



# CASSANDRA

---

Ashok Mahajan

Yahoo! DBA Team

3/28/2013

## Contents

Strengths of RDBMS.....	3
Constraints of RDBMS.....	3
NoSQL Database.....	3
Polyglot Persistence .....	4
Aggregate Oriented data model .....	5
Column Family Data Store .....	5
Cassandra .....	5
Cassandra Data Model .....	6
Column .....	6
Row .....	6
Column Family .....	6
Keyspace .....	7
Comparison of Cassandra with RDBMS: .....	7
Cassandra Architecture.....	7
Data partitioning .....	8
Features .....	8
Write operation .....	8
Replication Factor .....	8
Consistency Level .....	9
Transactions:.....	9
Possible Use Cases: .....	9
Testing.....	9
Setup of 3 node cluster .....	10
Create keyspace and Column family.....	12
Inserting and reading the data in Column family .....	12
Adding a column in column family.....	14
Availability.....	15
Deleting rows and columns from column family .....	16
Expiring Column Usage .....	16
Changing the replication factor .....	17

## Strengths of RDBMS

For years RDBMS have been the default choice for data store in enterprise application. They are powerful technologies. Due to their familiarity, stability, rich feature set and available support, they will continue to be the choice of data storage in future too. The strengths of RDBMS can be summarized as following:

- 1) Getting the persistent data: RDBMS allows more flexibility than file system in storing large amount of data in a way which helps in faster retrieval.
- 2) Concurrency and consistency which are mandatory OLTP database requirements.
- 3) Integration: Usually multiple application access the one single RDMS database making it a point of integration for data within an enterprise.
- 4) Standard model: Products offered by different vendors work in very similar way (Mysql, Oracle, Mssql server)

## Constraints of RDBMS

- 1) Processing of large scale data: Organizations these days are capturing more data and want to process it quickly. RDBMS offer many features like durability, consistency, range scan queries, query optimization. Each feature adds to cost of processing. Many of these features may not be required while processing big data.
- 2) RDBMS is primarily designed to run on single machine and scale vertically. Though we have Oracle RAC and Mysql server as clustered RDBMS, but they have disks as the single point of failure. Also the license cost is higher.
- 3) Even if we plan to partition data in RDBMS, It has to be done at application level, which is not convenient.
- 4) Usually it is more economic al to process the large data on a cluster of cheaper machines.
- 5) Application development productivity: The data in RDBMS has to be stored in relational tables (rows and columns). It cannot hold complex data structures like nested records and lists. However the memory can hold much richer data structure (xml or json documents): this has to be converted into relational compatible format. This is time consuming part of application development process.

## NoSQL Database

NoSQL is a vaguely defined term. However most of the databases which fall under this category have following common characteristics:

- 1) Not using the relational model

- 2) Designed to run on clusters
- 3) Schema less
- 4) Generally open source

## Polyglot Persistence

It is an accepted best practice that an enterprise application should be written in a mix of languages to take advantage of the fact that different languages are suitable for tackling different problems.

In last few years, data storage paradigm is also shifting in same direction. Polyglot persistence is nothing but using multiple data storage technologies, chosen based upon the way data is being used by individual applications.

In current database world, a typical web application data store may look like the following diagram. This data store is using Redis, Riak, Neo4J, MongoDB, Cassandra for different type of applications in addition to RDBMS which is being used for financial data and reporting.



## Aggregate Oriented data model

The relational database model takes the information we want to store and divides it into tables. Data is stored in normalized form. The relationship within the table is captured by foreign key relationships. It captures the data in the set of simple values. This data model facilitates ACID compliance.

On the other hand, aggregate oriented data model recognizes that we will access the data in units which are more complex than simple rows. Data which is accessed together, it is stored together. This data structure is very convenient for running the database on a cluster. It greatly improves simple read and write operation which are limited to single aggregate.

However, using aggregate introduces redundancy of data. It is difficult to manage the transactions which span multiple aggregates. This constraint is an obstacle to acid compliance.

Aggregate oriented databases fall into following three categories:

- 1) Key value store
- 2) Document store
- 3) Column family store

## Column Family Data Store

Column family databases store data in column families as rows that have many columns associated with row key. Column families are group of related data that is often accessed together. Though the terms columns and row sound similar to a RDBMS, but there are significant differences (section “Comparison of Cassandra with RDBMS”).

