
# Chapter 7

# Introduction to Cassandra

# 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 Big Table.

- Cassandra **does not compromise on availability**. Since it does not have master slave architecture. (No Single point of Failures)

# Features of Cassandra

# Features of Cassandra

- Open Source
- Distributed
- Decentralized (Server Symmetry)
- No single point of failure - No Master Node, so no Master failure.
- Column-oriented
- Peer to Peer - All nodes are equal
- Elastic Scalability

# Companies successfully deployed Cassandra

Netflix


- online DVD and Blu-Ray movie retailer
- Nielsen study showed that 38% of Americans use or subscribe to Netflix
- **Why Cassandra?**
- Using a central SQL database negatively impacted scalability and availability
- International Expansion required Multi-Datacenter solution
- Need for configurable Replication, Consistency, and Resiliency in the face of failure
- Cassandra on AWS offered high levels of scalability and availability
- Netflix can now handle **over 250 million active users** without downtime. Netflix can now process **10 million events per second** without failure.

# Companies successfully deployed Cassandra


Spotify

- Streaming digital music service
- Music for every moment on computer, phone, tablet, and more
- **Why Cassandra?**
- With more users, scalability problems arised using postgreSQL
- With multiple data centers, streaming replication in postgreSQL was problematic
  - ex: high write volumes
- Chose Cassandra
  - lack of single points of failure
  - no data loss confidence
  - Big Table design

**Cassandra supports horizontal scalability.**

•Can handle **10 million+ writes per second** without downtime.

• By adding more nodes, Spotify could easily scale without changing the architecture.

**Result:**

•**Spotify could now handle millions of users** without downtime.

# Companies successfully deployed Cassandra

Hulu

- a website and a subscription service offering on-demand streaming video media
- ~30 million unique viewers per month
- **Why Cassandra?**
- need for Availability
- need for Scalability
- Good Performance
- Nearly Linear Scalability
- Geo-Replication
- Minimal Maintenance Requirements
- Hulu could now serve **30 million+ users** without any performance drop.

# Companies successfully deployed Cassandra

- Twitter
- Cisco
- Adobe
- Ebay
- Rack space
- Uber
- Facebook
- Instagram

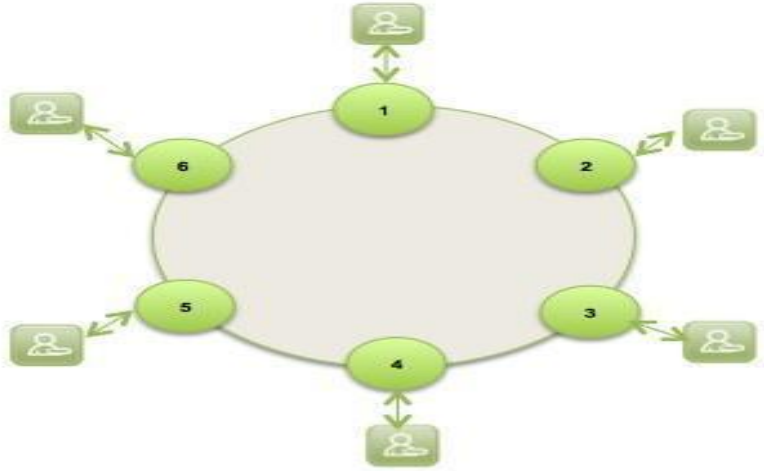
# Cassandra Architecture

# Cassandra Architecture

- Peer –to-Peer Architecture
- Gossip and Failure Detection
- Partitioner
- Replication Factor
- Anti- Entropy and Read repair
- Writes in Cassandra
- Hinted Handoffs
- Tunable Consistency

# Peer to Peer Network

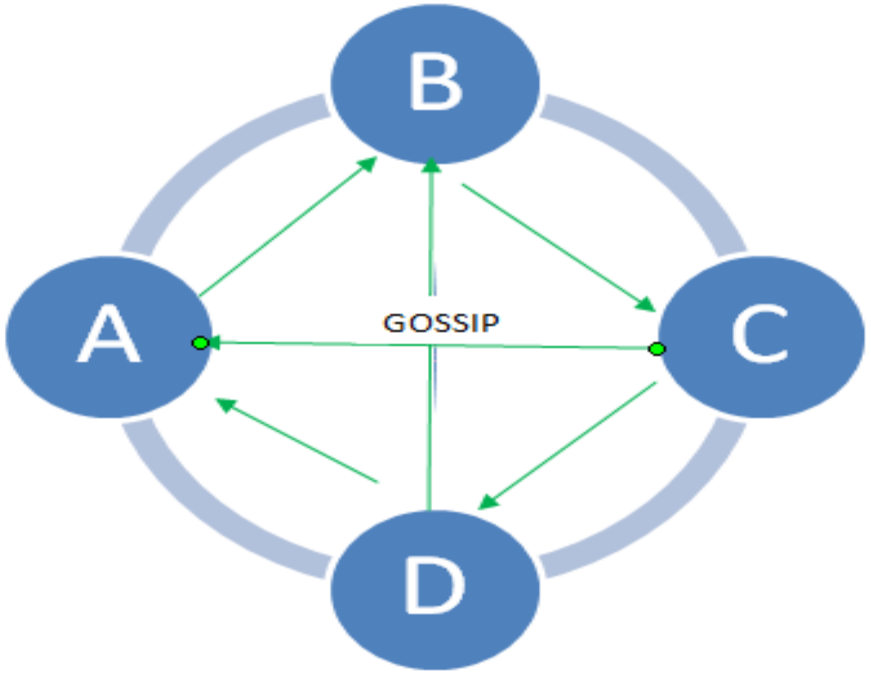
- Cassandra is designed to distribute and manage large data loads across multiple nodes in a cluster.
- Does not have Master Slave architecture( no SOP)
- A node in Cassandra is structurally identical to nay other node.
- In case any node fails, it definitely impacts the throughput.
- However incase of graceful degradation where everything does not come crashing at any given instant owing to node failure



