Grupo IskayPet®

The best for animals, the best for all

PRUEBA TÉCNICA PUESTO DATA ENGINEER

REQUISITOS PREVIOS

-Tener docker instalado
-Tener python instalado

PARTE 1: PYSPARK Y SQL

Descripción General

Esta primera parte tiene como objetivo evaluar tus habilidades en el uso de PySpark para manipular y analizar grandes volúmenes de datos. Para esta prueba, dispondrás de tres datasets en formato CSV que contienen información estadística sobre videos de YouTube para los países Canadá, Estados Unidos y México, **es importante tener en cuenta en los ejercicios que los datasets contienen más de una fila con estadísticas de un mismo vídeo ya que se pueden presentar estadísticas de un mismo vídeo en diferentes momentos del tiempo**.

LOS EJERCICIOS DEL 1-6 SE TIENEN QUE DESARROLLAR CON LOS MÉTODOS PROPIOS DE SPARK DATAFRAME Y LOS EJERCICIOS 7 y 8 SE TIENEN QUE DESARROLLAR MEDIANTE SPARK SQL

Este es el esquema de los datos que se van a utilizar:

1. `video_id` : Id del vídeo
2. `trending_date` : Fecha de registro de estadísticas
3. `title` : Título del vídeo
4. `channel_title` : Nombre del canal de Youtube
5. `category_id` : Id de la categoría del vídeo
6. `publish_time` : Fecha de publicación del vídeo
7. `tags` : Etiquetas del vídeo
8. `views` : Vistas del vídeo en la fecha 'trending_date'
9. `likes` : Likes del video en la fecha 'trending_date'
10. `dislikes` : Dislikes del video en la fecha 'trending_date'
11. `comment_count` : Número de comentarios en la fecha 'trending_date'
12. `thumbnail_link` : Link a la miniatura del vídeo
13. `comments_disabled` : Comentarios desactivados
14. `ratings_disabled` : Likes/Dislikes desactivados
15. `video_error_or_removed` : Video borrado o con error en subida
16. `description` : Descripción del vídeo

La prueba se desarrollará en un entorno de Docker con Spark configurado en modo local.

Estructura de la Carpeta

- `prueba-tecnica/`
  - `parte1/`
    - `data/`
      - `CAvideos.csv`
      - `USvideos.csv`
      - `MXvideos.csv`
    - `spark.py` (archivo donde desarrollarás los ejercicios)
    - `docker-compose.yaml` (archivo para levantar el entorno de spark con Docker)

Preparación de entorno Spark en local

1. Ejecuta el comando change directory para moverte a la carpeta parte1

   cd prueba-tecnica\parte1

2. Levanta el entorno docker con el siguiente comando:

   docker-compose up –d

3. Una vez terminado de levantar el cluster de Spark copia los ficheros de la carpeta parte1 al nodo maestro con el siguiente comando:

   docker cp -L . parte1-spark-master-1:/opt/bitnami/spark/prueba-tecnica/

4. Una vez copiados los ficheros necesarios para trabajar realiza el ejercicio 1 y ejecuta los siguientes comandos para comprobar el correcto funcionamiento de Pyspark en local:

   docker cp -L spark.py parte1-spark-master-1:/opt/bitnami/spark/prueba-tecnica/

   docker exec parte1-spark-master-1 spark-submit ./prueba-tecnica/spark.py

Enunciado de la Prueba Técnica

Ejercicio 1: Configuración de Spark

1 **Inicializa una sesión de Spark** llamada "prueba-tecnica", configura spark con la siguiente configuración `"spark.sql.legacy.timeParserPolicy"` con el valor `"LEGACY"`

2 Establece el nivel de logs en "ERROR".

Ejercicio 2: Carga y Conversión de Datos (UTILIZAR SPARK DATAFRAME)

3 **Carga los datasets CSV** correspondientes a Canadá, Estados Unidos y México en tres DataFrames diferentes (`df_ca`, `df_us`, `df_mx`).
4 **Convierte las columnas** `views` y `likes` de tipo `string` a `bigint` para los tres DataFrames.
5 **Renombra las columnas** `trending_date` a `statistics_date` y `publish_time` a `publish_date`.
6 **Convierte las columnas de fecha** `statistics_date` y `publish_date` al formato `yyyy-MM-dd`.
7 **Imprime el esquema** de los tres dataframes para comprobar los cambios realizados.

