HIVE concepts and hands on in detail

**HIVE is to process the data which we brought into HDFS on top of HADOOP.**

**(Simply HIVE is a SQL layer on top of HDFS)**

**Intro:**

* HIVE provides an SQL (or) OLAP layer on top of hadoop writing HQL(hive query language) that invokes MapReduce for processing in HDFS as a storage medium.
* Here the query which we write is called as HQL.
* Its similar t SQL, But SQL will be written only on top of structured data.
* HQL can be written on top of Structured / Semi Structured data.

Difference between HQL and SQL

|  |  |
| --- | --- |
| **HQL** | **SQL** |
| * Query time parsing * No transformation / cleansing is required before storing it.   **T**  **L**  **E**  **Requirement 1**  **Requirement2**  **Requirement 3**   * We can do both ELT and ETL * Flexible * Can handle Structured data / Semi structured data and Un structured data (but in real time Unstructured data are not handled rarely) * High latency and high through put * It’s used for experimentation. | * Load time parsing * Cleansing / transformation is required before storing in RDBMS   **Requirement 1**  **T**  **E**  **Requirement2**  **L**  **Requirement 3**   * Only ETL can be done * Its performance oriented * Can handle only structured data. * Low latency and low through put * It deals with prod work load |

|  |
| --- |
| **Concepts:**  **Latency (Measurement of response time for a finite dataset [in seconds])**  Also known as response time or access time, is a measure of how long it takes for the **database** to respond to a single request. ... The Sample time is the round-trip time it takes Foxhound to request and receive three sets of performance statistics from the target **database**.  **Throughput (Volume of data handled in a stipulated time frame [mb/sec])**  It is a measurement used to determine the performance of a **database** system. The **throughput** metric is a classical throughput measure characterizing the ability of the system to support a multi-user workload in a balanced way. |

**Hive is good for:**

1. Handling large volume
2. OLAP (online analytical processing - ELT)
3. Veracity (Raw data of its own type)
4. Backup and Availability
5. DB / DWH

* Migration
* Consolidation
* Data lake

**Hive is not good for:**

1. OLTP
2. Not a replacement for DB.

|  |
| --- |
| **Concept:**  Similar to SQOOP, HIVE can be connected to any tool provided the tool supports a JDBC/ODBC connector. |

**Architecture:**

|  |
| --- |
| JDBC / ODBC  CLI - command line interface  JDBC / ODBC  Thrift server  HIVE  Driver  Compiler, optimizer, executer |
| Name Node  Resource Manager  Data node + Task tracker  Hadoop |

**UI (User interface):**

Hive query can be issues on top of HDFS via three ways.

1. CLI (command line interface). It’s a black screen in which we type and execute the commands.
2. Can be done by any other tool using JDBC/ODBC connector supporter by the tool.

* **Thrift Clients** – As Apache Hive server is based on Thrift, so it can serve the request from all those languages that support Thrift.
* **JDBC Clients** – Apache Hive allows Java applications to connect to it using JDBC driver. It is defined in the class apache.hadoop.hive.jdbc.HiveDriver.
* **ODBC Clients** – ODBC Driver allows applications that support ODBC protocol to connect to Hive. For example JDBC driver, ODBC uses Thrift to communicate with the Hive server.

1. Can be access thru Web UI as well.

**Metadata:**

* Metadata is stored in the Meta store which is in a separate data base.
* By default Meta store is stored in the database called DERBY.
* While installing HIVE, DERBY will also be installed.
* But in real time production, Derby will not be used to store meta store
* Meta store is saved in DERBY, and the actual data is stored in HDFS.
* There are two types in which meta store can be configured
* Embedded

The meta store will be stored locally in the default data base such as DERBY

* Remote

In this case the meta store will be stored in a remote database such as mysql or any other database.. this is the case in real time production.

**Execution engine:**

* Typically it uses MapReduce
* It happens in batch. High latency and more i/o
* But we can use TEZ or SPARK
* It happens on line. Low latency, interactive, fast moving engine.



1. When a HIVE query is executed, its submitted to the Driver.
2. Driver asks the compiler to get the plan
3. Compiler does

* Some basic syntax / sematic checks
* Binds and packages the plan
* Requests meta data from meta store

1. Meta data is obtained from meta store.
2. Compiler sends back the plan to driver
3. Compiler submits the plan to the execution engine to get it executed.

Execution engine is typically MR. we can set a different execution by using the below command

* Set hive.execution.engine=TEZ or SPARK

Execution engine decides on whether to submit the job to YARN or just directly contact name node to get the result data based on the request.

1. Fetches the result

**Primitive data types:**

* Integer (tinyint, smallint, int, bigint)
* String (varchar, char)
* Timestamp (timestamp, date)
* Decimals
* …. And so on

**Complex data types:**

* Array(a,b,c,d)

Its is a collection of items of same data type.

* Struct (a ,b ,10,49)

Its is a collection of items of different data type.

* Map (key, value)

Eg: map(eng#90$mat#89)

* Union(int, array<>, struct())

It’s the collection of other complex data types.

**Traditional Database:**

In turn will be stored in a folder internally

Into tables

We do insert or load

**HIVE:**

We do insert or load

Into tables

In turn will be stored in a folder internally

Direct copy to HDFS using other application like SQOOP or manual movements

**HIVE Installation with Meta store configuration:**

1. Copy the hive tar file to your home path (/home/hduser/install) and extract using below command

cd /home/hduser/install/

tar xvzf apache-hive-1.2.2-bin.tar.gz

sudo mv apache-hive-1.2.2-bin /usr/local/hive

1. After you complete the above steps execute below commands to create directories and give permissions

hadoop fs -mkdir -p /user/hive/warehouse/

hadoop fs -chmod g+w /user/hive/warehouse hadoop fs -mkdir -p /tmp

hadoop fs -chmod g+w /tmp

**How to log in to HIVE thru CLI:**

* To log into HIVE, all we need is to type HIVE from home directory.
* It’s not mandatory that we need to type HIVE from home dir. But in which ever path we type HIVE, the embedded Meta store will be created locally in that path.
* So if we go to some other path and type HIVE, the hive prompt will be initiated. But a new meta store will be created and our past work will not be seen there

Note: For now we will see the concepts using Embedded meta store. Later we shall look to use Remote Meta store.

Eg: hive

**Create Database:**

The Create Database statement is used to create a database in Hive. By default, there is a database in Hive named default.

The general format of creating a database is as follows:

**CREATE (DATABASE|SCHEMA) [IF NOT EXISTS] database\_name**

**[COMMENT database\_comment] [LOCATION hdfs\_path]**

**[WITH DBPROPERTIES (property\_name=property\_value, ...)];**

**Where:**

**DATABASE|SCHEMA:** These are the same thing. These words can be used interchangeably.

**[IF NOT EXISTS]:** This is an optional clause. If not used, an error is thrown when there is an attempt to create a database that already exists.

**[COMMENT]:** This is an optional clause. This is used to place a comment for the database. This comment clause can be used to add a description about the database. The comment must be in single quotes.

**[LOCATION]:** This is an optional clause. This is used to override the default location with the preferred one.

**[WITH DBPROPERTIES]:** This is an optional clause. This clause is used to set properties for the database. These properties are key-value pairs that can be associated with the database to attach additional information with the database.

## DESCRIBE DATABASE [EXTENDED] db\_name;

create database retail;

When the above create command is issued, what happens in the back ground is HIVE initiates the -mkdir command to Linux.

like hadoop fs -mkdir <default location>/retail.db

hadoop fs -mkdir /user/hive/warehouse/retail.db

going forward all the tables and other objects created under this data base will be stored inside this folder.

create database if not exists retail

comment 'retail database for holding retail cust info' location '/user/hduser/hivestore'

with dbproperties ('Created by' = 'Inceptez', 'Created on' = '2019-01-01');

here when we give our own location, then the similar type of folder will be created in this path.

Like hadoop fs -mkdir /user/hduser/hivestore/retail.db

**Describe Data Bases:**

In this case when we describe the database:

describe database retail;

**Drop Data Bases:**

**DROP (DATABASE|SCHEMA) [IF EXISTS] database\_name [RESTRICT|CASCADE];**

**Where:**

**DATABASE|SCHEMA:** These are the same thing. These words can be used interchangeably.

**[IF EXISTS]:** This is an optional clause. If not used, an error is thrown when there is an attempt to drop a database that does not exist.

**[RESTRICT|CASCADE]:** This is an optional clause. RESTRICT is used to restrict the database from getting dropped if there are one or more tables present in the database. RESTRICT is the default behavior of the database. CASCADE is used to drop all the tables present in the database before dropping the database.

drop database retail\_tmp;

drop database if exists retail\_tmp;

drop database if exists retail\_tmp cascade;

**Use Database:**

The USE DATABASE command is used to switch to the database, or it sets the database as the working database. It is analogous to the one used in the other RDBMS. The general format of using a database is as follows:

## USE (DATABASE|SCHEMA)

DATABASE|SCHEMA: These are the same thing. These words can be used interchangeably.

use retail;

**To print the current database name in the CLI:**

set hive.cli.print.current.db=true;

the above command will display the currently used DB in the CLI.

Eg: hive (practice)> select \* from……….

**Show databases:**

The **SHOW DATABASE** command is used to list all the databases in the Hive metastore. The general format of using the SHOW DATABASE command is as follows:

show databases like 'ret\*';

**Create table for storing transactional records:**

The CREATE TABLE statement creates metadata in the database. The table in Hive is the way to read data from files present in HDFS in the table or a structural format.

The general format of using the CREATE TABLE command is as follows: CREATE [TEMPORARY] [EXTERNAL] TABLE [IF NOT EXISTS]

[db\_name.] table\_name

[(col\_name data\_type [COMMENT col\_comment], ...)] [COMMENT table\_comment]

[PARTITIONED BY (col\_name data\_type [COMMENT col\_comment], ...)] [CLUSTERED BY (col\_name, col\_name, ...) [SORTED BY (col\_name [ASC|DESC], ...)] INTO num\_buckets BUCKETS]

[ROW FORMAT row\_format] [STORED AS file\_format]

| STORED BY 'storage.handler.class.name' [WITH SERDEPROPERTIES (...)]

]

[LOCATION hdfs\_path]

[TBLPROPERTIES (property\_name=property\_value, ...)] [AS select\_statement];

**Let us take a look at all the parameters involved:**

**[TEMPORARY]:** This is an optional clause. This clause is used to create temporary tables. These tables once created are only present in the database until the session is active. Once the session comes to an end, all the temporary tables are deleted. Once a temporary table is created, you cannot access the permanent table in that session so you need to either drop or rename the temporary table to access the original one. You cannot create a partition or index on temporary tables.

**[EXTERNAL]:** This is an optional clause. This clause is used to create external tables the same as in the case of RDBMS. The external table works as a window for the data present in the file format in HDFS. For an external table, the data or file need not be present in the default location but can be kept anywhere in the filesystem and can be referred to from that location. Once the external table is dropped, data is not lost from that location.

**[IF NOT EXISTS]:** This is an optional clause. If there is an attempt to create a table that is already present in the database, an error is thrown. To avoid such an error, the IF NOT EXISTS clause is used. When this clause is used, Hive ignores the statement if the table already exists.

**[db\_name]:** This is an optional clause. This clause is used to create tables in the specified database.

**[COMMENT col\_comment]:** This is an optional clause. This is used to attach comments to a particular column. This comment clause can be used to add a description about the column. The comment must be in single quotes.

**[COMMENT table\_comment]:** This is an optional clause. This is used to attach comments to a table. This comment clause can be used to add a description about the table. The comment must be in single quotes.

**[PARTITIONED BY]:** This is an optional clause. This clause is used to create partitioned tables. There can be more than one partition columns in a table. Partitions in Hive work in the same way as

in any RDBMS. They speed up the query performance by keeping the data in specific partitions.

**[CLUSTERED BY]:** This is an optional clause. This clause is used for bucketing purposes. The table or partitions can be bucketed using CLUSTERED BY columns.

**[LOCATION hdfs\_path]:** This option is used while creating external tables. This is the location where files are placed, which is referred to by the external table for the data.

**[TBLPROPERTIES]:** This is an optional clause. This clause allows you to attach more information about the table in the form of a key-value pair.

**[AS select\_statement]: Create Table As Select**, popularly known as **CTAS**, is used to create a table based on the output of the other table or existing table.

