UNIT III NOSQL DATABASES 9

NoSQL – CAP Theorem – Sharding - Document based – MongoDB Operation: Insert, Update, Delete, Query, Indexing, Application, Replication, Sharding–Cassandra: Data Model, Key Space, Table Operations, CRUD Operations, CQL Types – HIVE: Data types, Database Operations, Partitioning – HiveQL – OrientDB Graph database – OrientDB Features

---

Cassandra:

Apache Cassandra is a highly scalable, high performance, distributed NoSQL database. Cassandra is designed to handle huge amounts of data across many commodity servers, providing high availability without a single point of failure.

Cassandra is a NoSQL database

NoSQL database is Non-relational database. It is also called Not Only SQL. It is a database that provides a mechanism to store and retrieve data other than the tabular relations used in relational databases. These databases are schema-free, support easy replication, have simple API, eventually consistent, and can handle huge amounts of data.

Important Points of Cassandra

- Cassandra is a column-oriented database.
- Cassandra is scalable, consistent, and fault-tolerant.
- Cassandra is created at Facebook. It is totally different from relational database management systems.
- Cassandra is being used by some of the biggest companies like Facebook, Twitter, Cisco, Rackspace, ebay, Twitter, Netflix, and more.

Data Model

Data model in Cassandra is totally different from normally we see in RDBMS.

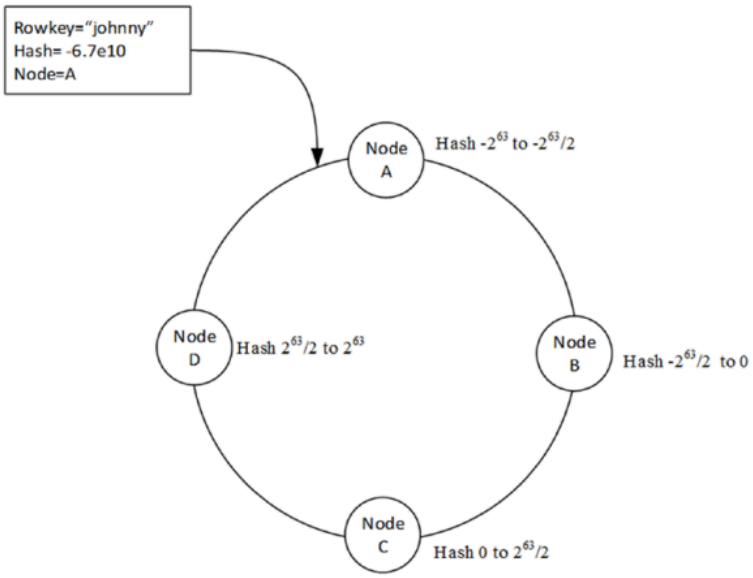
Cluster

Cassandra and other dynamo-based databases distribute data throughout the cluster by using consistent hashing. The rowkey (analogous to a primary key in an RDBMS) is hashed. Each node is allocated a range of hash values, and the node that has the specific range for a hashed key value takes responsibility for the initial placement of that data.

In the default Cassandra partitioning scheme, the hash values range from -263 to 263-1. Therefore, if there were four nodes in the cluster and we wanted to assign equal numbers of hashes to each node, then the hash ranges for each would be approximately as follows:

Node	Low Hash	High Hash
Node A	-2^{63}	$-2^{63}/2$
Node B	$-2^{63}/2$	0
Node C	0	$2^{63}/2$
Node D	$2^{63}/2$	2^{63}

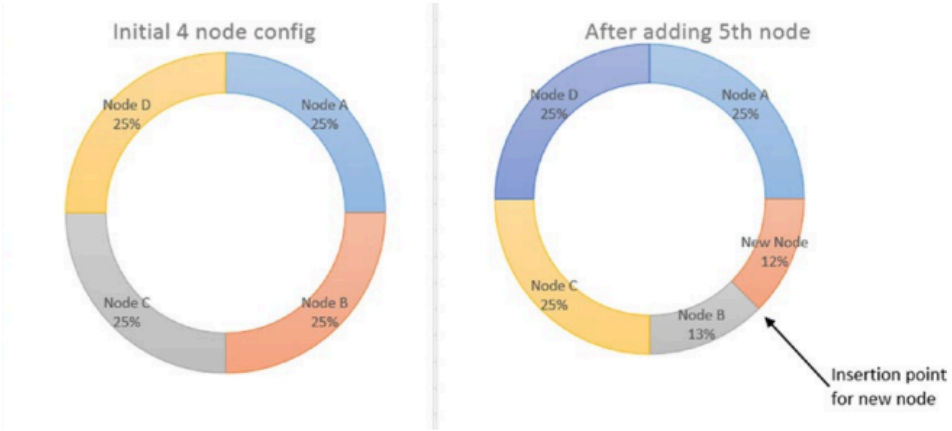
We usually visualize the cluster as a ring: the circumference of the ring represents all the possible hash values, and the location of the node on the ring represents its area of responsibility. Figure illustrates simple consistent hashing: the value for a rowkey is hashed, which determines its position on "the ring." Nodes in the cluster take responsibility for ranges of values within the ring, and therefore take ownership of specific rowkey values.



