Unit-2

Introduction to Cassandra

Dr. Selva kumar S
Assistant Professor
Dept. of CSE, BMSCE

Learning Objectives and Learning Outcomes

Learning Objectives	Learning Outcomes
Introduction to Cassandra	
1. To study the features of Cassandra.	a) To comprehend the reasons behind the popularity of NoSQL database.
2. To learn how to perform CRUD operations.	b) To be able to perform CRUD operations.
3. To learn about collections in Cassandra.	c) To distinguish between collections types such as SET, LIST and MAP.
4. To import from and export to CSV format.	d) To be able to successfully import from CSV.
	e) To be able to successfully export to CSV.

Agenda

- Apache Cassandra – An Introduction
- Features of Cassandra
  - Peer-to-Peer Network
  - Writes in Cassandra
  - Hinted Handoffs
  - Tunable Consistency: Read Consistency and Write Consistency
- CQL Data Types
- CQLSH
- CRUD : Insert, Update, Delete and Select
- Collections : Set, List and Map
- Time To Live (TTL)
- Import and Export

Apache Cassandra – An Introduction

Apache Cassandra - An Introduction

- Apache Cassandra was born at Facebook. After Facebook open sourced the code in 2008, Cassandra became an Apache Incubator project in 2009 and subsequently became a top-level Apache project in 2010.

- It is a column-oriented database designed to support peer-to-peer symmetric nodes instead of the master−slave architecture.

- It is built on Amazon’s dynamo and Google’s BigTable.

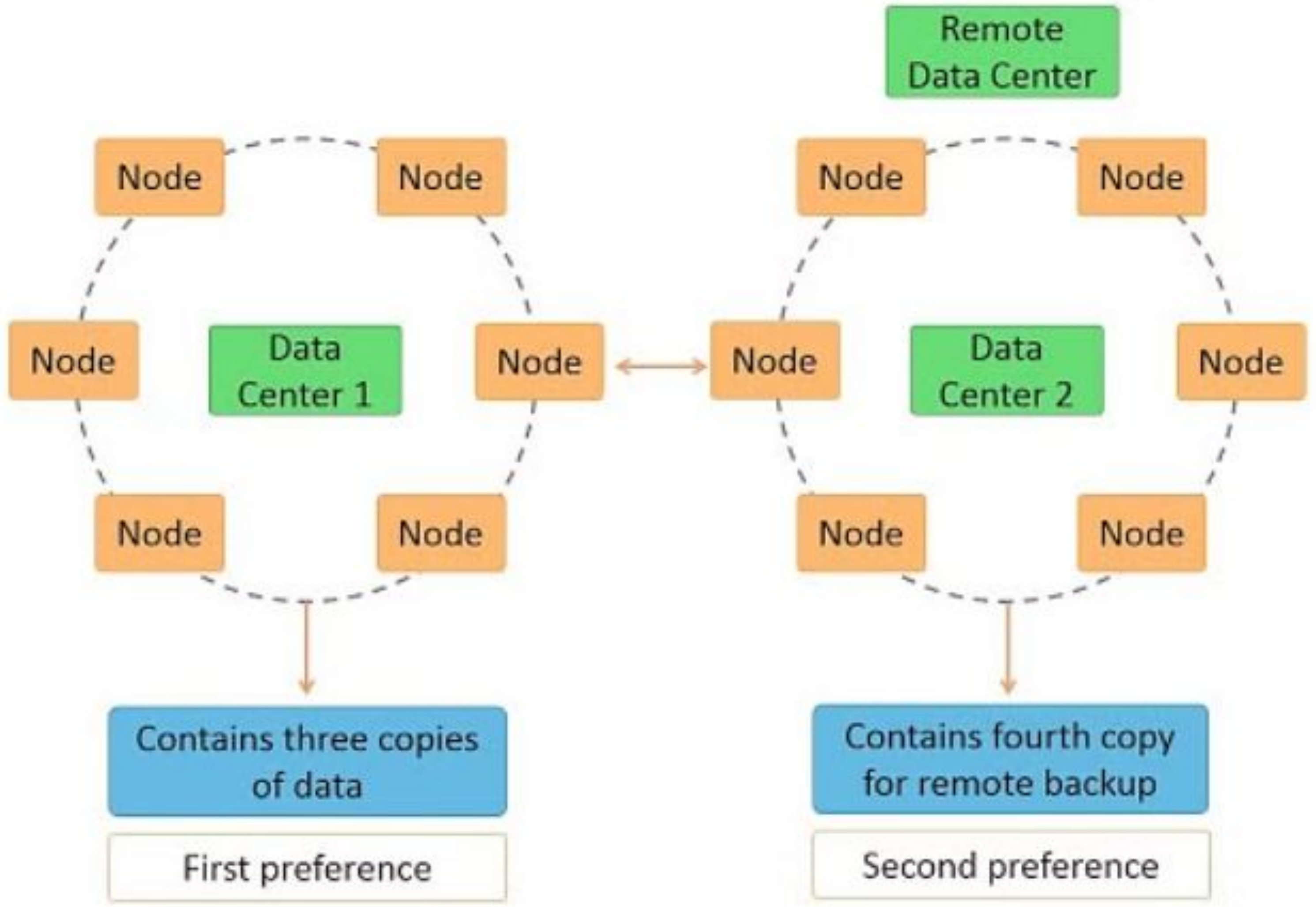
Features of Cassandra

Features of Cassandra

- Open Source
- Distributed
- Decentralized (Server Symmetry)
- No single point of failure
- Column-oriented
- Peer to Peer
- Elastic Scalability

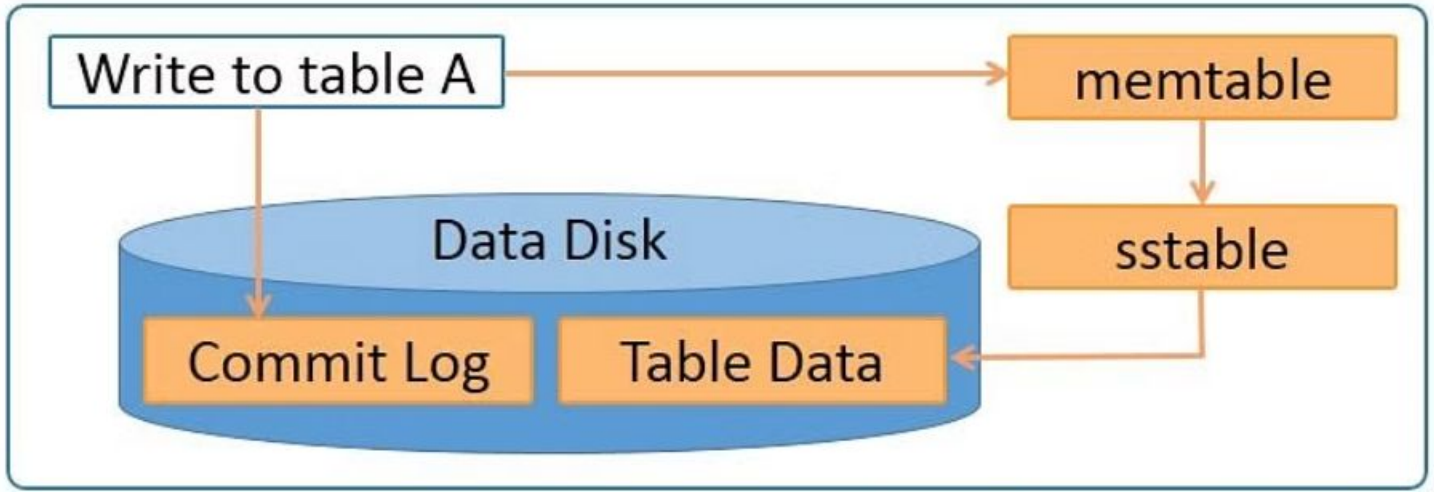
Peer to Peer Network

Sample Cassandra Cluster



Writes in Cassandra

Writes in Cassandra

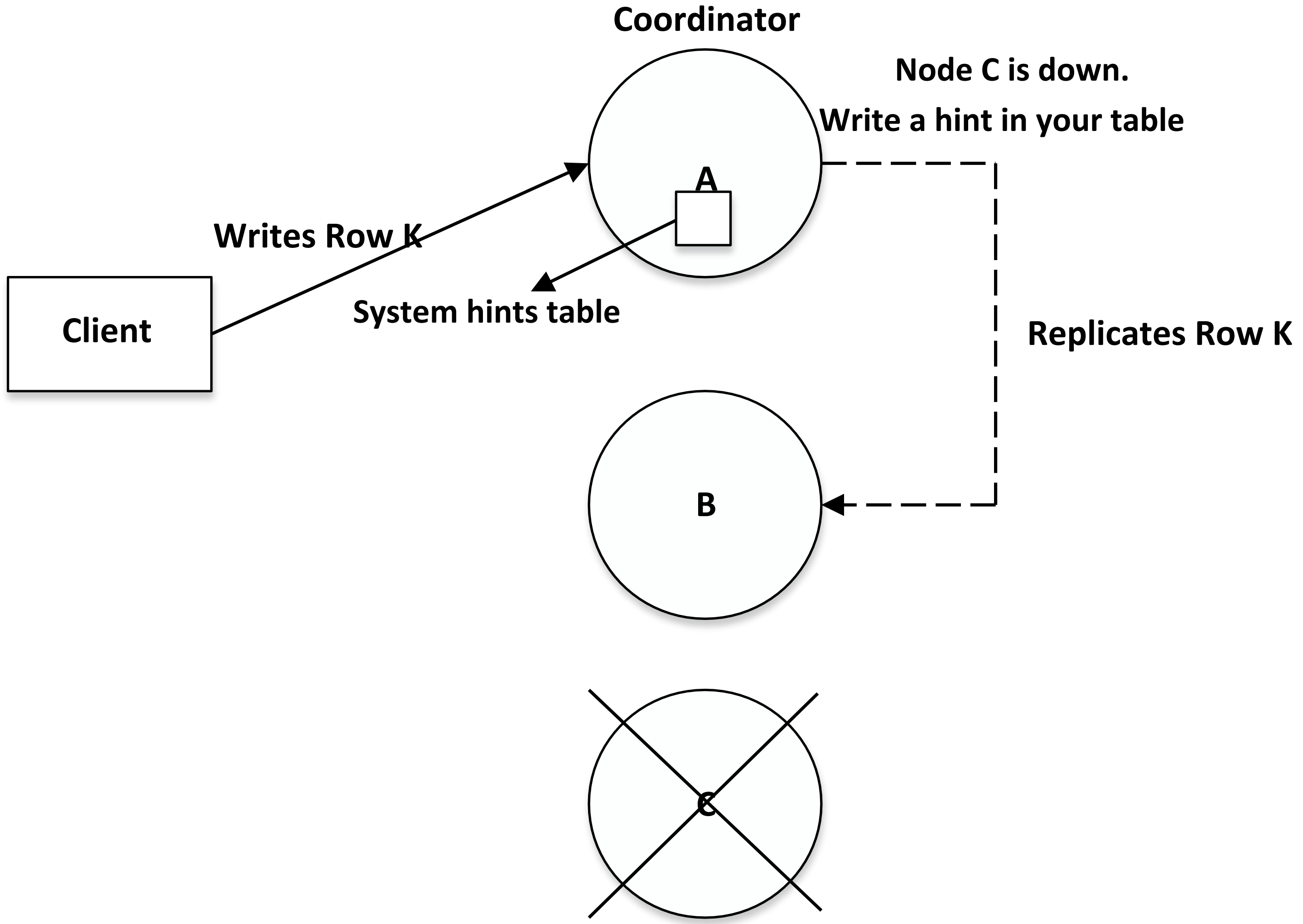


Writes in Cassandra

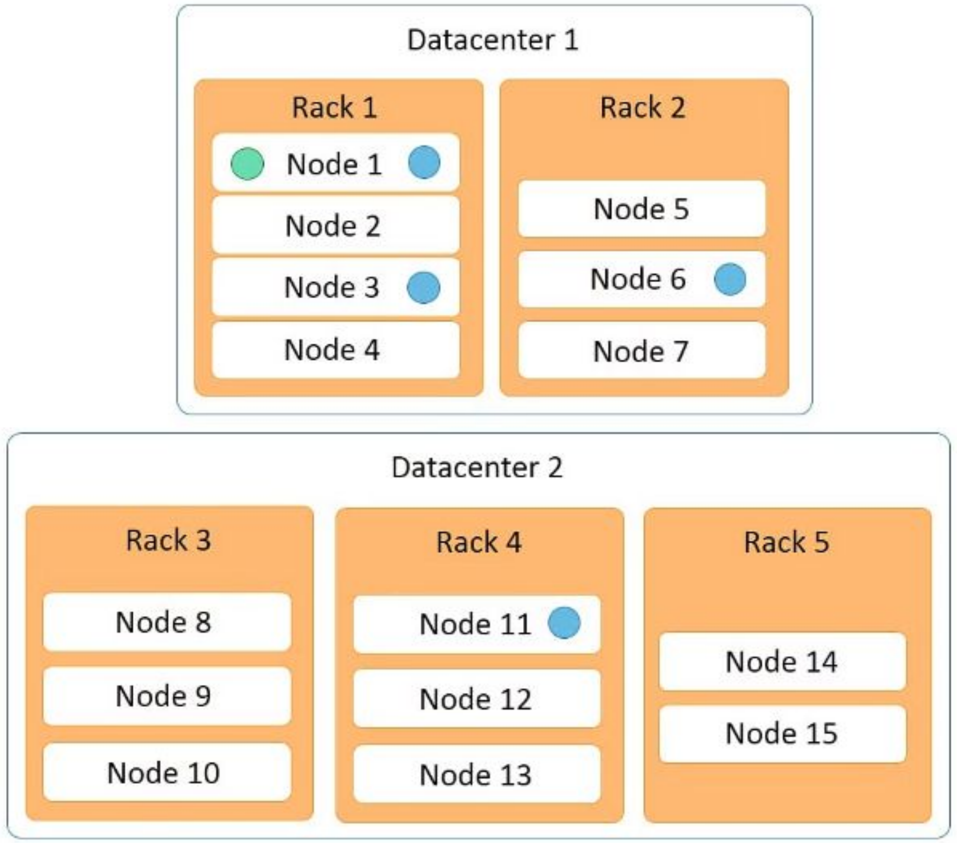
- A client that initiates a write request.
- It is first written to the commit log. A write is taken as successful only if it is written to the commit log.
- The next step is to push the write to a memory resident data structure called Memtable. A threshold value is defined in the Memtable.
- When the number of objects stored in the Memtable reaches a threshold, the contents of Memtable are flushed to the disk in a file called SSTable (Stored string Table). Flushing is a non-blocking operation.
- It is possible to have multiple Memtables for a single column family. One out of them is current and the rest are waiting to be flushed.

