

Introduction to CASSANDRA

Sophia Sandhu



Agenda

- What is Cassandra
- Cassandra History
- Cassandra Architecture
- How Cassandra Works
 - Gossip Protocol
 - Read and Write request
- Cassandra Modules
- Data Organization
- Cassandra Lab



What is Cassandra



- Apache open source framework
 - Cassandra is a distributed database management system designed for handling a high volume of structured data across commodity servers
 - NoSQL database: Wide Column Store
 - Its popularity is based on its speed and SQL-like query language people comfortable working with relational database systems.
 - CQL (Cassandra Query Language): Similar to SQL
 - No Fixed Schema
 - Data is replicated for recover/failover
-

What is Cassandra



- The Cassandra design team took the best bits from Amazon's Dynamo paper on key-value store design and Google's Bigtable paper on wide column store (also called extensible record store) design.
 - Cassandra, therefore, provides high-speed key access to data while also providing flexible columns and a schema-free, join-free, wide column store.
 - Developers who have used the Structured Query Language (SQL) in relational database management systems should find the Cassandra Query Language (CQL) familiar.
-

Cassandra History



- Cassandra was first developed at **Facebook** for inbox search.
 - Facebook open sourced it in July 2008.
 - Apache incubator accepted Cassandra in March 2009.
 - Cassandra is a top level project of Apache since February 2010.
 - The latest version of Apache Cassandra is 3.2.1.
-

Cassandra & Market



- Messaging
 - Companies that provides Mobile phones and messaging services.
- Internet of things Application
 - Applications where data is coming at very high speed from different devices or sensors.
- Product Catalogs and retail apps
 - Cassandra is used by many retailers for durable shopping cart protection and fast product catalog input and output.
- Social Media Analytics and recommendation engine.
- Some of its users:



Column based: Index By Row, Column Name and TimeStamp

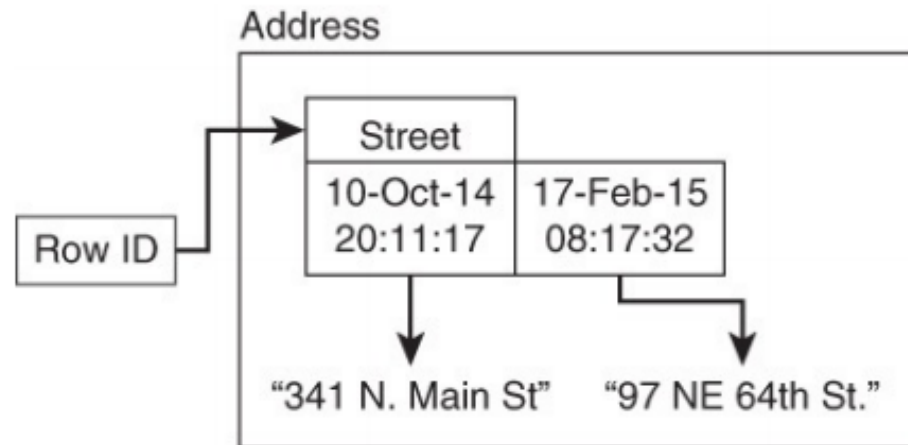


Figure 9.2 Data values are indexed by row identifier, column name, and time stamp. Multiple versions of a column value can exist. The latest version is returned by default when the column value is queried.

When a new value is written to a column based database, the old value is not overwritten. Instead, a new value is added along with a time stamp. The time stamp allows applications to determine the latest version of a column value.

Column based: Reading and Writing Atomic Rows

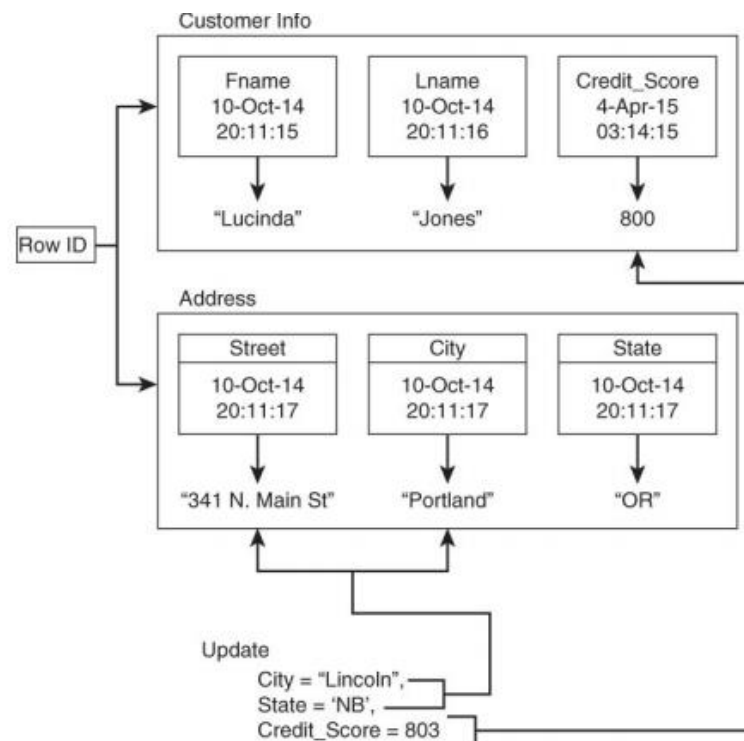


Figure 9.4 Read and write operations are atomic. All columns are read or written or none are.

For example, if a customer moves from Portland, Oregon, to Lincoln, Nebraska, and you update the customer's address, you would never find a case in which the city changes from Portland to Lincoln but the state does not change from Oregon to Nebraska.

Key Value vs Column Family

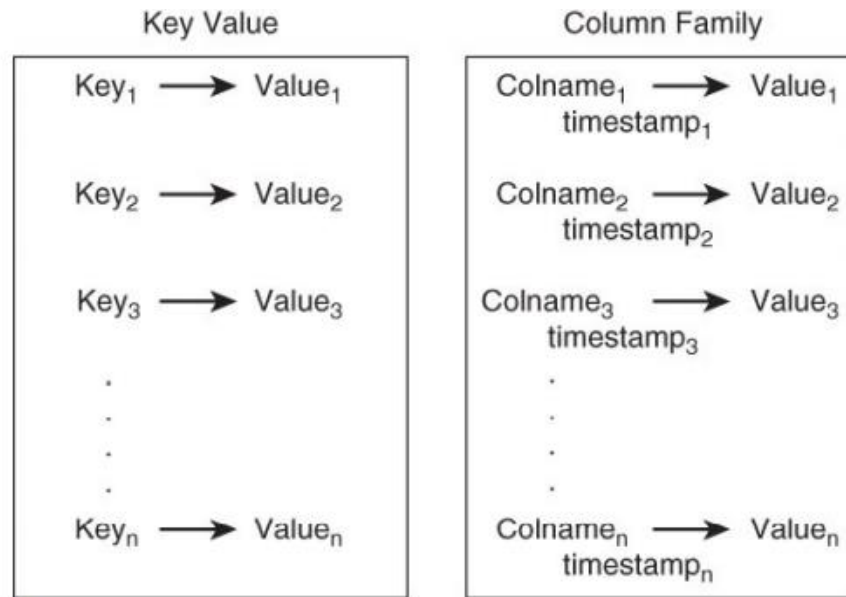


Figure 9.5 Keyspaces in key-value databases are analogous to column families in the way they maintain collections of attributes. Indexing, however, is different between the two database types.

Unlike key-value databases, the values in columns are indexed by a row identifier as well as by a column name (and time stamp).

For example, if a customer moves from Portland, Oregon, to Lincoln, Nebraska, and you update the customer's address, you would never find a case in which the city changes from Portland to Lincoln but the state does not change from Oregon to Nebraska.

Cassandra : Column Family Database

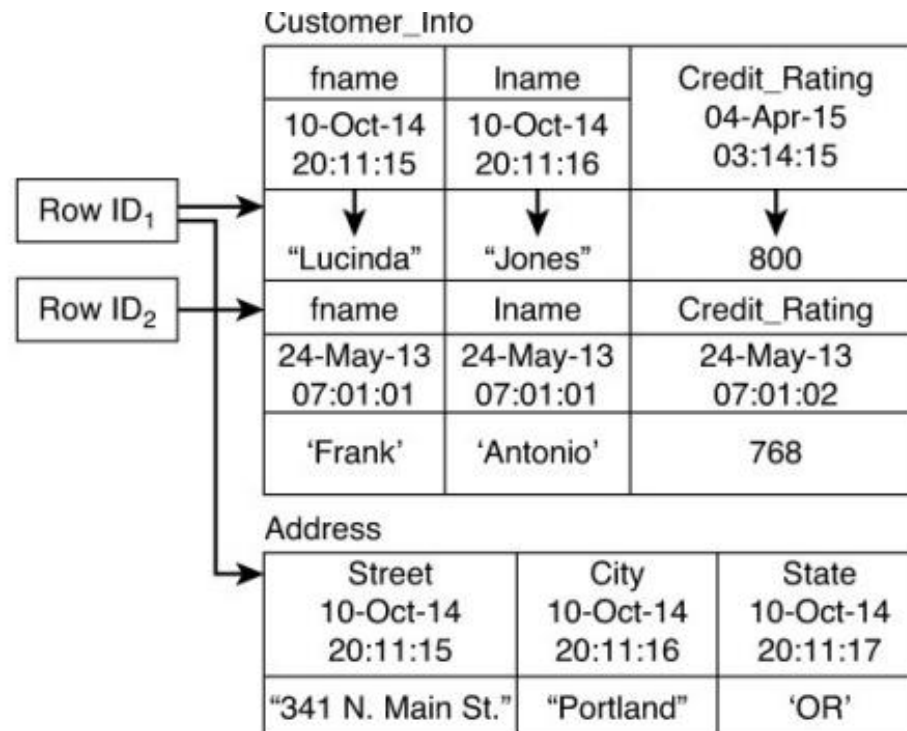


Figure 9.6 *Column family* databases, like document databases, may have values for some or all columns. Columns can be added programmatically as needed in both document and *column family* databases.

