Escuela de
Ingeniería y Arquitectura
Universidad Zaragoza

Universidad de Zaragoza

Facultad de Ingeniería

Memoria Práctica 5

Bases de Datos 2

Miércoles A 9:00-11:00

Sergio Isla Lasheras - 873983

Irene Pascual Albericio - 871627

Athanasios Usero Samarás - 839543

Mayo de 2025

Índice

1. Explotación de Cassandra con CQL:

A continuación se van a responder a las preguntas correspondientes del apartado:

1.1. Pregunta 1

Enunciado:

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'?

Respuesta:
Un keyspace en Cassandra define las opciones que se aplican a todas las tablas contenidas dentro del keyspace. El símil de un keyspace en Cassandra en una base de datos relacional sería una base de datos, ya que son donde se recopilan todas las tablas. *'replication_factor': '1'* significa que solo hay una copia de cada fila en el clúster.

1.2. Pregunta 2

Enunciado:

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?

Respuesta:
En Cassandra, la clave primaria (PRIMARY KEY) define cómo se distribuyen y organizan los datos en el clúster. Esta clave se divide en dos partes principales:

- `Partition Key`: La "partition key"determina en qué nodo del clúster se almacenará cada fila. Todas las filas con la misma partition key estarán agrupadas físicamente en el mismo conjunto de nodos. Cassandra calcula un hash de esta clave para ubicar la partición en el anillo.
- `Clustering Key`: La "clustering key"ordena las filas dentro de una misma partición. Permite definir el orden físico de los datos dentro del nodo. Se usa para realizar consultas eficientes dentro de una partición.

En el código, estos atributos son:

- `std_observable_cd`: Partition Key
- `person`: Clustering Key

Si no añadiésemos el "WITH CLUSTERING ORDER BY (person ASC)", Cassandra usará orden ascendente por defecto, por lo que en este caso no habría ningún cambio.

De todas formas, el hecho de explicitarlo, aumenta la claridad del diseño, se permite que se pueda cambiar el orden fácilmente si se desea y asegura el orden esperado incluso si se cambiase la versión o configuración del sistema.

1.3. Pregunta 3

Enunciado:

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.

Respuesta:

```
COPY laboratorio.hospital (person, std_observable_cd, std_observable_st,
    ↪ obs_value_nm)
FROM '/data/dataset.csv'
WITH DELIMITER=';' AND HEADER=TRUE;
```

1.4. Pregunta 4

Enunciado:

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.

Respuesta:

Para listar los códigos de pruebas diferentes se ha incluido en el fichero *laboratorio.cql* la siguiente consulta:

```
SELECT DISTINCT std_observable_cd FROM laboratorio.hospital;
```

Y nos ha devuelto el siguiente resultado:

```
 std_observable_cd
-------------------
          26464-8
          26484-6
           3173-2
           2951-2
          34714-6
          26515-7
           2160-0
          20570-8
           1968-7
          26449-9
           2823-3
          26444-0
           1988-5
          30428-7
           1920-8
           1742-6
           1975-2
          28539-5
           5902-2
          41652-9
          26453-1
           3255-7
          26474-7
          28540-3
           8124-0

(25 rows)
```

Para poder hallar el número total de pruebas diferentes se ha tratado de realizar una consulta empleando la función *COUNT*, pero nos han devuelto errores al tratar de ejecutar diferentes consultas empleando dicha función. De todas formas, la consulta anterior, después de devolver los resultados, ha devuelto el número de pruebas diferentes que es 25.

1.5. Pregunta 5

Enunciado:

Ejecuta en cqlsh:

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

¿Qué respuesta obtienes? Explica brevemente por qué.

Respuesta:
La respuesta obtenida ha sido la siguiente:

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

```
InvalidRequest: Error from server: code=2200 [Invalid query] message="SELECT
    ↪ DISTINCT queries must only request partition key columns and/or static
    ↪ columns (not std_observable_st)"
```

Esto se debe a que, en Cassandra, la consulta *SELECT DISTINCT* solo puede emplearse con columnas que formen parte de la clave de partición (*partition key*) o que estén definidas como columnas estáticas. En este caso, la columna *std_observable_st* no cumple ninguna de estas condiciones, lo que provoca el error. Solo puede usarse *SELECT DISTINCT* sobre columnas que sean clave de partición o estáticas, como podría ser el caso de *std_observable_cd* si cumple alguno de estos criterios.