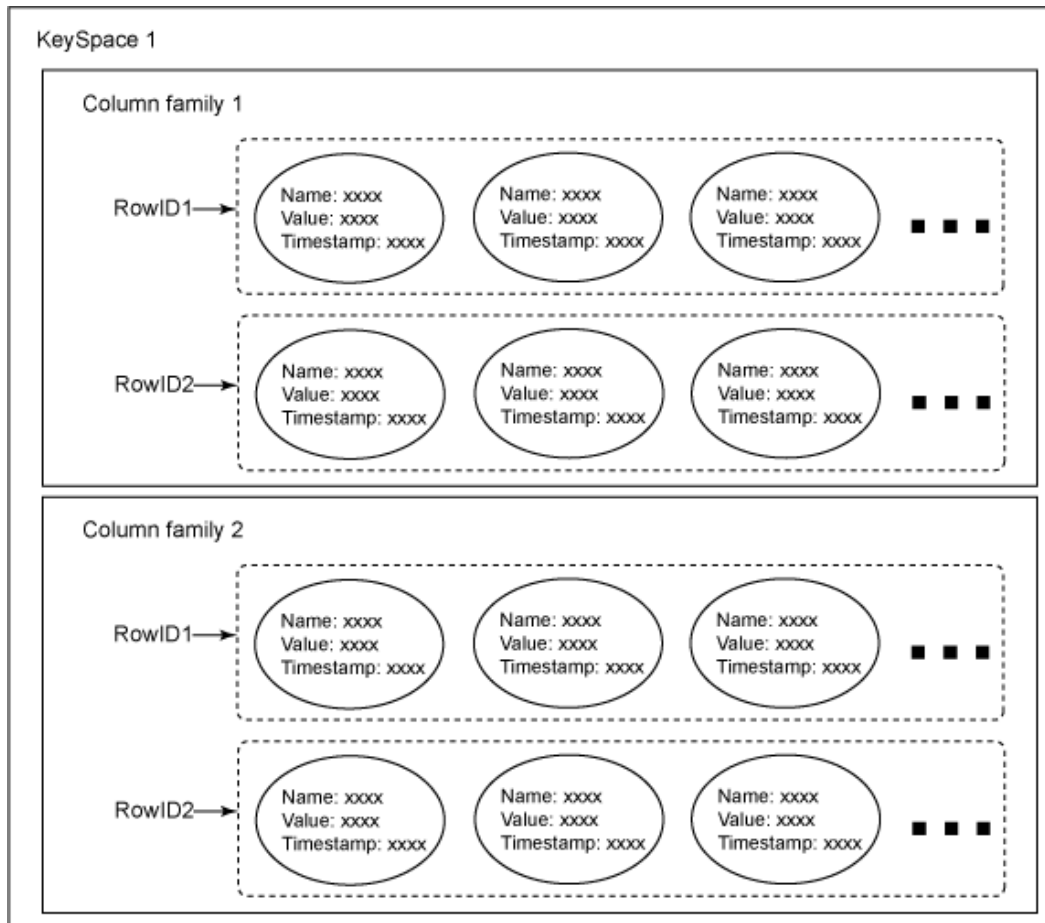
## Cassandra

Cassandra is one of the popular column family databases. It is a distributed database system. It can be described as fast and easily scalable with write operations spread across the cluster. The cluster doesn't have a master node. So any read and write can be handled by any node in the cluster.

Some of the strong points of Cassandra are:

- Highly scalable and highly available with no single point of failure
- Very high write throughput and good read throughput
- SQL-like query language (since 0.8)
- Support search through secondary indexes
- Tunable consistency and support for replication
- Flexible schema

## Cassandra Data Model



### Column

Basic unit of storage in Cassandra is column. Cassandra column consists of column name and column value. It is treated as a key value pair where column name behaves like a key. Value is always stored with timestamp. Timestamp is used to expire the data, resolve the conflicts or deal with the stale data.

### Row

A row is collection of columns, attached or linked to row key.