For example, if a customer moves from Portland, Oregon, to Lincoln, Nebraska, and you update the customer's address, you would never find a case in which the city changes from Portland to Lincoln but the state does not change from Oregon to Nebraska.

Cassandra Architecture: Peer-to-Peer



- Apache Cassandra, is designed for high availability, scalability, and consistency.
- Cassandra uses a peer-to-peer model.
- All Cassandra nodes run the same software. They may, however, serve different functions for the cluster.

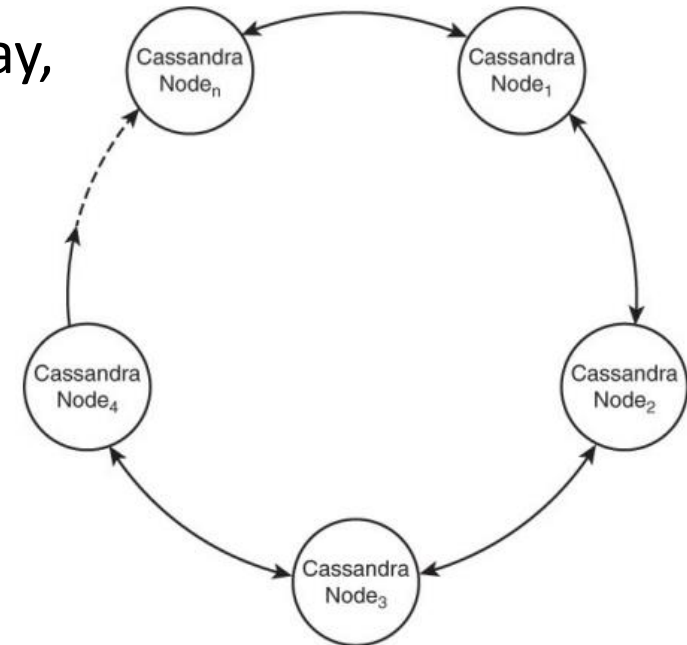


Figure 9.10 Cassandra uses a peer-to-peer architecture in which all nodes are the same.

Peer-to-Peer Advantage



- No node can be a single point of failure.
- Scaling up and down is fairly straightforward:
 - Servers are added or removed from the cluster.
 - When a node is removed, servers hosting replicas of data from the removed node respond to read and write requests.
- The servers in the cluster are responsible for managing a number of operations that a master server would handle, including the following:
 - Sharing information about the state of servers in the cluster
 - Ensuring nodes have the latest version of data
 - Ensuring write data is stored when the server that should receive the write is unavailable.

Peer-to-Peer Advantage



- The hardware failure can occur at any time. Any node can be down.
 - In case of failure data stored in another node can be used.
 - Hence, Cassandra is designed with its distributed architecture.
 - All the nodes exchange information with each other using Gossip protocol.
 - Gossip is a protocol in Cassandra by which nodes can communicate with each other.
-



- Each server can simply ping or request update information from each of the other servers.
 - But, if N is the number of servers, then $N \times (N-1)$ is the number of messages needed to update all servers with information about all other servers which can quickly increase the volume of traffic on the network and the amount of time each server has to dedicate to communicating with other servers.
-

Gossip Protocol



- Cassandra's gossip Protocol :
 - A node in the cluster initiates a gossip session with a randomly selected node.
 - The initiating node sends a starter message (known as a Gossip-DigestSyn message) to a target node.
 - The target node replies with an acknowledgment (known as a GossipDigestAck message).
 - After receiving the acknowledgment from the target node, the initiating node sends a final acknowledgment (a GossipDigestAck2 message) to the target node.
- In the course of this message exchange, each server is updated about the state of servers as known by the other server.
- In addition, version information about each server's state is exchanged. With this additional piece of data, each party in the exchange can determine which of the two has the most up-to-date data about each of the servers discussed.

"Eben Hewitt. Cassandra: The Definitive Guide. Sebastopol, CA: O'Reilly Media, Inc., 2010."



Apache

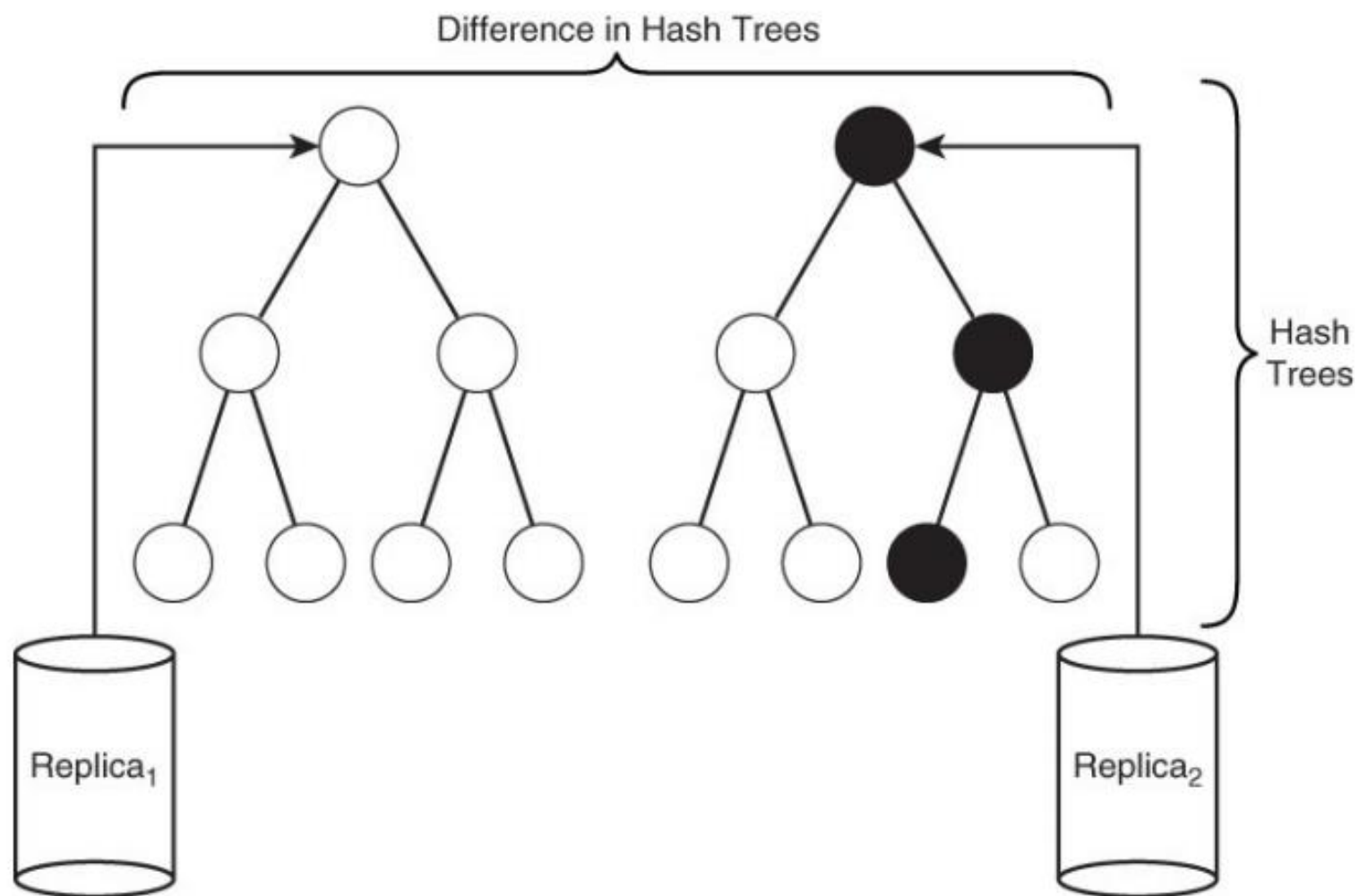


Figure 9.12 *Cassandra regularly compares replicas of data to ensure they are up to date. Hashes are used to make this a relatively fast operation.*

CASSANDRA READ REQUEST



- A client device needs to request data from the database. It issues a read operation and Node 1 receives the request.
- Node 1 uses the row key in the read request and looks up information about which node should process the request.
- It determines that Node 2 is responsible for data associated with that row key and passes the information on to it.
- Node 2 performs the read operation and sends the data back to Node 1.
- Node 1, in turn, passes the read results back to the client.

