Bases de Datos 2
Práctica 5
Bases de datos NoSQL y su explotación: Cassandra y Spark

Carlos Tellería, Sergio Ilarri, Jordi Bernad

1. Objetivos de la práctica

1. Familirizarse con el SGBD NoSQl Cassandra y su explotación con Spark.

2. Introducción

Esta práctica se estructura en una serie de preguntas que irán guiando al alumnado para descubrir los aspectos básicos del sistema gestor de bases de datos NoSQL Cassandra y su explotación con Spark. Las respuestas a todas las preguntas se deberán introducir en la memoria de la práctica.

2.1. Instalación del entorno

El fichero yaml adjunto al material de la práctica, `docker-compose.yml`, contiene la información necesaria para levantar con docker una instancia de Cassandra, un spark master y dos spark workers. Se pueden levantar todos los servicios ejecutando en línea de comandos dentro del directorio donde se encuentre el fichero `docker-compose.yml`:

```
$ docker-compose up
```

En todas las máquinas virtuales que se levantan, en el directorio `/data/` está montado el directorio de la máquina local desde el que se lanzó el docker compose.

2.2. Descripción de los datos

Los datos sobre los que se realizará la práctica se pueden descargar desde el enlace https://drive.google.com/file/d/1dlImPcOpu0DM0pHECIQdQW4-my1SeRt2/. El fichero `dataset.csv` descargado (tras descomprimirlo) contiene información sobre distintas pruebas de laboratorio de un hospital en formato csv con campos `separados por ';'`. Cada línea representa los resultados de una prueba a una persona. Los campos que contiene son:

- `person`: identificador (número entero) de la persona a la que se realizó la prueba.
- `std_observable_cd`: código alfanumérico que identifica la prueba realizada.
- `std_observable_st`: breve descripción textual del tipo de prueba.
- `obs_value_nm`: número real con el resultado de la prueba.

Importante: a una persona se le ha podido realizar la misma prueba varias veces. Por ejemplo, observad las líneas 2 y 24 del fichero csv:

```
02: 123498;"3255-7";Fibrinogen [Mass/volume];536.0
24: 123498;"3255-7";Fibrinogen [Mass/volume];579.0
```

3. Explotación de Cassandra con CQL

Una forma de administrar y explotar el sistema gestor de bases de datos Cassandra es mediante el lenguaje *Cassandra Query Languge*. Es un lenguaje muy similar a SQL, pero con muchas limitaciones. Una de las limitaciones más destacadas de CQL es la falta de joins en las consultas.

`cqlsh` es una utilidad que lanza un shell para trabajar en CQL. Se puede ejecutar `cqlsh` con docker mediante :

```
$ docker-compose exec cassandra cqlsh
```

El fichero `laboratorio.cql` es un script escrito en CQL donde se declara la tabla `hospital` dentro del keyspace `laboratorio`. Para ejecutarlo dentro del shell de CQL:

```
cqlsh> source '/data/laboratorio.cql'
```

Pregunta 1. Observa el código:

```
CREATE KEYSPACE IF NOT EXISTS laboratorio
WITH REPLICATION = { 'class': 'SimpleStrategy',
                     'replication_factor': '1'};
```

¿Qué es un keyspace en Cassandra? ¿A qué concepto lo podemos asimilar en una base de datos relacional? ¿Qué implicaciones tiene `'replication_factor': '1'`?

Pregunta 2. Observa el código:

```
CREATE TABLE laboratorio.hospital (
    std_observable_cd text,
    person int,
    std_observable_st text,
    obs_value_nm float,
    PRIMARY KEY ((std_observable_cd), person)
) WITH CLUSTERING ORDER BY (person ASC);
```

Investiga qué es una *partition key* y una *clustering key* en Cassandra y añade una breve explicación de estos conceptos en la respuesta a esta pregunta.
¿Qué atributos de la tabla son partition keys y clustering keys? Si no se hubiese añadido al final de la declaración `WITH CLUSTERING ORDER BY (person ASC)`, ¿qué efectos habría tenido sobre la base de datos?

Pregunta 3. Añade al fichero `laboratorio.cql` el código necesario para cargar los datos del fichero csv a la tabla `laboratorio.hospital`. Utiliza la instrucción `COPY` de CQL.

Pregunta 4. Añade al fichero `laboratorio.cql` el código necesario para hallar el número total de pruebas diferentes que se realizan en el laboratorio y sus códigos.

Pregunta 5. Ejecuta en cqlsh:

```
cqlsh> select distinct std_observable_cd, std_observable_st
       from laboratorio.hospital;
```

¿Qué respuesta obtienes? Explica brevemente por qué.

Pregunta 6. Ejecuta en cqlsh:

```
cqlsh> select count(*) from laboratorio.hospital;
```

¿Qué respuesta obtienes? Explica brevemente por qué.

Pregunta 7. Añade al fichero `laboratorio.cql` consultas CQL para:

- hallar la media de los valores numéricos observados (atributo `obs_value_nm`) para la prueba de código `3255-7`;
- hallar la media de los valores numéricos observados de todas las pruebas realizadas a la persona con identificador igual a `123498`;
- todas las pruebas con valor numérico observado mayor o igual a 100.

Atentos a los mensajes de respuesta para las dos últimas consultas.

