Auch HBase speichert Daten in Tabellen, die Reihen und Spalten haben. Allerdings sollte man Tabellen eher als multi-dimensionale Maps verstehen.

Table

Eine `Table` besteht aus mehrere `rows`

Row

Eine `Row` besteht aus einem `row key` und mindestens einer Spalte mit einem dazugehörigen Wert.
`Rows` werden nach dem `row key` Alphabetisch sortiert. Deshalb ist ein richtig gewählter `row key` essenziell.
Das Ziel ist es, dass zusammenhängende Daten möglichst nah beieinander sind.
Ein häufig verwendetes `row key` pattern ist die `website domain`. Dabei sind die `row keys` domains, die rückwärts abgespeichert werden.
Bsp:

```
org.apache.www
org.apache.mail
org.apache.jira
```

Column

Eine `Column` besteht aus einer `column family` und einem `column qualifier`. Diese sind mit einem `:` getrennt.

Column Family

Mit `Column Families` können wir die Daten von mehreren `Columns` zusammen abspeichern. Dies wird gemacht um schnell auf mehrere zusammenhängende Daten aus mehreren `Columns` zugreifen zu können.
Jede `Column Family` hat ein Set aus `storage properties`. Diese geben an ob die Daten zb. im Arbeitsspeicher gecached werden sollen, oder wie die Daten compressed werden. Alle `Rows` innerhalb der gleichen Tabelle haben alle `Column Families`, auch wenn sie dort keine Daten reingespeichert haben.

Column Qualifier

Ein `Column Qualifier` wird genutzt um einer `Column Family` einen `Index` zur verfügung zu stellen. Ein beispiel wäre:
Wir haben die Column Family `Content`, dann wäre ein Qualifier `content:html` und ein anderer `content:pdf`.
Während die `Column Families` bei der erstellung der Tabelle fix sind, können `Qualifier` verändert werden.
Außerdem können sie für jede Reihe unterschiedlich aussehen.

Cell

Eine `Cell` ist eine Kombination aus `Row`, `Column Family und Qualifier` und beinhaltet eine `Value` mit einem `Timestamp`, dieser repräsentiert die Version der Daten.

Timestamp

Mit jedem Wert wird auch ein `Timestamp` mitgeschrieben. Standardmäßig wird die Zeit des `Region Server` als `Timestamp` gespeichert, es besteht allerding die Möglichkeit den `Timestamp` selber zu wählen.

Mehr Infos

```
docker run -ti harisekhon/hbase
```

```
create 'Music_Playlist', 'Song', 'Singer'

list
```

Insert Data

Mit `put <table_name>, <row_id>, <colfamily:colname>, <value>` können wir eine `Row` Updaten oder eine neue `Row` erstellen. (Upsert)

```
put 'Music_Playlist', '1', 'Song:name', 'Songname 1'
put 'Music_Playlist', '1', 'Song:year', '2020'
put 'Music_Playlist', '1', 'Singer:name', 'Singer1 name'
put 'Music_Playlist', '1', 'Singer:country', 'DE'

get 'Music_Playlist', '1'
```

Update Data

Genau wie ein Insert können wir mit dem `put` Kommando Werte aktualisieren. Dann wir der Wert in der `Cell` überschrieben und der `Timestamp` wird auf die aktuelle Zeit gesetzt.

```
put 'Music_Playlist', '1', 'Song:name', 'Songname 1 changed',
```

Delete Data

Mit `delete <Table>, <row_id>` können wir eine ganze `Row` löschen. Mit `delete <Table>, <row_id>, <colFamily:colName>` können wir einen einzelne `Column` löschen

```
delete 'Music_Playlist', '1'
delete 'Music_Playlist', '1', Song:name'
```

Relations identisch zu Cassandra

Versionierung

Mit Hilfe von `Timestamps` können wir uns einen alten Wert anzeigen lassen:

```
get 'Music_Playlist', '1', {COULMN=>'Song:name', TIMESTAMP => <Timestamp>}
```

Alternativ kann man für eine Tabelle festlegen, wieviel Versionen beibehalten werden sollen:

```
alter 'Music_Playlist', NAME => 'Song', VERSIONS => 5
```

Hier werden also die 5 aktuellsten Werte beibehalten. Wenn wir jetzt einen neuen Wert setzen:

```
put 'Music_Playlist', 1, 'Song:name', 'New Song Name'
```

Und dann eine Abfrage machen mit:

```
get 'Music_Playlist', '1', {COLUMN=>'Song:name', VERSIONS=>2}
```

bekommen wir die letzten 2 Versionen der Daten.

Infos zu den Kommandos