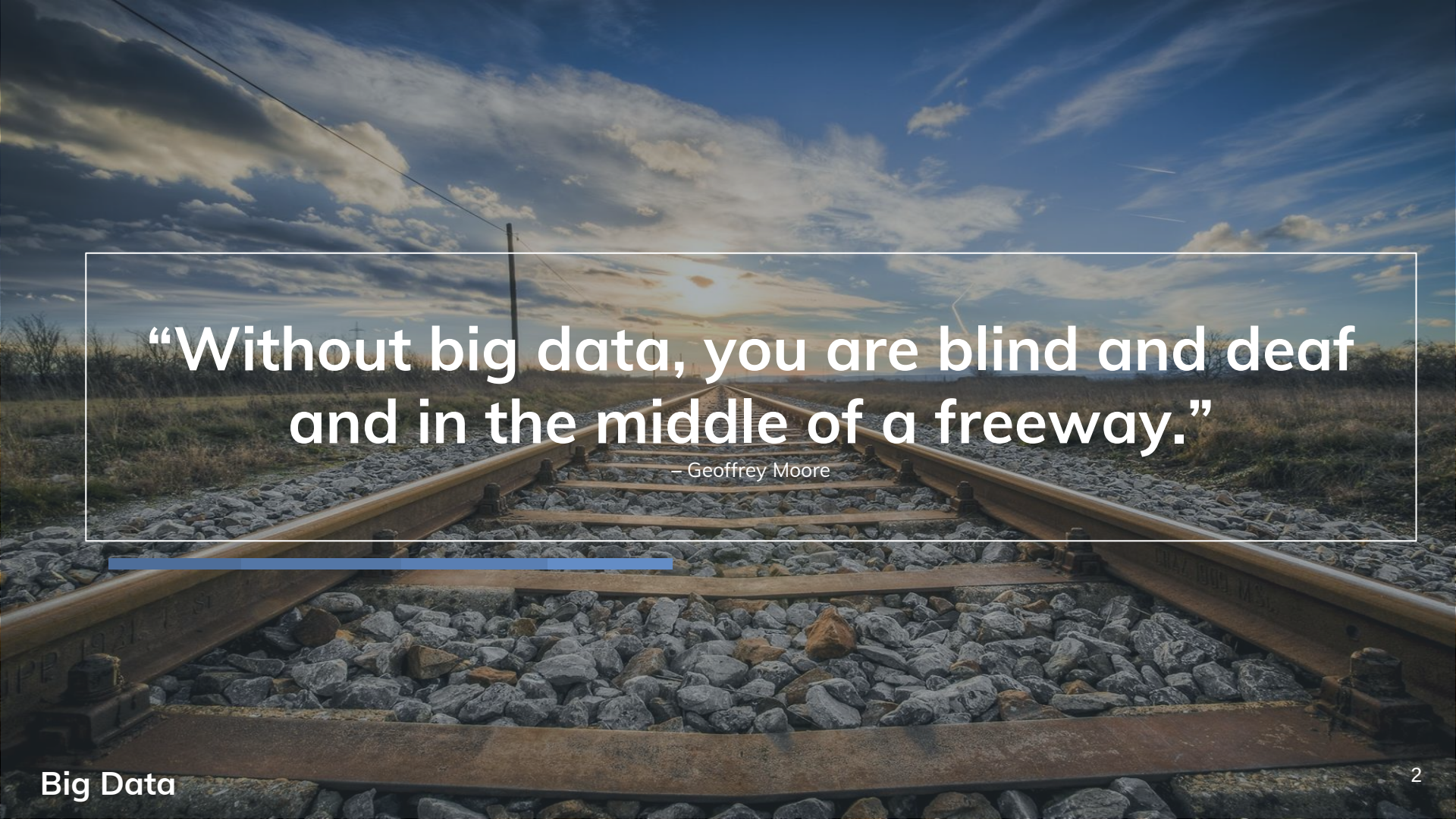
Big Data

Data Loading Tools

Trong-Hop Do

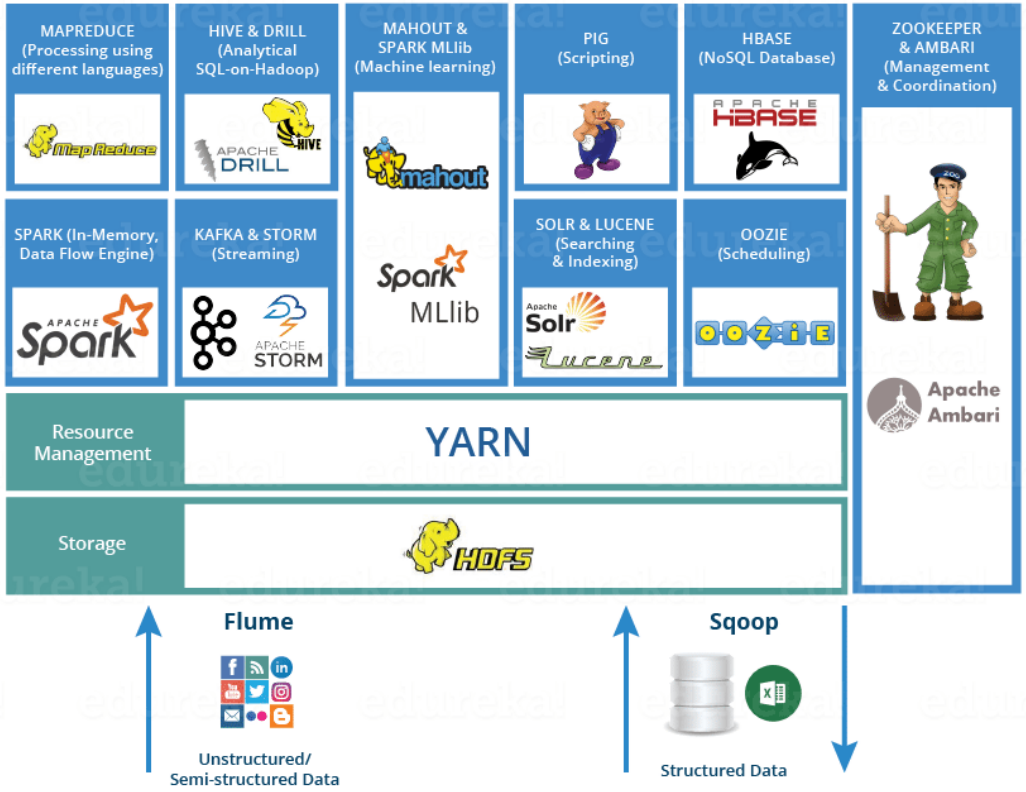
S³Lab
Smart Software System Laboratory



“Without big data, you are blind and deaf and in the middle of a freeway.”

