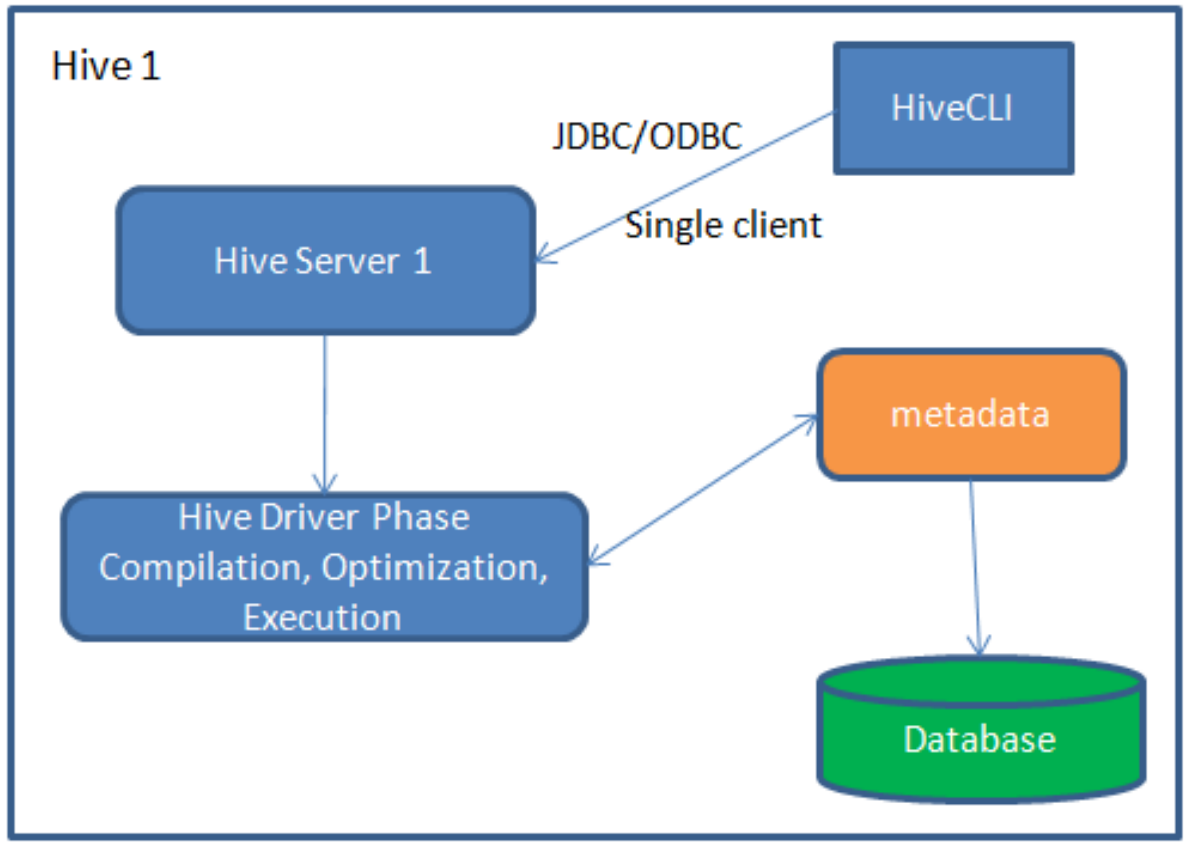
Hive-2

Hive Architecture – Version1 (deprecated)



