Cassandra Notes

Cassandra Usecase:

Mapping different data to relational DB

id	brand	color	screen type	publisher	author	model Id	name	size	title	...
Mob1	Samsung	Black				Galaxy S6			Samsung Galaxy6	...
Book 1				Riverhead	Khalid Hosseini		Kite Runner		Kite Runner	...
TV1	Sony	Black	Ultra HD			Bravia			Sony Bravia	...
Tshirt 1	Lee	Red/ Green						M	Lee Round shirt	...

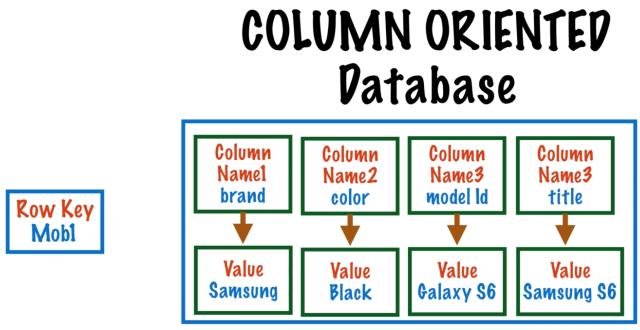
The Cassandra way

id	attribute Name	attribute Value
Mob1	brand	Samsung
Mob1	color	Black
Mob1	model Id	Galaxy S6
Mob1	title	Samsung Galaxy6
Book1	author	Khalid Hosseini

Relational DB to Casandra mapping:

Row id → Row key

Column 1 → One column in column group.



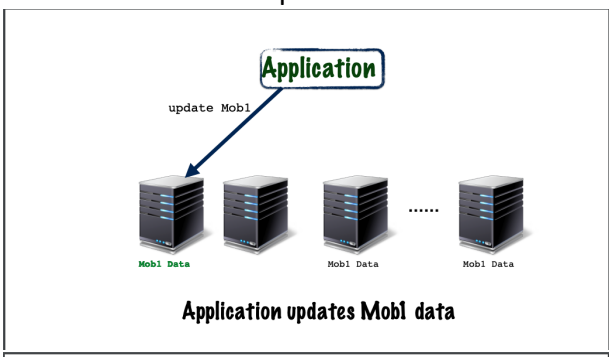
Advantages:

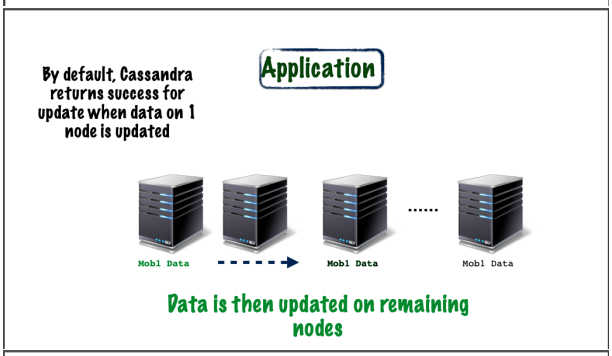
- Disk space minimized by not saving empty space cells
- Not bounded to any table schema
- De-centralized data so no SPOF.
- Elastically scalable – To scale new server needs to be added and data will be automatically distributed.
- Tuneable consistency

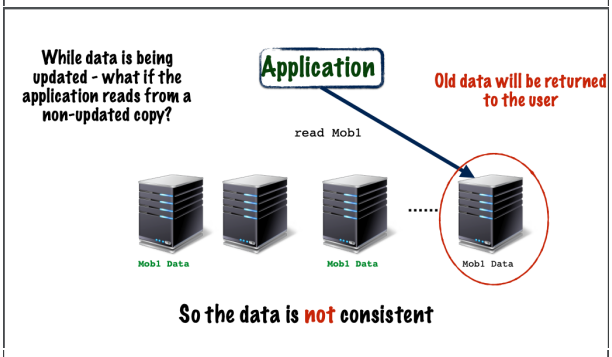
- High Availability
- Fault tolerant
- SQL like query language (CQL)
- Support secondary indexes