# Gossip Protocol and Failure Detection

- Gossip protocol is used for intra ring communication.
- It is a peer to peer communication protocol which eases discovery and sharing of location and state information with other nodes in cluster.
- Network Communication protocols inspired for **real life rumor spreading**.
- Example – Node A wish to search for pattern in data
  - Round 1 – Node A searches locally and then gossips with node B.
  - Round 2 – Node A,B gossips with C and D.
  - Round 3 – Nodes A,B,C and D gossips with 4 other nodes ……
- Round by round doubling makes protocol very robust.

# Gossip Protocol




PEER TO PEER DISTRIBUTION
MODEL OF CASSANDRA

# Failure Detection- Anti-Entropy and Read Repair

- For repairing unread data, Cassandra uses what's called an anti-entropy version of the gossip protocol.

- Anti-entropy and Read Repair

  - A cluster is made of several nodes.
  - To achieve fault tolerance, a given piece of data is replicated on one or more nodes.
  - A client can connect to any node in the cluster to read data.
  - **AntiEntropy means comparing all the replicas of each piece of data that exist (or are supposed to) and updating each replica to the newest version.**

  - Some times few nodes may respond with out –of date value . This can be repaired using **Read Repair** operation.

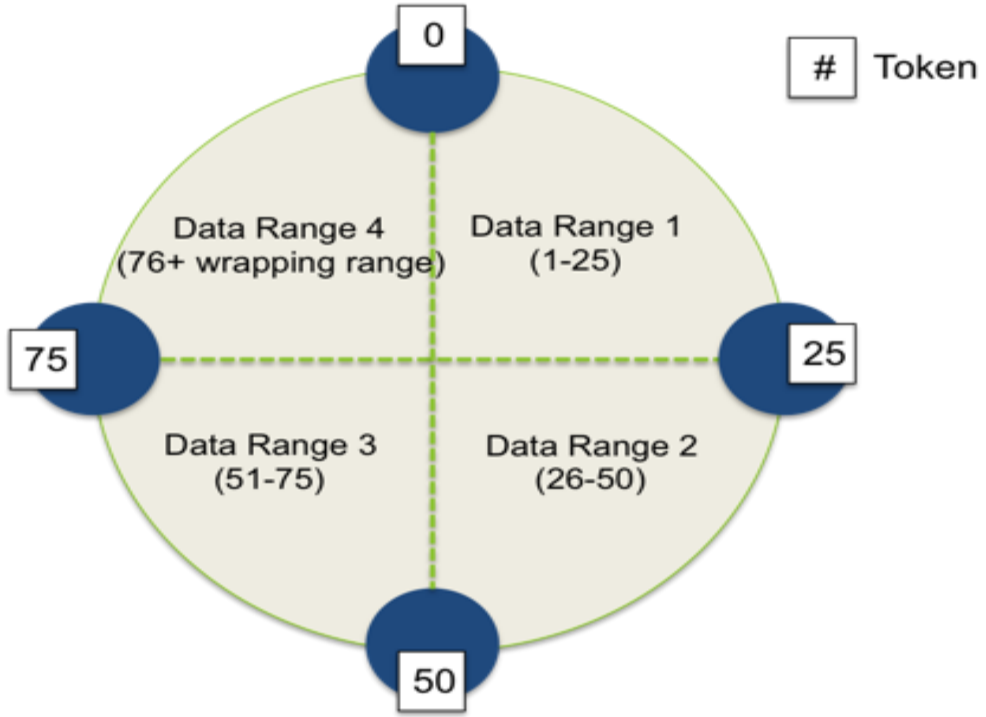
  - Read repair operation is performed either before or after returning the value to the client as per the specified consistency level.

# Partitioner

- A partitioner takes a call on how to distribute data on the various nodes in cluster. Nodes are *logically* structured in Ring Topology.

- It also determines a node on which to place the very first copy of the data.

- Basically a partitioner is a hash function to compute the token of the partition key.



# Replication

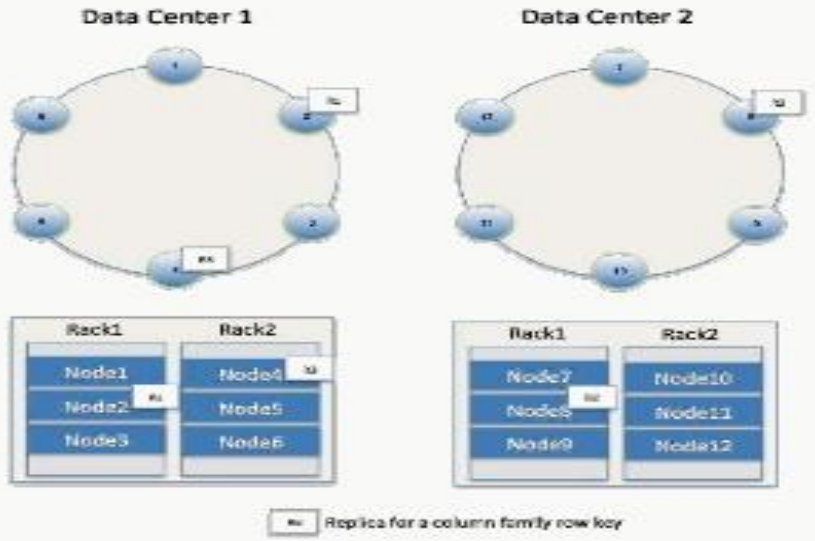
- Replication Factor(RF) determines the number of copies of data.
- RF should ideally be more than one and not more than the number of nodes in the cluster.
- Two replication strategies are,
  - Simple strategy
  - Network Topology strategy