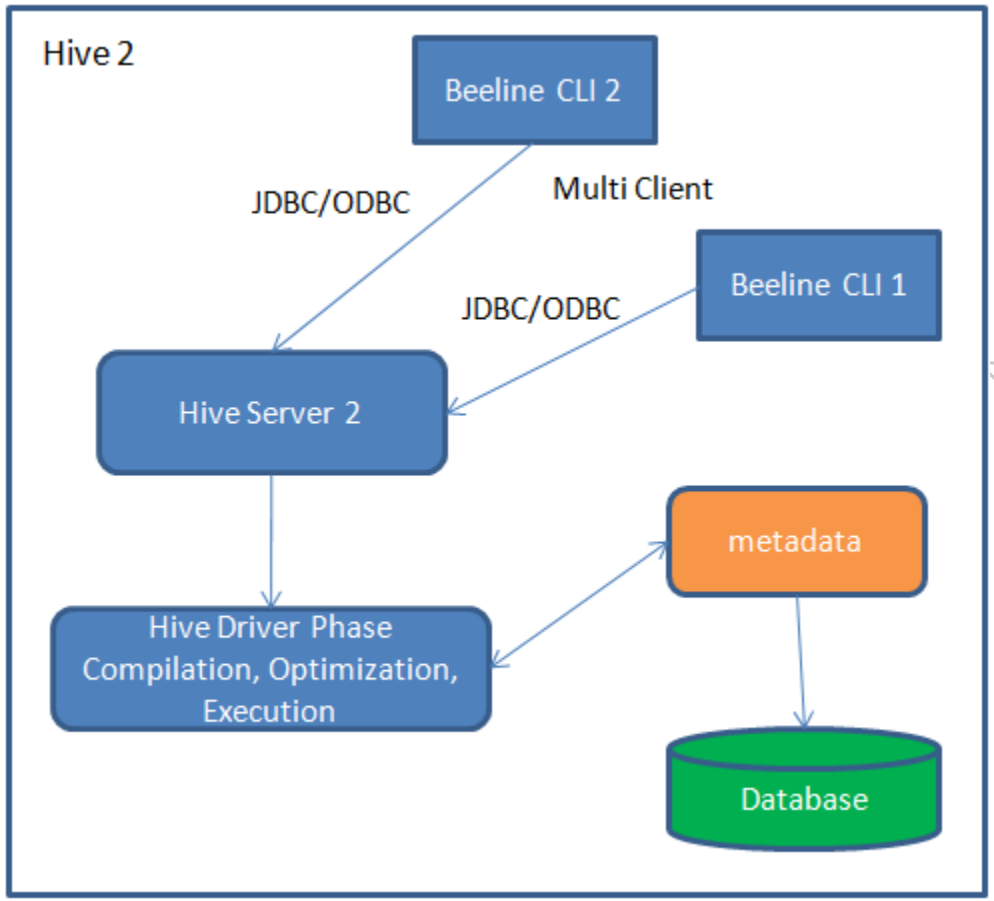
Older versions of hive used to support the command **`hive -service hiveserver`** to start HiveServer1.

There were some serious limitations in this architecture, some of them are –

Support for single user at a time.
No Session Mgmt.
No Authentication support.

Hive Cli is simple to use and is still a widely used interface. We have used in in last session, but we ran all of them in embedded mode.

