# cassandra

cassandra query language

tutorialspoint

SIMPLYEASYLEARNING

www.tutorialspoint.com

# About the Tutorial

Cassandra is a distributed database from Apache that is highly scalable and designed to manage very large amounts of structured data. It provides high availability with no single point of failure.

The tutorial starts off with a basic introduction of Cassandra followed by its architecture, installation, and important classes and interfaces. Thereafter, it proceeds to cover how to perform operations such as create, alter, update, and delete on keyspaces, tables, and indexes using CQLSH as well as Java API. The tutorial also has dedicated chapters to explain the data types and collections available in CQL and how to make use of user-defined data types.

# Audience

This tutorial will be extremely useful for software professionals in particular who aspire to learn the ropes of Cassandra and implement it in practice.

# Prerequisites

It is an elementary tutorial and you can easily understand the concepts explained here with a basic knowledge of Java programming. However, it will help if you have some prior exposure to database concepts and any of the Linux flavors.

# Copyright & Disclaimer

# Table of Contents

# Part 1: Cassandra Basics

# 1. INTRODUCTION

Apache Cassandra is a highly scalable, high-performance distributed database designed to handle large amounts of data across many commodity servers, providing high availability with no single point of failure. It is a type of NoSQL database. Let us first understand what a NoSQL database does.

## NoSQL Database

A NoSQL database (sometimes called as Not Only SQL) 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 primary objective of a NoSQL database is to have

- simplicity of design,
- horizontal scaling, and
- finer control over availability.

NoSql databases use different data structures compared to relational databases. It makes some operations faster in NoSQL. The suitability of a given NoSQL database depends on the problem it must solve.

## NoSQL vs. Relational Database

The following table lists the points that differentiate a relational database from a NoSQL database.

| Relational Database | NoSql Database |
|---|---|
| Supports powerful query language. | Supports very simple query language. |
| It has a fixed schema. | No fixed schema. |
| Follows ACID (Atomicity, Consistency, Isolation, and Durability). | It is only “eventually consistent”. |
| Supports transactions. | Does not support transactions. |

Besides Cassandra, we have the following NoSQL databases that are quite popular:

- **Apache HBase:** HBase is an open source, non-relational, distributed database modeled after Google's BigTable and is written in Java. It is developed as a part of Apache Hadoop project and runs on top of HDFS, providing BigTable-like capabilities for Hadoop.

- **MongoDB:** MongoDB is a cross-platform document-oriented database system that avoids using the traditional table-based relational database structure in favor of JSON-like documents with dynamic schemas making the integration of data in certain types of applications easier and faster.

# What is Apache Cassandra?

Apache Cassandra is an open source, distributed and decentralized/distributed storage system (database), for managing very large amounts of structured data spread out across the world. It provides highly available service with no single point of failure.

Listed below are some of the notable points of Apache Cassandra:

- It is scalable, fault-tolerant, and consistent.

- It is a key-value as well as a column-oriented database.

- Its distribution design is based on Amazon's Dynamo and its data model on Google's Bigtable.

- Created at Facebook, it differs sharply from relational database management systems.

- Cassandra implements a Dynamo-style replication model with no single point of failure, but adds a more powerful "column family" data model.

- Cassandra is being used by some of the biggest companies such as Facebook, Twitter, Cisco, Rackspace, ebay, Twitter, Netflix, and more.

# Features of Cassandra

Cassandra has become so popular because of its outstanding technical features. Given below are some of the features of Cassandra:

- **Elastic scalability:** Cassandra is highly scalable; it allows to add more hardware to accommodate more customers and more data as per requirement.

- **Always on architecture:** Cassandra has no single point of failure and it is continuously available for business-critical applications that cannot afford a failure.

- **Fast linear-scale performance:** Cassandra is linearly scalable, i.e., it increases your throughput as you increase the number of nodes in the cluster. Therefore it maintains a quick response time.

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

- **Easy data distribution:** Cassandra provides the flexibility to distribute data where you need by replicating data across multiple datacenters.

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

- **Fast writes:** 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.

## History of Cassandra

- Cassandra was developed at Facebook for inbox search.
- It was open-sourced by Facebook in July 2008.
- Cassandra was accepted into Apache Incubator in March 2009.
- It was made an Apache top-level project since February 2010.