**Types of HIVE tables:**

There are two types of tables in hive

* Managed table
* External table.

|  |  |
| --- | --- |
| **Managed Table** | **External Table** |
| Schema and data will be lost when dropped | Only Schema will be lost when dropped.  The data will remain un harmed as it it. |
| Typically used for staging layer | Typically used for Consumption layer |
| Typically Hive only loads and access | Typically External applications loads and access |
| Typically Location will not be specified, default warehouse location will be taken | Typically External location will be specified |
| Syntax:  Create table table\_name (…………) | Syntax:  Create **external** table table\_name (,…………) |

**Creating managed tables:**

create table game(name string, team string, position string, height int, weight int, age int)

row format delimited fields terminated by ','

lines terminated by '\n'

stored as textfile;

Now let’s describe this table.

If we do a normal describe, the columns along with its data type will be displayed.

hive (practice)> describe game;

OK

col\_name data\_type comment

name string

team string

position string

height int

weight int

age int

Time taken: 0.207 seconds, Fetched: 6 row(s)

If we do a formatted describe, more details of the table will be displayed as shown below.

* The columns along with the data types
* The creation date, time and details
* The storage path
* Type of the table
* Input / output format
* Partition, bucket details and so on.

Here an important concept in Out put format is, HiveIgnoreKeyText.

It means the output format should not be stored in key value format. Only the value should be displayed/stored.

hive (practice)> describe formatted game;

OK

col\_name data\_type comment

# col\_name data\_type comment

name string

team string

position string

height int

weight int

age int

# Detailed Table Information

Database: practice

Owner: hduser

CreateTime: Tue Apr 09 16:11:20 EDT 2019

LastAccessTime: UNKNOWN

Protect Mode: None

Retention: 0

Location: hdfs://localhost:54310/user/hive/warehouse/practice.db/game

Table Type: MANAGED\_TABLE

Table Parameters:

transient\_lastDdlTime 1554840680

# Storage Information

SerDe Library: org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe

InputFormat: org.apache.hadoop.mapred.TextInputFormat

OutputFormat: org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat

Compressed: No

Num Buckets: -1

Bucket Columns: []

Sort Columns: []

Storage Desc Params:

field.delim ,

line.delim \n

serialization.format ,

Time taken: 0.209 seconds, Fetched: 33 row(s)

When we give show create table\_name, it will display the short cut of creating the DDL statement.

hive (practice)> show create table game;

OK

createtab\_stmt

CREATE TABLE `game`(

`name` string,

`team` string,

`position` string,

`height` int,

`weight` int,

`age` int)

ROW FORMAT DELIMITED

FIELDS TERMINATED BY ','

LINES TERMINATED BY '\n'

STORED AS INPUTFORMAT

'org.apache.hadoop.mapred.TextInputFormat'

OUTPUTFORMAT

'org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat'

LOCATION

'hdfs://localhost:54310/user/hive/warehouse/practice.db/game'

TBLPROPERTIES (

'transient\_lastDdlTime'='1554840680')

Time taken: 0.194 seconds, Fetched: 18 row(s)

**Load the data into the table [ From Linux client]**

There are three ways in which the data can be loaded into a table.

1. Load the data using the command **“load data local inpath”**
2. **Insert into table\_name values ()**
3. Directly copy the data through LINUX COPY command to the file where table is created.

Let’s take a look one by one:

1. Load the data using the command **“load data local inpath”**

Load data local inpath '/home/hduser/dataset/game.csv' into table game;

See the place where its stored in HDFS

[hduser@Inceptez Datasets]$ hadoop fs -ls /user/hive/warehouse/practice.db/game/

Found 1 items

-rwxr-xr-x 1 hduser supergroup 49686 2019-04-09 17:05 /user/hive/warehouse/practice.db/game/game.csv

Look at the data:

[hduser@Inceptez Datasets]$ hadoop fs -cat /user/hive/warehouse/practice.db/game/game.csv | head -4

Adam Donachie, BAL , Catcher ,74,180,23

Paul Bako, BAL , Catcher ,74,215,35

Ramon Hernandez, BAL , Catcher ,72,210,31

Kevin Millar, BAL , First Baseman ,72,210,35

The above command will

* Access the file which is in LOCAL (linux file system)
* Will copy the file using -put command into the HDFS location where the table’s folder is available.

Load data local inpath '/home/hduser/dataset/game.csv' override into table game;

The above command will behave as same as the first.

But the difference here is,

* The first command will append the new data to the old data in the table if exists.
* The override command will delete the old data in the file and will copy this fresh data.

1. Direct insert into the table:

Insert into table game values (‘kannan’,’WCCG’,’cyclist’,186,77,31);

Insert into table game values (‘vandhana’,’WCCG’,’cyclist’,186,90,24);

Insert into table game values (‘selvi’,’WCCG’,’cyclist’,186,68,35);

Now look at the data in HDFS:

[hduser@Inceptez Datasets]$ hadoop fs -ls /user/hive/warehouse/practice.db/game/

Found 3 items

-rwxr-xr-x 1 hduser supergroup 30 2019-04-09 17:10 /user/hive/warehouse/practice.db/game/000000\_0

-rwxr-xr-x 1 hduser supergroup 32 2019-04-09 17:11 /user/hive/warehouse/practice.db/game/000000\_0\_copy\_1

-rwxr-xr-x 1 hduser supergroup 29 2019-04-09 17:20 /user/hive/warehouse/practice.db/game/000000\_0\_copy\_2

-rwxr-xr-x 1 hduser supergroup 49686 2019-04-09 17:05 /user/hive/warehouse/practice.db/game/game.csv

As we can notice in the above log, output of each insert statement will be created as a file in the table location.

Let’s see the data in it:

[hduser@Inceptez Datasets]$ hadoop fs -cat /user/hive/warehouse/practice.db/game/000000\_0

kannan,WCCG,cyclist,186,77,31

[hduser@Inceptez Datasets]$ hadoop fs -cat /user/hive/warehouse/practice.db/game/000000\_0\_copy\_1

vandhana,WCCG,cyclist,186,90,24

[hduser@Inceptez Datasets]$ hadoop fs -cat /user/hive/warehouse/practice.db/game/000000\_0\_copy\_2

selvi,WCCG,cyclist,186,68,35

lets see insert into select method:

insert into game (name,team,position,height,weight,age) select name,team,position,height,weight,age from game1;

look at HDFS:

[hduser@Inceptez Datasets]$ hadoop fs -ls /user/hive/warehouse/practice.db/game/

Found 6 items

-rwxr-xr-x 1 hduser supergroup 30 2019-04-09 17:10 /user/hive/warehouse/practice.db/game/000000\_0

-rwxr-xr-x 1 hduser supergroup 32 2019-04-09 17:11 /user/hive/warehouse/practice.db/game/000000\_0\_copy\_1

-rwxr-xr-x 1 hduser supergroup 29 2019-04-09 17:20 /user/hive/warehouse/practice.db/game/000000\_0\_copy\_2

-rwxr-xr-x 1 hduser supergroup 90 2019-04-09 17:41 /user/hive/warehouse/practice.db/game/000000\_0\_copy\_3

-rwxr-xr-x 1 hduser supergroup 49686 2019-04-09 17:05 /user/hive/warehouse/practice.db/game/game.csv

-rwxr-xr-x 1 hduser supergroup 125 2019-04-09 17:25 /user/hive/warehouse/practice.db/game/game1.csv

[hduser@Inceptez Datasets]$ hadoop fs -cat /user/hive/warehouse/practice.db/game/000000\_0\_copy\_3

KJ,PSS,Lead,170,32,82

JK,PSS,analyst,170,32,82

JT,PSS,Lead,170,32,78

JV,PSS,sse,170,32,78

|  |
| --- |
| **Concept: as** shown in the above **hadoop fs -ls** statement   * If we issue a “load data local inpath” , its just a copy of the file from local to the table’s folder in HDFS * Other than that output of each command on CLI will be created as a file in the table’s folder in HDFS. * It may be individual insert or insert-select. **So one file per one command.** |

**Select the loaded data**

select \* from game limit 10;

select \* from game where position='Catcher';

select \* from game order by 1 limit 10;

In the CLI while selecting the table if we need to see the column name,

we need to give the below command.

set hive.cli.print.header=true;

**Creating External tables:**

create external table externaltxnrecords (txnno INT, txndate STRING, custno INT, amount DOUBLE, category STRING, product STRING, city STRING, state STRING, spendby STRING)

row format delimited fields terminated by ',' stored as textfile

location '/user/hduser/hiveexternaldata';

|  |
| --- |
| Note: since we have given location, the table will be pointed to this location. A separate folder with table name won’t be created as done with default condition (without specifying location) |

**Describing metadata or schema of the table**

describe formatted txnrecords;

describe formatted externaltxnrecords;

show create table txnrecords;

**Data types in HIVE:**

We have primitive and complex data types.

**Primitive Types**

Types are associated with the columns in the tables. The following Primitive types are supported:

* Integers
* TINYINT—1 byte integer
* SMALLINT—2 byte integer
* INT—4 byte integer
* BIGINT—8 byte integer
* Boolean type
* BOOLEAN—TRUE/FALSE
* Floating point numbers
* FLOAT—single precision
* DOUBLE—Double precision
* Fixed point numbers
* DECIMAL—a fixed point value of user defined scale and precision
* String types
* STRING—sequence of characters in a specified character set
* VARCHAR—sequence of characters in a specified character set with a maximum length
* CHAR—sequence of characters in a specified character set with a defined length
* Date and time types
* TIMESTAMP — A date and time without a timezone ("LocalDateTime" semantics)
* TIMESTAMP WITH LOCAL TIME ZONE — A point in time measured down to nanoseconds ("Instant" semantics)
* DATE—a date
* Binary types
* BINARY—a sequence of bytes

**Using complex data types**

* **Structs**: the elements within the type can be accessed using the DOT (.) notation. For example, for a column c of type STRUCT {a INT; b INT}, the a field is accessed by the expression c.a
* **Maps** (key-value tuples): The elements are accessed using ['element name'] notation. For example in a map M comprising of a mapping from 'group' -> gid the gid value can be accessed using M['group']
* **Arrays** (indexable lists): The elements in the array have to be in the same type. Elements can be accessed using the [n] notation where n is an index (zero-based) into the array. For example, for an array A having the elements ['a', 'b', 'c'], A[1] retruns 'b'.

**Let’s see some examples:**

**ARRAY:**

**(Collection of items which is of same data type)**

Consider we need to create a table for Employee certification scores like

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **empid** | **empname** | **dept** | **certification level1 score** | **certification level 2 score** | **certification level3 score** |
| kannan | 1 | PSS | 89 | 98 | 78 |
| vandana | 2 | Enrol | 98 | 78 | 97 |
| kaushiq | 3 | Bill | 98 | 78 | 89 |

In case there are 100 levels of certification, we need to create 100 columns. Instead we can create one column which will be of type ARRAY. As shown below

|  |  |  |  |
| --- | --- | --- | --- |
| **empid** | **empname** | **dept** | **Certification score** |
| kannan | 1 | PSS | 89,98,78 |
| vandana | 2 | Enrol | 98,78,97 |
| kaushiq | 3 | Bill | 98,78,89 |

Create external table cert\_score (empid int, empname varchar(50), dept varchar(30), score array<int>)

Row format delimited

Fields terminated by ‘,’

Collection items terminated by ‘$’

Lines terminated by ‘\n’

Location ‘/user/hive/kjworks/’

* The arguments should be given in proper order.
* First row formal delimited
* Then only fields terminated should be given. If collection items terminated is give, then will through error
* The arguments should be given in the above shown order

The data should be in the below format if we want to load the data via “Load command” or direct copy.