# Two different Replication strategies

DATASTAX

## Cassandra Replication Strategies

- **Simple Strategy**: places the original row on a node determined by the partitioner. Additional replica rows are placed on the next nodes clockwise in the ring without considering rack or data center location
- **Network Topology Strategy**: allows for replication between different racks in a data center and/or between multiple data centers and the cloud. This strategy provides more control over where replica rows are placed

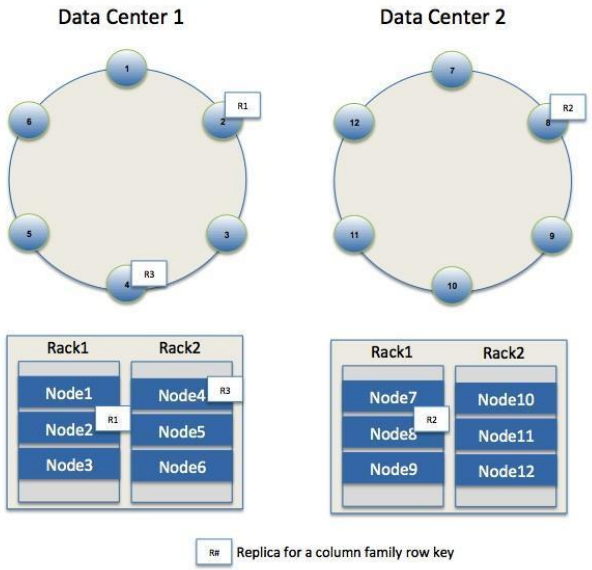


