**Advanced Databases**

**Cassandra CA**

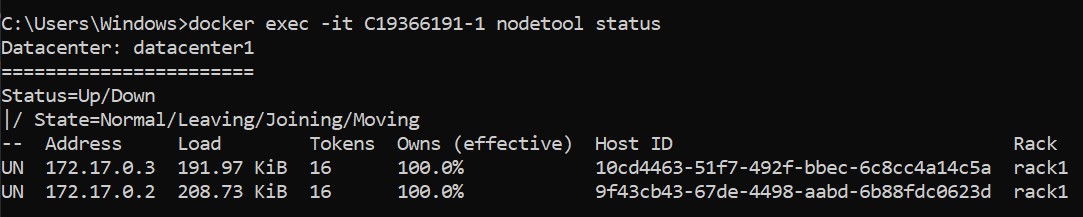
**Student Number:** C19366191

**Student Name:** Lina Mir

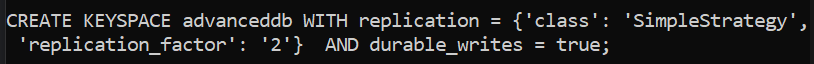
**Programme Code:** TU856

1. **Setting up the cluster and keyspace**

The cluster contains two nodes, one called C19366191-1 and the other called C19366191-2. They are both situated on rack1. They run on port 9042 and have pulled from the latest Cassandra container.



The UN shows the nodes are up and normal. The addresses on the nodes are two different ones and their size is different. They both own 100% of the nodes and share the data.



This is the keyspace that was created. It has a replication factor of 2. This means that there will be two copies of data. The class is SimpleStrategy. This is a SimpleStrategy because a single data centre is used and the single rack, this strategy will be used.

1. **Porting the data to Cassandra**
   1. **The query used on the fact table that will be used in the python file is below.**

A screenshot of a computer

Description automatically generated

This will get the name of the player, their player sk, the tournament name, year, the rank, and prize they won then.

* 1. **The python file submitted is what was used to port the data and create and populate the table in Cassandra. This was run in the terminal and the output was as follows:**

A computer screen capture

Description automatically generated with medium confidence

The uploading to Cassandra goes as far as row 28.

The JSON file is also submitted, where the results of the query are stored.

Text

Description automatically generated

The size of the load here is around 227kb and 224kb. This has now increased since the first time, where it was from 190kb to 200kb. Since more data has been added, this makes sense.

The command used to investigate the table was docker exec -it C19366191-1 nodetool tablestats advanceddb.factresults

The screenshot is below. Some key features to note from this:

* The write latency is 0.16010714285714286 ms which means it will take this amount of time to complete the most recent write request.
* The table name is here too, factresults.
* The memtable cell count is 28 and this means that 28 rows of data have been put in.
* The memtable size is 841, which is showing how much data is stored in the memtable.
* The bloom filter space is 0 which is the space used to store the bloom filter data.
* The bloom filter false positives ratio shows the fraction of all bloom filter checks resulting in false positives and here the number is 0.

Text

Description automatically generated

1. **Golf data** 
   1. **Basic query on golf data**

Tracing was set to be on.

The basic query done is select \* from factresults;

The output for this query is here:

Text

Description automatically generatedThis shows the stages it takes for the query. The first step is to execute the query and then to parse it. After that the statement is prepared and the ranges to the query are computed. Then the range request with concurrency of 1 is submitted and the rest of show in the screenshot.

* 1. **Query of non-primary index (without index)**

Tracing was set to be on.

The query of a non-primary index was done is select \* from factresults where t\_name = ‘Irish Open’ allow filtering;

The output for this query is here:

Text

Description automatically generated

This screenshot shows the process the query follows to get the output. The query is executed, and the statement is parsed and prepared. The ranges are computed and the range requests on 33 ranges with a concurrency of 1 are submitted. The concurrent range requests are submitted and then the seq scan is executed. 7 live rows are read and then the request is complete.