Kannan,1,PSS,89$98$78

Vandhana,2,enroll,98$78$97

Kaushiq,3,bill,98$78$89

Lets load the data

Load data local inpath ‘/home/hduser/Datasets/cert\_score.csv’ into table cert\_score;

Look at the data by issuing select:

hive> select \* from cert\_score;

OK

1 Kannan PSS [89,98,78]

2 Vandhana enroll [98,78,97]

3 Kaushiq bill [98,78,89]

4 selvi PSS [89,98,78]

5 jagan enroll [98,78,97]

6 vasanthi bill [98,78,89]

Time taken: 0.143 seconds, Fetched: 6 row(s)

Select a particular iteration of array.

hive>Set hive.cli.print.header=true;

hive> select empid, empname, dept, score[1] as level\_1\_score from cert\_score;

OK

empid empname dept level\_1\_score

1 Kannan PSS 98

2 Vandhana enroll 78

3 Kaushiq bill 78

4 selvi PSS 98

5 jagan enroll 78

6 vasanthi bill 78

Time taken: 0.237 seconds, Fetched: 6 row(s)

|  |
| --- |
| **Concept:**  **When a table is created as external, then we can issue DELETE or TRUNCATE commands on that table.**  **Will through the below error:**  hive (practice)> delete table cert\_score;  Usage: delete [FILE|JAR|ARCHIVE] <value> [<value>]\*  Query returned non-zero code: 1, cause: null |

**STRUCT:**

**(Collection of items which is of different data type)**

In case if we need to create a table for customer details which has multiple phone numbers and address information such as

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **custid** | **custname** | **phone\_no1** | **phone\_no2** | **zone** | **city** | **pincode** |
| 1 | kannan | 9994240458 | 9994240463 | z1 | coimbatore | 641042 |
| 2 | vandhana | 9994240459 | 9994240464 | z1 | coimbatore | 641042 |
| 3 | jagan | 9994240460 | 9994240465 | z3 | chennai | 641043 |
| 4 | selvi | 9994240461 | 9994240466 | z6 | madurai | 641044 |
| 5 | vasanthi | 9994240462 | 9994240467 | z9 | pondy | 641045 |

Using STRUCT we can simplify the usage as shown below.

|  |  |  |  |
| --- | --- | --- | --- |
| **custid** | **custname** | **phone array<int>** | **location struct<zone:char(2),city:varchar(15),pin:int>** |
| 1 | kannan | 9994240458$9994240463 | z1$coimbatore$641042 |
| 2 | vandhana | 9994240459$9994240464 | z1$coimbatore$641042 |
| 3 | jagan | 9994240460$9994240465 | z3$chennai$641043 |
| 4 | selvi | 9994240461$9994240466 | z6$madurai$641044 |
| 5 | vasanthi | 9994240462$9994240467 | z9$pondy$641045 |

create external table cust(custid int, custname varchar(50), phone array<bigint>, location struct<zone:char(2),city:varchar(20),pin:int>)

row format delimited

fields terminated by ','

collection items terminated by '$'

lines terminated by '\n'

location '/user/hive/kjworks/cust';

load data local inpath '/home/hduser/Datasets/cust.csv' into table cust;

hive> select \* from cust;

OK

cust.custid cust.custname cust.phone cust.location

1 kannan [9994240458,9994240463] {"zone":"z1","city":"coimbatore","pin":641042}

2 vandhana [9994240459,9994240464] {"zone":"z1","city":"coimbatore","pin":641042}

3 jagan [9994240460,9994240465] {"zone":"z3","city":"chennai","pin":641043}

4 selvi [9994240461,9994240466] {"zone":"z6","city":"madurai","pin":641044}

5 vasanthi [9994240462,9994240467] {"zone":"z9","city":"pondy","pin":641045}

Time taken: 0.116 seconds, Fetched: 5 row(s)

This is how we select specific items from array and stuct:

hive> select cust.custid,custname, phone[0] as phone\_no1,location.city as city, location.pin as pincode from cust;

OK

cust.custid custname phone\_no1 city pincode

1 kannan 9994240458 coimbatore 641042

2 vandhana 9994240459 coimbatore 641042

3 jagan 9994240460 chennai 641043

4 selvi 9994240461 madurai 641044

5 vasanthi 9994240462 pondy 641045

Time taken: 0.115 seconds, Fetched: 5 row(s)

**Map:**

**(Collections of items which will be of key:value pair)**

Lets take the below student marks table as example

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **studid** | **studname** | **dept** | **maths** | **english** | **tamil** | **physics** | **chemistry** | **science** | **computer** | **hindi** | **social** | **drawing** |
| 1 | Kannan | CSE | 76 | 86 | 56 | N/A | N/A | N/A | 67 | 67 | N/A | N/A |
| 2 | vandhana | ECE | 67 | 44 | 76 | N/A | N/A | 67 | N/A | 67 | N/A | N/A |
| 3 | selvi | BSC | 67 | 87 | 67 | 56 | 56 | N/A | N/A | N/A | 45 | 89 |
| 4 | jagan | EEE | 68 | 90 | 70 | 78 | 75 | N/A | N/A | N/A | N/A | N/A |
| 5 | vasanthi | ECE | 69 | 93 | 73 | N/A | N/A | 78 | N/A | 78 | N/A | N/A |
| 6 | kaushiq | MECH | 70 | 96 | 76 | 45 | N/A | N/A | N/A | 56 | 45 | N/A |

Even though every student is not applicable for all subjects, we need to maintain N/A for all those students who are not applicable for the corresponding subject.

Instead by using may<> we can simplify this design as show below

|  |  |  |  |
| --- | --- | --- | --- |
| **studid** | **studname** | **dept** | **marks** |
| 1 | Kannan | CSE | maths#76$english#86$tamil#56$computer#67$hindi#67 |
| 2 | vandhana | ECE | maths#76$english#44$tamil#76$science#67$hindi#67 |
| 3 | selvi | BSC | maths#76$english#87$tamil#67$physics#56$chemistry#56$social#45$drawing#89 |
| 4 | jagan | EEE | maths#78$english#89$tamil#59$physics#59$chemistry#86 |
| 5 | vasanthi | ECE | maths#45$english#67$tamil#67$science#45$hindi#76 |
| 6 | kaushiq | MECH | maths#67$english#86$tamil#45$physics#57$hindi#72$social#95 |

Let’s create a table to handle such data:

create external table student\_map(studid int, studname varchar(50), dept varchar(30),mark map<string,int>)

row format delimited

fields terminated by ','

collection items terminated by '$'

map keys terminated by '#'

lines terminated by '\n'

location '/user/hive/kjworks/student\_map/';

load data local inpath '/home/hduser/Datasets/stud\_map.csv' into table student\_map;

hive (kjpractice)> select \* from student\_map;

OK

1 Kannan CSE {"maths":76,"english":86,"tamil":56,"computer":67,"hindi":67}

2 vandhana ECE {"maths":76,"english":44,"tamil":76,"science":67,"hindi":67}

3 selvi BSC {"maths":76,"english":87,"tamil":67,"physics":56,"chemistry":56,"social":45,"drawing":89}

4 jagan EEE {"maths":78,"english":89,"tamil":59,"physics":59,"chemistry":86}

5 vasanthi ECE {"maths":45,"english":67,"tamil":67,"science":45,"hindi":76}

6 kaushiq MECH {"maths":67,"english":86,"tamil":45,"physics":57,"hindi":72,"social":95}

Time taken: 2.355 seconds, Fetched: 6 row(s)

hive (kjpractice)> select studid, studname, dept, mark["maths"] from student\_map;

OK

1 Kannan CSE 76

2 vandhana ECE 76

3 selvi BSC 76

4 jagan EEE 78

5 vasanthi ECE 45

6 kaushiq MECH 67

Time taken: 0.638 seconds, Fetched: 6 row(s)

**Exploring indexes**

Indexes are useful for increasing the performance of frequent queries based on certain columns. But Hive has limited a capability to index data as indexing large datasets requires sufficient additional storage space and processing overheads. Hive can index the columns to speed up some operations. It stores the indexed data in another table.

**When to use Indexing?**

Indexing can be use under the following circumstances:

* If the dataset is very large.
* If the query execution is more amount of time than you expected.
* If a speedy query execution is required.
* When building a data model.

Indexes are maintained in a separate table in Hive so that it won’t affect the data inside the table, which contains the data. Another major advantage for indexing in Hive is that indexes can also be partitioned depending on the size of the data we have.

**When not to use indexing?**

It is essential to know when and where indexing shouldn’t be used. They should not be used in the following scenarios:

* Indexes are advised to build on the columns on which you frequently perform operations.
* Building more number of indexes also degrade the performance of your query.
* Type of index to be created should be identified prior to its creation (if your data requires bitmap you should not create compact).This leads to increase in time for executing your query.

We hope this blog helped you in understanding Indexing in Hive. Keep visiting [our site](https://acadgild.com/) for more updates on BigData and other technologies.

**Types of index in HIVE**

**Compact indexing** stores the pair of indexed column’s value and its block id.

**Bitmap indexing** stores the combination of indexed column value and list of rows as a bitmap.

The following are the different operations that can be performed on Hive indexes:

* Creating index

create index idx\_custno on table txnrecords(custno) AS 'org.apache.hadoop.hive.ql.index.compact.CompactIndexHandler' WITH DEFERRED REBUILD;

* Showing index

hive (kjpractice)> show formatted index on student\_map;

OK

idx\_name tab\_name col\_names idx\_tab\_name idx\_type comment

stud\_index student\_map studid kjpractice\_\_student\_map\_stud\_index\_\_ compact

Time taken: 0.452 seconds, Fetched: 4 row(s)

* Alter index

alter index stud\_index on student\_map rebuild;

* Dropping index

Drop index stud\_index on student\_map;

**Hive partitioning**

Huge amount of data which is in the range of petabytes is getting stored in [**HDFS**.](http://data-flair.training/blogs/comprehensive-hdfs-guide-introduction-architecture-data-read-write-tutorial/) So due to this, it becomes very difficult for Hadoop users to query this huge amount of data. The Hive was introduced to lower down this burden of data querying. Apache Hive converts the SQL queries into [MapReduce](http://data-flair.training/blogs/hadoop-mapreduce-introduction-tutorial-comprehensive-guide/) jobs and then submits it to the [**Hadoop cluster**.](http://data-flair.training/blogs/install-configure-apache-hadoop-2-7-x-on-ubuntu/) When we submit a SQL query, Hive read the entire data-set. So, it becomes inefficient to run [**MapReduce jobs**](http://data-flair.training/blogs/hadoop-mapreduce-job-execution-flow/) over a large table. Thus this is resolved by creating partitions in tables. Apache Hive makes this job of implementing partitions very easy by creating partitions by its automatic partition scheme at the time of table creation. In Partitioning method, all the table data is divided into multiple partitions. Each partition corresponds to a specific value(s) of partition column(s). It is kept as a sub-record inside the table’s record present in the HDFS. Therefore on querying a particular table, appropriate partition of the table is queried which contains the query value. Thus this decreases the I/O time required by the query. Hence increases the performance speed. Hive Partitions is a way to organizes tables into partitions by dividing tables into different parts based on partition keys. Partition is helpful when the table has one or more Partition keys. Partition keys are basic elements for determining how the data is stored in the table.

Suppose a telecom organization generates 1 TB of data every day and different regional managers query this data based on their own state. For each query by a regional manager, Hive scans the complete data in HDFS and files the results for a particular state. The manager runs the same query daily for his own state analysis and the query gives the result in few hours on a 1 TB dataset. For analytics, the same query could be executed daily on a one-month or six month dataset. The query would take few hours on a month's data. If the data is somehow partitioned based on state, then when a regional manager runs the same query for his state, only the data of that state is scanned and the execution time could be reduced significantly to minutes.

## Hive Partitioning Advantages

* + Partitioning in Hive distributes execution load horizontally.
  + In partition faster execution of queries with the low volume scan of data takes place.
  + Partition avoids full table scan.

## Hive Partitioning Disadvantages

* + There is the possibility of too many small partition creations- too many directories if its applied on high cardinal values

|  |
| --- |
| **Partition** |
| Low cardinal columns (date, country etc) |
| Creates folder |
| Improves where clause performance |

|  |
| --- |
| Concept:  Low cardinality:   * High cardinal column = select count(distinct(column\_value)) = minimum. * Count of unique values should be very less. Such as country, department.   High cardinality:   * High cardinal column = select count(distinct(column\_value)) = higher * Count of unique values should be very high. Such as student\_reg\_ID |

## Partitioning can be done in one of the following two ways:

1. **Static partitioning**
2. **Dynamic partitioning**

|  |  |
| --- | --- |
| **Static Partition** | **Dynamic Partition** |
| Need to understand and define partition | No need to know about the data, automatically created |
| Load command or direct insert or insert-select | Direct insert or insert-select |
| Can be derived from external parameters such as file  name or region or timezone etc | Can be derived from one or more columns from the data |
| Faster when using load command | Slower load because of mapreduce invocation always. |

**Static partitioning**

* Insert input data files individually into a partition table is Static Partition.
* Usually when loading files (big files) into Hive tables static partitions are preferred.
* Static Partition saves your time in loading data compared to dynamic partition.
* You “statically” add a partition in the table and move the file into the partition of the table.
* You can perform Static partition on Hive Manage table or external table.