– Geoffrey Moore

Hadoop Ecosystem



Apache Hive Tutorial

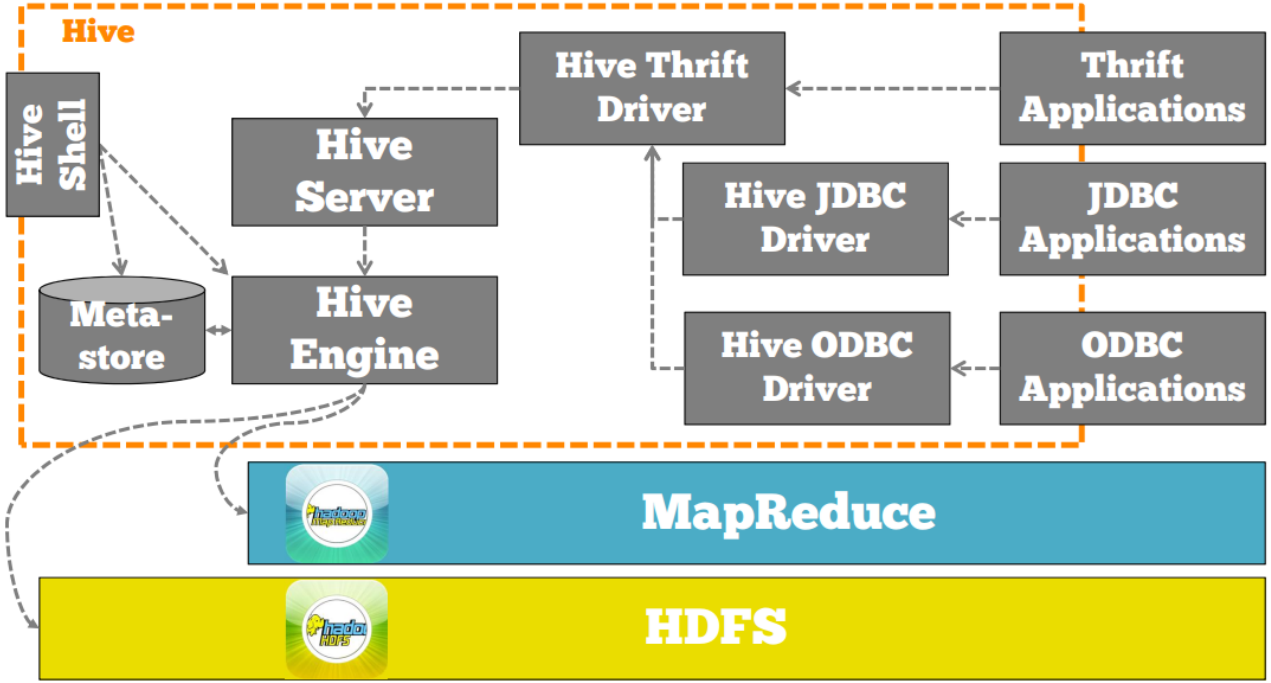
High-Level Data Process Components

Hive

- An **sql** like **interface** to Hadoop.
- Data warehouse infrastructure **built on top** of Hadoop
- Provide data **summarization**, **query** and **analysis**
- Query execution via **MapReduce**
- Hive interpreter convert the query to Map-reduce format.
- Open source project.
- Developed by Facebook
- Also used by Netflix, Cnet, Digg, eHarmony etc.

High-Level Data Process Components

***Hive** - architecture*



High-Level Data Process Components

Hive

- HiveQL example:

  SELECT customerId, max(total_cost) from hive_purchases GROUP BY customerId HAVING count(*) > 3;

Hive tutorial

Let us get started with Command Line Interface(CLI)

Command: **hive**

```
[cloudera@quickstart ~]$ hive

Logging initialized using configuration in jar:file:/usr/lib/hive/lib/hive-commo
n-1.1.0-cdh5.13.0.jar!/hive-log4j.properties
WARNING: Hive CLI is deprecated and migration to Beeline is recommended.
hive>
```

Hive tutorial

Create and show databases;

Command: **create database newdb;**

```
hive> create database newdb;
OK
Time taken: 0.398 seconds
hive>
```

Command: **show databases;**

```
hive> show databases;
OK
default
newdb
Time taken: 0.168 seconds, Fetched: 2 row(s)
hive>
```

Hive tutorial

Two types of table in Hive: managed table and external table

- For managed table, Hive is responsible for managing the data of a managed table. If you load the data from a file present in HDFS into a Hive *Managed Table* and issue a DROP command on it, the table along with its metadata will be deleted. So, the data belonging to the dropped *managed_table* no longer exist anywhere in HDFS and you can't retrieve it by any means. Basically, you are moving the data when you issue the LOAD command from the HDFS file location to the Hive warehouse directory.
- For *external table*, Hive is not responsible for managing the data. In this case, when you issue the LOAD command, Hive moves the data into its warehouse directory. Then, Hive creates the metadata information for the external table. Now, if you issue a DROP command on the *external table*, only metadata information regarding the external table will be deleted. Therefore, you can still retrive the data of that very external table from the warehouse directory using HDFS commands.

Hive tutorial

Create managed table (internal table)

```
hive> create table employee (ID int, name string, Salary float, Age int)
    > row format delimited
    > fields terminated by ','
    > ;
OK
Time taken: 0.186 seconds
```

```
hive> describe employee;
OK
id                      int
name                    string
salary                  float
age                     int
Time taken: 0.058 seconds, Fetched: 4 row(s)
```

Hive tutorial

Describe table

```
hive> describe formatted employee;
OK
# col_name              data_type               comment

id                      int
name                    string
salary                  float
age                     int

# Detailed Table Information
Database:               default
Owner:                  cloudera
CreateTime:             Mon Oct 12 05:27:24 PDT 2020
LastAccessTime:         UNKNOWN
Protect Mode:           None
Retention:              0
Location:               hdfs://quickstart.cloudera:8020/user/hive/warehouse/employee
Table Type:             MANAGED_TABLE
Table Parameters:
        transient_lastDdlTime   1602505644

# 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             ,
        serialization.format    ,
Time taken: 0.063 seconds, Fetched: 30 row(s)
```

