

Apache Cassandra



Outlines

- *Query language*
- *Connect Python to Apache Cassandra*

“

I don't always use Cassandra but when I do, I denormalize

- Meme.

Apache Cassandra

- ⬡ A distributed NoSQL database system for managing large amounts of data across many commodity servers, while providing highly available service and no single point of failure.



Key features

- ⬡ Distributed and decentralized
- ⬡ Fault tolerance
- ⬡ Tunable consistency
- ⬡ Column oriented
- ⬡ CQL query
- ⬡ High performance

Distributed and decentralized

- ⬡ Distributed: capable of running on multiple machines
- ⬡ Decentralized: No single point of failure.
- ⬡ No master-slave issues due to peer-to-peer architecture

Elastic Scalability

- ⬡ Cassandra scales horizontally adding more machines that have all or some of the data on
- ⬡ Adding of nodes increase performance throughput linearly
- ⬡ Decreasing and increasing the node count happen seamlessly.

High availability and fault tolerance

- ⬡ Multiple networked computers operating in a cluster
- ⬡ Facility for recognizing node failures
- ⬡ Forward failing over requests to another part of the system

Column oriented Key-Value Store

- ⬡ Data is stored in sparse multidimensional hash tables
- ⬡ A row can have multiple columns not necessarily the same amount of columns for each row
- ⬡ Each row has a unique key, which also determines partitioning

R1	C1 Key	C2 Key	C3Key
	C1 Value	C3 Value	C3 Value	
R2	C4 Key	C5 Key		
	C4 Value	C5 Value		
.....				

Cassandra Query Language

- ⬡ "CQL 3 is the default and primary interface into the Cassandra DBMS"
- ⬡ Familiar SQL-like syntax that maps to Cassandras storage engine and simplifies data modelling

Cassandra Query Language

```
CREATE TABLE songs (  
    Id uuid PRIMARY KEY, title text,  
    Album text, Artist text,  
    data blob );
```

```
SELECT * FROM songs  
WHERE id = 'a3e64f8f...';
```

```
SELECT * FROM songs ;
```

```
INSERT INTO songs (id, title, album, artist)  
VALUES( 'a3e64f8f...', 'Hazim ra3d', 'Spacetoon', 'Tarkan' );
```

Cassandra Query Language

```
INSERT INTO songs (id, title)  
VALUES( 'a3e64f8f...', 'Al Kanas');
```

This is Possible With Cassandra



Cassandra Query Language

The resulting table in RDMBS is this:

<u>id</u>	<u>title</u>	<u>artist</u>	<u>album</u>	<u>data</u>
a3e64f8f...	Hazim Ra3d	Tarkan	Spacetoan	<i>null</i>
g617Dd23...	Al Kanas	<i>null</i>	<i>null</i>	<i>null</i>

Cassandra Query Language

The resulting table in Cassandra is this:

<u>id</u>	<u>title</u>	<u>artist</u>	<u>album</u>	<u>data</u>
a3e64f8f...	Hazim Razd	Tarkan	Spacetoan	
g617Dd23...	Al Kanas			

MySQL Comparison

Statistics based on 50 GB Data

	Cassandra	MySQL
Average Write	0.12 ms	~300 ms
Average Read	15 ms	~350 ms

Stats provided by Authors using Facebook data.

Data Model

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.

Data Model

Keyspace

Outermost container for data (one or more column families), like database in RDBMS.



Column family

Contains Super columns or Columns (but not both).

