CSP554 - Big Data Technologies
Project Final Report
Health Care Management System Using NoSQL

Team Members:

Bhoomika Panduranga (A20503493)
Narendra Kumar Srivastava (A20529845)
Sanjitha Pathuri (A20524383)
Shadaan Arzeen (A20528043)

Overview:

NoSQL databases are designed to manage large volumes of unstructured, semi-structured, and structured data. They differ from traditional relational databases in their schema-less nature, scalability, flexibility, performance, and distributed setup. NoSQL databases are used in a variety of applications like web and mobile applications, real-time analytics, and IoT, due to their ability to handle substantial data and support real-time processing requirements[1].

Healthcare systems generate an immense amount of heterogeneous data, including patient records, diagnostics, and treatment information. A Healthcare Management System is crucial due to the unique and diverse nature of healthcare data, and demands database solutions that are flexible, scalable, and efficient.

As part of the Healthcare Management System project, we will thoroughly evaluate the performance, suitability, and viability of various NoSQL databases for healthcare data management. We will leverage MongoDB, Cassandra, Amazon DynamoDB in our evaluation. The primary objective is to identify the best NoSQL database for our specific needs and ensure that our data is managed efficiently and securely. By thoroughly evaluating these NoSQL databases, we can gain a better understanding of their strengths and weaknesses, and make an informed decision about which database to use for our data management needs.[2]

We will focus on creating a prototype healthcare system database, comprising tables for Patients, Medical Staff, Appointments, Diagnoses, Medications, Medical Records, Insurance, and Billing. We will implement these databases, document the process, and present a detailed comparative analysis in our report.

SQL NoSQL

Optimized for storage	Optimized for compute
Normalized/relational	Denormalized/hierarchical
Ad hoc queries	Instantiated views
Scale vertically	Scale horizontally
Good for OLAP	Built for OLTP at scale

Goals:

- To practically explore and understand the capabilities of different NoSQL databases
- To create a prototype database structure for healthcare applications in multiple NoSQL platforms.
- To provide an in-depth comparative analysis highlighting the strengths and limitations of each database in the context of healthcare data management.

Methodology:

1. Literature Review: The initial phase involves a detailed review of literature on the distinctive characteristics of NoSQL databases, focusing on their strengths, weaknesses, and optimal use cases.
2. Experimental Implementation: Creating a prototype of a Healthcare Management System database in various NoSQL platforms. This includes creating the databases, defining schema or structure, inserting sample data, and executing common queries. Screenshots, code snippets, and explanations will be documented for each step in the process.
3. Analysis and Comparison: The collected data and observations will be analyzed and compared. Metrics like performance, scalability, query capabilities, ease of use, and cost will be assessed. Charts, graphs, and visual representations will be utilized to effectively convey the comparative analysis.

Search of Literature:

MongoDB:

The open-source MongoDB NoSQL database management system stores data as documents that resemble JSON. Because it is a document-oriented database, as opposed to conventional relational databases, which store data in tables and rows, it saves data in collections of documents. Because of this, MongoDB is a great fit for applications that need to handle and store massive volumes of unstructured data.

The idea of collections and documents forms the foundation of MongoDB's architecture. Like a table in a relational database, a collection is a set of related documents. Documents are made up of pairs of fields and values, just as JSON objects. Documents may include arrays, other documents, and arrays of other documents[3]. MongoDB is an excellent choice for storing and handling complicated data because of its adaptable data model.

MongoDB uses documents that resemble JSON to store data. Accordingly, information is kept in key-value pairs, where a string serves as the key and any kind of JSON data—including characters, numbers, booleans, arrays, and objects—can serve as the value. Because of this, MongoDB's data schema is very user-friendly and adaptable. A wide range of data types, including dates, integers, booleans, texts, arrays, and objects, are also supported by MongoDB. A large range of data can be stored as a result.

Benefits:

- Flexible schema: MongoDB does not require a specified schema, adding new fields and changing existing fields is simple and does not involve altering the structure of the database.
- High performance: MongoDB is a high-performance database that can manage big volumes of data.
- Scalability: MongoDB is easily scalable to accommodate growing amounts of data.
- Replication: MongoDB allows for the replication of data across several servers in order to provide redundancy and high availability.
- Aggregation framework: Complex data analysis is made simple by MongoDB's robust aggregation architecture[3].

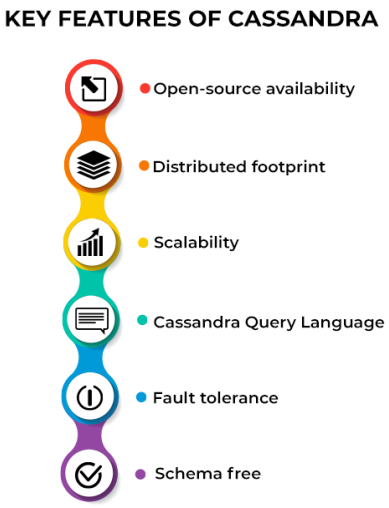
Limitations:

- Lack of maturity compared to relational databases: MongoDB is a very new technology and lacks the maturity of relational databases.
- Restricted join support: Unlike relational databases, MongoDB does not offer the same level of join functionality. This may make querying data dispersed across several collections more challenging.
- Less suitable than relational databases for transactional applications: MongoDB is less suitable than relational databases for transactional applications.

Cassandra:

Apache Cassandra is a NoSQL distributed database developed by Apache designed for large-scale data storage and high availability. It provides high performance, scalability, and fault tolerance, making it an excellent choice for applications that handle large amounts of data, especially in real-time scenarios. Cassandra's distributed architecture enables it to handle data across multiple nodes, ensuring that operations continue even if individual nodes fail.

Cassandra is a ring-topology database. For high availability and fault tolerance, data is stored in columns rather than rows and replicated across multiple nodes. Cassandra employs the Gossip protocol to ensure cluster consistency and CQL (Cassandra Query Language) to query data[4].



Benefits:

- High read and write performance: Cassandra can handle massive volumes of data with ease because of its distributed architecture and columnar storage model.
- Scalability: Cassandra can readily grow by adding more nodes in a horizontal fashion, which enables it to adjust to growing workloads and data volumes without experiencing a drop in performance.
- High availability: Cassandra maintains data integrity and application uptime by utilizing a replication mechanism that makes data available even in the event of a node failure.
- Fault tolerance: Cassandra is extremely resilient to node failures thanks to its distributed architecture, which keeps it running even when nodes are unavailable for maintenance or unanticipated circumstances arise.
- Flexible data storage: A variety of data types, including structured, semi-structured, and unstructured data, can be managed and stored with flexibility thanks to Cassandra's schema-less data model.

Limitations:

- Limited support for joins: Because Cassandra does not support classic SQL joins, developers must use different techniques for modeling complex queries in their data.
- Not appropriate for transactional systems: Cassandra is not meant for systems that need to adhere to strict ACID (Atomicity, Consistency, Isolation, Durability) guarantees.
- Complexity of initial setup and management: Compared to conventional relational databases, setting up and maintaining a Cassandra cluster can be more difficult, requiring knowledge of distributed systems and Cassandra-specific configurations.
- Concerns about data consistency: Although Cassandra's eventual consistency model guarantees eventual consistency of data across nodes, there might be a brief lag before data updates are reflected in all replicas.

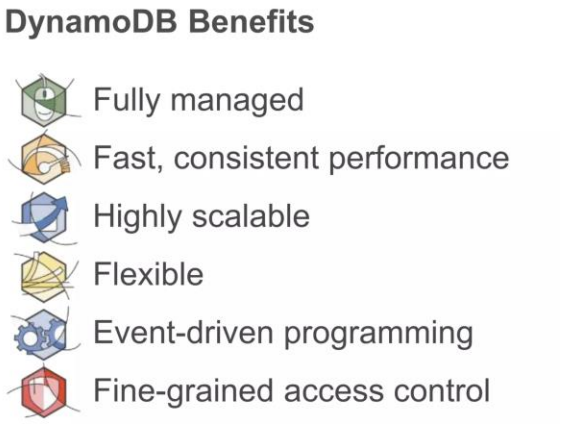
Amazon DynamoDB:

The most popular NoSQL cloud database offered by Amazon AWS Web Development Services is called DynamoDB. For high-performance and scalable applications, DynamoDB is a fully managed NoSQL cloud database platform for data processing, storage, and access. It is intended to provide consistent performance and single-digit millisecond responsiveness at any size. DynamoDB is a document database that uses a peer-to-peer approach that stores documents in key-value format.The data types that DynamoDB supports are as follows: Scalar types, which include Null, String, Binary, Boolean, and Number, are types that have only one value. Multi-valued data types (such as String, Number, and Binary sets) are sets, meaning that each of their values is unique. Complex data structures are represented by document types like lists and maps.

It is a highly resilient, scalable, and economical database built for high throughput, low latency, and volume workloads. Pay-as-you-go technology provided by DynamoDB eliminates the need for upfront software or hardware expenses.

Data Model:

Tables, Items, and Attributes are the three fundamental data model components that DynamoDB uses. Tables are collections of Items, whereas Items are collections of Attributes. In NoSQL databases like dynamoDB, data is stored in tables utilizing the key-value or document paradigm. Tables don't have any linked schemas.Basic information units called attributes are comparable to fields or columns in a relational database. One can have nested (multi-valued) attributes or scalar (one-valued) attributes. Everything may have special qualities all its own. It is not necessary to explain the properties or their types before storing an item. A set of characteristics that distinguishes one item from all others is called an item.Relational database rows are comparable to items. There is no limit to the number of items to be stored in the table. Each item in the table has a unique identifier or primary key.



Benefits:

- Low Latency and High Throughput: DynamoDB is appropriate for applications needing high speed and throughput since it responds quickly, with latency typically less than 10 milliseconds, and it can process millions of requests per second.
- Scalability and High Availability: DynamoDB is built for scalability and high availability. It guarantees data redundancy across several data centers, preventing downtime even in the case of data center or node failures. Because it automatically scales to meet changing workloads, it offers economical and high performance.
- Cost-effectiveness and serverless management are made possible by DynamoDB's pay-as-you-go business model, which lets users only pay for the resources they really utilize. Because AWS is a fully managed service, it takes care of infrastructure and maintenance, freeing developers to concentrate on developing applications rather than maintaining servers.
- Data Replication and Automatic Backup: SSDs are used to store data, and internal replication takes place between several availability zones or regions. For data security, DynamoDB automatically makes backups, which enables users to access data in the cloud. A single table can be horizontally scaled over several servers using the serverless architecture.
- Advanced Functionalities like DynamoDB Streams and TTL: With the help of the Time-To-Live (TTL) feature, expired data can be automatically deleted, saving money on storage and labor-intensive manual data maintenance. Real-time tracking of item-level changes before and after updates is made possible by DynamoDB Streams, which offer a time-ordered list of data updates during the preceding 24 hours.

Limitations:

- Restricted support for joins: Because DynamoDB does not support classic SQL joins, developers must use different techniques for modeling complex queries in data.
- DynamoDB is less appropriate for transactional systems since it is not made for applications that need to adhere to strict ACID (Atomicity, Consistency, Isolation, Durability) guarantees.
- Restricted data types: The data types that DynamoDB can handle are strings, numbers, booleans, lists, and maps. More data modeling might be necessary for complex data structures.
- Learning curve: Developers must pick up new ideas and methods because DynamoDB's data model and query language are different from those of conventional relational databases.
- Because DynamoDB is distributed, monitoring and debugging performance issues can be more complicated than with relational databases.

Comparison:

Feature	MongoDB	Cassandra	DynamoDB
Scalability	High	High	High
High Availability	High	High	High
Performance	High	High	High
Cost-effectiveness	Moderate	Moderate	High
Ease of Use	Moderate	Moderate	High
Data Model	Flexible	Schema-less	Key-value and document
Query Language	JSON-like	CQL	DynamoDB Query Language
Suitable for	Web applications, real-time applications, CMS, mobile applications, IoT	Large-scale data storage, real-time data handling, high-availability, scalable	High-volume workloads, low latency, high throughput

Overall, all three databases are highly scalable, highly available, and performant. However, they have different strengths and weaknesses. MongoDB is a good choice for applications that need a flexible data model, while Cassandra is a good choice for applications that need to store and process large amounts of data. DynamoDB is a good choice for applications that require low latency and high throughput.[4], [5]

Experimental Implementation:

We created a prototype of a Healthcare Management System database in MongoDB and Cassandra NoSQL platforms. This includes creating the databases, defining schema or structure, inserting sample data, and executing common queries. Screenshots, code snippets, and explanations are documented for each step in the process.

Database Design:

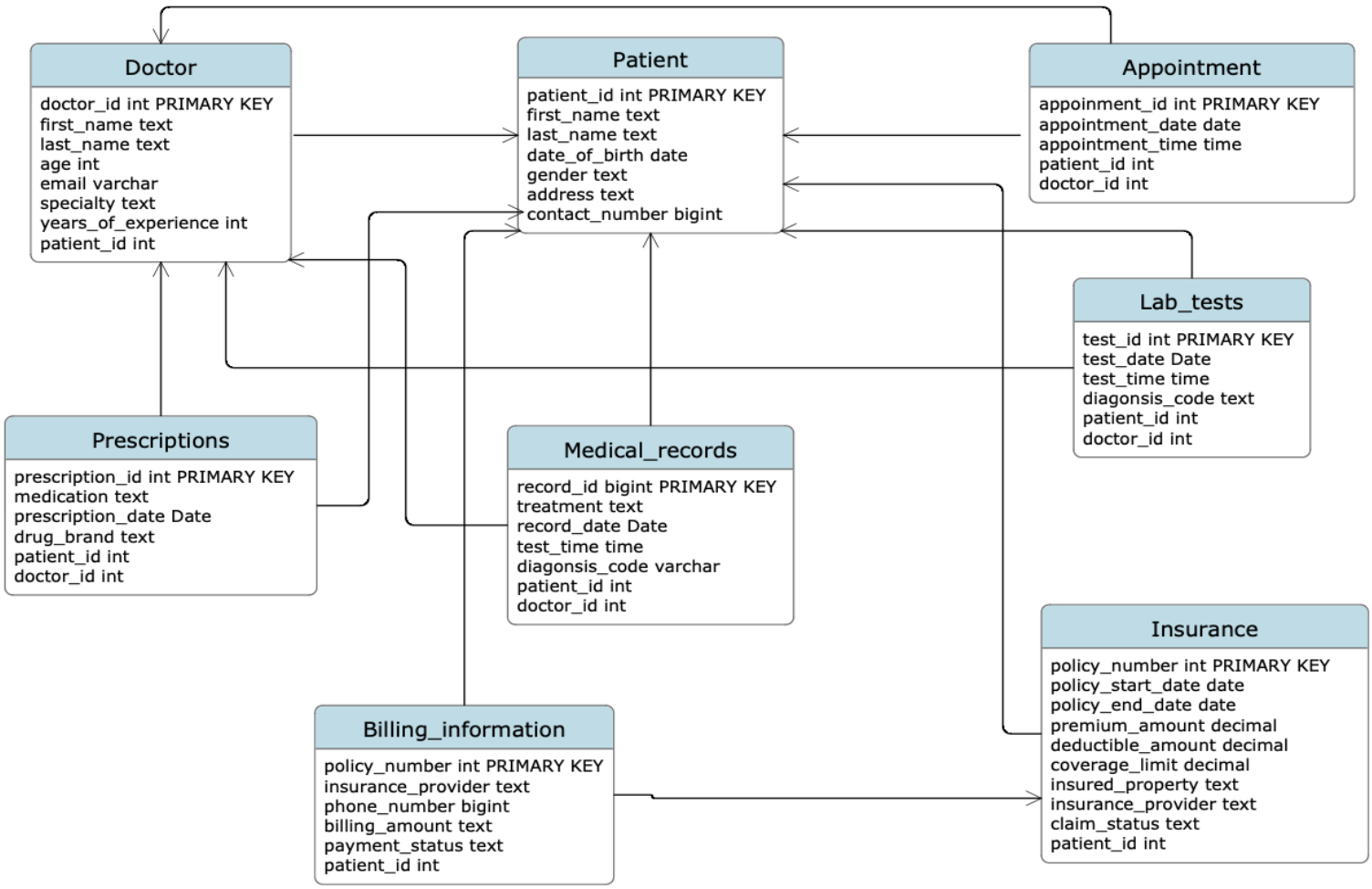
Building a healthcare database requires careful consideration of the information we need to store and how it all relates. Here are some essential tables that we included for our experimental Implementation of NoSQL databases:

Tables:

- Patient
- Doctor
- Appointment
- Prescriptions
- Medical records
- Lab tests
- Billing information
- Insurance

UML Diagram:

Below is the UML Model of the database design we used for our experimental implementation:



Showing the relationships between the tables.

MongoDB:

Environment Setup:

Accessing MongoDB:

- MongoDB Shell:
  - Install MongoDB on the local system.
  - Start the MongoDB server using the `mongod` command.
  - Connect to the server using the MongoDB shell (`mongo` command).
- MongoDB Drivers:
  - Choose a MongoDB driver based on the programming language (e.g., pymongo for Python, mongo-java-driver for Java).
  - Install the driver using package management tools (pip, Maven, etc.).
  - Connect to MongoDB using the driver in application code.
- Third-party Tools:
  - Tools like MongoDB Compass provide a graphical user interface for interacting with MongoDB.
  - Download and install MongoDB Compass.
  - Connect to MongoDB instances and manage databases visually.

Using MongoDB Shell:

- Setup:
  - Download MongoDB Community Server from the official MongoDB website.
  - Install the server and start it using the `mongod` command.
  - Open another terminal and connect to the server using the `mongo` command.
- Configuration:
  - MongoDB's configuration file is typically found at `/etc/mongod.conf` (Linux) or `C:\Program Files\MongoDB\Server\<version>\bin\mongod.cfg` (Windows).
  - Configure settings such as `dbPath`, `port`, and `logPath`.
- Database and Collection Creation:
  - Use the `use <database_name>` command to create a new database.
  - Collections are created implicitly when inserting data.
- Data Import:
  - Use the `mongoimport` command to import data from JSON, CSV, or BSON files.
  - Example: `mongoimport --db mydatabase --collection mycollection --file data.json`
- Query Executions:
  - Use commands like `db.<collection>.find()`, `db.<collection>.insert()`, `db.<collection>.update()`, and `db.<collection>.remove()` for basic operations.

Using MongoDB Drivers:

- Choose the Driver:
  - MongoDB provides official drivers for various programming languages.
  - Install the driver based on the application's language and framework.
- Connect to MongoDB:
  - In application code, establish a connection to MongoDB using the driver.
    Example (pymongo in Python):

```python
from pymongo import MongoClient

client = MongoClient('localhost', 27017)
db = client['mydatabase']
```

- Perform CRUD Operations:
  - Use driver-specific methods for performing CRUD operations.
    Example (pymongo):

```python
# Insert
db.mycollection.insert_one({"name": "John Doe", "age": 30})

# Query
result = db.mycollection.find({"name": "John Doe"})

# Update
db.mycollection.update_one({"name": "John Doe"}, {"$set": {"age": 31}})

# Delete
db.mycollection.delete_one({"name": "John Doe"})
```

Performance Analysis (MongoDB):

- Monitor with MongoDB Atlas or Ops Manager for real-time insights.
- Leverage MongoDB's profiling and logging features.
- Utilize third-party monitoring tools like MMS (MongoDB Management Service), Prometheus, or Grafana.
- Consider factors like index usage, query patterns, and hardware specifications for performance analysis.

