**Introduction**

Apache Cassandra is a distributed database for managing large amounts of structured data across many commodity servers, while providing highly available service and no single point of failure. Cassandra offers capabilities that relational databases and other NoSQL databases simply cannot match such as: continuous availability, linear scale performance, operational simplicity and easy data distribution across multiple data centers and cloud availability zones.

* **Cassandra Architecture Overview**

Cassandra’s architecture is responsible for its ability to scale, perform, and offer continuous uptime. Rather than using a legacy master-slave or a manual and  difficult-to-maintain sharded architecture, Cassandra has a masterless “ring” design that is elegant, easy to setup, and easy to maintain.

In Cassandra, all nodes play an identical role; there is no concept of a master node, with all nodes communicating with each other equally. Cassandra’s built-for-scale architecture means that it is capable of handling large amounts of data and thousands of concurrent users or operations per second -even across multiple data centers- as easily as it can manage much smaller amounts of data and user traffic.

Cassandra’s architecture also means that, unlike other master-slave or sharded systems, it has no single point of failure and therefore is capable of offering true continuous availability and uptime — simply add new nodes to an existing cluster without having to take it down.

[A diagram of a structure

AI-generated content may be incorrect.](http://localhost/wp-content/uploads/2023/09/apache-cassandra-diagrams-01.jpg)

* **Cassandra Model**

Cassandra is considered a **schema-less** data-store, but it is necessary to perform some configuration specific to your application.

**Keyspaces** are the upper-most namespace in Cassandra and typically you’ll see exactly one for each application. For each keyspace there are one or more **Column Families**(like tables). A column family is the namespace used to associate records of a similar kind.

Cassandra gives you record-level atomicity within a column family when making writes, and queries against them are efficient. These qualities are important to  
keep in mind when designing your data model.

In Cassandra there is no referential integrity, and the lack of support for secondary indexing makes it difficult to efficiently perform joins, so you must denormalize. You’re forced to think in terms of the queries you’ll perform and the results you expect since this is likely what the model will look like.

Another important distinction from traditional databases in that the order records are sorted is a design decision, and not something that can easily be changed later. A relational database will scan your table sequentially when performing a SELECT-WHERE like this, and since records are distributed throughout a Cassandra cluster based on key, the equivalent could mean contacting more than one node (possibly many). However, even with all of the data on a single machine, there comes a point when such an operation will become inefficient with a relational database, making it necessary to index the username attribute. Cassandra doesn’t currently support secondary indices like this.

In this tutorial we will work with the data from a fictitious video application that enables users to include information about the videos uploaded to a video-distribution system. You will learn how to:

1. Create KeySpaces and Tables.
2. Define Simple and Compound primary Keys.
3. Define Clusturing Columns.

All this tasks will be performed using the Cassandra CQL (*Cassandra Query Language*).

To execute the CQLs statement of this tutorial we will be using a Zeppelin note with the **cassandra**interpreter (**%cassandra**).

%cassandra

desc keyspaces;

**Creating KeySpaces and Tables**

To perform these tasks, we will start by creating a table schema and loading certain data from the videos. The video metadata consist of:

| **Column Name** | **Data Type** |
| --- | --- |
| video\_id | timeuuid |
| added\_date | timestamp |
| description | text |
| title | text |
| user\_id | uuid |

1. **Create the KeySpace**

Using CQL, we create a keyspace named **tutorial** and perform the **USE**command to switch to that keyspace. Use **SimpleStrategy**as the replication class and a replication factor of 1. (These parameters are related to hardware architecture where Cassandra is running)

%cassandra

# Keyspace creation for videos website

CREATE KEYSPACE IF NOT EXISTS tutorial

WITH REPLICATION = {

'class' : 'SimpleStrategy',

'replication\_factor': 1

};

%cassandra

# Change to tutorial Keyspace

Use tutorial;

1. **Create the Table**

Create the videos table with the required structure, with **video\_id** as the primary key.

%cassandra

# Videos table creation

CREATE TABLE IF NOT EXISTS videos (

video\_id TIMEUUID,

added\_date TIMESTAMP,

description TEXT,

title TEXT,

user\_id UUID,

PRIMARY KEY(video\_id)

);

1. **Load data Into The Table**

Load the newly created table with the videos.csv file using the COPY command. COPY does not require column names when the schema columns of the target table match the order of the fields in the CSV source file.

***Note:****The COPY command is only available in Cassandra Shell (CQSH). therefore to execute this command from Zeppelin we need to run it using the Shell interpreter (%sh).*

%sh

# Load videos.csv into videos table

cqlsh -e "COPY tutorial.videos (video\_id, added\_date, description, title, user\_id) FROM '/home/training/Data/videos.csv' WITH HEADER=true;"

1. **Check Data Is Loaded**

Use SELECT to verify that the data have been loaded properly. Include LIMIT to retrieve only the first 10 rows.

