1.1. Arquitectura Medallion

Describa detalladamente los niveles de la arquitectura Medallion en Databricks (Bronze, Silver, Gold), explicando:

Para este tipo de arquitectura por capas empezaremos por la capa bronce donde el propósito es almacenar los datos como llegaron, es decir desde su fuente origen (como archivos csv, json, etc) sin ser procesados con la diferencia que su formato cambia a Delta.

Como segunda capa tenemos la Silver que tiene como función la limpieza y estandarización de los datos es decir establecer los tipos de datos de ser necesario, regulación de nulos y registros duplicados.

Por último tenemos la capa Gold que su propósito principal es facilitar los datos previamente procesados para reportaría y análisis de negocio las transformaciones típicas en esta capa son agregaciones, cálculo de métricas y modelo en formato OLAP.

Esta arquitectura Medallion a diferencia de una tradicional tiene como ventaja en tratamiento de los datos por capas en todas sus capas cuenta con almacena distribuido en Delta lake el cual ofrece mayor rapidez en lectura y escritura.

1.2. Optimización en Spark

Explique las siguientes técnicas de optimización en Spark y cuándo usaría cada una:

a) **Particionamiento vs Bucketing**

Particionamiento: divide los datos entre múltiples archivos en función a una columna especifica por ejemplo cuando se requiere realizar consultas y se debe filtrar por unos de estos campos (fecha, país, etc.).

>> df.write.format("parquet").partitionBy("year")...

Bucketing: distribuye los datos en un numero especifico de buckets regulamente usado para el uso de joins y group by

>> df.write.format("parquet").bucketBy(10, "id_Regestro")...

b) **Cache vs Persist**

Cache: guardar los datos en memoria RAM, una vez cargado por primera vez su posterior consuntas son más rápidas usado para dataset pequeños o medianos.

>>df.cache()

Persist: paracido a cahce, pero este se pueda almacenarse en RAM o en disco comúnmente usado para datos demasiados grandes que no lleguen a caber en RAM.

c) **Broadcasting**

Envia una tabla pequeña a todos los nodos este sirve para evitar el shuffle en joins o group by en otras palabras evita la redistribución de los datos entre nodos.

d) **Repartitioning**

Registribuye los datos en un numero específico de particiones usado para balancear la carga en los nodos con la finalidad de mejorar el procesamiento de un dataset

>> df = df.repartition(8)

1.3. Delta Lake

¿Qué ventajas ofrece Delta Lake sobre formatos tradicionales como Parquet? Explique conceptos como:

A grandes rasgos se puede decir que dalta lake es una especie de parquet evolucionar ya que se forma de un formato parquet con las que se añade una serie de características como control de versiones, mantiene las transacciones ACID, manejo de datos dinámicos y una optimización de consultas.

1. Time travel nos permite consultas versiones antiguas de los datos

   >> df = spark.read.format("delta").option("versionAsOf", 5)...

2. Schema Enforcement y Evolution: Evita datos que no corresponde al esquema definido por ejemplo si se definió un tipo de datos float y se quiere insertar un string.
3. Transacciones ACID: data dake soporte ACID (Atomicidad, Consistencia, Aislamiento, Durabilidad) esto evita datos corruptos en el procesamiento.
4. Compactación y Z-Ordering:
   La compactación recude la cantidad de archivos pequeños a fin de mejorar la lectura
   >> OPTIMIZE delta.'dbfs:/mnt/data/archivo'

   Y el Z-Ordering: orderna los datos en disco para acelerar las consultas.
   >> OPTIMIZE delta.'dbfs:/mnt/data/archivo' ZORDER BY ('archivo_id')

1.4. Modelado Dimensional

Compare y contraste los esquemas Estrella y Copo de Nieve. ¿Cuándo emplearía cada uno en un entorno bancario y por qué?

El esquema de estrella se centra en una table de hecho rodeada de varias tablas de dimensiones las cuales se encuentras desnormalizadas, tiene una fácil implementación y manteamiento, pero tiene un mayor almacenamiento por la cualidad antes mencionada.

El esquema de copo de nieve es liminar al de estrella no obstante cuenta con subdimensiones para reducir la redundancia y el espacio de almacenamiento en las tablas de dimensiones principales, una de sus desventajas es que sus consultas son más lentas debida a la necesitada de múltiples joins entre tablas y una mayor complejidad en su mantenimiento.