The four-node cluster in Figure 8-10 is well balanced because every node is responsible for hash ranges of similar magnitude. But we risk unbalancing the cluster as we add nodes. If we double the number of nodes in the cluster, then we can assign the new nodes at points on the ring between existing nodes and the cluster will remain balanced. However, doubling the cluster is usually impractical: it's more economical to grow the cluster incrementally.

Early versions of Cassandra had two options when adding a new node. We could either remap all the hash ranges, or we could map the new node within an existing range. In the first option we obtain a balanced cluster, but only after an expensive rebalancing process. In the second option the cluster becomes unbalanced; since each node is

responsible for the region of the ring between itself and its predecessor, adding a new node without changing the ranges of other nodes essentially splits a region in half. Figure shows how adding a node to the cluster can unbalance the distribution of hash key ranges.



Order-Preserving Partitioning

The Cassandra partitioner determines how keys are distributed across nodes. The default partitioner uses consistent hashing, as described in the previous section. Cassandra also supports order-preserving partitioners that distribute data across the nodes of the cluster as ranges of actual (e.g., not hashed) rowkeys. This has the advantage of isolating requests for specific row ranges to specific machines, but it can lead to an unbalanced cluster and may create hotspots, especially if the key value is incrementing. For instance, if the key value is a timestamp and the order-preserving partitioner is implemented, then all new rows will tend to be created on a single node of the cluster. In early versions of Cassandra, the order-preserving petitioner might be warranted to optimize range queries that could not be satisfied in any other way; however, following the introduction of secondary indexes, the order-preserving petitioner is maintained primarily for backward compatibility, and Cassandra documentation recommends against its use in new applications.

Key Space

Keyspace is the outermost container for data in Cassandra. A keyspace is an object that is used to hold column families, user defined types. A keyspace is like a RDBMS database which contains column families, indexes, user defined types, data center awareness, strategy used in keyspace, replication factor, etc.

Following are the basic attributes of Keyspace in Cassandra:

- **Replication factor:** It specifies the number of machines in the cluster that will receive copies of the same data.

- **Replica placement Strategy:** It is a strategy which specifies how to place replicas in the ring.
- There are three types of strategies such as:
    1) Simple strategy (rack-aware strategy)
    2) old network topology strategy (rack-aware strategy)
    3) network topology strategy (datacenter-shared strategy)

    In Cassandra, "Create Keyspace" command is used to create keyspace.

Cassandra Create Keyspace

Cassandra Query Language (CQL) facilitates developers to communicate with Cassandra. The syntax of Cassandra query language is very similar to SQL. In Cassandra, "Create Keyspace" command is used to create keyspace.

Syntax:

CREATE KEYSPACE <identifier> WITH <properties>

Example:

Let's take an example to create a keyspace named "StudentDB".

CREATE KEYSPACE StudentDB WITH replication = {'class':'SimpleStrategy', 'replication_factor' : 3};

Different components of Cassandra Keyspace

Strategy: There are two types of strategy declaration in Cassandra syntax:

- **Simple Strategy:** Simple strategy is used in the case of one data center. In this strategy, the first replica is placed on the selected node and the remaining nodes are placed in clockwise direction in the ring without considering rack or node location.
- **Network Topology Strategy:** This strategy is used in the case of more than one data center. In this strategy, you have to provide a replication factor for each data center separately.

Using a Keyspace

To use the created keyspace, you have to use the USE command.

Syntax:

USE <identifier>

Cassandra Alter Keyspace

The "ALTER keyspace" command is used to alter the replication factor, strategy name and durable writes properties in created keyspace in Cassandra.

Syntax:

ALTER KEYSPACE <identifier> WITH <properties>

Cassandra Drop Keyspace

In Cassandra, "DROP Keyspace" command is used to drop keyspaces with all the data, column families, user defined types and indexes from Cassandra.