Hinted Handoffs

Hinted Handoffs



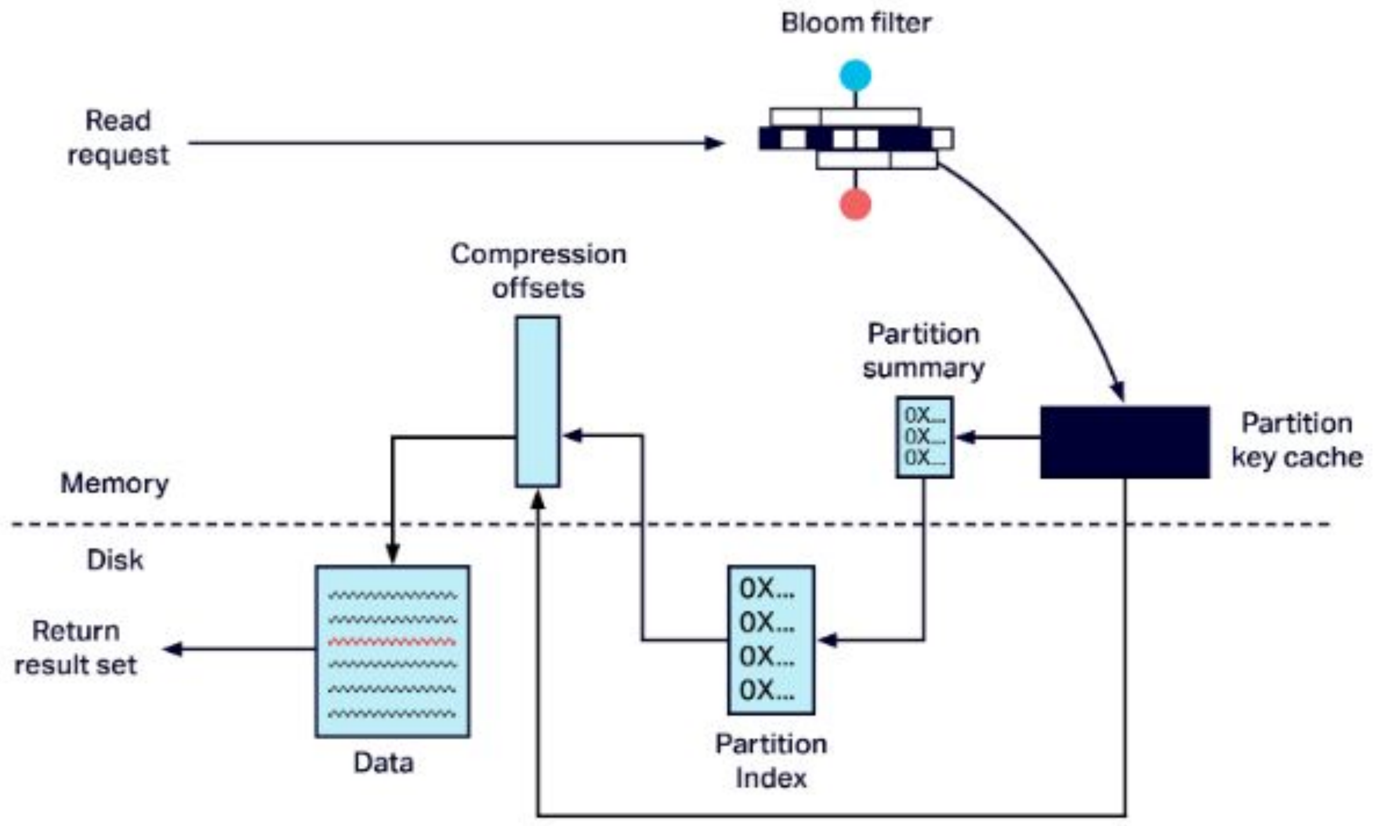
Cassandra read process



Cassandra read process

- Data on the same node is given first preference and is considered data local.
- Data on the same rack is given second preference and is considered rack local.
- Data on the same data center is given third preference and is considered data center local.
- Data in a different data center is given the least preference.
- Data in the memtable and sstable is checked first so that the data can be retrieved faster if it is already in memory.

Read Operations on a Node



Cassandra read process

- Cassandra checks the row cache for data presence. If present, the data is returned, and the request ends.

- The flow of request includes checking bloom filters. If the bloom filter indicates data presented in an SSTable, Cassandra continues to look for the required partition in the SSTable.

- The key cache is checked for the partition key presence. The cache hit provides an offset for the partition in SSTable. This offset is then used to retrieve the partition, and the request completes.

- Cassandra continues to seek the partition in the partition summary and partition index. These structures also provide the partition offset in an SSTable which is then used to retrieve the partition and return. The caches are updated if present with the latest data read.

Recap..

Which Topology is commonly used in Cassandra Installation?

Why should you start with 3 nodes?

What does horizontal scaling mean?

How is data written in Cassandra DB?

How is data read?

CAP Theorem?

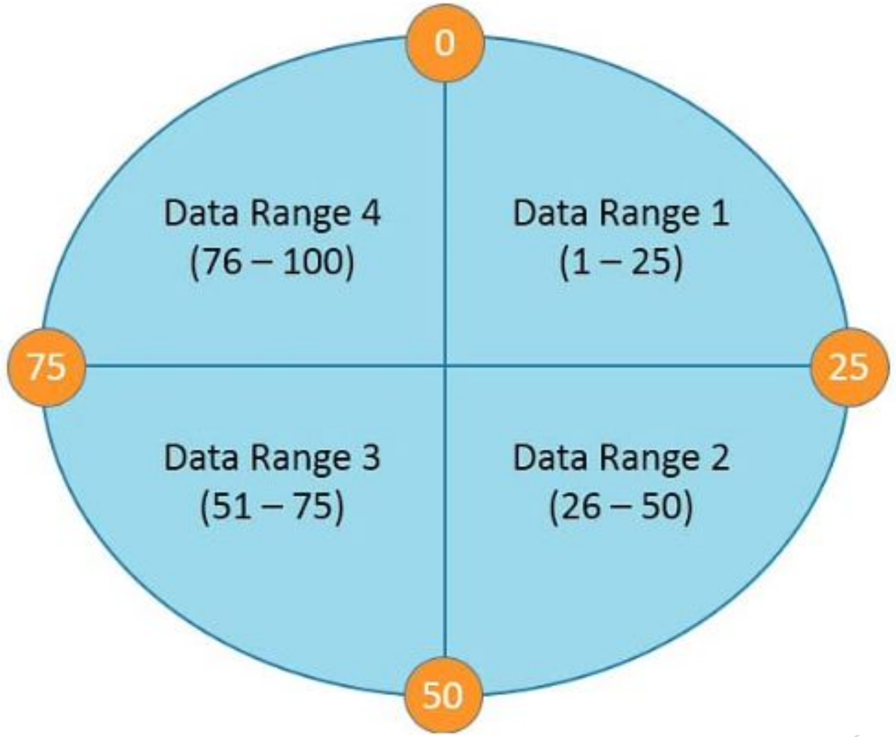
Data partitions

- Cassandra performs transparent distribution of data by horizontally partitioning the data in the following manner:

- A hash value is calculated based on the primary key of the data.
- The hash value of the key is mapped to a node in the cluster
- The first copy of the data is stored on that node.
- The distribution is transparent as you can both calculate the hash value and determine where a particular row will be stored.

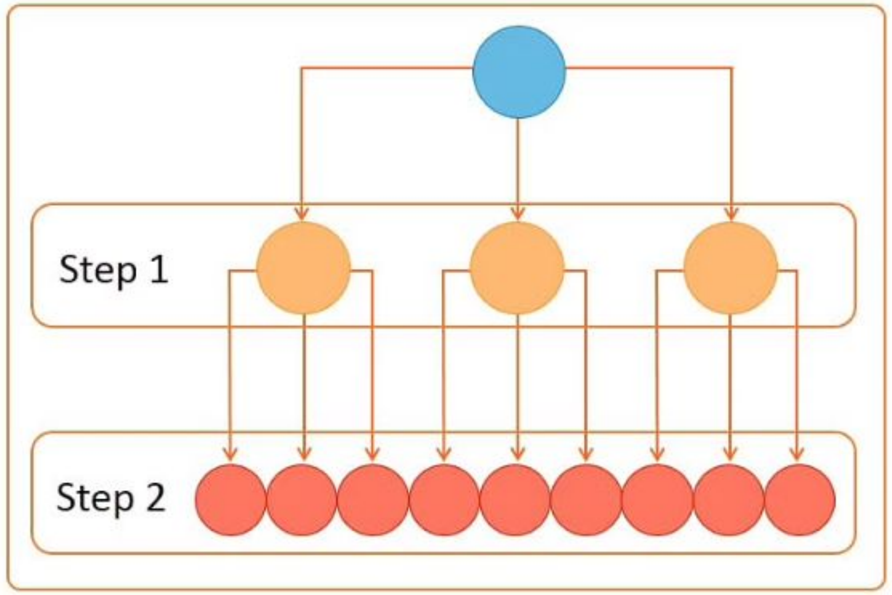
Data partitions

- The following diagram depicts a four node cluster with token values of 0, 25, 50 and 75.



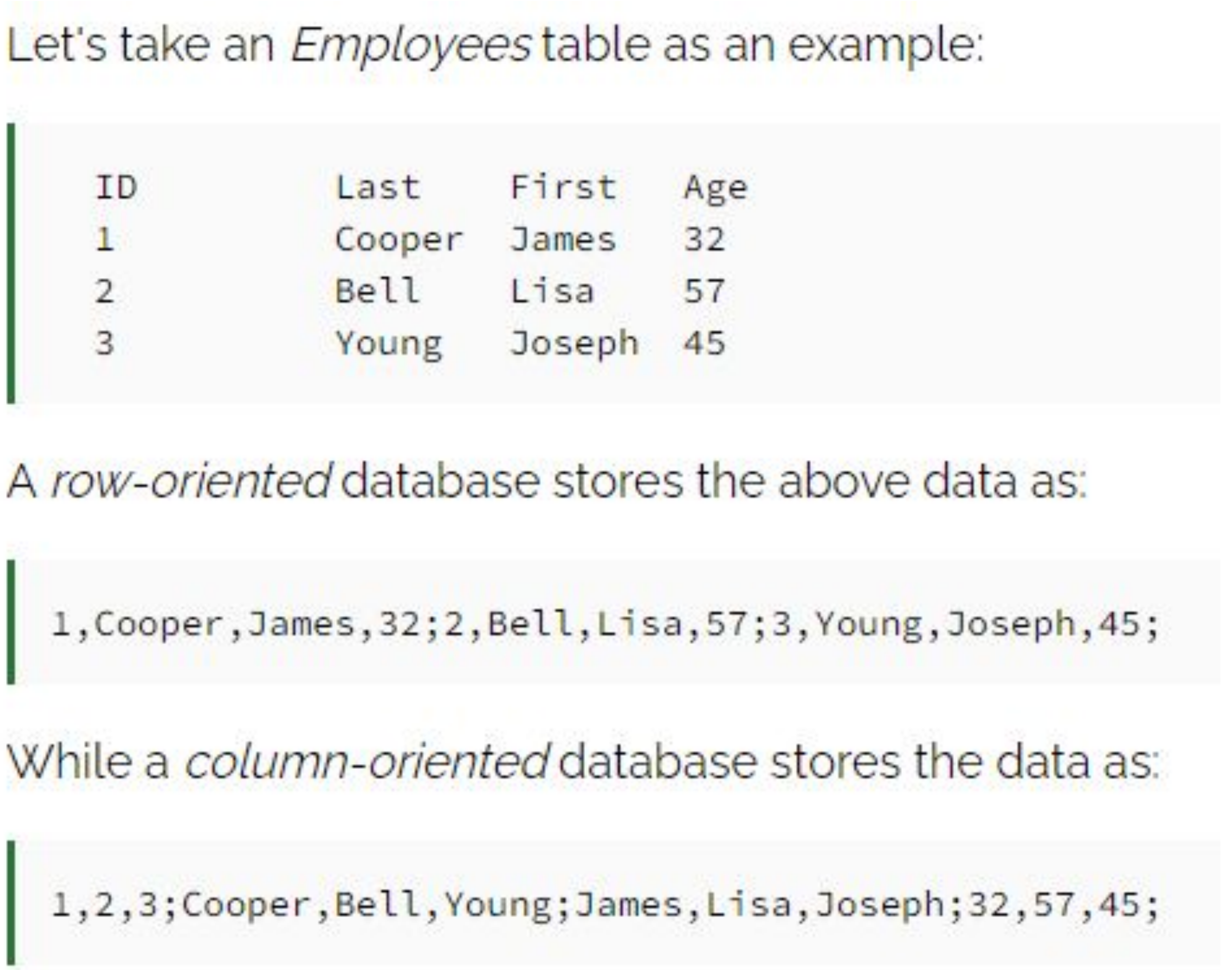
Gossip protocol

- Cassandra uses a gossip protocol to communicate with nodes in a cluster.
- It is an inter-node communication mechanism similar to the heartbeat protocol in Hadoop.
- The gossip process runs periodically on each node and exchanges state information with three other nodes in the cluster.
- Eventually, information is propagated to all cluster nodes.



Row-oriented and column-oriented data store

Cassandra does not store its data like either a *row-oriented* or a *column-oriented* database.



Let's take an *Employees* table as an example:

```
ID          Last    First   Age
1           Cooper  James   32
2           Bell    Lisa    57
3           Young   Joseph  45
```

A *row-oriented* database stores the above data as:

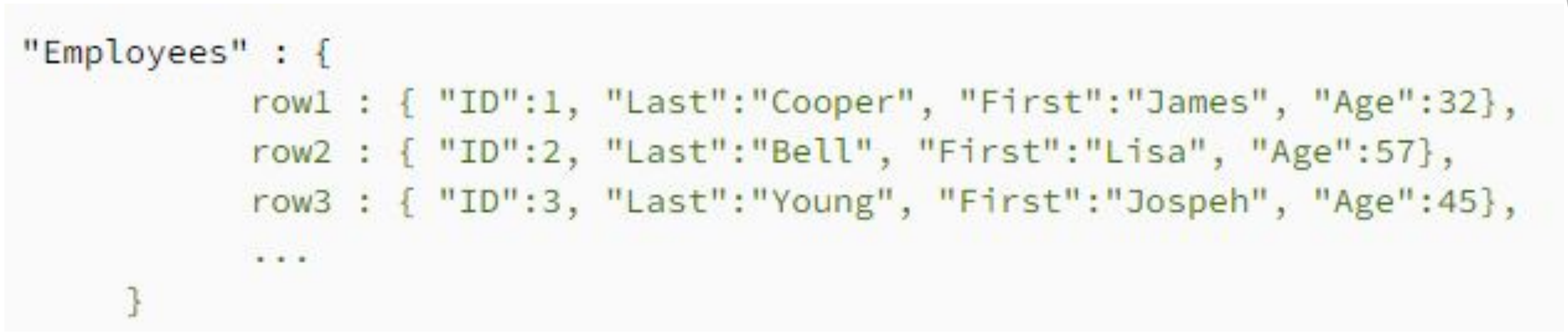
```
1,Cooper,James,32;2,Bell,Lisa,57;3,Young,Joseph,45;
```

While a *column-oriented* database stores the data as:

```
1,2,3;Cooper,Bell,Young;James,Lisa,Joseph;32,57,45;
```

Partitioned row store

Cassandra uses a *partitioned row store*, which means rows contain columns. A *column-family* database stores data with keys mapped to values and the values grouped into multiple column families.