1.6. Pregunta 6

Enunciado:

Ejecuta en cqlsh:

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

¿Qué respuesta obtienes? Explica brevemente por qué.

Respuesta:
La respuesta obtenida ha sido la siguiente:

```
cqlsh> select count(*) from laboratorio.hospital;
ReadTimeout: Error from server: code=1200 [Coordinator node timed out waiting for
    ↪  replica nodes' responses] message="Operation timed out - received only 0
    ↪  responses." info={'consistency': 'ONE', 'required_responses': 1, '
    ↪ received_responses': 0}
```

El error que muestra indica que se ha producido un timeout de lectura. Dicho error se ha producido debido a que la operación requería una única respuesta (*'consistency': 'ONE'*) de los nodos de réplica para completarse y a que no recibió ninguna respuesta antes de que se agotara el tiempo de espera (*'required_responses': 1, 'received_responses': 0*).
Este comportamiento puede deberse a que la consulta SELECT COUNT(*) en Cassandra no está optimizada, ya que requiere recorrer todas las particiones de la tabla de forma distribuida. Si la tabla contiene un gran volumen de datos, la operación puede resultar demasiado costosa y superar el tiempo de espera predefinido para lecturas.

1.7. Pregunta 7

Enunciado:

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.
¿Qué respuesta obtienes? Explica brevemente por qué.

Respuesta:
Para hallar la media de los valores numéricos observados para la medición del código 3255-7, se ha realizado la siguiente consulta:

```
SELECT AVG(obs_value_nm) FROM laboratorio.hospital WHERE std_observable_cd = '
    ↪ 3255-7';
```

Obteniendo como resultado el valor **590.41455**.
Para hallar la media de los valores numéricos observados de todas las pruebas realizadas a la persona con identificador igual a 123498, se ha realizado la siguiente consulta:

```
SELECT AVG(obs_value_nm) FROM laboratorio.hospital WHERE person = 123498 ALLOW
    ↪ FILTERING;
```

Al principio se ha ejecutado la consulta sin el uso de *ALLOW FILTERING*, lo cual nos ha dado un error y lo hemos tenido que poner para poder ejecutarla sin errores, siendo el resultado **68.96481**.
Para hallar todas las pruebas con valor numérico observado mayor o igual a 100, se ha realizado la siguiente consulta:

```
SELECT * FROM laboratorio.hospital WHERE obs_value_nm >= 100 ALLOW FILTERING;
```

En esta tercera consulta hemos tenido que poner *ALLOW FILTERING* por el mismo motivo que en la consulta anterior. Para esta consulta no se adjunta resultado debido a que el valor devuelto es demasiado extenso, aunque devuelve un total de **844726** filas.

2. Explotación de Cassandra con Spark:

2.1. Pregunta 8

Enunciado:

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.

Respuesta:

Este contraste de número de datos cargados se debe a la política *Last-Write-Wins* que aplica Cassandra. Dicha política la utiliza para resolver conflictos entre escrituras concurrentes o duplicadas. Se basa en una marca de tiempo asociada a cada escritura donde, si dos escrituras afectan a la misma celda, se conserva la versión con la marca de tiempo más reciente. Esto afecta principalmente a datos duplicados o múltiples escrituras a la misma clave, ya que Cassandra no almacenará múltiples versiones ni devolverá un error, sino que conservará solo la última versión.

2.2. Pregunta 9

Enunciado:

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.

Respuesta:

Para realizar la primera consulta del apartado 7, se ha realizado la siguiente consulta:

```
val apartado1 = laboratory_table.filter("std_observable_cd = '3255-7'").select(
    ↪ avg("obs_value_nm")).show()
```

Obteniendo como resultado **590.4145301377914**.

Para realizar la segunda consulta, se ha realizado lo siguiente:

```
val apartado2 = laboratory_table.filter("person = 123498").select(avg("
    ↪ obs_value_nm")).show()
```

Obteniendo como resultado **68.96480902250518**.