%cassandra

# Check the data is loaded

select \* from videos limit 10;

Count the number of rows in the table. The number must match the number of rows indicated by the COPY command.

%cassandra

# Get the number of rows

select count(\*) from videos;

Use SELECT with WHERE to get the data of a video with the title = ‘Become An Internet Enterprise’.

**Note**: The following query will not execute.

%cassandra

// select video by title

select \* from videos where title = 'Become An Internet Enterprise';

**Cause of the issue**

Cannot execute this query as it might involve data filtering and thus may have unpredictable performance.

If you want to execute this query despite the performance unpredictability, use ALLOW FILTERING

If the above error is thrown, it means that Cassandra assumes that it might not be able to execute the query is efficient manner. The possible option is to fetch all the data and filter out the data with non-matching value for the column. So Cassandra is warning you about this behavior and passing you the selection of choice correctly to solve the issue. Suggesting to “ALLOW FILTERING” does not mean it has to be done blindly. It is always better to revisit the data model.

**Tables Primary Keys**

In this section we will create a new table that enables you to find the videos by title and year (bear in mind that there may be videos with the same title but from different years).

We need to consult the data of the videos by searching by title and year (do not use ALLOW FILTERING if it is unnecessary to do so). The columns for the new table are:

| **Column Name** | **Data Type** |
| --- | --- |
| title | text |
| added\_year | int |
| added\_date | timestamp |
| description | text |
| user\_id | uuid |
| video\_id | uuid |

1. **Simple Primary Key**

We will create a second table called videos\_by\_title\_year with the above-mentioned structure, setting the combination of title and added\_year as primary key (since two videos with the same title are not allowed in the same year).

Write for this: PRIMARY KEY (title, added\_year).

%cassandra

// Create the new table

CREATE TABLE IF NOT EXISTS tutorial.videos\_by\_title\_year (

title TEXT,

added\_year INT,

added\_date TIMESTAMP,

description TEXT,

user\_id UUID,

video\_id UUID,

PRIMARY KEY(title, added\_year) // for querying by title and added\_year we need composite key

);

1. **Load data Into The Table**

Load the newly created table with the videos\_by\_title\_year.csv file using the COPY command. COPY does not require column names when the schema columns of the target table match the order of the fields in the CSV source file.

%sh

# Load the data into the table

cqlsh -e "copy tutorial.videos\_by\_title\_year from '/home/training/Data/videos\_by\_title\_year.csv' WITH DELIMITER = ',' AND HEADER = true and DATETIMEFORMAT='%d/%m/%Y';"

1. **Query The Table**

Get the data of videos with a specific title and year, such as:

title= 'Introduction To Apache Cassandra'

year= 2014

%cassandra

// select video by title and added\_year

select \* from videos\_by\_title\_year where title = 'Introduction To Apache Cassandra' and added\_year = 2014;

1. **Compound Primary Key**

Let’s Drop the table  and create it again but now writing PRIMARY KEY ((title, added\_year)) – pay attention to the double parentheses! – and execute the query from the previous steps.

%cassandra

// Drop the existing table

drop table videos\_by\_title\_year;

%cassandra

// create the table again with compound key

CREATE TABLE IF NOT EXISTS videos\_by\_title\_year (

title TEXT,

added\_year INT,

added\_date TIMESTAMP,

description TEXT,

user\_id UUID,

video\_id UUID,

PRIMARY KEY((title, added\_year)) // for querying by title and added\_year we need composite key

);

%sh

# Load the data into the table

cqlsh -e "copy tutorial.videos\_by\_title\_year from '/home/training/Data/videos\_by\_title\_year.csv' WITH DELIMITER = ',' AND HEADER = true and DATETIMEFORMAT='%d/%m/%Y';"

Let’s rerun the query from the previous section.

**Note**: The following query will not execute.

%cassandra

// select video by title

select \* from videos where title = 'Introduction To Apache Cassandra';

**The reason:**

hash function requires all partition key columns(‘…PRIMARY KEY((title, added\_year))…’), otherwise it returns incorect hash -> in fact it is not possible to find node without all columns from partition key.

%cassandra

// select video by title and added\_year

select \* from videos\_by\_title\_year where title = 'Introduction To Apache Cassandra' and added\_year = 2014;

**Clustering Columns**

Clustering columns determines the order of data in partitions. The whole point of a **column-oriented database** like Cassandra is to put adjacent data records next to each other for fast retrieval.

We want to assign tags to the videos and then search for videos with a specific tag and for those with a specific tag and from a specific year or range of years (before A, or after B, or between C and D, etc.).

In this section we will create a new table that enables you to search for videos by tag and range of years. To do so, we will create a new table.