### Column Family

Collection of similar rows is column family. Column family is similar to a table in RDBMS. However, the scheme of each row can be different i.e. different rows in same column family can have different number and types of column.

## Keyspace

Cassandra puts all the column families related to an application in single keyspace, which is equivalent to schema in RDBMS.

Following table compares the data model of Cassandra with Oracle.

RDBMS (Oracle)	Cassandra
Database Instance	Cluster (Group of nodes)
Schema	Keyspace
Table	Column Family
Row	Row
Column (Same for all rows)	Column (Can be different for different rows)

## Comparison of Cassandra with RDBMS:

Though a column family looks very similar to a table in RDBMS, there are significant differences:

- Relational columns are homogeneous across all rows in the table. A clear vertical relationship usually exists between data items. That is not the case with Cassandra columns. This is the reason Cassandra stores the column name with each data item (column).
- With the relational model, 2D data space is complete. Each point in the 2D space should have at least the null value stored there. Again, this is not the case with Cassandra, and it can have rows containing only a few items, while other rows can have millions of items.
- With a relational model, the schema is predefined and cannot be changed at runtime, while Cassandra lets users change the schema at runtime.
- Relational models support sophisticated queries that include JOINS, aggregations, and more. With a relational model, users can define the data schema without worrying about queries. Cassandra does not support JOINS and most SQL search methods. Therefore, schema has to be catered to the queries required by the application.

## Cassandra Architecture

A Cassandra instance is a collection of independent nodes that are configured together into a cluster. In a Cassandra cluster, all nodes are peers, meaning there is no master node or centralized management process.

Cassandra uses a protocol called **gossip** to discover location and state information about the other nodes participating in a Cassandra cluster. Gossip is a peer-to-peer communication protocol in which nodes periodically exchange state information about themselves and about other nodes they know about.

In Cassandra, the gossip process runs every second and exchanges state messages with up to three other nodes in the cluster. The nodes exchange information about themselves and about the other nodes that they have gossiped about, so all nodes quickly learn about all other nodes in the cluster. A gossip message has a version associated with it, so that during a gossip exchange, older information is overwritten with the most current state for a particular node.

When a node first starts up, it looks at its configuration file to determine the name of the Cassandra cluster it belongs to and which node(s), called seeds, to contact to obtain information about the other nodes in the cluster. These cluster contact points are configured in the `cassandra.yaml` configuration file for a node.

## Data partitioning

In Cassandra, the total data managed by the cluster is represented as a circular space or **ring**. The ring is divided up into ranges equal to the number of nodes, with each node being responsible for one or more ranges of the overall data. Before a node can join the ring, it must be assigned a token. The token determines the node's position on the ring and the range of data it is responsible for.

Column family data is partitioned across the nodes based on the row key. To determine the node where the first replica of a row will live, the ring is walked clockwise until it locates the node with a token value greater than that of the row key. Each node is responsible for the region of the ring between itself and its predecessor. With the nodes sorted in token order, the last node is considered the predecessor of the first node; hence the ring representation.

## Features

### Write operation

Any write operation is first written into commit log which are similar to redo logs in Oracle. Then it is written to memtable. Once the write operation is through with commit log and memtable, write is considered as successful. Writes are batched in the memory and periodically flushed into the SSTable. If a node goes down, the commit log is used to apply the change to node just like the redo log in Oracle.

### Replication Factor

Replication factor determines the number of replicas Cassandra will maintain for a keyspace. During the keyspace creation we specify the replication factor. A replication factor of three ensures that the data will be replicated in three nodes. All replicas are equally important; there is no primary or master replica. Default strategy places the first replica on a node determined by the partitioner. Additional replicas are placed on the next nodes clockwise in the ring.



## Consistency Level

Consistency level determines how many replicas should respond to declare a read or write operation successful. We can specify the consistency level of each operation depending on the application requirements.

There are three consistency levels used:

**One:** With consistency setting one, a write operation is successful if Cassandra is able to write in the commit log of one node. During read operation, Cassandra returns the database from the first replica even if the data is stale. The stale data will be made refresh using a mechanism call Read Repair.

**All:** Consistency level of ALL means that all the nodes having replica of data have to participate in the read and write operation. Even if one node is not available, operation will fail.