Para realizar la tercera consulta, se ha realizado lo siguiente:

```
val apartado3 = laboratory_table.filter("obs_value_nm >= 100").show()
```

Obteniendo como resultado unicamente las primeras 20 filas de entre todas las que devuelve la consulta (límite que ha establecido Spark), siendo ellas las siguientes:

```
+-----------------+------+------------+--------------------+
|std_observable_cd|person|obs_value_nm| std_observable_st|
+-----------------+------+------------+--------------------+
| 3173-2| 252| 121.4|aPTT in Blood by ...|
| 3173-2| 813| 121.4|aPTT in Blood by ...|
| 3173-2| 890| 121.4|aPTT in Blood by ...|
| 3173-2| 1596| 120.0|aPTT in Blood by ...|
| 3173-2| 1643| 120.0|aPTT in Blood by ...|
| 3173-2| 1644| 120.0|aPTT in Blood by ...|
| 3173-2| 2256| 121.4|aPTT in Blood by ...|
| 3173-2| 2622| 120.0|aPTT in Blood by ...|
| 3173-2| 2920| 121.4|aPTT in Blood by ...|
| 3173-2| 3869| 106.5|aPTT in Blood by ...|
| 3173-2| 4176| 120.0|aPTT in Blood by ...|
| 3173-2| 4489| 204.5|aPTT in Blood by ...|
| 3173-2| 4602| 204.5|aPTT in Blood by ...|
| 3173-2| 4820| 128.0|aPTT in Blood by ...|
| 3173-2| 5058| 106.5|aPTT in Blood by ...|
| 3173-2| 5341| 106.5|aPTT in Blood by ...|
| 3173-2| 7593| 106.5|aPTT in Blood by ...|
| 3173-2| 7945| 204.5|aPTT in Blood by ...|
| 3173-2| 7946| 204.5|aPTT in Blood by ...|
| 3173-2| 8271| 204.5|aPTT in Blood by ...|
+-----------------+------+------------+--------------------+
```

2.3. Pregunta 10

Enunciado:

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?

Respuesta:

La instrucción realiza una operación de pivotado sobre el *DataFrame* `laboratory_table`. Concretamente, agrupa los datos por la columna `person` y transforma los valores únicos de la columna `std_observable_cd` en nuevas columnas. Para cada combinación de persona y código observable, se calcula la suma de los valores de `obs_value_nm`.

El *DataFrame* resultante, llamado `pivotDF`, tendrá una primera columna llamada `person`, seguida de una columna por cada valor distinto de `std_observable_cd`. Cada una de esas columnas contiene la suma total de los valores `obs_value_nm` que corresponden a ese código observable para la persona indicada.

2.4. Pregunta 11

Enunciado:

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.

Respuesta:

Para hallar la media de los valores observados en la prueba 1742-6, se ha realizado la siguiente consulta:

```
val pregunta11 = laboratory_table.filter("std_observable_cd = '1742-6'").select(
    ↪ avg("obs_value_nm")).show()
```

La consulta aplica el método *filter* para buscar el código de prueba correspondiente, posteriormente se ha aplicado el método *select* para seleccionar que queremos que nos devuelva la media de los valores observados y, finalmente, se ha empleado el método *show* para mostrar por la terminal los resultados.
Dicha consulta nos ha devuelto que el resultado es: **37.79978569736477**.

2.5. Pregunta 12

Enunciado:

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

Respuesta:

Previo a realizar y ejecutar las consultas, se ha creado una vista en Spark en base al dataframe *laboratory_ table* que se ha empleado anteriormente.

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

A continuación, se muestra el código realizado para las querys de la pregunta 7:

```
val query1 = spark.sql("SELECT AVG(obs_value_nm) FROM lab WHERE std_observable_cd
    ↪  = '3255-7'").show()
val query2 = spark.sql("SELECT AVG(obs_value_nm) FROM lab WHERE person = 123498")
    ↪ .show()
val query3 = spark.sql("SELECT * FROM lab WHERE obs_value_nm >= 100").show()
```

