shot of a computer program

Description automatically generated

**Inserción de GTPC en PostgreSQL**

Creación de tabla:

* Crea una tabla llamada GTPC si no existe.
* Define dos columnas:
* customer\_id: Llave primaria.
* total\_cost: Representa el gasto total.

Inserción de datos:

* Convierte los datos procesados por Spark (almacenados en pandas) en una lista de tuplas para PostgreSQL.
* Usa executemany para insertar múltiples filas de manera eficiente.

A screen shot of a computer code

Description automatically generated

5.7 PMC (Productos Más Comprados)

**Preparación de los datos:**

* Similar al caso de GTPC, organiza los resultados en una lista de diccionarios con las claves product\_id y purchase\_count.

**Conversión a DataFrame:**

* Mismo procedimiento que GTPC: se convierten los datos a un DataFrame de pandas y luego a uno de Spark.

**Análisis con Spark:**

* Ordena los productos por la columna purchase\_count en orden descendente para conservar los productos más comprados en la parte superior.

A screen shot of a computer code

Description automatically generated

**Inserción de GTPC en PostgreSQL**

* + Similar al caso anterior

5.8 PGPC (Promedio General Por Cliente)

**Preparación de los datos:**

* Similar a GTPC, organiza los resultados en una lista de diccionarios con customer\_id y average\_cost.

**Conversión a DataFrame:**

* Idéntico al procedimiento de GTPC y PMC.

**Análisis con Spark:**

* Ordena los clientes por el ID para asegurar que los datos sean fáciles de interpretar y estén en orden lógico.



**Inserción de GTPC en PostgreSQL**

* Similar al caso anterior

5.9 FCPC (Frecuencia de Compras Por Cliente)

**Preparación de los datos:**

* Similar a las otras métricas, organiza los resultados en una lista de diccionarios con customer\_id y transaction\_count.

**Conversión a DataFrame:**

* Mismo procedimiento que GTPC, PMC y PGPC.

**Análisis con Spark:**

* Ordena los datos por el ID del cliente para mantener la consistencia.

A screen shot of a computer program

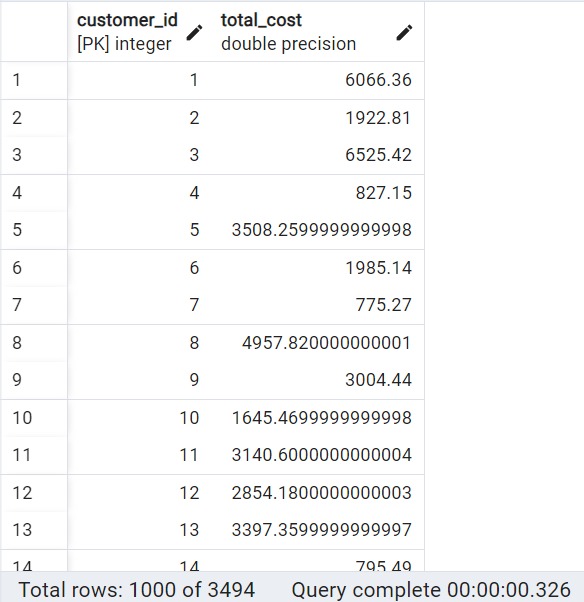
Description automatically generated

**Inserción de GTPC en PostgreSQL**

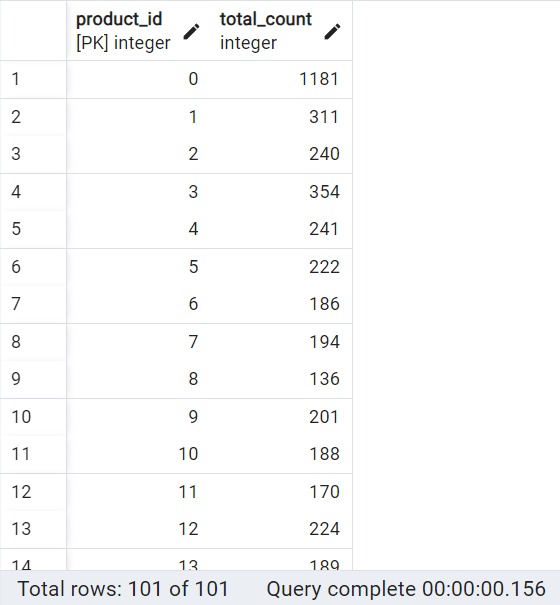
* Similar al caso anterior

6. Resultados

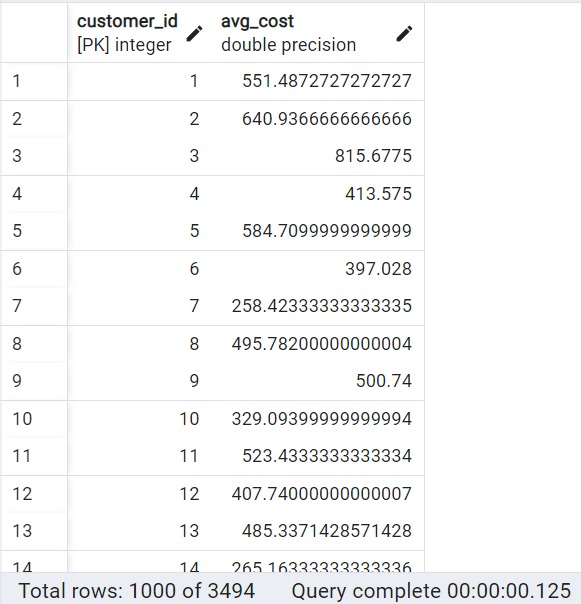
6.1 Gasto Total por Cliente (GTPC) / total\_spent\_per\_customer



6.2 Productos más Comprados (PMC) / product\_purchase\_count



6.3 Promedio de Gasto por Cliente (PGPC) / average\_spent\_per\_customer



6.4 Frecuencia de Compra por Cliente / transaction\_count\_per\_customer