Hive tutorial

Create external table

- Let’s try to create some external table

```
hive> create external table employee2 (ID int, name string, Salary float, Age int)
    > row format delimited
    > fields terminated by ','
    > stored as textfile;
OK
Time taken: 0.05 seconds
hive> describe employee2;
OK
id                      int
name                    string
salary                  float
age                     int
Time taken: 0.07 seconds, Fetched: 4 row(s)
```

Hive tutorial

```
hive> describe formatted employee2;
OK
# col_name              data_type               comment

id                      int
name                    string
salary                  float
age                     int

# Detailed Table Information
Database:               default
Owner:                  cloudera
CreateTime:             Mon Oct 12 05:35:01 PDT 2020
LastAccessTime:         UNKNOWN
Protect Mode:           None
Retention:              0
Location:               hdfs://quickstart.cloudera:8020/user/hive/warehouse/employee2
Table Type:             EXTERNAL_TABLE
Table Parameters:
        EXTERNAL                TRUE
        transient_lastDdlTime   1602506101

# 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             ,
        serialization.format    ,
Time taken: 0.06 seconds, Fetched: 31 row(s)
```

Hive tutorial

- Let’s check the directory **/user/hive/warehouse** in HDFS

```
[cloudera@quickstart ~]$ hdfs dfs -ls /user/hive/warehouse
Found 3 items
drwxrwxrwx   - cloudera supergroup          0 2020-10-12 05:27 /user/hive/warehouse/employee
drwxrwxrwx   - cloudera supergroup          0 2020-10-12 05:35 /user/hive/warehouse/employee2
drwxrwxrwx   - cloudera supergroup          0 2020-10-12 05:25 /user/hive/warehouse/newdb.db
[cloudera@quickstart ~]$
```

Hive tutorial

- Let’s create new external table and store data in home directory of HDFS

```
hive> create external table employee3 (ID int, name string, Salary float, Age int)
    > row format delimited
    > fields terminated by ','
    > location '/user/cloudera/emp';
OK
Time taken: 0.062 seconds
```

Hive tutorial

- Let’s rename the table and check the result

```
hive> Alter table employee3 RENAME TO emptable;
OK
Time taken: 0.126 seconds
```

```
hive> describe emptable;
OK
id                      int
name                    string
salary                  float
age                     int
Time taken: 0.068 seconds, Fetched: 4 row(s)
```

Hive tutorial

- Let’s add one more column to the table and check the result

```
hive> Alter table emptable add columns (surname string);
OK
Time taken: 0.113 seconds

hive> describe emptable;
OK
id                      int
name                    string
salary                  float
age                     int
surname                 string
Time taken: 0.064 seconds, Fetched: 5 row(s)
```

Hive tutorial

- Let’s change one column of the table and check the result

```
hive> Alter table emptable change name first_name string;
OK
Time taken: 0.084 seconds


hive> describe emptable;
OK
id                      int
first_name              string
salary                  float
age                     int
surname                 string
Time taken: 0.07 seconds, Fetched: 5 row(s)
```

Hive tutorial

LOAD Data from Local into Hive Managed Table

- Command: **LOAD DATA LOCAL INPATH '/home/cloudera/Desktop/Employee.csv' INTO TABLE employee;**

```
hive> LOAD DATA LOCAL INPATH '/home/cloudera/Desktop/Employee.csv' INTO TABLE employee;
Loading data to table default.employee
Table default.employee stats: [numFiles=1, totalSize=172]
OK
Time taken: 0.649 seconds
```

```
hive> select * from employee;
OK
NULL    name    NULL    NULL
20021   Le      1500.0  22
20022   Pham    1000.0  23
20023   Tran    2000.0  24
20024   Vu      3000.0  25
20025   Dao     2500.0  21
20026   Nam     2500.0  22
20027   Viet    3000.0  24
20028   Quoc    1000.0  22
Time taken: 0.292 seconds, Fetched: 9 row(s)
```

Why NULL?