# Replication

## Different Replication Policies

**Rack Unaware –** replicate data at N-1 successive nodes after its coordinator

**Rack Aware –** uses ‘Zookeeper’ to choose a leader which tells nodes the range they are replicas for

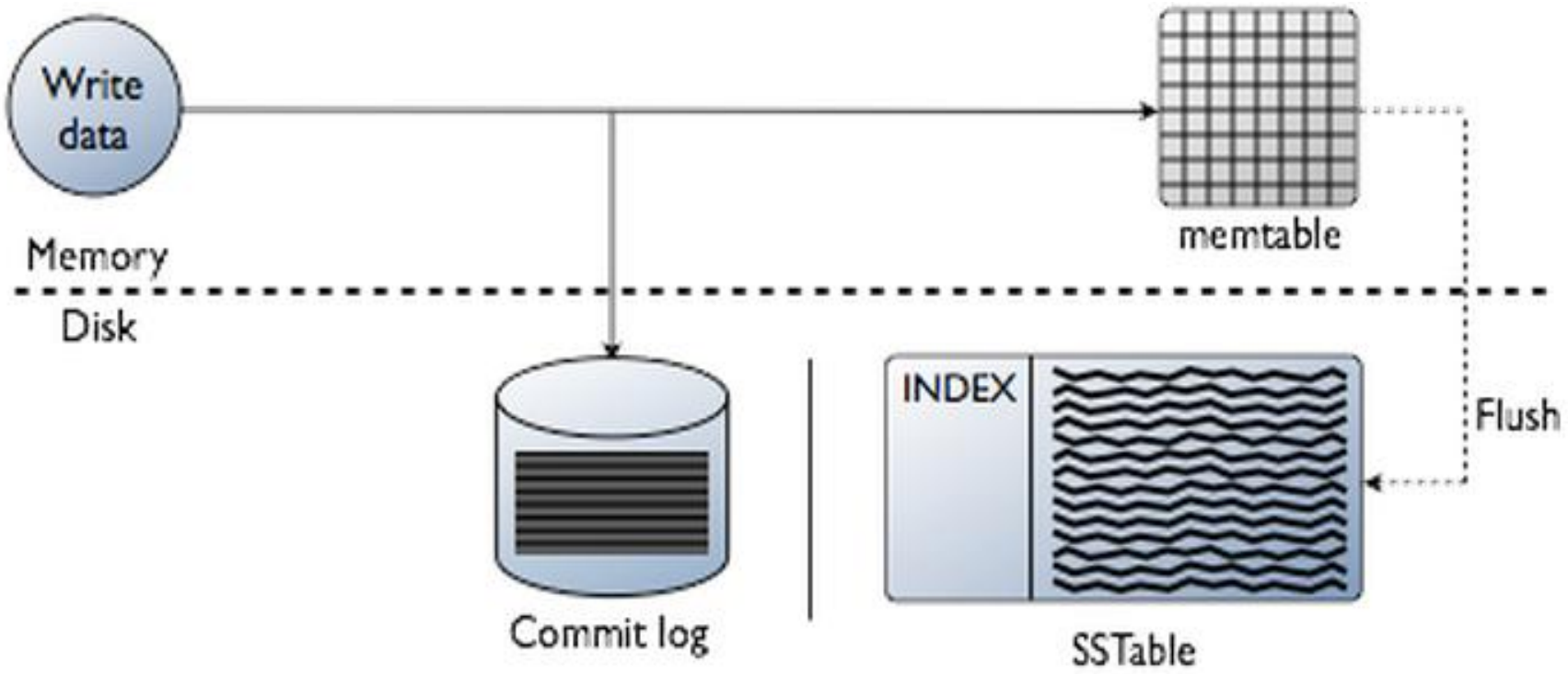



**Datacenter Aware –** similar to Rack Aware but leader is chosen at Datacenter level instead of Rack level.

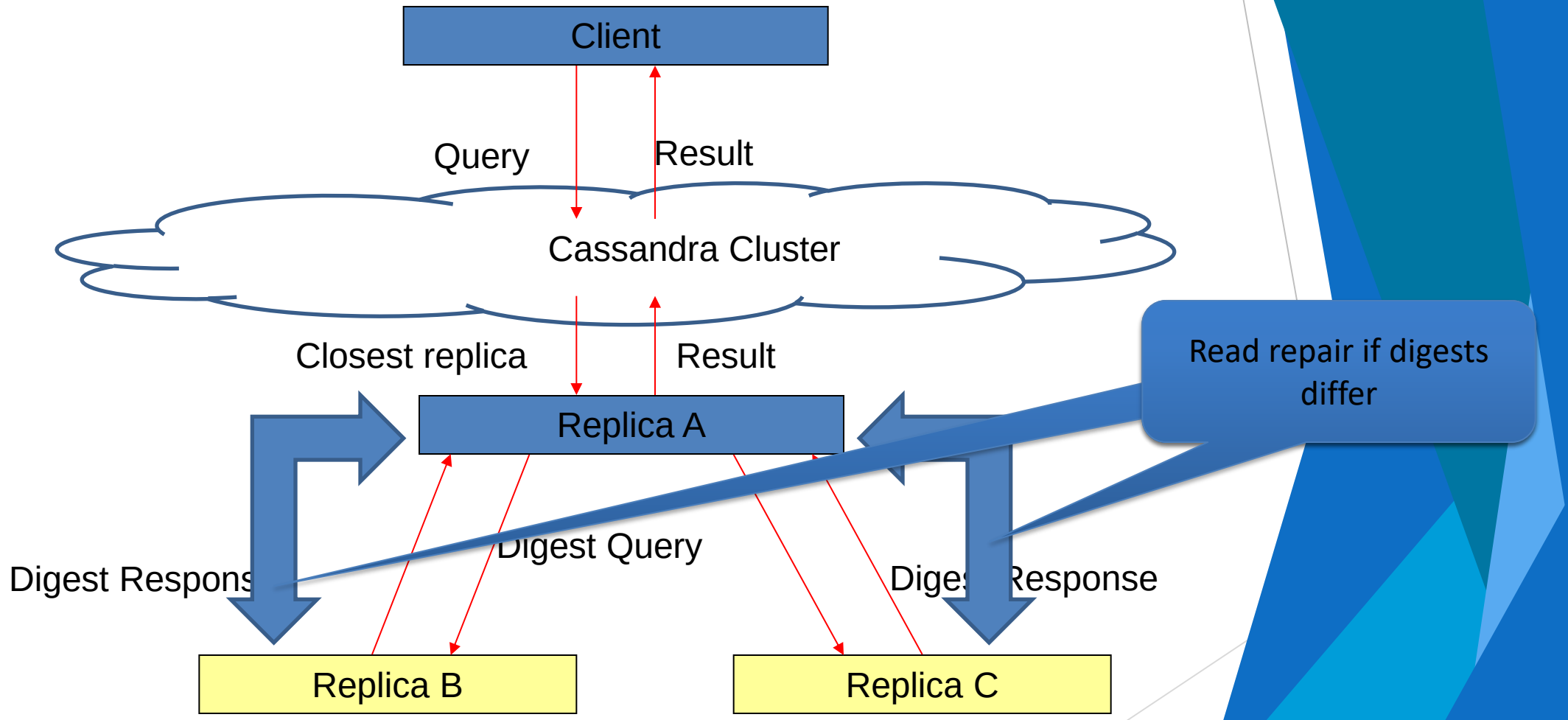
# Writes in Cassandra

- 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. ***(not interfere with ongoing write operations.)***
- 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.