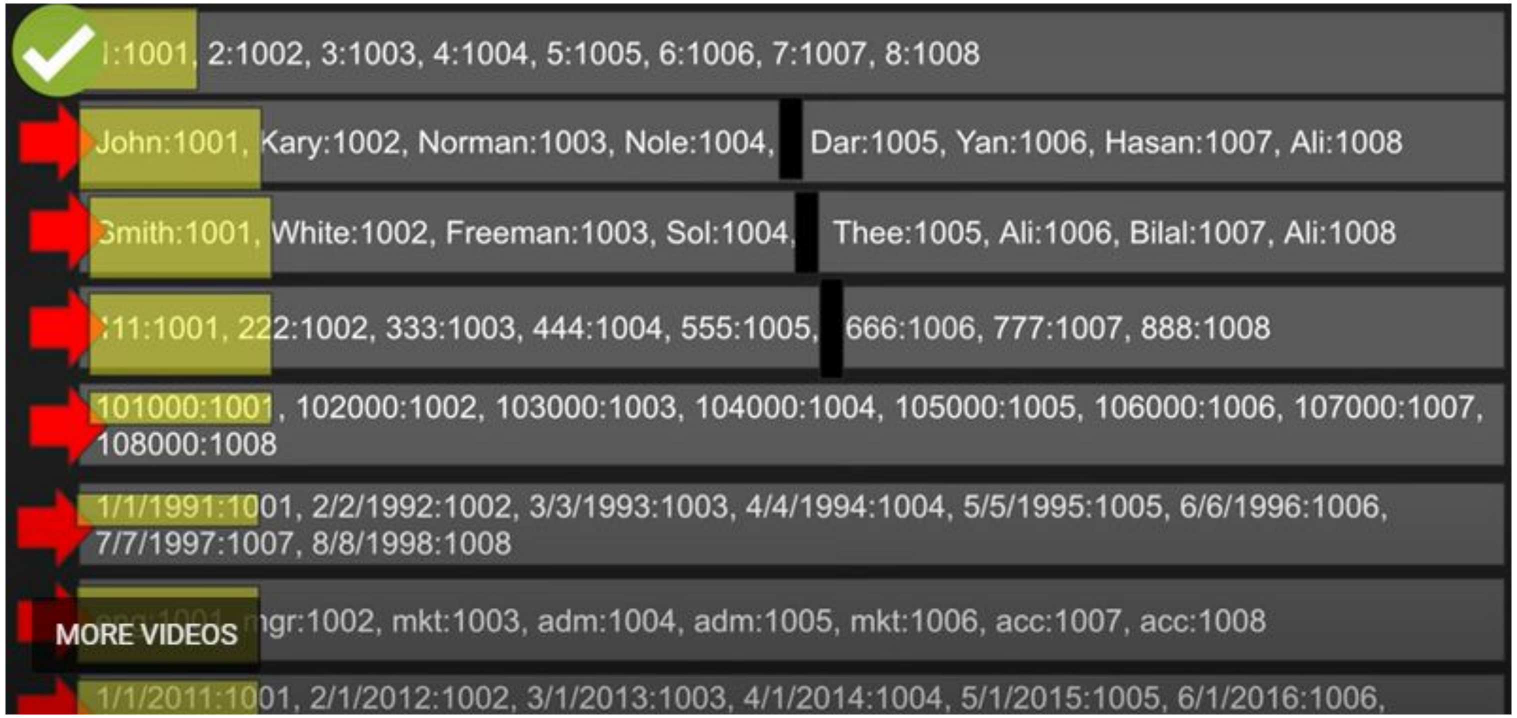


```
"Employees" : {
            row1 : { "ID":1, "Last":"Cooper", "First":"James", "Age":32},
            row2 : { "ID":2, "Last":"Bell", "First":"Lisa", "Age":57},
            row3 : { "ID":3, "Last":"Young", "First":"Jospeh", "Age":45},
            ...
      }
```

A *partitioned row store* has rows that contain columns, yet the number of columns in each row does not have to be the same

Example

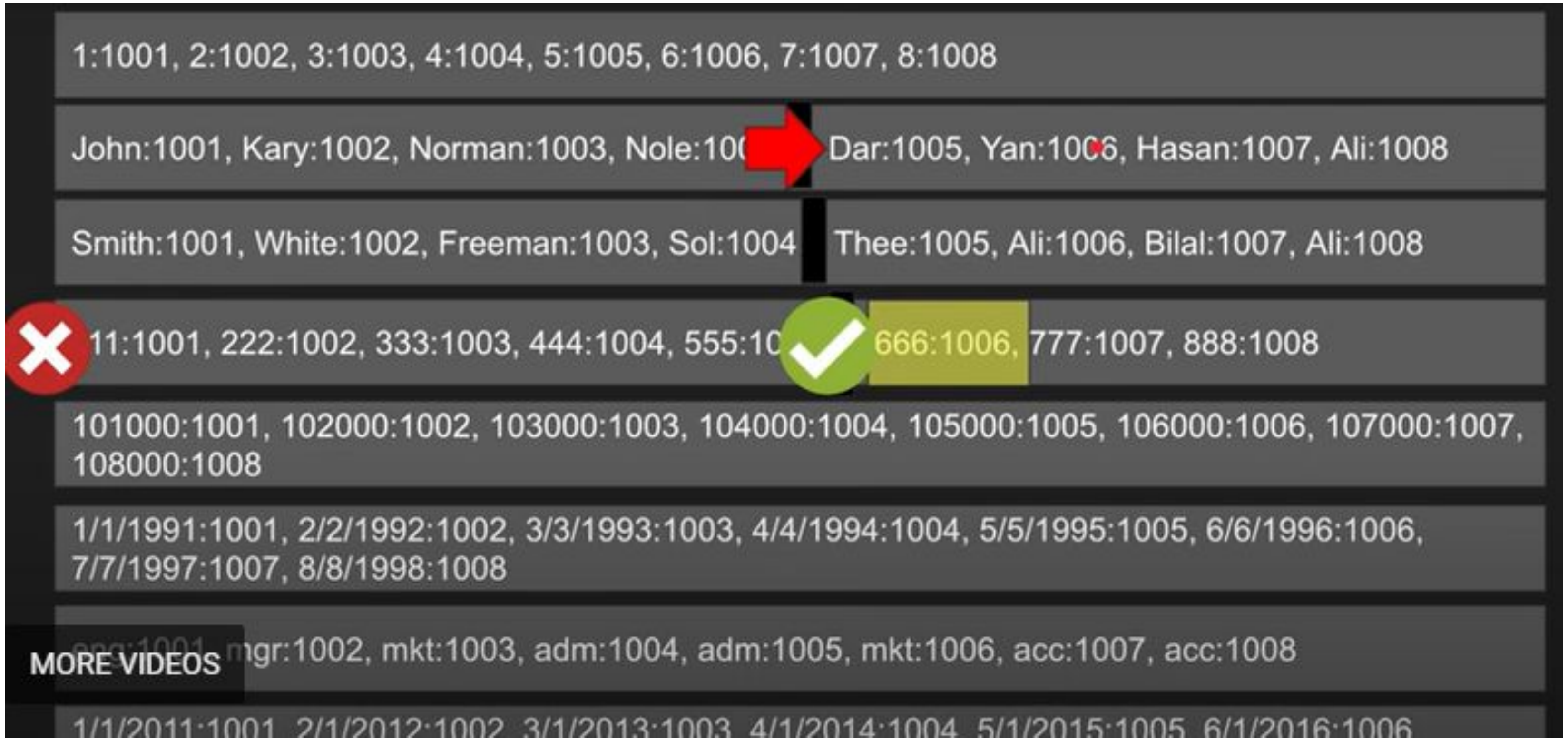
Select * from emp where id=1



```
1:1001, 2:1002, 3:1003, 4:1004, 5:1005, 6:1006, 7:1007, 8:1008
John:1001, Kary:1002, Norman:1003, Nole:1004, Dar:1005, Yan:1006, Hasan:1007, Ali:1008
Smith:1001, White:1002, Freeman:1003, Sol:1004, Thee:1005, Ali:1006, Bilal:1007, Ali:1008
111:1001, 222:1002, 333:1003, 444:1004, 555:1005, 666:1006, 777:1007, 888:1008
101000:1001, 102000:1002, 103000:1003, 104000:1004, 105000:1005, 106000:1006, 107000:1007,
108000:1008
1/1/1991:1001, 2/2/1992:1002, 3/3/1993:1003, 4/4/1994:1004, 5/5/1995:1005, 6/6/1996:1006,
7/7/1997:1007, 8/8/1998:1008
mgr:1002, mkt:1003, adm:1004, adm:1005, mkt:1006, acc:1007, acc:1008
1/1/2011:1001, 2/1/2012:1002, 3/1/2013:1003, 4/1/2014:1004, 5/1/2015:1005, 6/1/2016:1006,
```

example

Select first_name from emp where ssn=666



1:1001, 2:1002, 3:1003, 4:1004, 5:1005, 6:1006, 7:1007, 8:1008

John:1001, Kary:1002, Norman:1003, Nole:10 Dar:1005, Yan:1006, Hasan:1007, Ali:1008

Smith:1001, White:1002, Freeman:1003, Sol:1004 Thee:1005, Ali:1006, Bilal:1007, Ali:1008

11:1001, 222:1002, 333:1003, 444:1004, 555:10 666:1006, 777:1007, 888:1008

101000:1001, 102000:1002, 103000:1003, 104000:1004, 105000:1005, 106000:1006, 107000:1007, 108000:1008

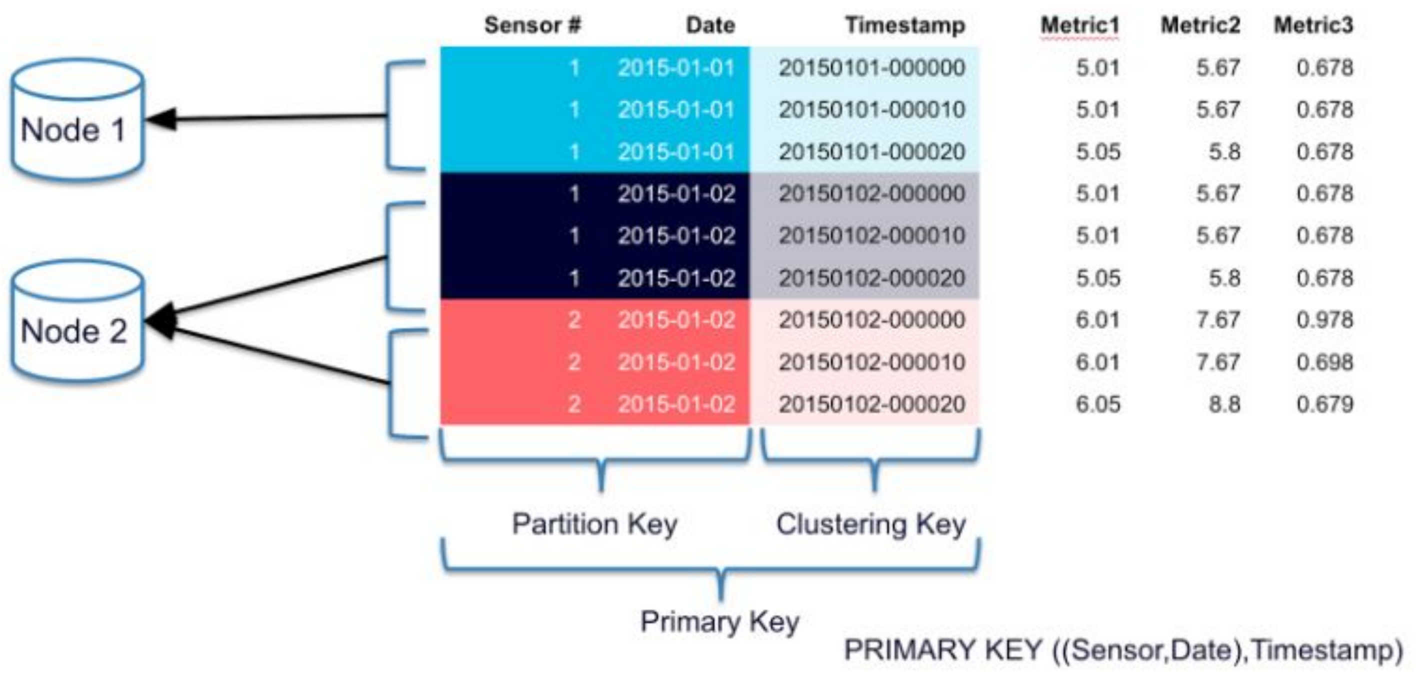
1/1/1991:1001, 2/2/1992:1002, 3/3/1993:1003, 4/4/1994:1004, 5/5/1995:1005, 6/6/1996:1006, 7/7/1997:1007, 8/8/1998:1008

MORE VIDEOS mgr:1002, mkt:1003, adm:1004, adm:1005, mkt:1006, acc:1007, acc:1008

1/1/2011:1001, 2/1/2012:1002, 3/1/2013:1003, 4/1/2014:1004, 5/1/2015:1005, 6/1/2016:1006

Data Partitioning

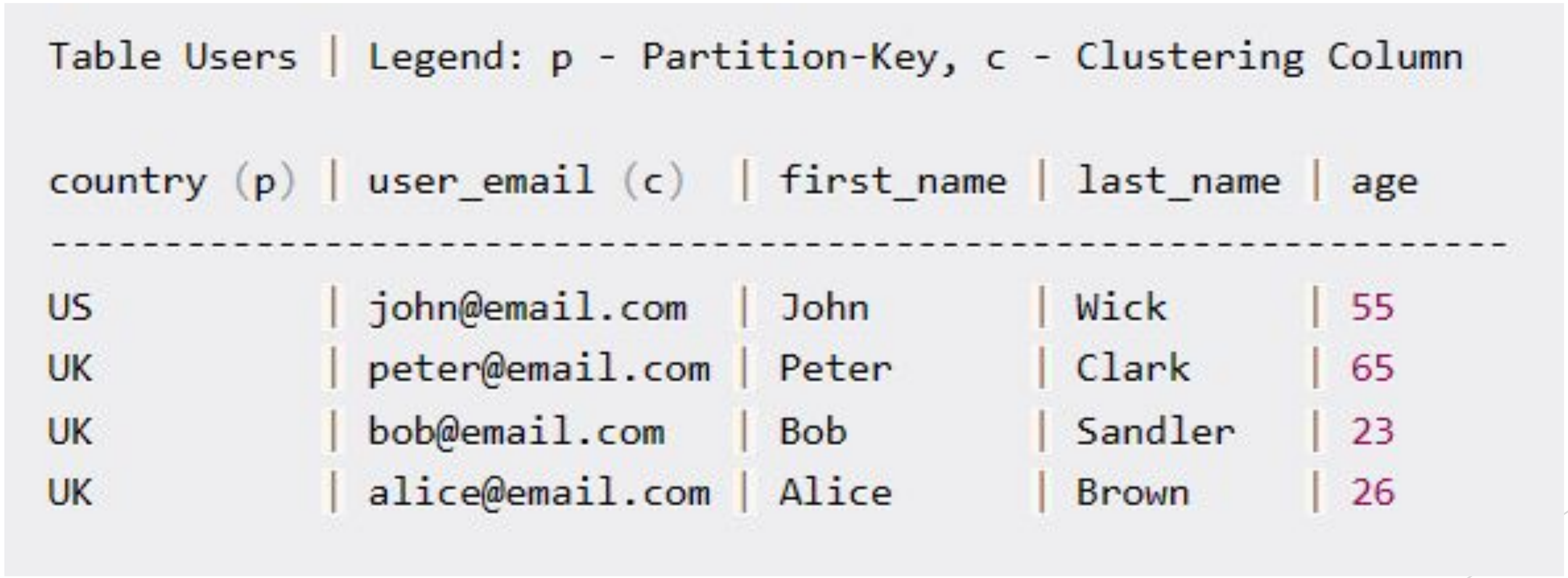
- Partitioning is a method of splitting and storing a single logical dataset in multiple databases.
- By distributing the data among multiple machines, a cluster of database systems can store larger datasets and handle additional requests.



