Overview on Cassandra

August 16, 2017

Authored by: Suja Vasudevan (215830)

Table of Contents

1. Document Purpose:

The Purpose of this document to give an overview about Cassandra to those who are beginners and would like to know what Cassandra is all about. This document helps to understand the various features and commands in Cassandra.

2. Document Scope:

This document explains the basic knowledge to be aware of and the features that Cassandra comprises.

3. Audience:

This document is purposeful for the beginners who would like to know an overview of Cassandra.

4. Introduction:

Cassandra is a highly scalable, distributed and high-performance NoSQL database. It is designed to handle huge amounts of data across many commodity servers, providing high availability without a single point of failure. Data is placed on different machines with more than one replication factor to attain a high availability. It is an open source and column oriented database.

5. What is NoSQL Database?

A NoSQL database is a database that provides a mechanism to store and retrieve data other than the tabular relations used in relational databases. These databases are schema-free, support easy replication, have simple API, eventually consistent, and can handle huge amounts of data.
The characteristics of a NoSQL database are

- Design simplicity
- Horizontal scaling
- High availability

NoSQL databases use different data structures compared to relational databases i.e., structured, unstructured and semi-structured data. It makes some operations faster in NoSQL. It is deployed in a horizontal fashion and has a decentralized structure.

6. When did Cassandra come into this world?

- Cassandra was developed at Facebook for inbox search.
- Facebook open sourced it in July 2008.

- Apache Incubator accepted it in March 2009.
- It was made an Apache top-level project since February 2010.

7. Features of Cassandra

- **High scalability** - Cassandra is highly scalable. It allows to add additional hardware to accommodate additional customers and additional data as per the requirement.

- **Master less architecture** - Cassandra has no single point of failure and data can be written or read on any node.

- **Linear-scale performance** - Cassandra is linearly scalable, i.e., it increases the throughput as the number of nodes in the cluster increases.

- **Flexible data storage** - Cassandra accommodates all possible data formats including: structured, semi-structured, and unstructured data. It can dynamically accommodate changes to the data structures according to the requirements.

- **Multi Data Center Replication** - Cassandra provides the flexibility to distribute data where it is needed by replicating data across multiple data centers.

- **Transaction support** - Cassandra supports properties like Atomicity, Consistency, Isolation, and Durability (ACID).

- **Flexible and Dynamic Data Model** - Cassandra was designed to run on cheap commodity hardware. It performs blazingly fast writes and can store hundreds of terabytes of data, without sacrificing the read efficiency.

- **Data Compression:** Cassandra can compress up to 80% data without any overhead.

- **Cassandra Query language:** Cassandra provides query language that is similar like SQL language. It is very easy for relational database developers moving from relational database to Cassandra. Cassandra provides a prompt Cassandra query language shell (cqlsh).

8. cqlsh and its commands

8.1 Shell Commands

The shell commands in cqlsh are below

- **HELP** - Displays help topics for all cqlsh commands.

Eg: cqlsh> help

- **CAPTURE** - Captures the output of a command and inserts it into a file.

  Eg: cqlsh> CAPTURE '<output file>'

- **CONSISTENCY** - Shows the current consistency level, or sets a new consistency level.

  Eg: cqlsh> CONSISTENCY

- **COPY** - Copies data to and from Cassandra. Below example copies data from a table to file.

  Eg: cqlsh> COPY emp (emp_id, emp_name, emp_sal) TO 'output1file';

- **DESCRIBE** - Describes the current cluster of Cassandra and its objects. Below example lists the tables and the next shows the table structure.

  Eg: cqlsh> describe tables;

  emp

  Eg: cqlsh> describe emp;

  Create table emp
  (emp_id PRIMARY KEY,
  emp_name text,
  emp_sal varint)

- **EXPAND** - Expands the output of a query vertically.

  Eg:
  cqlsh> expand on;
  cqlsh> select * from emp;
  @ Row 1
  ----------+----------
  emp_id | 1001
  emp_name | Mary
  emp_sal | 50000

  @ Row 2
  ----------+----------
  emp_id | 1002

```
 emp_name | Philip
 emp_sal | 60000

cqlsh> expand off;
```

- **EXIT** - This command helps to terminate from cqlsh.

  Eg: cqlsh> exit

- **SHOW** - Displays the details of current cqlsh session such as Cassandra version, host, or data type assumptions.

  Eg:
  cqlsh> show host;
  cqlsh> show version;

- **SOURCE** - Executes a file that contains CQL statements. Below shows the rows in emp table which is the CQL statement in the file.

  Eg:
  Contents of outfile – select * from emp
  cqlsh> source 'outfile';

```
 emp_id |  emp_name |  emp_sal
--------+-----------------+----------+
   1001 |  Mary          | 50000
   1002 |  Philip        | 60000
(2 rows)
```

8.2 CQL Data Definition Commands

- **CREATE KEYSPACE** - Creates a Key Space in Cassandra.