Lets see the hands on part of it:

Lets take the below kind of data to practice with partitions.

NAME,TEAM,POSITION,WEIGHT,HEIGHT,AGE

Adam Donachie, BAL , Catcher ,74,180,23

Paul Bako, BAL , Catcher ,74,215,35

Ramon Hernandez, BAL , Catcher ,72,210,31

Kevin Millar, BAL , First Baseman ,72,210,35

...................................

...................

.........

Lets try with all three method of loading data into table with static partitions.

(insert-select, load, direct-copy)

* Insert-select

|  |
| --- |
| Logic:  Since the structure of above show example is NAME,TEAM,POSITION,WEIGHT,HEIGHT,AGE  And we are going to partition based on TEAM and POSITION (columns which are in between other columns, we can do a direct load into table. Instead we need to insert by select from non partitioned table.  We can tweak the source file little bit and can load directly which we will see in next example. |

1. Create a staging kind of table to just load direct this above data.

create table game\_raw(name varchar(50),team varchar(5), position varchar(30), height int, weight int, age int)

row format delimited fields terminated by ','

lines terminated by '\n'

stored as textfile

location '/user/hive/kjworks/game\_raw';

load data local inpath '/home/hduser/Datasetd/game\_raw.vdb' into table game\_raw;

1. Create a table partitioned on TEAM and POSITION.

create table game\_static\_part(name varchar(50), height int, weight int, age int)

partitioned by (team varchar(5), position varchar(30))

row format delimited fields terminated by ','

lines terminated by '\n'

stored as textfile

location '/user/hive/kjworks/game\_static\_part';

1. insert into the partitioned table using insert-select from staging table method.

hive> insert into game\_static\_part partition(team='BAL',position='Relief Pitcher')

> select name , height, weight, age from game\_raw where team=' BAL' and position like '%Relief Pitcher%';

1. Look at the HDFS path where the table is pointed out.

[hduser@Inceptez Datasets]$ hadoop fs -ls /user/hive/kjworks/game\_static\_part

Found 1 items

drwxr-xr-x - hduser supergroup 0 2019-04-14 06:07 /user/hive/kjworks/game\_static\_part/team=BAL

[hduser@Inceptez Datasets]$ hadoop fs -ls /user/hive/kjworks/game\_static\_part/team=BAL

Found 1 items

drwxr-xr-x - hduser supergroup 0 2019-04-14 06:07 /user/hive/kjworks/game\_static\_part/team=BAL/position=Relief Pitcher

[hduser@Inceptez Datasets]$

[hduser@Inceptez Datasets]$ hadoop fs -ls "/user/hive/kjworks/game\_static\_part/team=BAL/position=Relief Pitcher"

Found 1 items

-rw-r--r-- 1 hduser supergroup 0 2019-04-14 06:07 /user/hive/kjworks/game\_static\_part/team=BAL/position=Relief Pitcher/000000\_0

[hduser@Inceptez Datasets]$ hadoop fs -cat "/user/hive/kjworks/game\_static\_part/team=BAL/position=Relief Pitcher/000000\_0"

Scott Williamson,72,180,31

John Parrish,71,192,29

Danys Baez,75,225,29

Chad Bradford,77,203,32

Jamie Walker,74,195,36

Brian Burres,73,182,26

Kurt Birkins,74,188,27

James Hoey,78,200,24

Sendy Rleal,73,180,27

Chris Ray,75,200,25

Jeremy Guthrie,73,200,28

[hduser@Inceptez Datasets]$

1. Lets query the table and see what we got.

hive (kjpractice)> select \* from game\_static\_part;

OK

game\_static\_part.name game\_static\_part.height game\_static\_part.weight game\_static\_part.age game\_static\_part.team game\_static\_part.position

Scott Williamson 72 180 31 BAL Relief Pitcher

John Parrish 71 192 29 BAL Relief Pitcher

Danys Baez 75 225 29 BAL Relief Pitcher

Chad Bradford 77 203 32 BAL Relief Pitcher

Jamie Walker 74 195 36 BAL Relief Pitcher

Brian Burres 73 182 26 BAL Relief Pitcher

Kurt Birkins 74 188 27 BAL Relief Pitcher

James Hoey 78 200 24 BAL Relief Pitcher

Sendy Rleal 73 180 27 BAL Relief Pitcher

Chris Ray 75 200 25 BAL Relief Pitcher

Jeremy Guthrie 73 200 28 BAL Relief Pitcher

Time taken: 0.382 seconds, Fetched: 11 row(s)

* Load data local to table

Let’s load the data from local to the table using LOAD DATA LOCAL INPATH command

1. Let’s create a source file with data as shown below.

KJ,170,32,82,PSS,Lead

JK,170,32,82,PSS,Lead

JT,170,32,78,PSS,Lead

JV,170,32,78,PSS,Lead

1. Load the data into the same partitioned table game\_static\_part:

load data local inpath '/home/hduser/Datasets/game1.csv' into table game\_static\_part partition(team='PSS',position='Lead');

[hduser@Inceptez Datasets]$ hadoop fs -ls /user/hive/kjworks/game\_static\_part/team=PSS

Found 1 items

drwxr-xr-x - hduser supergroup 0 2019-04-14 06:43 /user/hive/kjworks/game\_static\_part/team=PSS/position=Lead

[hduser@Inceptez Datasets]$ hadoop fs -ls /user/hive/kjworks/game\_static\_part/team=PSS/position=Lead

Found 1 items

-rw-r--r-- 1 hduser supergroup 91 2019-04-14 06:43 /user/hive/kjworks/game\_static\_part/team=PSS/position=Lead/game1.csv

[hduser@Inceptez Datasets]$ hadoop fs -cat /user/hive/kjworks/game\_static\_part/team=PSS/position=Lead/game1.csv

KJ,170,32,82,PSS,Lead

JK,170,32,82,PSS,Lead

JT,170,32,78,PSS,Lead

JV,170,32,78,PSS,Lead

1. Let’s load another file into same partition.

Source file:

lakshmi,74,180,23

guna,76,189,34

saran,34,178,35

kannan,34,187,35

load data local inpath '/home/hduser/Datasets/game1.csv' into table game\_static\_part partition(team='PSS',position='Lead');

1. Note that by doing “load data local inpath”, in background its just a “-PUT” command which is running.

hive (kjpractice)> dfs -ls /user/hive/kjworks/game\_static\_part/team=PSS/position=Lead;

Found 2 items

-rw-r--r-- 1 hduser supergroup 91 2019-04-14 06:43 /user/hive/kjworks/game\_static\_part/team=PSS/position=Lead/game1.csv

-rw-r--r-- 1 hduser supergroup 102 2019-04-14 08:17 /user/hive/kjworks/game\_static\_part/team=PSS/position=Lead/game2

hive (kjpractice)> dfs -cat /user/hive/kjworks/game\_static\_part/team=PSS/position=Lead/\*;

KJ,170,32,82,PSS,Lead

JK,170,32,82,PSS,Lead

JT,170,32,78,PSS,Lead

JV,170,32,78,PSS,Lead

lakshmi,74,180,23

guna,76,189,34

saran,34,178,35

kannan,34,187,35

hive (kjpractice)> select \* from game\_static\_part;

OK

game\_static\_part.name game\_static\_part.height game\_static\_part.weight game\_static\_part.age game\_static\_part.team game\_static\_part.position

Scott Williamson 72 180 31 BAL Relief Pitcher

John Parrish 71 192 29 BAL Relief Pitcher

Danys Baez 75 225 29 BAL Relief Pitcher

Chad Bradford 77 203 32 BAL Relief Pitcher

Jamie Walker 74 195 36 BAL Relief Pitcher

Brian Burres 73 182 26 BAL Relief Pitcher

Kurt Birkins 74 188 27 BAL Relief Pitcher

James Hoey 78 200 24 BAL Relief Pitcher

Sendy Rleal 73 180 27 BAL Relief Pitcher

Chris Ray 75 200 25 BAL Relief Pitcher

Jeremy Guthrie 73 200 28 BAL Relief Pitcher

KJ 170 32 82 PSS Lead

JK 170 32 82 PSS Lead

JT 170 32 78 PSS Lead

JV 170 32 78 PSS Lead

Lakshmi 74 180 23 PSS Lead

Guna 76 189 34 PSS Lead

Saran 34 178 35 PSS Lead

Kannan 34 187 35 PSS Lead

Time taken: 0.322 seconds, Fetched:20

row(s)

|  |
| --- |
| **Concept:**   * By comparing the above **-cat** command and **select query**, we can notice that the column “team” and “position” is not stored in the HDFS data record. * These partitioned column are taken from the filename **“team=PSS”** and **“position=Lead”.** * So while partitioned the record, the column in which the table is partitioned doesn’t want to be part of data. Instead it should be part of file name. so when the records inside the partition is pulled, the file name is also pulled as a column. * By this way we can avoid storing the partitioned column value inside the partitioned data. |

* Create directories as if partitions are created and copy the data directly into that directory using -PUT command.

Source data as shown below:

Raj,170,32,82,MSG,SUPERVISION

Gokul,170,32,82,MSG,SUPERVISION

Ponraj,170,32,78

Revathy,170,32,78

Senthil,189,45,54

Panneer,176,46,57

Create partition directories inside the table’s directories as required

hadoop fs -mkdir -p /user/hive/kjworks/game\_static\_part/team=MSG/position=Supervision/

copy the source file directly into the partitioned directory.

hadoop fs -put /home/hduser/Datasets/game1.csv /user/hive/kjworks/game\_static\_part/team=MSG/position=Supervision/

hive (kjpractice)> select \* from game\_static\_part where team='MSG';

OK

game\_static\_part.name game\_static\_part.height game\_static\_part.weight game\_static\_part.age game\_static\_part.team game\_static\_part.position

Time taken: 0.168 seconds

**Important**: now if we issue a select \* query, the recently copied data won’t be fetched.

Because we just create partitioned directories using -mkdir and copied the data directly into it using -put command.

But the HIVE meta data is not aware of what we did manually. So when a select \* query is issued, it goes and checks the meta data info and return empty result as our recent manual movement of data is not updated there.

|  |
| --- |
| **Concept:**   * In order to update the information about our manual work to meta store we need to perform **meta store check repair activity.** Which is **msck repair table table\_name;** * By doing **msck repair**, all the information about the partitions, buckets, index and any other objects will be updated in the meta store. |

hive (kjpractice)> msck repair table game\_static\_part;

OK

Partitions not in metastore: game\_static\_part:team=MSG/position=Supervision

Partitions missing from filesystem: game\_static\_part:team=BAL/position=Catcher

Repair: Added partition to metastore game\_static\_part:team=MSG/position=Supervision

Time taken: 0.456 seconds, Fetched: 3 row(s)

hive (kjpractice)> select \* from game\_static\_part **where team='MSG';**

OK

game\_static\_part.name game\_static\_part.height game\_static\_part.weight game\_static\_part.age game\_static\_part.team game\_static\_part.position

Raj 170 32 82 MSG Supervision

Gokul 170 32 82 MSG Supervision

Ponraj 170 32 78 MSG Supervision

Revathy 170 32 78 MSG Supervision

senthil 189 45 54 MSG Supervision

Panneer 176 46 57 MSG Supervision

Time taken: 0.284 seconds, Fetched: 6 row(s)

hive (kjpractice)> show partitions game\_static\_part;

OK

partition

team=BAL/position=Catcher these two partitions are created using insert-select

team=BAL/position=Relief Pitcher

team=MSG/position=Supervision this partition is created my manual movement ( -put )

team=PSS/position=Lead by Load data local inpath command

Time taken: 0.137 seconds, Fetched: 4 row(s)

**Dynamic Partition:**

* + - If hive dynamically decides on the creation of the partitions based on the data then it is dynamic partition.
    - Dynamic partitions can be only loaded using insert select.
    - Dynamic Partition takes more time in loading data compared to static partition as it uses mapreduce.
    - Dynamic partitions are not recommended on high cardinal column as it will create a lot of partitions which will be of no use.
    - You can perform dynamic partition on hive external table and managed table.
    - If you want to use the Dynamic partition in the hive then the mode is in non-strict mode.