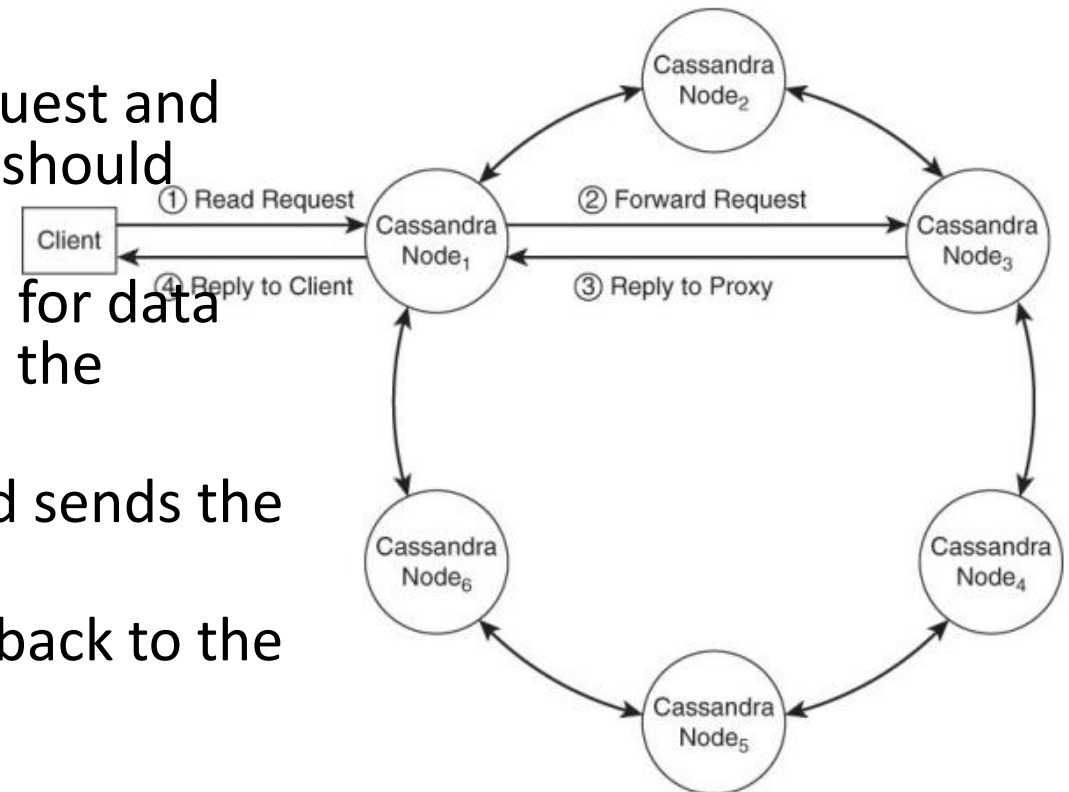


Figure 9.13 Any node in a Cassandra cluster can handle a client request. All nodes have information about the state of the cluster and can act as a proxy for a client, forwarding the request to the appropriate node in the cluster.

CASSANDRA WRITE REQUEST



- The client sends a request to Node 1.
- Node 1 queries its local copy of metadata about the cluster and determines that Node 3 should process this request.
- Node 1, however, knows that Node 3 is unavailable because the gossip protocol informed Node 1 about the status of Node 3 a few seconds ago.
- Node 1 initiates a hinted handoff.
 - A hinted handoff entails storing information about the write operation on a proxy node and periodically checking the status of the unavailable node.
- When that node becomes available again, the node with the write information sends, or “hands off,” the write request to the recently recovered node.

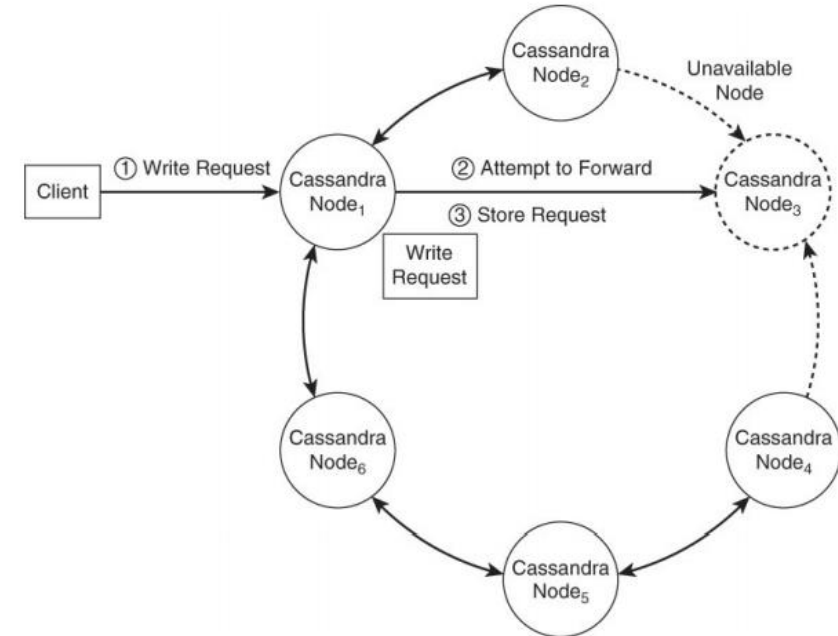
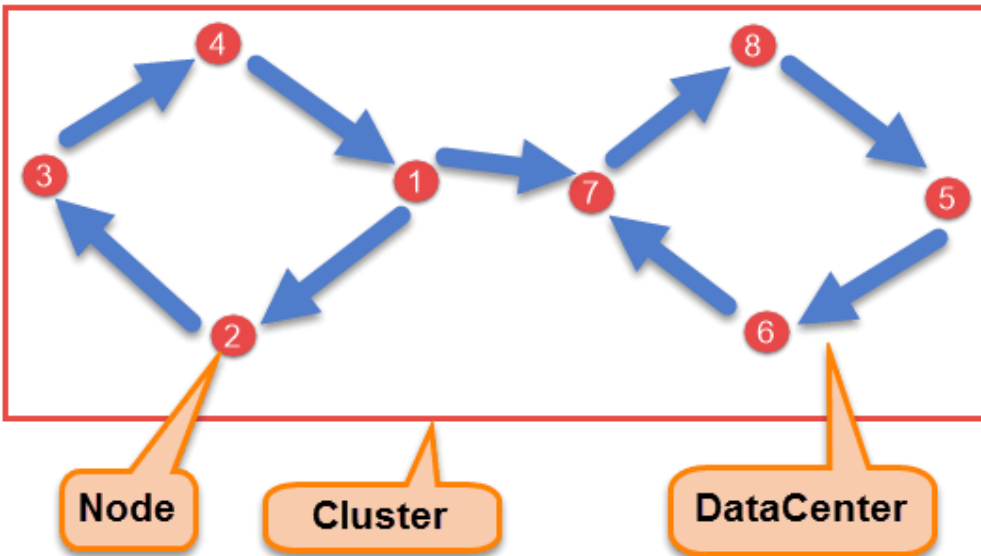


Figure 9.14 If a node is unavailable, then other nodes can receive write requests on its behalf and forward them to the intended node when it becomes available.

CASSANDRA MODULES



- **Node:** is the place where data is stored.
 - It is the basic component of Cassandra.
- **Data Center:** A collection of nodes are called data center.
 - Many nodes are categorized as a data center.
- **Cluster:** The cluster is the collection of many 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** – It is a special kind of cache. Bloom filters are accessed after every query.