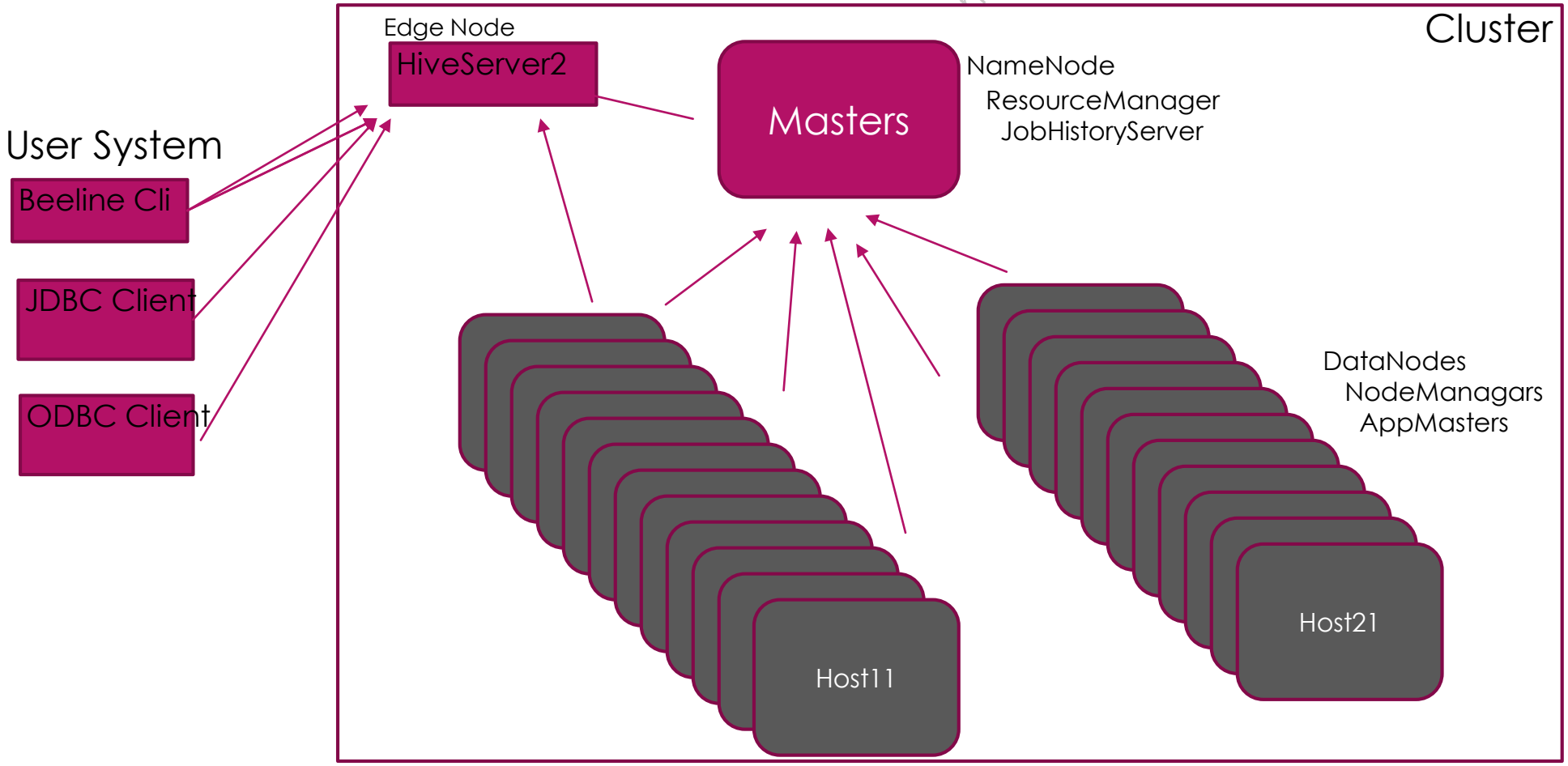
Hive Architecture – Version2



New Model.

HiveServer2 supports multiple clients, and has a better support for security (LDAP, Kerberos etc…)
Also, HiveServer2 Supports JDBC and ODBC driver connections (viz clients can benefit from hive now).
Beeline cli is a tool shipped with hive and is used for connecting to HiveServer2.
We could also use any JDBC program or any DB Client now to connect to Hive

Hive Physical Architecture



How to use the new mode

- By Default security is turned "ON", lets relax the security to get things rolling.
- Modify Core-Site.xml (of Hadoop) and add the below two properties. Replace yourhiveusername with the Unix Login name on which you are trying to start hiveserver2

```
<property>
  <name>hadoop.proxyuser.yourhiveusername.hosts</name>
  <value>*</value>
</property>
<property>
  <name>hadoop.proxyuser.yourhiveusername.groups</name>
  <value>*</value>
</property>
```

- Create $HIVE_HOME/conf/hive-site.xml with the below content

```
<configuration>
 <property>
     <name>hive.server2.authentication</name>
     <value>NONE</value>
   </property>
    <property>
     <name>hive.metastore.sasl.enabled</name>
     <value>false</value>
   </property>
   <property>
     <name>hive.server2.enable.doAs</name>
     <value>false</value>
   </property>
</configuration>
```

Running HiveServer2 and BeelineCli

- Update your environment and add HIVE_CONF_DIR and point it to $HIVE_HOME/conf
- Start hiveserver2 using the following command

```
hive --service hiveserver2
```

