SOLUCIÓN PRUEBA TÉCNICA INGENIERO DE DATOS – PUNTORED

Sección 1: Preguntas teóricas

Python

1. Las listas y los conjuntos son estructuras de datos, los cuales, almacenan colecciones de elementos. Sus diferencias son:

   List o listas: Mantienen el orden de los elementos, estas permiten tener elementos duplicados. Los elementos se pueden acceder mediante índices. Las listas son estructuras de datos mutables.

   Set o Conjuntos: Estos no garantizan un orden especifico y estos no permiten elementos duplicados. El conjunto es mutable, pero solo permite agregar o eliminar elementos del mismo.

   Ejemplo ubicado en el archivo ejemplos_seccion_1/ejemplos_sección_1

2. Generator es una función de Python que produce una secuencia de valores, la cual, los genera bajo demanda y no los almacena en memoria. Esto se crea con la palabra clave yield en lugar del return en la función, lo que pausa la ejecución de la función y reanudarla en la siguiente invocación de la función.

   Es útil usarla cuando trabajamos con grandes volúmenes de datos que no caben en memoria, cuando se necesita una iteración sin construir una lista.

   Ejemplo ubicado en el archivo ejemplos_seccion_1/ejemplos_sección_1

3. Pandas es una librería de Python, la cual, está diseñada para la manipulación, limpieza y análisis de datos. Python tiene estructuras de datos nativas como lo son las listas, diccionarios, entre otros. Pandas nos ofrece herramientas mucho más optimizadas y eficientes para manejar volúmenes de datos muy grandes.

   Ventajas de Pandas:

   - Nos ofrece manejo de datos faltantes como .fillna() o .dropna(), lo cual se facilita este manejo.
   - Estructura de datos DataFrame, este organiza los datos en formato tabular similar a Excel.
   - Operaciones sobre fechas con pd.to_datetime()
   - Métodos rápidos como .groupby(), .agg(), entre otros

Ejemplo ubicado en el archivo ejemplos_seccion_1/ejemplos_sección_1

4. Apply() y map() son funciones para aplicar a los datos en una estructura DataFrame o una Serie. Sus diferencias son:

    - Map() se aplica a una serie o una única columna, apply() se aplica o se usa en una única columna y a todo el DataFrame.
    - Map() aplica la función elemento por elemento de una única columna y apply aplica las funciones sobre las filas y columnas completas de un DataFrame.
    - Map() acepta funciones complejas, pero esto solo sobre valores individuales y apply() puede operar funciones complejas sobre filas y columnas completas.

    Ejemplo ubicado en el archivo ejemplos_seccion_1/ejemplos_sección_1

SQL

1. La consulta para el esquema dado usando JOIN y GROUP BY podría ser de la siguiente manera:

    ```
    SELECT d.nombre AS nombre_departamento, SUM(e.salario) AS salario_total
    FROM empleados e
    JOIN departamentos d ON e.departamento_id = d.id
    GROUP BY d.nombre
    ORDER BY salario_total DESC
    ```

2. Los JOIN o cada tipo de JOIN define cómo se combinan las filas de dos tablas según la condición.

    - INNER JOIN solo combina las filas con coincidencias en ambas tablas o si vemos como conjuntos las tablas sería la intersección de estos.

    - LEFT JOIN toma todas las filas de la tabla de la izquierda, con las
    - coincidencias en la tabla de la derecha o un NULL si no hay coincidencia.

    - FULL JOIN toma las filas de ambas tablas, con un NULL en los datos sin coincidencia o viéndolo desde conjuntos la unión de estos.

    Para los ejemplos tomamos la tabla empleados y departamentos del punto anterior.

INNER JOIN

SELECT e.nombre AS empleado, d.nombre AS departamento

FROM empleados e

INNER JOIN departamentos d ON e.departamento_id = d.id

Esta consulta empleados con departamento_id válido en la tabla departamento.

LEFT JOIN

SELECT e.nombre AS empleado, d.nombre AS departamento

FROM empleados e

LEFT JOIN departamentos_id ON e.departamento_id = d.id

Está consulta muestra todos los empleados, incluso si no tienen un departamento asignado mostrando un NULL en estos casos.

FULL JOIN

SELECT e.nombre AS empleado, d.nombre AS departamento

FROM empleados e

FULL JOIN departamentos d ON e.departamento_id =d.id

Está consulta muestra o incluye todos los empleados y todos los departamentos, en el caso que no tengan departamento asignado aparecerán con un NULL en departamento.