# 2. ARCHITECTURE

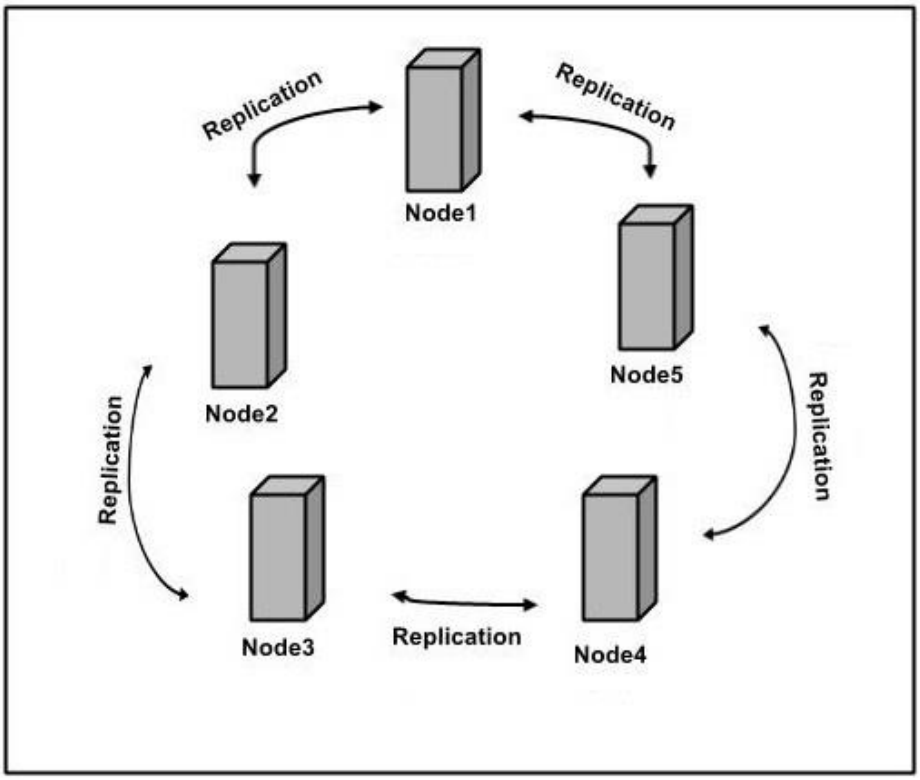
The design goal of Cassandra is to handle big data workloads across multiple nodes without any single point of failure. Cassandra has peer-to-peer distributed system across its nodes, and data is distributed among all the nodes in a cluster.

- All the nodes in a cluster play the same role. Each node is independent and at the same time interconnected to other nodes.
- Each node in a cluster can accept read and write requests, regardless of where the data is actually located in the cluster.
- When a node goes down, read/write requests can be served from other nodes in the network.

## Data Replication in Cassandra

In Cassandra, one or more of the nodes in a cluster act as replicas for a given piece of data. If it is detected that some of the nodes responded with an out-of-date value, Cassandra will return the most recent value to the client. After returning the most recent value, Cassandra performs a **read repair** in the background to update the stale values.

The following figure shows a schematic view of how Cassandra uses data replication among the nodes in a cluster to ensure no single point of failure.




**Note**: Cassandra uses the **Gossip Protocol** in the background to allow the nodes to communicate with each other and detect any faulty nodes in the cluster.

## Components of Cassandra

The key components of Cassandra are as follows:

- **Node:** It is the place where data is stored.
- **Data center:** It is a collection of related nodes.
- **Cluster:** A cluster is a component that contains one or more data centers.
- **Commit log:** The commit log is a crash-recovery mechanism in Cassandra. Every write operation is written to the commit log.
- **Mem-table:** A mem-table is a memory-resident data structure. After commit log, the data will be written to the mem-table. Sometimes, for a single-column family, there will be multiple mem-tables.
- **SSTable:** It is a disk file to which the data is flushed from the mem-table when its contents reach a threshold value.

- **Bloom filter:** These are nothing but quick, nondeterministic, algorithms for testing whether an element is a member of a set. It is a special kind of cache. Bloom filters are accessed after every query.

# Cassandra Query Language

Users can access Cassandra through its nodes using Cassandra Query Language (CQL). CQL treats the database (**Keyspace**) as a container of tables. Programmers use **cqlsh:** a prompt to work with CQL or separate application language drivers.

Clients approach any of the nodes for their read-write operations. That node (coordinator) plays a proxy between the client and the nodes holding the data.

## Write Operations

Every write activity of nodes is captured by the **commit logs** written in the nodes. Later the data will be captured and stored in the **mem-table**. Whenever the mem-table is full, data will be written into the **SStable** data file. All writes are automatically partitioned and replicated throughout the cluster. Cassandra periodically consolidates the SSTables, discarding unnecessary data.

## Read Operations

During read operations, Cassandra gets values from the mem-table and checks the bloom filter to find the appropriate SSTable that holds the required data.

# 3. DATA MODEL

The data model of Cassandra is significantly different from what we normally see in an RDBMS. This chapter provides an overview of how Cassandra stores its data.

## Cluster

Cassandra database is distributed over several machines that operate together. The outermost container is known as the Cluster. For failure handling, every node contains a replica, and in case of a failure, the replica takes charge. Cassandra arranges the nodes in a cluster, in a ring format, and assigns data to them.