- Start BeelineCli (in a separate window) using the following command

```
beeline -u jdbc:hive2://localhost:10000/default org.apache.hive.jdbc.HiveDriver
```

- The above BeelineCLI is read only, If you want to do any DML on Data (very rare case), use a username and password the below way.

```
beeline -u jdbc:hive2://localhost:10000/default org.apache.hive.jdbc.HiveDriver -n <yourunixhiveuser>
-p <yourunixhivepasswd>
```

- If you want to use a JDBC program (or) any DB tool, copy the JDBC Driver to your tool/programs classpath along with all Hadoop Libs (remember the libraries that we have used in HDFS first program, or use Maven), Hive JDBC driver can be located here

$HIVE_HOME/lib/hive-jdbc-2.0.0-standalone.jar

Stocks dataset

- Pick a sample CSV from github. The data format is as below

SYMBOL	TRADE_DATE	HI	LO	OPEN	CLOSE	TRADE_VOLUME
YHOO	24-Dec-15	34.19	34.74	34.1	34.11	3470658
YHOO	23-Dec-15	34.24	34.58	33.99	34.45	13469154

- Upload the data to /stocks/stocks.csv
- Create a table

```
create table stocks (
 stock_name string,
 trade_date string,
 hi double,
 lo double,
 open double,
 close double,
 traded_volume bigint)
ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' LINES TERMINATED BY '\n' ;
```

- Load Data

```
LOAD DATA  INPATH '/stocks/stocks.csv' OVERWRITE INTO TABLE
stocks;
```

- Some Analytics

```
select stock_name, trade_date, count(traded_volume) from stock
group by stock_name,trade_date;
```

Some more analytics

- Observe the #of MapReduces done for each of the below
- Multi-Diamensional Aggregation with Sort

  select stock_name, trade_date, count(traded_volume) from stocks group by stock_name,trade_date order by trade_date desc;

- Custom Sort with a date func

  select stock_name, from_unixtime(unix_timestamp(trade_date,'dd-MMM-yy')) as tradedate, count(traded_volume) from stocks group by stock_name,from_unixtime(unix_timestamp(trade_date,'dd-MMM-yy')) order by tradedate desc;