Ejercicio 3: Análisis de Valores Nulos (UTILIZAR SPARK DATAFRAME)

8 **Calcula el porcentaje de valores nulos** para cada columna en los tres DataFrames. Determina cuál de los tres datasets tiene el mayor porcentaje de valores nulos.

Ejercicio 4: Relleno de Valores Nulos (UTILIZAR SPARK DATAFRAME)

9 **Rellena los valores nulos** en los DataFrames utilizando las siguientes reglas:
   - Columnas numéricas: rellena con `0`.
   - Columnas de tipo `string`: rellena con `'default'`.
   - Columnas de `date`: rellena con `'9999-99-99'`.

10 **Recalcula el porcentaje de valores nulos** para asegurar que no quedan valores nulos en los DataFrames.

Ejercicio 5: Análisis Temporal (UTILIZAR SPARK DATAFRAME)

11 **Determina el año con más videos subidos** para cada país (Canadá, Estados Unidos y México) basado en la columna `publish_date`, ten en cuenta que pueden aparecer filas con el campo `video_id` repetidos.

Ejercicio 6: Análisis de Visitas Mensuales (UTILIZAR SPARK DATAFRAME)

12 **Identifica el mes con más visitas** en el año 2017 para los videos de México, ten en cuenta que pueden aparecer filas con el campo `video_id` repetidos y por consecuencia un vídeo puede tener varias filas con diferentes valores en el campo

`statistics_date` sólo contempla el registro que tenga mayor antigüedad en este campo

Ejercicio 7: Análisis de Audiencia Compartida (UTILIZAR SPARK SQL)

13 **Encuentra los videos con audiencia compartida** entre los tres países. Un video se considera compartido si aparece en los datasets de Canadá, Estados Unidos y México. Calcula el número total de videos con audiencia compartida. Ten en cuenta que pueden aparecer filas con el campo `video_id` repetidos.

Ejercicio 8: Análisis Comparativo (UTILIZAR SPARK SQL)

14 **Determina el video compartido entre los 3 países más visto y el video con más likes en Estados Unidos**. Usa las tablas resultantes del ejercicio anterior.

Entrega

- Archivo `spark.py que contenga los desarrollos de los 8 ejercicios con la solución conseguida`.
- Asegúrate de documentar tu código y explicar brevemente la lógica utilizada en cada ejercicio.

PARTE 2: PROGRAMACIÓN CON PYTHON

Título: Gestión de Cargas de Datos

Estructura de la Carpeta

- prueba-tecnica/
  - parte2/
    - data/
      - business_tables.json
      - origins_tables.json
    - src/
      - main.py

Descripción del Problema:

Como es conocido en el mundo de la ingeniería de datos nos dedicamos a realizar extracciones de datos desde los distintos sistemas operacionales que tiene una empresa, realizamos limpieza y transformaciones de esos datos para construir finalmente un modelo analítico, llamado modelo estrella o copo de nieve que es más eficiente para que los analistas de datos puedan sacar conclusiones orientadas al negocio.

Estas cargas de datos pueden tener distintas periodicidades, en este ejercicio se van a contemplar las periodicidades 'hourly', 'daily', 'weekly' y 'monthly' para las extracciones y transformaciones de datos.

La misión de este ejercicio es manejar el estado de estas cargas para establecer el estado 'PENDING' a las tablas operacionales y analíticas cuando se vayan a ejecutar dependiendo de la periodicidad.