Cassandra:

Environment Setup:

Accessing Cassandra:

- CQLSH (Cassandra Query Language Shell):
    - Install Cassandra on the local system.
    - Start the Cassandra server using the `cassandra` command.
    - Connect to Cassandra using CQLSH (`cqlsh` command).
- Cassandra Drivers:
    - Choose a Cassandra driver based on the programming language (e.g., DataStax Java Driver for Java, DataStax Python Driver for Python).
    - Install the driver using package management tools.
    - Connect to Cassandra using the driver in application code.
- Third-party Tools:
    - DBeaver, DataStax DevCenter, or other tools for visual interaction with Cassandra databases.

Using CQLSH:

- Setup:
    - Download and install Apache Cassandra from the official website.
    - Start the Cassandra server using the `cassandra` command.
    - Open another terminal and connect to Cassandra using `cqlsh`.
- Configuration:
    - Configuration settings are found in the `cassandra.yaml` file.
    - Configure settings such as `data_file_directories`, `cluster_name`, and `listen_address`.
- Database and Table Creation:
    - Use CQL commands (`CREATE KEYSPACE`, `CREATE TABLE`) for database and table creation.
- Data Import:
    - Use tools like `cqlsh COPY` or DataStax Bulk Loader (`dsbulk`) for importing data.
- Query Executions:
    - Use CQL (`SELECT`, `INSERT`, `UPDATE`, `DELETE`) for basic operations.

Using Cassandra Drivers:

- Choose the Driver:
    - Cassandra provides official drivers for various programming languages.
    - Install the driver based on the application's language and framework.
- Connect to Cassandra:

- In application code, establish a connection to Cassandra using the driver.
  Example (DataStax Java Driver):

```java
Cluster cluster = Cluster.builder().addContactPoint("localhost").build();
Session session = cluster.connect("mykeyspace");
```

- Perform CRUD Operations:
  - Use driver-specific methods for performing CRUD operations.
    Example (DataStax Java Driver):

```java
// Insert
session.execute("INSERT INTO mytable (id, name) VALUES (?, ?)", UUID.randomUUID(), "John Doe");

// Query
ResultSet result = session.execute("SELECT * FROM mytable WHERE name = 'John Doe'");

// Update
session.execute("UPDATE mytable SET age = 31 WHERE name = 'John Doe'");

// Delete
session.execute("DELETE FROM mytable WHERE name = 'John Doe'");
```

<u>Performance Analysis (Cassandra):</u>

- Monitor tools like DataStax OpsCenter for real-time insights.
- Use JMX metrics and nodetool commands for detailed performance analysis.
- Utilize third-party tools like Prometheus and Grafana for extended monitoring.
- Consider factors like compaction, tombstones, and read/write patterns for performance analysis.

Note: Detailed performance analysis may involve advanced topics such as schema design, partitioning, and optimization based on specific use cases.

IMPLEMENTATION:

MongoDB:

Configuration/Installation:

Mongodb install Mongoshell install Added mongoshell to path

```
Please enter a MongoDB connection string (Default: mongodb://localhost/): mongosh
mongosh
Current Mongosh Log ID: 656961987172657fb95964b2
Connecting to:          mongodb://127.0.0.1:27017/mongosh?directConnection=true&serverSelectionTimeoutMS=2000&appName=mo
ngosh+2.1.0
Using MongoDB:          7.0.4
Using Mongosh:          2.1.0

For mongosh info see: https://docs.mongodb.com/mongodb-shell/

------
   The server generated these startup warnings when booting
   2023-11-29T10:13:59.997-06:00: Access control is not enabled for the database. Read and write access to data and conf
iguration is unrestricted
------

mongosh>
```

```
mongosh> show dbs
admin     40.00 KiB
config   108.00 KiB
local     72.00 KiB
mongosh> use healthcare
switched to db healthcare
```

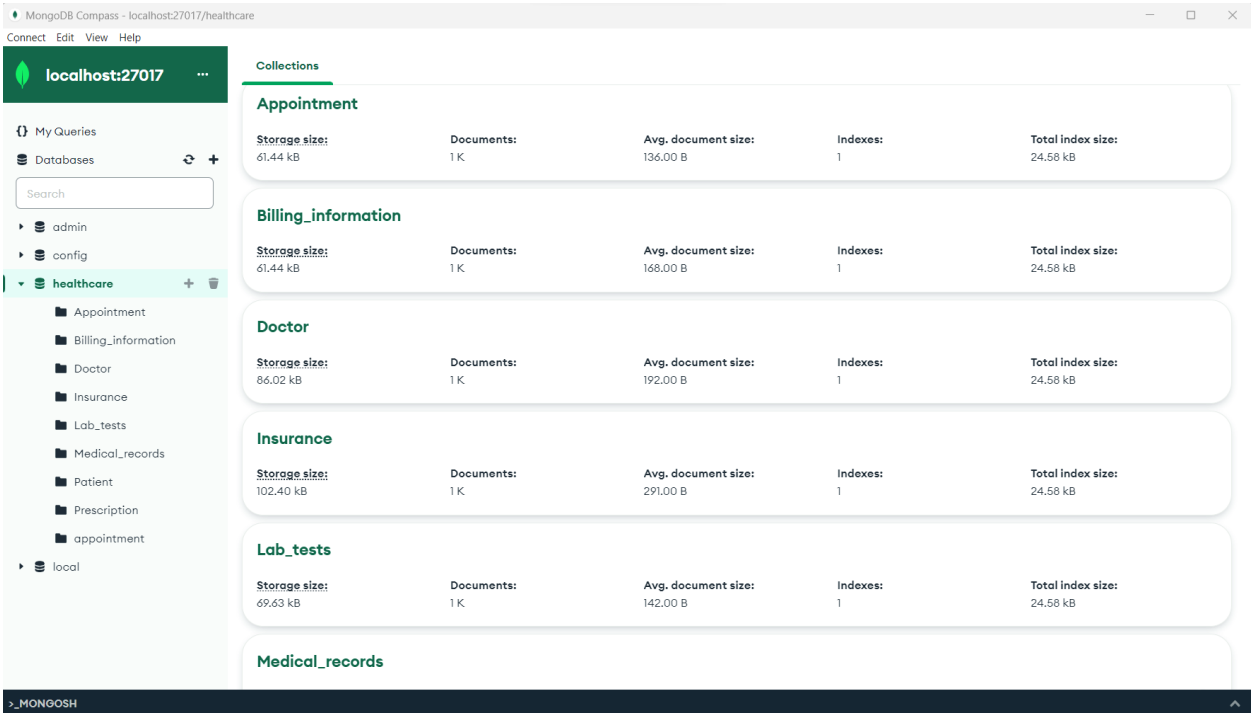
Measuring how long it takes to load the data(data is randomly generated).

```
healthcare> load("C:\\Users\\shada\\Downloads\\Appointment_insertion.js")
Time taken for insertion: 60 milliseconds
true
healthcare> db.appointment.createIndex({ "appointment_id": 1 }, { unique: true });
appointment_id_1
healthcare> load("C:\\Users\\shada\\Downloads\\Billing_information_insertion.js")
Time taken for insertion: 52 milliseconds
true
healthcare> load("C:\\Users\\shada\\Downloads\\Doctor_insertion.js")
Time taken for insertion: 45 milliseconds
true
healthcare> db.appointment.createIndex({ "doctor_id": 1 }, { unique: true });
doctor_id_1
healthcare> load("C:\\Users\\shada\\Downloads\\Insurance_insertion.js")
Time taken for insertion: 44 milliseconds
true
healthcare> db.appointment.createIndex({ "policy_number": 1 }, { unique: true });
policy_number_1
healthcare> load("C:\\Users\\shada\\Downloads\\Lab_tests_insertion.js")
Time taken for insertion: 37 milliseconds
true
healthcare> load("C:\\Users\\shada\\Downloads\\Medical_records_insertion.js")
Time taken for insertion: 36 milliseconds
true
healthcare> load("C:\\Users\\shada\\Downloads\\Patient_insertion.js")
Time taken for insertion: 39 milliseconds
true
healthcare> db.appointment.createIndex({ "patient_id": 1 }, { unique: true });
patient_id_1
healthcare> load("C:\\Users\\shada\\Downloads\\Prescriptions_insertion.js")
Error: ENOENT: no such file or directory, open 'C:\Users\shada\Downloads\Prescriptions_insertion.js'
healthcare> load("C:\\Users\\shada\\Downloads\\Prescriptions_insertion.js")
Time taken for insertion: 43 milliseconds
true
healthcare> db.appointment.createIndex({ "prescription_id": 1 }, { unique: true });
prescription_id_1
healthcare>
```

Creating index for prescription

```
healthcare> load("C:\\Users\\shada\\Downloads\\index_creation_time.js")
Time taken for index creation: 5 milliseconds
true
```

Created db with all the required mongodb collections and view the database in mongodb compass



Measuring how long does it take to update some fields of the data,

```
healthcare> load("C:\\Users\\shada\\Downloads\\update_single.js")
Time taken for single record updation: 12 milliseconds
true
healthcare>
```

Last_name was updated to keppler

```
_id: ObjectId('656963fb7172657fb9597c23')
patient_id: 619486079
first_name: "Jerrome"
last_name: "Keppler"
date_of_birth: "2020-08-17"
gender: "Male"
address: "2 Delaware Plaza"
contact_number: 4336373923
```

Measuring how long does it take to add a single new record, or delete a record?

```
healthcare> load("C:\\Users\\shada\\Downloads\\single_pres_rec.js")
Time taken for single record insertion: 4 milliseconds
true
healthcare>
```

Measuring how long does it take to add a new column? And similar operations.

Alter table

ALTER TABLE Healthcare Appointment

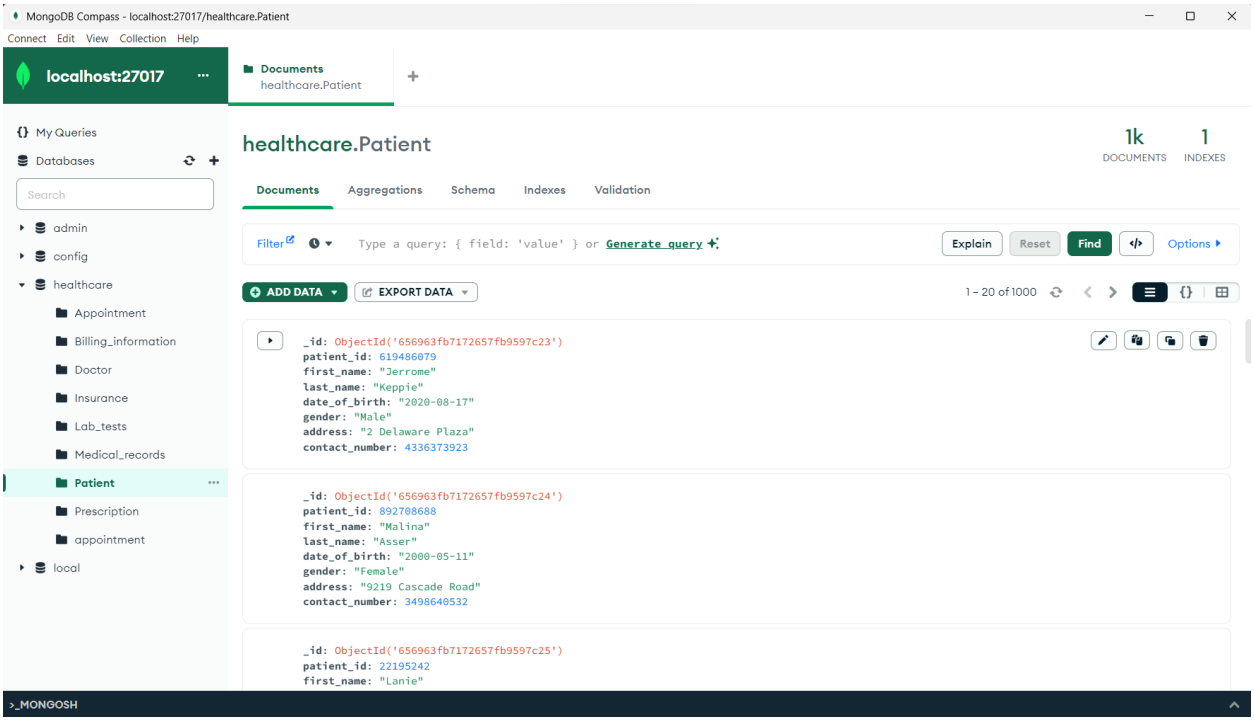
[ALTER column_name TYPE cql_type]

[ADD (column_definition_list)]

[DROP column_list | COMPACT STORAGE]

[RENAME column_name TO column_name]

[WITH table_properties];




```
healthcare> load("C:\\Users\\shada\\Downloads\\add_col.js")
Time taken for adding a new column: 73 milliseconds
true
healthcare>
```

New field added

```
  _id: ObjectId('656963fb7172657fb9597c23')
  patient_id: 619486079
  first_name: "Jerrome"
  last_name: "Keppie"
  date_of_birth: "2020-08-17"
  gender: "Male"
  address: "2 Delaware Plaza"
  contact_number: 4336373923
  new_field: "default_value"
```

```
healthcare> load("C:\\Users\\shada\\Downloads\\remove_query.js")
Time taken for deleting a column : 82 milliseconds
true
healthcare>
```

New field dropped

```
  _id: ObjectId('656963fb7172657fb9597c23')
  patient_id: 619486079
  first_name: "Jerrome"
  last_name: "Keppie"
  date_of_birth: "2020-08-17"
  gender: "Male"
  address: "2 Delaware Plaza"
  contact_number: 4336373923
```

Forming several different queries both simple and complex. To measure How long does it take to execute each query? Try using an index (if one is available for a database) to see if that helps.

simple queries:

1) patients with Cigna insurance provider SELECT * FROM Billing_Information Where insurance_provider='CIGNA'

```
healthcare> load("C:\\Users\\shada\\Downloads\\simple_query_bi.js")
Query Results:
[
  {
    _id: ObjectId('656962467172657fb959689c'),
    policy_number: 290575,
    phone_number: 4578859079,
    insurance_provider: 'Cigna',
    billing_amount: 'Yuan Renminbi',
    payment_status: 'Paid',
    patient_id: 892708688
  },
  {
    _id: ObjectId('656962467172657fb959689d'),
    policy_number: 745381,
    phone_number: 3178531339,
    insurance_provider: 'Cigna',
    billing_amount: 'Naira',
    payment_status: 'Paid',
    patient_id: 22195242
  },
  {
    _id: ObjectId('656962467172657fb95968a3'),
    policy_number: 635043,
    phone_number: 8937113816,
    insurance_provider: 'Cigna',
    billing_amount: 'Peso',
    payment_status: 'Partial',
    patient_id: 801335117
  },
```

2) doctors working for cardiology

```
healthcare> load("C:\\Users\\shada\\Downloads\\simple_query_doc.js")
Query Results:
[
  {
    _id: ObjectId('656962c47172657fb9596c83'),
    doctor_id: 11302,
    first_name: 'Aurilia',
    last_name: 'Mailey',
    age: 66,
    email: 'amailey0@blogspot.com',
    specialty: 'Cardiology',
    years_of_experience: 7,
    patient_id: 619486079
  },
  {
    _id: ObjectId('656962c47172657fb9596c86'),
    doctor_id: 11305,
    first_name: 'Marjorie',
    last_name: 'Stollenberg',
    age: 34,
    email: 'mstollenberg3@marriott.com',
    specialty: 'Cardiology',
    years_of_experience: 39,
    patient_id: 668808523
  },
  {
    _id: ObjectId('656962c47172657fb959705a'),
    doctor_id: 11535,
    first_name: 'Ibrahim',
    last_name: 'Rylstone',
    age: 28,
    email: 'irylstone6h@hao123.com',
    specialty: 'Cardiology',
    years_of_experience: 9,
    patient_id: 14003406
  }
]
Query time: 45 milliseconds
true
healthcare>
```

complex queries:

1) DISTINCT

```
healthcare> load("C:\\Users\\shada\\Downloads\\distinct_query.js")
Query Results:
[
  'Cardiology',
  'Dermatology',
  'Gastroenterology',
  'Neurology',
  'Oncology',
  'Pediatrics',
  'Psychiatry',
  'Surgery'
]
Query time: 5 milliseconds
true
healthcare>
```

2) SUM

```
healthcare> load("C:\\Users\\shada\\Downloads\\count_query.js")
Query Results:
341
Query time: 3 milliseconds
true
healthcare>
```

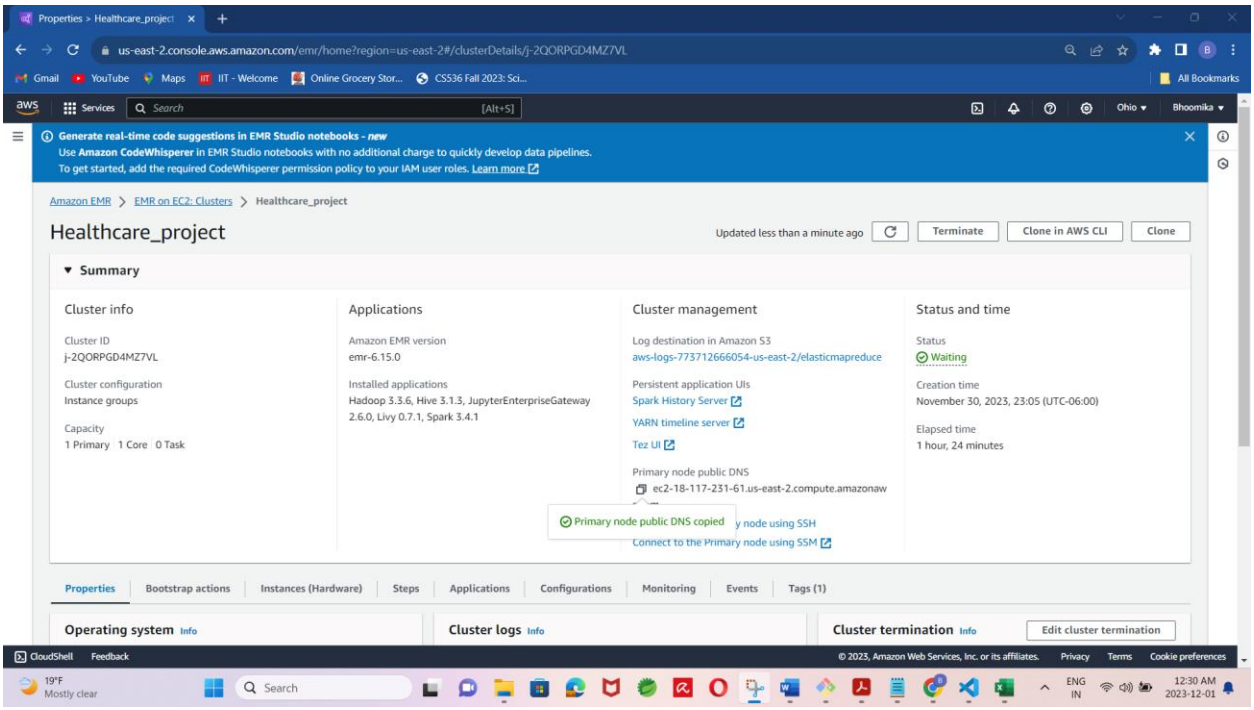
3) AVG

```
healthcare> load("C:\\Users\\shada\\Downloads\\avg_query.js")
Average Premium Amount: 5059.474190000001
Query time: 11 milliseconds
true
healthcare>
```