- Create an analyzed set, query the below table multiple times

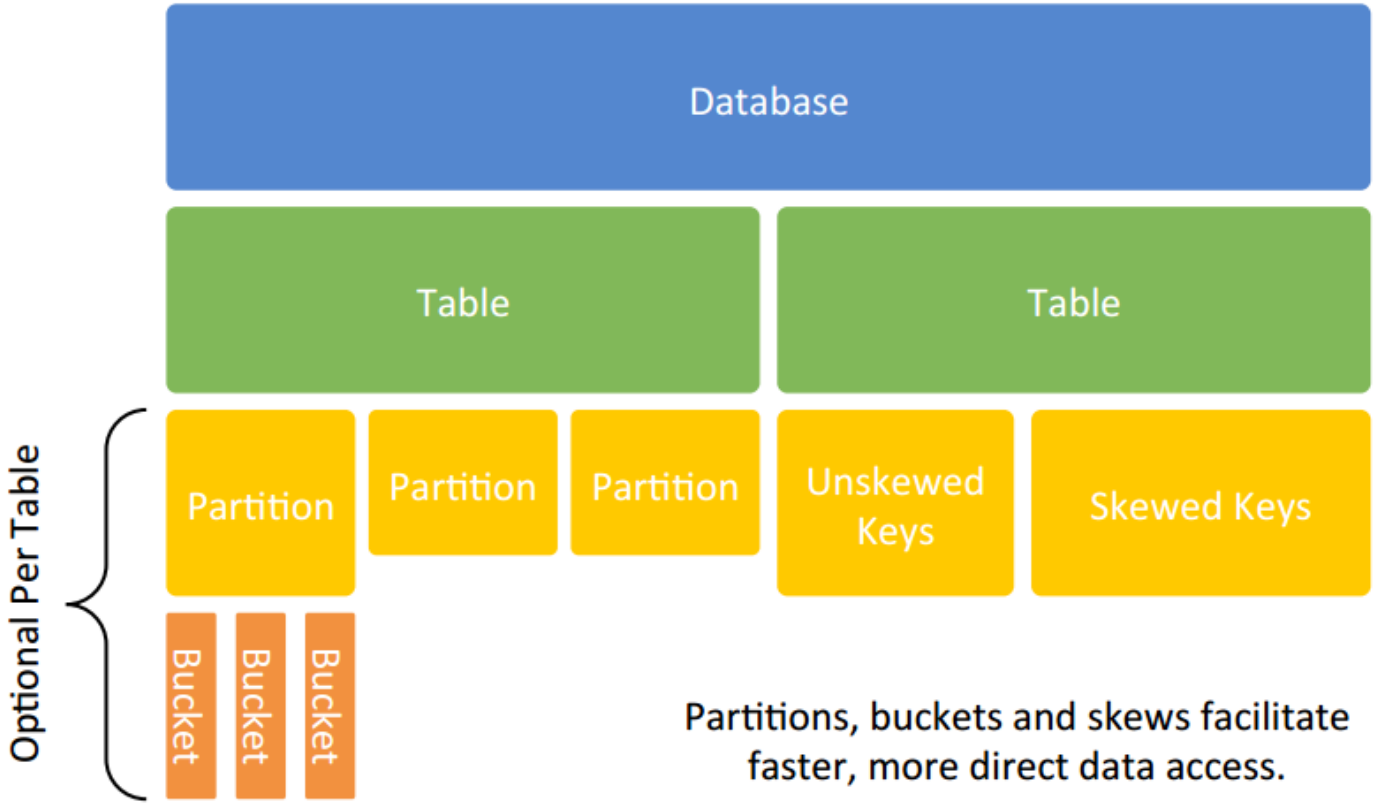
  create table stocks_analyzed_tv as select stock_name, sum(traded_volume) from stocks group by stock_name;

Partitions

- Segregating rows by a predefined condition into data buckets
- Partitioning makes analytic queries respond fast.
- When partitioning you will use 1 or more virtual columns.
- Virtual columns cause directories to be created in HDFS.
- Example –

```
CREATE   TABLE    sale(
     id   int,
     amount   decimal,
     ...
)partitioned by   (country string, state string);
```

Partitions


Database
Table
Table
Partition
Partition
Partition
Unskewed Keys
Skewed Keys
Bucket
Bucket
Bucket
Optional Per Table
Partitions, buckets and skews facilitate faster, more direct data access.

Stocks data now partitioned

- Create a Partitioned table

```
create table stocks_p (
trade_date string, hi double, lo double, open double, close double, traded_volume bigint)
PARTITIONED BY (stock_name string) ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' LINES TERMINATED BY '\n' ;
```

- Lets partition our stocks data –

```
INSERT INTO TABLE stocks_p PARTITION (stock_name='MSFT')  select trade_date, hi, lo, open, close,
traded_volume from stocks where stock_name = 'MSFT';

INSERT INTO TABLE stocks_p PARTITION (stock_name='APPL')  select trade_date, hi, lo, open, close,
traded_volume from stocks where stock_name = 'APPL';

INSERT INTO TABLE stocks_p PARTITION (stock_name='YHOO')  select trade_date, hi, lo, open, close,
traded_volume from stocks where stock_name = 'YHOO';
```


- Now select the data -

```
select * from stocks_p;

select * from stocks_p where stock_name='MSFT';
```

Partitioning cont…

- Observe that we are hard coding the partition Key in the Insert. You could as well do a dynamic partition insert the below way, this method is not used that often because a dynamic insert is heavy in resource consumption.

```
set hive.exec.dynamic.partition.mode=nonstrict;

INSERT OVERWRITE TABLE stocks_cp  PARTITION (stock_name) select stock_name, trade_date, hi, lo, open,
close, traded_volume from stocks
```

- Observe the warehouse directory structure.