Cassandra Modules : CLUSTER



- A cluster is a set of servers configured to function together.
- Servers sometimes have differentiated functions and sometimes they do not.
- Cassandra uses a single type of node.
- Each node shares similar responsibilities, including
 - Accepting read and write requests
 - Forwarding read and write requests to servers able to fulfill the requests
 - Gathering and sharing information about the state of servers in the clusters
 - Helping compensate for failed nodes by storing write requests for the failed node until it is restored

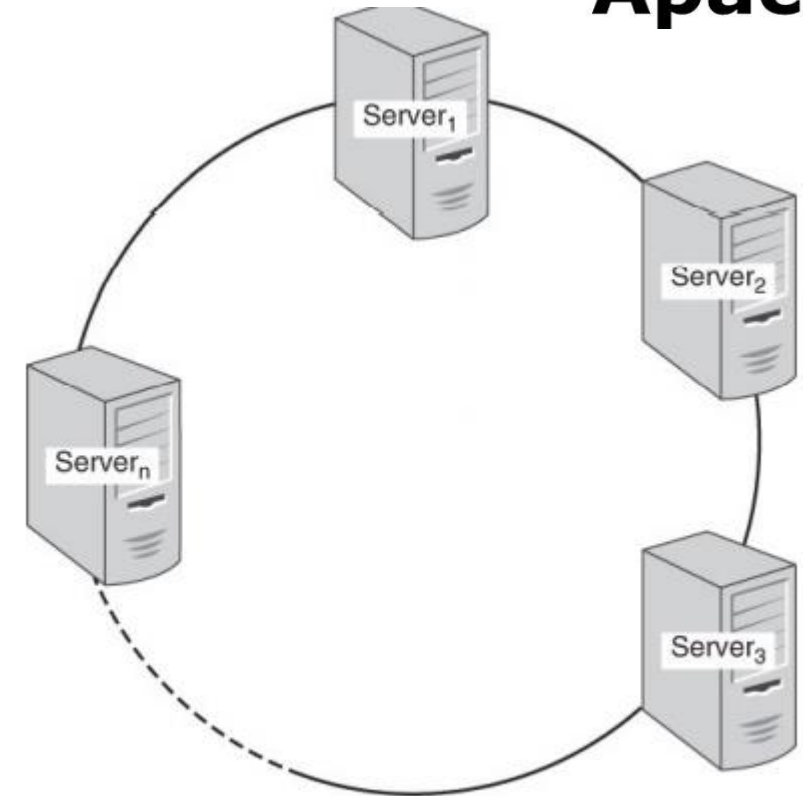


Figure 10.5 Clusters are collections of servers functioning together to implement a distributed service, such as a **column family** database.

Cassandra Modules :Partition



- A partition is a logical subset of a database.
- Partitions are usually used to store a set of data based on some attribute of the data .
- Each node or server within a column family cluster maintains one or more partition.
- When a client application requests data, the request is routed to a server with the partition containing the requested data.

Cassandra Modules :COMMIT LOG



- Instead of writing data immediately to its partition and disk block, column family databases can employ commit logs.
- These are append only files that always write data to the end of the file.
- In the event of a failure, the database management system reads the commit log on recovery.
- Any entries in the commit log that have not been saved to partitions are then written to appropriate partitions.

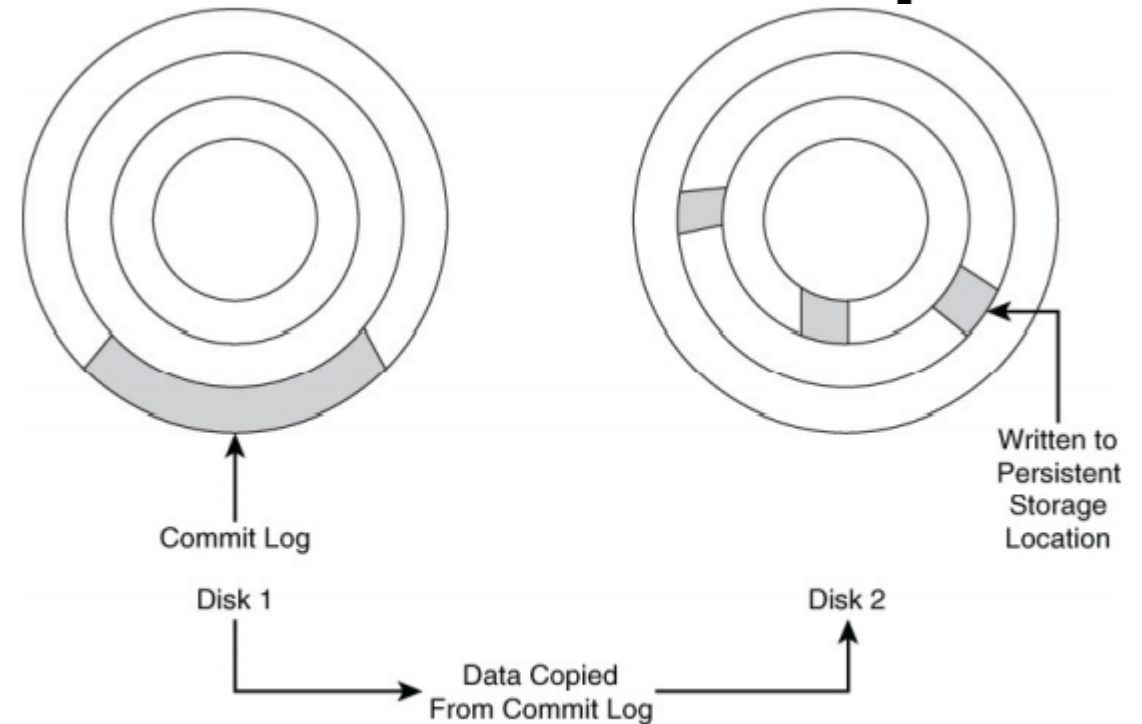


Figure 10.8 After a database failure, the recovery process reads the commit log and writes entries to partitions. The database remains unavailable to users until all commit log entries are written to partitions.

Cassandra Modules :BLOOM FILTER



- Anything that reduces the number of blocks read from disk or solid state device can help improve performance. Applying Bloom filters is one such technique.
- Bloom filters help reduce the number of read operations by avoiding reading partitions or other data structures that definitely do not contain a sought-after piece of data.

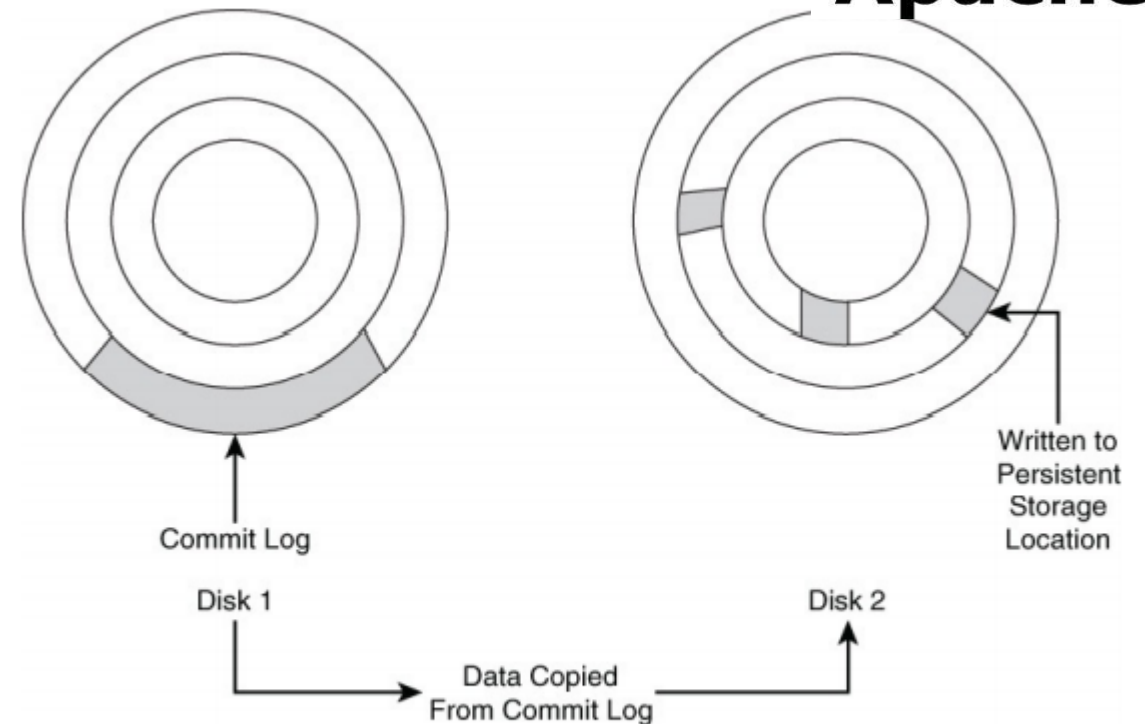


Figure 10.8 After a database failure, the recovery process reads the commit log and writes entries to partitions. The database remains unavailable to users until all commit log entries are written to partitions.

How Réplication Works?



- Data is placed on different machines with more than one replication factor that provides high availability and no single point of failure.
- Cassandra places replicas of data on different nodes based on these two factors.
 - Where to place next replica is determined by the **Replication Strategy**.
 - The total number of replicas placed on different nodes is determined by the **Replication Factor**.



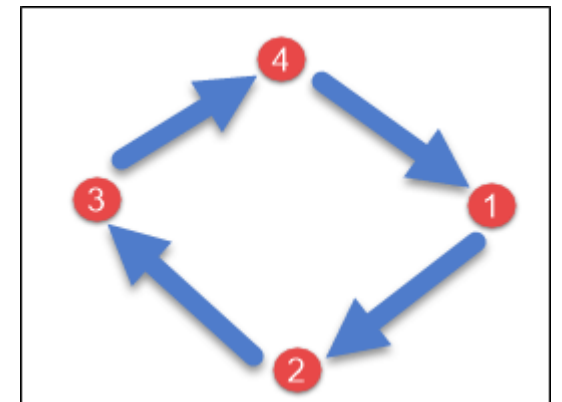
For ensuring there is no single point of failure, **replication factor must be three**.

