**NO SQL(Not a query language)**

NoSQL is an approach to database management that can accommodate data models like KEY-VALUE, DOCUMENT, TABULAR, GRAPH format.

NoSQL is NON-RELATIONAL, DISTRIBUTED, PARTITION TOLERANT and SCALABLE.

**DISTRIBUTED:** Data is stored in server in different geographical locations and according accessed to near place.

**PARTITION TOLERANT:** Ability to access data even when some machines go down. Like having same 3-4 copy of data.

**HORIZONTAL SCALING:** Horizontal scaling simply adds more machine resources to your existing machine infrastructure. **[NoSQL]**

**VERTICAL SCALING:** Vertical scaling adds power to your existing machine infrastructure by increasing power from CPU or RAM to existing machines.**[SQL]**

WHY NoSQL - a. Application development productivity.

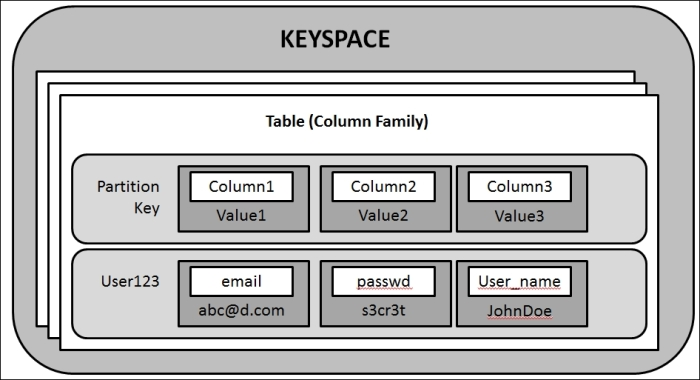
b. Large scale data.

NoSQL Database - **DataStax Astra, Aapche Cassandra**

DataStax *Astra* : Astra is the easiest cloud-native database built on Apache Cassandra.

Apache Cassandra is a free and open-source, distributed, wide-column store, NoSQL database management system designed to handle large amounts of data across many commodity servers, providing high availability with no single point of failure.

**Tabular Database: Organize data in rows and column(CQL used)**



A keyspace in a NoSQL data store is an object that holds together all column families of a design(like a table).

Partition key is basically like a row number.

**To check the key spaces :** describe keyspaces.

**To use keyspace :** use name\_of\_keyspace(tabular)

**To create a new database.**

CREATE TABLE IF NOT EXISTS books (

... bookid uuid,

... author text,

... title text,

... year int,

... categories set <test>,

... added timestamp,

... PRIMARY KEY (bookid)

… );

**A primary key in Cassandra consists of one or more partition keys and zero or more clustering key components. The order of these components always puts the partition key first and then the clustering key.**

**The primary goal of a partition key is to distribute the data evenly across a cluster and query the data efficiently. Using this primary key, we identify the records using a unique key.**

**Clustering Key - clustering is a storage engine process of sorting the data within a partition and is based on the columns defined as the clustering keys**.---> Primary key((partition key),clusteringkey1, clusteringkey2)

### **Composite Partition Key : If we need to combine more than one column value to form a single partition key, we use a composite partition key.**

**To insert data into tables:** INSERT INTO taable\_name(Column1, Colum2)

…VALUES(uuid(), ‘Text’, ‘test’ );

Create a tabular database with partition key and clustering key.

CREATE TABLE IF NOT EXISTS restaurant\_by\_country (

…country text,

... name text,

... cuisine text,

... url text,

... PRIMARY KEY ((country), name, url))

WITH CLUSTERING ORDER BY (name DESC, url ASC);

**Document Database()**

API- Application Programming Interface - They help in talking between technologies, they connect one another technology.

Token - AstraCS:LagFrOkRPlFfPXGKcfnSRabv:3355dfb50b091de9f99869e9f5c100b0babf71cdc5f710d6a56e68ea5d279525

In document database we're going to work with collections(simple key value pair like json objects)

We used **swagger UI** to create a document. You get the document ID . With document i we create an url. With url we get document object using hopscotch.

We also used to **hoppscotch** to get the object(hoppscotch is alternative to postman)

**KEY-VALUE type Database(CQL used)**

We are using GraphQL api here.

To create a table in GraphQl

mutation{

createTable(

keyspaceName:"keyvalue",

tableName: "shop\_inventory",

partitionKeys:[{name:"key",type:{basic:TEXT}}],

values:[{name:"value", type:{basic:TEXT}}]

)

}

Once the table is created, we can insert key value pair data.

To insert the key value data

mutation {

insertshop\_inventory (

value:{key:"3dr54",value:"beans"}

){

value {

key,

value

}

}

}

To query the table to see the data

query{

shop\_inventory {

values {

key,

value

}

}

}

To delete any data based on key

mutation {

deleteshop\_inventory (value: {key: "3dr54"}){

value {

key,

value

}

}

}

**GRAPH type Database(3 components)**

In traditional databases, the relationships between data is not established. But in the case of Graph Database, the relationships between data are prioritized.

Here the data is stored in different nodes and nodes are connected through edges.

Here we are using **DataStax Enterprise Graph and Docker**

In docker we can create a graph - docker network create graph(command).

**Nodes:** represent the objects or instances. They are equivalent to a row in database. The node basically acts as a vertex in a graph. The nodes are grouped by applying a label to each member.

**Relationships:** They are basically the edges in the graph. They have a specific direction, type and form patterns of the data. They basically establish relationship between nodes.

**Properties:** They are the information associated with the nodes.

PROJECT -1

AstraCS:orlFuphMbQtMnegCqhWPTUDF:332a819398ada93f16628076a222974893a59c08a924b5b60ed581eac2ca10fb

2b885eca-90cb-4b32-8214-b7873f2f0bc3

41240265-4654-4f80-bf73-700eb12feef5

031d6507-92eb-46ba-8dc9-f750d8d97b22

d31163fa-8a1b-4c4b-8c44-30048022385f