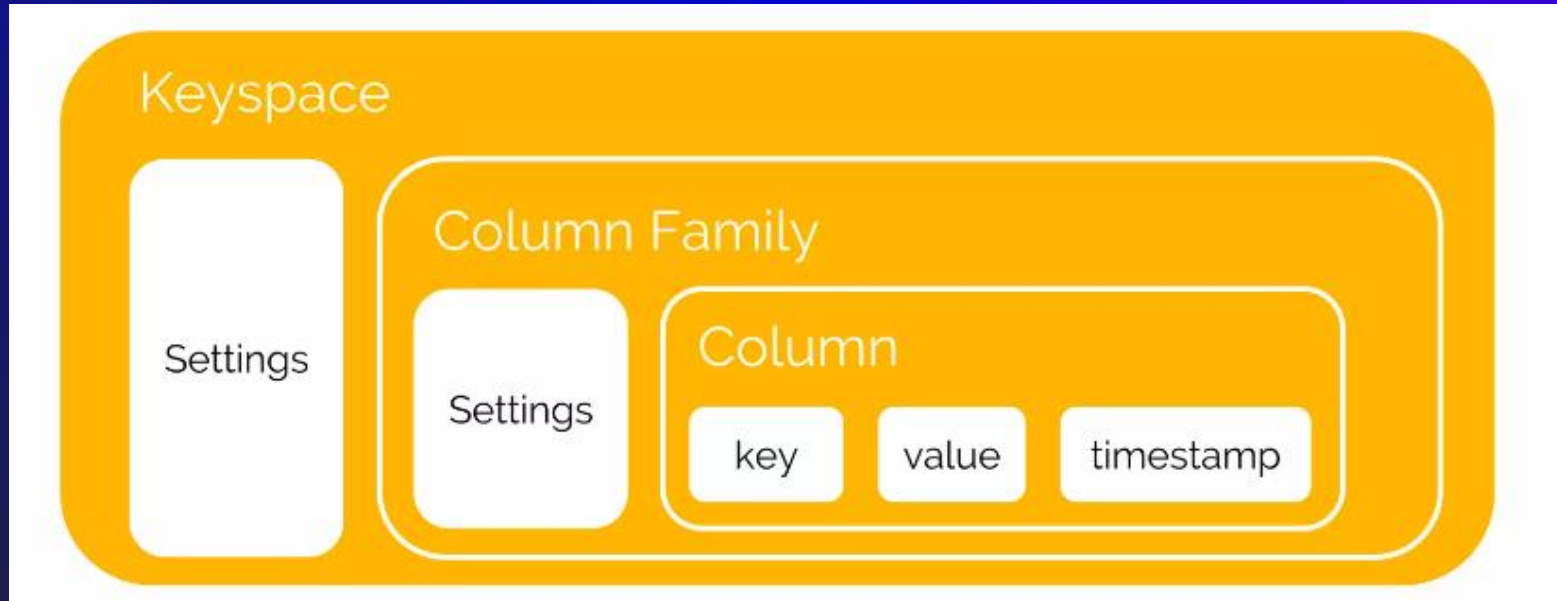


Column

Basic data structures with: key, value, timestamp

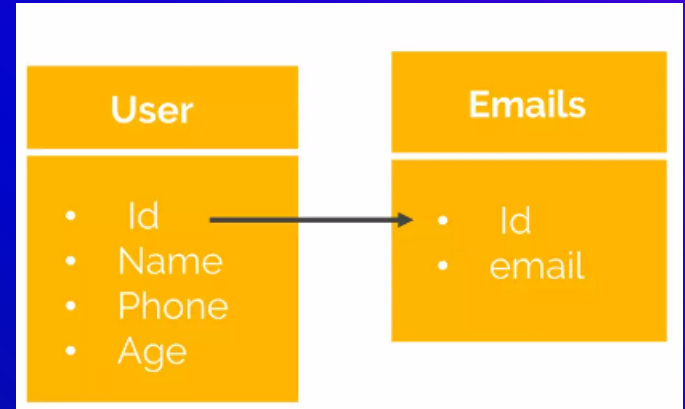


Data Model



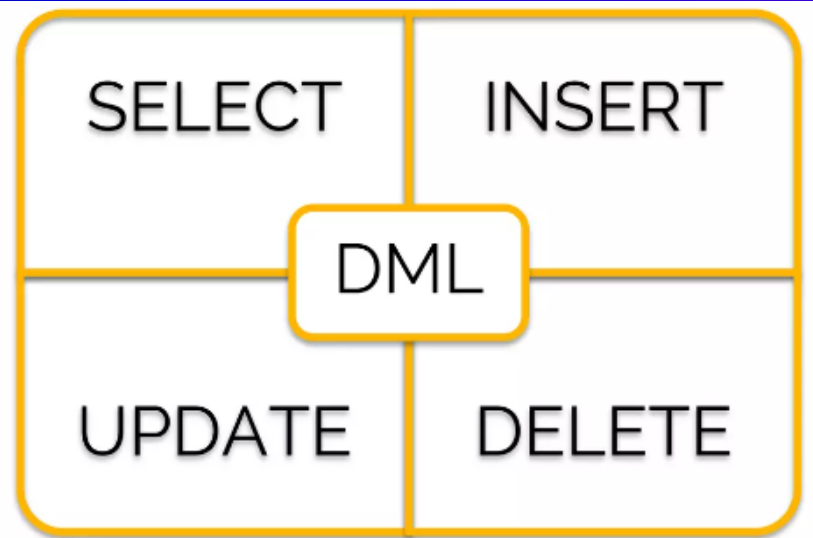
Example using CQL

- ⬡ The Following Slides will demonstrate different cases with different CQL interfaces like DDL, DML etc..



Interface DML

The DML Interface is
the Same With
Normal SQL DML



Interface DDL

Same as SQL, but with
keyspaces and types
option added.

DROP

- Type
- Keyspace , Table
- Index , Trigger

CREATE

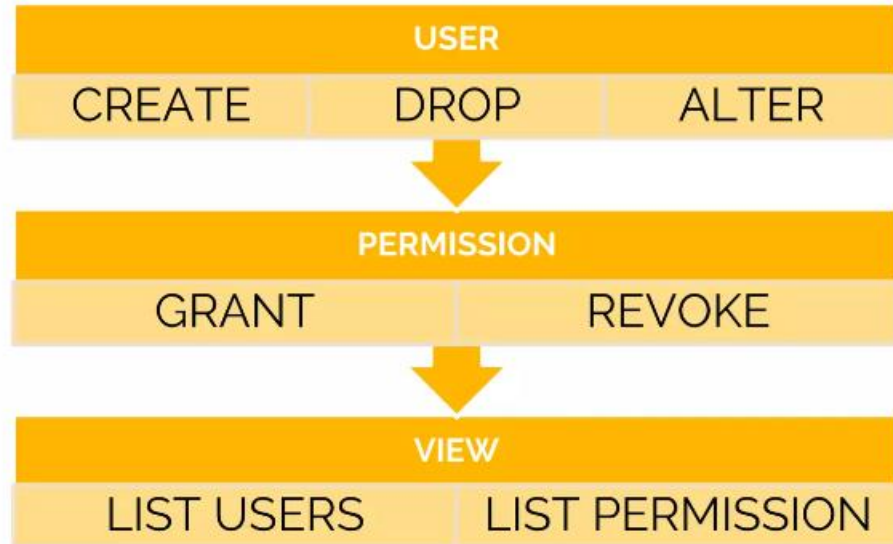
- Type
- Keyspace , Table
- Index , Trigger

