Cassandra

To store the data in Cassandra, we will create a column-family with the name products. The columns in this column-family will represent the various attributes of the products we want to store.

The columns are:

product\_id: a unique identifier for each product

name: the name of the product

manufacturer: the name of the manufacturer of the product

price: the price of the product

quantity: the number of units available in stock

features: a map containing various features of the product

reviews: a set of customer reviews for the product

CREATE TABLE products (

product\_id uuid PRIMARY KEY,

name text,

manufacturer text,

price float,

quantity int,

features map<text, text>,

reviews set<text>

);

INSERT INTO products (product\_id, name, manufacturer, price, quantity, features, reviews)

VALUES (uuid(), 'iPhone 13', 'Apple', 999.99, 100, {'color': 'blue', 'storage': '128GB'}, {'Great phone!', 'Amazing camera!'}),

(uuid(), 'Galaxy S21', 'Samsung', 899.99, 50, {'color': 'black', 'storage': '256GB'}, {'Best phone ever!', 'Love the screen!'}),

(uuid(), 'Surface Laptop 4', 'Microsoft', 1299.99, 25, {'color': 'platinum', 'processor': 'Intel Core i7'}, {'Great performance!', 'Beautiful design!'}),

(uuid(), 'iPad Pro', 'Apple', 799.99, 75, {'color': 'silver', 'storage': '256GB'}, {'Best tablet!', 'Fast and reliable!'}),

(uuid(), 'Watch Series 7', 'Apple', 399.99, 200, {'color': 'red', 'size': '45mm'}, {'Love the new design!', 'Amazing features!'});

10 different queries

1. Retrieve all products from the table:

SELECT \* FROM products;

1. Retrieve all products with a price less than $1000

SELECT \* FROM products WHERE price < 1000;

1. Get all products from a specific manufacturer is Samsung

SELECT \* FROM products WHERE manufacturer = ‘Samsung’;

1. Get name and price of the most expensive product

SELECT name, price FROM products ORDER BY price DESC LIMIT 1;

1. Update the quantity of a specific product (iPhone 13) to a new value (150):

UPDATE products SET quantity = 150 WHERE name = 'iPhone 13';

1. Delete Surface Laptop 4 from the table

DELETE FROM products WHERE name = 'Surface Laptop 4';

1. Find the average price of all products

SELECT AVG(price) FROM products;

1. Get the number of products with a quantity less than or equal to 50

SELECT COUNT(\*) FROM products WHERE quantity 50;

1. show the names and features of all products that are from Apple and have a storage capacity of 256GB

SELECT name, features FROM products WHERE manufacturer = 'Apple' AND features @> '{"storage": "256GB"}';

1. Retrieve the names and reviews of all products that have a review containing the word "amazing"

SELECT name, reviews FROM products WHERE reviews @> '{"Amazing features!"}' OR reviews @> '{"Amazing camera!"}';

Explanation of the codes:

The given code shows the creation of a table named "products" in Cassandra NoSQL database with six columns - product\_id, name, manufacturer, price, quantity, features, and reviews. The primary key is the product\_id column. Five rows are inserted into the table with the INSERT statement, each representing a product with its details.

10 Different queries:

Retrieve all products from the table:

The SELECT statement is used to retrieve all products from the "products" table.

Retrieve all products with a price less than $1000:

The SELECT statement with a WHERE clause is used to retrieve all products from the "products" table with a price less than $1000.

Get all products from a specific manufacturer is Samsung:

The SELECT statement with a WHERE clause is used to retrieve all products from the "products" table where the manufacturer is "Samsung."

Get name and price of the most expensive product:

The SELECT statement with the ORDER BY and LIMIT clauses is used to retrieve the name and price of the most expensive product from the "products" table.

Update the quantity of a specific product (iPhone 13) to a new value (150):

The UPDATE statement with a WHERE clause is used to update the quantity of the "iPhone 13" product to 150.

Delete Surface Laptop 4 from the table:

The DELETE statement with a WHERE clause is used to delete the "Surface Laptop 4" product from the "products" table.

Find the average price of all products:

The SELECT statement with the AVG function is used to find the average price of all products in the "products" table.

Get the number of products with a quantity less than or equal to 50:

The SELECT statement with the COUNT function and a WHERE clause is used to find the number of products in the "products" table with a quantity less than or equal to 50.

Show the names and features of all products that are from Apple and have a storage capacity of 256GB:

The SELECT statement with a WHERE clause and the @> operator is used to find all products in the "products" table that are from "Apple" and have a storage capacity of "256GB." The features column is a map data type, and the @> operator is used to check if the key-value pair exists in the map.

Retrieve the names and reviews of all products that have a review containing the word "amazing":

The SELECT statement with a WHERE clause and the @> operator is used to find all products in the "products" table that have a review containing the word "amazing." The reviews column is a set data type, and the @> operator is used to check if the set contains the given value.

Rationale of the design decision:

The given design decision shows the creation of a table with columns that represent the various attributes of the products. The primary key is the product\_id column, which ensures the uniqueness of each product in the table. The features column is a map data type, which allows us to store multiple key-value pairs for each product. The reviews column is a set data type, which allows us to store multiple reviews for each product. This design decision enables us to store and retrieve the product details efficiently.

Benefits:

* Scalability: Cassandra is designed to scale horizontally, allowing it to handle large amounts of data across multiple servers.
* High Availability: Cassandra has a distributed architecture that provides high availability and fault tolerance, ensuring that data remains available even if some nodes fail.
* Flexibility: Cassandra is schema-free, allowing you to add or modify columns at any time, without needing to change the schema of the entire database.
* Performance: Cassandra is optimized for high write throughput and low-latency read operations, making it ideal for applications that require real-time data processing.
* Replication: Cassandra allows you to replicate data across multiple data centers, providing better data locality and disaster recovery.

Drawbacks

* Complexity: Cassandra has a steep learning curve due to its distributed architecture and configuration requirements. This can make it challenging for developers who are not familiar with distributed systems.
* Querying: Cassandra does not support complex queries and joins like SQL databases, making it challenging to perform ad-hoc queries.
* Storage requirements: Cassandra stores data multiple times for fault tolerance, which can result in higher storage requirements compared to other databases.
* Consistency: Cassandra provides eventual consistency, meaning that data may not be immediately consistent across all nodes. This can lead to some consistency issues, especially in applications that require strong consistency.
* Cost: Cassandra can require significant resources to maintain, such as dedicated servers, storage, and personnel with expertise in distributed systems.