How Replication Works?



- **Simple Strategy**

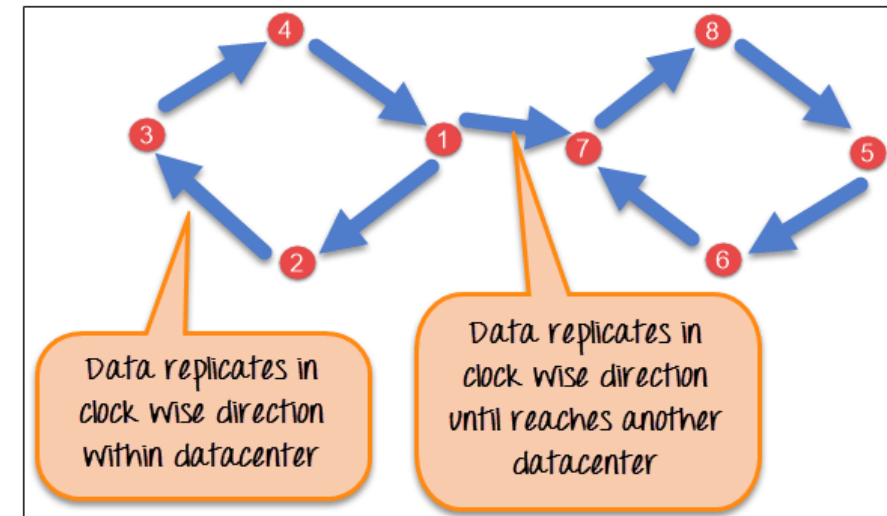
- Simple Strategy is used when you have just one data center.
- In the simplest case, the server for the first replica is determined by hash function.
- Additional replicas are placed according to the relative position of other servers.
 - For example, all nodes in Cassandra are in a logical ring.
 - Once the first replica is placed, additional replicas are stored on successive nodes in the ring in the clockwise direction.



How Replication Works?



- **Network Topology Strategy**
 - Used when you have more than two data centers.
- Replicas are set for each data center separately.
- Network Topology Strategy places replicas in the clockwise direction in the ring until reaches the first node in another rack.
- This strategy tries to place replicas on different racks in the same data center.
- This is due to the reason that sometimes failure or problem can occur in the rack.
 - Then replicas on other nodes can provide data.



How Replication Works?

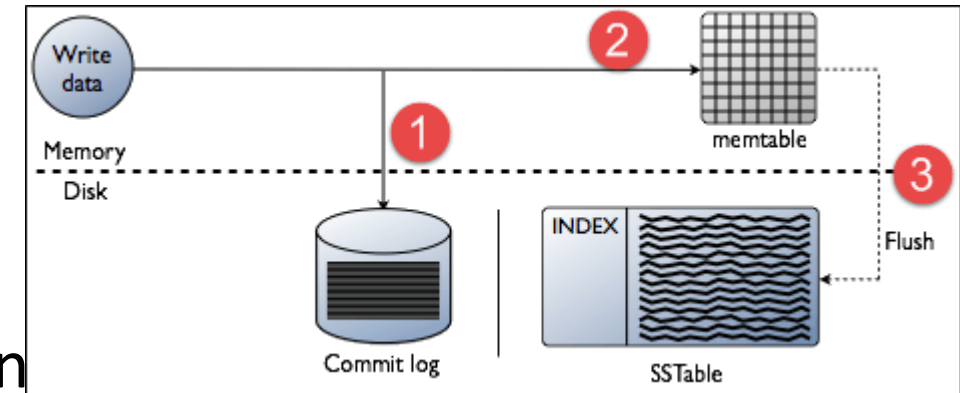


- One or more of the nodes in a cluster act as replicas for a given piece of data.
- If it is detected that some of the nodes responded with an out-of-date value, Cassandra will return the most recent value to the client.
- After returning the most recent value, Cassandra performs a **read repair** in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

Write Operations



- When write request comes to the node, first of all, it logs in the commit log.
- Then Cassandra writes the data in the mem-table.
- Data written in the mem-table on each write request also writes in commit log separately.
- Mem-table is a temporarily stored data in the memory while Commit log logs the transaction records for back up purposes.
- When mem-table is full, data is flushed to the SSTable data file.



Write Operations



- The coordinator sends a write request to replicas.
 - If all the replicas are up, they will receive write request regardless of their consistency level.
- **Consistency level** determines how many nodes will respond back with the success acknowledgment.
- The node will respond back with the success acknowledgment if data is written successfully to the commit log and **memTable**.
- For example, in a single data center with replication factor equals to three, three replicas will receive write request. If consistency level is one, only one replica will respond back with the success acknowledgment, and the remaining two will remain dormant.

READ Operations



- There are three types of read requests that a coordinator sends to replicas.
 - Direct request
 - Digest request
 - Read repair request
- The coordinator sends direct request to one of the replicas.
- A digest request is sent to the number of replicas specified by the consistency level and checks whether the returned data is an updated data.
- If any node gives out of date value, a background read repair request will update that data.
 - This process is called read repair mechanism.
 - 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.

KEYSPACE



- A keyspace is the top-level data structure in a column family database.
- All other data structures you would find in a database designer are contained within a keyspace.
- A keyspace is analogous to a schema in a relational database. Typically, you create one keyspace for each of your applications.

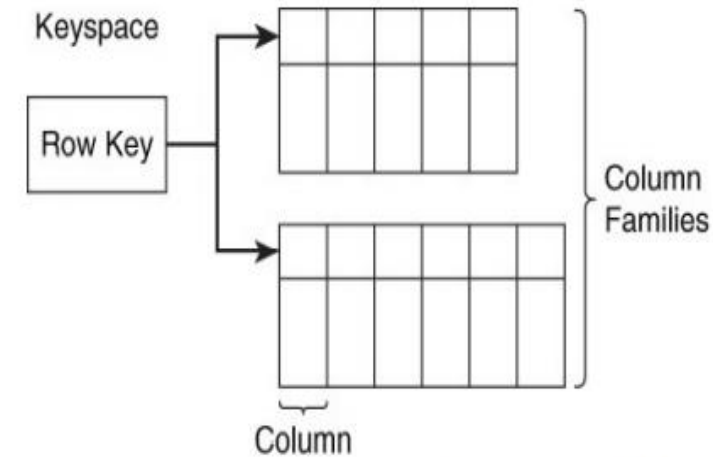


Figure 10.1 A keyspace is a top-level container that logically holds column families, row keys, and related data structures. Typically, there is one keyspace per application.

Row Key



- A row key uniquely identifies a row family. It serves some of the same primary key in a relational database
- In addition to uniquely identifying rows, row keys are also used to partition and order data
- In Cassandra, rows are stored in a table determined by an object known as a partitioner
 - Cassandra uses a random partitioner by default
 - The partitioner randomly distributes rows across nodes.
 - Cassandra also provides an order-preserving partitioner, which can provide lexicographic ordering.

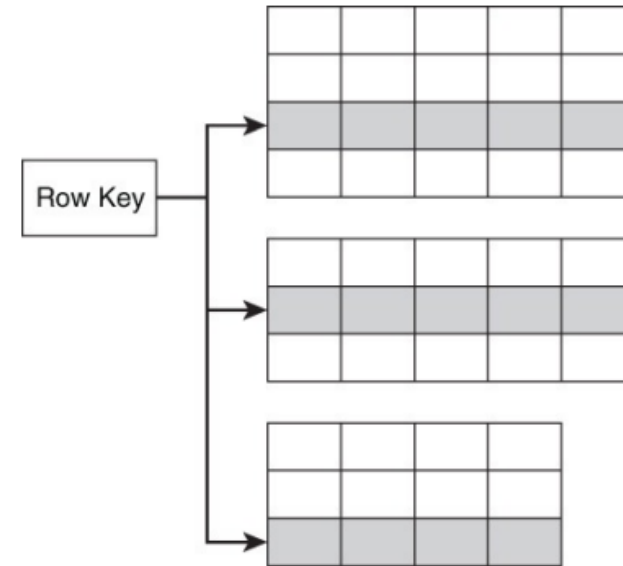
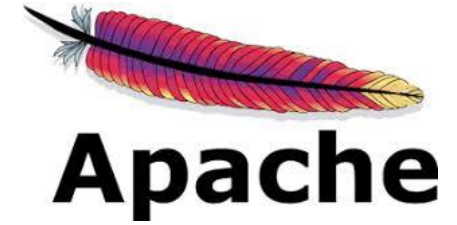


Figure 10.2 A row key uniquely identifies a row and has a role in determining the order in which data is stored.