ALTER

- Type
- Keyspace , Table
- Index , Trigger

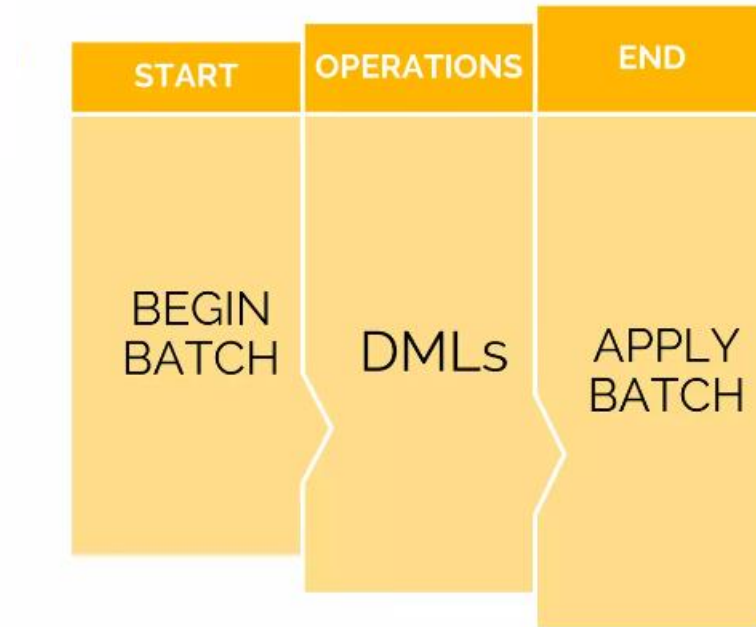
Interface DCL

Create users (Roles),
give them permission,
and start using them.



Interface TCL

For multiple operations use the **BATCH** command



Metadata using describe



keyspace

Describe keyspace name



Table

Describe table keyspace_name .table_name



Others

Describe keyspaces, tables, schema

Metadata Keys space

Query the defined key spaces using the SELECT statement.

```
SELECT * FROM  
system_schema.keyspaces
```

keyspace_name	durable_writes	replication
test	True	['class': 'org.apache']....
.....

Metadata Table

Getting information about tables in the test keyspace.

```
SELECT * FROM system_schema.tables  
WHERE keyspace_name = 'test';
```

keyspace_name	table_name
test	users
.....

Metadata Columns

Getting information about columns in the users tables.

```
SELECT * FROM system_schema.columns  
WHERE keyspace_name = 'test' AND table_name = 'users';
```

table_name	column_name	kind	type
users	age	regular	int
.....

Logging with System.log

To see what is happening in the database, you can use the system.log file in the Cassandra home to directory to track creational query.

```
{CASSANDRA HOME}/utils/cassandra.logdirISUNDEFINED/
```

Here is an Example

```
{CASSANDRA HOME}/utils/cassandra.logdirISUNDEFINED/
```

Logging with System.log

Here is an Example

```
INFO [main] 2018-11-08 23:48:36,960
MigrationManager.java:302 - Create new Keyspace:
    KeyspaceMetadata {name=system_traces,
    params=KeyspaceParams {durable_writes=true,
    replication=ReplicationParams
    {class=org.apache.cassandra.locator.SimpleStrategy,
    replication_factor=2 }
```

Logging with Tracing

It's an option to activate in the Cassandra database

```
TRACING [ ON | OFF]
```

The result will be on different keyspace called `system_traces`. In a table called `events`

```
USE system_traces;  
SELECT * FROM events;
```

Logging with Tracing

Example:

```
INSERT INTO product(id , name) VALUES (UUID(), 'Hello');
```

Result:

Execute CQL3 query

Parsing insert into product(id , name) values(UUID(), 'Hello');

Preparing statement

.....

Strengths

- ⬡ **Linear scale performance** The ability to add nodes without failures leads to predictable Increases in performance
- ⬡ **Supports multiple languages** Python, C#/.NET, C++, Ruby, Java, Go, and many more..
- ⬡ **Operational and developmental simplicity** There are no complex software tiers to be managed, so administration duties are greatly simplified.