El usado para el entorno bancario va a depender directamente de las necesidades de cada entidad; ventajas de esquema estrella: consultas rápidas y análisis, fácil de implementar y mantener, mejor para grandes volúmenes de datos, por otra parte esquema de copo de nieve ofrece menor almacenamiento por la normalización y mas eficiente en consultas con uniones mas complejas.

1.5. Jobs y Workflows en Databricks

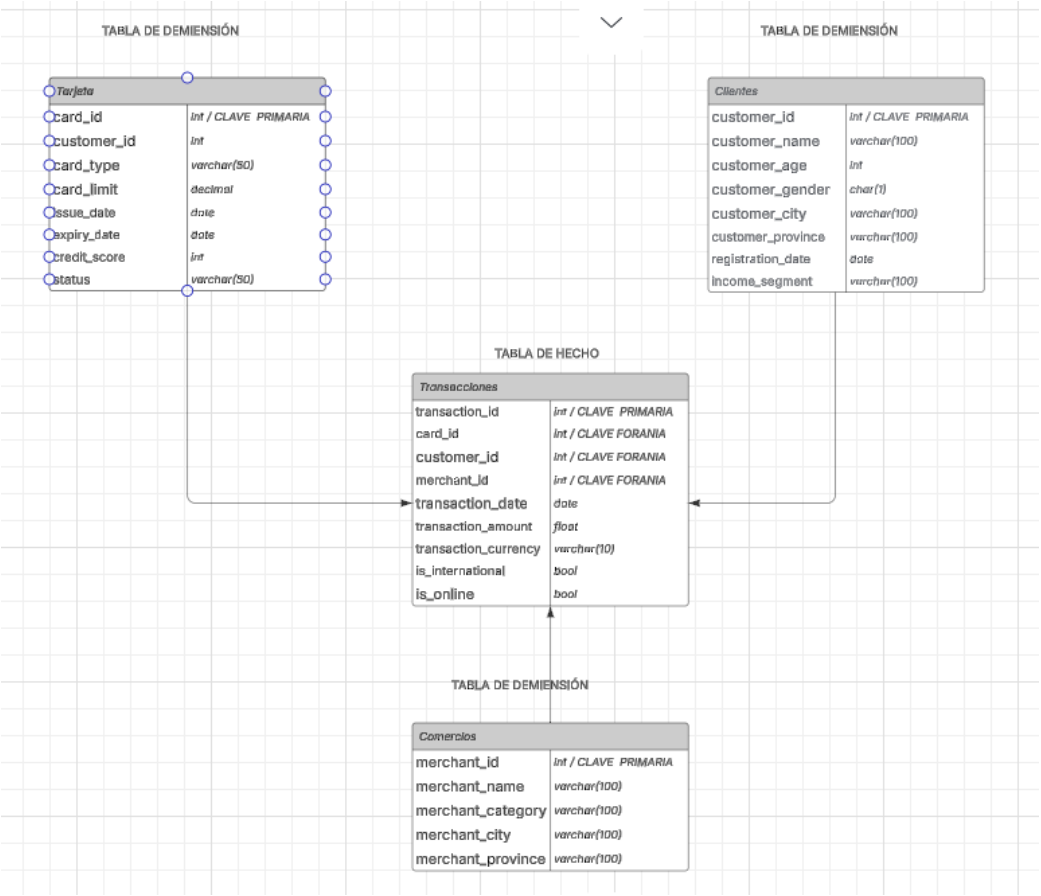
Explique cómo implementaría un workflow con dependencias en Databricks, las mejores prácticas para programar jobs, y cómo manejaría fallos y reintentos.

Un workflows son una seria de Jobs que se orquestar y tiene dependencia entre ellos, es decir su ejecución es secuencial o paralela, estos pueden ser notebooks o script y pueden ser programados con una frecuencia determinada, es necesario configurar sus dependencias entre los diferentes Jobs para obtener el flujo esperado. Como buenas practicas deberia optimizar clúster por ejemplo que se apague cuando estos Jobs culminen y configurar un tiempo de inactividad, configurar alertas y notificaciones en el caso de que unos de los Jobs no se haya ejecutado o presente problemas en su ejecución.