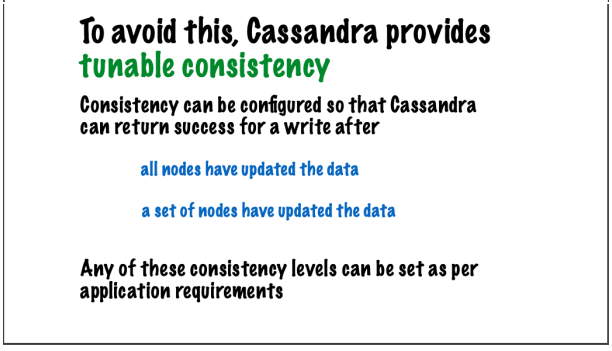
Limitations :

- Not ACID compliant









Cassandra should not be used

if data can be managed on a single system

if we want to instantly read what is written

eg Bank accounts, product orders

Cassandra data model:

- Column definition → name & data type
- Data types → int ,float ,double ,Boolean ,text ,blob ,counter ,UUID ,timestamp ,set ,list ,map
- Column family → Collection of one or more columns
- Columns can be added to the family at any point of time.
- Every column family has a primary key to uniquely identify rows in it
- Primary key cab be composed of multiple columns -- composite key
- Collection of column value pairs form a row
- Value of the primary key column is the row key
- In case of composite keys, the value of the 1st column in the composite key becomes the row key

row key value1	column key1	column key2	column key3
	column value1	column value2	column value3

- Cassandra has a built in key cache and row cache
- Key cache holds the location of the keys in memory
- **keys_cached** is the number of keys for which the key cache will hold the location data
- **key_cached** is set to 200000 by default
- row cache holds the entire data of the row in memory
- **row_cached** is the number of rows to be stored in memory
- **reload_row_cache** flag to specify if you want to pre populate the row cache

Create Column family:

```
cassandra@cqlsh:catalog> CREATE COLUMNFAMILY product
  productId varchar,
  title text,
  brand varchar,
  publisher varchar,
  length int,
  breadth int,
  height int,
  PRIMARY KEY(productId)
 );
```

Command to create column family

Verify column family:

```
cassandra@cqlsh:catalog> describe columnfamilies;
```

command to list all column families in the current keyspace

Describe the column family:

```
cassandra@cqlsh:catalog> describe product;
```

this command describes for this columnfamily

1. all columns
2. primary key
3. storage properties

Cassandra uses bloom filter to determine if a key is present in the columns.

Modify Column family:

```
cassandra@cqlsh:catalog> ALTER COLUMNFAMILY product ADD modelId text;
```

Insert Data:

```
cassandra@cqlsh:catalog> insert into product(productid, brand, modelid) values('MOB1
', 'Samsung', 'GalaxyS6');
```

Command to pass timestamp with row data

```
cassandra@cqlsh:catalog> insert into product(productid, title, breadth, length) valu
es('POST2', 'led zeppelin', 22, 36) USING TIMESTAMP 1468580580000;
```

Insert data with Time-To-Live (After TTL the data will automatically get deleted)

```
cassandra@cqlsh:catalog> insert into product(productid, title, breadth, length)
values('POST2', 'adele', 22, 36 ) USING TTL 86400;
```

Display Data:

```
cassandra@cqlsh:catalog> select * from product;
```

Update Data:

```
cassandra@cqlsh:catalog> UPDATE product set modelid='S6' where productid = 'MOB1';

cassandra@cqlsh:catalog> update product set breadth=18, length=25, height=2 where pr
oductid in ('COM1', 'COM2');
```

Collection Data Types:

Set –

Define a set:

```
cassandra@cqlsh:catalog> ALTER COLUMNFAMILY product ADD keyfeatures set<text>;
```

Insert into a set:

```
cassandra@cqlsh:catalog> insert into product(productid, title, brand, keyfeatures)
values('COM1', 'Acer One', 'Acer',{'detachable keyboard','multitouch display'});
```

Add element to set:

```
cassandra@cqlsh:catalog> update  product USING TTL 86400 set keyfeatures = keyfeatur
es + {'FLAT 10% off for 1 day'} where productid in ('COM1', 'COM2');
```

Remove element from set:

```
cassandra@cqlsh:catalog> update product set keyfeatures = keyfeatures - {'detachable
 keyboard'} where productid IN ('COM1', 'COM2');
```

Replace element from set:

Not Supported

List –

Define a list:

```
cassandra@cqlsh:catalog> ALTER COLUMNFAMILY product ADD service_type list<text>
```

Insert into a list:

```
cassandra@cqlsh:catalog> insert into product(productid, title, brand, service_type)
values('SOFA1', 'Urban Living Derby Sofa', 'Urban Living', ['Needs to Call Seller',
'Service Engineer will Come to the Site']);
```