Sensor #	Date	Timestamp	Metric1	Metric2	Metric3
1	2015-01-01	20150101-000000	5.01	5.67	0.678
1	2015-01-01	20150101-000010	5.01	5.67	0.678
1	2015-01-01	20150101-000020	5.05	5.8	0.678
1	2015-01-02	20150102-000000	5.01	5.67	0.678
1	2015-01-02	20150102-000010	5.01	5.67	0.678
1	2015-01-02	20150102-000020	5.05	5.8	0.678
2	2015-01-02	20150102-000000	6.01	7.67	0.978
2	2015-01-02	20150102-000010	6.01	7.67	0.698
2	2015-01-02	20150102-000020	6.05	8.8	0.679

Data Partitioning

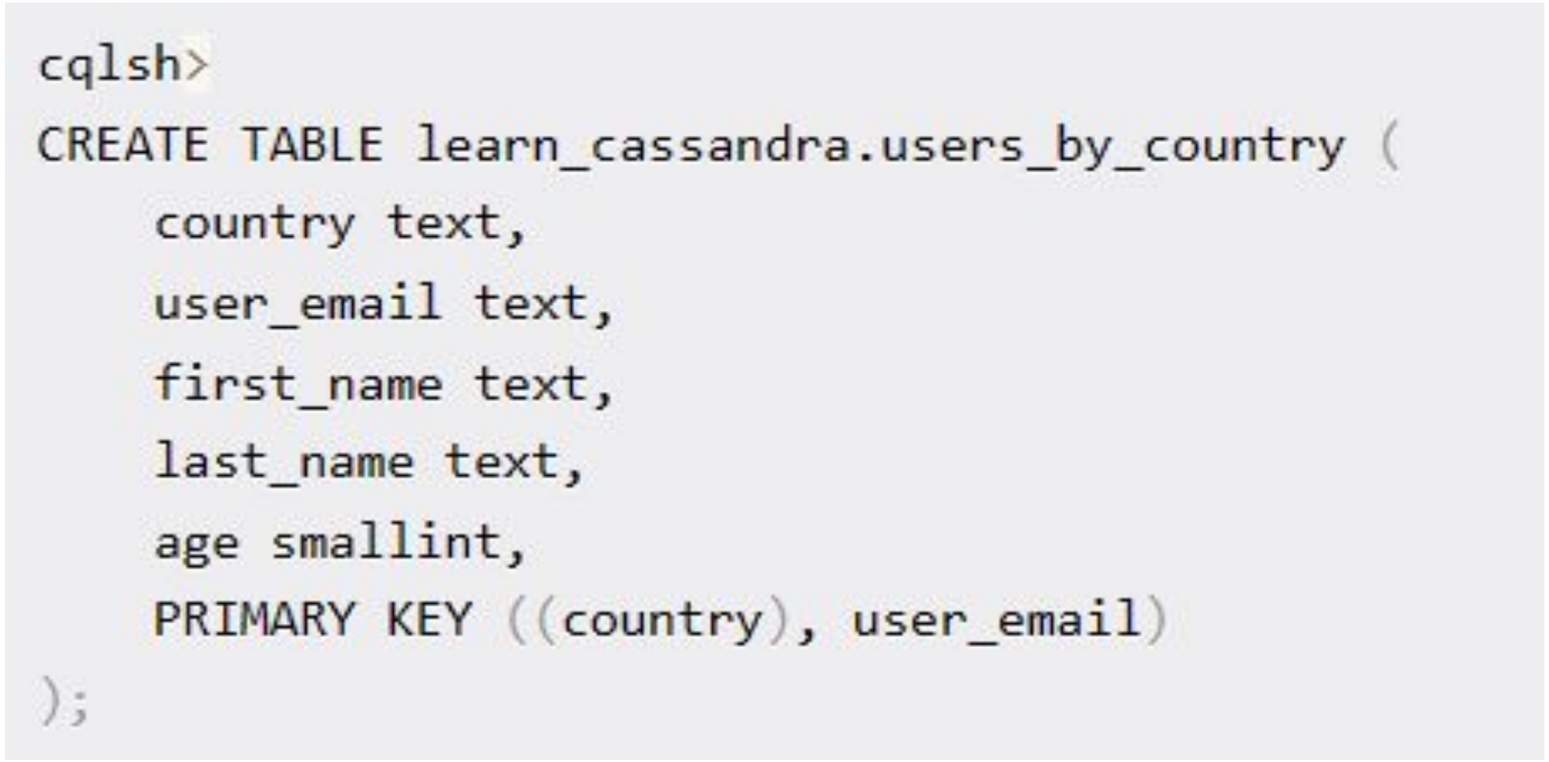
- In Cassandra, the primary key consists of 2 parts:
  - a mandatory partition key and
  - an optional set of clustering columns.



```
Table Users | Legend: p - Partition-Key, c - Clustering Column

country (p) | user_email (c)  | first_name | last_name | age
-------------------------------------------------------------
US          | john@email.com  | John       | Wick      | 55
UK          | peter@email.com | Peter      | Clark     | 65
UK          | bob@email.com   | Bob        | Sandler   | 23
UK          | alice@email.com | Alice      | Brown     | 26
```

Data Partitioning

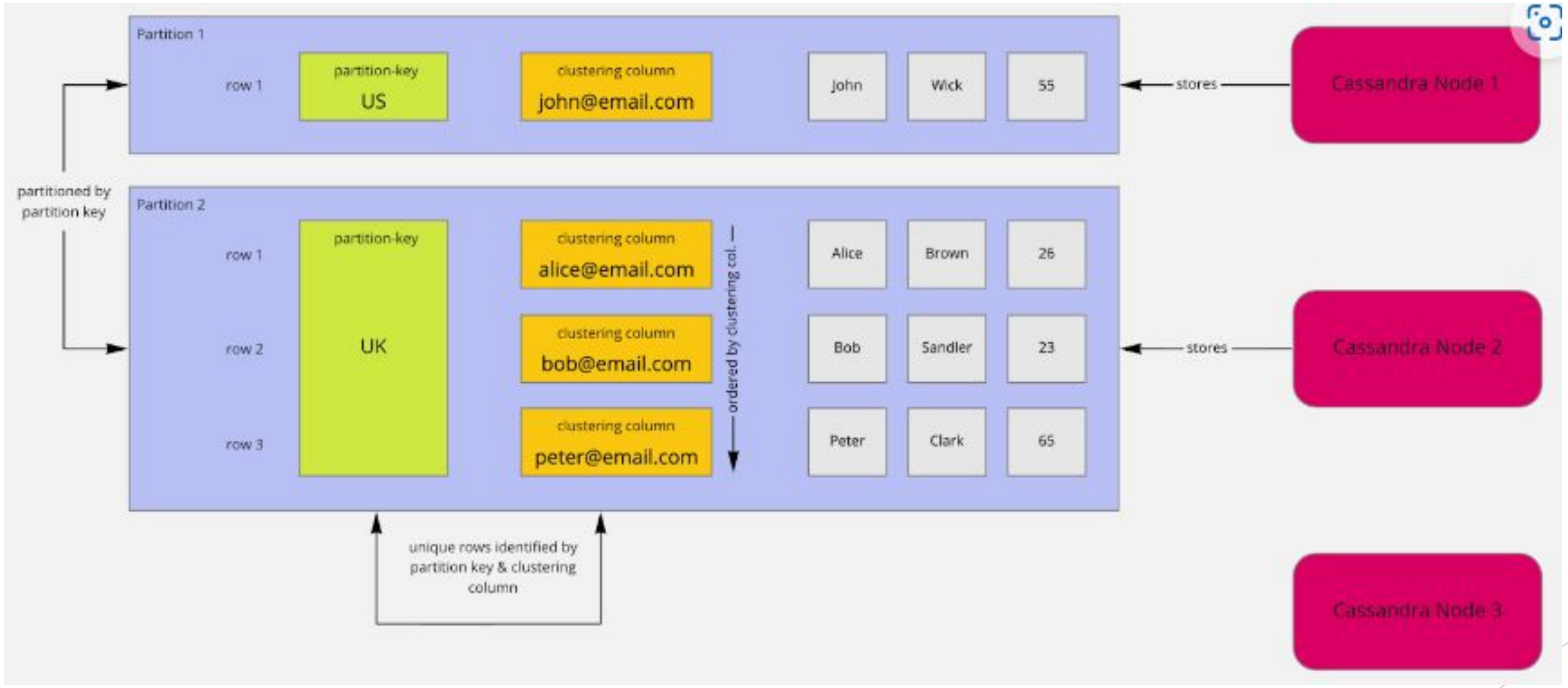


```
cqlsh>
CREATE TABLE learn_cassandra.users_by_country (
    country text,
    user_email text,
    first_name text,
    last_name text,
    age smallint,
    PRIMARY KEY ((country), user_email)
);
```

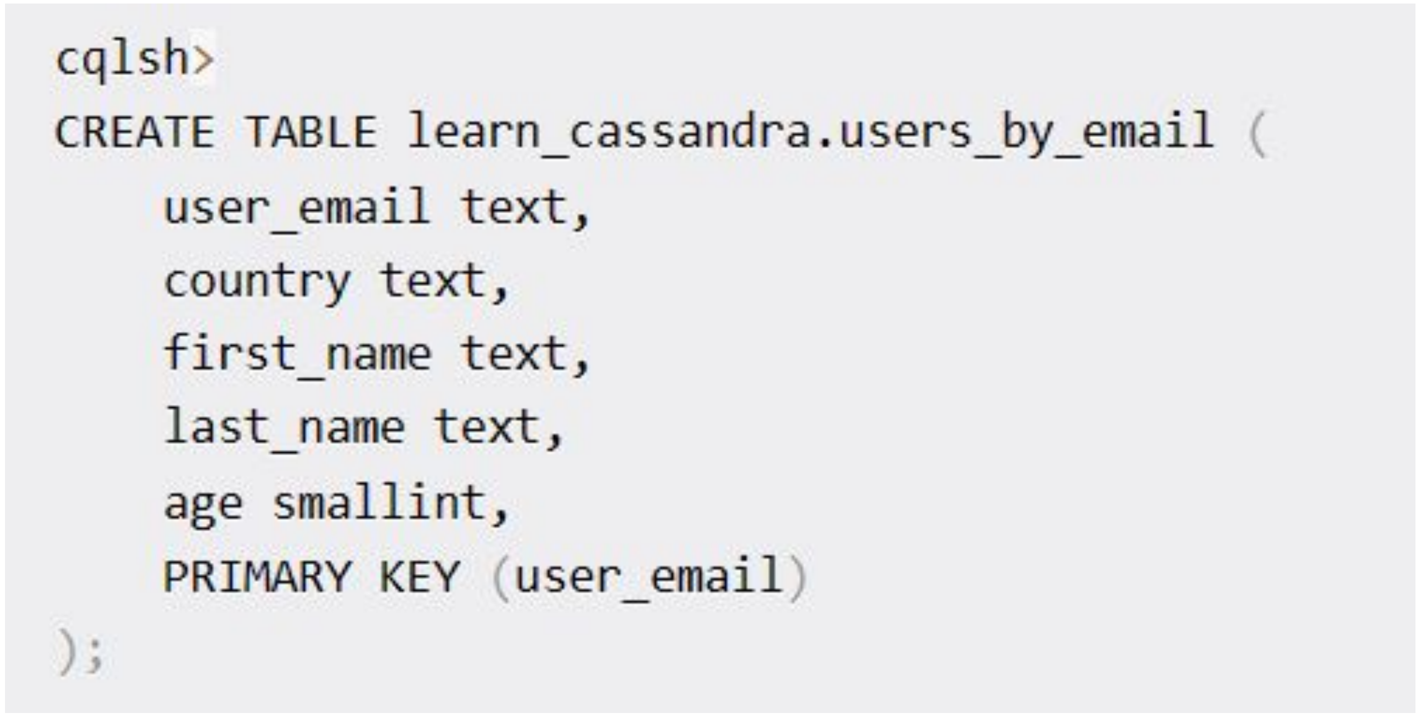
The first group of the primary key defines the partition key. All other elements of the primary key are clustering columns:

Data Partitioning

In the context of partitioning, the words partition and shard can be used interchangeably.



Data Partitioning

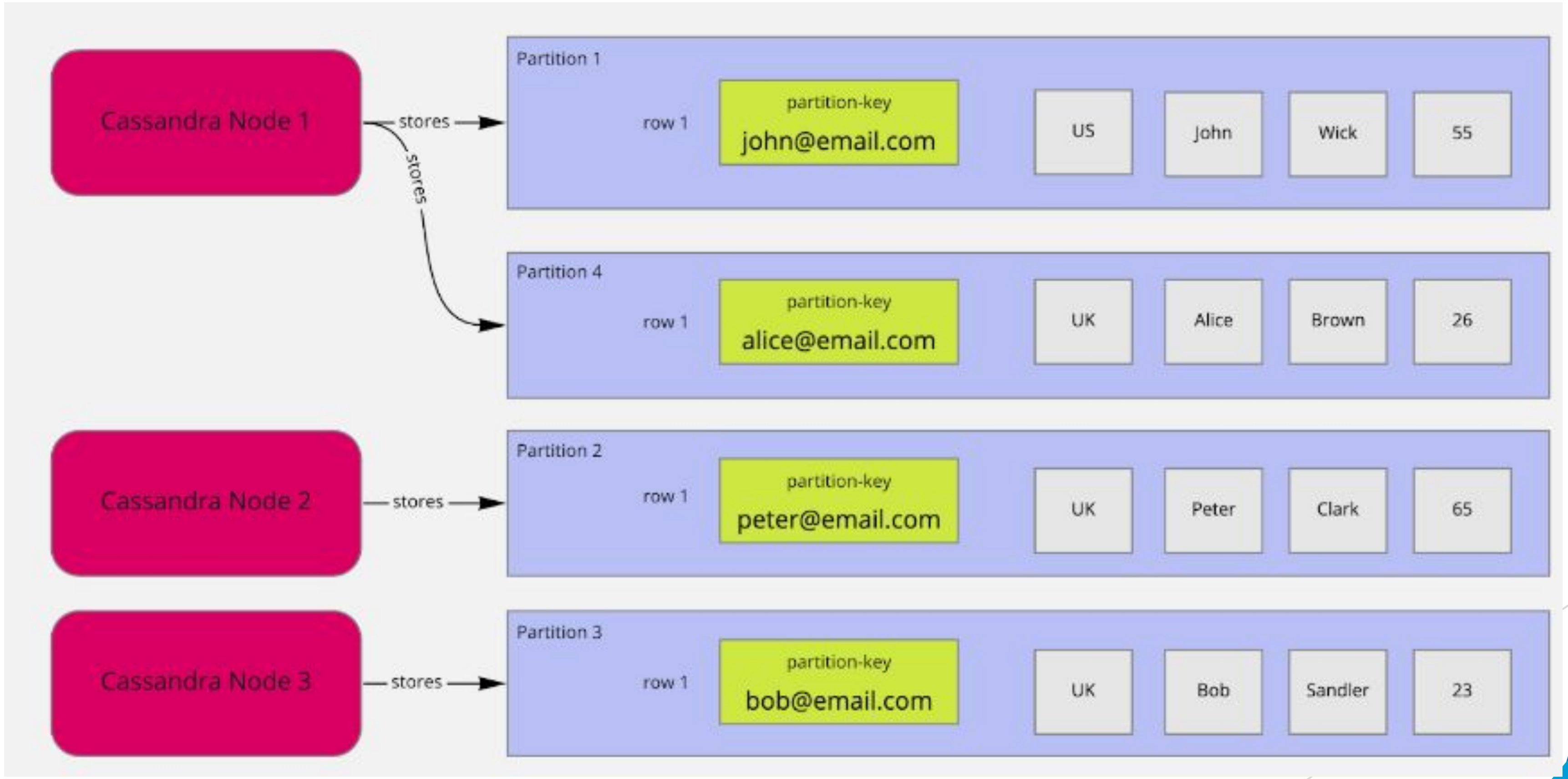


```
cqlsh>
CREATE TABLE learn_cassandra.users_by_email (
    user_email text,
    country text,
    first_name text,
    last_name text,
    age smallint,
    PRIMARY KEY (user_email)
);
```