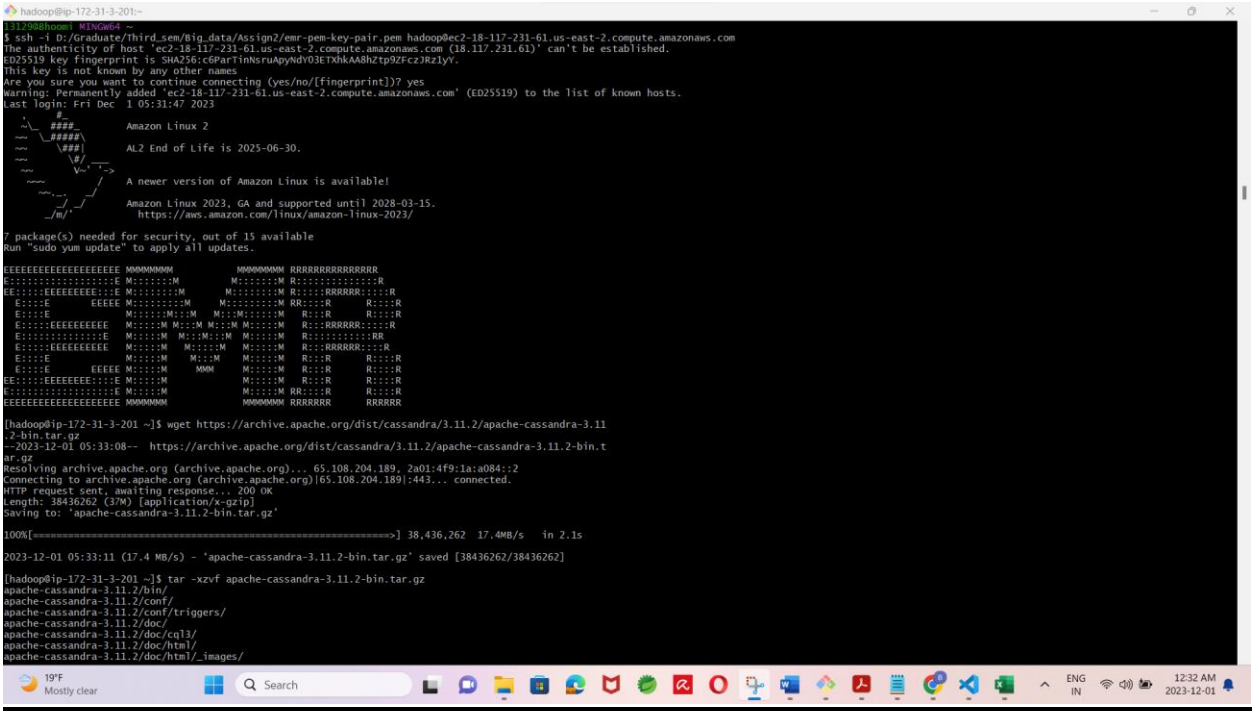
4) MAX

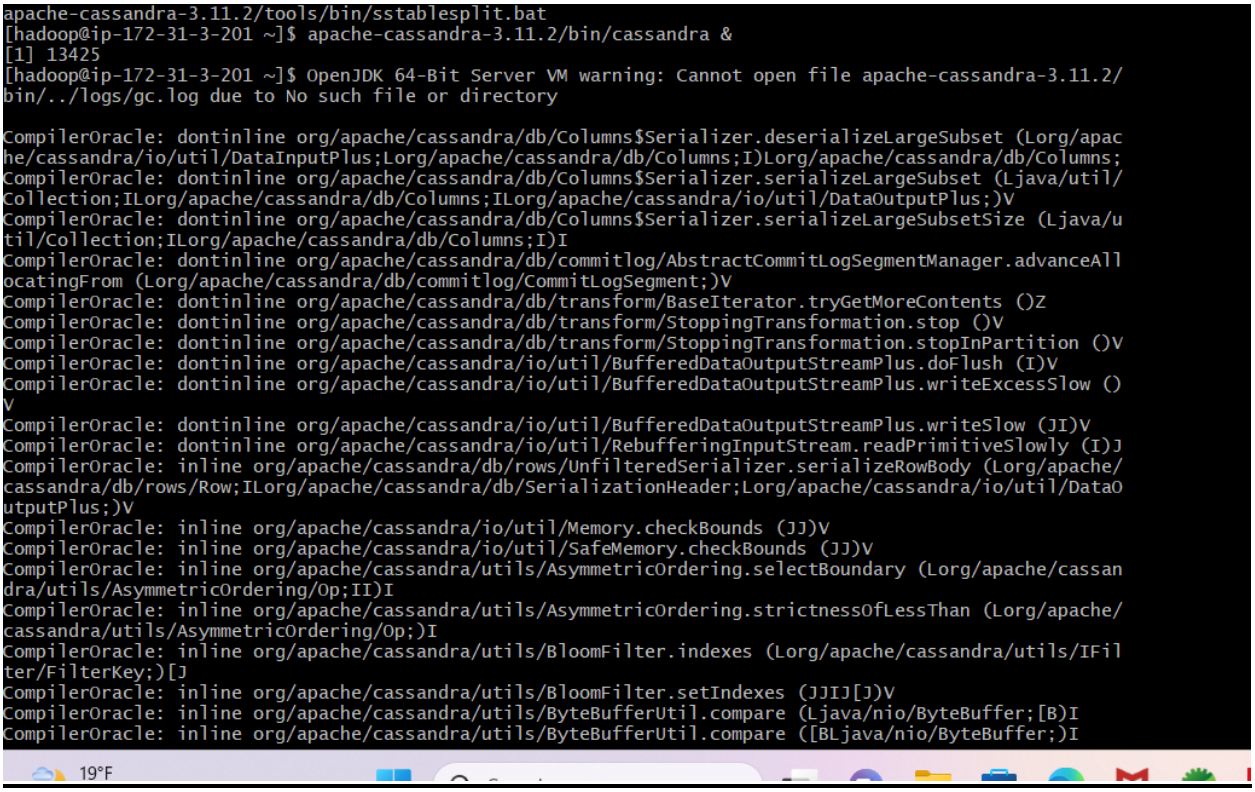
```
healthcare> load("C:\\Users\\shada\\Downloads\\max_query.js")
Max Coverage Limit: 999330.35
Query time: 9 milliseconds
true
healthcare>
```

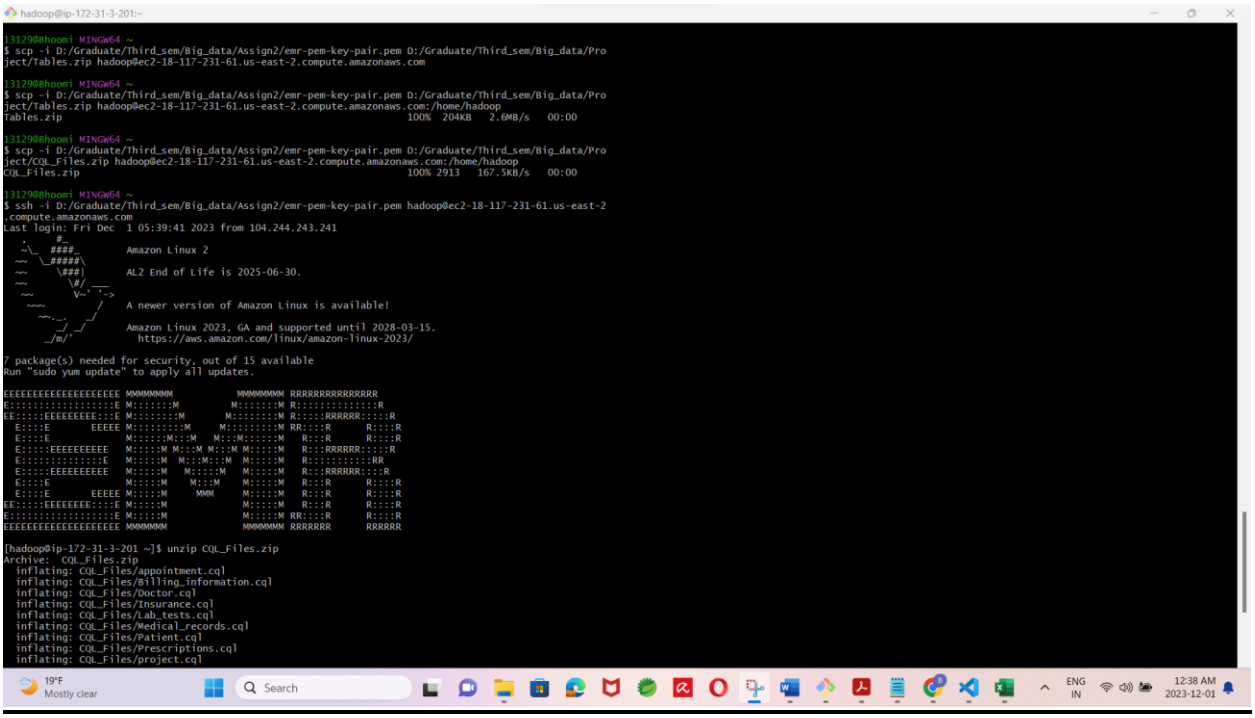
Cassandra: setting up EMR Cluster



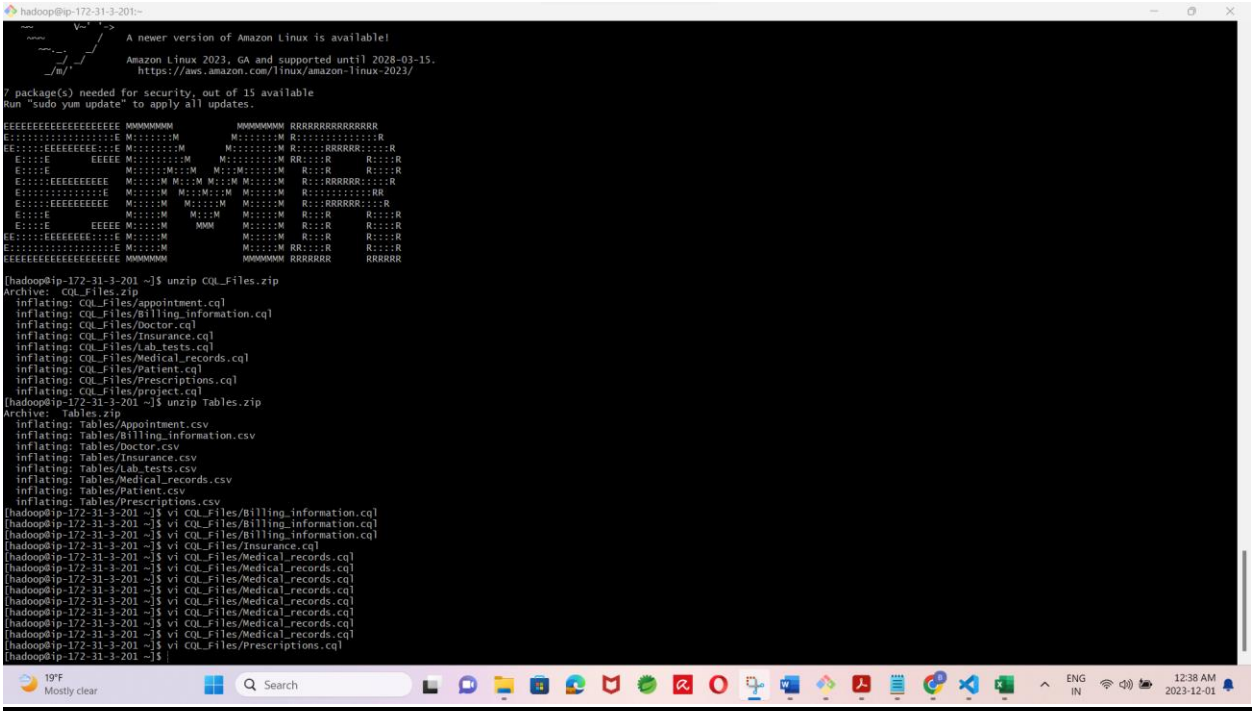
Installation/Configuration:



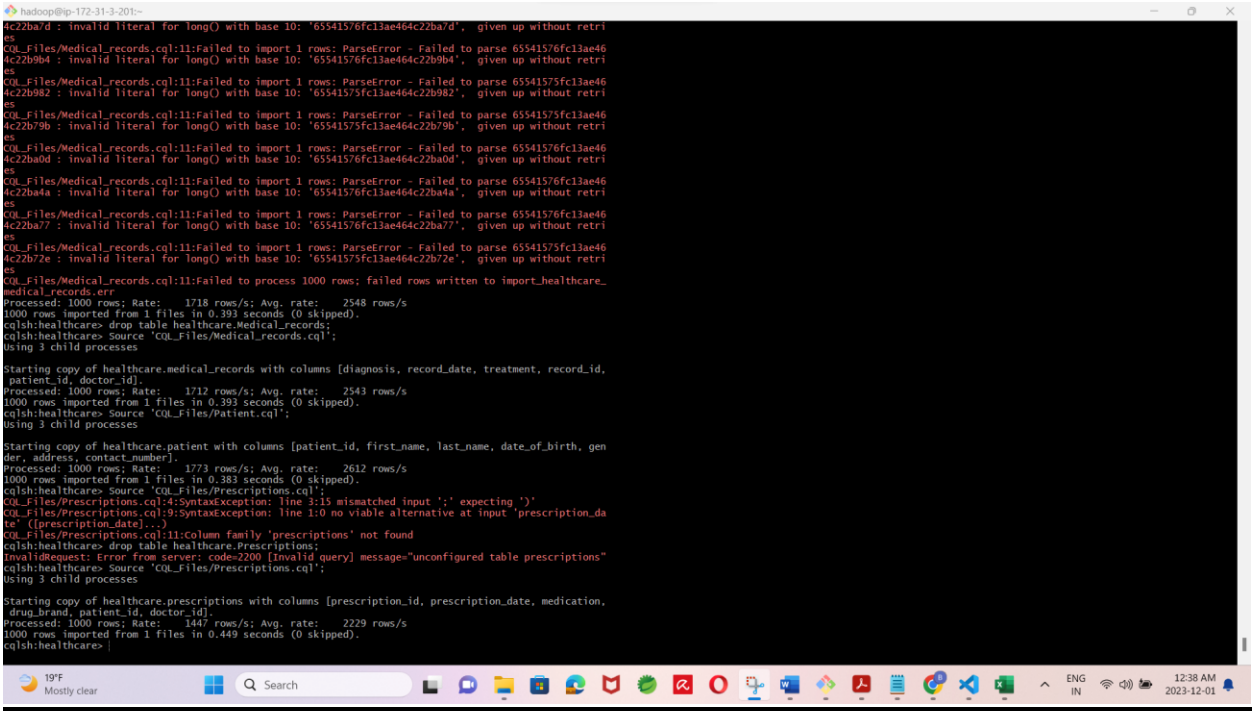


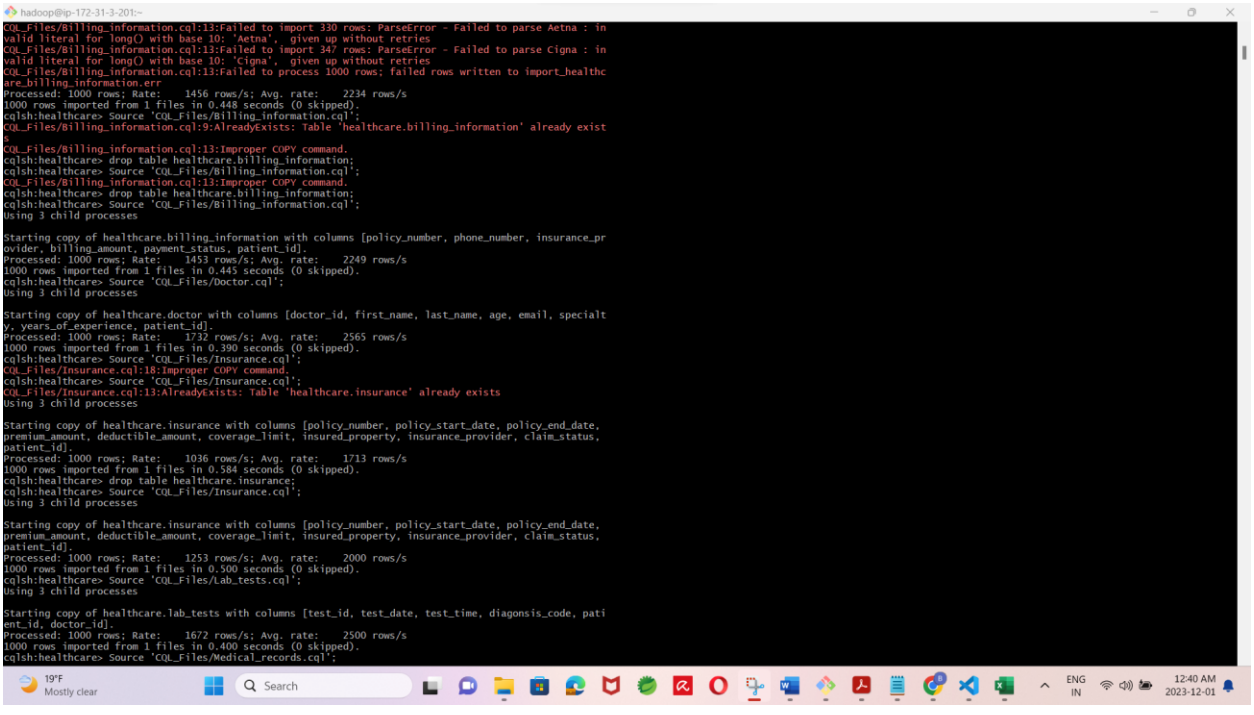


Importing cql scripts and data(csv) files:



Running cql scripts for table creation and data imports from csv files:






hadoop@ip-172-31-3-201:~
Last login: Fri Dec 1 05:32:05 2023 from 104.244.243.241
Amazon Linux 2
AL2 End of Life is 2025-06-30.
A newer version of Amazon Linux is available!
Amazon Linux 2023, GA and supported until 2028-03-15.
https://aws.amazon.com/linux/amazon-linux-2023/
7 package(s) needed for security, out of 15 available
Run "sudo yum update" to apply all updates.
[hadoop@ip-172-31-3-201 ~]$ apache-cassandra-3.11.2/bin/cqlsh
Connected to Test Cluster at 127.0.0.1:9042.
[cqlsh 5.0.1 | Cassandra 3.11.2 | CQL spec 3.4.4 | Native protocol v4]
Use HELP for help.
cqlsh> Source 'CQL_Files/appointment.cql';
CQL_Files/appointment.cql:8:ConfigurationException: Keyspace healthcare doesn't exist
CQL_Files/appointment.cql:10:Keyspace 'healthcare' not found.
cqlsh> Source 'CQL_Files/project.cql';
CQL_Files/project.cql:3:SyntaxException: line 2:0 mismatched input 'describe' expecting EOF (..., 'repl
[describe]...)' : 1 }
CQL_Files/project.cql:4:InvalidRequest: Error from server: code=2200 [Invalid query] message="Keyspace
'healthcare' does not exist"
cqlsh> CREATE KEYSPACE Healthcare WITH REPLICATION = { 'class' : 'SimpleStrategy', 'replication_factor'
: 1 }
... ;
cqlsh> describe keyspaces;
system_schema system healthcare
system_auth system_distributed system_traces
cqlsh> use healthcare
... ;
cqlsh:healthcare> Source 'CQL_Files/appointment.cql';
Using 3 child processes
Starting copy of healthcare.appointment with columns [appoinment_id, appointment_date, appointment_time
, patient_id, doctor_id].
Processed: 1000 rows; Rate: 1007 rows/s; Avg. rate: 1674 rows/s
1000 rows imported from 1 files in 0.598 seconds (0 skipped).
19°F
Mostly clear
Search
ENG
IN
12:41 AM
2023-12-01


hadoop@ip-172-31-3-201:~
131298Bhoomi MINGW64 ~
$ ssh -i D:/Graduate/Third_sem/Big_data/Assign2/emr-pem-key-pair.pem hadoop@ec2-
18-191-202-157.us-east-2.compute.amazonaws.com
Last login: Thu Nov 30 23:56:08 2023 from 104.244.243.241
Amazon Linux 2
AL2 End of Life is 2025-06-30.
A newer version of Amazon Linux is available!
Amazon Linux 2023, GA and supported until 2028-03-15.
https://aws.amazon.com/linux/amazon-linux-2023/
7 package(s) needed for security, out of 15 available
Run "sudo yum update" to apply all updates.
[hadoop@ip-172-31-4-95 ~]$ apache-cassandra-3.11.2/bin/cqlsh
Connected to Test Cluster at 127.0.0.1:9042.
[cqlsh 5.0.1 | Cassandra 3.11.2 | CQL spec 3.4.4 | Native protocol v4]
Use HELP for help.
cqlsh> Source 'CQL_Files/appoinment.cql';
system_schema system healthcare
system_auth system_distributed system_traces
Using 3 child processes
Starting copy of healthcare.appointment with columns [appoinment_id, appointment_date, appointment_time
, patient_id, doctor_id].
CQL_Files/appoinment.cql:14:Failed to import 0 rows: IOError - Can't open 'Tables/Appointment.csv' for
reading: no matching file found, given up after 1 attempts
Processed: 0 rows; Rate: 0 rows/s; Avg. rate: 0 rows/s
0 rows imported from 0 files in 0.213 seconds (0 skipped).
cqlsh> Connection to ec2-18-191-202-157.us-east-2.compute.amazonaws.com closed by remote host.
Connection to ec2-18-191-202-157.us-east-2.compute.amazonaws.com closed.
131298Bhoomi MINGW64 ~
$ ssh -i D:/Graduate/Third_sem/Big_data/Assign2/emr-pem-key-pair.pem hadoop@ec2-18-117-231-61.us-east-2
.compute.amazonaws.com
Last login: Fri Dec 1 05:32:05 2023 from 104.244.243.241
19°F
Mostly clear
Search
ENG
IN
12:41 AM
2023-12-01

Simple queries:

select

```
cqlsh:healthcare> select * from healthcare.Patient where first_name='Goddart'ALLOW FILTERING;

 patient_id | address       | contact_number | date_of_birth | first_name | gender | last_name
------------+---------------+----------------+---------------+------------+--------+-----------
  412618858 | 1 Rigney Road |     3844416249 |    2015-08-23 |    Goddart |   Male |     Rosso

(1 rows)

Tracing session: ed805e70-9015-11ee-abc7-51c9c445252e

 activity
                    | timestamp                  | source    | source_elapsed | client
-----------------------------------------------------------------------------------------------------------------
----------------------+----------------------------+-----------+----------------+-----------

   Execute CQL3 query | 2023-12-01 06:50:34.841000 | 127.0.0.1 |              0 | 127.0.0.1
          Parsing select * from healthcare.Patient where first_name='Goddart'ALLOW FILTERING; [Native-
Transport-Requests-1] | 2023-12-01 06:50:34.844000 | 127.0.0.1 |           3668 | 127.0.0.1
                                                                                  Preparing statement [Native-
Transport-Requests-1] | 2023-12-01 06:50:34.844000 | 127.0.0.1 |           3921 | 127.0.0.1
                                                                             Computing ranges to query [Native-
Transport-Requests-1] | 2023-12-01 06:50:34.845000 | 127.0.0.1 |           4161 | 127.0.0.1
 Submitting range requests on 257 ranges with a concurrency of 1 (0.0 rows per range expected) [Native-
Transport-Requests-1] | 2023-12-01 06:50:34.845000 | 127.0.0.1 |           4487 | 127.0.0.1
                                                                           Submitted 1 concurrent range requests [Native-
Transport-Requests-1] | 2023-12-01 06:50:34.846000 | 127.0.0.1 |           5170 | 127.0.0.1
              Executing seq scan across 0 sstables for (min(-9223372036854775808), min(-92233720368547
75808)] [ReadStage-2] | 2023-12-01 06:50:34.846000 | 127.0.0.1 |           5311 | 127.0.0.1
                                                                          Read 1000 live rows and 0 tombston
e cells [ReadStage-2] | 2023-12-01 06:50:34.852000 | 127.0.0.1 |          11423 | 127.0.0.1

     Request complete | 2023-12-01 06:50:34.852744 | 127.0.0.1 |          11744 | 127.0.0.1


cqlsh:healthcare>
```

```
cqlsh:healthcare> select * from doctor where specialty='cardiology' Allow Filtering;

 doctor_id | age | email | first_name | last_name | patient_id | specialty | years_of_experience
-----------+-----+-------+------------+-----------+------------+-----------+---------------------

(0 rows)

Tracing session: 3e0e7840-9016-11ee-abc7-51c9c445252e

 activity
                        | timestamp                  | source    | source_elapsed | client
-----------------------------------------------------------------------------------------------------------------
-------------------------+----------------------------+-----------+----------------+-----------

     Execute CQL3 query | 2023-12-01 06:52:49.988000 | 127.0.0.1 |              0 | 127.0.0.1
                  Parsing select * from doctor where specialty='cardiology' Allow Filtering; [Nativ
e-Transport-Requests-1] | 2023-12-01 06:52:49.988000 | 127.0.0.1 |            266 | 127.0.0.1
                                                                                     Preparing statement [Nativ
e-Transport-Requests-1] | 2023-12-01 06:52:49.988000 | 127.0.0.1 |            455 | 127.0.0.1
                                                                                Computing ranges to query [Nativ
e-Transport-Requests-1] | 2023-12-01 06:52:49.988000 | 127.0.0.1 |            737 | 127.0.0.1
 Submitting range requests on 257 ranges with a concurrency of 75 (1.35 rows per range expected) [Nativ
e-Transport-Requests-1] | 2023-12-01 06:52:49.989000 | 127.0.0.1 |           1245 | 127.0.0.1
                                                                              Submitted 1 concurrent range requests [Nativ
e-Transport-Requests-1] | 2023-12-01 06:52:49.990000 | 127.0.0.1 |           1971 | 127.0.0.1
                   Executing seq scan across 1 sstables for (min(-9223372036854775808), min(-922337203685
4775808)] [ReadStage-3] | 2023-12-01 06:52:49.991000 | 127.0.0.1 |           3400 | 127.0.0.1
                                                                             Read 250 live rows and 0 tombst
one cells [ReadStage-3] | 2023-12-01 06:52:49.995000 | 127.0.0.1 |           7779 | 127.0.0.1

       Request complete | 2023-12-01 06:52:49.995994 | 127.0.0.1 |           7994 | 127.0.0.1


cqlsh:healthcare>
```

```
(250 rows)

Tracing session: 7d3d8c90-9016-11ee-abc7-51c9c445252e

 activity
          | timestamp                  | source    | source_elapsed | client
-----------------------------------------------------------------------------------------------------------------
----------+----------------------------+-----------+----------------+-----------
                                                                                                          Execute
CQL3 query | 2023-12-01 06:54:35.994000 | 127.0.0.1 |              0 | 127.0.0.1
                                                                                 Parsing select * from doctor; [Native-Transport-
equests-1] | 2023-12-01 06:54:35.994000 | 127.0.0.1 |            317 | 127.0.0.1
                                                                                       Preparing statement [Native-Transport-
equests-1] | 2023-12-01 06:54:35.994000 | 127.0.0.1 |            449 | 127.0.0.1
                                                                                    Computing ranges to query [Native-Transport-
equests-1] | 2023-12-01 06:54:35.994000 | 127.0.0.1 |            652 | 127.0.0.1
 Submitting range requests on 257 ranges with a concurrency of 75 (1.35 rows per range expected) [Native-Transport-
equests-1] | 2023-12-01 06:54:35.995000 | 127.0.0.1 |           1072 | 127.0.0.1
                                                                                 Submitted 1 concurrent range requests [Native-Transport-
equests-1] | 2023-12-01 06:54:35.995000 | 127.0.0.1 |           1878 | 127.0.0.1
                  Executing seq scan across 1 sstables for (min(-9223372036854775808), min(-9223372036854775808)] [R
eadStage-2] | 2023-12-01 06:54:35.996000 | 127.0.0.1 |           2109 | 127.0.0.1
                                                                                  Read 100 live rows and 0 tombstone cells [R
eadStage-2] | 2023-12-01 06:54:35.998000 | 127.0.0.1 |           4716 | 127.0.0.1
                                                                                                          Reque
st complete | 2023-12-01 06:54:36.000205 | 127.0.0.1 |           6205 | 127.0.0.1
```

```
cqlsh:healthcare> select * from doctor where specialty='Cardiology' Allow Filtering;

 doctor_id | age | email                          | first_name     | last_name   | patient_id | specialty  | years_of_
experience
-----------+-----+--------------------------------+----------------+-------------+------------+------------+----------
-----------
     11464 |  67 |          zballoch4i@cnet.com |           Zack |     Balloch |  819407643 | Cardiology |
        37
     11321 |  44 |      ofennellyj@sakura.ne.jp |           Olva |    Fennelly |  390924633 | Cardiology |
        39
     11522 |  62 |           csofe64@utexas.edu |          Catha |        Sofe |  937545077 | Cardiology |
        32
     11322 |  30 |            mmcgillk@dell.com |     Montgomery |      McGill |  669017571 | Cardiology |
        23
     11487 |  34 |        bcastagno55@desdev.cn |        Baryram |    Castagno |  973387827 | Cardiology |
        21
     11330 |  69 |        dhallgaths@tumblr.com |        Derward |    Hallgath |  451085417 | Cardiology |
        30
     11305 |  34 |   mstollenberg3@marriott.com |       Marjorie | Stollenberg |  981089876 | Cardiology |
        39
     11395 |  30 | lhollyman21@chronoengine.com |        Lennard |    Hollyman |  872993792 | Cardiology |
        39
     11462 |  30 |  fdopson4g@elegantthemes.com |            Far |      Dopson |  483638406 | Cardiology |
        32
     11327 |  36 |          fbardillp@europa.eu |       Federico |     Bardill |  383354783 | Cardiology |
        36
     11534 |  56 |           jstorms6g@loc.gov |        Jamison |      Storms |  838486941 | Cardiology |
        30
     11405 |  27 |     utimbs2v@dailymotion.com |            Uri |       Timbs |  781909731 | Cardiology |
        37
     11320 |  47 |   nlendeni@elegantthemes.com |          Nolly |      Lenden |  958673895 | Cardiology |
        29
     11374 |  66 |        pvirgin20@foxnews.com |       Prentice |      Virgin |  819542153 | Cardiology |
        18
     11517 |  32 |     mbartholat5z@walmart.com |         Malvin |   Bartholat |  772647129 | Cardiology |
        37
     11302 |  66 |        amailey0@blogspot.com |        Aurilia |      Mailey |  822119060 | Cardiology |
        28
     11535 |  28 |       irylstone6h@hao123.com |        Ibrahim |    Rylstone |  747193545 | Cardiology |
         9
     11447 |  32 |           rceliz41@apple.com |        Raynard |       Celiz |  457685546 | Cardiology |
        38
     11354 |  68 |           hblasetti1g@ft.com |          Henka |    Blasetti |  924528870 | Cardiology |
        30
     11486 |  70 |        ngarrioch54@exblog.jp |        Ninetta |    Garrioch |  900240510 | Cardiology |
        36
     11502 |  54 |     cbuckthorp5k@twitpic.com |          Cayla |   Buckthorp |  797862327 | Cardiology |
        20
     11449 |  35 |  swoodman43@biblegateway.com |       Silvanus |     Woodman |  967019844 | Cardiology |
        35
     11483 |  49 |      jhurdidge51@oaic.gov.au |         Jarret |    Hurdidge |  589910221 | Cardiology |
        15
     11484 |  58 |            hsleney52@aol.com |        Helaine |      Sleney |  948692529 | Cardiology |
        34
     11404 |  69 | sjaray2u@constantcontact.com | Sheila-kathryn |       Jaray |  973927366 | Cardiology |
        32
     11428 |  38 |      rpaulig3i@csmonitor.com |         Robena |      Paulig |  786632343 | Cardiology |
        24
     11402 |  29 |          cforge2s@cpanel.net |      Cassandry |       Forge |  942341893 | Cardiology |
        18
     11465 |  30 |         yglawsop4j@amazon.de |          Yetty |     Glawsop |  221469783 | Cardiology |
        39
     11340 |  67 |      dferminger12@uol.com.br |         Dorris |   Ferminger |  967079673 | Cardiology |
        25
     11468 |  56 |      apool4m@themeforest.net |         Anetta |        Pool |  858545236 | Cardiology |
        37
     11310 |  25 |            dvyvyan8@ucla.edu |       Delainey |      Vyvyan |  790878684 | Cardiology |
        31

(31 rows)

Tracing session: dc218680-9016-11ee-abc7-51c9c115252c
```

```
      37
   11310 |  25 |            dvyvyan8@ucla.edu |       Delainey |      Vyvyan |  790878684 | Cardiology |
      31

(31 rows)

Tracing session: de248680-9016-11ee-abc7-51c9c445252e

 activity
          | timestamp                  | source    | source_elapsed | client
-----------------------------------------------------------------------------------------------------------------------------
-----------+----------------------------+-----------+----------------+-----------
                                                                                                                  Execute
CQL3 query | 2023-12-01 06:57:18.569000 | 127.0.0.1 |              0 | 127.0.0.1
                    Parsing select * from doctor where specialty='Cardiology' Allow Filtering; [Native-Transport-R
equests-1] | 2023-12-01 06:57:18.569000 | 127.0.0.1 |            197 | 127.0.0.1
                                                                                  Preparing statement [Native-Transport-R
equests-1] | 2023-12-01 06:57:18.569000 | 127.0.0.1 |            347 | 127.0.0.1
                                                                                 Computing ranges to query [Native-Transport-R
equests-1] | 2023-12-01 06:57:18.569000 | 127.0.0.1 |            552 | 127.0.0.1
 Submitting range requests on 257 ranges with a concurrency of 75 (1.35 rows per range expected) [Native-Transport-R
equests-1] | 2023-12-01 06:57:18.569000 | 127.0.0.1 |            849 | 127.0.0.1
                                                                Submitted 1 concurrent range requests [Native-Transport-R
equests-1] | 2023-12-01 06:57:18.570000 | 127.0.0.1 |           1339 | 127.0.0.1
              Executing seq scan across 1 sstables for (min(-9223372036854775808), min(-9223372036854775808)] [Re
adStage-3] | 2023-12-01 06:57:18.570000 | 127.0.0.1 |           1474 | 127.0.0.1
                                                                                   Read 250 live rows and 0 tombstone cells [Re
adStage-3] | 2023-12-01 06:57:18.575000 | 127.0.0.1 |           6341 | 127.0.0.1
                                                                                                                  Reques
t complete | 2023-12-01 06:57:18.578696 | 127.0.0.1 |           9696 | 127.0.0.1


cqlsh:healthcare>
```

```
cqlsh:healthcare> SELECT * FROM healthcare.Billing_information where insurance_provider='Cigna' ALLOW F
ILTERING;

 policy_number | billing_amount | insurance_provider | patient_id | payment_status | phone_number
---------------+----------------+--------------------+------------+----------------+--------------
        216452 |           Peso |              Cigna |  296897508 |           Paid |   9032040552
        132922 |           Euro |              Cigna |  781909731 |         Unpaid |   6402240815
        740685 |          Rupee |              Cigna |  544608506 |         Unpaid |   6705551614
        696574 |           Rial |              Cigna |  478632290 |         Unpaid |   4243186314
        953792 |           Ruble |              Cigna |  227010807 |           Paid |   6929228493
        292810 |            Won |              Cigna |  206636350 |         Unpaid |   7573715035
        737258 |          Rupee |              Cigna |  362633029 |        Partial |   4976665286
        626512 |         Dollar |              Cigna |  631870869 |           Paid |   5699056872
        800518 |          Krona |              Cigna |  666059384 |           Paid |   1066871080
        620253 |         Rupiah |              Cigna |  878816364 |         Unpaid |   4434648771
        976020 |         Dollar |              Cigna |  146106230 |         Unpaid |   6251637137
        492888 |        Bolivar |              Cigna |  747193545 |         Unpaid |   3403800934
        118137 |           Peso |              Cigna |  477166121 |        Partial |   2261071665
        209245 |          Koruna |              Cigna |  941996225 |           Paid |   5413926156
        613771 |  Yuan Renminbi |              Cigna |  952690846 |        Partial |   5317806181
        148432 |  Yuan Renminbi |              Cigna |  952307752 |         Unpaid |   9465993473
        486835 |           Peso |              Cigna |  134715238 |        Partial |   1775557274
        803460 |           Euro |              Cigna |  172196320 |         Unpaid |   9335767110
        197450 |          Zloty |              Cigna |  583820076 |        Partial |   8275203228
        872498 |           Birr |              Cigna |  383354783 |         Unpaid |   4453033515
        531988 |         Rupiah |              Cigna |  927918713 |         Unpaid |   2148963630
        380059 |  Yuan Renminbi |              Cigna |  341300963 |         Unpaid |   9474329873
        913101 |          Litas |              Cigna |  345537363 |           Paid |   1362967929
        572364 |           Ruble |              Cigna |  366580016 |           Paid |   5004470049
        244572 |           Peso |              Cigna |  429524030 |        Partial |   3368262550
        813681 |  Yuan Renminbi |              Cigna |  252815187 |         Unpaid |   8113673897
        719258 |  Yuan Renminbi |              Cigna |   55659397 |         Unpaid |   6966643962
        750225 |           Ruble |              Cigna |  920397895 |        Partial |   4758879151
        849608 |         Rupiah |              Cigna |   58847187 |           Paid |   4304634782
        403852 |            Lek |              Cigna |  581906070 |           Paid |   8998626609
        250387 |           Real |              Cigna |  973051723 |           Paid |   6007823632
        733475 |         Rupiah |              Cigna |  829467866 |        Partial |   3461758911
        125342 |           Peso |              Cigna |  879419974 |           Paid |   4826277840
        745879 |           Euro |              Cigna |  225299968 |           Paid |   9359535042
        198433 |           Real |              Cigna |  366575206 |         Unpaid |   2439109626
        384101 |           Real |              Cigna |  337089339 |        Partial |   1465400067
        124500 |          Krone |              Cigna |  967051114 |           Paid |   3422192130
        315983 |  Yuan Renminbi |              Cigna |  948692529 |        Partial |   4738775967
        111620 |          Shekel |              Cigna |  203000145 |           Paid |   7633862308
        412349 |  Yuan Renminbi |              Cigna |  998132204 |        Partial |   8874450519
        711238 |         Rupiah |              Cigna |  911206717 |        Partial |   5635423517
        744636 |          Dinar |              Cigna |  465471842 |        Partial |   3824446567
        869078 |         Ariary |              Cigna |  838486941 |         Unpaid |   8527016179
        995020 |            Sol |              Cigna |  707411426 |         Unpaid |   5249425003
        698638 |            Yen |              Cigna |  589802751 |           Paid |   4586926619
        555294 |           Ruble |              Cigna |  963998441 |           Paid |   4186719929
        915704 |         Dollar |              Cigna |  420377926 |        Partial |   6018644563
        273465 |         Rupiah |              Cigna |  202267689 |           Paid |   4235113634

(347 rows)
```

```
Tracing session: 25579b50-9017-11ee-abc7-51c9c445252e

 activity
                              | timestamp                  | source    | source_elapsed | client
--------------------------------------------------------------------------------------------------------------
-------------------------------+----------------------------+-----------+----------------+-----------

           Execute CQL3 query | 2023-12-01 06:59:18.021000 | 127.0.0.1 |              0 | 127.0.0.1
 Parsing SELECT * FROM healthcare.Billing_information where insurance_provider='Cigna' ALLOW FILTERING;
 [Native-Transport-Requests-1] | 2023-12-01 06:59:18.021000 | 127.0.0.1 |            175 | 127.0.0.1
                                                                               Preparing statement
 [Native-Transport-Requests-1] | 2023-12-01 06:59:18.021000 | 127.0.0.1 |            387 | 127.0.0.1
                                                                         Computing ranges to query
 [Native-Transport-Requests-1] | 2023-12-01 06:59:18.022000 | 127.0.0.1 |            618 | 127.0.0.1
        Submitting range requests on 42 ranges with a concurrency of 25 (4.05 rows per range expected)
 [Native-Transport-Requests-1] | 2023-12-01 06:59:18.022000 | 127.0.0.1 |            759 | 127.0.0.1
                                                              Submitted 1 concurrent range requests
 [Native-Transport-Requests-1] | 2023-12-01 06:59:18.022000 | 127.0.0.1 |            935 | 127.0.0.1
                                      Executing seq scan across 1 sstables for (751647, min(-92233
72036854775808)] [ReadStage-3] | 2023-12-01 06:59:18.022000 | 127.0.0.1 |           1060 | 127.0.0.1
                                                                          Read 134 live rows and 0
 tombstone cells [ReadStage-3] | 2023-12-01 06:59:18.024000 | 127.0.0.1 |           2592 | 127.0.0.1

             Request complete | 2023-12-01 06:59:18.024112 | 127.0.0.1 |           3112 | 127.0.0.1

cqlsh:healthcare>
```