# Writes in Cassandra



# Read Operation



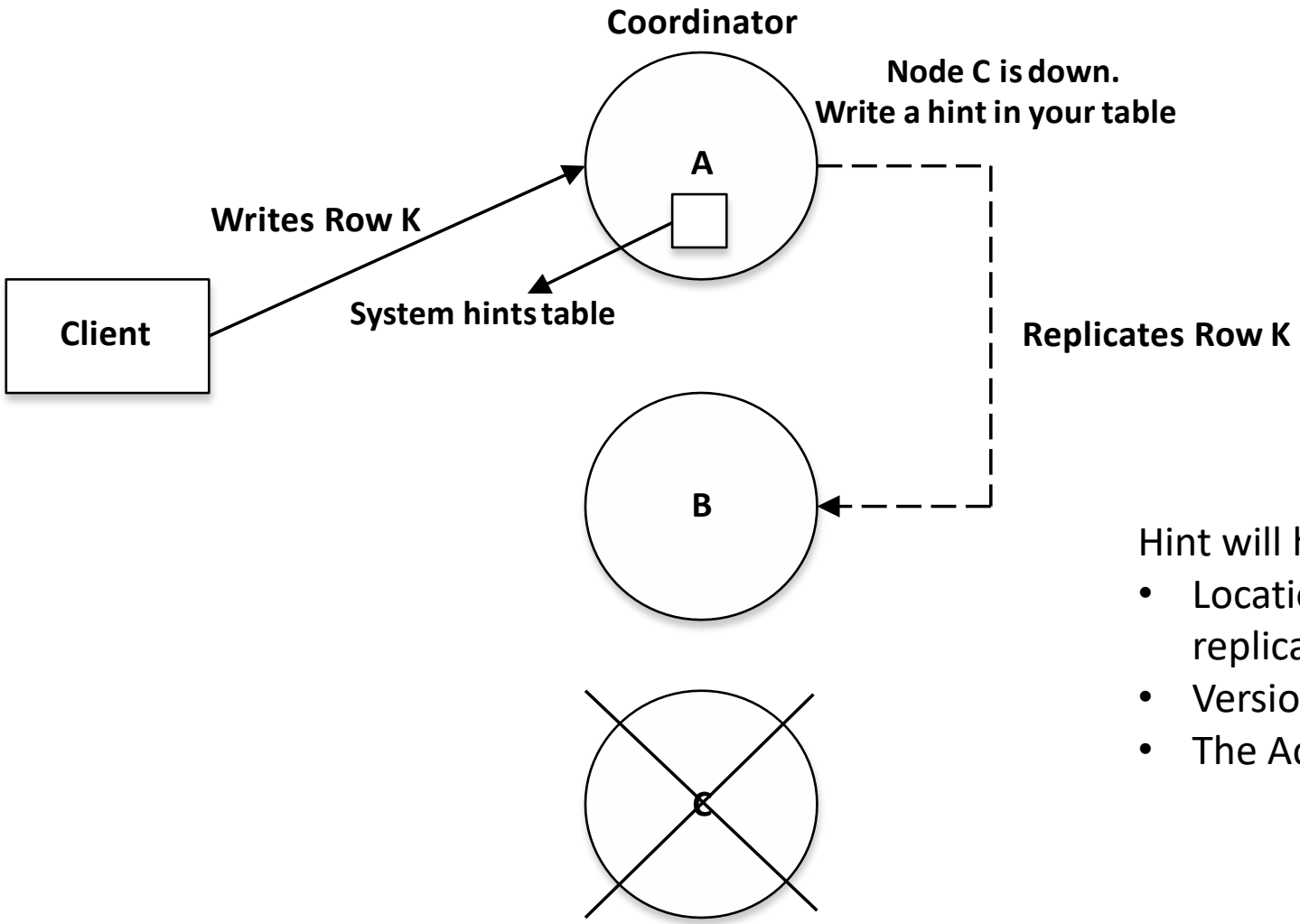

* Figure taken from Avinash Lakshman and Prashant Malik (authors of the paper) slides.

| Component | Location | Mutable/Immutable | Purpose |
|---|---|---|---|
| Commit Log | Disk (on each node) | Mutable | Ensures durability of writes |
| Memtable | Memory (RAM, on each node) | Mutable | Stores recent writes before flushing |
| SSTable | Disk (on each node) | Immutable | Stores flushed Memtable data permanently |

Immutable-they are never modified after being written.

# Hinted Handoffs

**Hinted Handoff** is a **fault-tolerance mechanism** in Cassandra that ensures **high availability** when a replica node is **temporarily unavailable**.




Hint will have the following data:

- Location of the node on which the replica is to be placed
- Version meta data
- The Actual data

# Tunable Consistency

- Consistency describes how and whether a system is left in a consistent state after an operation.
- In distributed data systems like Cassandra, this usually means that once a writer has written, all readers will see that write.
- On the contrary to the strong consistency used in most relational databases (**ACID** ) Cassandra is at the other end of the spectrum **(BASE for *Basically Available Soft-state Eventual consistency*).**
- Cassandra weak consistency comes in the form of eventual consistency which means the database eventually reaches a consistent state.
- As the data is replicated, the latest version of something is sitting on some node in the cluster, but older versions are still out there on other nodes, but eventually all nodes will see the latest version.

# 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. |

# Read Consistency

More specifically:
R=read replica count
W=write replica count
N=replication factor

**Q=QUORUM (Q = N / 2 + 1)**

- If W + R > N, you will have consistency
- W=1, R=N
- W=N, R=1
- W=Q, R=Q where Q = N / 2 + 1

- Cassandra provides consistency when R + W > N
- A Consistency Level of ONE means R or W is 1.
- A Consistency Level of QUORUM means R or W is ceiling((N+1)/2).
- A Consistency Level of ALL means R or W is N.
- So if you want to write with a Consistency Level of ONE and then get the same data when you read, you need to read with Consistency Level ALL.