  Syntax: CREATE KEYSPACE "Key Space Name" WITH replication = {'class': 'Strategy name', 'replication_factor': 'No. of replicas'} AND durable_writes = 'Boolean value';

  Strategy Name can be Simple Strategy or Network Topology Strategy or Old Network Topology Strategy

  Eg: cqlsh> CREATE KEYSPACE myspace WITH REPLICATION = {'class': 'NetworkTopologyStrategy', 'datacenter1' : 3} AND DURABLE_WRITES = false;

- **USE** - Connects to a created Key Space.

Syntax: USE <identifier>

Eg: cqlsh> USE myspace

Prompt would be as cqlsh:myspace>

- **ALTER KEYSPACE** - Changes the properties of a Key Space. Below example alters the durable writes to true.

  Syntax: ALTER KEYSPACE "Key Space Name" WITH replication = {'class': 'Strategy name', 'replication_factor': 'No. of replicas'};

  Eg: cqlsh> ALTER KEYSPACE myspace WITH REPLICATION = {'class': 'NetworkTopologyStrategy', 'datacenter1': 3} AND DURABLE_WRITES = true;

- **DROP KEYSPACE** - Removes a Key Space

  Syntax: DROP KEYSPACE "Key Space name"

  Eg: cqlsh> DROP KEYSPACE myspace

- **CREATE TABLE** - Creates a table in a Key Space.

  Syntax: CREATE TABLE tablename (
  Column1 name datatype PRIMARYKEY,
  Column2 name data type,
  Column3 name data type.
)

  Eg: cqlsh:myspace> Create table emp (emp_id PRIMARY KEY, emp_name text, emp_sal varint);

- **ALTER TABLE** - Modifies the column properties of a table.

  Syntax: ALTER (TABLE | COLUMNFAMILY) <tablename> <instruction>

  Eg: cqlsh:myspace> ALTER TABLE emp ADD emp_email text;
  cqlsh:myspace> ALTER TABLE emp DROP emp_email;

- **DROP TABLE** - Removes a table.

  Syntax: DROP TABLE <tablename>

Eg: cqlsh:myspace> DROP TABLE emp;

- **TRUNCATE** - Removes all the data from a table.

  Syntax: TRUNCATE <tablename>

  Eg: cqlsh:myspace> TRUNCATE student;

- **CREATE INDEX** - Defines a new index on a single column of a table.

  Syntax: CREATE INDEX <identifier> ON <tablename>

  Eg: cqlsh:myspace> CREATE INDEX name ON emp (emp_name);

- **DROP INDEX** - Deletes a named index.

  Syntax: DROP INDEX <identifier>

  Eg: cqlsh:myspace> drop index name;

8.3 CQL Data Manipulation Commands

- **INSERT** - Adds columns for a row in a table.

  Syntax: INSERT INTO <tablename> (<column1 name>, <column2 name>....) VALUES (<value1>, <value2>....) USING <option>

  Eg: cqlsh:myspace> INSERT INTO emp (emp_id, emp_name, emp_sal) VALUES(1001,'Mary', 50000);
  cqlsh:myspace> INSERT INTO emp (emp_id, emp_name, emp_sal) VALUES(1002,'Philip', 60000);

- **UPDATE** - Updates a column of a row.

  Syntax: UPDATE <tablename> SET <column name> = <new value>
  <Column name> = <value> WHERE <condition>

  Eg: cqlsh:myspace> UPDATE emp SET emp_sal=50000 WHERE emp_id=1002;

- **DELETE** - Deletes data from a table.

  Syntax: DELETE FROM <identifier> WHERE <condition>;

  Eg: cqlsh:myspace> DELETE emp_sal FROM emp WHERE emp_id=1002;

- **BATCH** - Executes multiple DML statements at once.

  Syntax: BEGIN BATCH
  <insert-stmt>/ <update-stmt>/ <delete-stmt>
  APPLY BATCH

  Eg: cqlsh:myspace> BEGIN BATCH
  INSERT INTO emp (emp_id, emp_name, emp_sal) values (1003,'Bean', 60000);
  UPDATE emp SET emp_sal = 55000 WHERE emp_id =1002;
  DELETE emp_sal FROM emp WHERE emp_id = 1001;
  APPLY BATCH;

8.4 CQL Clauses

- **SELECT** - This clause reads data from a table

- **WHERE** - The where clause is used along with select to read a specific data.

  Syntax: SELECT FROM <table name> WHERE <condition>;

  Eg: SELECT * FROM emp WHERE emp_id=1002;

9 .Conclusion:

This is an insight into Cassandra and its features. This would help you to step into the Big Data world from the Querying world. It's a fantastic solution to Big Data challenge.