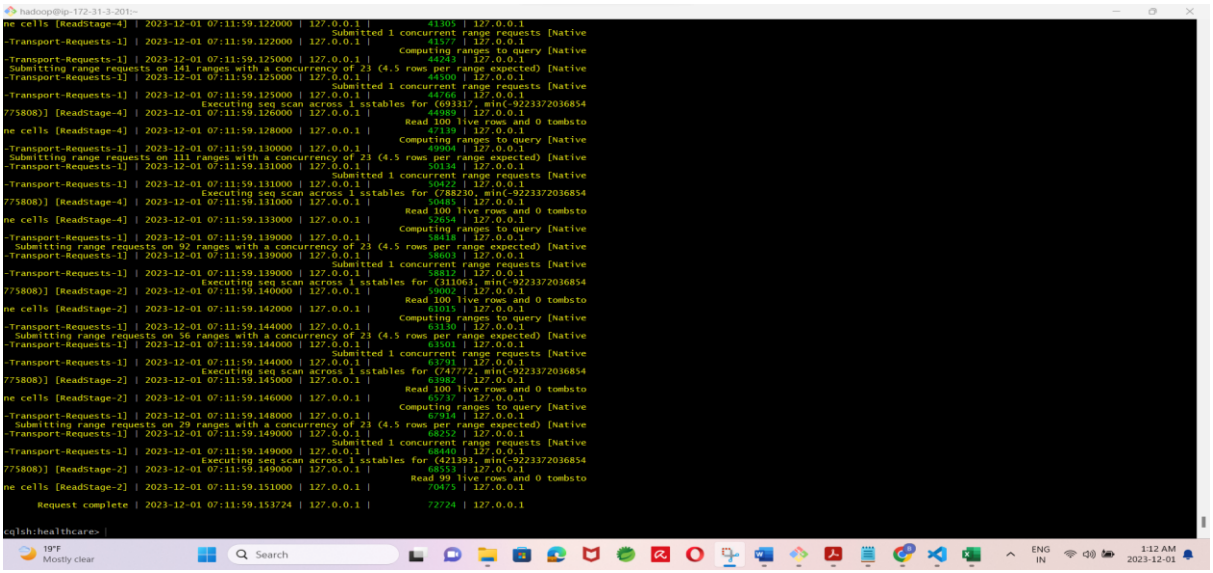
Complex queries:

Count:

```
.0.0.1

cqlsh:healthcare> select count(*) as unpaid_count from billing_information where payment_status='Unpaid' ALLOW FILTERING;

 unpaid_count
--------------
          341

(1 rows)

Warnings :
Aggregation query used without partition key


Tracing session: a554d880-9018-11ee-abc7-51c9c445252e

 activity                                                                                                          | timestamp                  | source    | source_elapsed | client
-------------------------------------------------------------------------------------------------------------------+----------------------------+-----------+----------------+--------
---
                                                                                                Execute CQL3 query | 2023-12-01 07:10:02.248000 | 127.0.0.1 |              0 | 127.0.0
.1
 Parsing select count(*) as unpaid_count from billing_information where payment_status='Unpaid' ALLOW FILTERING; [Native-Transport-Requests-1] | 2023-12-01 07:10:02.248000 | 127.0.0.1 |            183 | 127.0.0
.1
                                                                     Preparing statement [Native-Transport-Requests-1] | 2023-12-01 07:10:02.248000 | 127.0.0.1 |            362 | 127.0.0
.1
                                                                Computing ranges to query [Native-Transport-Requests-1] | 2023-12-01 07:10:02.249000 | 127.0.0.1 |           1071 | 127.0.0
.1
          Submitting range requests on 257 ranges with a concurrency of 25 (4.05 rows per range expected) [Native-Transport-Requests-1] | 2023-12-01 07:10:02.250000 | 127.0.0.1 |           1439 | 127.0.0
.1
                                                     Submitted 1 concurrent range requests [Native-Transport-Requests-1] | 2023-12-01 07:10:02.250000 | 127.0.0.1 |           1897 | 127.0.0
.1
                Executing seq scan across 1 sstables for (min(-9223372036854775808), min(-9223372036854775808)] [ReadStage-2] | 2023-12-01 07:10:02.250000 | 127.0.0.1 |           1956 | 127.0.0
```

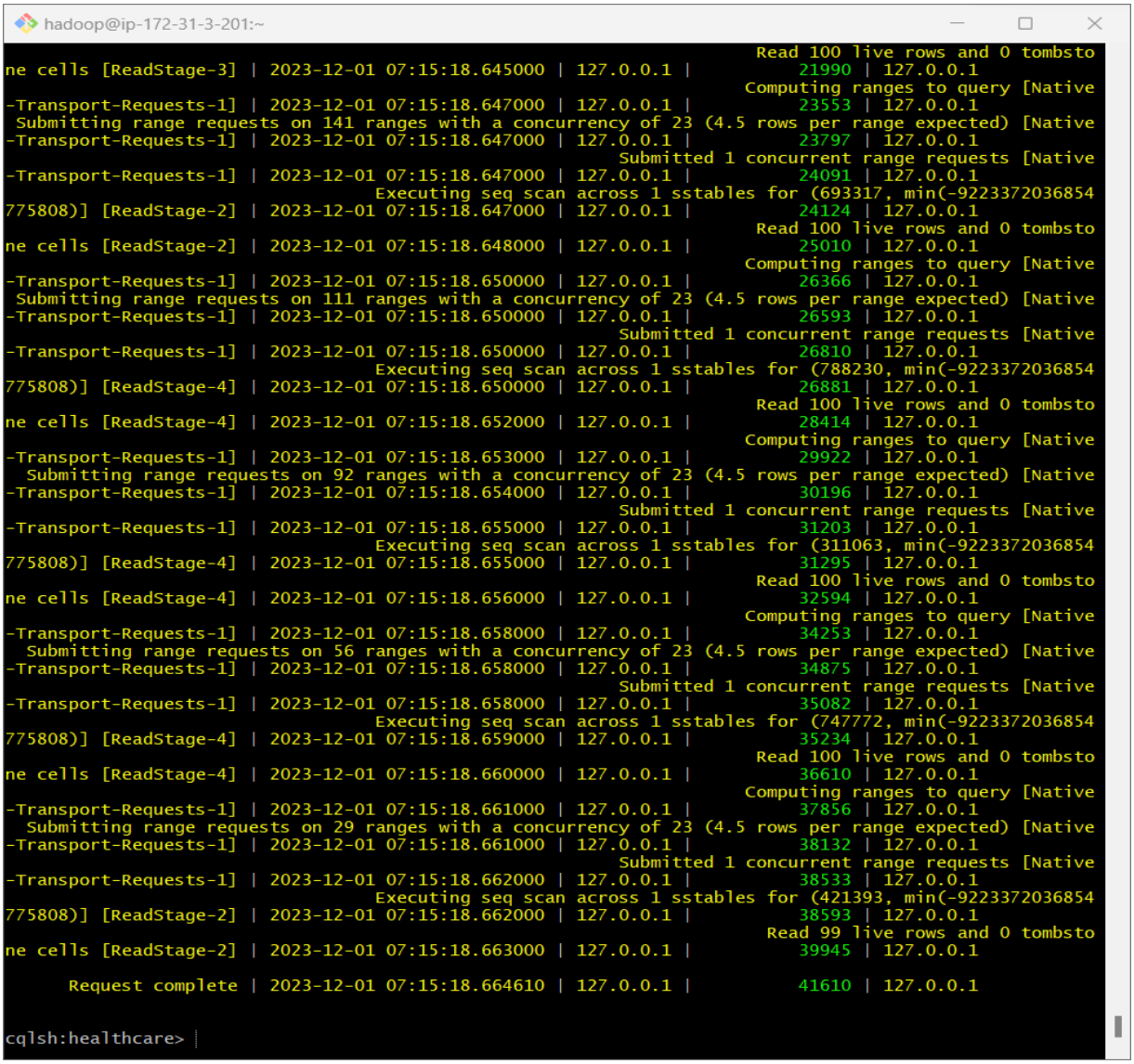
AVG

```
cqlsh:healthcare> select avg(premium_amount) as Averageprem From insurance Allow filtering;

 averageprem
-------------
     5058.63

(1 rows)

Warnings :
Aggregation query used without partition key
```

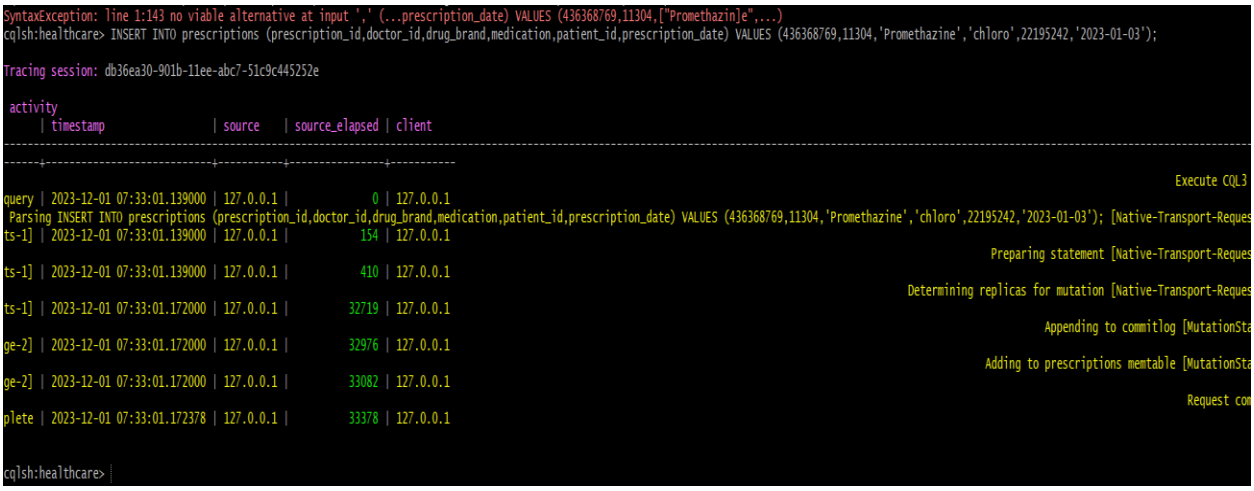


Max:

```
hadoop@ip-172-31-3-201:~
ne cells [ReadStage-2] | 2023-12-01 07:11:59.151000 | 127.0.0.1 |          70475 | 127.0.0.1

      Request complete | 2023-12-01 07:11:59.153724 | 127.0.0.1 |          72724 | 127.0.0.1


cqlsh:healthcare>
cqlsh:healthcare> select max(coverage_limit) as Maxcoverage from insurance;

 maxcoverage
-------------
   999330.35

(1 rows)

Warnings :
Aggregation query used without partition key


Tracing session: 61e7d6f0-9019-11ee-abc7-51c9c445252e

 activity
                        | timestamp                  | source    | source_elapsed | client
-------------------------------------------------------------------------------------------------------------------
------------------------+----------------------------+-----------+----------------+-----------

    Execute CQL3 query | 2023-12-01 07:15:18.623000 | 127.0.0.1 |              0 | 127.0.0.1
                                Parsing select max(coverage_limit) as Maxcoverage from insurance; [Native
-Transport-Requests-1] | 2023-12-01 07:15:18.624000 | 127.0.0.1 |            175 | 127.0.0.1
                                                                                Preparing statement [Native
-Transport-Requests-1] | 2023-12-01 07:15:18.624000 | 127.0.0.1 |            302 | 127.0.0.1
                                                                             Computing ranges to query [Native
-Transport-Requests-1] | 2023-12-01 07:15:18.625000 | 127.0.0.1 |           1847 | 127.0.0.1
 Submitting range requests on 257 ranges with a concurrency of 23 (4.5 rows per range expected) [Native
-Transport-Requests-1] | 2023-12-01 07:15:18.625000 | 127.0.0.1 |           2148 | 127.0.0.1
                                                                           Submitted 1 concurrent range requests [Native
-Transport-Requests-1] | 2023-12-01 07:15:18.626000 | 127.0.0.1 |           2532 | 127.0.0.1
                  Executing seq scan across 1 sstables for (min(-9223372036854775808), min(-9223372036854
775808)] [ReadStage-2] | 2023-12-01 07:15:18.626000 | 127.0.0.1 |           2653 | 127.0.0.1
                                                                                Read 100 live rows and 0 tombsto
ne cells [ReadStage-2] | 2023-12-01 07:15:18.627000 | 127.0.0.1 |           3839 | 127.0.0.1
                                                                             Computing ranges to query [Native
-Transport-Requests-1] | 2023-12-01 07:15:18.629000 | 127.0.0.1 |           5735 | 127.0.0.1
 Submitting range requests on 235 ranges with a concurrency of 23 (4.5 rows per range expected) [Native
-Transport-Requests-1] | 2023-12-01 07:15:18.630000 | 127.0.0.1 |           6619 | 127.0.0.1
                                                                           Submitted 1 concurrent range requests [Native
-Transport-Requests-1] | 2023-12-01 07:15:18.631000 | 127.0.0.1 |           7427 | 127.0.0.1
                                Executing seq scan across 1 sstables for (874486, min(-9223372036854
775808)] [ReadStage-3] | 2023-12-01 07:15:18.631000 | 127.0.0.1 |           7492 | 127.0.0.1
                                                                                Read 100 live rows and 0 tombsto
ne cells [ReadStage-3] | 2023-12-01 07:15:18.632000 | 127.0.0.1 |           8727 | 127.0.0.1
                                                                             Computing ranges to query [Native
-Transport-Requests-1] | 2023-12-01 07:15:18.634000 | 127.0.0.1 |          10436 | 127.0.0.1
 Submitting range requests on 205 ranges with a concurrency of 23 (4.5 rows per range expected) [Native
-Transport-Requests-1] | 2023-12-01 07:15:18.634000 | 127.0.0.1 |          10747 | 127.0.0.1
                                                                           Submitted 1 concurrent range requests [Native
-Transport-Requests-1] | 2023-12-01 07:15:18.634000 | 127.0.0.1 |          11117 | 127.0.0.1
                                Executing seq scan across 1 sstables for (728085, min(-9223372036854
```


hadoop@ip-172-31-3-201:~
ne cells [ReadStage-3] | 2023-12-01 07:15:18.645000 | 127.0.0.1 | 21990 | 127.0.0.1
Read 100 live rows and 0 tombsto
-Transport-Requests-1] | 2023-12-01 07:15:18.647000 | 127.0.0.1 | 23553 | 127.0.0.1
Computing ranges to query [Native
Submitting range requests on 141 ranges with a concurrency of 23 (4.5 rows per range expected) [Native
-Transport-Requests-1] | 2023-12-01 07:15:18.647000 | 127.0.0.1 | 23797 | 127.0.0.1
-Transport-Requests-1] | 2023-12-01 07:15:18.647000 | 127.0.0.1 | 24091 | 127.0.0.1
Submitted 1 concurrent range requests [Native
Executing seq scan across 1 sstables for (693317, min(-9223372036854
775808)] [ReadStage-2] | 2023-12-01 07:15:18.647000 | 127.0.0.1 | 24124 | 127.0.0.1
ne cells [ReadStage-2] | 2023-12-01 07:15:18.648000 | 127.0.0.1 | 25010 | 127.0.0.1
-Transport-Requests-1] | 2023-12-01 07:15:18.650000 | 127.0.0.1 | 26366 | 127.0.0.1
Submitting range requests on 111 ranges with a concurrency of 23 (4.5 rows per range expected) [Native
-Transport-Requests-1] | 2023-12-01 07:15:18.650000 | 127.0.0.1 | 26593 | 127.0.0.1
-Transport-Requests-1] | 2023-12-01 07:15:18.650000 | 127.0.0.1 | 26810 | 127.0.0.1
Executing seq scan across 1 sstables for (788230, min(-9223372036854
775808)] [ReadStage-4] | 2023-12-01 07:15:18.650000 | 127.0.0.1 | 26881 | 127.0.0.1
ne cells [ReadStage-4] | 2023-12-01 07:15:18.652000 | 127.0.0.1 | 28414 | 127.0.0.1
-Transport-Requests-1] | 2023-12-01 07:15:18.653000 | 127.0.0.1 | 29922 | 127.0.0.1
Submitting range requests on 92 ranges with a concurrency of 23 (4.5 rows per range expected) [Native
-Transport-Requests-1] | 2023-12-01 07:15:18.654000 | 127.0.0.1 | 30196 | 127.0.0.1
-Transport-Requests-1] | 2023-12-01 07:15:18.655000 | 127.0.0.1 | 31203 | 127.0.0.1
Executing seq scan across 1 sstables for (311063, min(-9223372036854
775808)] [ReadStage-4] | 2023-12-01 07:15:18.655000 | 127.0.0.1 | 31295 | 127.0.0.1
ne cells [ReadStage-4] | 2023-12-01 07:15:18.656000 | 127.0.0.1 | 32594 | 127.0.0.1
-Transport-Requests-1] | 2023-12-01 07:15:18.658000 | 127.0.0.1 | 34253 | 127.0.0.1
Submitting range requests on 56 ranges with a concurrency of 23 (4.5 rows per range expected) [Native
-Transport-Requests-1] | 2023-12-01 07:15:18.658000 | 127.0.0.1 | 34875 | 127.0.0.1
-Transport-Requests-1] | 2023-12-01 07:15:18.658000 | 127.0.0.1 | 35082 | 127.0.0.1
Executing seq scan across 1 sstables for (747772, min(-9223372036854
775808)] [ReadStage-4] | 2023-12-01 07:15:18.659000 | 127.0.0.1 | 35234 | 127.0.0.1
ne cells [ReadStage-4] | 2023-12-01 07:15:18.660000 | 127.0.0.1 | 36610 | 127.0.0.1
-Transport-Requests-1] | 2023-12-01 07:15:18.661000 | 127.0.0.1 | 37856 | 127.0.0.1
Submitting range requests on 29 ranges with a concurrency of 23 (4.5 rows per range expected) [Native
-Transport-Requests-1] | 2023-12-01 07:15:18.661000 | 127.0.0.1 | 38132 | 127.0.0.1
-Transport-Requests-1] | 2023-12-01 07:15:18.662000 | 127.0.0.1 | 38533 | 127.0.0.1
Executing seq scan across 1 sstables for (421393, min(-9223372036854
775808)] [ReadStage-2] | 2023-12-01 07:15:18.662000 | 127.0.0.1 | 38593 | 127.0.0.1
ne cells [ReadStage-2] | 2023-12-01 07:15:18.663000 | 127.0.0.1 | 39945 | 127.0.0.1
Read 99 live rows and 0 tombsto
Request complete | 2023-12-01 07:15:18.664610 | 127.0.0.1 | 41610 | 127.0.0.1
cqlsh:healthcare>

