**ratings.dat - tab separated list with the following fields**

|  |  |
| --- | --- |
| Field | Values Allowed |
| User ID | Integers numbered consecutively from 1 |
| Item ID | Integers numbered consecutively from 1 |
| Rating | An integer in the set {1,2,3,4,5} |
| timestamp | unix timestamps as integers (i.e. 1537648585) |

**movies.dat- tab separated list with the following fields**

The last 19 fields are the genres, a 1 indicates the movie is of that genre, a 0 indicates it is not; movies can be in several genres at once.

|  |  |
| --- | --- |
| Field | Values Allowed |

|  |  |
| --- | --- |
| movie id | Integers numbered consecutively from 1 (same as itemID above) |
| movie title | “Some string” |
| release date | DD-MMM-YYYY (i.e. 01-Jan-1995) |
| video release date |  |
| IMDb URL | “www.xyz.com” |
| unknown | ???? |
| Action | 0/1 |
| Adventure | 0/1 |
| Animation | 0/1 |
| Children's | 0/1 |
| Comedy | 0/1 |
| Crime | 0/1 |
| Documentary | 0/1 |
| Drama | 0/1 |
| Fantasy | 0/1 |
| Film-Noir | 0/1 |
| Horror | 0/1 |
| Musical | 0/1 |
| Mystery | 0/1 |
| Romance | 0/1 |
| Sci-Fi | 0/1 |
| Thriller | 0/1 |
| War | 0/1 |
| Western | 0/1 |

# Types of Relational Databases

Column Oriented

The common motivations of NoSQL design are:

• Easier deployment.

• Large scale data.

• Meeting the scalability and failover.

• Can be used as a Caching layer for storing the transaction data.

Key features of NoSQL are:

* Replicate/distribute data over many servers.
* In most of NoSQL data is partitioned and replicated across multiple nodes.

Definitions:

* Horizontal Scaling/Scaling Out - adding more machines to a pool of resources
* Vertical Scaling/Scaling Up - adding more CPU, RAM, etc. to existing machines. Scaling Up by adding new expensive big servers is difficult because of hardware limitations, involves lots of complex Join Operation, requires higher level of skills, and it is not reliable in some cases.
* Cap Theorem
  + Consistency – Ensures that all nodes see the same data at the same time
  + Availability – Every request gets a response of either success or failure
  + Partition Tolerant – continued functionality even if an instance of a database fails
* A database shard is a horizontal partition of data in a database or search engine

Cassandra is a

# Cassandra Features:

Hence online banking applications in general cannot cope with BASE properties, whereas for applications such as Facebook, reading an old value in an instance of time is acceptable, hence can possibly best fit for BASE.

Creat KeySpace

Name if keypace

With

Replication = {‘class’:’SimpleyStrategy,

‘replication\_factor’ :3}; i.e 3 replicas

Creat table appInstance(

Id uuid,

App\_name text,

Cpu\_time int,

Primary Key (id))

create type rating(entry int, iid int, rating int, tstamp timestamp);

This is because the first attribute specified in the primary key list is used as the partition key, which determines which node the row is stored on. The rest of the primary key is used as the clustering key, which determines how data is ordered on the disk.

Cassandra does not provide a mechanism to sort query results at query time, so we have to consider sort order when creating a table.

Partition key mapped to cluster using a hashmap

There's no distinction between master and worker nodes in Cassandra. All nodes run the same services.

Cassandra arranges the nodes in a cluster, in a ring format, and assigns data to them.

1. Insert all data into a table
   1. Table = key, uuid, iid, rating, time
2. get uid’s
   1. table = uids from data (thank you overwrite)
   2. select values
3. queries data for rating per uid
   1. select by uuid, add to map

Cassandra/conf/Cassandra.yaml en

[enable\_user\_defined\_functions](https://docs.datastax.com/en/dse/5.1/dse-admin/datastax_enterprise/config/configCassandra_yaml.html#configCassandra_yaml__enable_user_defined_functions) to true