Data Partitioning

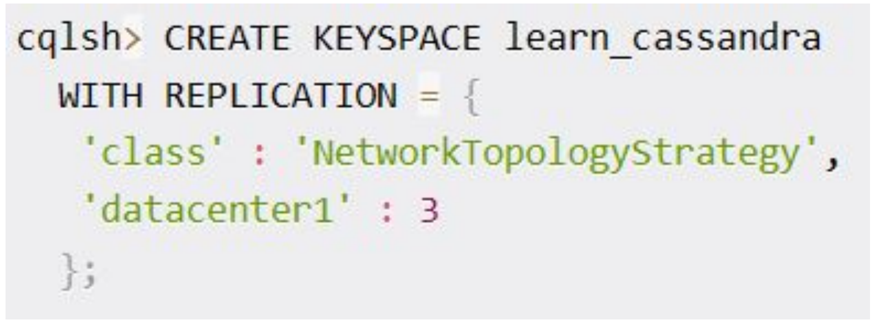
```
cqlsh> SELECT * FROM learn_cassandra.users_by_email WHERE user_email='alice@email.com';

cqlsh> SELECT * FROM learn_cassandra.users_by_email WHERE age=26 ALLOW FILTERING;
```



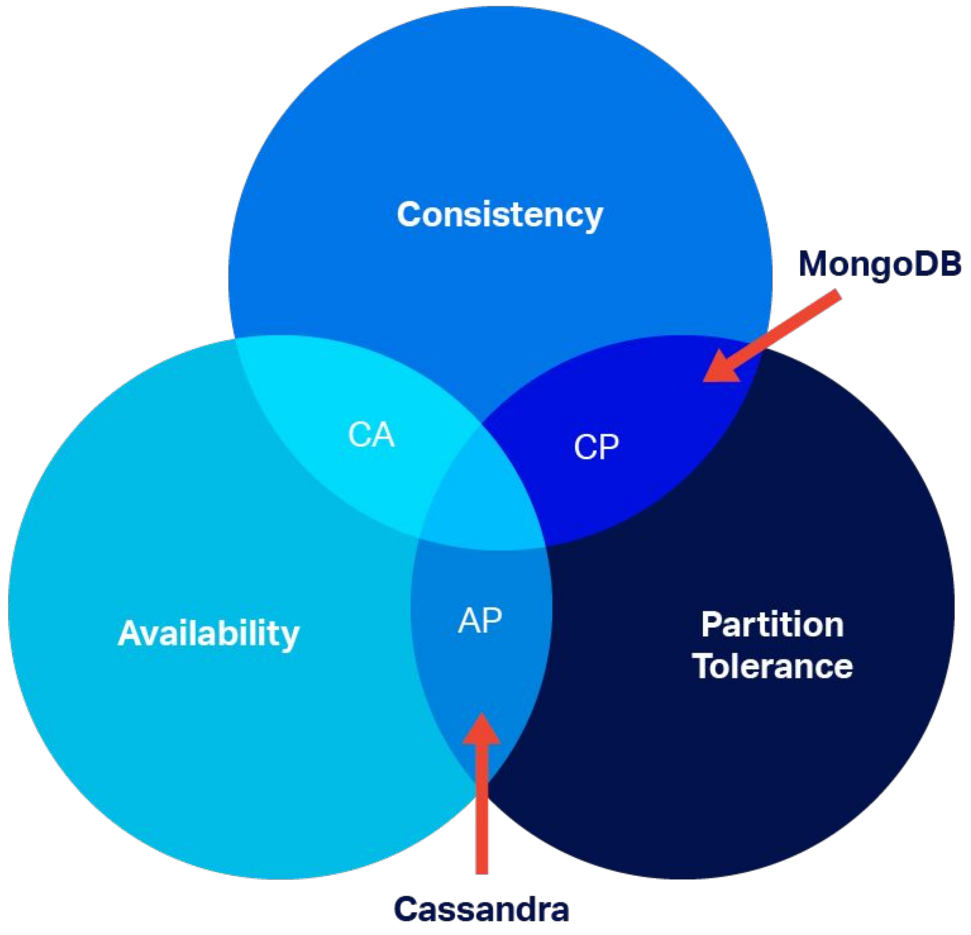
Replication

- Scalability using partitioning alone is limited.
- Consider a lot of write requests arriving for a single partition. All requests would be sent to a single node with technical limitations such as CPU, memory, and bandwidth. Additionally, you want to handle read and write requests if this node is not available.
- By duplicating data to different nodes, so called replicas, you can serve more data simultaneously from other nodes to improve latency and throughput.
- It also enables your cluster to perform reads and writes in case a replica is not available.
- In Cassandra, you need to define a replication factor for every keyspace.

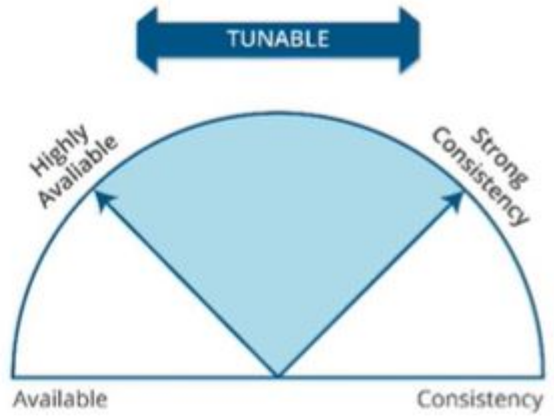


```
cqlsh> CREATE KEYSPACE learn_cassandra
  WITH REPLICATION = {
   'class' : 'NetworkTopologyStrategy',
   'datacenter1' : 3
  };
```

CAP Theorem


Consistency
MongoDB
CA
CP
AP
Availability
Partition Tolerance
Cassandra

Tunable Consistency



Tunable Consistency

- To ensure that Cassandra can provide the proper levels of consistency for its reads and writes, Cassandra extends the concept of eventual consistency by offering tunable consistency.
- One can tune Cassandra's consistency level per-operation, or set it globally for a cluster or datacenter.
- There is a tradeoff between operation latency and consistency:
- Higher consistency incurs higher latency.
- Lower consistency permits lower latency.
- One can control latency by tuning consistency.

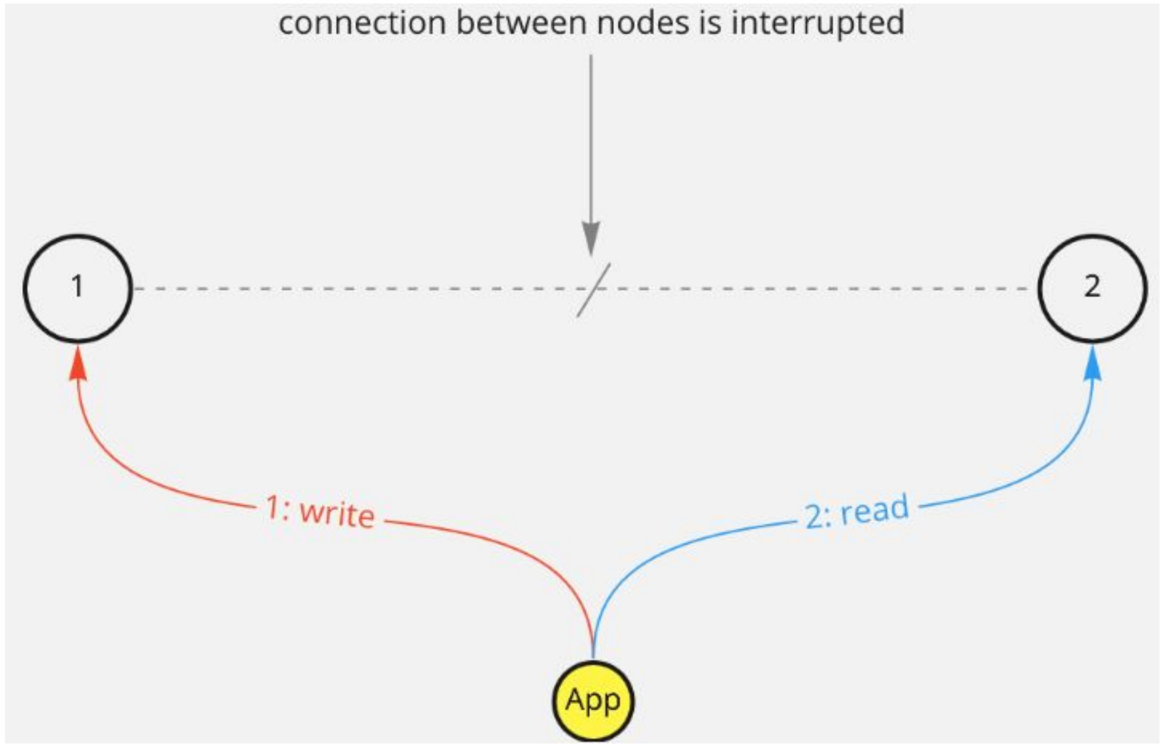
What does Strong Consistency?

- In contrast to eventual consistency, strong consistency means only one state of your data can be observed at any time in any location.

- For example, when consistency is critical, like in a banking domain, you want to be sure that everything is correct. You would rather accept a decrease in availability and increase of latency to ensure correctness.

Example

You want to write a single value to a table. The data is replicated in 2 nodes, and the connection between the nodes is interrupted. First, a write-request is sent to node 1. Then, data is read from node 2.
How do you manage this situation?



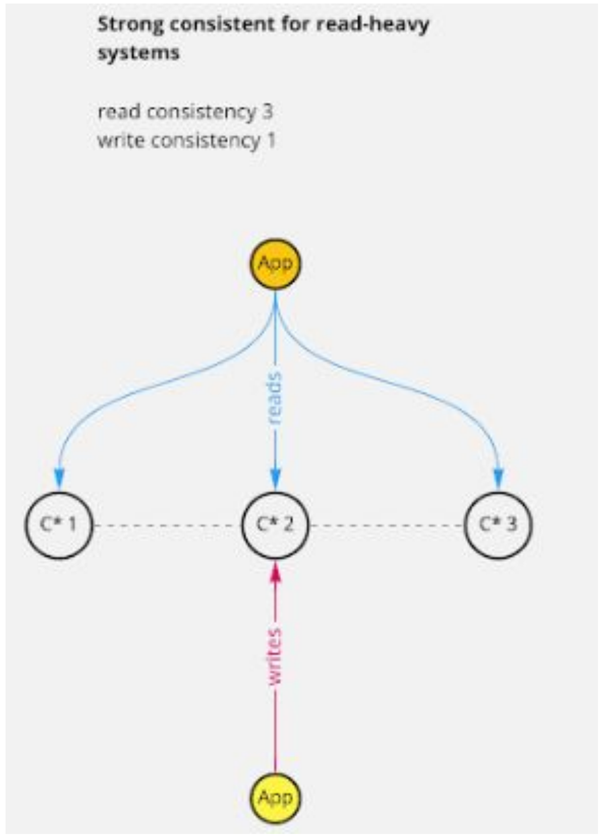
Calculating Consistency

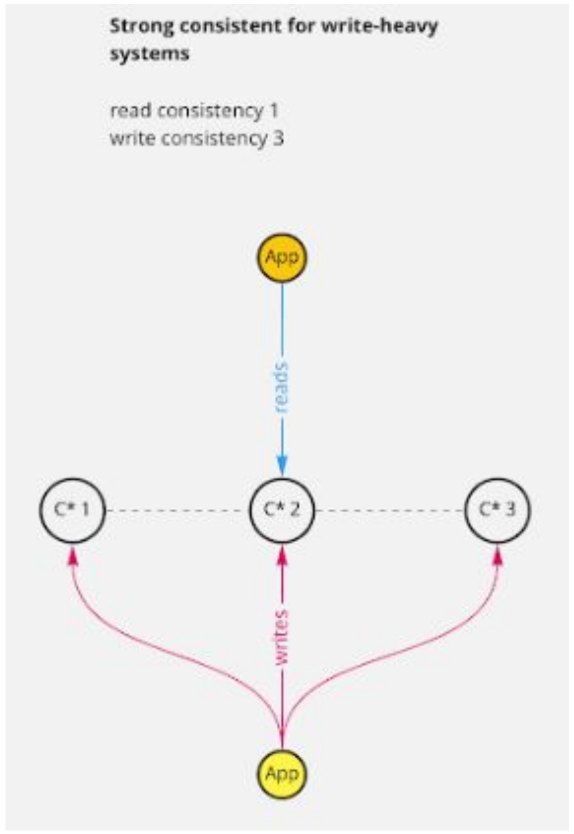
There is a very important formula that if true guarantees strong consistency:

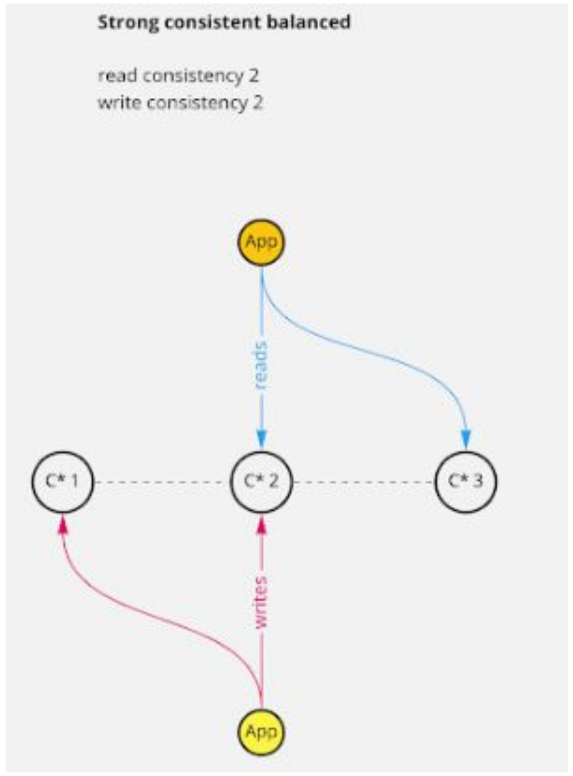
```
[read-consistency-level] + [write-consistency-level] > [replication-factor]
```

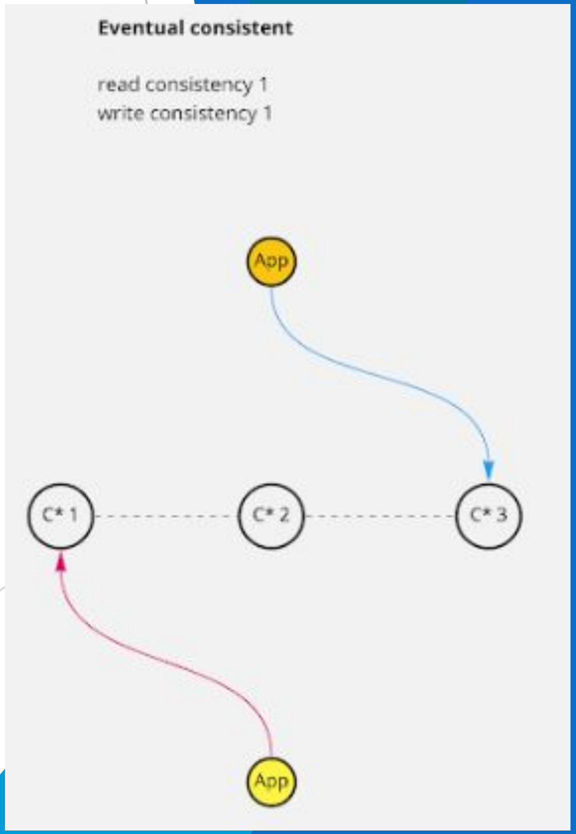
Consistency Level

You can shift read and write consistency levels to your favor if you want to keep strong consistency. Or you even give up strong consistency for better performance, which is also called eventual consistency:









Read Consistency

ONE	Returns a response from the closest node (replica) holding the data.
QUORUM	Returns a result from a quorum of servers with the most recent timestamp for the data.
LOCAL_QUORUM	Returns a result from a quorum of servers with the most recent timestamp for the data in the same data center as the coordinator node.
EACH_QUORUM	Returns a result from a quorum of servers with the most recent timestamp in all data centers.
ALL	This provides the highest level of consistency of all levels and the lowest level of availability of all levels. It responds to a read request from a client after all the replica nodes have responded.

