Universidad Carlos III

Arquitectura de Datos

Curso 2024-25

Práctica 2

Migración de una base de datos a `Cassandra`

Ingeniería Informática, Cuarto curso

Adrián Fernández Galán (NIA: 100472182, e-mail: 100472182@alumnos.uc3m.es)
César López Mantecón (NIA: 100472092, e-mail: 100472092@alumnos.uc3m.es)
Manuel Gómez-Plana Rodríguez (NIA: 100472310, e-mail: 100472310@alumnos.uc3m.es)

Prof . Lourdes Moreno López
Grupo: 81

Índice

1. Introducción **2**

2. Implementación y carga de las tablas **3**
2.1. Carga de datos en PySpark 3
2.2. Creación de las tablas intermedias 3
2.3. Creación de las tablas finales 4
2.4. Inserción de las tablas en Cassandra 5

3. Conclusiones **6**

1. Introducción

En este documento se recoge la parte de migración para el desarrollo de la práctica 2 de la asignatura *Arquitectura de Datos*. En esta práctica se tratará de completar una migración de una base de datos desde `MongoDB` a `Cassandra`. Además, se computarán nuevas tablas con el fin de permitir el análisis estadístico, aprovechando las cualidades de `Cassandra` para el análisis de datos gracias a su capacidad para la consulta masiva de datos de una misma columna.

La metodología usada para el desarrollo de este proyecto ha sido la siguiente: análisis de datos y casos de uso, realización del diseño lógico y físico orientado a `Cassandra` e implementación de casos de uso en forma e consulta. Adicionalmente se empleará la herramienta `PySpark` para realizar el primer volcado de datos.

En el diseño de consultas se tratará de aprovechar al máximo las capacidades de `Cassandra` en la lectura y escritura, dejando a la aplicación otra clase de operaciones. De esta forma, ambos sistemas trabajarán en conjunto, garantizando la eficiencia de la aplicación.

2. Implementación y carga de las tablas

Una vez analizados los casos de uso y definidas tanto las consultas como las tablas en el documento de diseño, es necesario tratar la implementación de esta base de datos. Como se ha mencionado en el otro documento, se ha usado PySpark para la carga y algo de preprocesado de la información original. Es por ello que este apartado trata en detalle el script creado para la carga de datos.

2.1. Carga de datos en PySpark

El primer paso para realizar la carga de datos en las tablas es crear una sesión de PySpark para leer los datos del *JSON* original. Para esto, primero hay que parsear el fichero original para que PySpark lo pueda procesar, algo que se ha realizado con la ayuda de un script extra que se encarga de introducir cada uno de los documentos originales en una lista, añadiendo los separadores entre documentos. Esto se ha realizado mediante el siguiente código:

```
import re

with open("./data/sample.json", "r", encoding="ISO-8859-1") as f:
    file_content = f.read()
    file_parsed = re.sub(r"(?<!}\s)\n(?!\s{)", "", file_content)
    file_parsed = re.sub(r" +", " ", file_parsed)
    file_parsed = re.sub(r"\n(?!\s])", ",\n", file_parsed)

with open("./data/sample_parsed.json", "w", encoding="utf-8") as f:
    f.write(file_parsed)
```

Listing 1: Parseo del JSON original

Una vez que se han obtenido los datos parseados, se cargan en una sesión de PySpark de la siguiente manera:

```
# ----- CREACION DE LA SESION DE PYSPARK -----
spark = SparkSession.builder \
    .appName("CargaDatos") \
    .config("spark.cassandra.connection.host", "127.0.0.1") \
    .config("spark.cassandra.connection.port", "9042") \
    .getOrCreate()

# ----- LECTURA DE FICHERO -----

# Leer el archivo NDJSON
json_df = spark.read.json(file_path)

# Aplicar la funcion al DataFrame original
json_df = rename_columns(json_df)
```

Listing 2: Carga de datos en PySpark

2.2. Creación de las tablas intermedias

Una vez que se han cargado los datos, es necesario crear las tablas en PySpark que servirán para alimentar las tablas explicadas en el apartado anterior. Las tablas de PySpark se corresponden con las tablas de color verde de los diagramas del punto 4 del pdf de diseño. Todas estas tablas se realizan de la misma manera, por lo que se usará la tabla de *speed tickets* como ejemplo al ser la más extensa. La creación de las tablas de PySpark se realiza de la siguiente manera:

```
# Seleccionar las columnas para la nueva tabla de velocidad
speed_ticket = json_df.filter(col("radar.speed_limit") < col("
    Record.speed")).select(
    col("Speed_ticket.Debtor.DNI").alias("dni_deudor"),
```

```
    col("vehicle.Owner.DNI").alias("dni_propietario"),
    col("vehicle.Driver.DNI").alias("dni_conductor"),
    col("Speed_ticket.Pay_date").alias("fecha_pago"),
    col("Speed_ticket.Amount").alias("cantidad"),
    concat(col("Record.date"), lit(" "), col("Record.time")).alias(
        "fecha_grabacion"),
    col("road.name").alias("carretera"),
    col("radar.mileage").alias("kilometro"),
    col("radar.direction").alias("sentido"),
    col("road.speed_limit").alias("velocidad_limite_carretera"),
    col("radar.speed_limit").alias("velocidad_limite_radar"),
    col("Record.speed").alias("velocidad_registrada"),
    col("vehicle.number_plate").alias("matricula"),
    col("Speed_ticket.State").alias("estado")
)
```

