Scenario Based Cassandra Interview Questions

1. **You are tasked with designing a music streaming service using Cassandra. How would you model the data to efficiently store user playlists and song metadata?**

   We could have a table for songs with song_id as the primary key and columns for song metadata (like title, artist, duration, genre). For user playlists, we could use another table that uses a composite primary key of (user_id, song_id). The reason for this choice of primary key is to enable us to quickly add and remove songs from a user's playlist, as well as retrieve all the songs in a user's playlist.

   CREATE TABLE songs (song_id UUID PRIMARY KEY, title text, artist text, duration int, genre text);
   CREATE TABLE playlists (user_id UUID, song_id UUID, added_at timestamp, PRIMARY KEY (user_id, song_id));

2. **In a distributed blog platform, how would you design a schema to efficiently handle the retrieval of the latest blog posts from a particular user in Cassandra?**

   we could use a TimeUUID type for the post_id. This allows us to sort posts by the time they were created. By making (user_id, post_id) the primary key for the posts table, we can quickly retrieve the latest posts for a particular user using a range query on post_id.

   CREATE TABLE posts (user_id UUID, post_id timeuuid, title text, content text, PRIMARY KEY (user_id, post_id));

3. **How would you design your Cassandra data model for a real-time chat application to ensure all messages are delivered to all participants, and older chats are retrieved quickly?**

   consider using a table for messages with a primary key of (chat_id, timestamp). This would allow us to retrieve messages for a chat in chronological order. To ensure all messages are delivered to all participants, we could use a "message delivery" table that keeps track of which users have received each message.

   CREATE TABLE messages (chat_id UUID, message_time timestamp, user_id UUID, message text, PRIMARY KEY (chat_id, message_time));

CREATE TABLE message_delivery (message_id UUID, user_id UUID, delivered boolean, PRIMARY KEY (message_id, user_id));

4. **If you're designing a ride-sharing application like Uber or Lyft, how would you leverage Cassandra for handling real-time locations of millions of riders and drivers?**

   we could use a table for storing the location of users. The primary key could be (user_id, timestamp), which allows us to keep track of the latest location of all riders and drivers. If we needed to query locations based on proximity, we could use a technique called geohashing.

   CREATE TABLE locations (user_id UUID, location_time timestamp, lat decimal, long decimal, PRIMARY KEY (user_id, location_time));

5. **You are designing a Cassandra database for a large e-commerce website. How would you create the data model to support operations such as listing all the products in a certain category, keeping track of users' shopping carts, and storing users' order history?**

   we could use a table for products with category_id and product_id as a part of the primary key. This would allow us to quickly retrieve all products within a specific category.

   CREATE TABLE products (category_id int, product_id UUID, name text, price decimal, description text, PRIMARY KEY (category_id, product_id));

   For a shopping cart, we could use another table that tracks the items in each user's cart. Here, user_id would be a partition key, and product_id would be a clustering column, allowing us to quickly add, remove, and retrieve products for a specific user's cart.

   CREATE TABLE carts (user_id UUID, product_id UUID, quantity int, PRIMARY KEY (user_id, product_id));

   For order history, a separate table could be used, which allows us to quickly retrieve all the orders placed by a specific user.

   CREATE TABLE orders (user_id UUID, order_id timeuuid, product_id UUID, quantity int, order_time timestamp, PRIMARY KEY (user_id, order_id));

6. **In a gaming application, player profiles are read frequently and updated occasionally, but each player's profile could be quite large. How would you design the data model in Cassandra to support this use case efficiently?**

consider using a user profile table with user_id as the primary key. The player's profile could contain data like the username, email, achievements, and other game-specific data. Given that profiles are read frequently but updated occasionally, we can also utilize caching techniques (like Cassandra's built-in row caching or using an external caching system like Redis) to reduce read latencies.

CREATE TABLE player_profiles (player_id UUID PRIMARY KEY, username text, email text, achievements list<text>);

7. **You are working with a social networking site and you need to create a feature to retrieve the posts from friends that a user follows. How would you model your data in Cassandra to efficiently support this operation?**

   We can create a "user_posts" table that uses (user_id, post_id) as the primary key. This enables us to quickly retrieve all posts made by a specific user.

   CREATE TABLE user_posts (user_id UUID, post_id timeuuid, content text, PRIMARY KEY (user_id, post_id));

   To fetch posts from friends, we would need to first fetch the list of friends for the user and then do individual reads for each friend. This operation can be made efficient by using an appropriate data model for storing friend lists and using concurrent reads to fetch posts.