Insert:


SyntaxException: line 1:143 no viable alternative at input ',' (...prescription_date) VALUES (436368769,11304,["Promethazin]e",...)
cqlsh:healthcare> INSERT INTO prescriptions (prescription_id,doctor_id,drug_brand,medication,patient_id,prescription_date) VALUES (436368769,11304,'Promethazine','chloro',22195242,'2023-01-03');
Tracing session: db36ea30-901b-11ee-abc7-51c9c445252e
activity
| timestamp | source | source_elapsed | client
query | 2023-12-01 07:33:01.139000 | 127.0.0.1 | 0 | 127.0.0.1
Execute CQL3
Parsing INSERT INTO prescriptions (prescription_id,doctor_id,drug_brand,medication,patient_id,prescription_date) VALUES (436368769,11304,'Promethazine','chloro',22195242,'2023-01-03'); [Native-Transport-Reques
ts-1] | 2023-12-01 07:33:01.139000 | 127.0.0.1 | 154 | 127.0.0.1
Preparing statement [Native-Transport-Reques
ts-1] | 2023-12-01 07:33:01.139000 | 127.0.0.1 | 410 | 127.0.0.1
Determining replicas for mutation [Native-Transport-Reques
ts-1] | 2023-12-01 07:33:01.172000 | 127.0.0.1 | 32719 | 127.0.0.1
Appending to commitlog [MutationSta
ge-2] | 2023-12-01 07:33:01.172000 | 127.0.0.1 | 32976 | 127.0.0.1
Adding to prescriptions memtable [MutationSta
ge-2] | 2023-12-01 07:33:01.172000 | 127.0.0.1 | 33082 | 127.0.0.1
Request com
plete | 2023-12-01 07:33:01.172378 | 127.0.0.1 | 33378 | 127.0.0.1
cqlsh:healthcare>

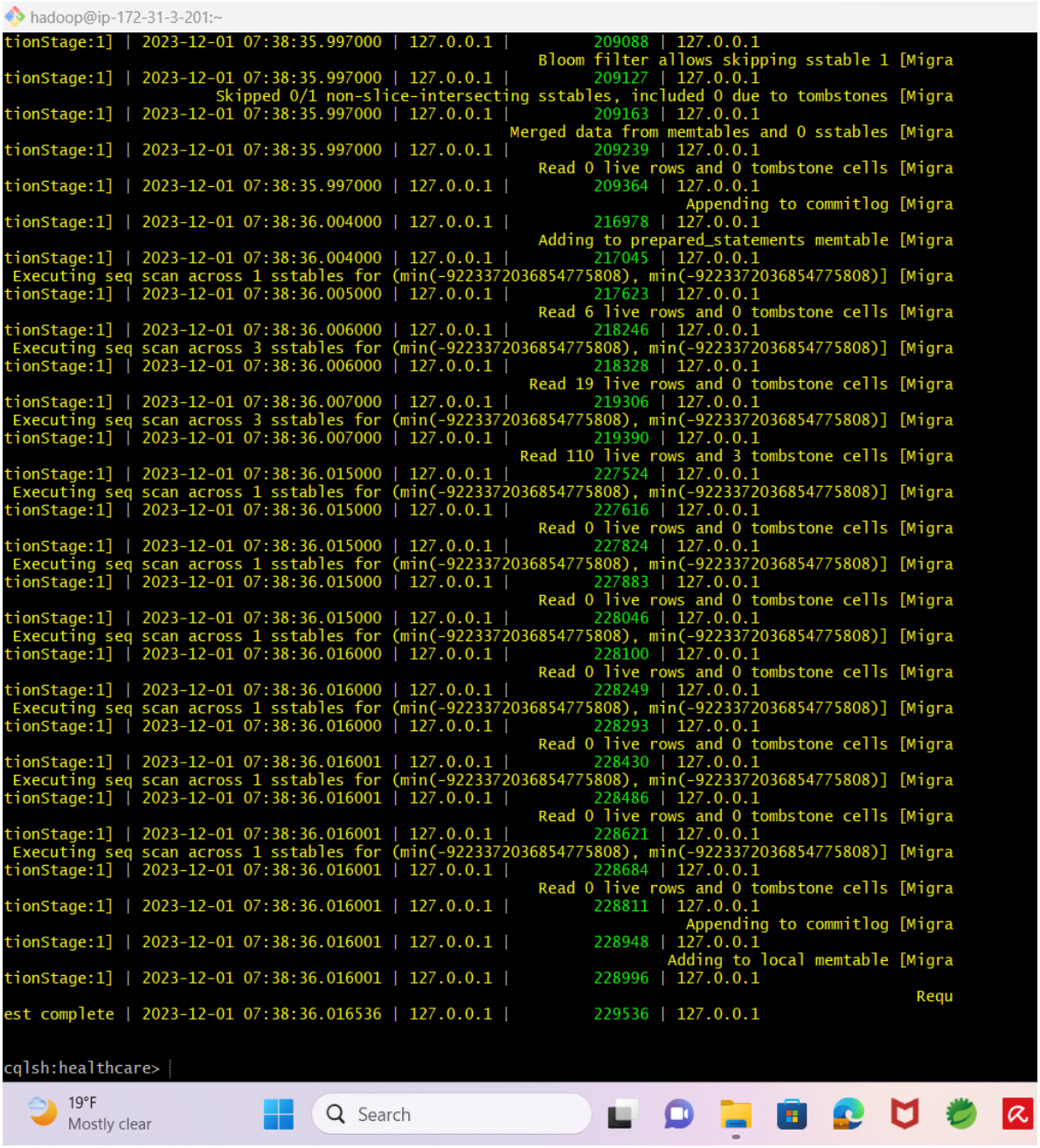
Delete

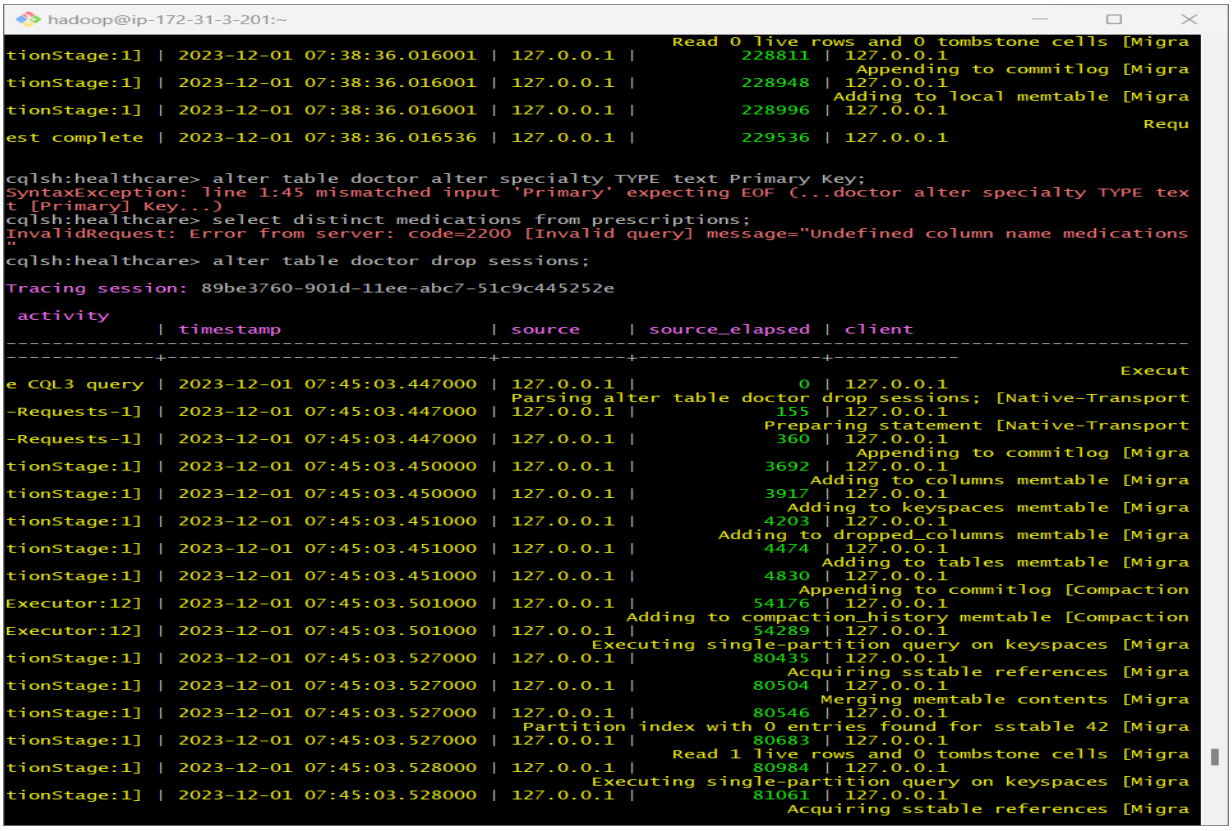
```
cqlsh:healthcare> delete from prescriptions where prescription_id=380802288;

Tracing session: 11f384c0-901c-11ee-abc7-51c9c445252e

 activity                                                                                                    | tim
estamp                     | source      | source_elapsed | client
-----------------------------------------------------------------------------------------------------------+----
-----------------------+-----------+----------------+-----------
                                                                                          Execute CQL3 query | 202
3-12-01 07:34:32.972000 | 127.0.0.1 |              0 | 127.0.0.1
 Parsing delete from prescriptions where prescription_id=380802288; [Native-Transport-Requests-1] | 202
3-12-01 07:34:32.972000 | 127.0.0.1 |            148 | 127.0.0.1
                                               Preparing statement [Native-Transport-Requests-1] | 202
3-12-01 07:34:32.972000 | 127.0.0.1 |            378 | 127.0.0.1
                                  Determining replicas for mutation [Native-Transport-Requests-1] | 202
3-12-01 07:34:32.972000 | 127.0.0.1 |            611 | 127.0.0.1
                                             Appending to commitlog [Native-Transport-Requests-1] | 202
3-12-01 07:34:32.972000 | 127.0.0.1 |            718 | 127.0.0.1
                                   Adding to prescriptions memtable [Native-Transport-Requests-1] | 202
3-12-01 07:34:32.973000 | 127.0.0.1 |            787 | 127.0.0.1
                                                                                        Request complete | 202
3-12-01 07:34:32.972873 | 127.0.0.1 |            873 | 127.0.0.1

cqlsh:healthcare>
```

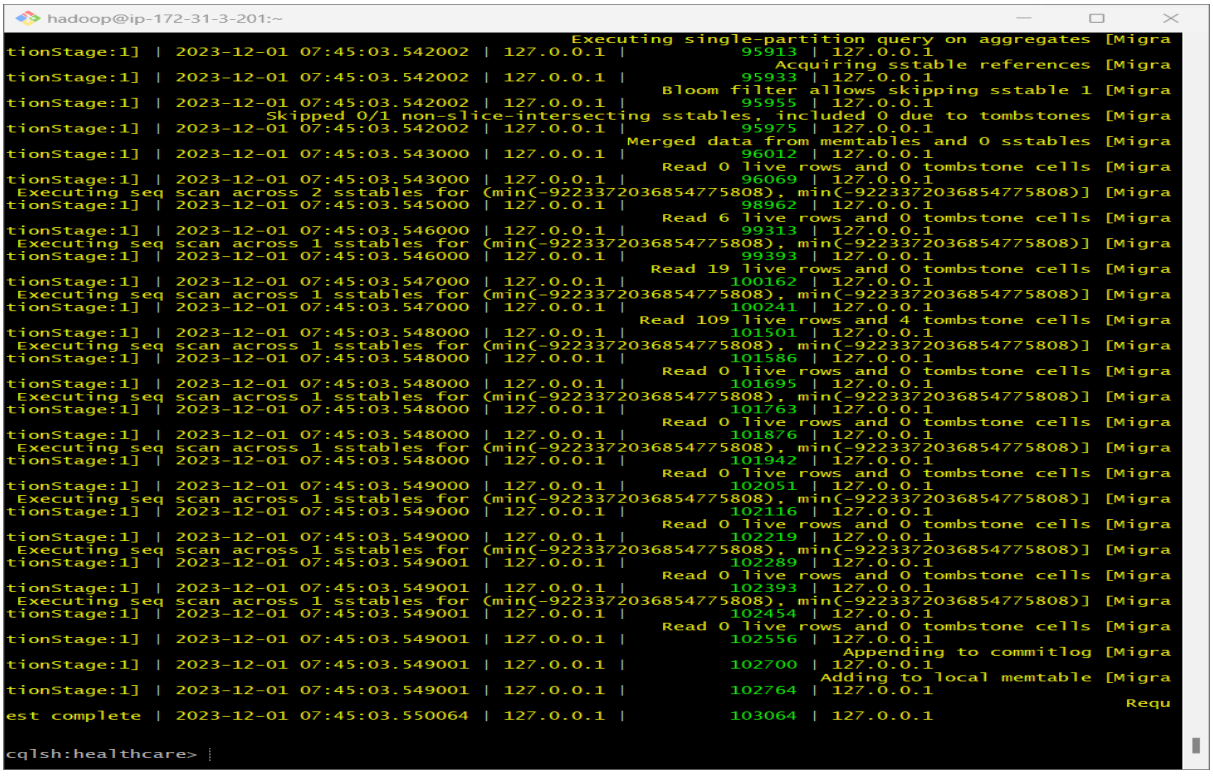
alter table

```
hadoop@ip-172-31-3-201:~
                                   Adding to prescriptions memtable [Native-Transport-Requests-1] | 202
3-12-01 07:34:32.973000 | 127.0.0.1 |            787 | 127.0.0.1
                                                                                        Request complete | 202
3-12-01 07:34:32.972873 | 127.0.0.1 |            873 | 127.0.0.1

cqlsh:healthcare> alter table doctor add column sessions;
InvalidRequest: Error from server: code=2200 [Invalid query] message="Unknown type healthcare.sessions"
cqlsh:healthcare> alter table doctor add column sessions int;
SyntaxException: line 1:39 mismatched input 'int' expecting EOF (...table doctor add column sessions [i
nt]...)
cqlsh:healthcare> alter table doctor add sessions int;

Tracing session: a2ae25b0-901c-11ee-abc7-51c9c445252e

 activity
             | timestamp                  | source    | source_elapsed | client
-----------------------------------------------------------------------------------------------------
-------------+----------------------------+-----------+----------------+-----------
                                                                                              Execut
e CQL3 query | 2023-12-01 07:38:35.787000 | 127.0.0.1 |              0 | 127.0.0.1
                                       Parsing alter table doctor add sessions int; [Native-Transport
-Requests-1] | 2023-12-01 07:38:35.788000 | 127.0.0.1 |            199 | 127.0.0.1
                                                                    Preparing statement [Native-Transport
-Requests-1] | 2023-12-01 07:38:35.788000 | 127.0.0.1 |            370 | 127.0.0.1
                                                                           Appending to commitlog [Migra
tionStage:1] | 2023-12-01 07:38:35.793000 | 127.0.0.1 |           6035 | 127.0.0.1
                                                                       Adding to columns memtable [Migra
tionStage:1] | 2023-12-01 07:38:35.794000 | 127.0.0.1 |           6241 | 127.0.0.1
                                                                     Adding to keyspaces memtable [Migra
tionStage:1] | 2023-12-01 07:38:35.794000 | 127.0.0.1 |           6650 | 127.0.0.1
                                                                         Adding to tables memtable [Migra
tionStage:1] | 2023-12-01 07:38:35.794000 | 127.0.0.1 |           6990 | 127.0.0.1
                                                   Executing single-partition query on keyspaces [Migra
tionStage:1] | 2023-12-01 07:38:35.915000 | 127.0.0.1 |         127274 | 127.0.0.1
                                                                    Acquiring sstable references [Migra
tionStage:1] | 2023-12-01 07:38:35.915000 | 127.0.0.1 |         127396 | 127.0.0.1
                                                                        Merging memtable contents [Migra
tionStage:1] | 2023-12-01 07:38:35.915000 | 127.0.0.1 |         127452 | 127.0.0.1
                                            Partition index with 0 entries found for sstable 40 [Migra
tionStage:1] | 2023-12-01 07:38:35.915000 | 127.0.0.1 |         127697 | 127.0.0.1
                                                          Read 1 live rows and 0 tombstone cells [Migra
tionStage:1] | 2023-12-01 07:38:35.916000 | 127.0.0.1 |         128524 | 127.0.0.1
                                                   Executing single-partition query on keyspaces [Migra
tionStage:1] | 2023-12-01 07:38:35.917000 | 127.0.0.1 |         129691 | 127.0.0.1
                                                                    Acquiring sstable references [Migra
tionStage:1] | 2023-12-01 07:38:35.917000 | 127.0.0.1 |         129791 | 127.0.0.1
                                                                        Merging memtable contents [Migra
tionStage:1] | 2023-12-01 07:38:35.917000 | 127.0.0.1 |         129830 | 127.0.0.1
                                                                        Key cache hit for sstable 40 [Migra
tionStage:1] | 2023-12-01 07:38:35.917000 | 127.0.0.1 |         129953 | 127.0.0.1
                                                          Read 1 live rows and 0 tombstone cells [Migra
tionStage:1] | 2023-12-01 07:38:35.918000 | 127.0.0.1 |         130102 | 127.0.0.1
                                                       Executing single-partition query on types [Migra
tionStage:1] | 2023-12-01 07:38:35.918000 | 127.0.0.1 |         130251 | 127.0.0.1
                                                                    Acquiring sstable references [Migra
tionStage:1] | 2023-12-01 07:38:35.918000 | 127.0.0.1 |         130320 | 127.0.0.1
                                                          Bloom filter allows skipping sstable 1 [Migra
```