## Keyspace

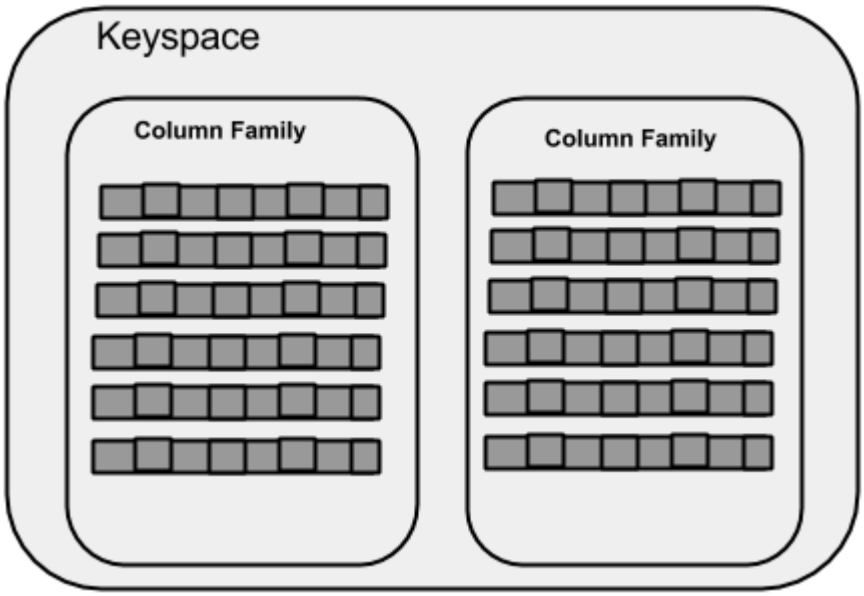
Keyspace is the outermost container for data in Cassandra. The basic attributes of a Keyspace in Cassandra are:

- **Replication factor**: It is the number of machines in the cluster that will receive copies of the same data.
- **Replica placement strategy**: It is nothing but the strategy to place replicas in the ring. We have strategies such as **simple strategy** (rack-aware strategy), **old network topology strategy** (rack-aware strategy), and **network topology strategy** (datacenter-shared strategy).
- **Column families**: Keyspace is a container for a list of one or more column families. A column family, in turn, is a container of a collection of rows. Each row contains ordered columns. Column families represent the structure of your data. Each keyspace has at least one and often many column families.

The syntax of creating a Keyspace is as follows:

```
CREATE KEYSPACE Keyspace name
WITH replication = {'class': 'SimpleStrategy', 'replication_factor' : 3};
```

The following illustration shows a schematic view of a Keyspace.




## Column Family

A column family is a container for an ordered collection of rows. Each row, in turn, is an ordered collection of columns. The following table lists the points that differentiate a column family from a table of relational databases.

| Relational Table | Cassandra Column Family |
|---|---|
| A schema in a relational model is fixed. Once we define certain columns for a table, while inserting data, in every row all the columns must be filled at least with a null value. | In Cassandra, although the column families are defined, the columns are not. You can freely add any column to any column family at any time. |
| Relational tables define only columns and the user fills in the table with values. | In Cassandra, a table contains columns, or can be defined as a super column family. |

A Cassandra column family has the following attributes:

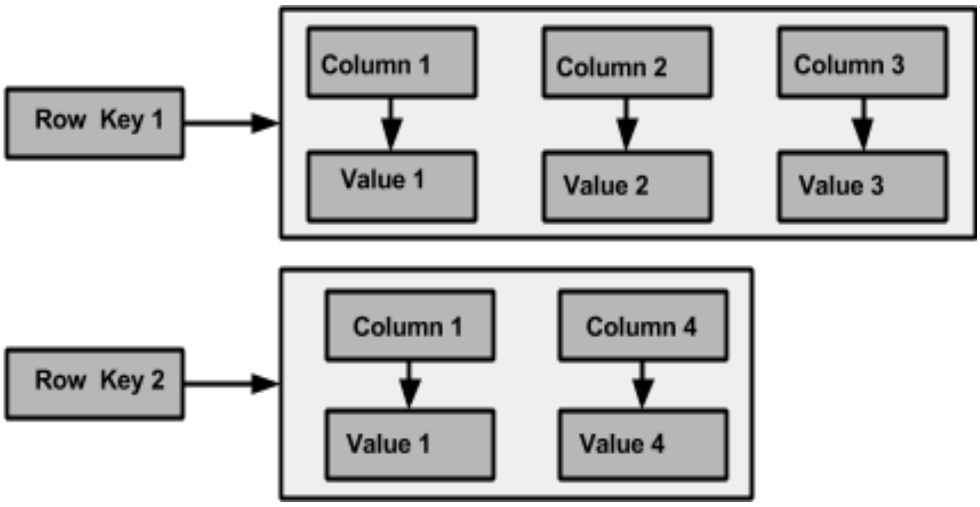
- **keys_cached** It represents the number of locations to keep cached per SSTable.

- **rows_cached** It represents the number of rows whose entire contents will be cached in memory.

- **preload_row_cache** It specifies whether you want to pre-populate the row cache.

**Note:** Unlike relational tables where a column family's schema is not fixed, Cassandra does not force individual rows to have all the columns.

The following figure shows an example of a Cassandra column family.




## Column

A column is the basic data structure of Cassandra with three values, namely key or column name, value, and a time stamp. Given below is the structure of a column.

| Column | | |
|---|---|---|
| name : byte[] | value : byte[] | clock : clock[] |

End of ebook preview

If you liked what you saw…

Buy it from our store @ **https://store.tutorialspoint.com**