Add element to list:

```
cassandra@cqlsh:catalog> update product set service_type = service_type + ['Service
 engineer will inspect the damages'] where productid='SOFA1';
```

Remove element from list:

```
cassandra@cqlsh:catalog> update product set service_type = service_type - ['Needs to
 Call Seller'] where productid = 'SOFA1';
```

Replace element from list:

```
  cassandra@cqlsh:catalog> update product set service_type[1]='service engineer will c
 all for appointment' where productid='SOFA1';
```

Map –

Define a map:

```
cassandra@cqlsh:catalog> ALTER COLUMNFAMILY product ADD camera map<text,text>;
```

Insert into a map:

```
cassandra@cqlsh:catalog> insert into product(productid, title, brand, camera) values
('COM1', 'Acer One', 'Acer', {'front':'VGA', 'rear':'2MP'} ) ;
```

Add element to map:

```
cassandra@cqlsh:catalog> update product set camera = camera + {'frontVideo':'1MP'} w
here productid = 'COM1' ;
```

Remove element from map:

```
cassandra@cqlsh:catalog> delete camera['frontVideo'] from product where productid =
'COM1';
```

Replace element from map:

```
cassandra@cqlsh:catalog> update product set camera['front'] = '1MP' where productid
= 'COM1' ;
```

Counter –

- It is a special type
- Counter has to be stored in a dedicated column family
- Create a new column family with primary key and only counter column
- Cannot create an index on counter column

Define a counter:

```
cassandra@cqlsh:catalog> create columnfamily productviewcount(productid text primar
y key, viewcount COUNTER);
```

Update counter:

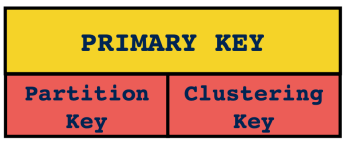
```
cassandra@cqlsh:catalog> update productviewcount set viewcount = viewcount + 1 wher
e productid = 'COM1';
```

Concepts:

Primary Keys →

- Set of columns which can uniquely identify rows
- It also tells Cassandra how to store and distribute data.

A Primary key consists of 2 parts



Partition key decides how the data is distributed

Clustering key decides how the data is stored on the disk

- Partition Key – What nodes in the cluster receives the data for that column family
- Clustering Key – It decides how the data is stored in the disk on a single node
- If there are multiple columns in a primary key (composite key), the first key is the partition key and the remaining columns form the clustering key
- Each node in Cassandra is assigned a unique token
- This token determines which rows this node is going to store
- Token is generated by a hashing function called **Partitioner**
- It generates hash values in the range [-2^63 to +2^63 -1]
- Tokens are distributed among nodes

Single data center

Calculate tokens by dividing the hash range by the number of nodes in the cluster

Multiple data center

First calculate the tokens for each data center

And then divide the tokens among the nodes in the datacenter

- Token generation and distribution logic

Let's say the range for tokens is 1-100

Number of nodes in the cluster is 5

Node 1: Tokens 1-20

Node 2: Tokens 21-40

Node 3: Tokens 41-60

Node 4: Tokens 61-80

Node 4: Tokens 81-100

- Partitioner Implementations:
  - Random Partitioner – Distributes data across the cluster using MD5 hash
  - Murmur3Partitioner – (Default) Distributes data across the cluster using MurmurHash (better performance)
- The columns in the clustering key determine how the data is stored on the disk in a single node
- For a compound primary key which has a partition key and 2 clustering key

PK1	CK-a	CK-b	rest of the columns

- If multiple rows have same partition key – rows are sorted by value of clustering keys

PK1	CK-a1	CK-b1	rest of the columns
	CK-a2	CK-b2	rest of the columns
	CK-a3	CK-b3	rest of the columns

Restriction on Partition Keys →

- *We cannot have a where clause with a single key from a composite partition key*

Ex –

```
cassandra@cqlsh:catalog> CREATE COLUMNFAMILY skus (
       sellerId varchar,
       productId varchar,
       skuId varchar,
       title text,
       listingId varchar,
       isListingCreated boolean,
       timeofskucreation text,
       PRIMARY KEY((sellerId,skuId), timeofskucreation, productId));
```

  - Partition key comprises of skuId and the SellerId

```
cassandra@cqlsh:catalog> select * from skus where sellerid = ('Decor');
```

  - The above line will give an error as in the where clause the skuId is not present.
  - The token for the row is generated by hashing the data of the partition key (i.e., sellerId and skuId).
  - Without a token, coordinator node will not be able to identify the node which owns the partition.