Column



- A column is the data structure for storing a single value in a database .
- Columns are members of column family databases.
 - Database designers define column families when they create a database.
 - However, developers can add columns any time after that.
 - Just as you can insert data into a relational table, you can create new columns in column family databases.
- Columns have three parts:
 - A column name
 - A time stamp or other version stamp
 - A value

Column Families



- Column families are collections of related columns.
- Columns that are frequently used together should be grouped into the same column family.
 - For example, a customer's address information, such as street, city, state, and zip code, should be grouped together in a single column family.

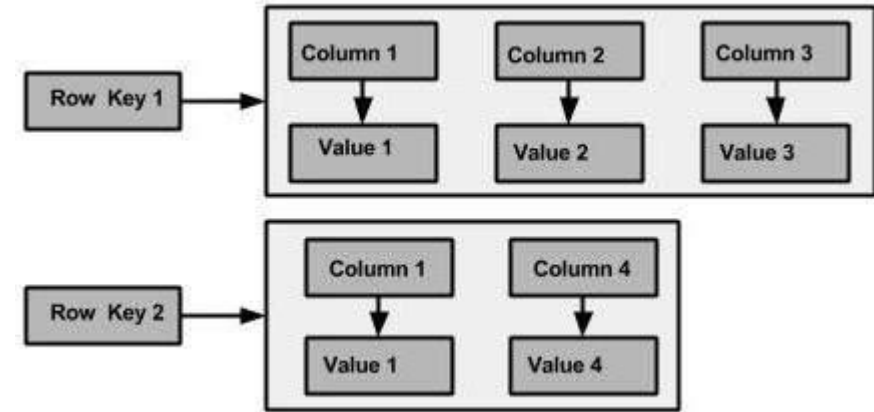
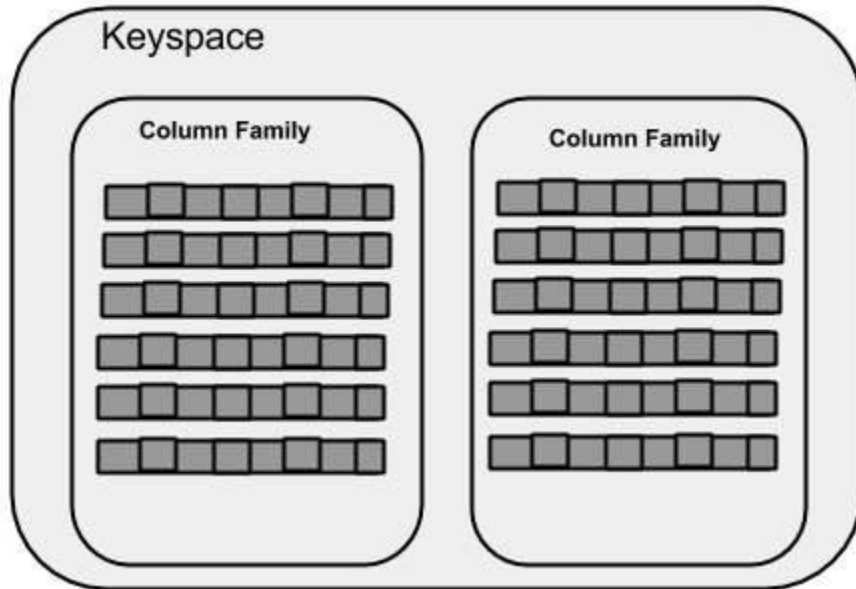
Street	City	State	Province	Zip	Postal Code	Country
178 Main St.	Boise	ID		83701		U.S.
89 Woodridge	Baltimore	MD		21218		U.S.
293 Archer St.	Ottawa		ON		K1A 2C5	Canada
8713 Alberta DR	Vancouver		BC		VSK 0A1	Canada

Figure 10.4 Column families are analogous to relational database tables: They store multiple rows and multiple columns. There are, however, significant differences between the two, including varying columns by row.

KEYSPACE & Column Family



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



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

Cassandra Data Model



Cassandra Data Model	Relational Data Model
Keyspace	Database
Column Family	Table
Partition Key	Primary Key
Column Name/Key	Column Name
Column Value	Column Value

Cassandra Data Model

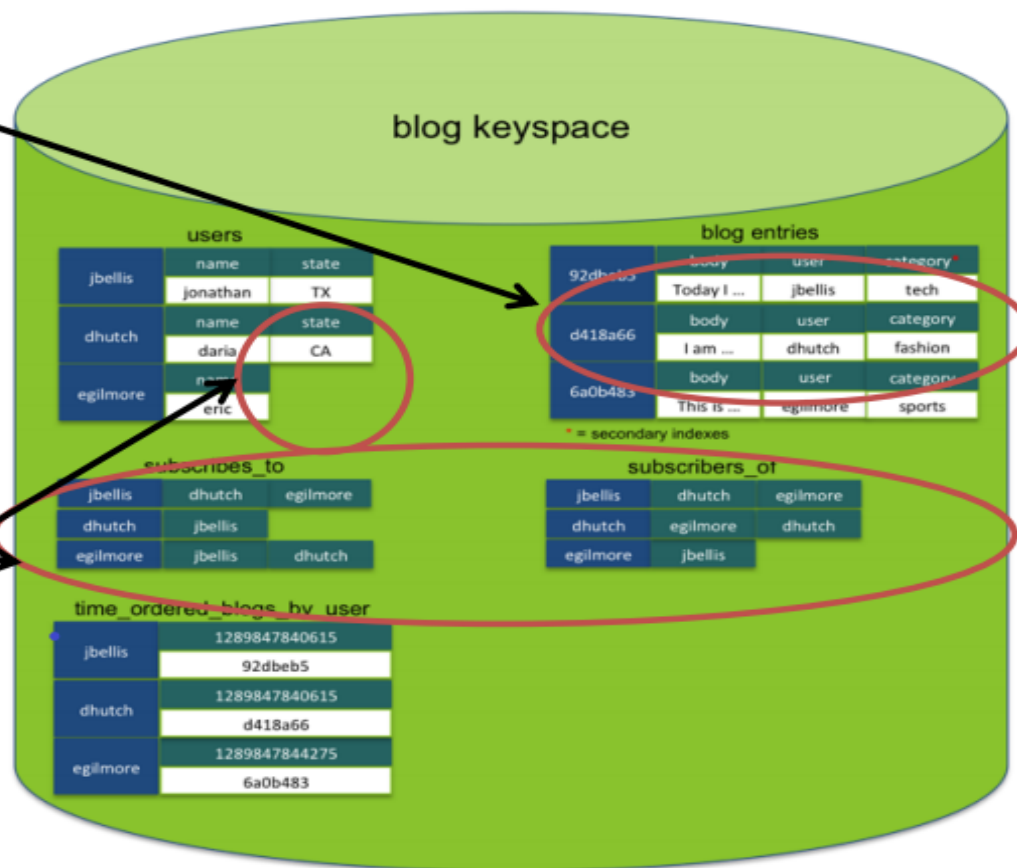


Column Families:

- Like SQL tables
- but may be unstructured (client-specified)
- Can have index tables

Hence “column-oriented databases” / “NoSQL”

- No schemas
- Some columns missing from some entries
- “Not Only SQL”
- Supports get(key) and put(key, value) operations
- Often write-heavy workloads



Data Organization



- Cassandra organizes data into partitions.
 - Each partition consists of multiple columns.
 - Partitions are stored on a node.
 - Nodes are generally part of a cluster where each node is responsible for a fraction of the partitions.
 - When inserting records, Cassandra will hash the value of the inserted data's partition key;
 - Cassandra uses this hash value to determine which node is responsible for storing the data.
-

Data model goals



- Spread data evenly around the cluster.
 - Partitions are distributed around the cluster based on a hash of the partition key.
 - To distribute work across nodes, it's desirable for every node in the cluster to have roughly the same amount of data.
 - Minimize the number of partitions read.
 - Partitions are groups of columns that share the same partition key.
 - Since each partition may reside on a different node, the query coordinator will generally need to issue separate commands to separate nodes for each partition we query.
 - Satisfy a query by reading a single partition. This means we will use roughly one table per query. Supporting multiple query patterns usually means we need more than one table. Data duplication is encouraged.
-

Data model goals



- Satisfy a query by reading a single partition.
 - This means we will use roughly one table per query.
 - Supporting multiple query patterns usually means we need more than one table.
 - Data duplication is encouraged.
-

Cassandra Column Family



- The Column families are established with the CREATE TABLE command.
- Column families are represented in Cassandra as a map of sorted maps.
- The partition key acts as the lookup value;
- The sorted map consists of column keys and their associated values.

```
Map<PartitionKey, SortedMap<ColumnKey, ColumnValue>>
```

PARTITION KEYS



- The partition key is responsible for distributing data among nodes.
 - A partition key is the same as the primary key when the primary key consists of a single column.
 - Partition keys belong to a node.
 - Cassandra is organized into a cluster of nodes, with each node having an equal part of the partition key hashes.
-