19°F
Mostly clear
Search


hadoop@ip-172-31-3-201:~
tionStage:1] | 2023-12-01 07:38:35.997000 | 127.0.0.1 | 209088 | 127.0.0.1
Bloom filter allows skipping sstable 1 [Migra
tionStage:1] | 2023-12-01 07:38:35.997000 | 127.0.0.1 | 209127 | 127.0.0.1
Skipped 0/1 non-slice-intersecting sstables, included 0 due to tombstones [Migra
tionStage:1] | 2023-12-01 07:38:35.997000 | 127.0.0.1 | 209163 | 127.0.0.1
Merged data from memtables and 0 sstables [Migra
tionStage:1] | 2023-12-01 07:38:35.997000 | 127.0.0.1 | 209239 | 127.0.0.1
Read 0 live rows and 0 tombstone cells [Migra
tionStage:1] | 2023-12-01 07:38:35.997000 | 127.0.0.1 | 209364 | 127.0.0.1
Appending to commitlog [Migra
tionStage:1] | 2023-12-01 07:38:36.004000 | 127.0.0.1 | 216978 | 127.0.0.1
Adding to prepared_statements memtable [Migra
tionStage:1] | 2023-12-01 07:38:36.004000 | 127.0.0.1 | 217045 | 127.0.0.1
Executing seq scan across 1 sstables for (min(-9223372036854775808), min(-9223372036854775808)] [Migra
tionStage:1] | 2023-12-01 07:38:36.005000 | 127.0.0.1 | 217623 | 127.0.0.1
Read 6 live rows and 0 tombstone cells [Migra
tionStage:1] | 2023-12-01 07:38:36.006000 | 127.0.0.1 | 218246 | 127.0.0.1
Executing seq scan across 3 sstables for (min(-9223372036854775808), min(-9223372036854775808)] [Migra
tionStage:1] | 2023-12-01 07:38:36.006000 | 127.0.0.1 | 218328 | 127.0.0.1
Read 19 live rows and 0 tombstone cells [Migra
tionStage:1] | 2023-12-01 07:38:36.007000 | 127.0.0.1 | 219306 | 127.0.0.1
Executing seq scan across 3 sstables for (min(-9223372036854775808), min(-9223372036854775808)] [Migra
tionStage:1] | 2023-12-01 07:38:36.007000 | 127.0.0.1 | 219390 | 127.0.0.1
Read 110 live rows and 3 tombstone cells [Migra
tionStage:1] | 2023-12-01 07:38:36.015000 | 127.0.0.1 | 227524 | 127.0.0.1
Executing seq scan across 1 sstables for (min(-9223372036854775808), min(-9223372036854775808)] [Migra
tionStage:1] | 2023-12-01 07:38:36.015000 | 127.0.0.1 | 227616 | 127.0.0.1
Read 0 live rows and 0 tombstone cells [Migra
tionStage:1] | 2023-12-01 07:38:36.015000 | 127.0.0.1 | 227824 | 127.0.0.1
Executing seq scan across 1 sstables for (min(-9223372036854775808), min(-9223372036854775808)] [Migra
tionStage:1] | 2023-12-01 07:38:36.015000 | 127.0.0.1 | 227883 | 127.0.0.1
Read 0 live rows and 0 tombstone cells [Migra
tionStage:1] | 2023-12-01 07:38:36.015000 | 127.0.0.1 | 228046 | 127.0.0.1
Executing seq scan across 1 sstables for (min(-9223372036854775808), min(-9223372036854775808)] [Migra
tionStage:1] | 2023-12-01 07:38:36.016000 | 127.0.0.1 | 228100 | 127.0.0.1
Read 0 live rows and 0 tombstone cells [Migra
tionStage:1] | 2023-12-01 07:38:36.016000 | 127.0.0.1 | 228249 | 127.0.0.1
Executing seq scan across 1 sstables for (min(-9223372036854775808), min(-9223372036854775808)] [Migra
tionStage:1] | 2023-12-01 07:38:36.016000 | 127.0.0.1 | 228293 | 127.0.0.1
Read 0 live rows and 0 tombstone cells [Migra
tionStage:1] | 2023-12-01 07:38:36.016001 | 127.0.0.1 | 228430 | 127.0.0.1
Executing seq scan across 1 sstables for (min(-9223372036854775808), min(-9223372036854775808)] [Migra
tionStage:1] | 2023-12-01 07:38:36.016001 | 127.0.0.1 | 228486 | 127.0.0.1
Read 0 live rows and 0 tombstone cells [Migra
tionStage:1] | 2023-12-01 07:38:36.016001 | 127.0.0.1 | 228621 | 127.0.0.1
Executing seq scan across 1 sstables for (min(-9223372036854775808), min(-9223372036854775808)] [Migra
tionStage:1] | 2023-12-01 07:38:36.016001 | 127.0.0.1 | 228684 | 127.0.0.1
Read 0 live rows and 0 tombstone cells [Migra
tionStage:1] | 2023-12-01 07:38:36.016001 | 127.0.0.1 | 228811 | 127.0.0.1
Appending to commitlog [Migra
tionStage:1] | 2023-12-01 07:38:36.016001 | 127.0.0.1 | 228948 | 127.0.0.1
Adding to local memtable [Migra
tionStage:1] | 2023-12-01 07:38:36.016001 | 127.0.0.1 | 228996 | 127.0.0.1
Requ
est complete | 2023-12-01 07:38:36.016536 | 127.0.0.1 | 229536 | 127.0.0.1
cqlsh:healthcare>
19°F
Mostly clear
Search

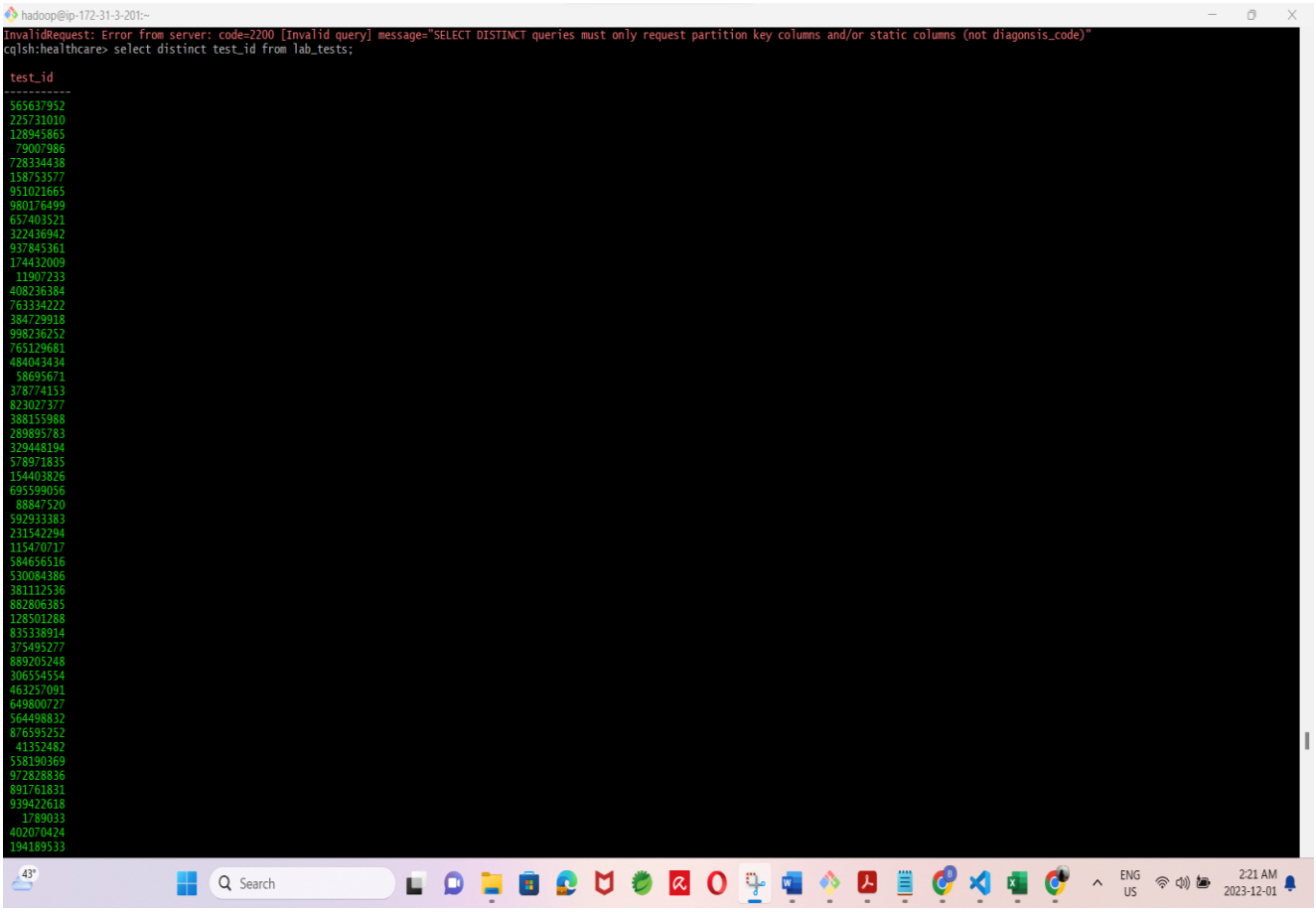

hadoop@ip-172-31-3-201:~
Read 0 live rows and 0 tombstone cells [Migra
tionStage:1] | 2023-12-01 07:38:36.016001 | 127.0.0.1 | 228811 | 127.0.0.1
Appending to commitlog [Migra
tionStage:1] | 2023-12-01 07:38:36.016001 | 127.0.0.1 | 228948 | 127.0.0.1
Adding to local memtable [Migra
tionStage:1] | 2023-12-01 07:38:36.016001 | 127.0.0.1 | 228996 | 127.0.0.1
Requ
est complete | 2023-12-01 07:38:36.016536 | 127.0.0.1 | 229536 | 127.0.0.1
cqlsh:healthcare> alter table doctor alter specialty TYPE text Primary Key;
SyntaxException: line 1:45 mismatched input 'Primary' expecting EOF (...doctor alter specialty TYPE tex
t [Primary] Key...)
cqlsh:healthcare> select distinct medications from prescriptions;
InvalidRequest: Error from server: code=2200 [Invalid query] message="Undefined column name medications
"
cqlsh:healthcare> alter table doctor drop sessions;
Tracing session: 89be3760-901d-11ee-abc7-51c9c445252e
activity
| timestamp | source | source_elapsed | client
Execut
e CQL3 query | 2023-12-01 07:45:03.447000 | 127.0.0.1 | 0 | 127.0.0.1
Parsing alter table doctor drop sessions; [Native-Transport
-Requests-1] | 2023-12-01 07:45:03.447000 | 127.0.0.1 | 155 | 127.0.0.1
Preparing statement [Native-Transport
-Requests-1] | 2023-12-01 07:45:03.447000 | 127.0.0.1 | 360 | 127.0.0.1
Appending to commitlog [Migra
tionStage:1] | 2023-12-01 07:45:03.450000 | 127.0.0.1 | 3692 | 127.0.0.1
Adding to columns memtable [Migra
tionStage:1] | 2023-12-01 07:45:03.450000 | 127.0.0.1 | 3917 | 127.0.0.1
Adding to keyspaces memtable [Migra
tionStage:1] | 2023-12-01 07:45:03.451000 | 127.0.0.1 | 4203 | 127.0.0.1
Adding to dropped_columns memtable [Migra
tionStage:1] | 2023-12-01 07:45:03.451000 | 127.0.0.1 | 4474 | 127.0.0.1
Adding to tables memtable [Migra
tionStage:1] | 2023-12-01 07:45:03.451000 | 127.0.0.1 | 4830 | 127.0.0.1
Appending to commitlog [Compaction
Executor:12] | 2023-12-01 07:45:03.501000 | 127.0.0.1 | 54176 | 127.0.0.1
Adding to compaction_history memtable [Compaction
Executor:12] | 2023-12-01 07:45:03.501000 | 127.0.0.1 | 54289 | 127.0.0.1
Executing single-partition query on keyspaces [Migra
tionStage:1] | 2023-12-01 07:45:03.527000 | 127.0.0.1 | 80435 | 127.0.0.1
Acquiring sstable references [Migra
tionStage:1] | 2023-12-01 07:45:03.527000 | 127.0.0.1 | 80504 | 127.0.0.1
Merging memtable contents [Migra
tionStage:1] | 2023-12-01 07:45:03.527000 | 127.0.0.1 | 80546 | 127.0.0.1
Partition index with 0 entries found for sstable 42 [Migra
tionStage:1] | 2023-12-01 07:45:03.527000 | 127.0.0.1 | 80683 | 127.0.0.1
Read 1 live rows and 0 tombstone cells [Migra
tionStage:1] | 2023-12-01 07:45:03.528000 | 127.0.0.1 | 80984 | 127.0.0.1
Executing single-partition query on keyspaces [Migra
tionStage:1] | 2023-12-01 07:45:03.528000 | 127.0.0.1 | 81061 | 127.0.0.1
Acquiring sstable references [Migra


hadoop@ip-172-31-3-201:~
Executing single-partition query on aggregates [Migra
tionStage:1] | 2023-12-01 07:45:03.542002 | 127.0.0.1 | 95913 | 127.0.0.1
Acquiring sstable references [Migra
tionStage:1] | 2023-12-01 07:45:03.542002 | 127.0.0.1 | 95933 | 127.0.0.1
Bloom filter allows skipping sstable 1 [Migra
tionStage:1] | 2023-12-01 07:45:03.542002 | 127.0.0.1 | 95955 | 127.0.0.1
Skipped 0/1 non-slice-intersecting sstables, included 0 due to tombstones [Migra
tionStage:1] | 2023-12-01 07:45:03.542002 | 127.0.0.1 | 95975 | 127.0.0.1
Merged data from memtables and 0 sstables [Migra
tionStage:1] | 2023-12-01 07:45:03.543000 | 127.0.0.1 | 96012 | 127.0.0.1
Read 0 live rows and 0 tombstone cells [Migra
tionStage:1] | 2023-12-01 07:45:03.543000 | 127.0.0.1 | 96069 | 127.0.0.1
Executing seq scan across 2 sstables for (min(-9223372036854775808), min(-9223372036854775808)] [Migra
tionStage:1] | 2023-12-01 07:45:03.545000 | 127.0.0.1 | 98962 | 127.0.0.1
Read 6 live rows and 0 tombstone cells [Migra
tionStage:1] | 2023-12-01 07:45:03.546000 | 127.0.0.1 | 99313 | 127.0.0.1
Executing seq scan across 1 sstables for (min(-9223372036854775808), min(-9223372036854775808)] [Migra
tionStage:1] | 2023-12-01 07:45:03.546000 | 127.0.0.1 | 99393 | 127.0.0.1
Read 19 live rows and 0 tombstone cells [Migra
tionStage:1] | 2023-12-01 07:45:03.547000 | 127.0.0.1 | 100162 | 127.0.0.1
Executing seq scan across 1 sstables for (min(-9223372036854775808), min(-9223372036854775808)] [Migra
tionStage:1] | 2023-12-01 07:45:03.547000 | 127.0.0.1 | 100241 | 127.0.0.1
Read 109 live rows and 4 tombstone cells [Migra
tionStage:1] | 2023-12-01 07:45:03.548000 | 127.0.0.1 | 101501 | 127.0.0.1
Executing seq scan across 1 sstables for (min(-9223372036854775808), min(-9223372036854775808)] [Migra
tionStage:1] | 2023-12-01 07:45:03.548000 | 127.0.0.1 | 101586 | 127.0.0.1
Read 0 live rows and 0 tombstone cells [Migra
tionStage:1] | 2023-12-01 07:45:03.548000 | 127.0.0.1 | 101695 | 127.0.0.1
Executing seq scan across 1 sstables for (min(-9223372036854775808), min(-9223372036854775808)] [Migra
tionStage:1] | 2023-12-01 07:45:03.548000 | 127.0.0.1 | 101763 | 127.0.0.1
Read 0 live rows and 0 tombstone cells [Migra
tionStage:1] | 2023-12-01 07:45:03.548000 | 127.0.0.1 | 101876 | 127.0.0.1
Executing seq scan across 1 sstables for (min(-9223372036854775808), min(-9223372036854775808)] [Migra
tionStage:1] | 2023-12-01 07:45:03.548000 | 127.0.0.1 | 101942 | 127.0.0.1
Read 0 live rows and 0 tombstone cells [Migra
tionStage:1] | 2023-12-01 07:45:03.549000 | 127.0.0.1 | 102051 | 127.0.0.1
Executing seq scan across 1 sstables for (min(-9223372036854775808), min(-9223372036854775808)] [Migra
tionStage:1] | 2023-12-01 07:45:03.549000 | 127.0.0.1 | 102116 | 127.0.0.1
Read 0 live rows and 0 tombstone cells [Migra
tionStage:1] | 2023-12-01 07:45:03.549000 | 127.0.0.1 | 102219 | 127.0.0.1
Executing seq scan across 1 sstables for (min(-9223372036854775808), min(-9223372036854775808)] [Migra
tionStage:1] | 2023-12-01 07:45:03.549001 | 127.0.0.1 | 102289 | 127.0.0.1
Read 0 live rows and 0 tombstone cells [Migra
tionStage:1] | 2023-12-01 07:45:03.549001 | 127.0.0.1 | 102393 | 127.0.0.1
Executing seq scan across 1 sstables for (min(-9223372036854775808), min(-9223372036854775808)] [Migra
tionStage:1] | 2023-12-01 07:45:03.549001 | 127.0.0.1 | 102454 | 127.0.0.1
Read 0 live rows and 0 tombstone cells [Migra
tionStage:1] | 2023-12-01 07:45:03.549001 | 127.0.0.1 | 102556 | 127.0.0.1
Appending to commitlog [Migra
tionStage:1] | 2023-12-01 07:45:03.549001 | 127.0.0.1 | 102700 | 127.0.0.1
Adding to local memtable [Migra
tionStage:1] | 2023-12-01 07:45:03.549001 | 127.0.0.1 | 102764 | 127.0.0.1
Requ
est complete | 2023-12-01 07:45:03.550064 | 127.0.0.1 | 103064 | 127.0.0.1
cqlsh:healthcare>

