





















**Answer 1:** It only worked for Table 3.

The depiction of key assignments is shown in the following table:

|  |  |  |  |
| --- | --- | --- | --- |
| **TABLE** | **PRIMARY KEY** | **PARTITION KEY** | **CLUSTER KEY** |
| sentiment1 | dt | - | - |
| sentiment2 | - | ch,dt | - |
| sentiment3 | - | ch | dt |
| sentiment4 | - | ch | cu,dt |
| sentiment5 | - | ch,dt | cu |

Partition keys in Cassandra tend to distribute the partition across the columns that use the selected partition key. These values are arranged in an ascending order and therefore, operators like ‘>’, ‘<’, ’>=’ and ‘<=’ cannot be used successfully. Alternatively, the use of token functions is encouraged; as the query contains ‘>=’ and ‘<=’, which is only allowed in cluster key and not in a partition key.

Table sentiment1 has ‘dt’ as a primary key which doesn’t allow filtering for the ‘ch=facebook’ as called in the query.

Table sentiment2 has a partition key ‘ch,dt’ which doesn’t allow the use of ‘>=’ AND ‘<=’ as called in the query.

Table sentiment4 has a partition key ‘ch’ and cluster key ‘cu,dt’ and therefore doesn’t allow the use of ‘>=’ and ‘<=’ as called in the query as Cassandra will have to scan the entire partition.

Table sentiment5 has a partition key ‘ch,dt’ and cluster key ‘cu’ and therefore doesn’t allow the query operator.

**Table sentiment 3, however, has a partition key ‘ch’ and cluster key ‘dt’, which can allow the ‘>=’ and ‘<=’ operators called in the query. Clustering columns support the use of ‘>’, ‘<’, ‘>=’ and ‘<=’ operators.**

**Answer 2:**

Two techniques:

1. The ALLOW FILTERING clause provides flexibility for querying in Cassandra, but at the cost of performance issues. For the SELECT query to run, the query should include the WHERE clause containing the full partition key, otherwise no results will be retrieved. The required query for this question constitutes ‘sent’, which is not partition key for any tables, and therefore it will not display any results.

Cassandra needs “Allow Filtering” along with the use of WHERE with the inclusion of the partition key because otherwise, if it isn’t used, Cassandra runs the search query on the entire table for desired results, therefore using up resources such as efficiency and time.

The use of “Allow Filtering” ensures that Cassandra orders the rows first, then searches for desired results by filtering the undesired results in the respective ordered rows. It would still use plentiful resources if the table constitutes a substantial amount of rows. Therefore, it isn’t really favorable in such a situation.

1. The Secondary Index use is much better as it ensures that Cassandra searches for the rows as mentioned explicitly in the Index declarations, and only retrieves results from the matched rows. Indexes are commonly used to improve the performance of queries.

**Answer 3:**

One way to solve this problem is to assign a TTL to data while it is being inserted. The Time To Live, or TTL for short, can be explicitly specified for the data, until it is collected and meets the criteria for the range specified in the query. At the commencement of the TTL period, the data automatically expires from the tables and novel data is inserted with a specific TTL for a certain time range, until the tweets are cumulated. When the data has expired, it can be labeled as Tombstones. These Tombstones do consume storage, but also live up to certain desired timespans only. TTL can be used to mark these expired rows with Tombstones, and tweet results can be amalgamated in specific time periods.