Sección 1: Data pipeline

Objetivo: Crear un data pipeline con las herramientas disponibles por el usuario

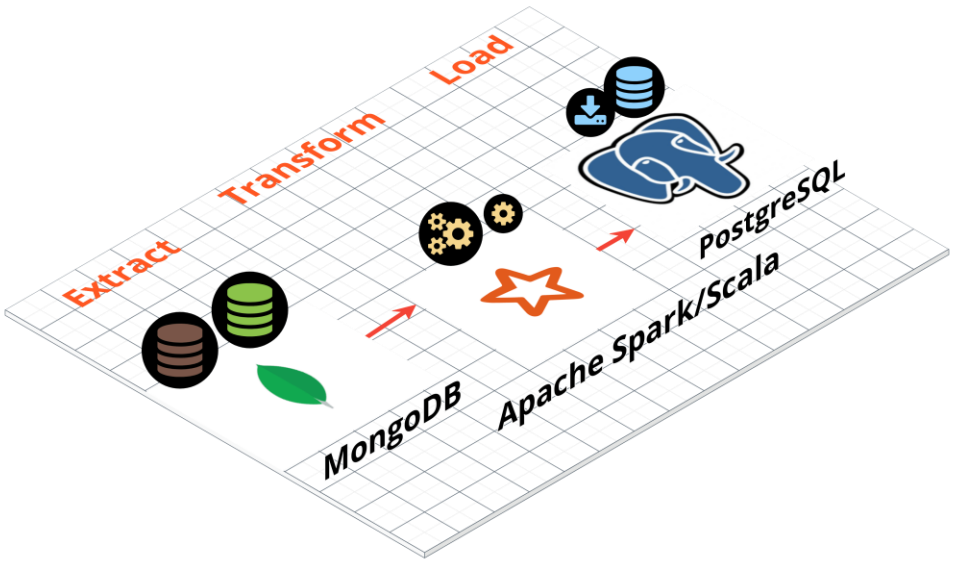
Los ejercicios de programación tienen que incluir los procedimientos de instalación y ejecución de las herramientas a utilizar y los scripts que realizaran los procedimientos. Se puede realizar a través de Dockers. Pueden incluir pruebas unitarias o de integración. Se puede compartir por GitHub o cualquier repositorio o en un zip.

Nota: Junto con esta guía te compartimos 1 data set con la información sobre las compras de dos compañías ficticias que procesan con nosotros.

1.1 Carga de información

La información proporcionada se debe de cargar en alguna base de datos. Puede ser estructurada o no estructurada. Ejemplo: MySQL, Postgres, MongoDB, etc. Incluye comentarios del por qué elegiste ese tipo de base de datos.

- De una forma general he decidido utilizar Mongo DB para la carga de información, Apache Spark/Scala para las transformaciones necesarias y finalmente la persistencia en Postgres tal como lo han pedido.



- Para la carga de información utilicé Mongo DB ya que proporciona la integración por medio de un conector con Spark, desde mi punto de vista me beneficia usar MongoDB siendo NoSQL, ya que podemos persistir información no estructurada en caso de que existiera dentro del dataset que me han proporcionado.

- Para importar el CSV proporcionado a MongoDB, utilicé el siguiente comando:

```
mongoimport -d test -c testc --type csv --file data_prueba_tecnica.csv --headerline --ignoreBlanks
```

1.2 Extracción

Se debe de realizar un procedimiento de extracción de la información anterior por medio de algún lenguaje de programación que permita procesarlo. El formato final de la información extraída puede ser CSV, Avro, Parquet o el que se considere más adecuado. Agrega comentarios acerca del por qué tuviste que utilizar el lenguaje y el formato que elegiste. También platícanos si te encontraste con algún reto a la hora de extraer la información.

- Para la extracción, utilicé Apache Spark/Scala ya que es un potente motor de procesamiento diseñado para brindar velocidad y facilidad de uso, Spark sobresale particularmente cuando se requiere un rendimiento rápido. MongoDB es una popular base de datos NoSQL y es confiable para realizar este tipo de procesos, considero que la integración de estas dos tecnologías de Big Data también ahorra tiempo y costo computacional ya que se usa una sola tecnología de base de datos.

- Considerando que eran pocos registros, Spark me permite persistir de manera local y decidí hacerlo en un formato Parquet, ya que es el formato más comprimido a comparación de un Avro o un CSV, si el dataset hubiera tenido un número considerable de registros y columnas, por buenas prácticas sólo se persiste de forma local una muestra no mayor a 10,000 registros, esto para evitar que colapse MongoDB o Spark (localmente)
  Cuando el proceso de extracción se realiza en producción no se debería limitar la cantidad de registros a extraer.