Para dichas querys el resultado obtenido es el siguiente: **590.4145301377914** para la primera query, **68.96480902250518** para la segunda query y la siguiente tabla para la tercera query.

```
+-----------------+------+------------+--------------------+
|std_observable_cd|person|obs_value_nm| std_observable_st|
+-----------------+------+------------+--------------------+
| 1988-5| 33| 129.5|C reactive protei...|
| 1988-5| 54| 158.0|C reactive protei...|
| 1988-5| 97| 158.0|C reactive protei...|
| 1988-5| 98| 158.0|C reactive protei...|
```

```
| 1988-5| 99| 122.9|C reactive protei...|
| 1988-5| 113| 122.9|C reactive protei...|
| 1988-5| 132| 112.7|C reactive protei...|
| 1988-5| 163| 116.2|C reactive protei...|
| 1988-5| 166| 128.9|C reactive protei...|
| 1988-5| 184| 112.7|C reactive protei...|
| 1988-5| 206| 116.2|C reactive protei...|
| 1988-5| 207| 164.11|C reactive protei...|
| 1988-5| 212| 118.7|C reactive protei...|
| 1988-5| 258| 112.7|C reactive protei...|
| 1988-5| 261| 112.7|C reactive protei...|
| 1988-5| 263| 112.7|C reactive protei...|
| 1988-5| 314| 145.0|C reactive protei...|
| 1988-5| 332| 116.3|C reactive protei...|
| 1988-5| 344| 112.7|C reactive protei...|
| 1988-5| 370| 116.2|C reactive protei...|
+-----------------+------+------------+--------------------+
only showing top 20 rows
```

A continuación, se muestra el código realizado para la query de la pregunta 11:

```
val query4 = spark.sql("SELECT AVG(obs_value_nm) FROM lab WHERE std_observable_cd
    ↪  = '1742-6'").show()
```

Para el cual se ha obtenido el siguiente resultado: **37.79978569736477**.

2.6. Pregunta 13

Enunciado:

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

Respuesta:

Las consultas que se han propuesto han sido consultas que nos han dado errores en el anterior apartado, empezando con la de hallar el número total de pruebas diferentes que se realizan en el laboratorio y, una consulta realizada en la pregunta 7, hallar todas las pruebas con valor numérico observado mayor o igual a 100. El código correspondiente a dichas consultas se presenta a continuación:

```
val query5 = spark.sql("SELECT COUNT(*) FROM (SELECT DISTINCT std_observable_cd
    ↪ FROM lab)").show()
val query6 = spark.sql("SELECT * FROM lab WHERE obs_value_nm >= 100").show()
```

El resultado obtenido en la primera consulta es **25** y para la segunda consulta es la siguiente tabla:

```
+-----------------+------+------------+--------------------+
|std_observable_cd|person|obs_value_nm| std_observable_st|
+-----------------+------+------------+--------------------+
| 26464-8| 3865| 155.6|Leukocytes [#/vol...|
| 26464-8| 3866| 155.6|Leukocytes [#/vol...|
| 26464-8| 4840| 199.0|Leukocytes [#/vol...|
| 26464-8| 5057| 155.6|Leukocytes [#/vol...|
| 26464-8| 5168| 307.9|Leukocytes [#/vol...|
| 26464-8| 5510| 199.0|Leukocytes [#/vol...|
```

```
| 26464-8| 6224| 156.2|Leukocytes [#/vol...|
| 26464-8| 6585| 307.9|Leukocytes [#/vol...|
| 26464-8| 6995| 156.2|Leukocytes [#/vol...|
| 26464-8| 7703| 307.9|Leukocytes [#/vol...|
| 26464-8| 9510| 177.1|Leukocytes [#/vol...|
| 26464-8| 9623| 137.87|Leukocytes [#/vol...|
| 26464-8| 10845| 137.87|Leukocytes [#/vol...|
| 26464-8| 10983| 155.6|Leukocytes [#/vol...|
| 26464-8| 12891| 156.2|Leukocytes [#/vol...|
| 26464-8| 12892| 156.2|Leukocytes [#/vol...|
| 26464-8| 14030| 307.9|Leukocytes [#/vol...|
| 26464-8| 14495| 307.9|Leukocytes [#/vol...|
| 26464-8| 14946| 137.87|Leukocytes [#/vol...|
| 26464-8| 15807| 156.2|Leukocytes [#/vol...|
+----------------+------+------------+--------------------+
```