Write Consistency

ALL	This is the highest level of consistency of all levels as it necessitates that a write must be written to the commit log and Memtable on all replica nodes in the cluster.
EACH_QUORUM	A write must be written to the commit log and Memtable on a quorum of replica nodes in all data centers.
QUORUM	A write must be written to the commit log and Memtable on a quorum of replica nodes.
LOCAL_QUORUM	A write must be written to the commit log and Memtable on a quorum of replica nodes in the same data center as the coordinator node. This is to avoid latency of inter-data center communication.
ONE	A write must be written to the commit log and Memtable of at least one replica node.
TWO	A write must be written to the commit log and Memtable of at least two replica nodes.
THREE	A write must be written to the commit log and Memtable of at least three replica nodes.
LOCAL_ONE	A write must be sent to, and successfully acknowledged by, at least one replica node in the local data center.

CQL Data types

CQL Data types

Int	32 bit signed integer
Bigint	64 bit signed long
Double	64-bit IEEE-754 floating point
Float	32-bit IEEE-754 floating point
Boolean	True or false
Blob	Arbitrary bytes, expressed in hexadecimal
Counter	Distributed counter value
Decimal	Variable – precision integer
List	A collection of one or more ordered elements
Map	A JSON style array of elements
Set	A collection of one or more elements
Timestamp	Date plus time
Varchar	UTF 8 encoded string
Varint	Arbitrary-precision integers
Text	UTF 8 encoded string

CQLSH

CRUD - Keyspace

To create a keyspace by the name “Students”

```
CREATE KEYSPACE Students WITH REPLICATION = {
            'class':'SimpleStrategy', 'replication_factor':1
 };
```

```
USE keyspace_name;

ALTER KEYSPACE excelsior WITH replication = {'class': 'SimpleStrategy',
'replication_factor' : 4};

DROP KEYSPACE excelsior;
```

CRUD - Create Table

To create a column family or table by the name “student_info”.

```
CREATE TABLE Student_Info
  ( RollNo int PRIMARY KEY,
  StudName text,
  DateofJoining timestamp,
  LastExamPercent double
  );
```

Create Table

```
CREATE TABLE t (
    a int,
    b int,
    c int,
    d int,
    PRIMARY KEY ((a, b), c, d)
);
INSERT INTO t (a, b, c, d) VALUES (0,0,0,0);
INSERT INTO t (a, b, c, d) VALUES (0,0,1,1);
INSERT INTO t (a, b, c, d) VALUES (0,1,2,2);
INSERT INTO t (a, b, c, d) VALUES (0,1,3,3);
INSERT INTO t (a, b, c, d) VALUES (1,1,4,4);
SELECT * FROM t;


a | b | c | d
---+---+---+---
 0 | 0 | 0 | 0        1
 0 | 0 | 1 | 1
 0 | 1 | 2 | 2        2
 0 | 1 | 3 | 3
 1 | 1 | 4 | 4        3

(5 rows)
```

Create table

```
CREATE TABLE t2 (
    a int,
    b int,
    c int,
    d int,
    PRIMARY KEY (a, b, c)
);
INSERT INTO t2 (a, b, c, d) VALUES (0,0,0,0);
INSERT INTO t2 (a, b, c, d) VALUES (0,0,1,1);
INSERT INTO t2 (a, b, c, d) VALUES (0,1,2,2);
INSERT INTO t2 (a, b, c, d) VALUES (0,1,3,3);
INSERT INTO t2 (a, b, c, d) VALUES (1,1,4,4);
SELECT * FROM t2;
```

output

```
 a | b | c | d
---+---+---+---
 1 | 1 | 4 | 4
 0 | 0 | 0 | 0
 0 | 0 | 1 | 1
 0 | 1 | 2 | 2
 0 | 1 | 3 | 3

(5 rows)
```

SELECT * FROM t2 WHERE a = 0 AND b > 0 and b <= 3;

```
 a | b | c | d
---+---+---+---
 0 | 1 | 2 | 2
 0 | 1 | 3 | 3

(2 rows)
```

Create Table Syntax

```
Syntax:
CREATE TABLE [ IF NOT EXISTS ] tablename '('
 columndefinition  ( ',' columndefinition )*
 [ ',' PRIMARY KEY '(' primary_key ')' ]
 ')' [ WITH table_options ]
columndefinition::= columnname cql_type [ STATIC ] [ PRIMARY KEY]
primary_key::= partition_key [ ',' clusteringcolumns ]
partition_key::= columnname  | '(' columnname ( ',' columnname )* ')'
clusteringcolumns::= columnname ( ',' columnname )*
TABLE_OPTIONS:TABLE_OPTIONS:=: compact storage [ AND table_options ]
 | clustering ORDER BY '(' clustering_order ')'
 [ AND table_options ]  | options
clustering_order::= columnname (ASC | DESC) ( ',' columnname (ASC | DESC) )*
```

Example

```
CREATE TABLE monkey_species (
    species text PRIMARY KEY,
    common_name text,
    population varint,
    average_size int
) WITH comment='Important biological records';

CREATE TABLE timeline (
    userid uuid,
    posted_month int,
    posted_time uuid,
    body text,
    posted_by text,
    PRIMARY KEY (userid, posted_month, posted_time)
) WITH compaction = { 'class' : 'LeveledCompactionStrategy' };

CREATE TABLE loads (
    machine inet,
    cpu int,
    mtime timeuuid,
    load float,
    PRIMARY KEY ((machine, cpu), mtime)
) WITH CLUSTERING ORDER BY (mtime DESC);
```

CRUD - Insert

To insert data into the column family “student_info”.

```
BEGIN BATCH
INSERT INTO student_info
(RollNo,StudName,DateofJoining,LastExamPercent
) VALUES (1,'Michael Storm','2012-03-29', 69.6)
INSERT INTO student_info
(RollNo,StudName,DateofJoining,LastExamPercent
) VALUES (2,'Stephen Fox','2013-02-27', 72.5)
APPLY BATCH;
```

Insert - Example

```
INSERT INTO NerdMovies (movie, director, main_actor, year)
   VALUES ('Serenity', 'Joss Whedon', 'Nathan Fillion', 2005)
   USING TTL 86400;


INSERT INTO NerdMovies JSON '{"movie": "Serenity", "director": "Joss Whedon", "year": 2005}';
```

CRUD - Select

To view the data from the table “student_info”.

```
SELECT *
          FROM student_info;
```

Select example

```
CREATE TABLE users (
    username text PRIMARY KEY,
    firstname text,
    lastname text,
    birth_year int,
    country text
);

CREATE INDEX ON users(birth_year);

// All users are returned
SELECT * FROM users;

// All users with a particular birth year are returned
SELECT * FROM users WHERE birth_year = 1981;

SELECT * FROM users WHERE birth_year = 1981 AND country = 'FR';

SELECT * FROM users WHERE birth_year = 1981 AND country = 'FR' ALLOW FILTERING;
```

CRUD - Create Index

To create an index on the “studname” column of the “student_info”
column family use the following statement

CREATE INDEX ON student_info(studname);

CRUD - Update

To update the value held in the “StudName” column of the “student_info” column family to “David Sheen” for the record where the RollNo column has value = 2.

Note: An update updates one or more column values for a given row to the Cassandra table. It does not return anything.

UPDATE student_info SET StudName = 'David Sheen' WHERE RollNo = 2;

CRUD - Delete

To delete the column “LastExamPercent” from the “student_info” table for the record where the RollNo = 2.

Note: Delete statement removes one or more columns from one or more rows of a Cassandra table or removes entire rows if no columns are specified.

DELETE LastExamPercent FROM student_info WHERE RollNo=2;

Collections

Collections

When to use collection?

Use collection when it is required to store or denormalize a small amount of data.

What is the limit on the values of items in a collection?

The values of items in a collection are limited to 64K.

Where to use collections?

Collections can be used when you need to store the following:

1. Phone numbers of users.
2. Email ids of users.

Collections - Set

To alter the schema for the table “student_info” to add a column “hobbies”.

ALTER TABLE student_info ADD hobbies set<text>;

Collections - Set

To update the table “student_info” to provide the values for “hobbies” for the student with Rollno =1.

```
UPDATE student_info
     SET hobbies = hobbies + {'Chess, Table

                Tennis'} WHERE RollNo=1;
```

Collections - List

To alter the schema of the table “student_info” to add a list column “language”.

ALTER TABLE student_info ADD language list<text>;

Collections - List

To update values in the list column, “language” of the table “student_info”.

```
UPDATE student_info
        SET language = language + ['Hindi,
            English'] WHERE RollNo=1;
```

Collections - Map

To alter the “users” table to add a map column “todo”.

```
ALTER TABLE users
    ADD todo map<timestamp, text>;
```

Collections - Map

To update the record for user (user_id = ‘AB’) in the “users” table.

```
UPDATE users
    SET todo =
    { ‘2014-9-24’: ‘Cassandra Session’,
    ‘2014-10-2 12:00’ : ‘MongoDB Session’
    }
        WHERE user_id = ‘AB’;
```

Time To Live

Time To Live

Data in a column, other than a counter column, can have an optional expiration period called TTL (time to live). The client request may specify a TTL value for the data. The TTL is specified in seconds.

```
CREATE TABLE userlogin(
userid int primary key, password text
);
INSERT INTO userlogin (userid, password) VALUES (1,'infy') USING TTL


30; SELECT TTL (password)


        FROM userlogin
                WHERE userid=1;
```

Export to CSV

Export data to a CSV file

Export the contents of the table/column family “elearninglists” present in the “students” database to a CSV file (d:\elearninglists.csv).

COPY elearninglists (id, course_order, course_id, courseowner, title)
TO
'd:\elearninglists.csv';

Import from CSV

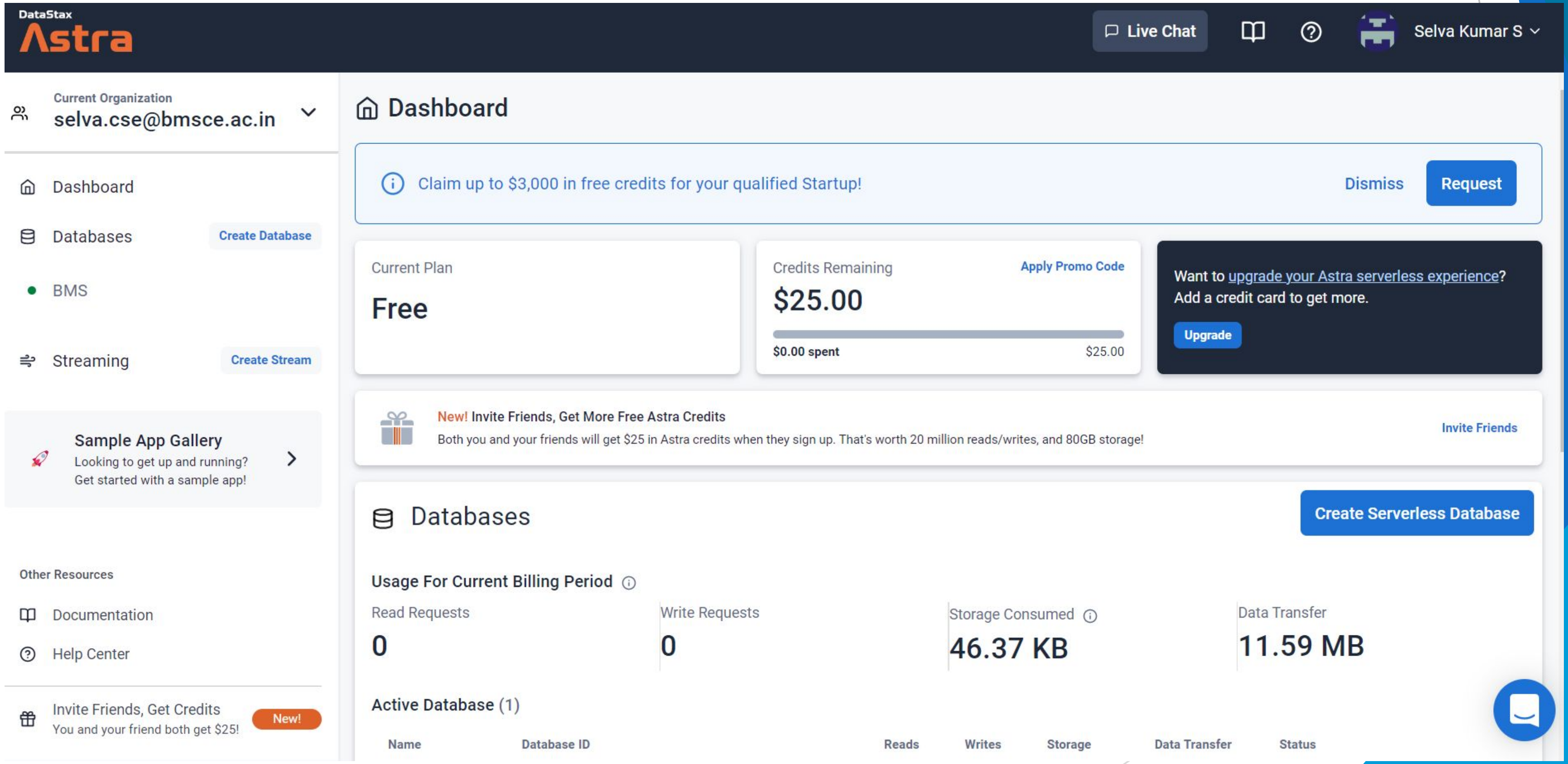
Import data from a CSV file

To import data from “D:\elearninglists.csv” into the table “elearninglists” present in the “students” database.

```
COPY elearninglists (id, course_order, course_id, courseowner,
title)
FROM 'd:\elearninglists.csv';
```