Strengths (2)

- ⬡ **Ability to deploy across data centers** Cassandra can be deployed across multiple, geographically dispersed data centers
- ⬡ **Cloud availability** Installations in cloud environments
- ⬡ **Peer to peer architecture** Cassandra follows a peer-to-peer architecture, instead of master-slave architecture

Strengths (3)

- ⬡ **Flexible data model** Supports modern data types with fast writes and reads
- ⬡ **Fault tolerance** Nodes that fail can easily be restored or replaced
- ⬡ **High Performance** Cassandra has demonstrated brilliant performance under large sets of data

Strengths (4)

- ⬡ **Schema-free/Schema-less** In Cassandra, columns can be created at your will within the rows. Cassandra data model is also famously known as a schema-optional data model
- ⬡ **AP-CAP** Cassandra is typically classified as an AP system, meaning that availability and partition tolerance are generally considered to be more important than consistency in Cassandra

Weaknesses (1)

Use Cases where is better to avoid using Cassandra:

- ⬡ If there are too many joins required to retrieve the data
- ⬡ To store configuration data
- ⬡ During compaction, things slow down and throughput degrades
- ⬡ Basic things like aggregation operators are not supported
- ⬡ Range queries on partition key are not supported

Weaknesses (2)











Use Cases where is better to avoid using Cassandra

- ⬡ If there are transactional data which require 100% consistency
- ⬡ Cassandra can update and delete data but it is not designed to do so

Use DataStax to try Cassandra

DATAStax

Let's find out how to use datastax



Welcome to Astra, Mohamed 🙌

Accelerate your workflows, access recently visited resources, and explore Astra's integrations and documentation!

Upgrade

What's New

7 steps · 15 minutes

Getting Started with Vector Search ⚡

Learn Vector Search basics like creating a schema and reading vector data with CQL and Python.

[Go to guide](#)

5 steps · 10 minutes

Overview of Astra DB 🌿

Understand Astra DB and how it differs from its underlying technology, Apache Cassandra®.

[Go to guide](#)


4 steps · 15 minutes


Intro to data with Astra ⚡


Get an overview of how to execute basic CRUD (Create, Read, Update, Delete) commands in Astra using CQL.

[Go to guide](#)


Quick Access

 Create a Database →


 Create a Streaming Tenant →


 Invite your team →

Recent Resources

**workshops**
Vector Database · Active

Recommended Integrations

**GCP Dataflow**
Open-source, unified programming model



Go to guide to learn how to use it

The screenshot displays the Astra DB dashboard. On the left is a vertical sidebar with icons for database management, navigation, and user profile. The main content area features a welcome message 'Welcome to Astra, Mohamed' with a user icon. Below this is a sub-header 'Accelerate your workflows, access recently visited resources, and explore Astra's integrations and documentation!' and an 'Upgrade' button. A 'What's New' section contains three guides: 'Getting Started with Vector Search' (7 steps, 15 minutes), 'Overview of Astra DB' (5 steps, 10 minutes), and 'Intro to data with Astra' (4 steps, 15 minutes). Each guide includes a brief description and a 'Go to guide' link. A blue arrow points to the 'Go to guide' link for 'Intro to data with Astra'. Below the guides is a 'Quick Access' section with three buttons: 'Create a Database', 'Create a Streaming Tenant', and 'Invite your team'. At the bottom, there are two sections: 'Recent Resources' showing 'workshops' for 'Vector Database · Active', and 'Recommended Integrations' showing 'GCP Dataflow' as an 'Open-source, unified programming model'. A chat icon is visible in the bottom right corner.

Welcome to Astra, Mohamed

Accelerate your workflows, access recently visited resources, and explore Astra's integrations and documentation! [Upgrade](#)

What's New

- Getting Started with Vector Search** ⚡
7 steps · 15 minutes
Learn Vector Search basics like creating a schema and reading vector data with CQL and Python.
[Go to guide](#)
- Overview of Astra DB** 🌱
5 steps · 10 minutes
Understand Astra DB and how it differs from its underlying technology, Apache Cassandra®.
[Go to guide](#)
- Intro to data with Astra** ⚡
4 steps · 15 minutes
Get an overview of how to execute basic CRUD (Create, Read, Update, Delete) commands in Astra using CQL.
[Go to guide](#)

Quick Access

- [Create a Database](#) →
- [Create a Streaming Tenant](#) →
- [Invite your team](#) →

Recent Resources

- workshops**
Vector Database · Active

Recommended Integrations

- GCP Dataflow**
Open-source, unified programming model

Creating database on datastax cloud

DS

1 Create a database

First, create a database with the following database and keyspace name with the button below.

Database Created

Your database '**workshops**' was successfully created and activated on 2023-07-04T09:56:13Z.

Database Name

workshops

Keyspace Name

chatsandra

Database Name:

workshops

Keyspace Name:

chatsandra











In this guide

- Create a database
- Create tables
- Execute CRUD operations
- Summary

Recommended Links

- [Intro to Apache Cassandra for Developers](#)
- [CQL quick reference](#)

To write create a table you have to use CQL console





2 Create tables

The next step is to create tables.

2a Launch the CQL Console and login to the database

Open the CQL Console using the button below.


You may want to put the console and this guide side by side for easy copying.

 Launch CQL Console 


2b Describe keyspaces and set keyspace context to chatsandra

Now you're ready to rock. Creating tables is easy, but before we create one, we need to tell the database which keyspace to use.

First, **DESCRIBE** all the keyspaces in the database. This gives you a list of the available keyspaces.