- De los retos que me encontré al extraer la información fue la alineación de versiones entre Spark y MongoDB por el conector.
- También fue eliminar el campo "_id" generado automáticamente por MongoDB y eliminé todos aquellos "id" nulos, dado que el esquema en postgres está definido como NOT NULL.

1.3 Transformación

Se propone el siguiente esquema para la información.

Cargo
id varchar(24 NOT NULL
company_name varchar(130) NULL
company_id varchar(24) NOT NULL
amount decimal(16,2) NOT NULL
status marchar(30) NOT NULL
created_at timestamp NOT NULL
updated_at timestamp NULL

Realiza las transformaciones necesarias para que la información extraída cumpla con el esquema. Puedes realizarlas con el lenguaje de programación de tu preferencia. Incluye comentarios acerca de que transformaciones tuviste que realizar y que retos te encontraste en la implementación de estos mecanismos de transformación.

- Para el punto de las transformaciones he utilizado Spark/Scala, dado que el proceso requería análisis y transformaciones en lugar de modelos predictivos, además tanto Spark como Scala están escritos en Java lo que los hace más compatibles de forma nativa y por ende un poco más rápidos a comparación de PySpark.

Las transformaciones interesantes con las que me encontré, fueron las siguientes:

- Una vez visualizada y analizada la información se requirieron de varias transformaciones de tipo condicional en los campos **company_name** y **company_id**, ya que me enfrenté a la problemática de que en estos campos había valores no válidos, que pude resolver mediante condicionales y así obtener sólo las dos compañías ficticias que se mencionaron (lo cual yo tomé como regla de negocio).

- El campo amount, tenía datos de tipo float pero también tenía datos en notación científica, por lo que tuve que transformar estos valores a tipo decimal(16,2) sin embargo eran valores muy altos los cuales excedían la precisión de 16 dígitos, por lo que todos aquellos valores que excedían ese valor, coloqué el valor más alto que podía tener (99999999999999.99)

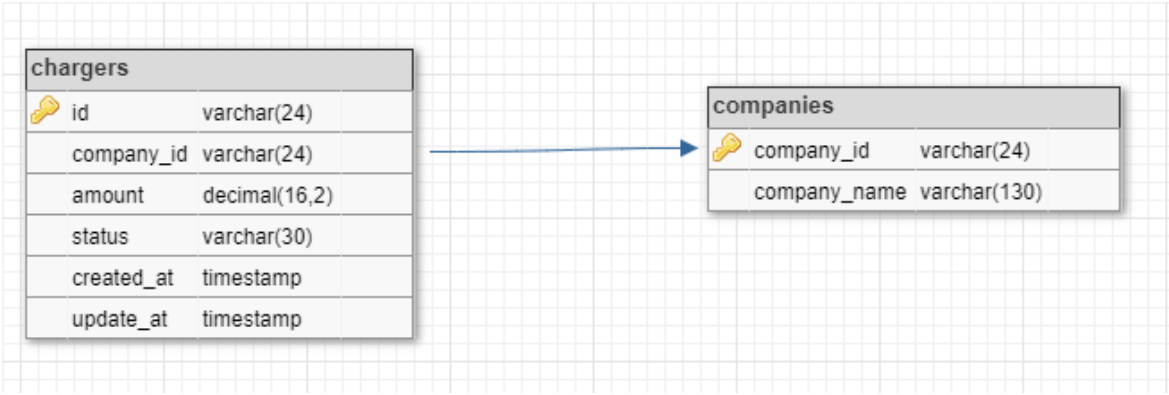
- Se agregaron Test Unitarios en cada uno de los métodos.

1.4 Dispersión de la información

Se debe de utilizar una base de datos Postgres. En esta base se va a crear un esquema estructurado basado en el ejercicio anterior, pero debemos de crear una tabla llamada charges donde tendremos la información de las transacciones y otra llamada companies donde incluiremos la información de las compañías. Estas tablas deberán de estar relacionadas.

Cargaremos la información del dataset en estas dos tablas. Incluye el diagrama de base de datos resultado de este ejercicio.

- Diagrama BD



La razón por a cuál he decidido que ambas tablas se relacionen con el campo **company_id** y no **id,** es porque sólo tenemos dos compañías diferentes existentes y un total de 9997 id's diferentes, si yo relacionaba las tablas con el id iba a tener esos 9997 registros en cada tabla y al momento de realizar la vista esto sería más costoso**.**

1.5 SQL

Diseña una vista en la base de datos Postgres de las tablas donde cargamos la información transformada para que podamos ver el monto total transaccionado por día para las diferentes compañías.

El siguiente script genera la vista solicitada:

```
CREATE VIEW total_amount AS

SELECT companies.company_id, companies.company_name, charges.created_at,
SUM(charges.amount)

FROM companies

INNER JOIN charges

ON charges.company_id = companies.company_id

GROUP BY companies.company_id, companies.company_name, charges.created_at;
```