Example 1:
Creating a table using HBase Shell

HBase has a command-line shell

```
hbase(main):001:0>
```

This shell can be used to create tables, insert and read data etc

```
hbase(main):001:0> create 'notifications','attributes','metrics'
```

Create a table to hold notifications data

```
hbase(main):001:0> create 'notifications','attributes','metrics'
```

This is the name of the table

```
hbase(main):001:0> create 'notifications' 'attributes','metrics'
```

These are column families

column families

```
hbase(main):001:0> create 'notifications' 'attributes','metrics'
```

Column families are groups of columns which are usually semantically related

column families

```
hbase(main):001:0> create 'notifications' 'attributes','metrics'
```

When you create a table in HBase, you don't have to specify the columns in the table

column families

```
hbase(main):001:0> create 'notifications' 'attributes','metrics'
```

For instance, this notification table might have columns like type, text , timestamp etc

column families

```
hbase(main):001:0> create 'notifications' 'attributes','metrics'
```

columns like type, text, timestamp etc

These columns are defined on the fly when you insert data for a specific a row id

column families

```
hbase(main):001:0> create 'notifications' 'attributes','metrics'
```

Every column has to belong to some column family

column families

```
hbase(main):001:0> create 'notifications','attributes','metrics'
```

The attributes column family may have columns like type, timestamp

column families

```
hbase(main):001:0> create 'notifications','attributes','metrics'
```

The metrics column family may have columns like #clicks, # views

column families

```
hbase(main):001:0> create 'notifications','attributes','metrics'
```

Every table must have at least 1 column family

column families

```
hbase(main):001:0> create 'notifications','attributes','metrics'
```

Column families, unlike columns are usually created at the time of table creation

column families

```
hbase(main):001:0> create 'notifications','attributes','metrics'
```

It is possible to add or change column families later, but this is rarely done

Example 2:

Inserting data into a table using HBase Shell

put

```
put 'notifications',2, 'attributes:for_user','Chaz'
```

Data is inserted into HBase tables using the put operation

```
put 'notifications',2, 'attributes:for_user','Chaz'
```

id	type	for user	from user	timestamp
1	Friend request status	Ryan	Jessica	146710201
2	Comment	Chaz	Daniel	146711200
3	Comment	Rick	Brendan	1467112205
4	Like	Rick	Brendan	1467112213

row	column	value
2	for user	Chaz

Each entry in a HBase table is like a cell in a traditional table

row	column	value
2	for user	Chaz

```
put 'notifications',2, 'attributes:for_user','Chaz'
```

With put, you insert data 1 cell at a time

row	column	value
2	for user	Chaz

```
put 'notifications',2, 'attributes:for_user','Chaz'
```

The table name

row	column	value
2	for user	Chaz

```
put 'notifications',2,'attributes:for_user','Chaz'
```

id	type	for user	from user	timestamp
1	Friend request status	Ryan	Jessica	146710201
2	Comment	Chaz	Daniel	146711200
3	Comment	Rick	Brendan	1467112205
4	Like	Rick	Brendan	1467112213

The row id

row	column	value
2	for user	Chaz

```
put 'notifications',2,'attributes:for_user','Chaz'
```

id	type	for user	from user	timestamp
1	Friend request status	Ryan	Jessica	146710201
2	Comment	Chaz	Daniel	146711200
3	Comment	Rick	Brendan	1467112205
4	Like	Rick	Brendan	1467112213

The column

row	column	value
2	for user	Chaz

```
put 'notifications',2, 'attributes:for_user','Chaz'
```

id	type	for user	from user	timestamp
1	Friend request status	Ryan	Jessica	146710201
2	Comment	Chaz	Daniel	146711200
3	Comment	Rick	Brendan	1467112205
4	Like	Rick	Brendan	1467112213

The value

row	column	value
2	for user	Chaz

```
put 'notifications',2, 'attributes:for_user','Chaz'
```

The column name is specified along with it's column family

row	column	value
2	for user	Chaz

```
put 'notifications',2, 'attributes:for_user','Chaz'
```

Every column must belong to a column family

row	column	value
2	for user	Chaz

```
put 'notifications',2, 'attributes:for_user','Chaz'
```

An HBase table is like a sorted map

Key

row	column	value
2	for user	Chaz

```
put 'notifications',2, 'attributes:for_user','Chaz'
```

Key

Value

row	column	value
2	for user	Chaz

```
put 'notifications',2, 'attributes:for_user','Chaz'
```

Key

Value

```
put 'notifications',2, 'attributes:for_user','Chaz'
put 'notifications',2, 'attributes:type','Comment'
```

With the put operation, we are inserting new keys into the map

Example 3:

Updating data using the HBase Shell

put (contd)

The put operation has 2 purposes:

Inserting values for new keys (row id, column)

Updating the value for existing keys

Inserting values for new keys

Say we want to track the number of times a notification was opened/clicked on

```
put 'notifications',2, 'metrics:open',0
```

Updating the value for existing keys

Inserting values for new keys

```
put 'notifications',2, 'metrics:open',0
```

Create row id 2 in notifications table
If the row id does not exist

Updating the value for existing keys

Inserting values for new keys

```
put 'notifications', 2, 'metrics:open',0
```

Create column open for the row id 2 in the metrics column family

Updating the value for existing keys

Inserting values for new keys

```
put 'notifications',2, 'metrics:open',0
```

Insert the value 0 for the row id 2 and column open

Updating the value for existing keys

Updating the value for existing keys

```
put 'notifications',2, 'metrics:open',0
```

```
put 'notifications',2, 'metrics:open',1
```

When someone clicks or opens the notification, update the value for this key

Updating the value for existing keys

```
put 'notifications',2, 'metrics:open',0
```

```
put 'notifications',2, 'metrics:open' 1
```

Since the row id and column already exist, the value is just updated

Updating the value for existing keys

```
put 'notifications',2, 'metrics:open' 0
put 'notifications',2, 'metrics:open' 1
```

HBase actually does not lose the old value

Updating the value for existing keys

```
put 'notifications',2, 'metrics:open',0
put 'notifications',2, 'metrics:open',1
```

The history of updates for a specific key is maintained and retrievable

Updating the value for existing keys

```
put 'notifications',2, 'metrics:open',0
```

1467184097569

```
put 'notifications',2, 'metrics:open',1
```

1467181276487

Each version is stored with the created timestamp

Updating the value for existing keys

```
put 'notifications',2, 'metrics:open',0
```

1467184097569

```
put 'notifications',2, 'metrics:open',1
```

1467181276487

During retrieval, the latest version is retrieved by default

Example 4:

Retrieving data using the HBase Shell

get

```
get 'notifications',2
```

Data is retrieved from HBase tables using the get operation

```
get 'notifications',2
```

You can retrieve
data for 1 row id
at a time

```
get 'notifications',2
```

The default behavior is to return all the columns for the specified row id

```
get 'notifications',2
```

get has 2 mandatory arguments

```
get 'notifications',2
```

The table name

```
get 'notifications',2
```

The row id

```
get 'notifications',2
```

```
COLUMN                          CELL
 attributes:for_user            timestamp=1467179977249, value=Chaz
 attributes:type                timestamp=1467181276487, value=Comment
 metrics:open                   timestamp=1467184097569, value=1
```

The column names with their column families

```
get 'notifications',2
```

```
COLUMN                          CELL
 attributes:for_user            timestamp=1467179977249, value=Chaz
 attributes:type                timestamp=1467181276487, value=Comment
 metrics:open                   timestamp=1467184097569, value=1
```

The values for these columns

```
get 'notifications',2
```

```
COLUMN                          CELL
 attributes:for_user            timestamp=1467179977249, value=Chaz
 attributes:type                timestamp=1467181276487, value=Comment
 metrics:open                   timestamp=1467184097569, value=1
```

The timestamp when the value was last updated

```
get 'notifications',2
```

```
COLUMN                          CELL
 attributes:for_user            timestamp=1467179977249, value=Chaz
 attributes:type                timestamp=1467181276487, value=Comment
 metrics:open                   timestamp=1467184097569, value=1
```

In HBase, value + timestamp is called a cell

```
get 'notifications',2
```

```
COLUMN                          CELL
 attributes:for_user            timestamp=1467179977249, value=Chaz
 attributes:type                timestamp=1467181276487, value=Comment
 metrics:open                   timestamp=1467184097569, value=1
```

A row id, column can have multiple cells

```
get 'notifications',2
```

```
COLUMN                          CELL
 attributes:for_user            timestamp=1467179977249, value=Chaz
 attributes:type                timestamp=1467181276487, value=Comment
 metrics:open                   timestamp=1467184097569, value=1
```

By default, the cell with the latest timestamp is retrieved

```
get 'notifications',2,'metrics:open'
```

```
COLUMN
 metrics:open
```

```
CELL
timestamp=1467184097569, value=1
```

You can ask get to retrieve values for specific columns

```
get 'notifications',2,'metrics:open','attributes:type'
```

```
COLUMN
 attributes:type
 metrics:open
```

```
 CELL
timestamp=1467181276487, value=Comment
timestamp=1467184097569, value=1
```

You can also specify a list of columns

Example 5:

Retrieving a range of row ids using the HBase Shell

scan

get operations only
allow you to retrieve
1 row id at a time

HBase tables are sorted maps, ie. row ids are sorted

With the scan operation, you can retrieve row ids within a specified range

```
scan 'notifications'
```

```
ROW                COLUMN+CELL
 1                 column=attributes:text, timestamp=1467179437142, value="Hi there! Buy this thing
                   "
 1                 column=attributes:type, timestamp=1467179418482, value=promotion
 2                 column=attributes:for_user, timestamp=1467179977249, value=Chaz
 2                 column=attributes:type, timestamp=1467181276487, value=Comment
 2                 column=metrics:open, timestamp=1467184097569, value=1
```

Returns all values from the notifications table

```
scan 'notifications'
```

```
ROW                COLUMN+CELL
 1                 column=attributes:text, timestamp=1467179437142, value="Hi there! Buy this thing
                   "
 1                 column=attributes:type, timestamp=1467179418482, value=promotion
 2                 column=attributes:for_user, timestamp=1467179977249, value=Chaz
 2                 column=attributes:type, timestamp=1467181276487, value=Comment
 2                 column=metrics:open, timestamp=1467184097569, value=1
```

All columns for row id 1

```
scan 'notifications'
```

```
ROW          COLUMN+CELL
 1           column=attributes:text, timestamp=1467179437142, value="Hi there! Buy this thing
             "
 1           column=attributes:type, timestamp=1467179418482, value=promotion
 2           column=attributes:for_user, timestamp=1467179977249, value=Chaz
 2           column=attributes:type, timestamp=1467181276487, value=Comment
 2           column=metrics:open, timestamp=1467184097569, value=1
```

All columns for row id 2

```
scan 'notifications',{COLUMNS => ['attributes:type'],LIMIT => 1,STARTROW => "2"}
```

You can pass in a dictionary with some specifications to scan

```
scan 'notifications',{COLUMNS => ['attributes:type'],LIMIT => 1,STARTROW => "2"}
```

COLUMNS

A list of column names
(with column family)

```
scan 'notifications',{COLUMNS => ['attributes:type'],LIMIT => 1,STARTROW => "2"}
```

LIMIT

The number of values to be returned

```
scan 'notifications',{COLUMNS => ['attributes:type'],LIMIT => 1,STARTROW => "2"}
```

STARTROW

Only values starting from this row id

You can specify the end row id using STOPROW

```
scan 'notifications',{COLUMNS => ['attributes:type'],LIMIT => 1,STARTROW => "2"}
```

STARTROW STOPROW

Note: The row id has to be passed as a string

```
scan 'notifications', {COLUMNS => ['attributes:type'],LIMIT => 1,STARTROW => "2"}
```

COLUMNS
LIMIT
STARTROW
STOPROW

Any or all of these options can be specified

Example 6:

Deleting data using the HBase Shell

delete

```
delete 'notifications',2, 'attributes:for_user'
```

Data is deleted from HBase tables using the delete operation

row id	column
2	for user

```
delete 'notifications',2, 'attributes:for_user'
```

Delete data 1 cell at a time

row id	column
2	for user

```
delete 'notifications',2, 'attributes:for_user'
```

```
get 'notifications',2
```

Before delete

```
COLUMN                          CELL
 attributes:for_user            timestamp=1467179977249, value=Chaz
 attributes:type                timestamp=1467181276487, value=Comment
 metrics:open                   timestamp=1467184097569, value=1
```

This cell is deleted

```
get 'notifications',2
```

Before delete

```
COLUMN
 attributes:for_user
 attributes:type
 metrics:open
```

```
CELL
timestamp=1467179977249, value=Chaz
timestamp=1467181276487, value=Comment
timestamp=1467184097569, value=1
```

After delete

```
COLUMN
 attributes:type
 metrics:open
```

```
CELL
timestamp=1467181276487, value=Comment
timestamp=1467184097569, value=1
```

Example 7:

Deleting a table using the HBase Shell

drop

Delete an HBase table using the drop command

```
drop 'notifications'
```

```
ERROR: Table notifications is enabled. Disable it first.
```

Before deleting a table, it must be disabled first

```
drop 'notifications'
```

```
ERROR: Table notifications is enabled. Disable it first.
```

When a table is in use, HBase keeps an index of the the row ids in memory

```
drop 'notifications'
```

```
ERROR: Table notifications is enabled. Disable it first.
```

HBase also keeps a log of recent changes in memory, which are periodically flushed to disk

```
drop 'notifications'
```

```
ERROR: Table notifications is enabled. Disable it first.
```

Disabling will flush all recent changes to disk and remove the row id index from memory

```
disable 'notifications'
```

```
drop 'notifications'
```

```
list
```

Use the list command to check whether the table has been dropped

```
disable 'notifications'
```

```
drop 'notifications'
```

```
list
```

Before drop

```
TABLE
notifications
test
```

After drop

```
TABLE
test
```