 **Command to execute**

```
DESC KEYSPACES;
```




In this guide

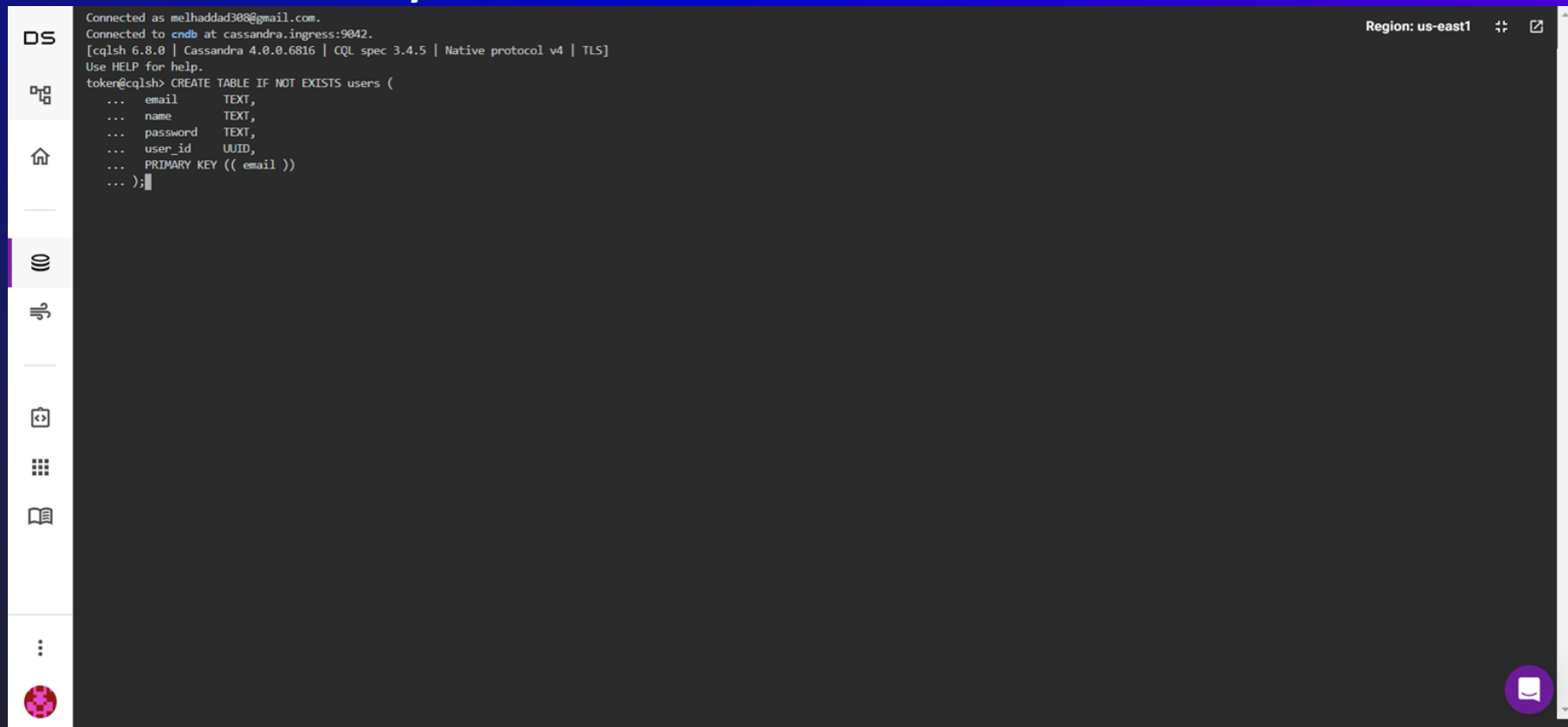
- Create a database
- Create tables
- Execute CRUD operations
- Summary

Recommended Links

- [Intro to Apache Cassandra for Developers](#)
- [CQL quick reference](#)