Bucketing

- Bucketing is an optimization technique that allows you to segment large sets of data and store the related information at a common place to optimize query performance
- An optimization technique suited for non-cardinal data.
- Differs from partition in the way that –
  - Partitions are better suited with those columns/keys which have high cardinality
  - We need to know the keys upfront for creating partition.
  - Better suited if you know there could be equi-sized set of rows after partitioning. For example, in our stocks data set, we know that there could be equi distribution if data is splitted by stock_name.
  - Bucketing is not a virtual column on the schema.
- A Hash function is used to distribute sets of data to buckets.
- Bucketed datasets provide efficient sampling of data, imagine you wanted to test your query on a subset of data rather on the whole lot.
- Buckets aid in Joins, MR can use the hashkeys of source data buckets to infer which target table to pick the data for joining.
- You can combine bucketing with partitioning to take advantage of both

Bucketing on our stocks data

- Create a table with buckets

```
set hive.enforce.bucketing = true;
create table stocks_cp (
trade_date string, hi double, lo double, open double,
close double, traded_volume bigint)
PARTITIONED BY (stock_name string)
CLUSTERED BY (trade_date) INTO 4 BUCKETS
ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' LINES TERMINATED BY '\n' ;
```

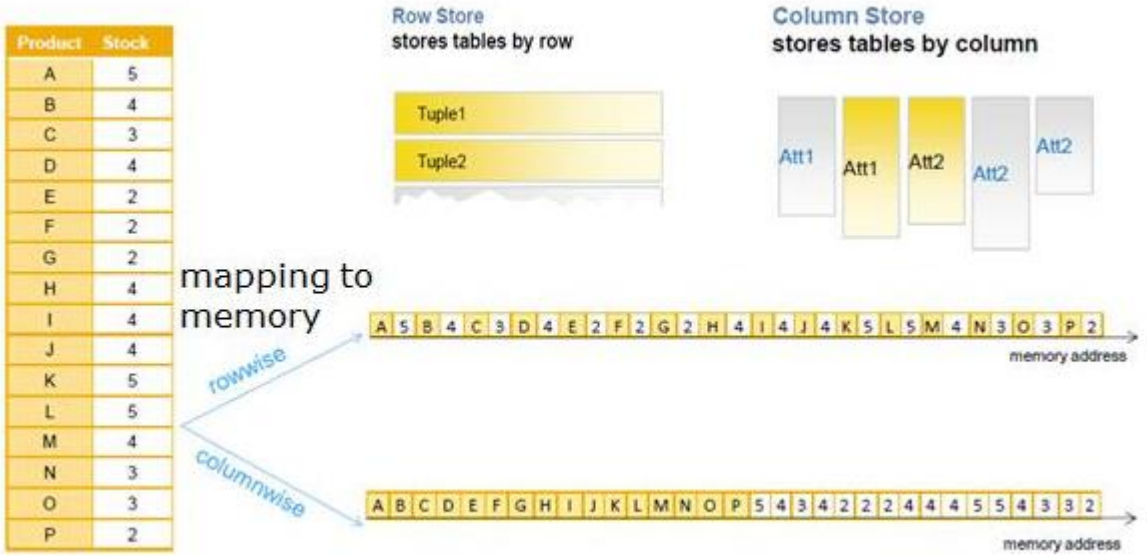
- Insert Data

```
INSERT INTO TABLE stocks_cp PARTITION (stock_name='MSFT')  select trade_date, hi, lo, open, close, traded_volume from stocks where stock_name = 'MSFT';
INSERT INTO TABLE stocks_cp PARTITION (stock_name='APPL')  select trade_date, hi, lo, open, close, traded_volume from stocks where stock_name = 'APPL';
INSERT INTO TABLE stocks_cp PARTITION (stock_name='YHOO')  select trade_date, hi, lo, open, close, traded_volume from stocks where stock_name = 'YHOO';
```

Hive Persistence formats

- Built-in Formats:
  - RCFile
  - ORCFile
  - Avro
  - Delimited Text
  - Regular Expression
  - S3 Logfile
  - Typed Bytes
- 3rd-Party Addons:
  - JSON
  - XML