Parte 2: Caso práctico de modelado de datos (30%)

Contexto

El banco está diseñando un nueva data mart para analizar el comportamiento de uso de tarjetas de crédito. Se busca segmentar clientes para campañas personalizadas y detectar patrones de gasto que permitan ofrecer productos adecuados a cada perfil.



Análisis del modelo, se escogió el modelo estrella a pesar de que existe una relación entre tarjeta y cliente, pero para ese caso es practico dicha relación ya que un cliente puede tener mas de una tarjeta y a su vez nos ayuda a optimizar las consultas.

Granularidad y estrategia de actualización:

Hechos: para transacción la granularidad es a nivel de transacciones mismo y su nivel de actualización debería ser de tipo 1, registrar sola la ultima consulta.

Dimensiones:

Para Cliente la granularidad es a nivel de cliente y su nivel de actualización debería ser tipo 2 tener un historial de los cambios como edad, dirección o cuidad.

Para Tarjeta la granularidad es a nivel de Tarjetas y su nivel de actualización debería ser tipo 2 tener un historial de los cambios como límite de crédito y estado de la tarjeta.

Para Comercio la granularidad es a nivel de comercio mismo y su nivel de actualización debería ser de tipo 1, solo actualizar la información si cambia, como el nombre o la categoría.

Explique cómo implementaría este modelo usando la arquitectura Medallion:

Capa 1: Bronce.

En este nivel contiene los datos en crudo es decir se toma la información de las diferentes fuentes y se las almacena si ser procesada.

Estructura de Tablas en el Nivel Bronze:

- **Tabla de Transacciones**

  transaction_id (INT)

  card_id (INT)

  merchant_id (INT)

  transaction_date (DATE)

  transaction_amount (FLOAT)

  transaction_currency (VARCHAR)

  is_international (BOOLEAN)

  is_online (BOOLEAN)

- **Tabla de Clientes**

  customer_id (INT)

  customer_name (VARCHAR)

  customer_age (INT)

  customer_gender (CHAT)

  customer_address (VARCHAR)

customer_city (VARCHAR)

customer_province (VARCHAR)

registration_date (DATE)

income_segment (VARCHAR)

- **Tabla de Tarjetas**

  card_id (INT)

  customer_id (INT)

  card_type (VARCHAR)

  card_limit (DECIMAL)

  issue_date (DATE)

  expiry_date (DATE)

  credit_score (INT)

  status (VARCHAR)

- **Tabla de Comercios**

  merchant_id (INT)

  merchant_name (VARCHAR)

  merchant_category (VARCHAR)

  merchant_city (VARCHAR)

  merchant_province (VARCHAR)

Frecuencia de Actualización en el Nivel Bronze:

Transacciones: Diaria (500,000 registros/día).

Clientes: Semanal (2,000,000 registros/semanales).

Tarjetas: Diaria (3,000,000 registros/día).

Comercios: Mensual (100,000 registros/mensuales).

Estrategias de Particionamiento en el Nivel Bronze:

Transacciones: Se puede particionar por transaction_date para mejorar el rendimiento en consultas basadas en fechas.

Clientes: No es necesario particionar, pero se podría hacerlo por customer_id si se requiere acceso eficiente a clientes específicos.

Tarjetas: Particionar por card_id para facilitar las consultas relacionadas con tarjetas.

Comercios: No es necesario particionar, pero se podría hacer por merchant_id si se busca optimizar consultas de comercios específicos.

CAPA 2: SILVER

En este nivel se realiza la limpieza de datos, se valida registros duplicados y registros nulos, y por último podemos enriquecer los datos creando columna con información relevante.

En la table de hechos se puede agregar por el ejemplo en customer_id del cliente y el status de la tarjeta y de ser necesario realizar el cambio de algún tipo de datos según corresponda el análisis.

CAPA 3: GOLD

Estos datos se encuentran lista para el análisis y la generación de informes, trasformaciones comunes en este nivel: agregaciones y calculo de métricas.

Parte 3: Implementación de pipeline de datos con PySpark (50%)

Se adjunta enlace para la revisión de la implementación:

https://github.com/rjbl16/prueba_tecnica_bb