Syntax:

DROP keyspace KeyspaceName ;

Table Operations, - CRUD Operations

Cassandra Create Table

In Cassandra, CREATE TABLE command is used to create a table. Here, column family is used to store data just like table in RDBMS. So, you can say that CREATE TABLE command is used to create a column family in Cassandra.

Syntax:

```
CREATE TABLE tablename(
column1 name datatype PRIMARYKEY,
column2 name data type,
column3 name data type.
)
```

There are two types of primary keys:

1. Single primary key: Use the following syntax for single primary key.
   Primary key (ColumnName)
2. Compound primary key: Use the following syntax for a single primary key.
   Primary key(ColumnName1,ColumnName2 . . .)

Example:

Let's take an example to demonstrate the CREATE TABLE command.
Here, we are using the already created Keyspace "StudentDB".

```
CREATE TABLE student(
student_id int PRIMARY KEY,
student_name text,
student_city text,
student_fees varint,
student_phone varint
);
SELECT * FROM student;
```

Cassandra Alter Table

ALTER TABLE command is used to alter the table after creating it. You can use the ALTER command to perform two types of operations:

- Add a column
- Drop a column

Syntax:

ALTER (TABLE | COLUMNFAMILY) <tablename> <instruction>

Adding a Column

You can add a column in the table by using the ALTER command. While adding column, you have to aware that the column name is not conflicting with the existing column names and that the table is not defined with compact storage option.

Syntax:

ALTER TABLE table name ADD new column datatype;

After using the following command:

ALTER TABLE student ADD student_email text;

A new column is added. You can check it by using the SELECT command.

Dropping a Column

You can also drop an existing column from a table by using ALTER command. You should check that the table is not defined with compact storage option before dropping a column from a table.

Syntax:

ALTER table name DROP column name;

Example:

After using the following command:

ALTER TABLE student DROP student_email;

Now you can see that a column named "student_email" is dropped now. If you want to drop the multiple columns, separate the column name by ",".

Cassandra DROP table

DROP TABLE command is used to drop a table.

Syntax:

DROP TABLE <tablename>

Example:

After using the following command:

DROP TABLE student;

The table named "student" is dropped now. You can use DESCRIBE command to verify if the table is deleted or not. Here the student table has been deleted; you will not find it in the column families list.

Cassandra Truncate Table

TRUNCATE command is used to truncate a table. If you truncate a table, all the rows of the table are deleted permanently.

Syntax:

TRUNCATE <tablename>

Cassandra Batch

In Cassandra BATCH is used to execute multiple modification statements (insert, update, delete) simultaneously. It is very useful when you have to update some column as well as delete some of the existing.

Syntax:

BEGIN BATCH
<insert-stmt>/ <update-stmt>/ <delete-stmt>
APPLY BATCH

Use of WHERE Clause

WHERE clause is used with SELECT command to specify the exact location from where we have to fetch data.

Syntax:

SELECT FROM <table name> WHERE <condition>;
SELECT * FROM student WHERE student_id=2;

Cassandra Update Data

UPDATE command is used to update data in a Cassandra table. If you see no result after updating the data, it means data is successfully updated otherwise an error will be returned. While updating data in Cassandra table, the following keywords are commonly used:

- Where: The WHERE clause is used to select the row that you want to update.
- Set: The SET clause is used to set the value.
- Must: It is used to include all the columns composing the primary key.

Syntax:

UPDATE <tablename>
SET <column name> = <new value>
<column name> = <value>....
WHERE <condition>

Cassandra DELETE Data

DELETE command is used to delete data from Cassandra table. You can delete the complete table or a selected row by using this command.

Syntax:

DELETE FROM <identifier> WHERE <condition>;

Delete an entire row

To delete the entire row of the student_id "3", use the following command:
DELETE FROM student WHERE student_id=3;

Delete a specific column name

Example:

Delete the student_fees where student_id is 4.

DELETE student_fees FROM student WHERE student_id=4;

HAVING Clause in SQL

The HAVING clause places the condition in the groups defined by the GROUP BY clause in the SELECT statement. This SQL clause is implemented after the 'GROUP BY' clause in the 'SELECT' statement. This clause is used in SQL because we cannot use the WHERE clause with the SQL aggregate functions. Both WHERE and HAVING clauses are used for filtering the records in SQL queries.