Se proporcionan dos archivos JSON que contienen información sobre las tablas de origen y las tablas de negocio asociadas en el modelo estrella. Cada registro en los archivos JSON contiene los siguientes campos:

Estructura de los Archivos JSON:

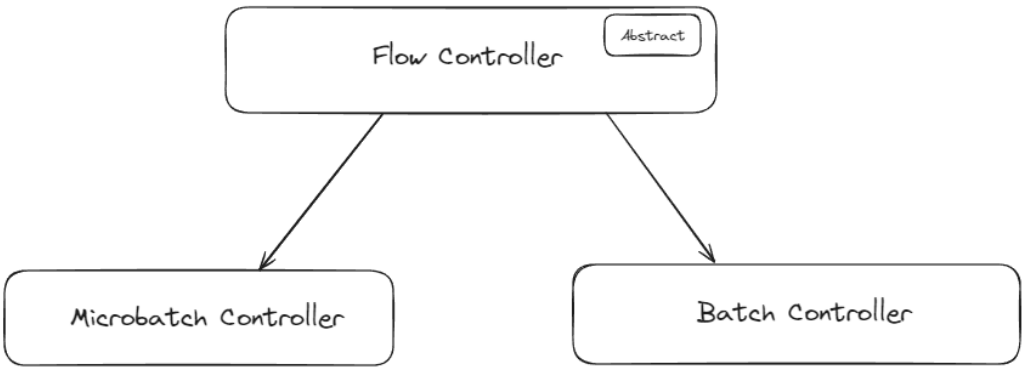
- **`origins_tables.json`:** Este archivo contiene información sobre las tablas de origen de los sistemas operacionales. Cada registro en este archivo tiene los siguientes campos:
    - `table_name`: Nombre de la tabla.
    - `schema`: Esquema al que pertenece la tabla.
    - `origin`: Base de datos operacional de la tabla.
    - `business_tables_ids`: Lista de identificadores de las tablas de negocio asociadas.
    - `periodicity`: Periodicidad de la tabla.
    - `status`: Estado de la tabla.

    Una tabla origen tiene como primary key la combinación de los campos `table_name` y `schema`.

- **`business_tables.json`:** Este archivo contiene información sobre las tablas del modelo estrella asociadas a las tablas de origen. Cada registro en este archivo tiene los siguientes campos:
    - `star_table_id`: Nombre de la tabla en el modelo estrella.
    - `origin_tables_ids`: Lista de diccionarios que identifican las tablas de origen con los campos `table_name`, `schema` y `origin`.
    - `periodicity`: Periodicidad de la tabla.
    - `status`: Estado de la tabla.

    Una tabla analítica tiene como primary key el campo `star_table_id`

Para realizar el ejercicio se deberá construir una jerarquía de clases como se ve en la siguiente imagen:



El objetivo es que se actualicen los campos de `status` de ambos ficheros JSON al valor 'PENDING' según los siguientes criterios:

Cargas microbatch:

Si el tipo de la carga es 'hourly':

- Se deberán actualizar a 'PENDING' los registros de ambos ficheros cuyo campo `periodicity` sea igual a 'hourly'.

Cargas batch:

Si el tipo de la carga es 'daily' se deberá:

- Actualizar a 'PENDING' los registros de ambos ficheros cuyo campo periodicity sea igual a 'daily'.
- Si el día que se ejecuta la carga es el lunes se deberán actualizar también los registros que tengan periodicity igual a 'weekly'.
- Si es día uno del mes se actualizarán también los registros con periodicity igual a 'monthly'.

La funcionalidad de conseguir las tablas orígenes y tablas de negocio a actualizar debe estar en las clases Microbatch Controller y Batch Controller dependiendo si la carga es de tipo 'hourly' o 'daily' respectivamente.

Se recomienda utilizar el patrón de método plantilla para esta solución, es por esto por lo que en la imagen de la jerarquía de clases aparece la clase FlowController como abstracta.

Ejemplo de patrón plantilla: https://refactoring.guru/es/design-patterns/template-method/python/example

El resto de la lógica del programa debe ser igual para ambos tipos de carga y debe realizarse en la clase FlowController.

El fichero main.py sirve como punto de entrada del programa en él se establecen las rutas a los ficheros JSON y el tipo de la carga.

```
workload_type = 'hourly'

path_origins_tables = './parte2/data/origins_tables.json'
path_business_tables = './parte2/data/business_tables.json'
```

Tip: Utiliza programación funcional con funciones map o filter siempre que se pueda y sea más cómodo.

Entregables:

- Archivos `batch_controller.py`, `microbatch_controller.py`, `flow_controller.py` y `main.py` que contengan la implementación de los controladores y el flujo principal del programa.

Buena suerte y cualquier duda aquí estamos para lo que necesites. ¡Mucho ánimo!