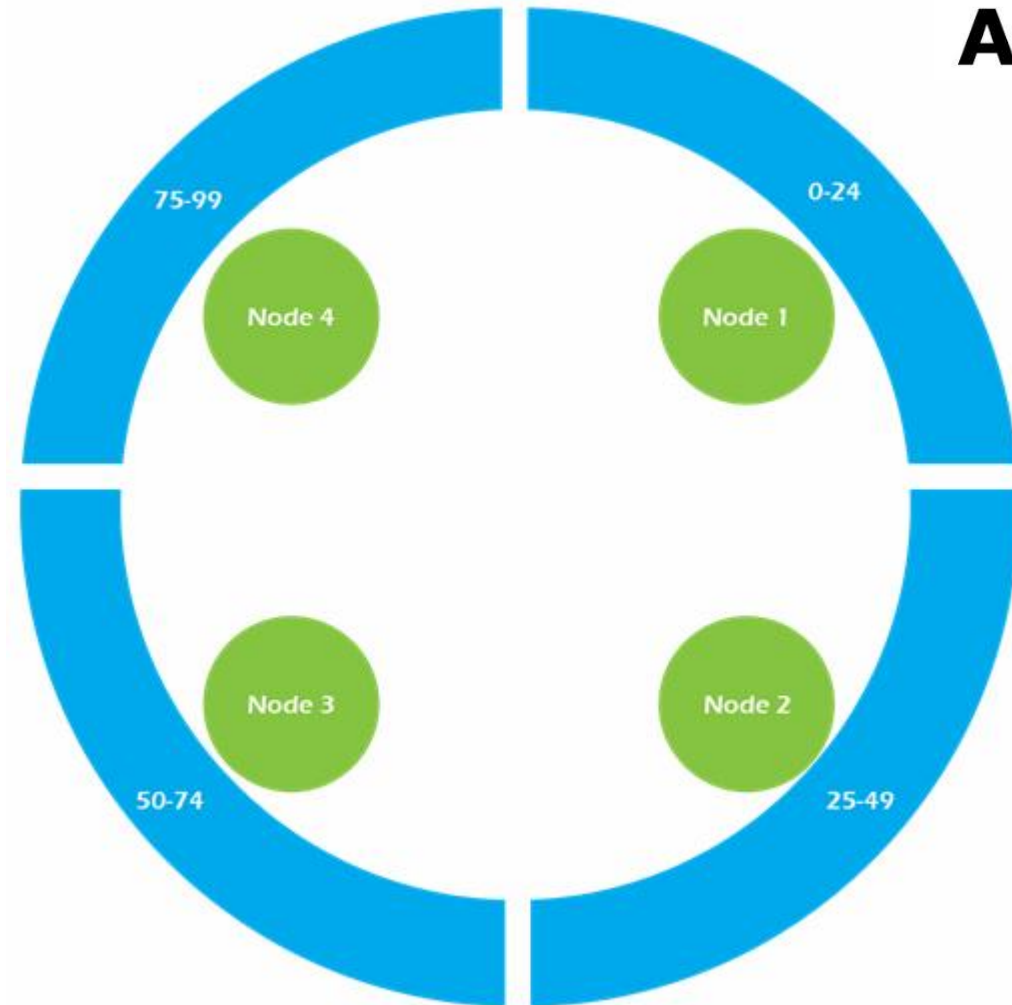
CASSANDRA PARTITION KEYS



Partition Key	Column name	...	Column name
	Column value	...	Column value
	Timestamp	...	Timestamp
	Time to live	...	Time to live

CASSANDRA PARTITION KEYS

Imagine we have a four node
Cassandra cluster.
Node 1 is responsible for partition key
hash values 0-24;
Node 2 is responsible for partition key
hash values 25-49; and so on.

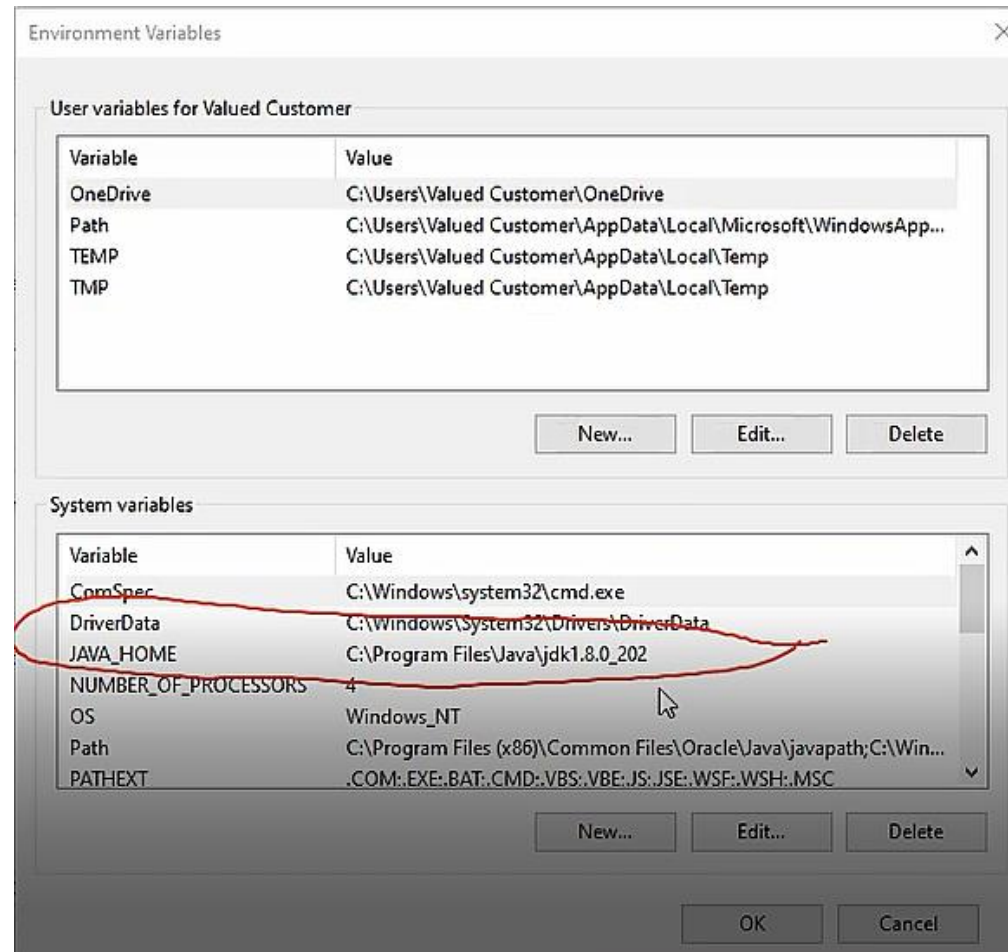


CASSANDRA LAB

Download and Install

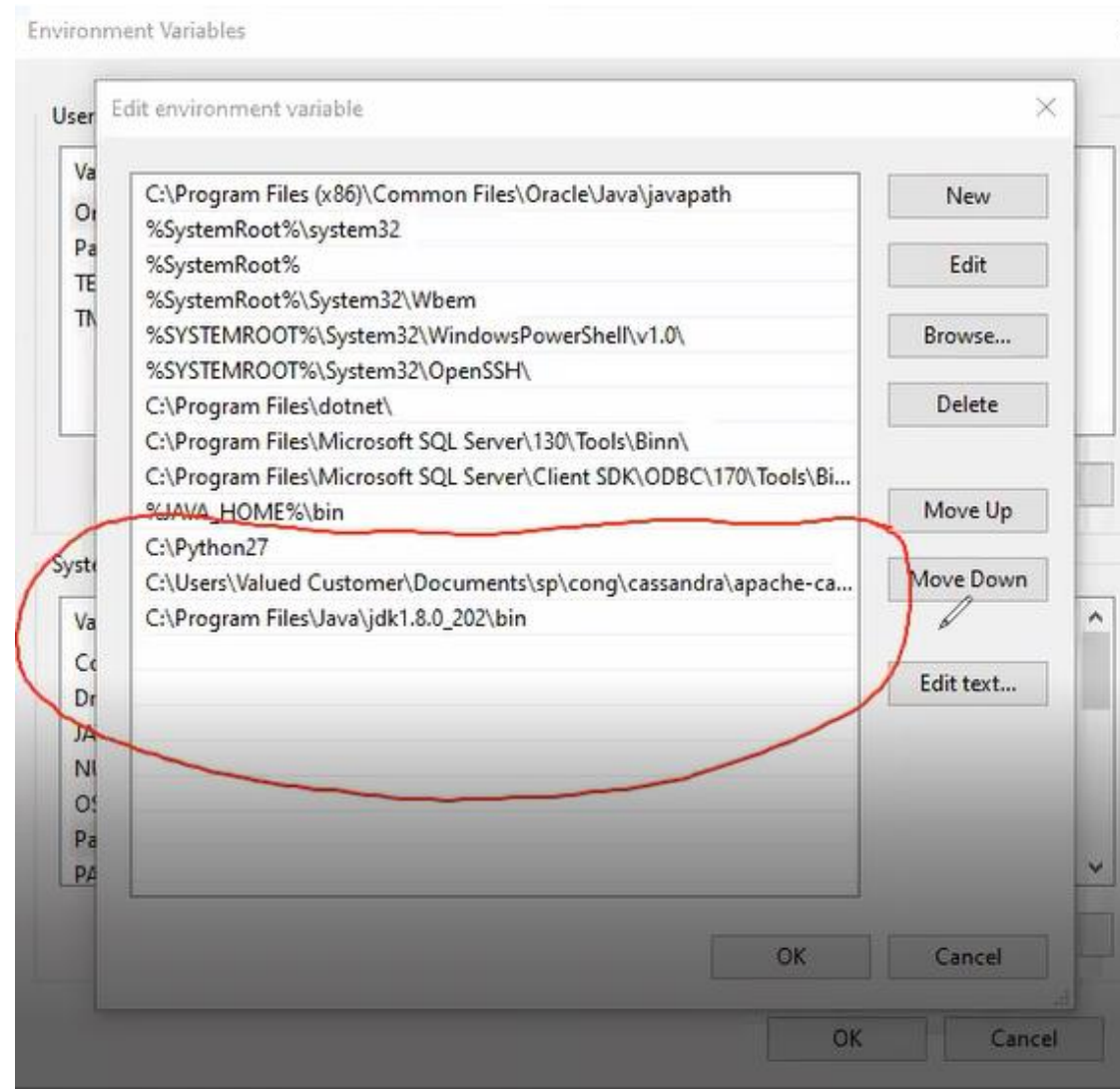
- [JDK – 1.8](#)
- [Python 2.7](#)
- [Download Apache Cassandra](#)