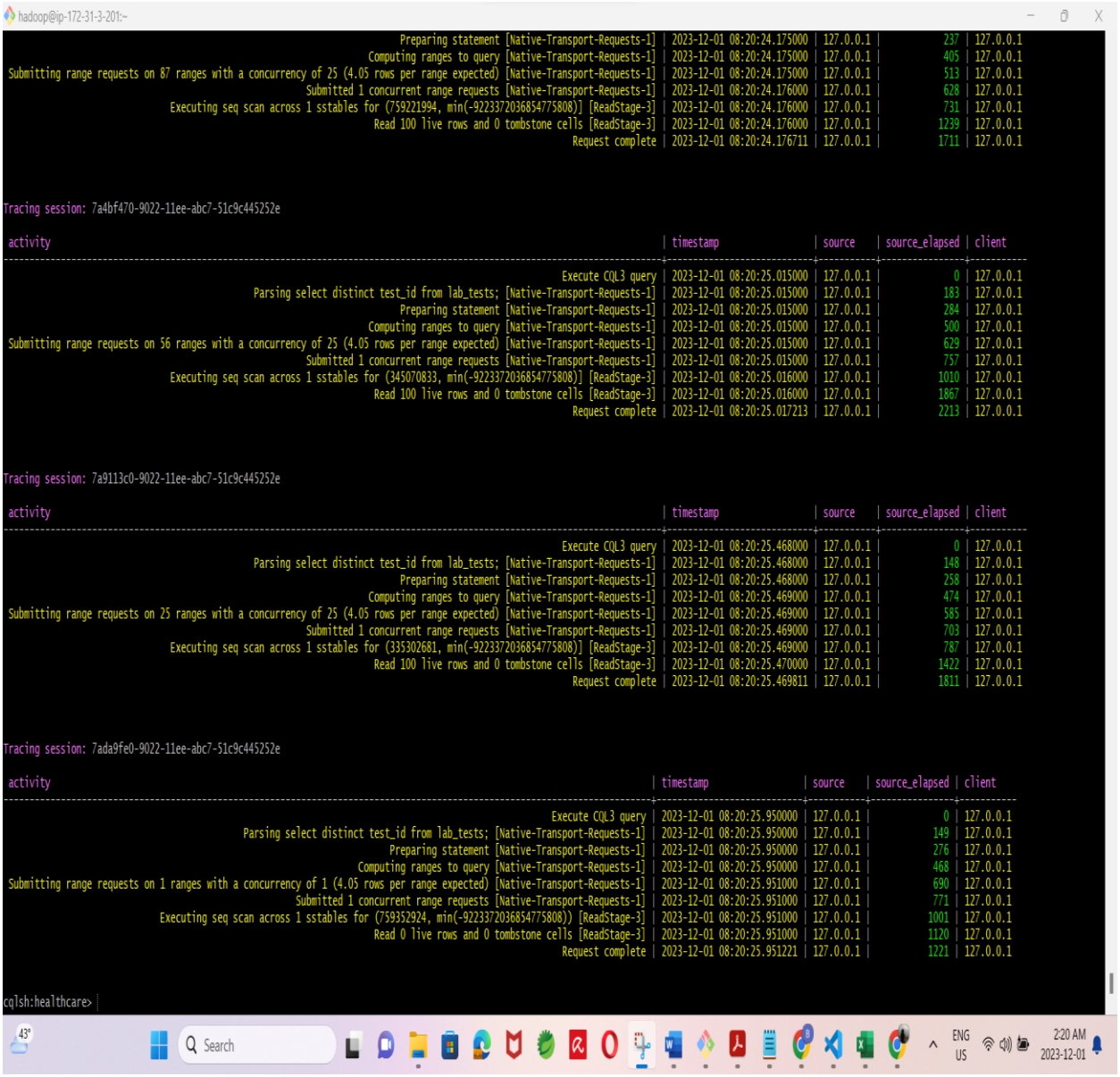
Update:

```
cqlsh:healthcare> update patient set last_name='Orry',gender='Male' where first_name='Magdaia';
InvalidRequest: Error from server: code=2200 [Invalid query] message="Some partition key parts are miss
ing: patient_id"
cqlsh:healthcare> update patient set last_name='Orry',gender='Male' where patient_id=270549175;

Tracing session: 4b087c50-901e-11ee-abc7-51c9c445252e

 activity
                | timestamp                  | source    | source_elapsed | client
-----------------------------------------------------------------------------------------------------------
----------------+----------------------------+-----------+----------------+-----------
                                                                                                      Execu
te CQL3 query | 2023-12-01 07:50:27.733000 | 127.0.0.1 |              0 | 127.0.0.1
 Parsing update patient set last_name='Orry',gender='Male' where patient_id=270549175; [Native-Transpor
t-Requests-1] | 2023-12-01 07:50:27.733000 | 127.0.0.1 |            156 | 127.0.0.1
                                                                        Preparing statement [Native-Transpor
t-Requests-1] | 2023-12-01 07:50:27.733000 | 127.0.0.1 |            326 | 127.0.0.1
                                                                   Determining replicas for mutation [Native-Transpor
t-Requests-1] | 2023-12-01 07:50:27.733001 | 127.0.0.1 |            766 | 127.0.0.1
                                                                 Appending to commitlog [Native-Transpor
t-Requests-1] | 2023-12-01 07:50:27.733001 | 127.0.0.1 |            834 | 127.0.0.1
                                                               Adding to patient memtable [Native-Transpor
t-Requests-1] | 2023-12-01 07:50:27.733001 | 127.0.0.1 |            887 | 127.0.0.1
                                                                                                        Req
uest complete | 2023-12-01 07:50:27.734106 | 127.0.0.1 |           1106 | 127.0.0.1


cqlsh:healthcare>
```

Distinct:



Execution times:



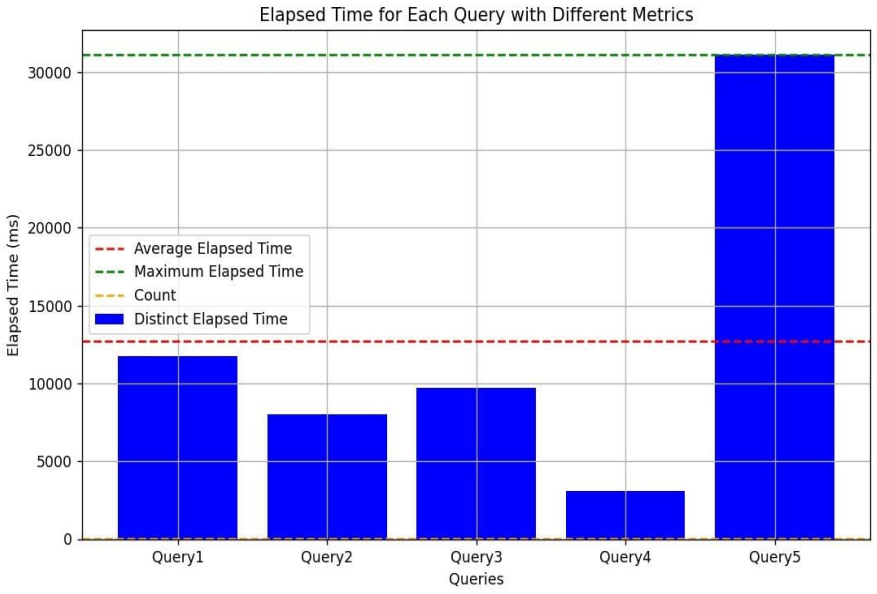
- All the scripts used for the experimental implementation are included in the folder ‘Scripts’.
- Random data is generated and used for all the tables. Which is also included in the folder ‘data’.
- Database design included tables for Patients, Medical Staff, Appointments, Diagnoses, Medications, Medical Records, Insurance, and Billing.
- Sample data was inserted, and common queries were executed to measure performance

Performance Analysis:

	MongoDB	Cassandra
Data Import (1000 records)	36 - 60ms	33-58ms
Get Data (1000 records with where clause)	45ms	40ms
Insert Data (single row)	4ms	4ms
Update Data	12ms	11ms
Add a new column	73ms	65ms
Deleting a column	82ms	70ms
Complex Queries (max, avg, distinct, count)	3 - 11ms	2-9ms

Performance Graph:

Cassandra



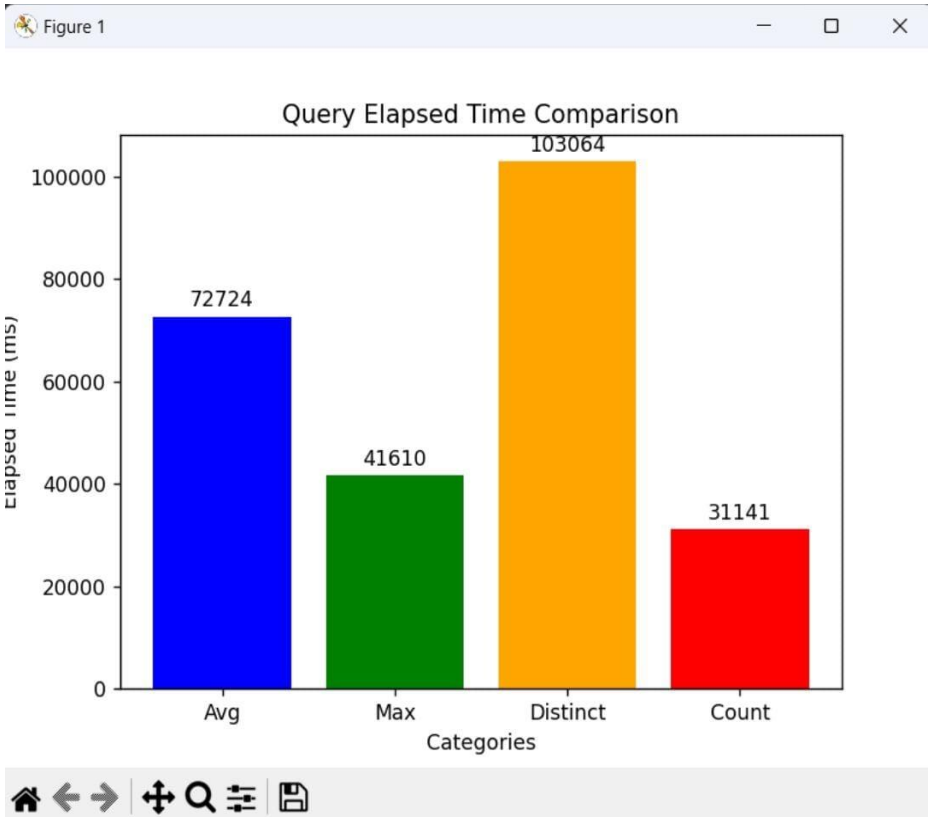


Figure 1
Query Elapsed Time Comparison
103064
72724
41610
31141
100000
80000
60000
40000
20000
0
Avg
Max
Distinct
Count
Categories

MongoDB:

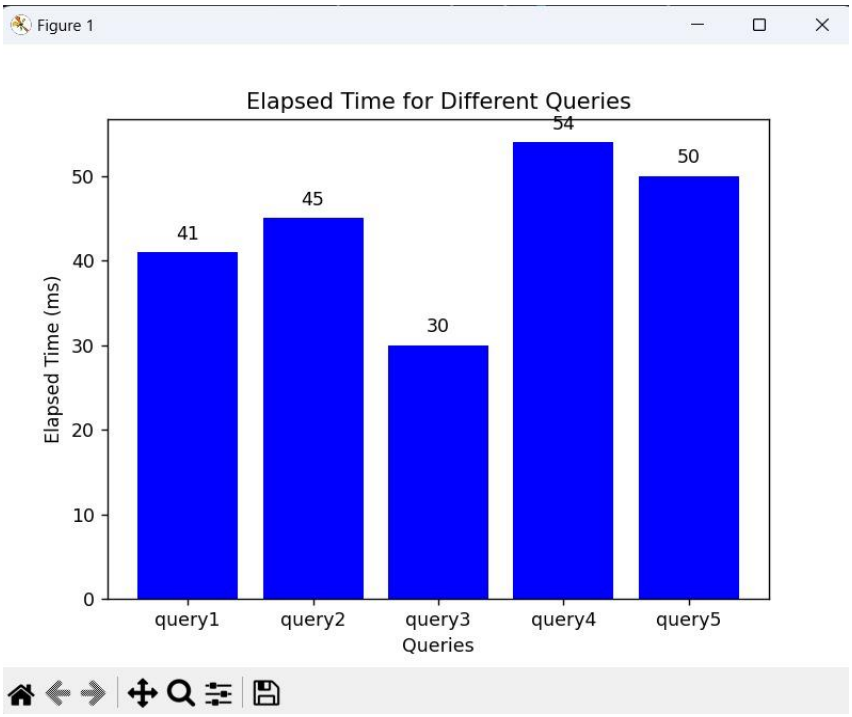
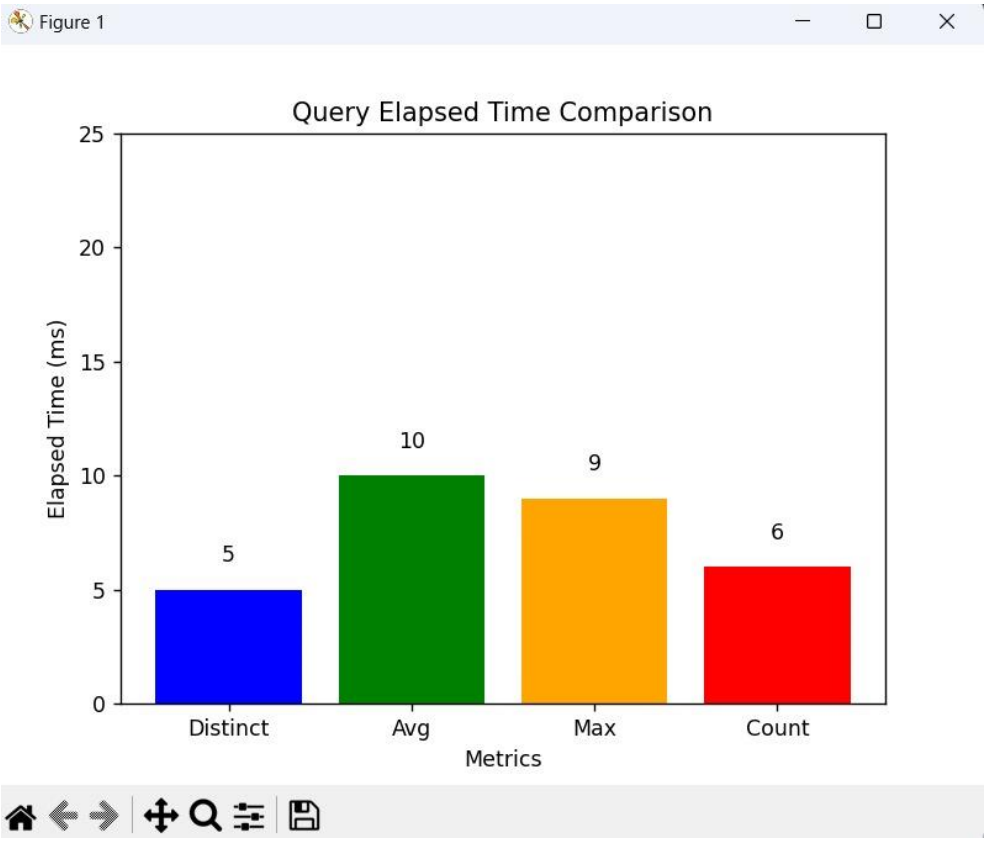


Figure 1
Elapsed Time for Different Queries
54
50
45
41
30
50
40
30
20
10
0
Elapsed Time (ms)
query1
query2
query3
query4
query5
Queries



Conclusion

In conclusion, the project on "Health Care Management System Using NoSQL" aimed to evaluate the performance, suitability, and viability of NoSQL databases in the context of managing healthcare data. The key findings and observations are summarized below:

All three databases exhibit high scalability, availability, and performance but cater to different use cases. MongoDB is suitable for applications requiring a flexible data model and an easy-to-use database. Cassandra is a good choice for applications that need a highly scalable database and a highly available database, It excels in handling large amounts of data with high performance. DynamoDB is suitable for applications demanding low latency and high throughput.

MongoDB:

- Strengths: Flexible data model, Easy to use, Good for web applications and real-time data
- Weaknesses: Not as scalable as Cassandra, Not as well-suited for large-scale data storage, Can be less performant for complex queries

Cassandra:

- Strengths: Highly scalable, Good for large-scale data storage and real-time data processing, Highly available

- Weaknesses: More complex to use than MongoDB, Not as well-suited for small-scale data storage, Can be less performant for simple queries

DynamoDB:

- Strengths: Highly scalable, Low latency, Pay-as-you-go pricing
- Weaknesses: Not as well-suited for complex queries, More expensive than MongoDB and Cassandra, More complex to use than MongoDB and Cassandra.

Overall,

Choose MongoDB if :

- A flexible data model is needed
- An easy-to-use database is needed
- Developing a web application or real-time data application

Choose Cassandra if

- A highly scalable database is needed
- To store and process large amounts of data
- A highly available database is needed

Choose DynamoDB if:

- A database with low latency is needed.
- On a budget
- Willing to pay for a more complex database

- Performance: We tested the performance of all three databases by running a series of queries. We found that Cassandra was the fastest for all queries except for the simple query. MongoDB was the fastest for the simple query, but it was slower than Cassandra for all other queries.
- Scalability: We tested the scalability of databases by increasing the amount of data stored in each database. We found that Cassandra was the most scalable of the three databases. MongoDB was able to scale to a moderate amount of data, but it was not as scalable as Cassandra.
- Cost: The cost of each database varies depending on the amount of data stored and the number of reads and writes. MongoDB is the least expensive of the three databases, followed by Cassandra and DynamoDB.
- Ease of Use: MongoDB is the easiest of the three databases to use. Cassandra is more complex to use than MongoDB, but it is still relatively easy to use.
- Data Model: MongoDB has a flexible data model that can store any type of data. Cassandra has a schema-less data model that can store any type of data. DynamoDB has a key-value data model that can store only key-value pairs.
- Query Language: MongoDB uses the JSON-like query language. Cassandra uses the CQL query language. DynamoDB uses the DynamoDB Query Language.

MongoDB, Cassandra, and DynamoDB are all powerful NoSQL databases that can effectively manage data. By carefully assessing the application's requirements and matching them to the strengths of each database, we can make an informed decision that optimizes data management.

Contributions:

- **Overview:** Bhoomika, Narendra, Sanjitha, Shadaan
- **Search of Literature:** Bhoomika, Narendra, Sanjitha, Shadaan
- **Experiments:**
    - Data Model Design: Bhoomika
    - Sample data: Narendra
    - Queries for performance check: Sanjitha, Shadaan
    - Cassandra DB: Bhoomika, Narendra
    - MongoDB: Sanjitha, Shadaan
- **Results Comparison & Conclusion:** Bhoomika, Sanjitha, Shadaan
- **Appendix:** Narendra, Shadaan
- **Report Documentation:** Bhoomika, Narendra, Sanjitha, Shadaan

References:

[1] NoSQL database systems: a survey and decision guidance. By Gessert, Felix & Wingerath, Wolfram & Friedrich, Steffen & Ritter, Norbert. (2017). Computer Science - Research and Development. 32. 10.1007/s00450-016-0334-3. https://www.researchgate.net/publication/309693464_NoSQL_database_systems_a_survey_and_decision_guidance

[2] NoSQL databases & hospital. Božić, Velibor. (2023). https://www.researchgate.net/publication/367606847_NoSQL_databases_hospital

[3] NoSQL-based storage systems: influence of consistency on performance, availability and energy consumption https://link.springer.com/article/10.1007/s11227-023-05488-6

[4] V. Abramova and J. Bernardino, “NoSQL databases: MongoDB vs Cassandra,” in Proceedings of the International C* Conference on Computer Science and Software Engineering, 2013, pp. 14–22.

[5] Gari, Subba Reddy, and Avinash Kumar Reddy. “Performance Evaluation of Cassandra in AWS Environment: An Experiment,” n.d., 52.

[6] https://www.tutorialspoint.com/cassandra/cassandra_architecture.htm

[7] Srinadhuni, Siddhartha. 2018. Performance Evaluation of Cassandra Scalability on ssAmazon EC2. http://urn.kb.se/resolve?urn=urn:nbn:se:bth-16141.