Listing 3: Creación de tablas en PySpark

2.3. Creación de las tablas finales

Al conseguir las tablas que servirán para alimentar las tablas sobre las que haremos las consultas, se pueden construir las tablas finales. De la misma manera que en el apartado anterior, todas se realizan de siguiendo el mismo algoritmo, por lo que se usará la tabla de *sanciones* como ejemplo al ser la más extensa. Estas tablas normalmente requieren de uniones entre los datos, creando así una generación muy extensa, por lo que se decidió crear una función para la generación de cada tabla, las cuáles cuentan con una estructura similar a la siguiente:

```
def gen_sanciones():

    # Calcula los speeding tickets
    speed = speed_ticket.select("dni_deudor", "dni_propietario", "
        dni_conductor", "fecha_grabacion", "estado", "matricula", "
        cantidad").withColumn("tipo", lit("velocidad"))

    # Calcula los clearence tickets
    clearance = clearance_ticket.select("dni_deudor", "
        dni_propietario","dni_conductor", "fecha_grabacion", "estado
        ", "matricula", "cantidad").withColumn("tipo", lit("
        clearance"))

    # Calcula los stretch ticket
    stretch = clearance_ticket.select("dni_deudor", "
        dni_propietario","dni_conductor", "fecha_grabacion", "estado
        " ,"matricula", "cantidad").withColumn("tipo", lit("stretch"
        ))

    # Obtiene impago y reorganiza
    impago = impago_sanciones.select("dni_deudor","dni_propietario"
        , "dni_conductor", "fecha_grabacion", "cantidad", "matricula
        ").withColumn("tipo", lit("impago")).withColumn("estado",
        lit("stand by"))
    impago = impago.select("dni_deudor", "dni_propietario", "
        dni_conductor", "fecha_grabacion", "estado", "matricula", "
        cantidad", "tipo")

    # Obtiene carne y reorganiza
    carne = discrepancia_carne.select("dni_propietario", "
        dni_conductor", "fecha_record", "matricula").withColumn("
        tipo", lit("discrepancia carne")).withColumn("estado", lit("
```

```
        stand by")).withColumn("cantidad", lit(1000)).withColumn("
            dni_deudor", discrepancia_carne["dni_conductor"])
    carne = carne.select("dni_deudor", "dni_propietario", "
            dni_conductor", "fecha_record", "estado", "matricula", "
            cantidad", "tipo")

    # Obtiene desperfectos y reorganiza
    desperfectos = vehiculo_deficiente.select("dni_propietario", "
            dni_conductor", "fecha_record", "matricula").withColumn("
            tipo", lit("discrepancia carne")).withColumn("estado", lit("
            stand by")).withColumn("cantidad", lit(1000)).withColumn("
            dni_deudor", vehiculo_deficiente["dni_propietario"])
    desperfectos = desperfectos.select("dni_deudor", "
            dni_propietario", "dni_conductor", "fecha_record", "estado",
             "matricula","cantidad", "tipo")

    return speed.union(clearance).union(impago).union(stretch).
            union(carne).union(desperfectos)
```

Listing 4: Carga de datos en PySpark

2.4. Inserción de las tablas en Cassandra

Una vez que se tienen las tablas finales creadas en PySpark, se procede al último paso, la inserción de datos en Cassandra. Para ello, se define una función que se encarga de escribir en Cassandra una tabla, la cuál tiene la siguiente forma:

```
def write_to_cassandra(table, name, mode):
    table.write.format("org.apache.spark.sql.cassandra")\
    .options(table=name, keyspace=KEYSPACE)\
    .mode(mode)\
    .save()
```

Listing 5: Carga de datos en PySpark

Un problema encontrado en la inserción fueron las fechas de las grabaciones. Para poder insertar una fecha que lograse aportar una mayor individualidad a las multas, se decidió juntar la fecha de grabación con el tiempo, logrando así un timestamp para las fechas. Para poder insertar estos datos correctamente, fue necesario cambiar el tipo de las columnas, algo que se realizó con la siguiente función:

```
def convertir_formato_fecha(df, columna_fecha):
    """
    Convierte el formato de una columna de fechas a timestamps

    :param df: DataFrame de PySpark
    :param columna_fecha: Nombre de la columna que contiene las
            fechas a convertir
    :return: DataFrame con la columna de fecha convertida
    """
    # Convertir la columna de fecha a formato timestamp
    df_converted = df.withColumn(
        columna_fecha,
        to_timestamp(col(columna_fecha), 'dd/MM/yyyy HH:mm:ss.SSS')
    )

    return df_converted
```

Listing 6: Carga de datos en PySpark

Con estos pasos, se consiguió insertar los datos en Cassandra.

3. Conclusiones

Esta parte de la práctica nos ha ayudado a comprender la potencia que poseen PySpark y otras herramientas similares en el desarrollo de bases de datos. La inserción ordenada y en masa que permite hacer PySpark sobre tablas creadas en varios lenguajes gestores de bases de datos hace que reluzca frente a otras herramientas.

A través de la migración, hemos comprendido que es un proceso más complicado de lo que pensábamos, en el que hay que tener en cuenta muchos aspectos como pueden ser el cálculo de columnas o el tipo de las columnas.