GETTING INITIAL HANDS-ON EXPERIENCE

- Create account : Datastax.com



Dashboard

Claim up to $3,000 in free credits for your qualified Startup! Dismiss Request

Current Plan	Credits Remaining
Free	$25.00
	$0.00 spent / $25.00

Apply Promo Code

Want to upgrade your Astra serverless experience? Add a credit card to get more. Upgrade

New! Invite Friends, Get More Free Astra Credits
Both you and your friends will get $25 in Astra credits when they sign up. That's worth 20 million reads/writes, and 80GB storage! Invite Friends

Databases — Create Serverless Database

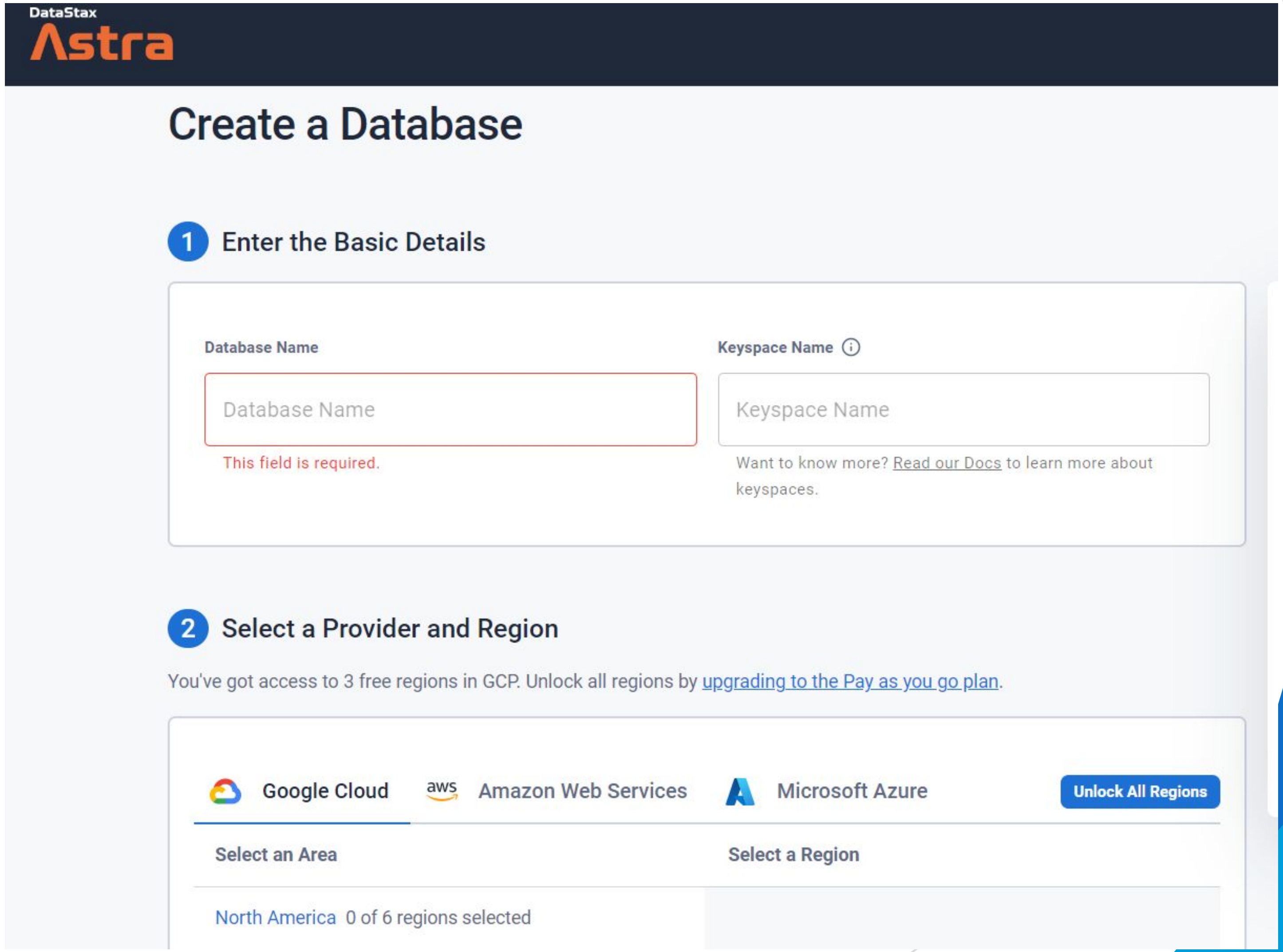
Usage For Current Billing Period

Read Requests	Write Requests	Storage Consumed	Data Transfer
0	0	46.37 KB	11.59 MB

Active Database (1)

Name	Database ID	Reads	Writes	Storage	Data Transfer	Status

Create a database



DataStax Astra

Create a Database

1 Enter the Basic Details

Database Name

Database Name

This field is required.

Keyspace Name

Keyspace Name

Want to know more? Read our Docs to learn more about keyspaces.

2 Select a Provider and Region

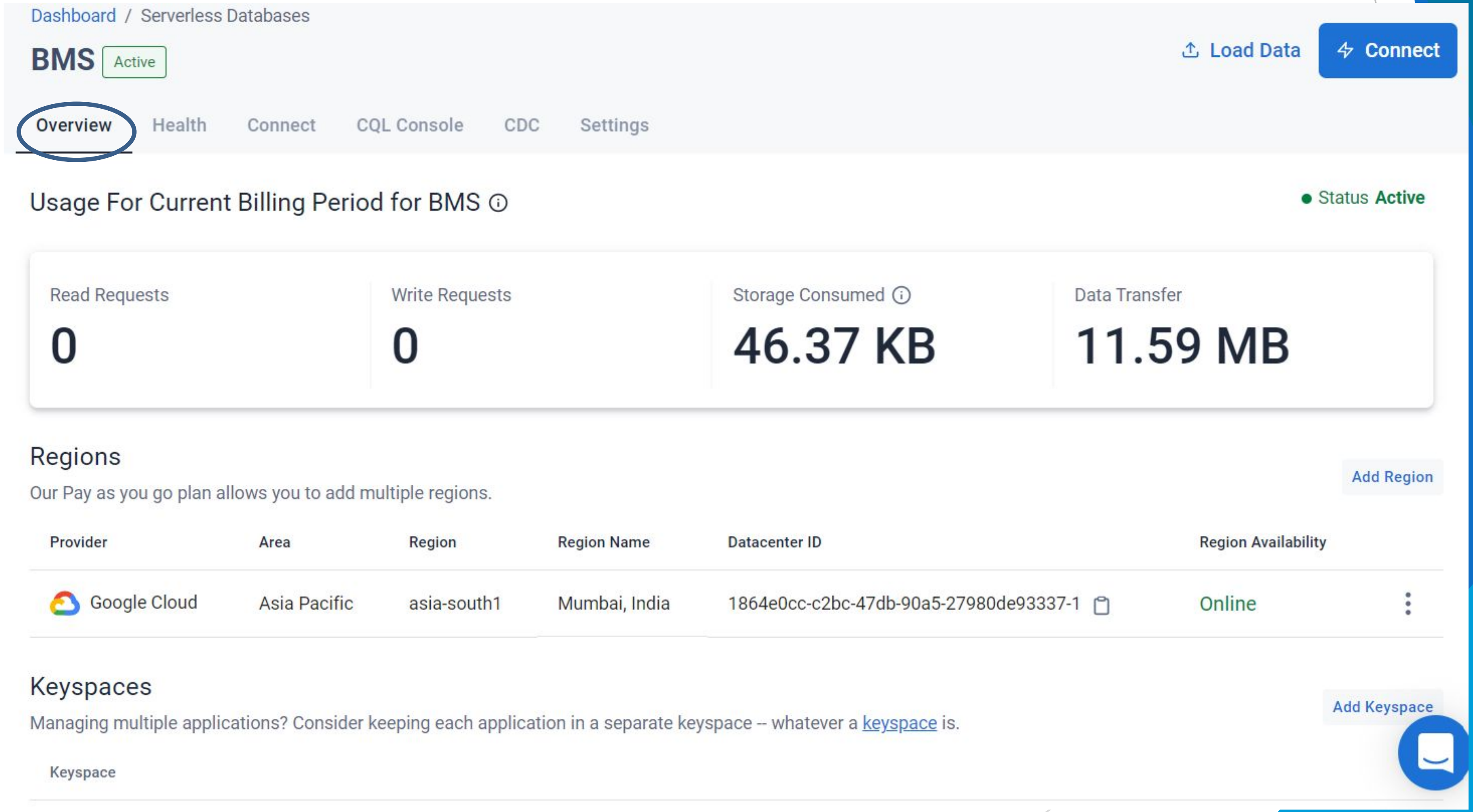
You've got access to 3 free regions in GCP. Unlock all regions by upgrading to the Pay as you go plan.

Google Cloud | Amazon Web Services | Microsoft Azure | Unlock All Regions

Select an Area | Select a Region

North America 0 of 6 regions selected

dashboard


Dashboard / Serverless Databases

BMS Active

Load Data | Connect

Overview | Health | Connect | CQL Console | CDC | Settings

Usage For Current Billing Period for BMS

● Status Active

Read Requests	Write Requests	Storage Consumed	Data Transfer
0	0	46.37 KB	11.59 MB

Regions

Our Pay as you go plan allows you to add multiple regions.

Add Region

Provider	Area	Region	Region Name	Datacenter ID	Region Availability
Google Cloud	Asia Pacific	asia-south1	Mumbai, India	1864e0cc-c2bc-47db-90a5-27980de93337-1	Online

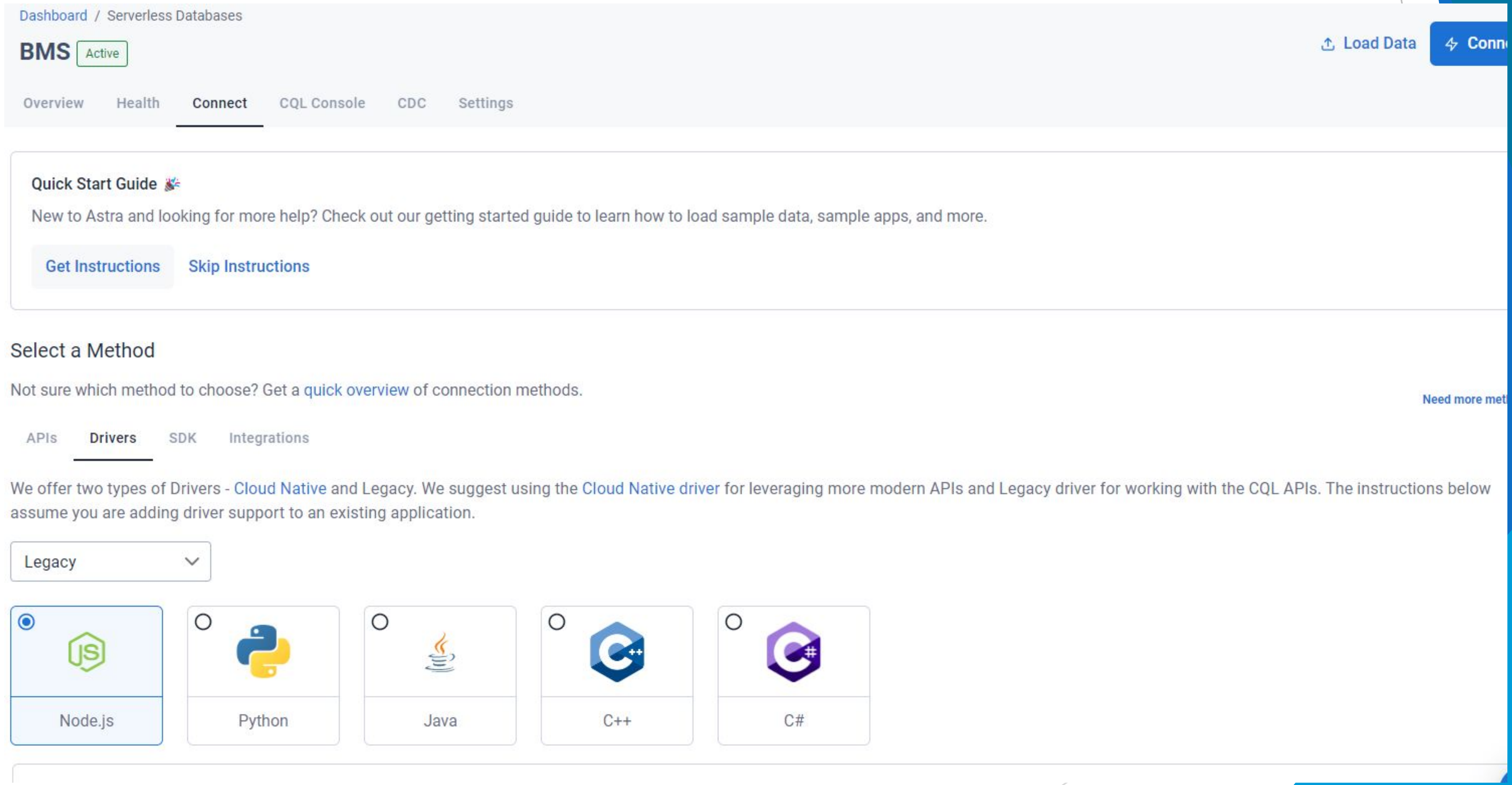
Keyspaces

Managing multiple applications? Consider keeping each application in a separate keyspace -- whatever a keyspace is.

Add Keyspace

Keyspace

Connect tab



Dashboard / Serverless Databases

BMS Active

Load Data Conn

Overview Health Connect CQL Console CDC Settings

Quick Start Guide

New to Astra and looking for more help? Check out our getting started guide to learn how to load sample data, sample apps, and more.

Get Instructions Skip Instructions

Select a Method

Not sure which method to choose? Get a quick overview of connection methods.

