1. What is NoSQL data base?

A NoSQL (originally referring to "non SQL" or "non relational") database provides a mechanism for storage and retrieval of data that is modeled in means other than the tabular relations used in relational databases. ... NoSQL databases are increasingly used in big data and real-time web applications.

1. How does data get stored in NoSQl database?

Key-value stores are the simplest NoSQL databases. Every single item in the database is stored as an attribute name (or 'key'), together with its value.

1. What is a column family in HBase?

In the HBase data model columns are grouped into column families, which must be defined up front during table creation. Column families are stored together on disk, which is why HBase is referred to as a column-oriented data store.

1. How many maximum number of columns can be added to HBase table?

10,000

5. Why columns are not defined at the time of table creation in HBase?

All column members of a column family have the same prefix. For example, the columns courses:history and courses:math are both members of the courses column family. The colon character (:) delimits the column family from the . The column family prefix must be composed of printable characters. The qualifying tail, the column family qualifier, can be made of any arbitrary bytes. Column families must be declared up front at schema definition time whereas columns do not need to be defined at schema time but can be conjured on the fly while the table is up an running.

Physically, all column family members are stored together on the filesystem. Because tunings and storage specifications are done at the column family level, it is advised that all column family members have the same general access pattern and size characteristics.

6. How does data get managed in HBase?

NoSQL databases are designed for scalability where unstructured data is spread across multiple nodes. When data volumes increase you just need to add another node to accommodate the growth. The lack of structure in NoSQL databases relaxes stringent requirements of consistency enforced in relational databases to improve speed and agility. Hbase, MongoDB and Cassandra are the three major options that provide NoSQL capabilities. The options differ in the features they provide, so the decision on which to use is informed by the workload that will be handled. The main difference between Hbase and Cassandra databases is the consistency model they implement. Cassandra implements eventual consistency which guarantees writes are available. This provides excellent write scaling but suffers a penalty when reading because for consistency in reads you have to read from many copies of data. On the other hand HBase provides a strong consistency model that excels at scaling reads but does not scale on writes as well as Cassandra does.

Hbase is natively supported on Hadoop and it is the subject of this tutorial. The main characteristics that make Hbase an excellent data management platform are fault tolerance, speed and usability. Fault tolerance is provided by automatic fail-over, automatically sharded and load balanced tables, strong consistency in row level operations and replication. Speed is provided by almost real time lookups, in memory caching and server side processing. Usability is provided by a flexible data model that allows many uses, a simple Java API and ability to export metrics.

7. What happens internally when new data gets inserted into HBase table?

Now what’s neat about NoSQL-style databases like this (and Endeca Server is the same) is that you can define individual columns just by using them. For example, I could create columns for the airport name, airport city, airport state and airport code just by using their name in a data load, prefixing those column names with the named of a previously-defined column family. Using the HBase Shell, for example, I could issue the following PUT commands to insert the first row of data into this HBase table, like this:

put 'geog\_origin’,’SFO','origin:airport\_name','San Francisco, CA: San Francisco'

put 'geog\_origin’,’SFO','origin:city’,’San Francisco, CA'

put 'geog\_origin’,’SFO',’origin':state','California'

put 'geog\_origin’,'SFO',’origin':id’,'14771'

Now my HBase table conceptually looks like this: NewImage If I then want to use another column under the “origin” column family for LAX, I can just do so by using it in the next set of PUT commands, like this:

put 'geog\_origin','LAX’,origin:airport\_name','Los Angeles, CA: Los Angeles'

put 'geog\_origin','LAX','origin:city','Los Angeles, CA'

put 'geog\_origin','LAX','origin:state','California'

put 'geog\_origin','LAX','origin:region’,’West Coast'

put 'geog\_origin','LAX','origin:id','12892'

NewImage Each column within column families has its values individually set, retrieved and deleted using PUT, GET and DELETE commands, and as long as you prefix the column name with one of the previously-defined column-family names and provide the key value for the row you’re interested in, HBase database tables are very flexible and were designed for simple product catalog-type applications running on hundreds of sharded server nodes for companies of the likes of Amazon, Google and Facebook (see this HBase “Powered-by” page for more examples of organizations using HBase). But what HBase very much isn’t is a relational database like Oracle, Microsoft SQL server or even Apache Hive, databases that we’re much more likely to store data warehouse-type data in. In the previous post I showed how Hive table structures can in-fact be put over HBase tables, mapping HBase columns to Hive columns, and then HiveQL INSERT INTO TABLE … SELECT commands can be used to bulk-load these HBase tables with initial sets of data. So back to the original question - what’s the best way to then incrementally load and refresh these HBase tables, and I can I still use HiveQL for this? In my original post, I defined Hive tables over my HBase ones using the Hive-on-Hbase (yum install hive-hbase) package and associated Hive storage handler; for example, the Hive table that provided SQL access over the flight\_delays HBase tables was defined like this: