SQL data base	No SQL database
strudtured	unstructured
Vertical scaling	Horizontal scaling
Third party tolls are needed for backup	Replica sets can be used, highly available
Slower than nosql	Faster since the data is stored in distributed manner
Transactions management e.g bank transaction	Transaction management is difficult Media application

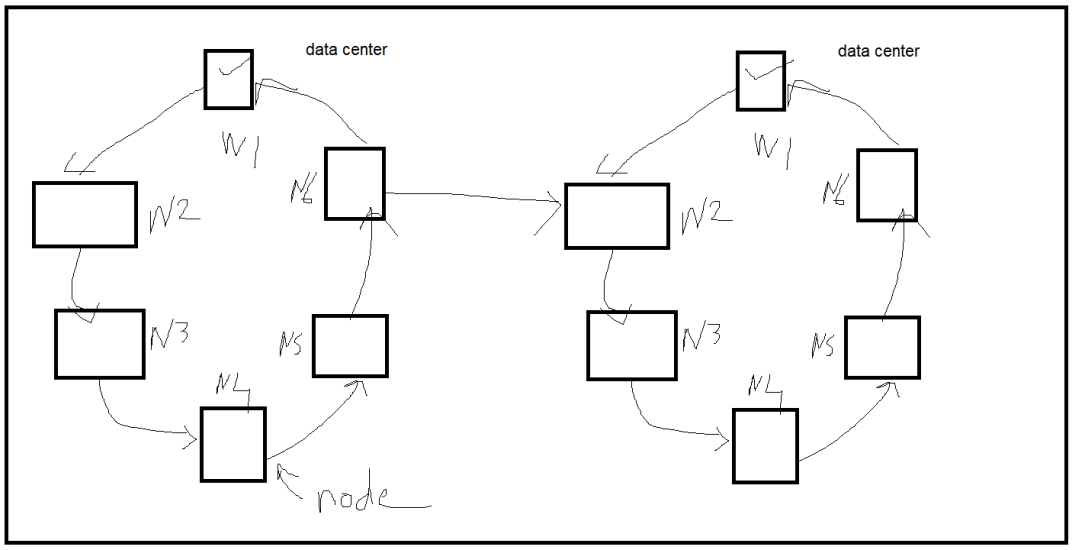
Cassandra

- It's a nosql database, so unstructured
- It is developed by at Facebook by Avinash Lakshman and Prashant malik
- It is written in Java language
- It is available for users since 2008 or 2009

Features

- Highly scalable
- Highly available---Fault tolerant
- Cassandra supports acid property
- Cassandra supports structured as well as unstructured data
- It stores and display data in column based format

Cassandra architecture



Components of cassandra

Data center---- collection of nodes

Node----- A node on which data is stored

Cluster--- contains one or more data centers

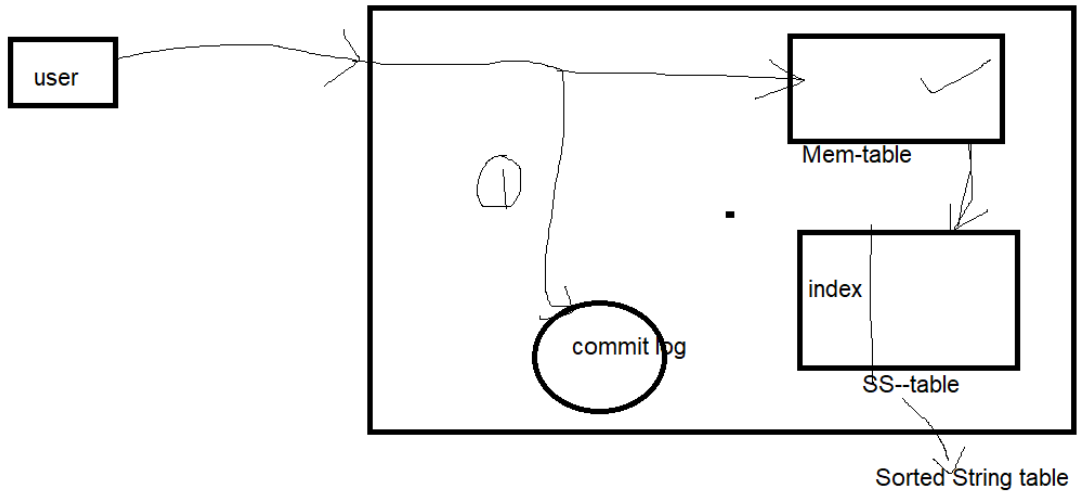
Mem-table---It is a table which resides in RAM. After making entry in commit log the data will be written in mem table. There can be multiple mem tables

SSTable(sorted string table)-----the data will written in SStable when the threshold for mem table is reached. It is a disk file in which data will be stored

Commit log---- every write operation will be written inside this log , helpful for recovery

Bloom filters – these are algorithms to test an element is there in the set of records

Write operations



Read operations

1. Direct request
2. Digest request
3. Read repair request

The CAP theorem: is Cassandra AP or CP?

The famous “CAP” theorem states that a distributed database system can **only guarantee two out of these three characteristics in case of a failure scenario**: Consistency, Availability, and Partition Tolerance:

1. **Consistency:** This means “no stale data.” A query returns the most recent value. If one of the servers returns outdated information, then your system is inconsistent.
2. **Availability:** This basically means “uptime.” If servers fail but still gives a response, then your system is available.
3. **Partition Tolerance:** This is the ability of a distributed system to survive “network partitioning.” Network partitioning means part of the servers cannot reach the second part.

Figure 3. CAP Theorem governing databases.

Any database system, including Cassandra, has to guarantee partition tolerance: It must continue to function during data losses or system failures. To achieve partition tolerance, databases have to either prioritize consistency over availability “CP,” or availability over consistency or “AP”.

Cassandra is usually described as an “AP” system, meaning it errs on the side of ensuring data availability even if this means sacrificing consistency. But that’s not the whole picture. **Cassandra is configurably consistent: You can set the Consistency Level you need and tune it to be more AP or CP according to your use case**

Replication factor—Number of machines in cluster on which the copy should be done

Replica placement strategy

1. Simple strategy
   If you are working with single data centre then use this strategy, replication factor will decide to copy on how many nodes , replication always starts in clockwise direction
2. Network topology
   If you are working with more than 2 data centre then use this strategy, replication factor will decide to copy on how many nodes to write , replication always starts in clockwise direction, till it reaches to next data center

Cassaandra doesnot support joins , group by,aggregate functions

Keyspace is similar to database

And column families are like table

Data types

CQL Type	Constants	Description
ascii	Strings	US-ascii character string
bigint	Integers	64-bit signed long
blob	blobs	Arbitrary bytes in hexadecimal
boolean	Booleans	True or False
counter	Integers	Distributed counter values 64 bit
decimal	Integers, Floats	Variable precision decimal
double	Integers, Floats	64-bit floating point
float	Integers, Floats	32-bit floating point
frozen	Tuples, collections, user defined types	stores cassandra types
inet	Strings	IP address in ipv4 or ipv6 format
int	Integers	32 bit signed integer
list	Duplicate values are allowed , represented in []	Collection of elements
map	Keys are unique, represented in {}	JSON style collection of elements
set	Only unique values are allowed,{}	Collection of elements
text	strings	UTF-8 encoded strings
timestamp	Integers, Strings	ID generated with date plus time
timeuuid	uuids	Type 1 uuid
tuple	Read only and fixed size	A group of 2,3 fields
uuid	uuids	Standard uuid
varchar	strings	UTF-8 encoded string
varint	Integers	Arbitrary precision integer

CQL type compatibility

CQL data types have strict requirements for conversion compatibility. The following table shows the allowed alterations for data types:

Data type may be altered to:	Data type
ascii, bigint, boolean, decimal, double, float, inet, int, timestamp, timeuuid, uuid, varchar, varint	blob
int	varint
text	varchar
timeuuid	uuid
varchar	text

Clustering columns have even stricter requirements, because clustering columns mandate the order in which data is written to disk. The following table shows the allow alterations for data types used in clustering columns:

Data type may be altered to:	Data type
int	varint
text	varchar
varchar	text

To create key space

CREATE KEYSPACE hr WITH replication = {'class': 'SimpleStrategy', 'replication_factor': '1'};

CREATE KEYSPACE hr WITH replication = {'class': 'NetworTopologyStartegy', ‘DC1’:1,’DC2’:3} and durable_writes=false;