**Quorum:** Read or write operation is successful when  $RF/2+1$  (RF is replication factor) replica respond to the request. In other words majority of replica nodes respond to the request.

Therefore the consistency level and replication factor directly impacts the availability of data. Low consistency level can be used when we require high performance and tolerate some data loss.

## Transactions:

Cassandra doesn't have the concept of transactions. A write is atomic at the row level. That means inserting or updating columns for a given row key will be treated as a single write and will either succeed or fail.

## Possible Use Cases:

Event logging system: With its ability to store any data structures, they good choice to store event information. All the application can write their event to Cassandra with their own columns with row key as application name.

Blog entries: Using column families we can store blog entries with tags, categories and links in different columns of a row.

Expiring Usage: Cassandra has expiring columns. As every column in Cassandra is stored time stamp, the expiring columns get deleted after predefined period of time. This can be used to show ad banners on a website for a specific time period or provide demo access to users for a limited period of time.

## Testing

This section provides details about setting up a three node Cassandra cluster and exploring the basic features described in previous sections.

## Setup of 3 node cluster

Steps to setup a same 3 node Cassandra cluster:

Hosts:

sp2-ccdwdb-001.corp.sp2.yahoo.com (66.94.255.235)  
sp2-ccdwdb-002.corp.sp2.yahoo.com (66.94.255.236)  
sp2-ccdwdb-003.corp.sp2.yahoo.com (66.94.255.237)

Prerequisites:

- 1) Execute the command `python -V` to check the version of python. If the version is lower than 2.6, install Python version 2.6 or above. Most of the servers in Yahoo! already have this required version of Python.

Download Python 2.7.3 tar file from the website.

Extract the tar file in the home directory and go into the directory `Python-2.7.3`

Execute following commands in `${HOME}/Python-2.7.3`

```
./configure
./make
./make install
-bash-3.2$ python -V
Python 2.7.3
```

- 2) Install Java Java version 1.6 or above:

Download the rpms package for Java 6 and above and install them.

```
-rw-r--r-- 1 amahajan users 21M Feb 20 12:10 jre-6u32-linux-amd64.rpm
-rw-r--r-- 1 amahajan users 56M Feb 20 12:10 jdk-6u32-linux-amd64.rpm
```

Add following environment variables in `.bash_profile`:

```
export JAVA_HOME=/usr/java/jdk1.6.0_32
export PATH=$JAVA_HOME:$PATH
```

- 3) Install Cassandra on individual nodes.

Download Cassandra tar file (`apache-cassandra-1.2.1-bin.tar`), Extract the file in the home directory. Execute following command to start the Cassandra server:

**`bin/cassandra -f`**

This will bring up the single node Cassandra instance on the node.

Now we need to convert the 3 single node Cassandra clusters into one 3-node cluster.

- 4) Edit the configuration file `cassandra.yaml` on all nodes as following:

Node 1 (On `sp2-ccdwdb-001.corp.sp2.yahoo.com`, 66.94.255.235)

```
-bash-3.2$ pwd
/home/amahajan/apache-cassandra-1.2.1/conf
-bash-3.2$ hostname
sp2-ccdwdb-001.corp.sp2.yahoo.com
-bash-3.2$ cat cassandra.yaml | grep -i "66.94"
    - seeds: "66.94.255.235"
listen_address: 66.94.255.235
rpc_address: 66.94.255.235
```

Node 2 (On `sp2-ccdwdb-002.corp.sp2.yahoo.com`, 66.94.255.236)

```
-bash-3.00$ pwd
/home/amahajan/apache-cassandra-1.2.1/conf
-bash-3.00$ hostname
sp2-ccdwdb-002.corp.sp2.yahoo.com
-bash-3.00$ cat cassandra.yaml | grep -i 66.94
    - seeds: "66.94.255.235"
listen_address: 66.94.255.236
rpc_address: 66.94.255.236
```

Node 3 (On `sp2-ccdwdb-003.corp.sp2.yahoo.com`, 66.94.255.237)

```
-bash-3.2$ pwd
/home/amahajan/apache-cassandra-1.2.1/conf
-bash-3.2$ hostname
sp2-ccdwdb-003.corp.sp2.yahoo.com
-bash-3.2$ cat cassandra.yaml | grep -i 66.94
    - seeds: "66.94.255.235"
listen_address: 66.94.255.237
rpc_address: 66.94.255.237
```

The value of parameter `seeds` is same on all three nodes, which binds the Cassandra instance on three nodes into a single cluster with first node as seed node.