# Write Consistency

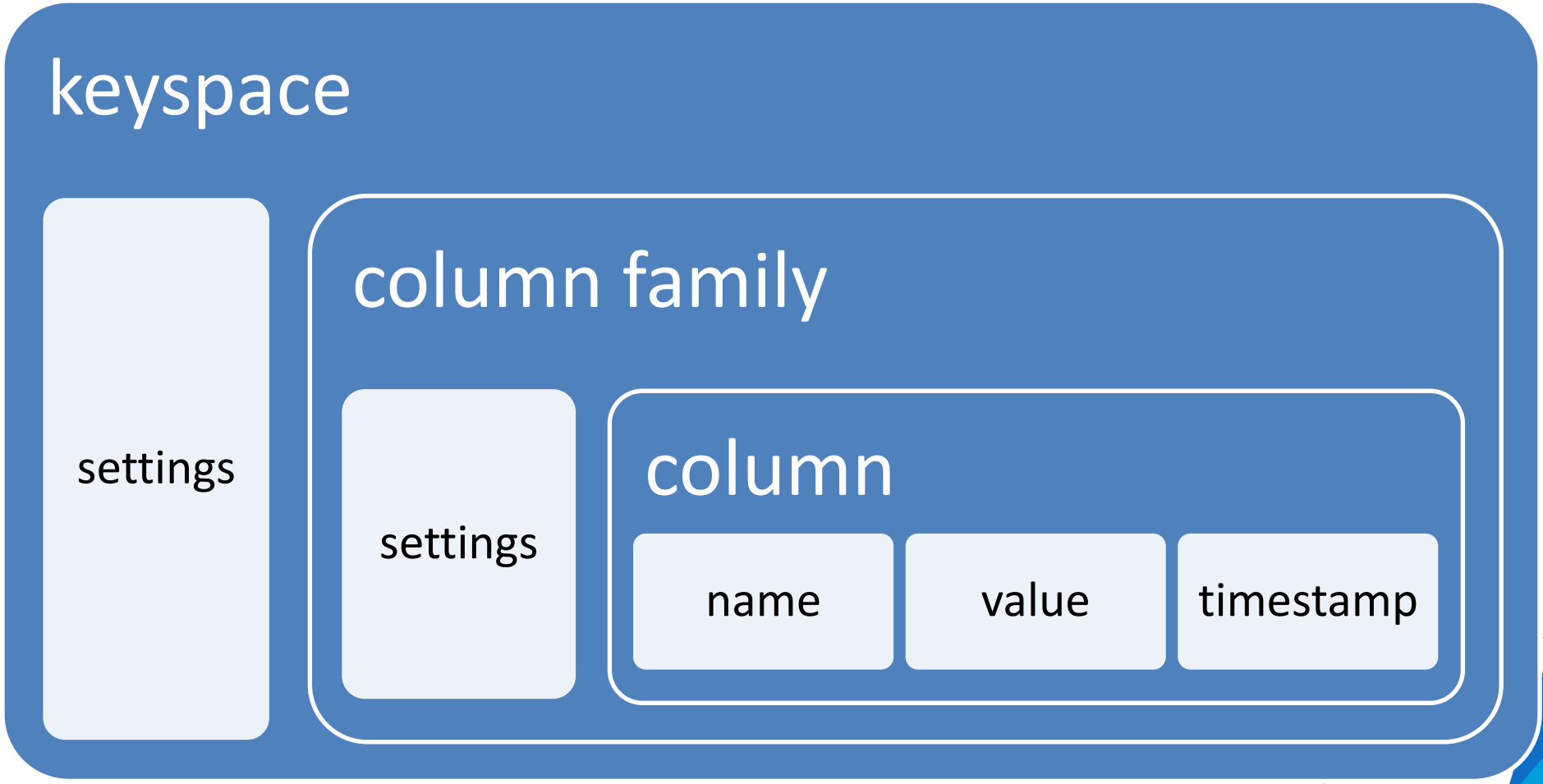
| | |
|---|---|
| **ALL** | **This is the highest level of consistency of all levels as it necessitates that a write must be written to the commit log and Memtable on all replica nodes in the cluster.** |
| **EACH_QUORUM** | A write must be written to the commit log and Memtable on a quorum of replica 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. |

# Trade-offs of Tunable Consistency

| Consistency Level | Availability | Performance | Consistency |
|---|---|---|---|
| ANY / ONE | High | Fast | Weak (Eventual Consistency) |
| QUORUM | Medium | Medium | Stronger Consistency |
| ALL | Low | Slow | Strong Consistency |

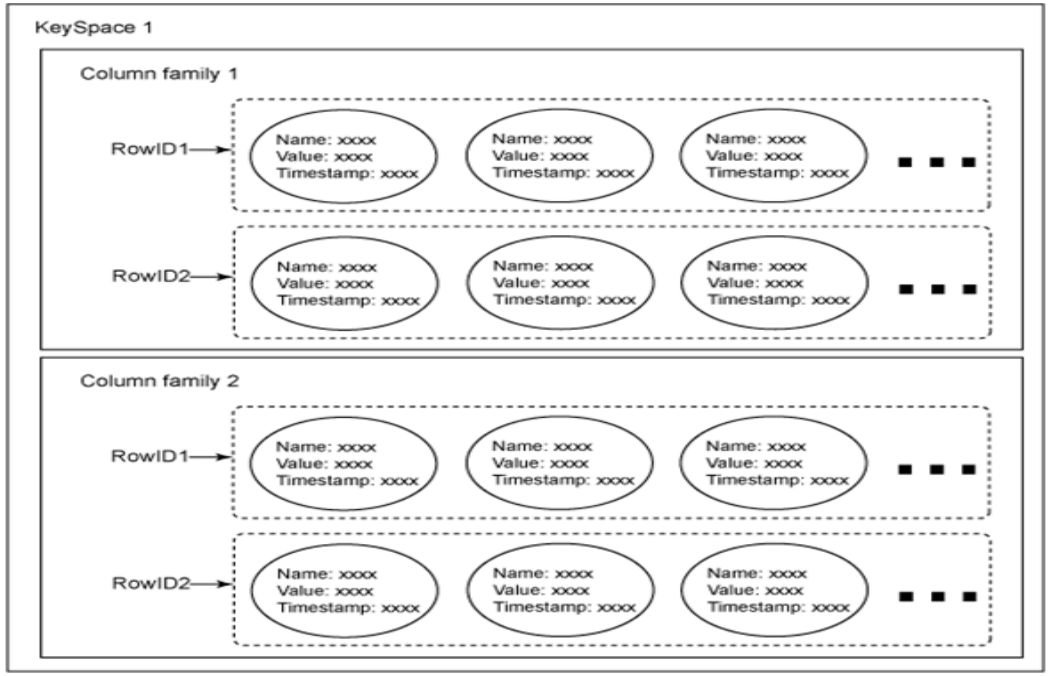
- **High availability (ANY, ONE)** is best for systems prioritizing uptime (e.g., social media, IoT).
- **Stronger consistency (QUORUM, ALL)** is best for critical applications (e.g., banking, inventory systems).