You will write every command here

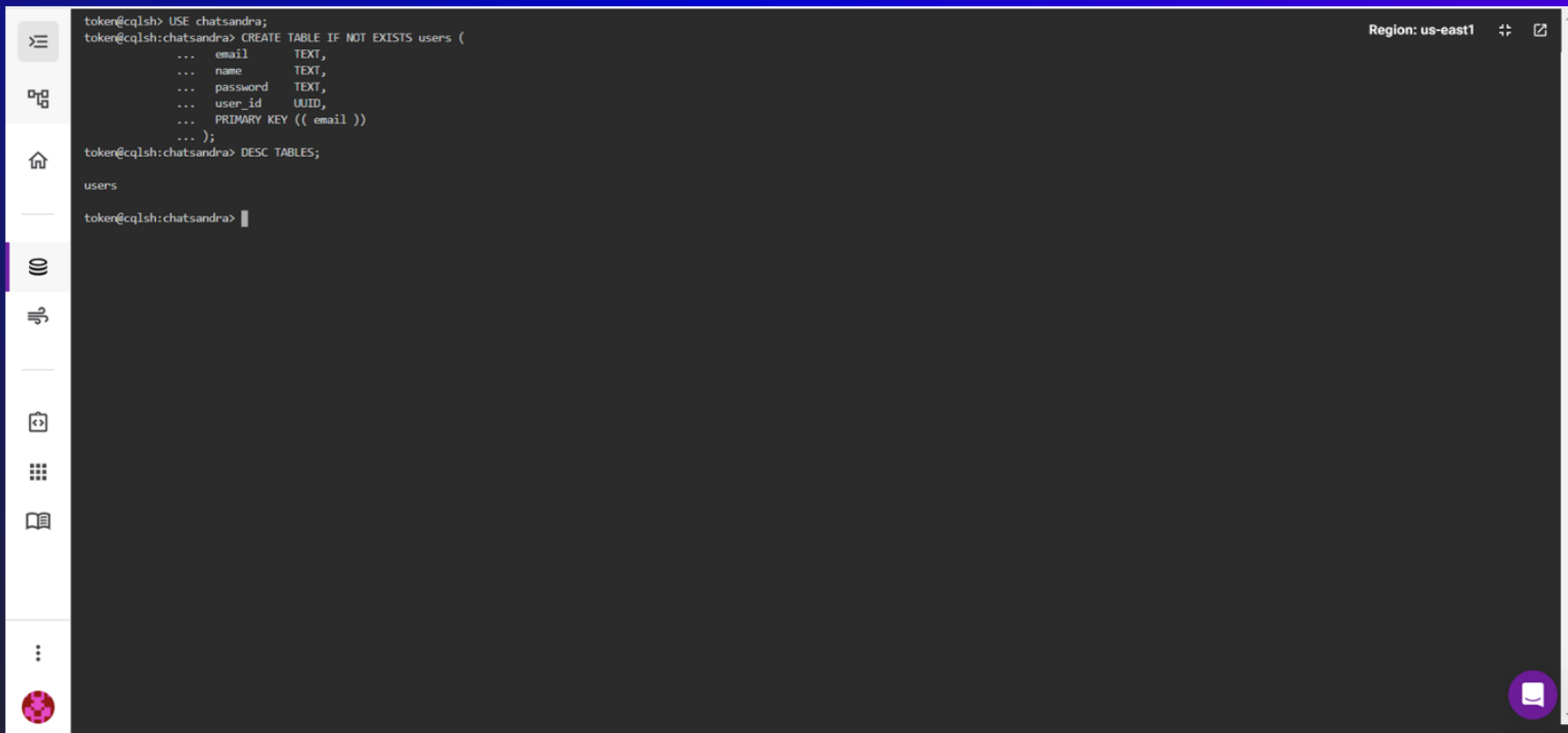


Connected as melhaddad308@gmail.com.
Connected to cndb at cassandra.ingress:9042.
[cqlsh 6.8.0 | Cassandra 4.0.0.6816 | CQL spec 3.4.5 | Native protocol v4 | TLS]
Use HELP for help.

```
token@cqlsh> CREATE TABLE IF NOT EXISTS users (  
...   email      TEXT,  
...   name       TEXT,  
...   password   TEXT,  
...   user_id    UUID,  
...   PRIMARY KEY (( email ))  
... );
```

Region: us-east1

You will write every command here



The image shows a terminal window with a dark background and a light-colored sidebar on the left. The sidebar contains several icons: a hamburger menu, a database icon, a home icon, a server icon (highlighted with a purple bar), a network icon, a calendar icon, a grid icon, a book icon, and a red circular icon at the bottom. The terminal text is as follows:

```
token@cqlsh> USE chatsandra;
token@cqlsh:chatsandra> CREATE TABLE IF NOT EXISTS users (
    ...     email      TEXT,
    ...     name       TEXT,
    ...     password   TEXT,
    ...     user_id    UUID,
    ...     PRIMARY KEY (( email ))
    ... );
token@cqlsh:chatsandra> DESC TABLES;

users

token@cqlsh:chatsandra> |
```

In the top right corner of the terminal window, it says "Region: us-east1".

Then we can make crud operation

DS

3 Execute CRUD operations

CRUD stands for "create, read, update, and delete". They are the basic types of commands you need to work with ANY database to maintain data for your applications.

3a (C)RUD = create = insert data

We created the **users** table in the step above. Add data to this table with the **INSERT** statement. Begin by inserting three rows into the **users** table.

Copy and paste the following in your CQL Console: *(We provided a few variations to get a feel for the syntax needed.)*

Commands to execute

```
INSERT INTO users (  
  email,    // TEXT  
  name,     // TEXT  
  password, // TEXT  
  user_id   // UUID: id of a user  
)  
VALUES (  
  'otzi@mail.com',  
  'Otzi Oney',  
  '123456',  
  11111111-1111-1111-1111-111111111111  
)  
);  
  
INSERT INTO users (email, name, password, user_id) VALUES (
```