- 5) Stop the Cassandra instances on individual nodes and restart it. The seed node (node 1 in our case) has to be started first.

```
sp2-ccdwdb-001.corp.sp2.yahoo.com - PuTTY
-bash-3.2$ pwd
/home/amahajan
-bash-3.2$ cd apache-cassandra-1.2.1/bin
-bash-3.2$ ./nodetool -host sp2-ccdwdb-001.corp.sp2.yahoo.com ring
Note: Ownership information does not include topology; for complete information, specify a keyspace

Datacenter: datacenter1
=====
Address          Rack      Status State   Load        Owns
  Token
66.94.255.236    rack1    Up      Normal  98.57 KB     32.10%
-807260055033509985
66.94.255.237    rack1    Up      Normal  100.84 KB    63.35%
3612905708855427759
66.94.255.235    rack1    Up      Normal  90.05 KB     4.56%
4453399640808141293
-bash-3.2$ █
```

## Create keyspace and Column family

In following example we are creating a keyspace (ie schema) DEMO. Under DEMO we are creating a column family (table) user.

```
[default@unknown] create keyspace DEMO;
b755858e-48c0-3d57-b072-f7fdeb35684f
```

```
[default@unknown] use demo;
Authenticated to keyspace: DEMO
```

```
[default@DEMO] create column family Users with key_validation_class = 'UTF8Type' and comparator =
'UTF8Type' and default_validation_class = 'UTF8Type';
7674898e-e456-3e46-ba11-fad227c5ec40
```

## Inserting and reading the data in Column family

Inserting and reading the data in column family is done through set and get commands:

```
[default@DEMO] set Users[1234][name] = scott;
Value inserted.
```

Elapsed time: 53 msec(s).

[default@DEMO] set Users[1234][password] = tiger;

Value inserted.

Elapsed time: 2.38 msec(s).

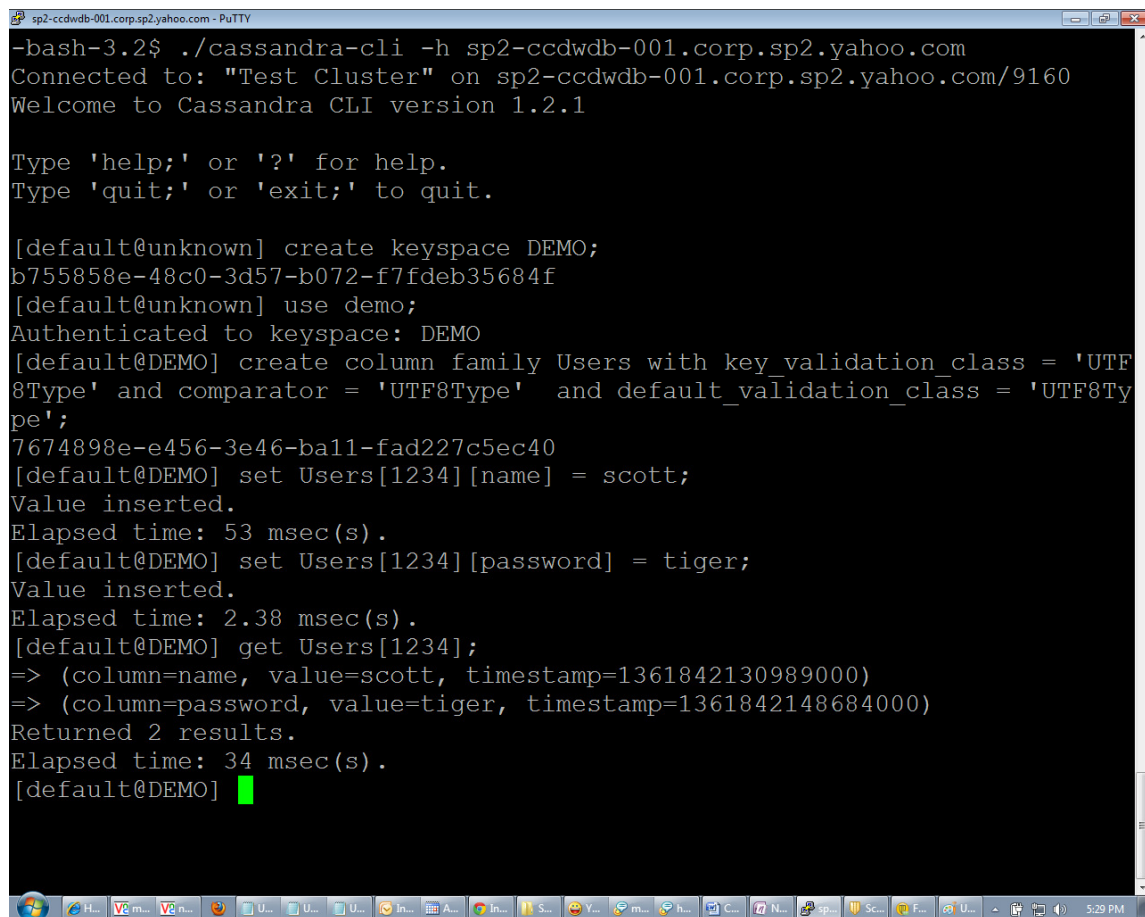
[default@DEMO] get Users[1234];