## Set the below environmental variables

set hive.exec.dynamic.partition.mode=nonstrict; 🡪 important

* *B default, dynamic partition is strict in hive. Which will not allow us to create dynamic partitions.*
* *It’s because without understanding the data if we create dynamic partition, there are lot of change to apply partition on high cardinal column which will degrade the performance.*
* *However, a query across all partitions could trigger an enormous MapReduce job if the table data and number of partitions are large. A highly suggested safety measure is putting Hive into strict mode, which prohibits queries of partitioned tables without a WHERE clause that filters on partitions. You can set the mode to nonstrict, as in the following session:*

|  |
| --- |
| Concept:Note: Here the partition column is added in the last and included in the select statement in the last. |

Let’s now create the same table as in above example , in a different name.

create table game\_static\_dynm(name varchar(50), height int, weight int, age int)

partitioned by (team varchar(5), position varchar(30))

row format delimited fields terminated by ','

lines terminated by '\n'

stored as textfile

location '/user/hive/kjworks/game\_static\_part';

Note: here with dynamic partition we can only load the data using insert-select method or individual insert method.

The reason behind it is, as like static partition, we will not be have knowledge of how many partitions will be created and the names of partitions as well. So direct load is not possible here.

hive> insert into game\_static\_dynm partition(team,position)

> select name , height, weight, age ,team, position from game\_raw;

After the above insert statement I got the below error, because number of partitions created were more than the max allowed which is configurable.

Caused by: org.apache.hadoop.hive.ql.metadata.HiveFatalException: [Error 20004]: Fatal error occurred when node tried to create too many dynamic partitions. The maximum number of dynamic partitions is controlled by hive.exec.max.dynamic.partitions and hive.exec.max.dynamic.partitions.pernode. Maximum was set to: 100

…………………………………..

FAILED: Execution Error, return code 2 from org.apache.hadoop.hive.ql.exec.mr.MapRedTask

MapReduce Jobs Launched:

Stage-Stage-1: Map: 1 HDFS Read: 0 HDFS Write: 0 FAIL

Total MapReduce CPU Time Spent: 0 msec

The reason behind the above error is, out of two columns which we used in partition, POSITION is high cardinal column.

So I deleted 50% of data from source and tried to run the same command again. The other option is we can increase the threshold value in conf. which would be hive-site.xml

Then again tried to insert the data using insert-select.

hive> insert into game\_static\_dynm partition(team,position)

> select name , height, weight, age ,team, position from game\_raw;

[hduser@Inceptez Datasets]$ hadoop fs -ls /user/hive/kjworks/game\_dynm\_part

Found 5 itemsusage: hive

drwxr-xr-x - hduser supergroup 0 2019-04-14 07:17 /user/hive/kjworks/game\_dynm\_part/team= ANA

drwxr-xr-x - hduser supergroup 0 2019-04-14 07:17 /user/hive/kjworks/game\_dynm\_part/team= BAL

drwxr-xr-x - hduser supergroup 0 2019-04-14 07:17 /user/hive/kjworks/game\_dynm\_part/team= BOS

drwxr-xr-x - hduser supergroup 0 2019-04-14 07:17 /user/hive/kjworks/game\_dynm\_part/team= CLE

drwxr-xr-x - hduser supergroup 0 2019-04-14 07:17 /user/hive/kjworks/game\_dynm\_part/team= CWS

lets explore **“team=ANA”**

[hduser@Inceptez Datasets]$ hadoop fs -ls "/user/hive/kjworks/game\_dynm\_part/team= ANA"

Found 7 items

drwxr-xr-x - hduser supergroup 0 2019-04-14 07:17 /user/hive/kjworks/game\_dynm\_part/team= ANA/position= Catcher

drwxr-xr-x - hduser supergroup 0 2019-04-14 07:17 /user/hive/kjworks/game\_dynm\_part/team= ANA/position= First Baseman

drwxr-xr-x - hduser supergroup 0 2019-04-14 07:17 /user/hive/kjworks/game\_dynm\_part/team= ANA/position= Outfielder

drwxr-xr-x - hduser supergroup 0 2019-04-14 07:17 /user/hive/kjworks/game\_dynm\_part/team= ANA/position= Relief Pitcher

drwxr-xr-x - hduser supergroup 0 2019-04-14 07:17 /user/hive/kjworks/game\_dynm\_part/team= ANA/position= Shortstop

drwxr-xr-x - hduser supergroup 0 2019-04-14 07:17 /user/hive/kjworks/game\_dynm\_part/team= ANA/position= Starting Pitcher

drwxr-xr-x - hduser supergroup 0 2019-04-14 07:17 /user/hive/kjworks/game\_dynm\_part/team= ANA/position= Third Baseman

Now lets explore into **“team=ANA/position= Catcher”**

[hduser@Inceptez Datasets]$ hadoop fs -ls "/user/hive/kjworks/game\_dynm\_part/team= ANA/position= Catcher\*"

Found 1 items

-rw-r--r-- 1 hduser supergroup 66 2019-04-14 07:17 /user/hive/kjworks/game\_dynm\_part/team= ANA/position= Catcher /000000\_0

Browse the file inside **“team=ANA/position= Catcher/000000\_0”**

[hduser@Inceptez Datasets]$ hadoop fs -cat "/user/hive/kjworks/game\_dynm\_part/team= ANA/position= Catcher /000000\_0"

Jose Molina,74,220,32

Jeff Mathis,72,180,24

Mike Napoli,72,205,25

|  |
| --- |
| **NOTE:**  usage: hive *aruguments*  **-d,--define <key=value>** Variable subsitution to apply to hive commands. e.g. -d A=B or --define A=B  **--database <databasename>** Specify the database to use  **-e <quoted-query-string>** SQL from command line  **-f <filename>** SQL from files  **-H,--help** Print help information  **--hiveconf <property=value>** Use value for given property --hivevar <key=value> Variable substitution to apply to  hive commands. e.g. --hivevar A=B  **-i <filename>** Initialization SQL file  **-S,--silent** Silent mode in interactive shell  **-v,--verbose** Verbose mode (echo executed SQL to the console) |

**Additional options:**

**Adding new partitions Partition**

The following command can be used to add new partitions to a table:

hive (kjpractice)> alter table game\_static\_part add partition (team='IBST',position='volunteers') location '/user/hive/kjworks/game\_static\_part/';

hive (kjpractice)> dfs -ls /user/hive/kjworks/game\_static\_part;

Found 3 items

drwxr-xr-x - hduser supergroup 0 2019-04-14 06:33 /user/hive/kjworks/game\_static\_part/team=BAL

drwxr-xr-x - hduser supergroup 0 2019-04-14 06:56 /user/hive/kjworks/game\_static\_part/team=MSG

drwxr-xr-x - hduser supergroup 0 2019-04-14 06:43 /user/hive/kjworks/game\_static\_part/team=PSS

as show above , after we added a new partition if we list the HDFS location, we can see the new partition created. But when we load data the new partitions can be seen in HDFS.

hive (kjpractice)> show partitions game\_static\_part;

OK

partition

team=BAL/position=Catcher

team=BAL/position=Relief Pitcher

team=IBST/position=volunteers

team=MSG/position=Supervision

team=PMR/position=analyst

team=PSS/position=Lead

Time taken: 0.501 seconds, Fetched: 6 row(s)

You can add multiple partitions to a table using the preceding command.

**Rename a Partition**

hive (kjpractice)> alter table game\_static\_part partition (team='BAL',position='Catcher') rename to partition (team='BALL',position='fielder');

OK

Time taken: 1.239 seconds

hive (kjpractice)> show partitions game\_static\_part;

OK

partition

team=BAL/position=Relief Pitcher

team=BALL/position=fielder

team=IBST/position=volunteers

team=MSG/position=Supervision

team=PMR/position=analyst

Time taken: 0.949 seconds, Fetched: 5 row(s)

**Drop a Partition**

hive (kjpractice)> alter table game\_static\_part drop if exists partition (team='PSS');

Dropped the partition team=PSS/position=Lead

OK

Time taken: 7.545 seconds

hive (kjpractice)> dfs -ls /user/hive/kjworks/game\_static\_part;

Found 2 items

drwxr-xr-x - hduser supergroup 0 2019-04-14 06:33 /user/hive/kjworks/game\_static\_part/team=BAL

drwxr-xr-x - hduser supergroup 0 2019-04-14 06:56 /user/hive/kjworks/game\_static\_part/team=MSG

**Bucketing**

Unlike partitioning where each value for the slicer or partition key gets its own space, in clustering a hash is taken for the field value and then distributed across buckets. In the example, we created 10 buckets and are clustering on state. Each bucket then would contain multiple states.

Bucketing is a technique that allows you to decompose your data into more manageable parts, that is, fix the number of buckets. Usually, partitioning provides a way of segregating the data of a Hive table into multiple files or directories. Partitioning is used to increase the performance of queries, but the partitioning technique is efficient only if there are a limited number of partitions. Partitioning doesn't perform well if there are a large number of partitions; for example, we are doing partitioning on a column that has large number of unique values, then there will be a large number of partitions. To overcome the problem of partitioning, Hive provides the concept of bucketing. In bucketing, we specify the fixed number of buckets in which entire data is to be decomposed. Bucketing concept is based on the hashing principle, where same type of keys are always sent to the same bucket. In bucketing, records with the same bucketed columns will always go to the same bucket. When data is inserted into a bucketed table, the following formula is used to derive the bucket into which record should be inserted:

* As the data files are equal sized parts, map-side joins will be faster on bucketed tables than non- bucketed tables. In Map-side join, a mapper processing a bucket of the left table knows that the matching rows in the right table will be in its corresponding bucket, so it only retrieves that bucket .
* Similar to partitioning, bucketed tables provide faster query responses than non-bucketed tables.
* Bucketing concept also provides the flexibility to keep the records in each bucket to be sorted by one or more columns. This makes map-side joins even more efficient

## Partition vs Buckets:

|  |  |
| --- | --- |
| **Partition** | **Buckets** |
| Low cardinal columns (date, country etc)   * Because if it is created on high cardinal column, number of directories will be created which will in order degrade the performance. | Low or high cardinal columns   * It can be applied on both type of columns because it applies hash algorithm to calculate in which bucket which value should go. |
| Creates folder | Creates files |
| Improves where clause performance | Improves Joins and where clause performance |

**Bucket number = hash\_function(bucketing\_column) mod num\_buckets**

Once bucketing is enabled, you can create a bucketed table using the following command:

CREATE [EXTERNAL] TABLE [db\_name.]table\_name [(col\_name data\_type [COMMENT col\_comment], ...)]

CLUSTERED BY (col\_name data\_type [COMMENT col\_comment], ...) INTO N BUCKETS;

## Create bucketed table over the partitioned table

Use the following command set **hive.enforce.bucketing = true**; allows the correct number of reducers and the cluster by column to be automatically selected based on the table.

set hive.enforce.bucketing = true ;

## Let’s see with an example (only bucketing)

To create a table with buckets, we need to use the key work clustered by.

CREATE TABLE `game\_bucket`(

`name` varchar(50),

`team` varchar(5),

`position` varchar(30),

`height` int,

`weight` int,

`age` int)

clustered by (position) sorted by (name) into 10 buckets

ROW FORMAT DELIMITED

FIELDS TERMINATED BY ','

LINES TERMINATED BY '\n'

STORED AS INPUTFORMAT

'org.apache.hadoop.mapred.TextInputFormat'

OUTPUTFORMAT

'org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat'

LOCATION

'hdfs://localhost:54310/user/hive/kjworks/game\_bucket';

|  |
| --- |
| **Concept:**   * Bucketing is a method to evenly distribute the data across many files. * Create multiple buckets and then place each record into one of the buckets based on some logic mostly some hashing algorithm. * Bucketing feature of Hive can be used to distribute/organize the table/partition data into multiple files such that similar records are present in the same file. * While creating a Hive table, a user needs to give the columns to be used for bucketing and the number of buckets to store the data into. Which records go to which bucket are decided by the Hash value of columns used for bucketing. * **[Hash(column(s))] MOD [Number of buckets]** * Hash value for different columns types is calculated differently. For **int** columns, the hash value is equal to the value of int. * For **String** columns, the hash value is calculated using some computation on each character present in the String. * Data for each bucket is stored in a separate HDFS file under the table directory on HDFS. * Inside each bucket, we can define the arrangement of data by providing the SORT BY column while creating the table. |

Let’s load the data using insert-select way.

insert into table game\_bucket

select name, team, position, height, weight, age from game\_raw\_part1;

now lets check the HDFS location

hive (kjpractice)> dfs -ls /user/hive/kjworks/game\_bucket;

Found 10 items

-rw-r--r-- 1 hduser supergroup 0 2019-04-15 23:14 /user/hive/kjworks/game\_bucket/000000\_0

-rw-r--r-- 1 hduser supergroup 0 2019-04-15 23:14 /user/hive/kjworks/game\_bucket/000001\_0

-rw-r--r-- 1 hduser supergroup 5743 2019-04-15 23:14 /user/hive/kjworks/game\_bucket/000002\_0

-rw-r--r-- 1 hduser supergroup 2183 2019-04-15 23:14 /user/hive/kjworks/game\_bucket/000003\_0

-rw-r--r-- 1 hduser supergroup 0 2019-04-15 23:14 /user/hive/kjworks/game\_bucket/000004\_0

-rw-r--r-- 1 hduser supergroup 302 2019-04-15 23:14 /user/hive/kjworks/game\_bucket/000005\_0

-rw-r--r-- 1 hduser supergroup 5441 2019-04-15 23:14 /user/hive/kjworks/game\_bucket/000006\_0

-rw-r--r-- 1 hduser supergroup 6894 2019-04-15 23:14 /user/hive/kjworks/game\_bucket/000007\_0

-rw-r--r-- 1 hduser supergroup 0 2019-04-15 23:14 /user/hive/kjworks/game\_bucket/000008\_0

-rw-r--r-- 1 hduser supergroup 878 2019-04-15 23:14 /user/hive/kjworks/game\_bucket/000009\_0

All records with Hash(position) mod 10 == 0 goes into this file ../ 000000\_0 .