- *IN query will fail if all the columns of the partition key are not restricted (restricted means that all the keys in the partition key needs to be present)*

- *Range Query restriction*
    - Range query using “>” on the partition key

```
cassandra@cqlsh:catalog> select * from skus where sellerId > 'Chroma' and skuid > '
ChromaSKU1';
```

    - We can perform range operations efficiently only if the stored data is sorted
    - Since the partitions are distributed across the cluster, we cannot perform range operations on them

We cannot use > , >=, <=, < operator directly on the partition key

Only IN and = operators are allowed on the partition key

- *Use token function to do a range query.*

```
cassandra@cqlsh:catalog> select * from skus where token(sellerid,skuid) >= token('C
hroma', 'ChromaSKU1') ;
```

- *Order by operation not supported on partition key.*

Restriction on Clustering Keys →

← TODO →

Secondary Indexes:

- For use cases which are beyond primary keys , we use secondary index
- Use to access data using a non-primary column
- Secondary Index are stored in separate column family
- Secondary Index are best used when the indexed column does not have high cardinality (distinct values in a column)
- Secondary indexes are not replicated to other nodes
- Every query on the secondary index is forwarded to all the nodes, results are then merged and returned to client
- Secondary indexes should not be used on counter columns

Restriction on Secondary Indexes →

```
cassandra@cqlsh:catalog> create index listingprodindex on
listings(productid);
```

- With secondary index, either restrict all partition keys or do not restrict partition keys at all
- IN operator is not supported in secondary index
- Range based query not supported, can be used with ALLOW FILTERING but with poor performance
- Use of clustering columns with index is not preferred
- Order By clause not supported with secondary index

- Query with collections using the CONTAINS (for Set) ,CONTAINS KEY (for Map) keyword

Allow Filtering →

```
cassandra@cqlsh:catalog>select * from listings where
productid > 'SOFA' ALLOW FILTERING;
```

- Cassandra will first find and load all the rows in the listing and then filter out the rows where sellerId is lexically <= SOFA

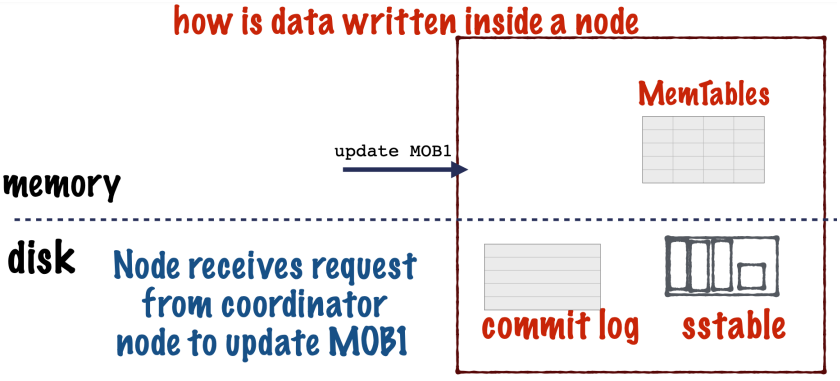
Write Consistency →

- Consistency ensures that the data read from any node in the cluster is the same i.e., consistent
- Read and Write can have separate consistency in Cassandra.
- Write consistency: number of replica nodes on which the write must succeed before returning success to the client
- Read consistency : number of replica nides to check before returning data to client
- Consistency level (WRITE): ONE, ALL,QUORUM, LOCAL_QUORUM
- If a node in the cluster is down then the coordinator keeps the data for that node in a file called as hint file. This mechanism is called as hinted handoff.

Read Consistency →

- Consistency level (WRITE): ONE, ALL,QUORUM, LOCAL_QUORUM
- Read repair is when the data in one of the node is old as compared to the other nodes and the data is updated during a read request.

Storage Components in Cassandra →



- When a Node receives request from a coordinator node to update data
  - Commit log is first to be updated on the disk
  - MemTables (in memory hash tables) are then updated and a response is sent that the write is successful
  - The data is flushed to the SSTable (Sorted String table) at a later point of time which is on the disk