Need more met

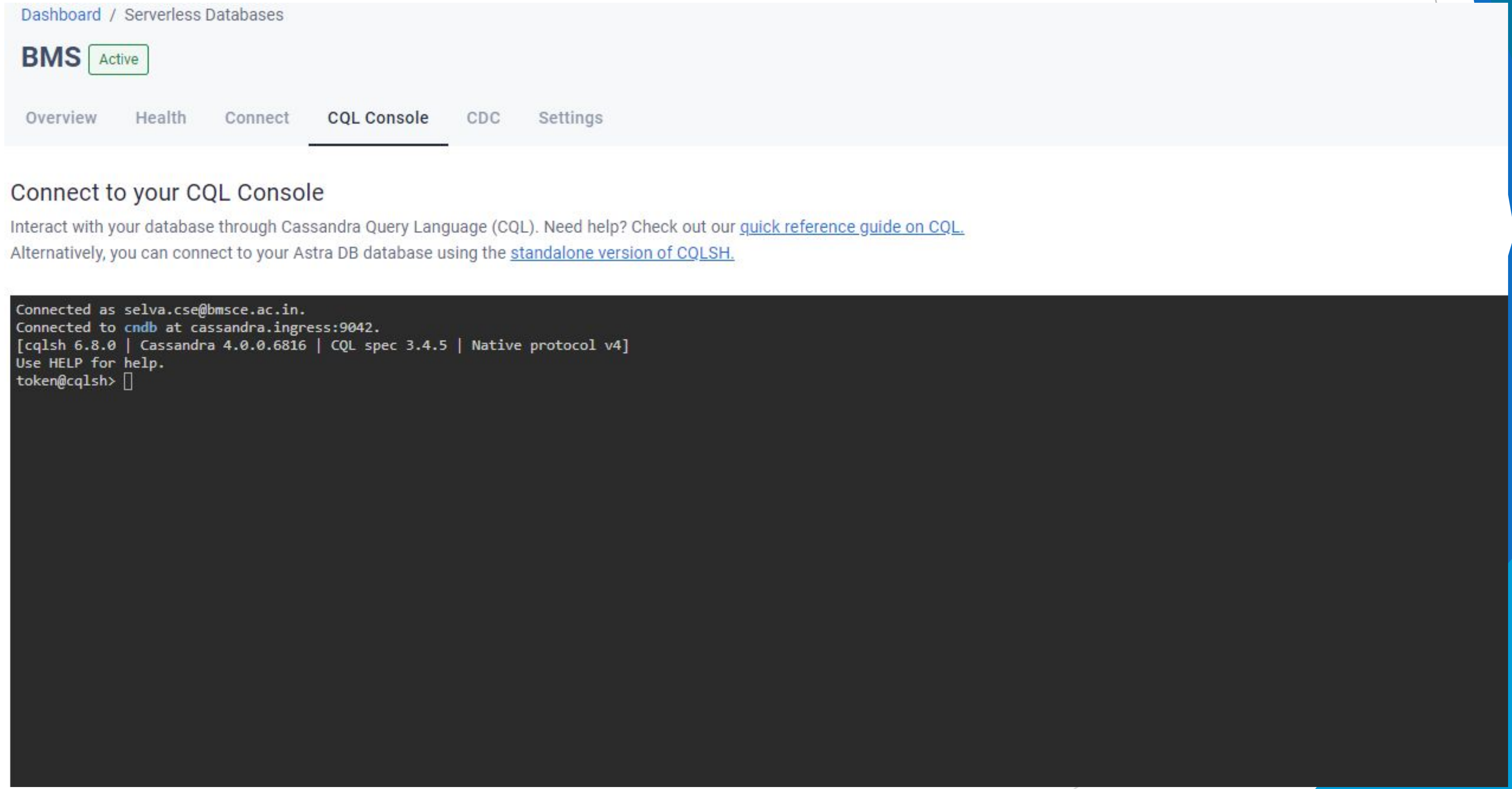
APIs Drivers SDK Integrations

We offer two types of Drivers - Cloud Native and Legacy. We suggest using the Cloud Native driver for leveraging more modern APIs and Legacy driver for working with the CQL APIs. The instructions below assume you are adding driver support to an existing application.

Legacy

[✓ Node.js] Python Java C++ C#

CQL console


Dashboard / Serverless Databases

BMS Active

Connect to your CQL Console

Interact with your database through Cassandra Query Language (CQL). Need help? Check out our quick reference guide on CQL.
Alternatively, you can connect to your Astra DB database using the standalone version of CQLSH.

```
Connected as selva.cse@bmsce.ac.in.
Connected to cndb at cassandra.ingress:9042.
[cqlsh 6.8.0 | Cassandra 4.0.0.6816 | CQL spec 3.4.5 | Native protocol v4]
Use HELP for help.
token@cqlsh> 
```

Cql data definition commands

- **CREATE KEYSPACE −** Creates a KeySpace in Cassandra.
- **USE −** Connects to a created KeySpace.
- **ALTER KEYSPACE −** Changes the properties of a KeySpace.
- **DROP KEYSPACE −** Removes a KeySpace
- **CREATE TABLE −** Creates a table in a KeySpace.
- **ALTER TABLE −** Modifies the column properties of a table.
- **DROP TABLE −** Removes a table.
- **TRUNCATE −** Removes all the data from a table.
- **CREATE INDEX −** Defines a new index on a single column of a table.
- **DROP INDEX −** Deletes a named index.

CQL Data Manipulation Commands

- **INSERT** − Adds columns for a row in a table.
- **UPDATE** − Updates a column of a row.
- **DELETE** − Deletes data from a table.
- **BATCH** − Executes multiple DML statements at once.

- CQL Clauses
- SELECT **-**− This clause reads data from a table

demo

```
describe keyspaces;
or desc keyspaces;
use demo;
describe tables;
describe table emp;
describe type emp;
```

Demo contd..

```
create table student ( name text, usn text, mob int, id int, primary key(id));

desc student;

 select * from student;

insert into student(id, usn,name,mob) values (1, '1bm20cs001','abc',2334233423);

update student set name='xyz' where id=1;

update student set name='xyz' where id=2;  // no error instead create new row
```

Demo contd..

```
CREATE TABLE emp( emp_id int PRIMARY KEY,  emp_name text,  emp_city text, emp_sal varint,
emp_phone varint );

Select * from emp;

INSERT INTO emp (emp_id, emp_name, emp_city,
 emp_phone, emp_sal) VALUES(1,'ram', 'Hyderabad', 9848022338, 50000);

UPDATE emp SET emp_city='Delhi',emp_sal=50000 WHERE emp_id=2;

DELETE emp_sal FROM emp WHERE emp_id=3;
```

Demo contd..

uuid() function which is very important to insert value and to uniquely generates “guaranteed unique” UID value Universally.

```
Create table function4(Id uuid primary key, name text);

Insert into function4 (Id, name) values (1, ‘Ashish’); // fails

Insert into function4(Id, name) values (now(), ‘Ashish’);  //correct
```

Demo contd..

```
Alter table emp add (emails set<text>, frameworks list<text>, previous_jobs map<text, int>);
INSERT INTO emp (emp_id, emp_name, emp_city, emp_phone, emp_sal, emails,
frameworks,previous_jobs) VALUES(6,'ram', 'Hyderabad', 9848022338, 50000,
{‘ram@gmail.com’,’ram@hotmail.com’}, [‘C#’,’java’,’node.js’],{‘Amazon’:4,’TCS’:3});
```

Interfacing and Interacting with NoSQ

In programming language to connect application with database there is a programming Pattern.

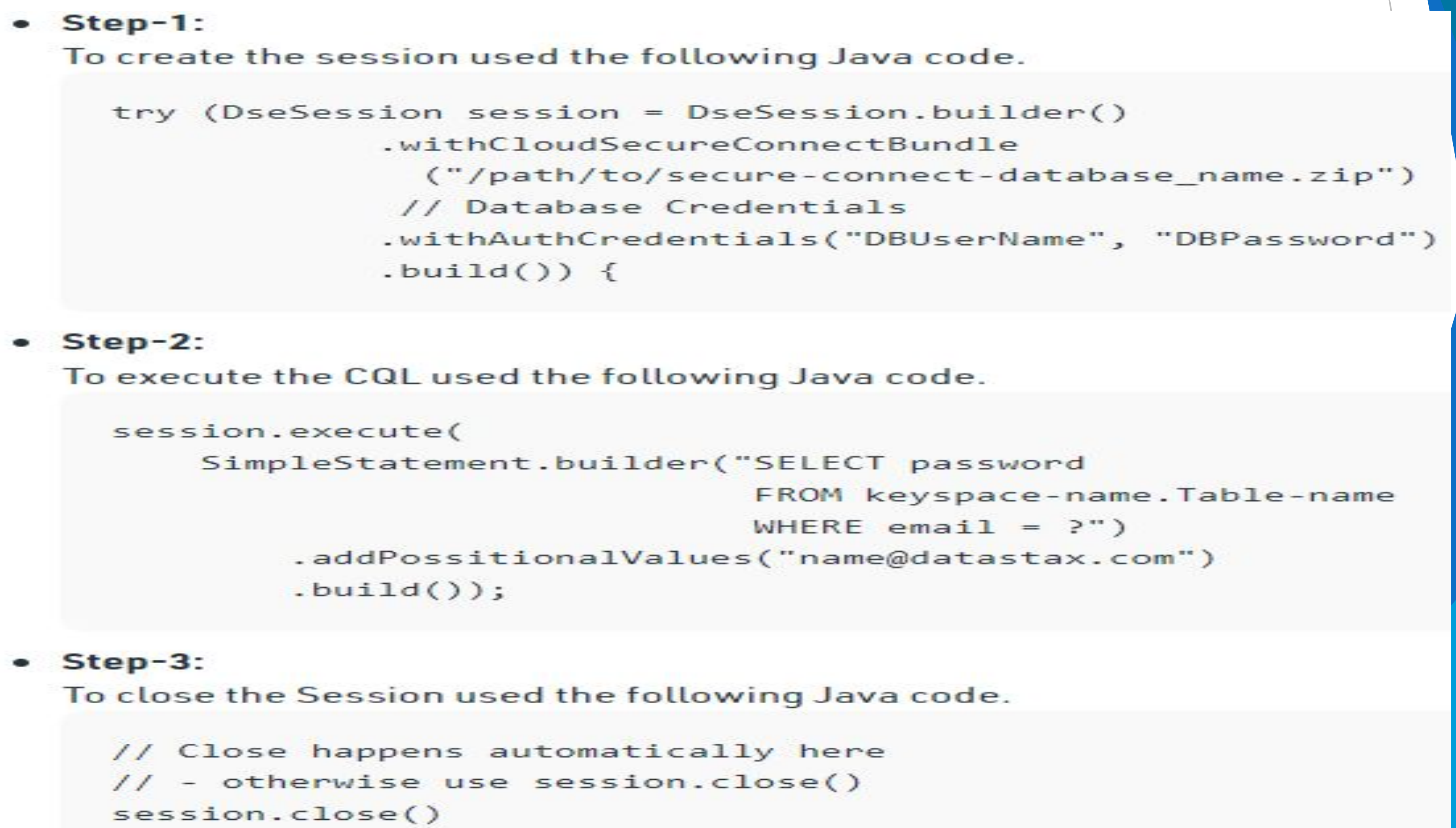
Three Easy steps are following :

Create a connection (which is called a Session)

Use the session to execute the query.

Close the connection/session.

java code to connect db



- **Step-1:**
  To create the session used the following Java code.

```
try (DseSession session = DseSession.builder()
            .withCloudSecureConnectBundle
              ("/path/to/secure-connect-database_name.zip")
             // Database Credentials
            .withAuthCredentials("DBUserName", "DBPassword")
            .build()) {
```

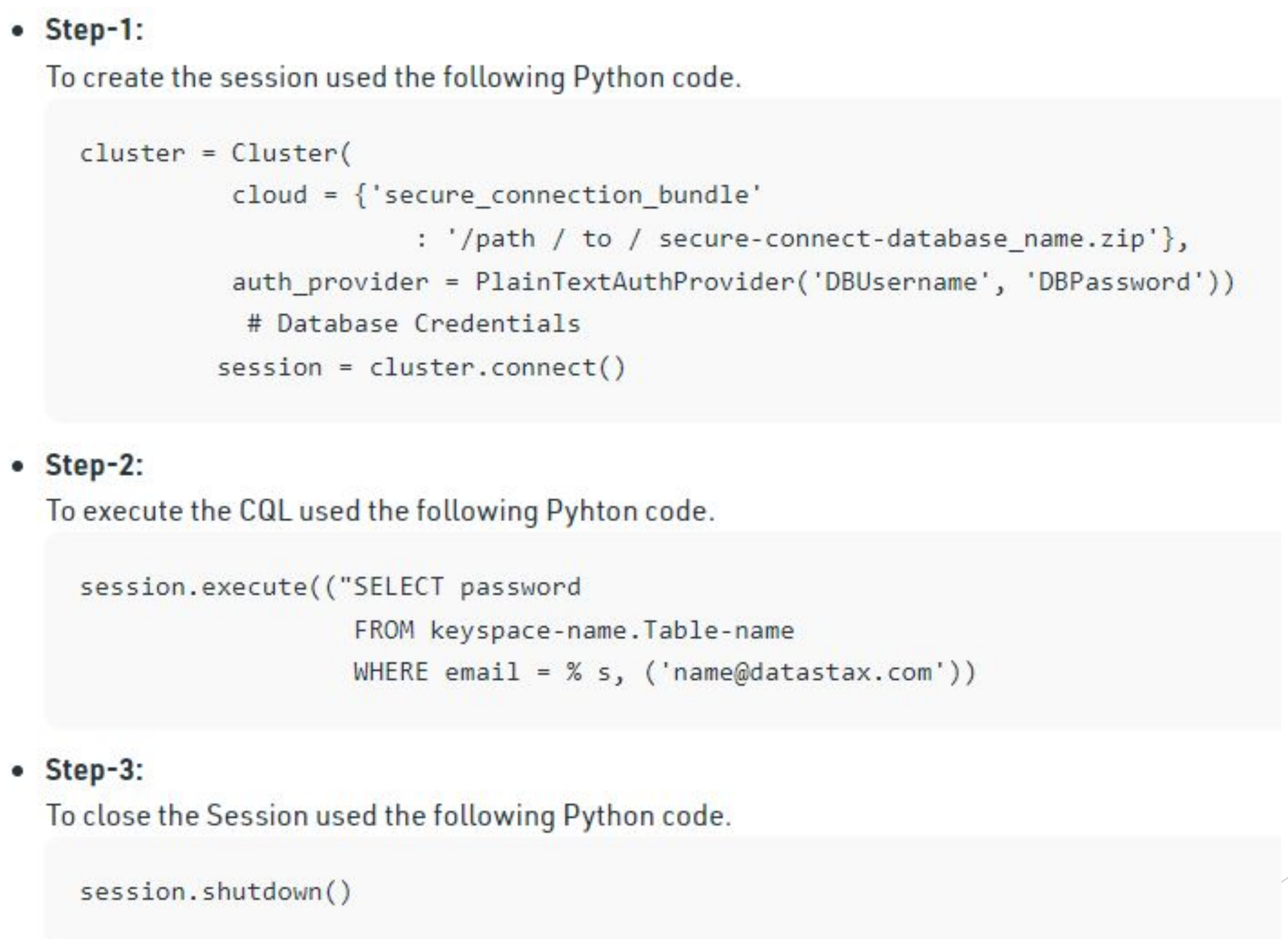
- **Step-2:**
  To execute the CQL used the following Java code.

```
session.execute(
     SimpleStatement.builder("SELECT password
                              FROM keyspace-name.Table-name
                              WHERE email = ?")
        .addPossitionalValues("name@datastax.com")
        .build());
```

- **Step-3:**
  To close the Session used the following Java code.

```
// Close happens automatically here
// - otherwise use session.close()
session.close()
```

python code to connect db



- **Step-1:**
  To create the session used the following Python code.

```
cluster = Cluster(
          cloud = {'secure_connection_bundle'
                      : '/path / to / secure-connect-database_name.zip'},
          auth_provider = PlainTextAuthProvider('DBUsername', 'DBPassword'))
           # Database Credentials
         session = cluster.connect()
```

- **Step-2:**
  To execute the CQL used the following Pyhton code.

```
session.execute(("SELECT password
                  FROM keyspace-name.Table-name
                  WHERE email = % s, ('name@datastax.com'))
```

- **Step-3:**
  To close the Session used the following Python code.

```
session.shutdown()
```

Answer a few quick questions …

Answer Me

- What is Cassandra?
- Comment on Cassandra writes.
- What is your understanding of tunable consistency?
- What are collections in CQLSH? Where are they used?

References …

Further Readings

- http://www.datastax.com/documentation/cassandra/2.0/cassandra/getting S tartedCassandraIntro.html
- http://www.datastax.com/documentation/cql/3.1/pdf/cql31.pdf
- http://www.datastax.com/documentation/cassandra/2.0/cassandra/dml/dm l_config_consistency_c.html

Thank you