Hash(position) mod 10 == 1 goes into this file ../ 000000\_1 .

Hash(position) mod 10 == 2 goes into this file ../ 000000\_2 as show below .

hive (kjpractice)> dfs -cat /user/hive/kjworks/game\_bucket/000002\_0;

Jose Vidro, SEA, Second Baseman ,71,193,33

Zach Miner, DET, Starting Pitcher ,75,200,25

Zach Duke, PIT, Starting Pitcher ,74,207,24

Yoslan Herrera, PIT, Starting Pitcher ,74,200,26

Woody Williams, HOU, Starting Pitcher ,72,200,41

Wandy Rodriguez, HOU, Starting Pitcher ,71,160,28

...................................

....................

hive (kjpractice)> dfs -cat /user/hive/kjworks/game\_bucket/000003\_0;

Matt Treanor, FLA, Catcher ,74,220,31

Josh Wilson, WAS, Shortstop ,73,178,26

Josh Bard, SD , Catcher ,75,215,29

Rob Bowen, SD , Catcher ,75,216,26

Yadier Molina, STL, Catcher ,71,225,25

Oswaldo Navarro, SEA, Shortstop ,72,150,22

Troy Tulowitzki, COL, Shortstop ,75,205,22

.........................

.............

Lets load data from another table to the same bucketing table:

insert into table game\_bucket

select name, team, position, height, weight, age from game\_raw\_part2;

Look at the HDFS location:

hive (kjpractice)> dfs -ls /user/hive/kjworks/game\_bucket ;

Found 20 items

-rw-r--r-- 1 hduser supergroup 0 2019-04-15 23:14 /user/hive/kjworks/game\_bucket/000000\_0

-rw-r--r-- 1 hduser supergroup 0 2019-04-15 23:24 /user/hive/kjworks/game\_bucket/000000\_0\_copy\_1

-rw-r--r-- 1 hduser supergroup 0 2019-04-15 23:14 /user/hive/kjworks/game\_bucket/000001\_0

-rw-r--r-- 1 hduser supergroup 0 2019-04-15 23:24 /user/hive/kjworks/game\_bucket/000001\_0\_copy\_1

-rw-r--r-- 1 hduser supergroup 5743 2019-04-15 23:14 /user/hive/kjworks/game\_bucket/000002\_0

-rw-r--r-- 1 hduser supergroup 7919 2019-04-15 23:24 /user/hive/kjworks/game\_bucket/000002\_0\_copy\_1

-rw-r--r-- 1 hduser supergroup 2183 2019-04-15 23:14 /user/hive/kjworks/game\_bucket/000003\_0

-rw-r--r-- 1 hduser supergroup 3099 2019-04-15 23:24 /user/hive/kjworks/game\_bucket/000003\_0\_copy\_1

-rw-r--r-- 1 hduser supergroup 0 2019-04-15 23:14 /user/hive/kjworks/game\_bucket/000004\_0

-rw-r--r-- 1 hduser supergroup 0 2019-04-15 23:24 /user/hive/kjworks/game\_bucket/000004\_0\_copy\_1

-rw-r--r-- 1 hduser supergroup 302 2019-04-15 23:14 /user/hive/kjworks/game\_bucket/000005\_0

-rw-r--r-- 1 hduser supergroup 585 2019-04-15 23:24 /user/hive/kjworks/game\_bucket/000005\_0\_copy\_1

-rw-r--r-- 1 hduser supergroup 5441 2019-04-15 23:14 /user/hive/kjworks/game\_bucket/000006\_0

-rw-r--r-- 1 hduser supergroup 5505 2019-04-15 23:25 /user/hive/kjworks/game\_bucket/000006\_0\_copy\_1

-rw-r--r-- 1 hduser supergroup 6894 2019-04-15 23:14 /user/hive/kjworks/game\_bucket/000007\_0

-rw-r--r-- 1 hduser supergroup 8029 2019-04-15 23:25 /user/hive/kjworks/game\_bucket/000007\_0\_copy\_1

-rw-r--r-- 1 hduser supergroup 0 2019-04-15 23:14 /user/hive/kjworks/game\_bucket/000008\_0

-rw-r--r-- 1 hduser supergroup 0 2019-04-15 23:25 /user/hive/kjworks/game\_bucket/000008\_0\_copy\_1

-rw-r--r-- 1 hduser supergroup 878 2019-04-15 23:14 /user/hive/kjworks/game\_bucket/000009\_0

-rw-r--r-- 1 hduser supergroup 1207 2019-04-15 23:25 /user/hive/kjworks/game\_bucket/000009\_0\_copy\_1

hive (kjpractice)> dfs -cat /user/hive/kjworks/game\_bucket/000002\_0\_copy\_1;

A.J. Burnett, TOR, Starting Pitcher ,76,230,30

Jose Valentin, NYM, Second Baseman ,70,195,37

Jose Lopez, SEA, Second Baseman ,74,170,23

Jose Contreras, CWS, Starting Pitcher ,76,224,35

Jose Castillo, PIT, Second Baseman ,73,200,26

Jorge De La Rosa, KC , Starting Pitcher ,73,190,26

Jorge Cantu, TB , Second Baseman ,73,184,25

.......................

................

hive (kjpractice)> dfs -cat /user/hive/kjworks/game\_bucket/000003\_0\_copy\_1;

Chris Coste, PHI, Catcher ,73,200,34

Angel Berroa, KC , Shortstop ,72,175,29

Erick Aybar, ANA, Shortstop ,70,170,23

Jason LaRue, KC , Catcher ,71,200,33

Alex Cintron, CWS, Shortstop ,74,199,28

Ivan Rodriguez, DET, Catcher ,69,218,35

Chad Moeller, CIN, Catcher ,75,210,32

Gerald Laird, TEX, Catcher ,74,220,27

...........................

.............

Now lets load the same data which we tried for the first time.

Expectation is the data distribution across the buckets should be as same as the first time.

insert into table game\_bucket

select name, team, position, height, weight, age from game\_raw\_part1;

hive (kjpractice)> dfs -ls /user/hive/kjworks/game\_bucket;

Found 30 items

-rw-r--r-- 1 hduser supergroup 0 2019-04-15 23:14 /user/hive/kjworks/game\_bucket/000000\_0

-rw-r--r-- 1 hduser supergroup 0 2019-04-15 23:24 /user/hive/kjworks/game\_bucket/000000\_0\_copy\_1

-rw-r--r-- 1 hduser supergroup 0 2019-04-15 23:35 /user/hive/kjworks/game\_bucket/000000\_0\_copy\_2

-rw-r--r-- 1 hduser supergroup 0 2019-04-15 23:14 /user/hive/kjworks/game\_bucket/000001\_0

-rw-r--r-- 1 hduser supergroup 0 2019-04-15 23:24 /user/hive/kjworks/game\_bucket/000001\_0\_copy\_1

-rw-r--r-- 1 hduser supergroup 0 2019-04-15 23:35 /user/hive/kjworks/game\_bucket/000001\_0\_copy\_2

-rw-r--r-- 1 hduser supergroup 5743 2019-04-15 23:14 /user/hive/kjworks/game\_bucket/000002\_0

-rw-r--r-- 1 hduser supergroup 7919 2019-04-15 23:24 /user/hive/kjworks/game\_bucket/000002\_0\_copy\_1

-rw-r--r-- 1 hduser supergroup 5743 2019-04-15 23:35 /user/hive/kjworks/game\_bucket/000002\_0\_copy\_2

-rw-r--r-- 1 hduser supergroup 2183 2019-04-15 23:14 /user/hive/kjworks/game\_bucket/000003\_0

-rw-r--r-- 1 hduser supergroup 3099 2019-04-15 23:24 /user/hive/kjworks/game\_bucket/000003\_0\_copy\_1

-rw-r--r-- 1 hduser supergroup 2183 2019-04-15 23:35 /user/hive/kjworks/game\_bucket/000003\_0\_copy\_2

-rw-r--r-- 1 hduser supergroup 0 2019-04-15 23:14 /user/hive/kjworks/game\_bucket/000004\_0

-rw-r--r-- 1 hduser supergroup 0 2019-04-15 23:24 /user/hive/kjworks/game\_bucket/000004\_0\_copy\_1

-rw-r--r-- 1 hduser supergroup 0 2019-04-15 23:35 /user/hive/kjworks/game\_bucket/000004\_0\_copy\_2

hive (kjpractice)> dfs -cat /user/hive/kjworks/game\_bucket/000002\_0\_copy\_2;

Jose Vidro, SEA, Second Baseman ,71,193,33

Zach Miner, DET, Starting Pitcher ,75,200,25

Zach Duke, PIT, Starting Pitcher ,74,207,24

Yoslan Herrera, PIT, Starting Pitcher ,74,200,26

Woody Williams, HOU, Starting Pitcher ,72,200,41

Wandy Rodriguez, HOU, Starting Pitcher ,71,160,28

Wade Miller, CHC, Starting Pitcher ,74,220,30

Virgil Vasquez, DET, Starting Pitcher ,75,205,25

....................

..............

|  |
| --- |
| **Concept:**   * Hash(position) mod 10 == 2 goes into this file ../ 000000\_2 * What we learnt is , if the value from of position is ‘Catcher’, Then hash(Catcher) mod 10 == 2 then that record goes to bucket 2 (000000\_2). * It doesn’t mean that, only record with position=’Catcher’ will go to bucket 2. * The word ‘Catcher’ will be converted to some hash *int* value and mod will be calculated using that value. * If hash(int\_value) mod no\_of\_buckets == 2 then it goes to bucket no 2. * If hash(int\_value of some other string value like Catcher) mod no\_of\_buckets ==2, it goes to bucket no 2 again. |

## Now lets see any example : Partition and buckets together.

Let’s create a table which has both partition and buckets.

CREATE TABLE `game\_partition\_bucket`(

`name` varchar(50),

`height` int,

`weight` int,

`age` int)

partitioned by ( team varchar(5), position varchar(30))

clustered by (name) sorted by (name) into 5 buckets

ROW FORMAT DELIMITED

FIELDS TERMINATED BY ','

LINES TERMINATED BY '\n'

STORED AS INPUTFORMAT

'org.apache.hadoop.mapred.TextInputFormat'

OUTPUTFORMAT

'org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat'

LOCATION

'hdfs://localhost:54310/user/hive/kjworks/game\_partition\_bucket';

Lets load the data into this partition-bucket table.

hive (kjpractice)> insert into table game\_partition\_bucket partition(team=' BAL',position=' Catcher ')

> select name, height, weight, age from game\_raw\_part1;

[hduser@Inceptez kanna1]$ hadoop fs -ls /user/hive/kjworks/game\_partition\_bucket

drwxr-xr-x - hduser supergroup 0 2019-04-16 00:35 /user/hive/kjworks/game\_partition\_bucket/team= BAL

[hduser@Inceptez kanna1]$ hadoop fs -ls '/user/hive/kjworks/game\_partition\_bucket/team= BAL'

Found 1 items

drwxr-xr-x - hduser supergroup 0 2019-04-16 00:37 /user/hive/kjworks/game\_partition\_bucket/team= BAL/position= Catcher

[hduser@Inceptez kanna1]$ hadoop fs -ls "/user/hive/kjworks/game\_partition\_bucket/team= BAL/position= Catcher "

Found 5 items

-rw-r--r-- 1 hduser supergroup 2188 2019-04-16 00:37 /user/hive/kjworks/game\_partition\_bucket/team= BAL/position= Catcher /000000\_0

-rw-r--r-- 1 hduser supergroup 2102 2019-04-16 00:37 /user/hive/kjworks/game\_partition\_bucket/team= BAL/position= Catcher /000001\_0

-rw-r--r-- 1 hduser supergroup 2127 2019-04-16 00:37 /user/hive/kjworks/game\_partition\_bucket/team= BAL/position= Catcher /000002\_0

-rw-r--r-- 1 hduser supergroup 2279 2019-04-16 00:37 /user/hive/kjworks/game\_partition\_bucket/team= BAL/position= Catcher /000003\_0

-rw-r--r-- 1 hduser supergroup 2113 2019-04-16 00:37 /user/hive/kjworks/game\_partition\_bucket/team= BAL/position= Catcher /000004\_0

[hduser@Inceptez kanna1]$ hadoop fs -cat "/user/hive/kjworks/game\_partition\_bucket/team= BAL/position= Catcher /000002\_0"

Mike Stanton,73,215,40

Seth McClung,78,235,26

Sean Marshall,79,205,25

Sean Burnett,71,190,24

Scott Thorman,75,200,25

Scott Spiezio,74,220,34

.................