3. Al trabajar con bases de datos a gran escala o con millones de registros las consultas pueden volverse lentas si estas no están optimizadas, mi estrategia para mejorar el rendimiento de la consulta es la siguiente:

    - Usar índices para acelerar la búsqueda de datos, en especial en columnas usadas son WHERE, con algún JOIN y ORDER BY.
    - Evito el SELECT * y solo selecciono los datos que necesito.
    - USAR la sentencia EXPLAIN ANALYZE para entender como el motor de base de datos está ejecutando la consulta y cuando tarda esta.
    - Optimizar los JOIN usando EXIST en lugar de IN cuando esto sea posible, ya que en listas grandes es mejor usar EXIST.

- Usar particionamiento de tablas, es decir, dividir la tabla en partes más pequeñas para así acelerar las consultas sobre los datos específicos.

4. Diferencia entre el HAVING y el WHERE en SQL

    - WHERE filtra los registros antes de la agrupación GROUP BY y está no funciona con agregaciones como SUM, COUNT, entre otras, diciendo así que se usa WHERE para filtrar antes de realizar la agrupación sobre columnas individuales.

    - HAVING filtra después de la agrupación GROUP BY y esta funciona con agrupaciones SUM, COUNT y demás, es decir que, usamos HAVING para filtrar después de agrupar sobre los resultados de agregaciones.

Amazon Web Services (AWS)

1. La diferencia entre los servicios de AWS Amazon S3, Amazon RDS y Amazon Redshift, es el propósito ya que este es diferente y podemos diferenciarlo así:

    - Amazon S3: Este tiene el tipo de almacenamiento de objetos y su uso principal es guardar archivos y datos no estructurados.

    - Amazon RDS: Su tipo de almacenamiento es una base de datos relacional y su uso principal es para aplicaciones transaccionales con SQL.

    - Amazon Redshift: este es un DataWarehouse y su uso principal es para analizar o realizar análisis de datos a gran escala.

2. Los servicios Amazon DynamoDB, Amazon RDS y Amazon Redshift tienen propósitos diferentes y se usan según los requerimientos y particularidades del negocio. A continuación, describo brevemente cuando usaría cada servicio.

    - Usar **Amazon DynamoDB:** Cuando se tengan altas tasas de lectura y escritura ya que cuenta con escalabilidad automática, esta tiene una baja latencia en tiempo real en cada consulta. Cuando los datos no son estructurados o estos son semiestructurados por ejemplo JSON o datos clave-valor y también cuando las aplicaciones no tienen un esquema rígido o NoSQL. También lo usaría para datos en eventos de tiempo real o streaming de datos.

    - Usar **Amazon RDS**: Este es un servicio de base de datos relacional o SQL, se usaría cuando se necesitan bases de datos relacionales o con

una estructura fija como tablas, relaciones, llave primaria y foránea; cuando se requieren consultas SQL eficientes. También cuando se tiene aplicaciones transaccionales con procesos de lectura y escritura frecuente en tiempo real y cuando se requiere que la solución sea gestionada y escalable, es decir, cuando se quiere que el servicio se encargue de copias de seguridad, monitoreo y escalabilidad. Cabe aclarar que Amazon RDS soporta los motores MySQL, PostgreSQL, MariaDB, SQL server, Oracle y Aurora.

- Usar **Amazon Redshift**: Este es una bodega de datos o un Data Warehouse diseñado para ejecutar consultas analíticas. Se usaría cuando se necesite analizar grandes volúmenes de datos (Big Data) de manera eficiente siendo esta de millones o billones de registros, Se usaría también cuando se usan consultas analíticas y BI para agregaciones, tendencias, dashboards y este tiene integración con Tableau, Power BI y AWS QuickSight. Se usaría cuando se necesita consultas SQL rápidas en datos históricos e4n bases relacionales y optimización de Joins para análisis de series temporales y se usaría cuando se quieren almacenar datos optimizados para su lectura.

3. Las diferencias entre **AWS Lambda** y **AWS EC2** son las siguientes:

    - El tipo de computación en AWS Lambda es Serverless o sin servidor a comparación de AWS EC2 que es con un servidor virtual.

    - La escalabilidad es automática en AWS Lambda y en AWS EC2 es manual o basada en Auto Scaling.

    - El tiempo de ejecución en AWS Lambda es limitado a q5 minutos por cada invocación y en AWS EC2 es ilimitada.

    - EL uso de AWS Lambda es bajo ejecución de código bajo demanda, eventos y microservicios; y AWS EC2 es para aplicaciones web, bases de datos o cargas de trabajo persistentes.

    - La gestión del servidor en AWS Lambda AWS lo gestiona todo y en AWS EC2 el usuario gestiona OS, actualizaciones y seguridad.

4. Un mecanismo seguro para acceder a un bucket de S3 sin claves de acceso en el código y recomendado como mejores prácticas en AWS, es usar IAM Roles en lugar de una credencial estática como Access Key o Secret Key, para así permitir que un servicio acceda a S3 ya que esto mejora la seguridad y facilita la gestión de permisos.