In this guide

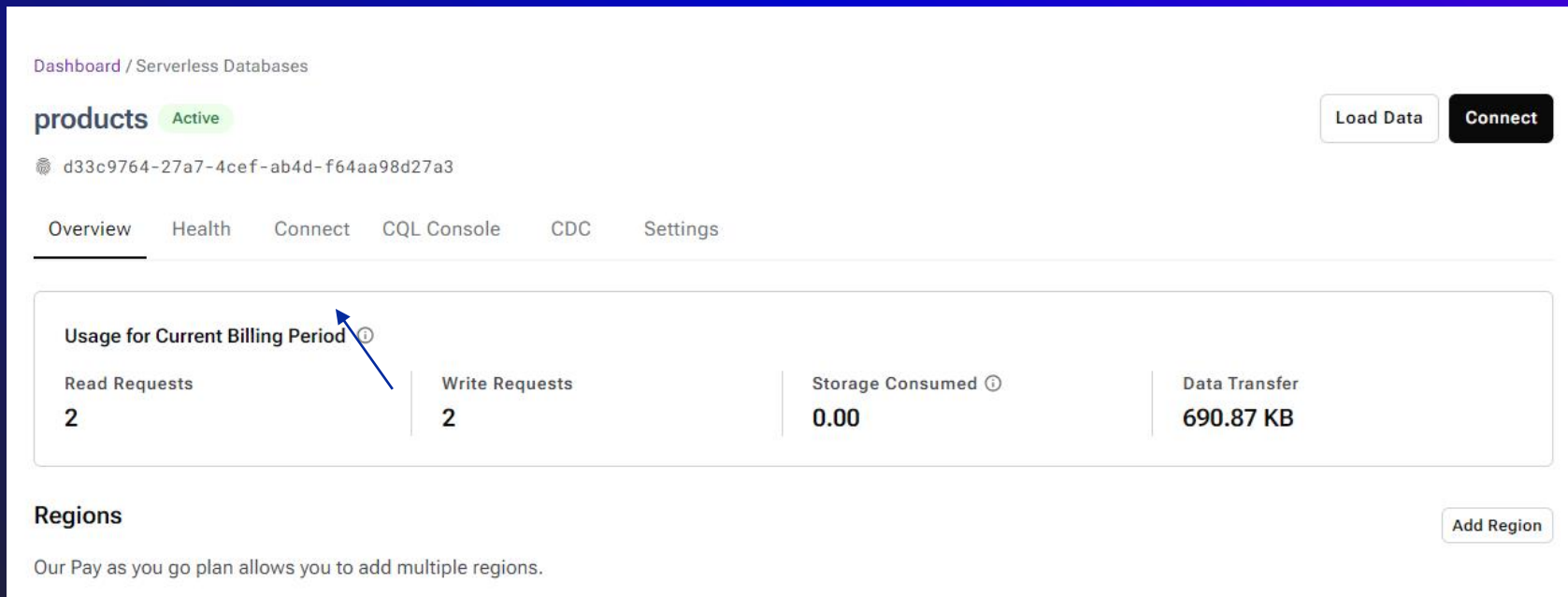
- Create a database
- Create tables
- Execute CRUD operations
- Summary

Recommended Links

- Intro to Apache Cassandra for Developers
- CQL quick reference

Connect DataStax with Python


Step1: after making a database you will have this dashboard, click on “Connect”



The screenshot shows the DataStax dashboard for a serverless database named 'products'. The dashboard includes a navigation bar with tabs for Overview, Health, Connect, CQL Console, CDC, and Settings. The 'Connect' tab is highlighted. In the top right corner, there are two buttons: 'Load Data' and 'Connect'. The main content area displays usage metrics for the current billing period, including Read Requests (2), Write Requests (2), Storage Consumed (0.00), and Data Transfer (690.87 KB). A blue arrow points to the 'Usage for Current Billing Period' header. Below the usage metrics, there is a section for 'Regions' with an 'Add Region' button.

Dashboard / Serverless Databases

products Active Load Data Connect

 d33c9764-27a7-4cef-ab4d-f64aa98d27a3

[Overview](#) [Health](#) [Connect](#) [CQL Console](#) [CDC](#) [Settings](#)

Usage for Current Billing Period ⓘ

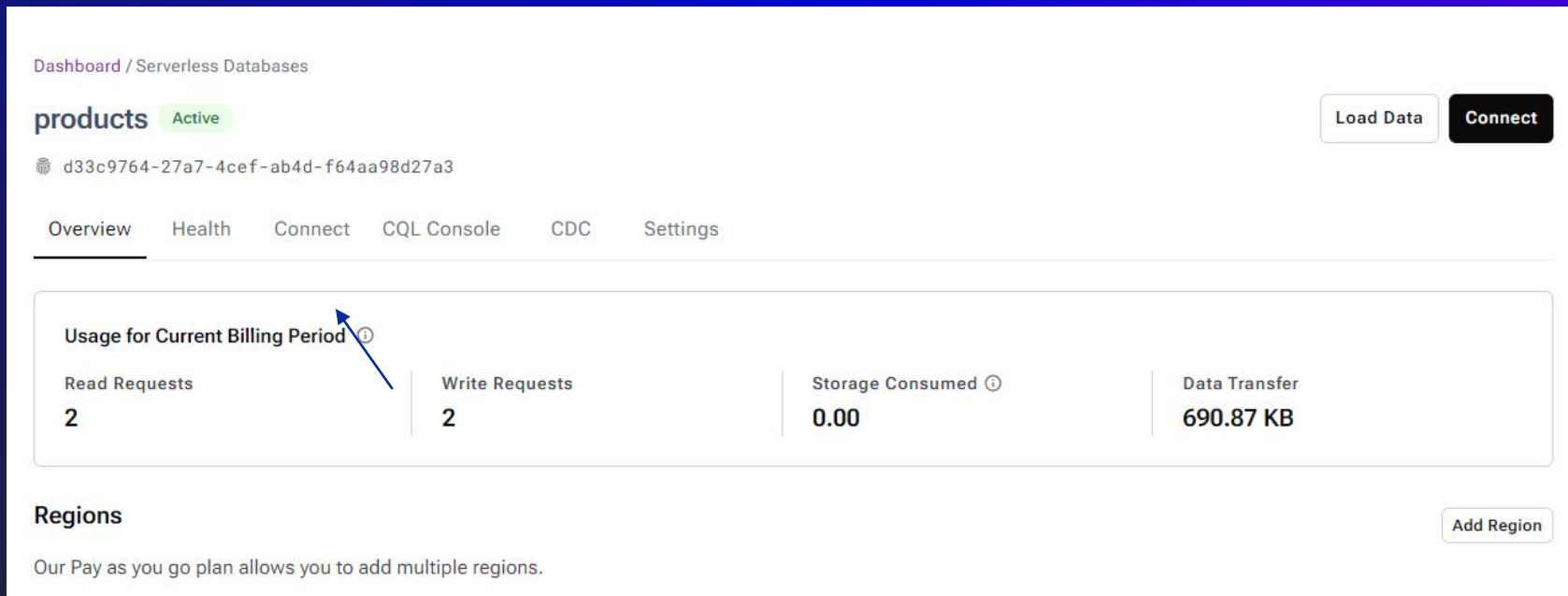
Read Requests	Write Requests	Storage Consumed ⓘ	Data Transfer
2	2	0.00	690.87 KB