=> (column=name, value=scott, timestamp=1361842130989000)

=> (column=password, value=tiger, timestamp=1361842148684000)

Returned 2 results.

Elapsed time: 34 msec(s).



```
sp2-ccdwdb-001.corp.sp2.yahoo.com - PuTTY
-bash-3.2$ ./cassandra-cli -h sp2-ccdwdb-001.corp.sp2.yahoo.com
Connected to: "Test Cluster" on sp2-ccdwdb-001.corp.sp2.yahoo.com/9160
Welcome to Cassandra CLI version 1.2.1

Type 'help;' or '?' for help.
Type 'quit;' or 'exit;' to quit.

[default@unknown] create keyspace DEMO;
b755858e-48c0-3d57-b072-f7fdeb35684f
[default@unknown] use demo;
Authenticated to keyspace: DEMO
[default@DEMO] create column family Users with key_validation_class = 'UTF8Type' and comparator = 'UTF8Type' and default_validation_class = 'UTF8Type';
7674898e-e456-3e46-ba11-fad227c5ec40
[default@DEMO] set Users[1234][name] = scott;
Value inserted.
Elapsed time: 53 msec(s).
[default@DEMO] set Users[1234][password] = tiger;
Value inserted.
Elapsed time: 2.38 msec(s).
[default@DEMO] get Users[1234];
=> (column=name, value=scott, timestamp=1361842130989000)
=> (column=password, value=tiger, timestamp=1361842148684000)
Returned 2 results.
Elapsed time: 34 msec(s).
[default@DEMO] █
```

## Adding a column in column family

If the application needs to store an additional column for an incoming record, this can be done on the fly. There is no need to first change the metadata. In following example for the key 1235, we are storing age also, which has not been stored in previous records.

```
[default@DEMO] set Users[1235][name] = John;  
Value inserted.  
Elapsed time: 27 msec(s).  
[default@DEMO] set Users[1235][password] = tiger;  
Value inserted.  
Elapsed time: 1.75 msec(s).  
[default@DEMO] set Users[1235][age] = 21;  
Value inserted.  
Elapsed time: 2.44 msec(s).
```

```
[default@DEMO] list users;  
Using default limit of 100  
Using default column limit of 100
```

```
-----  
RowKey: 1234  
=> (column=name, value=scott, timestamp=1361842130989000)  
=> (column=password, value=tiger, timestamp=1361842148684000)  
-----
```

```
RowKey: 1235  
=> (column=age, value=21, timestamp=1361843342210000)  
=> (column=name, value=John, timestamp=1361843321931000)  
=> (column=password, value=tiger, timestamp=1361843331386000)
```

```
2 Rows Returned.  
Elapsed time: 39 msec(s).
```

```
sp2-ccdwdb-001.corp.sp2.yahoo.com - PuTTY
-bash-3.2$ ./cassandra-cli -h sp2-ccdwdb-001.corp.sp2.yahoo.com
Connected to: "Test Cluster" on sp2-ccdwdb-001.corp.sp2.yahoo.com/9160
Welcome to Cassandra CLI version 1.2.1

Type 'help;' or '?' for help.
Type 'quit;' or 'exit;' to quit.

[default@unknown] use demo;
Authenticated to keyspace: DEMO
[default@DEMO] list users;
Using default limit of 100
Using default column limit of 100
-----
RowKey: 1234
=> (column=name, value=scott, timestamp=1361561160745000)
=> (column=password, value=tiger, timestamp=1361561192048000)
-----
RowKey: 1235
=> (column=age, value=21, timestamp=1361561632324000)
=> (column=name, value=Kapil, timestamp=1361561597213000)
=> (column=password, value=changemenow, timestamp=1361561617375000)

2 Rows Returned.
Elapsed time: 39 msec(s).
[default@DEMO] █
```

