Assignment No. 4

Aim: Design a Database Schema for an application using Cassandra (CQL).

Pre-Requisites: DBMS, MongoDB

Objective:

1.Efficiently store and retrieve product information.

2.Enable fast retrieval of order details for a given customer.

3.Facilitate quick retrieval of customer information.

4.Support the tracking of order history.

Outcomes:

1.Scalable and distributed storage for high availability.

2.Low-latency queries for product information, customer details, and order history.

3.Flexibility to handle growth in data and traffic.

Theory:

Designing a database schema for an application using Cassandra Query Language (CQL) involves considering the characteristics of Cassandra, which is a NoSQL database designed for horizontal scalability, high availability, and fault tolerance.

Here are some key principles and considerations when creating a database schema for a Cassandra-based application:

1.Denormalization:

- Cassandra encourages denormalization to optimize for read performance. It's common to duplicate data across tables to avoid complex joins and to ensure that data is available in the form needed for specific queries.

2.Query-Driven Schema Design:

- Design your schema based on the queries your application will perform. Cassandra is optimized for specific query patterns, so the schema should align with the types of queries your application will execute.

3.Understanding Data Distribution:

- Cassandra distributes data across nodes based on the partition key. The choice of partition key is crucial for achieving even distribution and efficient query performance. It's essential to select a partition key that evenly distributes data and prevents hotspots.

4.Partitioning and Clustering Keys:

- Define a good partition key to distribute data evenly across nodes. Additionally, use clustering keys to define the order of data within a partition, as this impacts the physical storage and retrieval of data.

5.Avoiding Secondary Indexes:

- Cassandra supports secondary indexes, but their use comes with performance considerations. Avoid using too many secondary indexes, as they can impact write performance and may not scale well in distributed environments.

6.Understanding Compaction:

- Cassandra uses a process called compaction to merge and organize data on disk. It's important to understand the compaction strategy and tune it according to your application's needs.

7.Materialized Views:

- Cassandra supports materialized views, which allow you to create alternative views of your data. This can be useful for handling different query patterns without requiring complex query logic.

8.Time-Series Data:

- If your application involves time-series data, consider using time-based strategies such as time window compaction or time bucketing to optimize data storage and retrieval.

9.Cassandra Data Types:

- Use appropriate data types for your columns. Cassandra supports various data types, including collections (lists, sets, maps), which can be useful for modeling certain types of data.

Here are key theoretical aspects and principles associated with Cassandra and CQL:

Distributed Architecture:

- Cassandra is designed to be distributed, allowing it to scale horizontally by adding more nodes to the cluster. This distributed architecture provides fault tolerance and high availability.

Peer-to-Peer Model:

- Cassandra follows a peer-to-peer model where all nodes in the cluster have equal status. Each node can accept read and write requests, and there is no single point of failure.

No Single Point of Failure:

- Cassandra is built to ensure high availability and fault tolerance. Data is replicated across multiple nodes, and if one node fails, the system can continue to operate with the remaining nodes.

CAP Theorem:

- Cassandra adheres to the CAP theorem, which states that in a distributed system, it's impossible to simultaneously provide all three of the following guarantees: Consistency, Availability, and Partition tolerance. Cassandra is designed to provide high Availability and Partition tolerance, making it an AP system.

Eventual Consistency:

- Cassandra provides eventual consistency, meaning that given enough time and assuming no further updates, all replicas of a piece of data will converge to the same value. This model allows for high availability and performance in distributed environments.

Conclusion:

Hence, we have successfully studied Design of Database Schema for an application using Cassandra (CQL)

Frequently Asked Questions:

1.Retrieving all FAQ entries for a specific category.

2.Fetching details of a particular FAQ entry.

3.Updating an FAQ entry.

4.Adding a new FAQ entry.