Sección 2: Prueba Práctica de SQL

- Escribe una consulta para obtener los 5 clientes con mayor monto total en ventas en los últimos 6 meses.

  La consulta se encuentra en consultas_seccion_2/consulta_1

- Escribe una consulta para calcular el ticket promedio de ventas por cliente en el último año.

  La consulta se encuentra en consultas_seccion_2/consulta_2

- Escribe una consulta para obtener el nombre completo de los clientes y su monto total de ventas.

  La consulta se encuentra en consultas_seccion_2/consulta_3

- Escribe una consulta para obtener el ingreso promedio de ventas por mes.

  La consulta se encuentra en consultas_seccion_2/consulta_4

- Escribe una consulta para calcular el ranking de clientes por ventas en el último año.

  La consulta se encuentra en consultas_seccion_2/consulta_5

- Escribe una consulta para calcular el total de ventas por cliente y luego selecciona solo los clientes cuyo total de ventas sea superior al promedio general.

  La consulta se encuentra en consultas_seccion_2/consulta_6

Sección 3: Prueba práctica Python y AWS

1. Diseñe un pipeline para extraer los datos de la base y disponibilizar la información que requiere cada proveedor en batch diario usando los servicios de AWS.

   Arquitectura

   - AWS Lambda trigger, la cual, se programa una función Lambda con EventBrigde para ejecutar el pipeline diariamente.
   - AWS Glue para ETL:
     - Extrae los datos desde Amazon RDS ya sea con motor MySQL o PostgreSQL.
     - Transforma los datos agrupándolos por cliente y fecha.

        - Generación de archivos en Amazon S3 organizados por proveedor.
    - Amazon S3 Almacena los archivos en formato JSON o Parquet según la necesidad.
    - Amazon API Gateway más AWS Lambda, lo cual, permite a los proveedores consultar los archivos desde S3.

Diagrama

Ejemplo de consulta SQL para AWS Glue (Batch)

Consulta ubicada en Python_seccion_3/consulta_1

2. Diseñe un pipeline para extraer los datos de la base y disponibilizar la información que requier cada proveedor en tiempo real usando los servicios de AWS.

   Arquitectura
    - Amazon RDS » AWS DMS este para poder capturar los cambios en la base de datos.
    - AMAZON Kinesis Data Streams:
        - Este recibe los eventos de las ventas en tiempo real.
        - Segmenta los datos según el producto.
    - AWS Lambda:
        - Se agrega para transacciones por cliente y fecha.
        - Escribe los datos en Amazon DynamoDB para consultar rápidamente.
    - Amazon API Gateway + Laambda
        - Le permite a los proveedores consultar la información en tiempo real.

Diagrama

3. Script en Python para extraer la información de la base de datos en el formato que requiere el proveedor.

   Script ubicado en python_seccion_3/script_1.py

4. Script en Python para disponibilizar la información en el API en formato JSON, implementar logs y manejo de errores.

   Script ubicado en python_seccion_3/script_2.py

Preguntas rápidas:

- ¿Windows o Linux?: He sido usuario de Windows desde hace muchos años por lo que me siento más cómodo como usuario de él, pero he tenido acercamientos de Linux y a su consola lo que me permite estar abierto a ser usuario de Linux.

- ¿MySQL o PostgreSQL?: Elijo MySQL porque la he usado más, pero el preferir una u otra dependería de los requerimientos y necesidades del negocio para ver cuál de ellas nos beneficia más para este.

- ¿Batch processing o streaming?: Streaming la elijo cuando es necesario tener respuestas rápidas o se necesita procesamiento de datos en tiempo real y Batch processing cuando el procesamiento es necesario por lotes o por tiempos o frecuencias especificas por ejemplo una vez al mes de las ventas por agente comercial para conocer la venta total del mes y comparar con sus metas mensuales.

- ¿ETL o ELT?: Para un alto grado de limpieza y estandarización o con una estructura bien definida de los datos elijo ETL, pero para grandes volúmenes de datos sin estructura definida es mejor el ELT

- ¿Parquet o CSV?: Parquet porque este es muy rápido y eficaz en consultas en grandes volúmenes de datos o Big Data, es menos pesado o necesita menor capacidad de almacenamiento y lo más importante es que los tipos de datos están bien definidos. CSV lo elegiría cuando debo leer esto en una herramienta como lo es Excel para pre visualizar los datos o tener un tratamiento en los datos de un paquete en específico y este es legible casi que por todas las herramientas o compartirlos fácilmente.

- ¿Spark o Pandas?: Spark para grandes volúmenes de datos ya que el procesamiento de este es distribuido, pero si son volúmenes pequeños o quizás medianos usaría Pandas ya que se procesan en la CPU y RAM disponible del equipo que se dispone.