## Availability

The data being inserted into the column family is being replicated transparently in other nodes as per the replication factor. In our example, the data in column family users will be available through node 2 and node 3 even if the node 1 goes down which was used to insert the data:

### On second (sp2-ccdwdb-002.corp.sp2.yahoo.com)

Connected to: "Test Cluster" on 66.94.255.236/9160  
Welcome to Cassandra CLI version 1.2.1

Type 'help;' or '?' for help.  
Type 'quit;' or 'exit;' to quit.

```
[default@unknown] use demo;
Authenticated to keyspace: DEMO
[default@DEMO] list users;
Using default limit of 100
Using default column limit of 100
-----
RowKey: 1234
=> (column=name, value=scott, timestamp=1361842130989000)
```

```
=> (column=password, value=tiger, timestamp=1361842148684000)
```

-----

RowKey: 1235

```
=> (column=age, value=21, timestamp=1361843342210000)
```

```
=> (column=name, value=John, timestamp=1361843321931000)
```

```
=> (column=password, value=tiger, timestamp=1361843331386000)
```

2 Rows Returned.

**Elapsed time: 59 msec(s).**

## Deleting rows and columns from column family

```
[default@DEMO] del users[1235][password];
```

column removed.

Elapsed time: 9.9 msec(s).

```
[default@DEMO] get users[1235];
```

```
=> (column=age, value=23, timestamp=1361911177808000)
```

```
=> (column=name, value=John, timestamp=1361843321931000)
```

Returned 2 results.

**Elapsed time: 2.69 msec(s).**

## Expiring Column Usage

The time to live (TTL) is a setting that makes a column self delete a specified number of seconds after the insertion time. In following example we are updating a column password with TTL value of 10 seconds. After 10 seconds of doing so, column is automatically deleted.

```
[default@DEMO] get users[1237];
```

```
=> (column=age, value=22, timestamp=1364344866405000)
```

```
=> (column=name, value=Andy, timestamp=1364344795404000)
```

Returned 2 results.

```
[default@DEMO] set users[1237][password]='abc123' with ttl = 10;
```

Value inserted.

Elapsed time: 2.02 msec(s).

```
[default@DEMO] get users[1237];
```

```
=> (column=age, value=22, timestamp=1364344866405000)
```

```
=> (column=name, value=Andy, timestamp=1364344795404000)
```

```
=> (column=password, value=abc123, timestamp=1364345048166000, ttl=10)
```

Returned 3 results.

Elapsed time: 2.08 msec(s).

```
[default@DEMO] get users[1237];
```

```
=> (column=age, value=22, timestamp=1364344866405000)
```

```
=> (column=name, value=Andy, timestamp=1364344795404000)
```



=> (column=password, value=abc123, timestamp=1364345048166000, ttl=10)

Returned 3 results.

Elapsed time: 2.45 msec(s).