8. **You have a Cassandra cluster with multiple datacenters. How would you configure the replication strategy to ensure low latency reads and writes, as well as redundancy across different geographic regions?**

   we would use the NetworkTopologyStrategy for replication. This strategy allows us to specify the number of replicas in each datacenter. The placement of replicas is done in a way that ensures one replica is placed in each rack, which maximizes redundancy and provides protection against rack failures. This strategy is also aware of the network topology and routes requests to the closest replicas, providing low latency reads and writes.

   CREATE KEYSPACE mykeyspace WITH REPLICATION = { 'class' : 'NetworkTopologyStrategy', 'dc1' : 3, 'dc2' : 2 };

9. **If you are developing a stock market application which needs to store and retrieve real-time and historical stock prices, how would you model the data in Cassandra?**

   We need to store both real-time and historical stock prices. This is a time-series data problem and can be modeled in Cassandra by using a wide-row model. We could

use a table with (stock_symbol, timestamp) as the primary key and price as a column. This enables us to quickly insert new prices and retrieve historical price data for a specific stock.

CREATE TABLE stock_prices (stock_symbol text, price_time timestamp, price decimal, PRIMARY KEY (stock_symbol, price_time));

10. **Given the task of developing a Cassandra-based backend for an IoT application that constantly receives huge amounts of sensor data, how would you design the data model and which compaction strategy would you choose to handle this use case?**

    Considering the constant stream of sensor data, we can model the data as time-series data. Similar to the stock market application, we could use a table with (sensor_id10. For the IoT application, considering the constant stream of sensor data, we can model the data as time-series data. Similar to the stock market application, we could use a table with (sensor_id, timestamp) as the primary key. The choice of compaction strategy depends on the read/write pattern. If data is mostly written and rarely read, a strategy like Leveled Compaction would be ideal. For frequent reads of recent data, Date Tiered or Time Window Compaction might be more suitable.

    CREATE TABLE sensor_data (sensor_id UUID, recorded_at timestamp, value decimal, PRIMARY KEY (sensor_id, recorded_at));

11. **Consider a system where you need to track user events on a website and then display a timeline of these events to the user. How would you design the Cassandra tables to efficiently handle this use case?**

    We could create an 'events' table with a composite primary key of (user_id, timestamp), allowing us to retrieve all events for a particular user in chronological order.

    CREATE TABLE events (user_id UUID, event_time timestamp, event_type text, event_data text, PRIMARY KEY (user_id, event_time));

12. **Suppose you have to implement a distributed task queue with Apache Cassandra where tasks have different priorities. How would you design this?**

    we could use a table 'tasks' with columns for task_id, priority, status, and other task-related data. The primary key can be (status, priority, task_id), which ensures that tasks with higher priority (lower priority number) come first.

    CREATE TABLE tasks (status text, priority int, task_id UUID, data text, PRIMARY KEY (status, priority, task_id));

13. **Imagine you're developing a movie recommendation system. How would you design a Cassandra schema to efficiently store and retrieve user ratings for various movies?**

    We can create a 'ratings' table where the primary key is (user_id, movie_id), and there's a column for rating. This allows quick update and retrieval of a user's rating for a movie.

    CREATE TABLE ratings (user_id UUID, movie_id UUID, rating int, PRIMARY KEY (user_id, movie_id));

14. **You are tasked with developing an online collaborative document editing platform. How would you leverage Cassandra to store document data and handle simultaneous edits from multiple users?**

    we could store each document as a collection of operations or changes, rather than the document content itself. Each operation could be an insertion, deletion, or update, and should include the position in the document and the content to be inserted or updated.

    CREATE TABLE document_changes (doc_id UUID, change_time timestamp, user_id UUID, operation text, position int, content text, PRIMARY KEY (doc_id, change_time));

15. **You have a requirement to store time-series data for an application monitoring system, where you need to track system metrics every minute. How would you model your data in Cassandra to serve this requirement?**

    We could create a 'metrics' table with (system_id, metric_name, timestamp) as the primary key, allowing us to retrieve all values of a specific metric for a particular system in a given time range.

    CREATE TABLE metrics (system_id UUID, metric_name text, recorded_at timestamp, value decimal, PRIMARY KEY ((system_id, metric_name), recorded_at));