By default durable_writes=true; which means that write in the commit log;

To use key space

Use hr;

To create table and insert record in the table

create table customer(cno int primary key, cname text,mobile text)

insert into customer(cno,cname,mobile) values(10,'xxx','33333');

cqlsh:hr> insert into customer(cno,cname) values(11,'yyyy');

to see all the rows

cqlsh:hr> select * from customer;

cqlsh:hr>create table account(accno int primary key,name text,ammount varint);

create table

1. Create table employee(
   Empid int,
   Emp_salary int,
   Emp_lastname varchar,
   Emp_firstname text,
   Emp_dob date,
   Emp_deptno int,
   Emp_comm float,

   Primary key((Empid,Emp_salary),Emp_firstname,emp_dob));

   Partition key - Empid,Emp_salary
   Cluster key- Emp_firstname,emp_dob;

2. To see the details of table
   Describe emp

3. How alter table
   Alter table emp add manager_desc varchar;

   To modify data type of column but the new data type should be compatible with the previous one
   Alter table emp alter manager_desc text;

   To drop column
   Alter table emp drop manager_desc;

4. To drop the table
   Drop table emp;

5. Trucate table

Truncate table emp;

6. Retrieval of the data
   Select * from employee
   Rules for where clause

   1. You have all partition key column only with = operator

   Select * from emp where empid=123 ------error

Select * from emp where empid=123 and emp_salary=2000 ---- right

   2. Use clustering column in the same order as it is defined in the table

   Select * from employee where empid=12 and emp_salary=1200 and emp_dob='2029-12-11'; --- wrong because we are skipping emp_firstname

Select * from employee where empid=12 and emp_salary=1200 and emp_firstname='kishori' and emp_dob='2029-12-11'; ----right

Select * from employee where empid=12 and emp_salary=1200 and emp_dob='2029-12-11' and emp_firstname='kishori'; -----right

Select * from emp where empid=123 and emp_salary=2000 and emp_firstname='kishori' ----right

   3. You cannot use = operator only on non primary key column without partition key and if you want to use it then use "Allow filtering" manditory

Select * from emp

Where emp_comm=100; ----- error

Select * from employee

... Where emp_comm=100 ALLOW FILTERING; ----rigth

   4. You cannot use = operator on only cluster key columns without partition key
      Select * from employee where empid=13 and emp_salary=1110 and emp_firstname='Rajan' and emp_dob='1999-11-11';---right

Select * from emp where emp_first_name="kishori" and emp_dob='2029-12-11'; ---wrong

   5. In operator is allowed on all the columns of partition key but it slows the performance
      Select * from employee where empid in (12,13) and emp_salary in (1200,1110); ---right
      Select * from employee where empid in (12,13) ; ----wrong
      Select * from employee where emp_salary in (1200,1110);----wrong
   6. > , <, <=,>= operators are not allowed on partition key

   Select * from emp where empid = 1100 and emp_salary = 1200

7. > , <, <=,>= operators can be used only on cluster key columns, and partition key columns

Select * from emp

Where empid=100 and emp_salary=1002 and emp_firstname=”xxxx” and emp_dob=’1999-12-11’ and emp_comm>1; ----wrong

Select * from employee

Where empid=12 and emp_salary=1110 and emp_firstname='kishori' and emp_dob>'1998-12-11';

To update the data

Update accounts

Set balance=4444

Where acid=222;

delete acname from account where acid=222;

cqlsh:hr> select * from account;

```
 acid | acname | balance
------+--------+---------
  111 |    xxx |    4444
  222 |   null |    3456
  333 |    xxx |    3456

(3 rows)
```

cqlsh:hr> delete from account where acid=222;

cqlsh:hr> select * from account;

```
 acid | acname | balance
------+--------+---------
  111 |    xxx |    4444
  333 |    xxx |    3456

(2 rows)
```

cqlsh:hr>

Order by can used only on cluster key but where clause should have equality condition based on partition key;

select * from employee where empid=13 and emp_salary=1110 order by emp_firstname desc,emp_dob desc;

select * from employee where empid=13 and emp_salary=1110 order by emp_firstname desc;

select * from employee where empid=13 and emp_salary=1110 order by emp_dob desc;---wrong