**[default@DEMO] get users[1237];**

**=> (column=age, value=22, timestamp=1364344866405000)**

**=> (column=name, value=Andy, timestamp=1364344795404000)**

**Returned 2 results.**

**Elapsed time: 1.69 msec(s).**

## Changing the replication factor

The replication factor of the keyspace can be changed. However, we need to execute the repair command to align the existing data with the new replication factor.

```
[default@unknown] update keyspace DEMO with placement_strategy =  
'org.apache.cassandra.locator.SimpleStrategy' and strategy_options = {replication_factor:3};  
01694cdd-eda2-3060-bd05-15166a6b7115  
[default@unknown]
```

```
-bash-3.2$ ./nodetool -h 66.94.255.235 repair  
[2013-03-18 15:01:10,339] Nothing to repair for keyspace 'system'  
[2013-03-18 15:01:10,345] Starting repair command #1, repairing 1 ranges for keyspace  
schema1  
[2013-03-18 15:01:10,377] Repair command #1 finished  
[2013-03-18 15:01:10,382] Starting repair command #2, repairing 3 ranges for keyspace DEMO  
[2013-03-18 15:01:11,217] Repair session 57a820e0-9017-11e2-9649-e7a9f6a7d1dd for range  
(4453399640808141293,-8072600550335099985] finished  
[2013-03-18 15:01:11,218] Repair session 57f4e150-9017-11e2-9649-e7a9f6a7d1dd for range (-  
8072600550335099985,3612905708855427759] finished  
[2013-03-18 15:01:11,283] Repair session 58190b20-9017-11e2-9649-e7a9f6a7d1dd for range  
(3612905708855427759,4453399640808141293] finished  
[2013-03-18 15:01:11,283] Repair command #2 finished  
[2013-03-18 15:01:11,289] Nothing to repair for keyspace 'system_auth'  
[2013-03-18 15:01:11,293] Nothing to repair for keyspace 'system_traces'  
-bash-3.2$
```

```
-bash-3.00$ ./nodetool -h 66.94.255.236 repair  
[2013-03-18 15:42:37,237] Nothing to repair for keyspace 'system'  
[2013-03-18 15:42:37,258] Starting repair command #1, repairing 1 ranges for keyspace  
schema1  
[2013-03-18 15:42:37,261] Repair command #1 finished
```

```
[2013-03-18 15:42:37,272] Starting repair command #2, repairing 3 ranges for keyspace DEMO
[2013-03-18 15:42:37,691] Repair session 21f56d80-901d-11e2-9b70-212e9e5f89e8 for range
(4453399640808141293,-8072600550335099985] finished
[2013-03-18 15:42:37,691] Repair session 221cf2b0-901d-11e2-9b70-212e9e5f89e8 for range (-
8072600550335099985,3612905708855427759] finished
[2013-03-18 15:42:37,753] Repair session 222aae50-901d-11e2-9b70-212e9e5f89e8 for range
(3612905708855427759,4453399640808141293] finished
[2013-03-18 15:42:37,753] Repair command #2 finished
[2013-03-18 15:42:37,760] Nothing to repair for keyspace 'system_auth'
[2013-03-18 15:42:37,762] Nothing to repair for keyspace 'system_traces'
```

```
-bash-3.2$ ./nodetool -h 66.94.255.236 repair
[2013-03-18 15:48:03,799] Nothing to repair for keyspace 'system'
[2013-03-18 15:48:03,804] Starting repair command #3, repairing 1 ranges for keyspace
schema1
[2013-03-18 15:48:03,805] Repair command #3 finished
[2013-03-18 15:48:03,815] Starting repair command #4, repairing 3 ranges for keyspace DEMO
[2013-03-18 15:48:03,996] Repair session e497ef70-901d-11e2-9b70-212e9e5f89e8 for range
(4453399640808141293,-8072600550335099985] finished
[2013-03-18 15:48:03,996] Repair session e4a3d650-901d-11e2-9b70-212e9e5f89e8 for range (-
8072600550335099985,3612905708855427759] finished
[2013-03-18 15:48:04,020] Repair session e4a7f500-901d-11e2-9b70-212e9e5f89e8 for range
(3612905708855427759,4453399640808141293] finished
[2013-03-18 15:48:04,020] Repair command #4 finished
[2013-03-18 15:48:04,029] Nothing to repair for keyspace 'system_auth'
[2013-03-18 15:48:04,033] Nothing to repair for keyspace 'system_traces'
```