# Data Model


keyspace
settings
column family
settings
column
name
value
timestamp

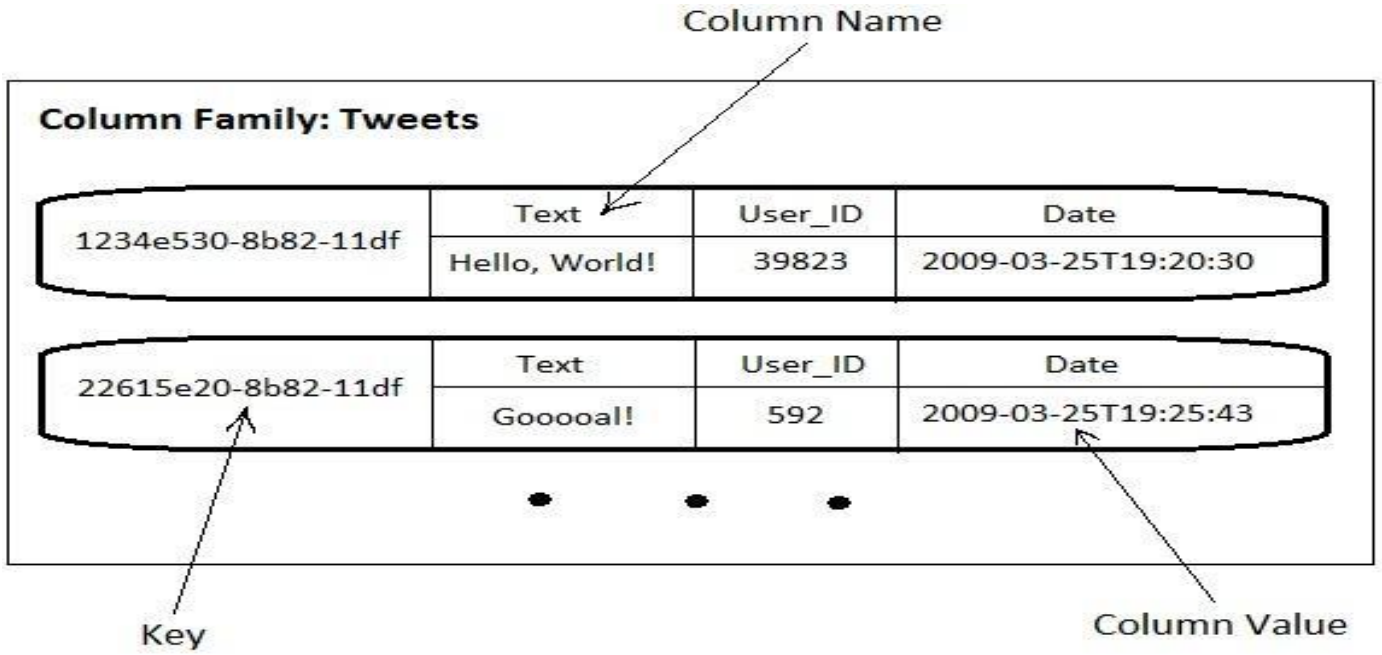
# Cassandra data model

- Cassandra is a *column oriented NoSQL system*
- Column families: sets of key-value pairs
  - **column family as a table and key-value pairs as a row (using relational database analogy)**
- A row is a collection of columns labeled with a name



# Cassandra Row

- the value of a row is itself a sequence of key-value pairs
- such nested key-value pairs are *columns*
- key = column name
- a row must contain at least 1 column.



# Key Space

- A Key Space is a group of column families together.
- It is only a logical grouping of column families and provides an isolated scope for names

# Facebook Inbox Search

- Cassandra developed to address this problem.
- 50+TB of user messages data in 150 node cluster on which Cassandra is tested.
- Search user index of all messages  in 2 ways.
  - Term search : search by a key word
  - Interactions search : search by a user id

| Latency Stat | Search Interactions | Term Search |
|---|---|---|
| Min | 7.69 ms | 7.78 ms |
| Median | 15.69 ms | 18.27 ms |
| Max | 26.13 ms | 44.41 ms |

# Comparison with MySQL

- MySQL > 50 GB Data
Writes Average : ~300 ms
Reads Average : ~350 ms

- Cassandra > 50 GB Data
Writes Average : 0.12 ms
Reads Average : 15 ms

- Stats provided by Authors using facebook data.

# Cassandra Query Language 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
 };
```

# To Describe existing key spaces

**DESCRIBE KEYSPACES**;

To get more details about key space like Keyspace name, durable writes, strategy class, etc.

**SELECT * FROM system.schema_keyspaces;**

To use keyspace

**> USE KEYSPACE NAME**

# 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
  );
```

# CRUD - Insert

- Insert writes one or more columns to a record in Cassandra table automatically.
- Insert statement does not return output.
- Primary key is must.

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;
```

# CRUD - Select

To view the data from the table “student_info”.