* 1. **Adding a secondary index to golf data**

Tracing was set to be on.

The index that was created was on the rank, create index player\_rank on factresults(rank);

The query that was performed was select \* from factresults where t\_name = ‘Irish Open’ allow filtering;

This is the output that was given on this query.

Graphical user interface, text

Description automatically generated

The process that the query followed was it parsed the query, then the statement was prepared and the indexes that could be applicable were not found. This was because the query did not use the column that was indexed. Then the ranges to query were computed and the range requests on 33 ranges with a concurrency of 14 were submitted. The rows per range were 7.2. One concurrent range was submitted and the seq scan was executed. 7 live rows were read, and the request was completed.

The node status was performed on the table now. This is the output:

Text

Description automatically generated

The read latency on this table now is 1.766142857142857 ms. The write latency on the table is 0.15088571428571426 ms. The write count is 35 now. The Index summary off heap memory used is 16 and the bloom filter space used is 24 while the bloom filter false positive ratio is 0.

* 1. **Adding an SASI index to golf data to facilitate pattern matching**

The SASI index that was put on the table is the following:



This index was put on the tournament name as a prefix SASI index.

This is the query and the results that was run using pattern matching to see it in action:

Timeline

Description automatically generated with medium confidence

It shows the details of the players matching the tournament name with the letter I, showing the Irish tournaments.

Tracing was set to on for this query.

A picture containing text

Description automatically generated

The tracing output shows that it went through these stages to execute the query. It parsed the query first, then prepared the statement and scanned using the index created above. Then it computed ranges to the query and submitted range requests. It went through many other stages as seen in the screenshot.

This is what nodetool tablestats shows:

Text

Description automatically generated

This shows now that the space used is more. The write count has increased from 7 to 35 since the last change. The number of partitions is 5 whereas the previous ones is 7. The compacted partition minimum bytes are now 73. The compacted maximum is not 103 and the mean bytes are 93.

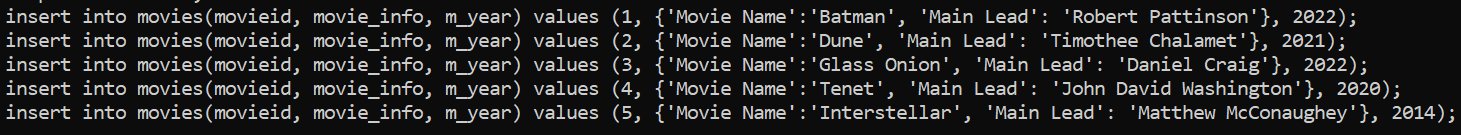
1. Data including collection data type
   1. Creating and inserting into table

<<use tracing, capture the output and include it here, comment on what is happening>>

The table that was created was a movie table. This contained collection data type, which is a map with two text fields, one for the movie name and the main lead.



Data was inserted into the table.



* 1. Query of non-primary index (without index)

This is the query without the index. It gets the movies where the year was 2022.

A picture containing text

Description automatically generated

This is the query with tracing on:

Graphical user interface

Description automatically generated with medium confidence

It starts with parsing and then prepares the statement. It computes the ranges to query and then submits the range requests on 33 ranges with a concurrency of 1. It submits the concurrent range requests. It executes the seq scan across 0 sstables. And it had read 5 lives rows and 0 tombstone cells.

* 1. Adding a secondary index

<<use tracing, capture the output and include it here, comment on what is happening>>

<<use nodetool tablestats to investigate your table, comment on key pieces of

Information>>

The index was created on the map column as shown below.



The query that was done this time was this:

Text

Description automatically generated

This found the movie information where the movie name was batman.

Then with tracing on this was the output.

A screenshot of a computer

Description automatically generated with medium confidence

It went through these following stages. After parsing and preparing the statement, it finds the index mean cardinalities and scans with the index. It submits a range request on 33 ranges and goes through more stages.

The tablestats appear are shown.

Text

Description automatically generated

The write count has increased since the last tablestats and the