Regions Add Region

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

Connect DataStax with Python

Step 2: Download the application token and Secure Connect Bundle.



The screenshot shows the DataStax Cloud dashboard for a serverless database instance. The breadcrumb navigation at the top left reads 'Dashboard / Serverless Databases'. The instance name 'products' is displayed with an 'Active' status tag. A unique identifier 'd33c9764-27a7-4cef-ab4d-f64aa98d27a3' is shown below the name. On the top right, there are 'Load Data' and 'Connect' buttons. A horizontal menu below the instance name includes 'Overview' (which is underlined), 'Health', 'Connect', 'CQL Console', 'CDC', and 'Settings'. The main content area features a 'Usage for Current Billing Period' section with a blue arrow pointing to an information icon. This section contains four metrics: Read Requests (2), Write Requests (2), Storage Consumed (0.00), and Data Transfer (690.87 KB). Below this is a 'Regions' section with an 'Add Region' button and a note stating 'Our Pay as you go plan allows you to add multiple regions.'

Dashboard / Serverless Databases

products Active

d33c9764-27a7-4cef-ab4d-f64aa98d27a3

Load Data Connect

Overview Health Connect CQL Console CDC Settings

Usage for Current Billing Period ⓘ

Read Requests	Write Requests	Storage Consumed ⓘ	Data Transfer
2	2	0.00	690.87 KB

Regions

Add Region

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

Now let's go to our Editor....

Connect DataStax with Python

Step 3: Install Cassandra's driver.



```
pip install cassandra-driver
```


Connect DataStax with Python

Step 4: Verify that the DataStax Python driver installed successfully:



```
python -c 'import cassandra; print (cassandra.__version__)'
```

Connect DataStax with Python

Step 5: make configurations, replace file name with your file



```
from cassandra.cluster import Cluster
from cassandra.auth import PlainTextAuthProvider
import json

# This secure connect bundle is autogenerated when you download your SCB,
# if yours is different update the file name below
cloud_config= {
    'secure_connect_bundle': 'secure-connect-products.zip'
}
```

Connect DataStax with Python

Step 5: make configurations and connect to the database.



```
# This token json file is autogenerated when you donwload your token,  
# if yours is different update the file name below  
with open("          Token file name          .json") as f:  
    secrets = json.load(f)  
  
CLIENT_ID = secrets["clientId"]  
CLIENT_SECRET = secrets["secret"]  
  
auth_provider = PlainTextAuthProvider(CLIENT_ID, CLIENT_SECRET)  
cluster = Cluster(cloud=cloud_config, auth_provider=auth_provider)  
session = cluster.connect()
```

Connect DataStax with Python

Step 6: Check the connection.



```
row = session.execute("select release_version from system.local").one()
if row:
    print(row[0])
else:
    print("An error occurred.")
```

Connect DataStax with Python

Step 7: Close connection.



```
session.shutdown()  
cluster.shutdown()
```


“

Let's practice on real use Case



Use case : Analyzing Product Data

Create database.



```
# Create Product Table
create_table_query = '''
    CREATE TABLE IF NOT EXISTS supermarket_products.products (
        product_id UUID PRIMARY KEY,
        name TEXT,
        price DECIMAL,
        category TEXT
    )
...

session.execute(create_table_query)
```

Use case : Analyzing Product Data

Insert data.

```
import uuid

data_to_insert = [
    (uuid.UUID("1e34567a-6dcd-11ec-8d3d-0242ac130003"), "Product A", 25.99, "Electronics"),
    (uuid.UUID("2f45678b-6dcd-11ec-8d3d-0242ac130003"), "Product B", 39.99, "Clothing")
]

for values in data_to_insert:
    session.execute(
        """
        INSERT INTO supermarket_products.products (product_id, name, price, category)
        VALUES (%s, %s, %s, %s)
        """,
        values
    )
```


Use case : Analyzing Product Data

Do your analysis.



```
import pandas as pd

query = 'SELECT * FROM supermarket_products.products'
result = session.execute(query)

data_list = []
for row in result:
    data_list.append(row)

# Create a pandas DataFrame from the retrieved data
df = pd.DataFrame(data_list, columns=['product_id', 'category', 'name', 'price' ])

# Perform analysis on the DataFrame
average_price_by_category = df.groupby('category')['price'].mean()
```

Thank
you!