Add JAVA_HOME



Edit Path Variable

- Give path for the following
 - Python 2.7
 - Cassandra's bin folder
 - JDK bin



- Goto Cassandra/bin and modify and save the following files
 - Cassandra.bat
 - Nodetool.bat

```
C:\Users\Valued Customer\Documents\sp\cong\cassandra\apache-cassandra-3.11.9\bin\cassandra.bat - Notepad++
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
NestedLoops.java Test.java nodetool.bat cassandra.bat
13 @REM WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
14 @REM See the License for the specific language governing permissions and
15 @REM limitations under the License.
16
17 @echo off
18 if "%OS%" == "Windows_NT" setlocal
19
20 set ARG=%1
21 set INSTALL="INSTALL"
22 set UNINSTALL="UNINSTALL"
23 set JAVA_HOME=C:\Program Files\Java\jdk1.8.0_202
24
25
26 pushd %~dp0..
27 if NOT DEFINED CASSANDRA_HOME set CASSANDRA_HOME=%CD%
28 popd
```










Cassandra.bat

```
12 @REM distributed under the License is distributed on an "AS IS" BASIS,
13 @REM WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
14 @REM See the License for the specific language governing permissions and
15 @REM limitations under the License.
16 set JAVA_HOME=C:\Program Files\Java\jdk1.8.0_202
17 @echo off
```

Nodetool.bat

Running Cassandra

- From Cassandra's bin folder run the following as administrator

<input type="checkbox"/> Name	Date modified	Type	Size
 cassandra	2020-10-29 7:36 AM	File	11 KB
 cassandra	2021-03-25 11:39 PM	Windows Batch File	8 KB
 cassandra.in	2020-10-29 7:36 AM	Windows Batch File	4 KB
 cassandra.in.sh	2020-10-29 7:36 AM	SH File	4 KB
 cassandra	2020-10-29 7:36 AM	Windows PowerShell Script	13 KB
 cqlsh	2020-10-29 7:36 AM	File	2 KB
 cqlsh	2020-10-29 7:36 AM	Windows Batch File	2 KB
 cqlsh	2020-10-29 7:36 AM	PY File	97 KB
 debug-cql	2020-10-29 7:36 AM	File	3 KB

```
Windows PowerShell

*-----*
*-----*

WARNING! Automatic page file configuration detected.
It is recommended that you disable swap when running Cassandra
for performance and stability reasons.

*-----*
*-----*
*-----*
*-----*

WARNING! Detected a power profile other than High Performance.
Performance of this node will suffer.
Modify conf\cassandra.env.ps1 to suppress this warning.

*-----*
*-----*
CompilerOracle: dontinline org/apache/cassandra/db/Columns$Serializer.deserializeLargeSubset (Lorg/apache/cassandra/io/u
til/DataInputPlus;Lorg/apache/cassandra/db/Columns;I)Lorg/apache/cassandra/db/Columns;
CompilerOracle: dontinline org/apache/cassandra/db/Columns$Serializer.serializeLargeSubset (Ljava/util/Collection;ILorg/
apache/cassandra/db/Columns;ILorg/apache/cassandra/io/util/DataOutputPlus;)V
CompilerOracle: dontinline org/apache/cassandra/db/Columns$Serializer.serializeLargeSubsetSize (Ljava/util/Collection;IL
org/apache/cassandra/db/Columns;I)I
```

```
C:\Windows\System32\cmd.exe

WARNING: console codepage must be set to cp65001 to support utf-8 encoding on Windows platforms.
If you experience encoding problems, change your console codepage with 'chcp 65001' before starting cqlsh.

Connected to Test Cluster at 127.0.0.1:9042.
[cqlsh 5.0.1 | Cassandra 3.11.9 | CQL spec 3.4.4 | Native protocol v4]
Use HELP for help.
WARNING: pyreadline dependency missing. Install to enable tab completion.
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.
- **SELECT** – This clause reads data from a table
- **WHERE** – The where clause is used along with select to read a specific data.
- **ORDERBY** – The orderby clause is used along with select to read a specific data in a specific order.

Cassandra Vs. SQL

- MySQL is the most popular (and has been for a while)
- On > 50 GB data
- MySQL
 - Writes 300 ms avg
 - Reads 350 ms avg
- Cassandra
 - Writes 0.12 ms avg
 - Reads 15 ms avg

CQL DataTypes

Data Type	Constants	Description
ascii	strings	Represents ASCII character string
bigint	bigint	Represents 64-bit signed long
blob	blobs	Represents arbitrary bytes
Boolean	booleans	Represents true or false
counter	integers	Represents counter column
decimal	integers, floats	Represents variable-precision decimal
double	integers	Represents 64-bit IEEE-754 floating point
float	integers, floats	Represents 32-bit IEEE-754 floating point
inet	strings	Represents an IP address, IPv4 or IPv6
int	integers	Represents 32-bit signed int
text	strings	Represents UTF8 encoded string
timestamp	integers, strings	Represents a timestamp
timeuuid	uuids	Represents type 1 UUID
uuid	uuids	Represents type 1 or type 4
		UUID
varchar	strings	Represents UTF8 encoded string
varint	integers	Represents arbitrary-precision integer

Commands/Queries

- This command copies data to and from Cassandra to a file.
 - COPY emp (emp_id, emp_city, emp_name, emp_phone,emp_sal) TO 'myfile';
- Describe commands
 - describe cluster;
 - describe keyspaces;
 - describe tables;
 - describe table emp;
- To run commands stored in a file
 - source '/home/hadoop/Cassandra_Lab/inputfile';

Creating and Using Keyspace

- CREATE KEYSPACE <identifier> WITH <properties>

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

- To use a keyspace
 - USE NoSql;

Alter and dropping Keyspace

- `ALTER KEYSPACE Nosql WITH replication =
{ 'class': 'NetworkTopologyStrategy',
'replication_factor' : 3};`
- `DROP KEYSPACE "KeySpace name"`
 - `DROP KEYSPACE NoSql`

Creating Table

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

Altering and dropping Table

- ALTER TABLE table name ADD new column datatype;
- ALTER table name DROP column name;
- DROP TABLE <tablename>
- TRUNCATE <tablename>

Executing multiple modification statements

```
BEGIN BATCH
... INSERT INTO emp (emp_id, emp_city, emp_name, emp_phone, emp_sal)
      values( 4, 'Oakville', 'Sophia', 647-234-6789, 60000);
... UPDATE emp SET emp_sal = 50000 WHERE      emp_id =3;
... DELETE emp_city FROM emp WHERE emp_id = 2;
... APPLY BATCH;
```

Creating and dropping an Index

- `CREATE INDEX name ON emp1 (emp_name) ;`
- `drop index name;`

Select Clause

- `select * from emp;`
- `SELECT emp_name, emp_sal from emp;`
- `SELECT * FROM emp WHERE emp_sal=50000;`

Future Reading

- [Cassandra](#) : Official Website
- [Cassandra Wikipedia](#) : Wikipedia reference for Cassandra

References

- Chapter 9, 10- NoSql for Mere Mortals , Dan Sullivan. Addison Wesley.

Thanks