...........

[hduser@Inceptez kanna1]$ hadoop fs -ls "/user/hive/kjworks/game\_partition\_bucket/team= BAL/position= Catcher "

Found 10 items

-rw-r--r-- 1 hduser supergroup 2188 2019-04-16 00:37 /user/hive/kjworks/game\_partition\_bucket/team= BAL/position= Catcher /000000\_0

-rw-r--r-- 1 hduser supergroup 2188 2019-04-16 01:13 /user/hive/kjworks/game\_partition\_bucket/team= BAL/position= Catcher /000000\_0\_copy\_1

-rw-r--r-- 1 hduser supergroup 2102 2019-04-16 00:37 /user/hive/kjworks/game\_partition\_bucket/team= BAL/position= Catcher /000001\_0

-rw-r--r-- 1 hduser supergroup 2102 2019-04-16 01:13 /user/hive/kjworks/game\_partition\_bucket/team= BAL/position= Catcher /000001\_0\_copy\_1

-rw-r--r-- 1 hduser supergroup 2127 2019-04-16 00:37 /user/hive/kjworks/game\_partition\_bucket/team= BAL/position= Catcher /000002\_0

-rw-r--r-- 1 hduser supergroup 2127 2019-04-16 01:13 /user/hive/kjworks/game\_partition\_bucket/team= BAL/position= Catcher /000002\_0\_copy\_1

-rw-r--r-- 1 hduser supergroup 2279 2019-04-16 00:37 /user/hive/kjworks/game\_partition\_bucket/team= BAL/position= Catcher /000003\_0

-rw-r--r-- 1 hduser supergroup 2279 2019-04-16 01:13 /user/hive/kjworks/game\_partition\_bucket/team= BAL/position= Catcher /000003\_0\_copy\_1

-rw-r--r-- 1 hduser supergroup 2113 2019-04-16 00:37 /user/hive/kjworks/game\_partition\_bucket/team= BAL/position= Catcher /000004\_0

-rw-r--r-- 1 hduser supergroup 2113 2019-04-16 01:13 /user/hive/kjworks/game\_partition\_bucket/team= BAL/position= Catcher /000004\_0\_copy\_1

[hduser@Inceptez kanna1]$ hadoop fs -cat "/user/hive/kjworks/game\_partition\_bucket/team= BAL/position= Catcher /000002\_0\_copy\_1"

Mike Stanton,73,215,40

Seth McClung,78,235,26

Sean Marshall,79,205,25

Sean Burnett,71,190,24

Scott Thorman,75,200,25

Scott Spiezio,74,220,34

Scott Schoeneweis,72,195,33

...................

......

Now let’s do CPU timing comparison between the tables

* Without partition/bucket
* With partition
* With partition and bucket.

Simple select from bucketed table:

Select \* from game\_partition\_bucket where team=’ BAL’ and position= ‘ Catcher ‘;

Below query is to select only bucket 3:

Select name, team,height,weight from game\_partition\_bucket tablesample(bucket 3 out of 10);

Below query is to take a sample of particular percentage in the entire data:

Select name, team,height,weight from game\_partition\_bucket tablesample(0.1 percent);

**============================================**

**Remote Metastore configuration**

**============================================**

1. MYSQL DB changes:

=================

Start the mysql service

------------------------------

## sudo service mysqld start

1. Login to mysql using root user to create metastore db:

------------------------------------------------------------------------

## mysql -u root -p

Enter password: **root**

## create database metastore; USE metastore;

**SOURCE /usr/local/hive/scripts/metastore/upgrade/mysql/hive-schema-0.14.0.mysql.sql; show tables;**

**GRANT all on \*.\* to 'hiveuser'@localhost identified by 'hivepassword'; flush privileges;**

**quit;**

1. **HIVE-SITE.XML configuration to connect to remote metastore**

**===========================================================**

**Copy the hive-site\_rms\_transction.xml config file provided to /usr/local/hive/conf**

**-----------------------------------------------------------------------------------------------------------------------------------**

**cp /home/hduser/hive/hive-site\_rms\_transction.xml /usr/local/hive/conf/hive-site.xml**

* Here so far we were using hive\_site\_template which was in the local files sytem /usr/local/hive
* It was using the data base DERBY as default data base to store the HIVE’s meta data.
* But in production DERBY will not be used because of its low range of usability.
* so usually in production some other larger data base such as mysql , sql server will be used to store the metastore remotely.
* Derby will be in the local system. Remote meta store can be accessed in distributed fashion.

Copy the dependent mysql connector jdbc jar to get hive connected with remote metastore

## cp -p /home/hduser/install/mysql-connector-java.jar /usr/local/hive/lib/

* Need to copy the jdbc connector of the data base which we choose to store the meta store.

**Note : Below are the additions made in the hive-site.xml**

<property>

<name>hive.metastore.uris</name>

<value>thrift://localhost:9083</value>

<description>IP address (or fully-qualified domain name) and port of the metastore host</description>

< /property>

<property>

<name>javax.jdo.option.ConnectionURL</name>

<value>jdbc:mysql://localhost/metastore</value>

<description>the URL of the MySQL database</description>

< /property>

<property>

<name>javax.jdo.option.ConnectionDriverName</name>

<value>com.mysql.jdbc.Driver</value>

</property>

<property>

<name>javax.jdo.option.ConnectionUserName</name>

<value>hiveuser</value>

</property>

<property>

<name>javax.jdo.option.ConnectionPassword</name>

<value>hivepassword</value>

</property>

<property>

<name>datanucleus.autoCreateSchema</name>

< value>false</value>

</property>

<property>

< name>hive.stats.dbclass</name>

<value>jdbc:mysql</value>

</property>

## Running the services before starting hive session

**==================================================**

**Note: Run the below command in a different terminal and leave the terminal as is hive --service metastore**

**Note: Use the below commands to start hive always: (when ever if we need to start again)**

**sudo service mysqld start hive --service metastore**

Starting Hive CLI with remote metastore

## hive

**Export hive data to a file:**

**---------------------------------**

**insert overwrite local directory '/home/hduser/exptxnrecords' row format delimited fields terminated by ','**

**select txnno,txndate,custno,amount, product,city,state,spendby from txnrecords where category='Games';**

**Serialization and deserialization formats and data types**

Serialization and deserialization formats are popularly known as **SerDes**. Hive allows the framework to

read or write data in a particular format. These formats parse the structured, semistructured or unstructured data bytes stored in HDFS in accordance with the schema definition of Hive tables. Hive provides a set of in-built SerDes and also allows the user to create custom SerDes based on their data definition.

These are as follows:

Json Serde Xml Serde

Lazy Simple Serde Csv Serde

Regex Serde

ORC

AVRO

PARQUET

JSON

**Banking xml data for serde:**

**Copy the serde jars and other dependent jars to hive/lib**

**cp -p /home/hduser/install/hivexmlserde-1.0.5.3.jar /usr/local/hive/lib/**

* Each SERDE files has its own jar lib file to tell HIVE how to use that SERDE file.
* Above jar file is for XML file. Like wise we have jar files for other serde files

Lets consider the below xml data as source. And we need to load it into a table.

<record customer\_id="0000-JTALA">

<income>200000</income>

<demographics>

<gender>F</gender>

<agecat>1</agecat>

<edcat>1</edcat>

<jobcat>2</jobcat>

<empcat>2</empcat>

<retire>0</retire>

<jobsat>1</jobsat>

<marital>1</marital>

<spousedcat>1</spousedcat>

<residecat>4</residecat>

<homeown>0</homeown>

<hometype>2</hometype>

<addresscat>2</addresscat>

</demographics>

<financial>

<income>18</income>

<creddebt>1.003392</creddebt>

<othdebt>2.740608</othdebt>

<default>0</default>

</financial>

</record>

<record customer\_id="0000-KDELL">

<income>10000</income>

<demographics>

<gender>M</gender>

<agecat>2</agecat>

<edcat>1</edcat>

<jobcat>2</jobcat>

<empcat>3</empcat>

<retire>1</retire>

<jobsat>1</jobsat>

<marital>1</marital>

<spousedcat>1</spousedcat>

<residecat>4</residecat>

<homeown>0</homeown>

<hometype>3</hometype>

<addresscat>2</addresscat>

</demographics>

<financial>

<income>20</income>

<creddebt>1.002292</creddebt>

<othdebt>2.113208</othdebt>

<default>0</default>

</financial>

</record>

And lets create the table and load it

**CREATE TABLE xml\_bank(customer\_id STRING, income BIGINT, demographics map<string,string>, financial map<string,string>)**

**ROW FORMAT SERDE 'com.ibm.spss.hive.serde2.xml.XmlSerDe' WITH SERDEPROPERTIES (**

**"column.xpath.customer\_id"="/record/@customer\_id", "column.xpath.income"="/record/income/text()", "column.xpath.demographics"="/record/demographics/\*", "column.xpath.financial"="/record/financial/\*"**

**)**

**STORED AS**

**INPUTFORMAT 'com.ibm.spss.hive.serde2.xml.XmlInputFormat' OUTPUTFORMAT 'org.apache.hadoop.hive.ql.io.IgnoreKeyTextOutputFormat' location '/user/hduser/xmlserdebank/'**