Columnar format (a generic model, not Hive specifi

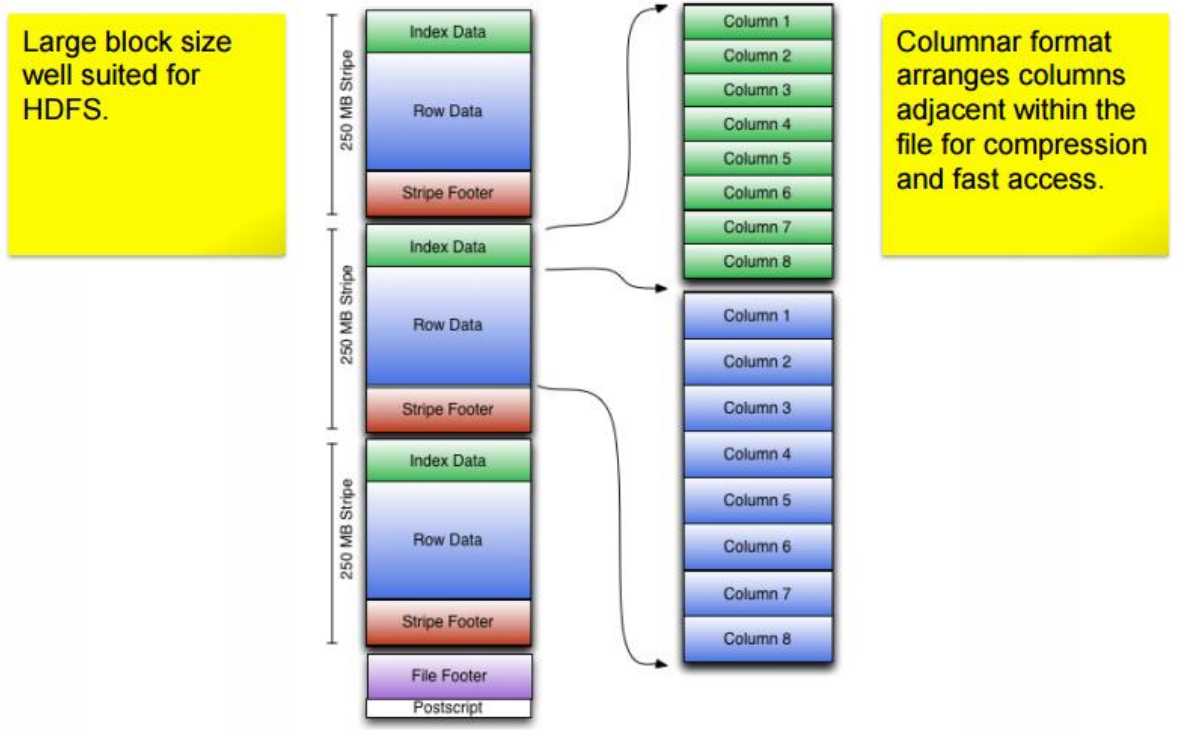


ORC File

```
create table stocks_orc (
trade_date string, hi double, lo double,
open double, close double, traded_volume bigint)
PARTITIONED BY (stock_name string)
CLUSTERED BY (trade_date) INTO 4 BUCKETS
ROW FORMAT DELIMITED FIELDS TERMINATED BY ','
LINES TERMINATED BY '\n'
STORED AS ORCFILE;
```

```
INSERT INTO TABLE stocks_orc PARTITION
(stock_name='MSFT')  select trade_date, hi, lo,
open, close, traded_volume from stocks where
stock_name = 'MSFT';
```


ORC Fileformat



UDF

- User Defined Functions in Hive are created by extending org.apache.hadoop.hive.ql.exec.UDF and overriding evaluate method
- Lets do a UDF which provides us a weekday given a date in dd-MMM-yy format.
- Compile the UDF, and upload it to HDFS, say under /jars dir, in your beelineCLI register the UDF -

```
CREATE FUNCTION WeekDay AS 'com.ixat.hive.HiveUDFTest.WeekDayUDF' USING
JAR 'hdfs://localhost:8020/jars/myudf.jar';
```

- Use the UDF

```
select stock_name, WeekDay(trade_date) from stocks;
```