Syntax of HAVING clause in SQL

SELECT column_Name1, column_Name2, ..., column_NameN aggregate_function_name(column_Name) GROUP BY

Example:

SELECT SUM(Emp_Salary), Emp_City FROM Employee GROUP BY Emp_City;

the following query with the HAVING clause in SQL:

SELECT SUM(Emp_Salary), Emp_City FROM Employee GROUP BY Emp_City HAVING SUM(Emp_Salary)>12000;

MIN Function with HAVING Clause:

If you want to show each department and the minimum salary in each department, you have to write the following query:

SELECT MIN(Emp_Salary), Emp_Dept FROM Employee GROUP BY Emp_Dept;

MAX Function with HAVING Clause:

SELECT MAX(Emp_Salary), Emp_Dept FROM Employee GROUP BY Emp_Dept;

AVERAGE CLAUSE:

SELECT AVG(Emp_Salary), Emp_Dept FROM Employee_Dept GROUP BY Emp_Dept;

SQL ORDER BY Clause

- Whenever we want to sort the records based on the columns stored in the tables of the SQL database, then we consider using the ORDER BY clause in SQL.
- The ORDER BY clause in SQL will help us to sort the records based on the specific column of a table. This means that all the values stored in the column on which we are applying the ORDER BY clause will be sorted, and the corresponding column values will be displayed in the sequence in which we have obtained the values in the earlier step.

Syntax to sort the records in ascending order:

SELECT ColumnName1,...,ColumnNameN FROM TableName ORDER BY ColumnName ASC;

Syntax to sort the records in descending order:

SELECT ColumnName1,...,ColumnNameN FROM TableName ORDER BY ColumnNameDESC;

Syntax to sort the records in ascending order without using ASC keyword:

SELECT ColumnName1,...,ColumnNameN FROM TableName ORDER BY ColumnName;

CQL Types

CQL defines built-in data types for columns. The counter type is unique.

CQL Type	Constants supported	Description
ascii	strings	US-ASCII character string
bigint	integers	64-bit signed long
blob	blobs	Arbitrary bytes (no validation), expressed as hexadecimal
boolean	booleans	true or false
counter	integers	Distributed counter value (64-bit long)
date	strings	Value is a date with no corresponding time value; Cassandra encodes date as a 32-bit integer representing days since epoch (January 1, 1970). Dates can be represented in queries and inserts as a string, such as 2015-05-03 (yyyy-mm-dd)
decimal	integers, floats	Variable-precision decimal
double	integers, floats	64-bit IEEE-754 floating point
float	integers, floats	32-bit IEEE-754 floating point
frozen	User-defined types,collections, tuples	A frozen value serializes multiple components into a single value. Non-frozen types allow updates to individual fields. Cassandra treats the value of a frozen type as a blob. The entire value must be overwritten.
inet	strings	IP address string in IPv4 or IPv6 format, used by the python-cql driver and CQL native protocols

int	integers	32-bit signed integer
list	n/a	A collection of one or more ordered elements: [literal, literal, literal].
map	n/a	A JSON-style array of literals: { literal : literal, literal : literal ... }
set	n/a	A collection of one or more elements: { literal, literal, literal }
smallint	integers	2 byte integer
text	strings	UTF-8 encoded string
time	strings	Value is encoded as a 64-bit signed integer representing the number of nanoseconds since midnight. Values can be represented as strings, such as 13:30:54.234.
timestamp	integers, strings	Date and time with millisecond precision, encoded as 8 bytes since epoch. Can be represented as a string, such as 2015-05-03 13:30:54.234.
timeuuid	uuids	Version 1 UUID only
tinyint	integers	1 byte integer
tuple	n/a	A group of 2-3 fields.
uuid	uuids	A UUID in standard UUID format
varchar	strings	UTF-8 encoded string
varint	integers	Arbitrary-precision integer

Comparison of Cassandra and MongoDB

Sl.No	Cassandra	MongoDB
1.	Cassandra is high performance distributed database system.	MongoDB is cross-platform document-oriented database system.
2.	Cassandra is written in Java	MongoDB is written in C++.
3.	Cassandra stores data in tabular form like SQL format.	MongoDB stores data in JSON format.

4.	Cassandra is licensed by Apache.	MongoDB is licensed by AGPL and drivers by Apache.
5.	Cassandra is mainly designed to handle large amounts of data across many commodity servers.	MongoDB is designed to deal with JSON-like documents and access applications easier and faster.
6.	Cassandra provides high availability with no single point of failure.	MongoDB is easy to administer in the case of failure.