4. Explotación de Cassandra con Spark

Como habéis podido observar en la sección anterior, el lenguaje CQL es muy limitado para realizar consultas. Spark es una solución a este problema. Spark es un motor para analizar y procesar grandes cantidades de datos en entornos distribuidos escrito en el lenguaje de programación Scala. Dispone de APIs para programar en Scala, python, Java y R. En esta práctica usaremos la API de Scala mediante un shell interactivo. Para lanzar este shell de Spark junto con los conectores necesarios para Cassandra, ejecutar en la línea de comandos:

```
$ docker-compose exec spark spark-shell --packages
com.datastax.spark:spark-cassandra-connector_2.12:3.3.0
```

Spark se puede ejecutar en cualquier sistema operativo con una máquina virtual de Java (JVM). Como fuente de datos, Spark admite cualquier sistema de ficheros de los sistemas operativos más comunes, sistemas de ficheros en la nube como Azure Storage o Amazon S3, el sistema de ficheros de Hadoop HDFS, o mediante los conectores adecuados, bases de datos como Cassandra o HBase.

Uno de los tipos de datos básicos para trabajar con Spark son los Dataframes. Un Dataframe es una estructura de datos rectangular similar a los dataframes de python o R. Los dataframes en Spark disponen de una rica colección de métodos para realizar consultas que investigaréis en las próximas preguntas.

El fichero adjunto al enunciado de la práctica, `pivot.sc`, es un script de Scala para cargar la tabla `hospital` del keyspace `laboratorio` creada en la sección anterior. Para ejecutar el script en el spark-shell:

```
scala> :load /data/pivot.sc
```

La tabla está cargada en la variable (inmutable) `laboratory_table` de tipo `DataFrame`. Para contar el número de filas de la tabla se puede usar:

```
scala> laboratory_table.count
```

Pregunta 8. En la sección anterior, se han cargado los datos del fichero csv a la tabla de Cassandra. Al final de la carga, el SGBD informó de que se habían importado 11.354.000 de filas (una por cada línea del fichero csv). Sin embargo, `laboratory_table.count` nos indica que la tabla tiene muchas menos filas. Explica la razón. Ten en cuenta la observación " *Importante*" que se hizo en la sección 2.2 e investiga sobre la política *Last-Write-Wins* (LWW) para resolver conflictos en Cassandra.

Pregunta 9. Investiga cómo se resuelven las consultas de la Pregunta 7 usando los métodos `filter`, `select` y la función de agregación `avg`. Puedes consultar ejemplos en https://sparkbyexamples.com/spark/spark-sql-aggregate-functions/
Añade las consultas al fichero `pivot.sc`

Pregunta 10. Explica qué tarea realiza la instrucción del fichero `pivot.sc`

```
val pivotDF = laboratory_table.groupBy("person").
    pivot("std_observable_cd").
    sum("obs_value_nm")
```

¿Qué columnas tendrá el dataframe `pivotDF`? ¿Qué datos contiene?

Pregunta 11. Añade el código necesario en el fichero `pivot.sc` para hallar la media de los valores observados en la prueba `1742-6` utilizando el método `select` de `DataFrame`.

4.1. Spark SQL

La mayoría de desarrolladores del mundo, por una razón u otra, es fácil que estén familiarizados con el lenguaje SQL. Spark SQL es una forma extremadamente sencilla de crear consultas, si se tienen conocimientos de SQL. Dada una o varias variables de tipo dataframe en Spark, la forma de proceder para usar Spark SQL es la siguiente:

- Dado un dataframe, por ejemplo, `laboratory_table`, se crea una vista en Spark con:

  ```
  scala> laboratory_table.createOrReplaceTempView("lab")
  ```

  Se crea una vista/tabla del dataframe `laboratory_table` de nombre `lab`.

- Usando el objeto `spark` (creado automáticamente al lanzar el spark-shell) que representa una *Spark session* y el método `sql` podemos ejecutar cualquier consulta ANSI SQL sobre las vistas/tablas que estén en la Spark session. Por ejemplo,

  ```
  scala> val consultaDF = spark.sql("select * from lab")
  ```

  nos guardaría en el dataframe `consultaDF` el resultado de la consulta.

  En la cadena de caracteres del método `sql` que contiene la consulta SQL podemos hacer incluso joins de varias vistas que tengamos almacenadas

Pregunta 12. Utilizando Spark SQL, añade el código necesario en el fichero `pivot.sc` para realizar las consultas de la Pregunta 7 y Pregunta 11.

Pregunta 13. Añade a `pivot.sc` dos consultas más que no se puedan realizar con CQL.

5. Entrega de la práctica

Se debe entregar un fichero zip denominado *p5-<nia>.zip* (donde *<nia>* representa el NIA del coordinador/responsable del grupo) con el siguiente contenido:

1. Fichero *autores.txt*: contendrá el nombre y apellidos de los autores de la práctica y sus NIAs.

2. Directorio *fuentes*: contendrá todo el código desarrollado para la práctica, ficheros `laboratorio.cql` y `pivot.sc`.

3. Memoria en pdf de la práctica que contendrá las respuestas a todas las preguntas de este enunciado, junto con una sección de impresiones personales y la carga/distribución de trabajo entre los componentes del grupo.

Instrucciones de envío. El fichero anterior se someterá a través de Moodle (se habilitará una tarea a tal efecto) utilizando la cuenta del coordinador/responsable del grupo.

Fecha límite de entrega. La fecha límite de entrega es el **17 de mayo de 2025** (incluido).