Data type

Collection

1. Set ---
    a. stores unique values
    b. Represent in {}
    c. Mutable
    d. No indexing is possible
2. List
3. Map
4. Tuple

Create table student(

Id int primary key,

Name text,

Courses set<text>);

Insert into student values(123,'Rajas',{'Python','Perl','PHP'})

1. To overwrite existing courses

Update student

Set courses={'a','b','c','d'}

Where id=131sdfj

2. To add in existing courses

Update student

Set courses=courses+{'a','b','c','d'}

Where id=131sdfj

1. To delete from the set

Update student

Set courses=courses-{python}

Where id=131sdfj

2. To remove all elements from the set

Update student

Set courses={}

Where id=131sdfj

Delete courses from student where id=131sdfj

3. List
    a. Uses []
    b. Duplicates are allowed
    c. Ordered collection, indexing is possible
    d. Mutable

Create table emp11(

Id int primary key,

Name text,

Companies list<text>)

1. To insert data

Insert into employee(id,name,companies) values(11,'Rajan',['cognizant','cabgemin','Tech-M'])

2. Update list add at the beginning
   Update employee set companies=['Wipro']+companies where id=1

3. Update list add at the end
   Update employee set companies=companies+['Wipro'] where id=1

4. Update list at particular location, value will be overwritten
   Update emp11 set companies[2]= 'Wipro' where id=1

5. to remove the element from particular position
   delete companies[2] from employee where id=1;
   To remove the specified element from the list if you know the value
   Update employee set companies[2]= ['Wipro'] where id=1

4. Map ------
    a. Store data in key value format
    b. Keys should be unique
    c. Data is stored in {}
    d. Values can bw retrieved by using keys.

Create table empdata(id int, name text, companies map<text,int>)

1. Insert into empdata(id,name,companies) values(111,'xxx',{'cognizant':5,'igate':3})
2. Update companies to add new key value pair
   Update empdata set companies=companies+{'acceture':5}
3. Update value of a particular key in maps

Update empdata set companies[‘igate’]=5 where id=111;

4. Delete a particular key value pair
   Delete companies[‘igate’] from employee where id=111;
5. Delete multiple keys also
   Update set companies=companies-{‘cognizant’,’igate’} where id=111;

Tuple

1. Duplicates are allowed
2. Uses()
3. Ordered collection,Indexing is possible
4. Tuples are immutable

Create table studdata(id int primary key,name text, marks tuple<int,text,int>);

insert into studdata(id,name,marks)values(1,'a',(1,'a',1))

to update marks for id 1

update studdata set marks=(1,’xx’,2);

Create table trainee(id int primary key,name text, marks tuple<int,text,tuple<int,int,int>>);

insert into trainee(id,name,marks) values(11,'Revati',(90,'A',(20,40,80)))

delete marks from trainee where id=11;

update trainee set marks=(10,'a',(12,13,15)) where id=11;

----------create user defined data type

Create type address(street text,zipcode text,city text)

To create a table friend

Create table friend(fid int, name text,loc address);

Insert into friend(fid,name,loc) values(12,’Ashwini’,{street:’kothrud’, zipcode:’234456’, city:’Pune’})

Use FROZEN keyword when in user defined data type part of the type is not allowed to change but entire value can be overwritten

Create table friend(fid int, name text,loc FROZEN<address>);

Create table friendtest(fid int, name text,loc LIST<FROZEN<address>>);

To alter the data type

Alter type address add country text;

To rename field of a type

Alter type address rename country to cont;

Batch operations in cassandara

```
Begin batch
 Insert into empdata(id,name,companies) values(111,'xxx',{'cognizant':5,'igate':3});
Insert into empdata(id,name,companies) values(112,yyy',{'TechM':5,'igate':3});
Delete companies from empdata where id=123;
Apply batch;
```

Indexes in Cassandra

```
Create index idxname on empdata(name)
```

To add data in json format

```
INSERT INTO Registration JSON'
{
"Emp_id": 2000,
"current_address":  { "Emp_id":1,"h_no" : "A 210", "city" :
                    "delhi", "state" : "DL",
                    "pin_code" :"201307",
                    "country" :"india"},
"Emp_Name": "Ashish Rana"}' ;
```