```
SELECT *
        FROM student_info;
```

To view only those record where the roll no column either has a value 1 or 2 or 3

```
SELECT *FROM student_info WHERE RollNO IN (1,2,3)
```

# 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);
```

**To execute the query using Index**

```
SELECT * FROM student_info WHERE studname=‘Stephen Fox’;
```

**To Create another index on column LastExamPercent**

```
CREATE INDEX ON student_info(LastExamPercent);
```

# Other opertaors

- Limit
- AS  (alias)
- Order By
- Group By
- Desc
- ASC ( the default sorting order)

- **SELECT rollno, hobbies, language, lastexampercent FROM student_info LIMIT 2;**

- **SELECT rollno, language AS “known language” FROM  student_info;**

# 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;
```

# Update Primary Key

```
UPDATE   student_infoSET rollno=6 WHERE rollNo = 2;
```

**But it returns Bad request . It does not allow to update primary key column.**

# Updating more than one column of  a row

1. Before update

```
Select rollno, studname,lastexam percent from student_info where rollno=3;
```

2. Applying update

```
UPDATE  student_info SET  lastexampercent  =  85 WHERE rollNo = 3;
```

**Another Example:**

```
UPDATE Movies SET col1 = val1, col2 = val2 WHERE movieID = key1;
```

# Update a column in several rows at once:

```
UPDATE users SET state = 'TX' WHERE user_uuid IN (88b8fd18-b1ed-4e96-bf79-4280797cba80, 06a8913c-c0d6-477c-937d-6c1b69a95d43, bc108776-7cb5-477f-917d-869c12dfffa8)
```

# 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;**

# Delete Row from table

```
DELETE FROM student_info WHERE rollno=2;
```

# Why ALLOW FILTERING?

Let’s take for example the following table:

```
CREATE TABLE blogs (blogId int,
                    time1 int,
                    time2 int,
                    author text,
                    content text,
                    PRIMARY KEY(blogId, time1, time2));
```

To execute the following query:

```
SELECT * FROM blogs;
```

# Why ALLOW FILTERING?

If you now want only the data at a specified time1

```
SELECT * FROM blogs WHERE time1 = 1418306451235;
```

In response, you will receive the following error message:

*Bad Request: Cannot execute this query as it might involve data filtering and thus may have unpredictable performance. If you want to execute this query despite the performance unpredictability, use ALLOW FILTERING.*

# Why ALLOW FILTERING?

Cassandra knows that it might not be able to execute the query in an efficient way.

It is therefore warning you: “Be careful. Executing this query as such might not be a good idea
as it can use a lot of your computing resources”.

**The only way Cassandra can execute this query is by retrieving all the rows from the table blogs and then by filtering out the ones which do not have the requested value for the time1 column.**

```
SELECT * FROM blogs WHERE time1 = 1418306451235 ALLOW FILTERING;
```

# 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.

# Different Collections

## SET collection

A column of type set consists of unordered unique values.

## LIST collection

When the order of elements is required we can use list. List allows to store the same values multiple times.

## MAP collection

Map is used to map one thing to another. A map is a pair of typed values.

Each element of the map is stored as a Cassandra column.

# 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;

        DELETE language[1] FROM student_info
            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’;
```

# More Practice on Collections

1. TO create a table “ users” with an “emails” column. Email column type as SET.

2. To insert values into the “ emails” column of the “ users” table.

3. Add an element to a set using the UPDATE command and the addition (+) operator.

4. To remove an element from a set using the subtraction (-) operator.

```
UPDATE users SET emails= emails-{XX@gmail.com} WHERE user_id='XX';
```

5. To alter the users table to add a column, “top places of type list.

Note:

To append use + operator.

# COUNTER

A counter is a special column that is changed in increments.

For example we may need a counter column to count the number of times a particular book is issued from the library by the student.

Step1:

```
CREATE TABLE library_book ( Counter_ value counter,
                            Book_Name varchar,
                            stud_name varchar,
                            PRIMARY KEY(Book_Name, stud_name)
                                );
```

# COUNTER

Step2:

```
UPDATE library_book SET counter_value=counter_value+1

WHERE Book_Name='Business Analytics' AND stud_name='jeet';
```

Step 3: Increase the value of counter

```
UPDATE library_book SET counter_value=counter_value+1

WHERE Book_Name='Business Analytics' AND stud_name='shann';
```

Step 4:

```
UPDATE library_book SET counter_value=counter_value+1

WHERE Book_Name='Business Analytics' AND stud_name='jeet';
```

# 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;
```

# ALTER COMMAND

Alter command can be used to change the structure of the table/column.

1. Change the type of the column.

```
ALTER TABLE sample ALTER sampleid TYPE int;
```

2. To delete a column.

```
ALTER TABLE sample DROP sampleid;
```

# DROP command

**Drop a table**

DROP COLUMNFAMILY Table name ;

**Drop a data base**

**DROP KEYSPACE Keyspace name;**

# 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';
```

## IMPORT from STDIN

```
COPY persons (id, fname, lname)  FROM STDIN;
```

## EXPORT from STDOUT

```
COPY elearninglists (id, course id, title, owner) TO STDOUT;
```

# Querying System Tables

1. SELECT * FROM system.schema_keyspaces;
2. SELECT * FROM system.schema_columnfamilies;
3. SELECT * FROM system.schema_column;

# 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?

# Summary please…

Ask a few participants of the learning program to summarize the lecture.

# References …

# Further Readings

- http://www.datastax.com/documentation/cassandra/2.0/cassandra/gettingStartedCassandraIntro.html
- http://www.datastax.com/documentation/cql/3.1/pdf/cql31.pdf
- http://www.datastax.com/documentation/cassandra/2.0/cassandra/dml/dml_config_consistency_c.html

# Thank you