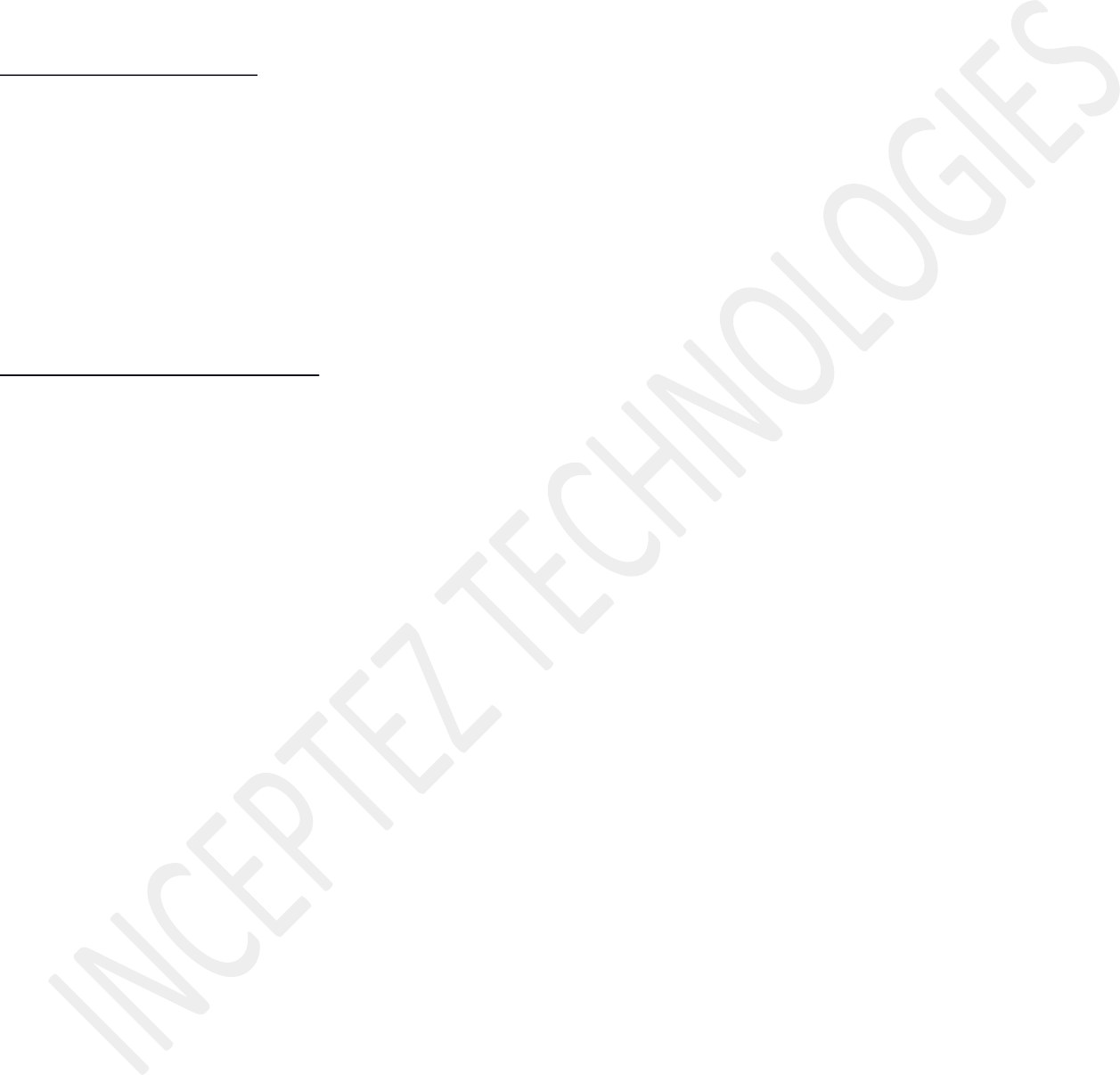
**TBLPROPERTIES (**

**"xmlinput.start"="<record customer", "xmlinput.end"="</record>" );**

**Copy the file to HDFS**

**hadoop fs -mkdir /user/hduser/xmlserdebank/**

**hadoop fs -put /home/hduser/hive/data/bankserde.xml /user/hduser/xmlserdebank/**



**select \* from xml\_bank;**

**UDF using Python code:**

**create table customerall(custno string, firstname string, lastname string, age int,profession string) row format delimited fields terminated by ',';**

**load data local inpath '/home/hduser/hive/data/custs' into table customerall; SELECT \* FROM customerall limit 10;**

**Create the below python udf:**

**vi /home/hduser/mask.py**

**#!/usr/bin/env pythonimport sys**

**import string import hashlib**

**while True:**

**line = sys.stdin.readline() if not line:**

**break**

**line = string.strip(line, "\n ")**

**custno, firstname, lastname, age, profession = string.split(line, "\t")**

**print "\t".join([custno, firstname,lastname,hashlib.sha256(age).hexdigest(), hashlib.md5(profession).hexdigest()])**

**add FILE /home/hduser/mask.py;**

**select transform(custno,firstname,lastname,age,profession) using 'python /home/hduser/mask.py' as (custno,firstname,lastname,age,profession) from customerall limit 10;**

========================

## Hive Sqoop Integeration

========================

**Create a hive table using sqoop as like mysql table**

**sqoop create-hive-table --connect jdbc:mysql://127.0.0.1/test --username hiveuser --password hivepassword --table customer \**

**--hive-table retail.customer;**

**Import data from sqoop directly to hive customer table**

**-------------------------------------------------------------------------**

**hadoop fs -rmr /user/hduser/customer**

**sqoop import --connect jdbc:mysql://127.0.0.1/test --username hiveuser –password hivepassword --table customer \**

**--hive-import -hive-table retail.customer\_data -m 1**

Importing data into Hive

sqoop import --connect jdbc:mysql://localhost/fleetdb \

--username root --password root \

--table driver \

-m 3 --split-by driverId \

--target-dir '/user/fleetdb/' --delete-target-dir \

--hive-import \

--create-hive-table \

--hive-table fleetdb.driver;

[hduser@Inceptez ~]$ hadoop fs -ls /user/hive/warehouse

Found 10 items

drwxr-xr-x - hduser supergroup 0 2019-06-20 07:55 /user/hive/warehouse/\_tmp.txnrecord\_es

drwxr-xr-x - hduser supergroup 0 2019-06-24 10:17 /user/hive/warehouse/\_tmp.txnrecord\_es\_id

drwxr-xr-x - hduser supergroup 0 2019-03-30 04:05 /user/hive/warehouse/custdb.db

drwxr-xr-x - hduser supergroup 0 2019-03-26 23:16 /user/hive/warehouse/customerall

drwxr-xr-x - hduser supergroup 0 2019-06-25 23:37 /user/hive/warehouse/fleetdb.db

drwxr-xr-x - hduser supergroup 0 2019-03-29 20:49 /user/hive/warehouse/retail.db

drwxrwxr-x - hduser supergroup 0 2019-03-24 21:13 /user/hive/warehouse/retail1.db

drwxr-xr-x - hduser supergroup 0 2019-06-20 07:47 /user/hive/warehouse/txnrecord

drwxr-xr-x - hduser supergroup 0 2019-06-20 07:50 /user/hive/warehouse/txnrecord\_es

drwxr-xr-x - hduser supergroup 0 2019-06-24 10:17 /user/hive/warehouse/txnrecord\_es\_id

Even though we give target-dir, the data is imported into the database folder created in the warehouse folder.

Lets see inside the table folder.

[hduser@Inceptez ~]$ hadoop fs -ls /user/hive/warehouse/fleetdb.db/driver

Found 3 items

-rw-r--r-- 1 hduser supergroup 636 2019-06-25 23:37 /user/hive/warehouse/fleetdb.db/driver/part-m-00000

-rw-r--r-- 1 hduser supergroup 652 2019-06-25 23:37 /user/hive/warehouse/fleetdb.db/driver/part-m-00001

-rw-r--r-- 1 hduser supergroup 617 2019-06-25 23:37 /user/hive/warehouse/fleetdb.db/driver/part-m-00002

When we again run the same command, the data is appended to the same folder “driver” but as new files.

[hduser@Inceptez ~]$ hadoop fs -ls /user/hive/warehouse/fleetdb.db/driver

Found 6 items

-rw-r--r-- 1 hduser supergroup 636 2019-06-25 23:37 /user/hive/warehouse/fleetdb.db/driver/part-m-00000

-rw-r--r-- 1 hduser hadoop 636 2019-06-25 23:52 /user/hive/warehouse/fleetdb.db/driver/part-m-00000\_copy\_1

-rw-r--r-- 1 hduser supergroup 652 2019-06-25 23:37 /user/hive/warehouse/fleetdb.db/driver/part-m-00001

-rw-r--r-- 1 hduser hadoop 652 2019-06-25 23:52 /user/hive/warehouse/fleetdb.db/driver/part-m-00001\_copy\_1

-rw-r--r-- 1 hduser supergroup 617 2019-06-25 23:37 /user/hive/warehouse/fleetdb.db/driver/part-m-00002

-rw-r--r-- 1 hduser hadoop 617 2019-06-25 23:52 /user/hive/warehouse/fleetdb.db/driver/part-m-00002\_copy\_1

**Below are the major commands used in the –hive-import..**

| **Sqoop Command Option** | **Description** |
| --- | --- |
| --hive-home <directory> | Overrides $HIVE\_HOME. |
| --hive-import | Imports tables into Hive using Hive's default delimiters if none are explicitly set. |
| --hive-overwrite | Overwrites existing data in the Hive table. |
| --create-hive-table | Creates a hive table during the operation. If this option is set and the Hive table already exists, the job will fail. Set to false by default. |
| --hive-table <table\_name> | Specifies the table name to use when importing data into Hive. |
| --hive-drop-import-delims | Drops the delimiters \n, \r, and \01 from string fields when importing data into Hive. |
| --hive-delims-replacement | Replaces the delimiters \n, \r, and \01 from strings fields with a user-defined string when importing data into Hive. |
| --hive-partition-key | Specifies the name of the Hive field on which a sharded database is partitioned. |
| --hive-partition-value <value> | A string value that specifies the partition key for data imported into Hive. |
| --map-column-hive <map> | Overrides the default mapping from SQL type to Hive type for configured columns. |

Importing data into HBase

sqoop import --connect jdbc:mysql://localhost/fleetdb --username root --password root \

--table driver -m 3 --split-by driverId \

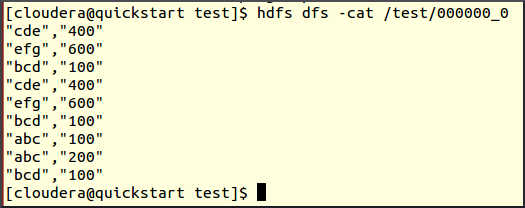
--hbase-table fleetdb \

--column-family driver \

--hbase-row-key driverId ;

Hadoop – Export Hive Data with Quoted Values into Flat File and Example

In general, quoted values are values which are enclosed in single or double quotation marks. Usually, quoted values files are system generated where each and every fields in flat files is either enclosed in **SINGLE** or **DOUBLE** quotation mark. In this article, we will check how to e**xport Hadoop Hive data with quoted values into flat file**such as CSV file format.



Quoted values are such as:

“1”, “Johny”, “1, NYC”

“2”, “Tim”, “10, DC”

**Export Hive Data with Quoted Values into Flat File**

The possible solution could be [create external table](http://dwgeek.com/hive-create-external-tables-examples.html/) by using Hive **CSV SerDe** (Serializer/Deserializer). It provides a way to specify custom delimiters, quote, and escape characters.

Below is the Hive external table example that you can use to unload table with values enclosed in quotation mark:

CREATE EXTERNAL TABLE quoted\_file(name string, amount int)

ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.OpenCSVSerde'

WITH SERDEPROPERTIES (

"separatorChar" = ",",

"quoteChar" = "\""

)

STORED AS TEXTFILE

LOCATION '/test';

Below is the Hive external table example that you can use to unload table with values enclosed in quotation mark:

CREATE EXTERNAL TABLE quoted\_file(name string, amount int)

ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.OpenCSVSerde'

WITH SERDEPROPERTIES (

"separatorChar" = ",",

"quoteChar" = "\""

)

STORED AS TEXTFILE

LOCATION '/test';

Text file:

* Behaviour – each line is a record/data, and line are terminated by a newline char
* Read/write – good write performance but slow reads
* Compression – do not support block compression.
* Splittable – text files are inherently splittable on \n character
* Schema evolution – limited schema evolution (new fields can only be appended to existing fields while old fields can never be deleted)

Sequence file:

* Behaviour – each record is stored as a key value pair in binary format
* Read/write – good write performance than text files
* Compression – support block compression
* Splitable – are splittble
* Schema evol – limited schema evol similar to text file (doesn’t store metadata. New fileds can only be appended to existing fields while existing fields can never be delted)

AVRO:

* Behaviour – it’s a file format plus a serialization and deserialization framework. Avro uses JSON for defining data types and serializes data in a compact binary format.
* Read/write – average read/write perforamce
* Compression – support block compression
* Splittable – avro files are splittable
* Schema evol – was mainly designed for full schema evolution. It stores the metadata along with the file. Its best suited if the files schema will frequently change. Fileds can renamed , added, deleted while old files can still be read with the new schema.

Columnar file formats:

* In columnar file format instead of just storing rows of data adjacent to one another we also store column values adjacent to each other.
* So datasets are partitioned both horizontally and vertically

Types:

RC Files: (row columnar)

* Behaviour – these are flat files consisting of binary key/ value pairs, and it shares much similarlity with sequence file.
* Read/write – was developed for faster reads but with a compromise with write performance
* Compression – provides significant block compression, can be compressed with high compression ratios
* Splittable – avro files are splittable
* Schema evol – was mainly designed for faster reads. So no schema evolution.

ORC files:

Optimized RC files. Better version of RC.

Parquet:

Most famous and standard in real time projects.

* Behaviour – it’s a columnar file format, similar to RC and ORC. Parquet stores nested data structure in a flat columnar format.
* Read/write – faster reads with slow writes.
* Compression – supports compression mostly with snappy algorithm.
* Splittable – parquet files are conditionally splittable.
* Schema evol – limited schema evol similar to text file (doesn’t store metadata. New fileds can only be appended to existing fields while existing fields can never be delted)