We need to consult the data of the videos by searching by title and year (do not use ALLOW FILTERING if it is unnecessary to do so). The columns for the new table are:

| **Column Name** | **Data Type** |
| --- | --- |
| tag | text |
| added\_year | int |
| video\_id | uuid |
| added\_date | timestamp |
| description | text |
| title | text |
| user\_id | uuid |

1. **Defining Clustured Columns**

We will create a new table called videos\_by\_tag\_year  with the above-mentioned structure.

%cassandra

// Create table

CREATE TABLE videos\_by\_tag\_year (

tag text,

added\_year int,

video\_id timeuuid,

added\_date timestamp,

description text,

title text,

user\_id uuid,

PRIMARY KEY ((tag), added\_year, video\_id)

) WITH CLUSTERING ORDER BY (added\_year desc, video\_id asc);

1. **Load data Into The Table**

Load the newly created table with the videos\_and\_tags.csv file using the COPY command.

%sh

# Load the data again into the table

cqlsh -e "copy tutorial.videos\_by\_tag\_year (video\_id, added\_year, added\_date, description, tag, title, user\_id) FROM '/home/training/Data/videos\_and\_tags.csv' WITH HEADER=true;"

Load the data from the videos\_encoding.csv file using the COPY command.

%sh

# Load the data again into the table

cqlsh -e "copy tutorial.videos\_by\_tag\_year (video\_id, added\_year, added\_date, description, tag, title, user\_id) FROM '/home/training/Data/videos\_and\_tags.csv' WITH HEADER=true;"

1. **Query The Table**

Get the data of videos with a specific tag and year, such as:

tag= 'azure'

added\_year= between 2011 and 2016

%cassandra

// Query the table

select \* from videos\_by\_tag\_year

WHERE tag = 'azure'

and added\_year > 2012

and added\_year < 2016

ORDER BY added\_year;

**Defining New Types**

After reviewing the structure of the tables, we want to add the **tags** for a video to the original table videos and include information about the video encoding. To do so, we will add a column of type SET and another with a compound type defined by the user.

Collection columns are multi-valued columns designed to store a small amount of data that are always retrieved in their entirety. A collection cannot be nested inside another collection.

SET: Typed collection of unique values / Ordered by values.  
LIST: Typed collection of non-unique values / Ordered by position.  
MAP: Typed collection of key-value pairs / Ordered by unique keys.

The altered videos table structure is:

| **Column Name** | **Data Type** |
| --- | --- |
| video\_id | timeuuid |
| added\_date | timestamp |
| description | text |
| encoding | video\_encoding |
| tags | set<text> |
| title | text |
| user\_id | uuid |

The encoding data structure is described bellow:

| **Column Name** | **Data Type** |
| --- | --- |
| bit\_rates | set<text> |
| encoding | text |
| height | int |
| width | int |

1. **Creating User Defined Type**

Create a new data type to save the encoding of the videos, with the following command:

%cassandra

// Create User Defined Type

CREATE TYPE video\_encoding (

encoding TEXT,

height INT,

width INT,

bit\_rates SET <TEXT>

);

1. **Altering The Table**

Will start by truncating the videos table to remove the data. From here we will alter the table to add the **tags** and **video\_encoding** columns. Then we load the table with the given CSV file.

%cassandra

// Remove all the rows from the table

TRUNCATE Table videos;

%cassandra

// Add a new column to the table

ALTER TABLE videos ADD tags SET<TEXT>;

Load videos\_and\_tags\_set.csv into videos table.

%sh

# Load tags into videos table

cqlsh -e "COPY tutorial.videos (video\_id, added\_date, description, tags, title, user\_id) FROM '/home/training/Data/videos\_and\_tags\_set.csv' WITH HEADER=true;"

Add the encoding column to videos table.

%cassandra

// Add encoding UDT to the table

ALTER TABLE videos ADD encoding FROZEN<video\_encoding>;

Load videos\_encodings.csv into videos table.

%sh

# Load videos\_encodings.csv into videos table

cqlsh -e "COPY tutorial.videos (video\_id, encoding) FROM '/home/training/Data/videos\_encoding.csv' WITH HEADER=true;"

1. **Query The Table**

Run a query to retrieve the first 10 rows of the videos table.

%cassandra

// Retrieve the first 10 rows of the videos table.

SELECT \* FROM videos LIMIT 10;

**Summary**

Cassandra is a great tool for developers and especially for organizations that have huge quantities of data to manage. Its amazing features and architecture make it the right choice for a lot of businesses.  In this tutorial we covered Cassandra fundamentals CQL commands to create and manage tables. We learned how to define simple and compound primary keys as defining clustering columns. Also we discovered how to create a User Defined Type and how to import files into Cassandra tables.

Open Zeppelin Note

[First Steps With Cassandra](http://localhost:19995/#/notebook/2JBD91H4H)