Hive tutorial

LOAD Data from Local into Hive Managed Table

- Check the schema of the table and the .csv file

```
hive> describe employee;
OK
id                      int
name                    string
salary                  float
age                     int
Time taken: 0.082 seconds, Fetched: 4 row(s)
hive>
```

```
[cloudera@quickstart ~]$ cat /home/cloudera/Desktop/Employee.csv
ID,name,Salary,Age
20021,Le,1500,22
20022,Pham,1000,23
20023,Tran,2000,24
20024,Vu,3000,25
20025,Dao,2500,21
20026,Nam,2500,22
20027,Viet,3000,24
20028,Quoc,1000,22[cloudera@quickstart ~]$
```

Hive tutorial

LOAD Data from Local into Hive Managed Table

```
hive> drop table employee;
OK
Time taken: 0.074 seconds
hive> create table employee (ID int, name string,Salary float, Age int)
    > row format delimited
    > fields terminated by ','
    > ;
OK
Time taken: 0.051 seconds
```

*Employee.csv

```
20021,Le,1500,22
20022,Pham,1000,23
20023,Tran,2000,24
20024,Vu,3000,25
20025,Dao,2500,21
20026,Nam,2500,22
20027,Viet,3000,24
20028,Quoc,1000,22
```

Hive tutorial

LOAD Data from Local into Hive Managed Table

```
hive> LOAD DATA LOCAL INPATH '/home/cloudera/Desktop/Employee.csv' INTO TABLE employee;
Loading data to table default.employee
Table default.employee stats: [numFiles=1, totalSize=153]
OK
Time taken: 0.154 seconds
hive> select * from employee;
OK
20021   Le      1500.0  22
20022   Pham    1000.0  23
20023   Tran    2000.0  24
20024   Vu      3000.0  25
20025   Dao     2500.0  21
20026   Nam     2500.0  22
20027   Viet    3000.0  24
20028   Quoc    1000.0  22
Time taken: 0.038 seconds, Fetched: 8 row(s)
```

Hive tutorial

LOAD Data from Local into Hive External Table

- Command: **LOAD DATA LOCAL INPATH '/home/cloudera/Desktop/Employee.csv' INTO TABLE employee2;**

```
hive> LOAD DATA LOCAL INPATH '/home/cloudera/Desktop/Employee.csv' INTO TABLE employee2;
Loading data to table default.employee2
Table default.employee2 stats: [numFiles=1, totalSize=153]
OK
Time taken: 0.155 seconds

hive> select * from employee2;
OK
20021   Le      1500.0  22
20022   Pham    1000.0  23
20023   Tran    2000.0  24
20024   Vu      3000.0  25
20025   Dao     2500.0  21
20026   Nam     2500.0  22
20027   Viet    3000.0  24
20028   Quoc    1000.0  22
Time taken: 0.041 seconds, Fetched: 8 row(s)
```

Hive tutorial

Difference between managed and external table

```
[cloudera@quickstart ~]$ hdfs dfs -ls /user/hive/warehouse
Found 3 items
drwxrwxrwx   - cloudera supergroup          0 2020-10-12 05:27 /user/hive/warehouse/employee
drwxrwxrwx   - cloudera supergroup          0 2020-10-12 05:35 /user/hive/warehouse/employee2
drwxrwxrwx   - cloudera supergroup          0 2020-10-12 05:25 /user/hive/warehouse/newdb.db
[cloudera@quickstart ~]$
```

```
[cloudera@quickstart ~]$ hdfs dfs -ls /user/hive/warehouse/employee
Found 1 items
-rwxrwxrwx   1 cloudera supergroup        153 2020-10-12 07:26 /user/hive/warehouse/employee/Employee.csv
[cloudera@quickstart ~]$ hdfs dfs -ls /user/hive/warehouse/employee2
Found 1 items
-rwxrwxrwx   1 cloudera supergroup        153 2020-10-12 07:31 /user/hive/warehouse/employee2/Employee.csv
```

Hive tutorial

Difference between managed and external table

- Let’s drop the external table

```
hive> drop table employee2;
OK
Time taken: 0.054 seconds
hive> select * from employee2;
FAILED: SemanticException [Error 10001]: Line 1:14 Table not found 'employee2'
```

- Then check the directory associated with this external table in HDFS

```
[cloudera@quickstart ~]$ hdfs dfs -ls /user/hive/warehouse/employee2
Found 1 items
-rwxrwxrwx   1 cloudera supergroup        153 2020-10-12 07:31 /user/hive/warehouse/employee2/Employee.csv
```

Hive tutorial

Difference between managed and external table

- Let's drop the internal table

```
hive> drop table employee;
OK
Time taken: 0.091 seconds
hive> select * from employee;
FAILED: SemanticException [Error 10001]: Line 1:14 Table not found 'employee'
```

- Then check the directory associated with this internal table in HDFS

```
[cloudera@quickstart ~]$ hdfs dfs -ls /user/hive/warehouse/employee
ls: `/user/hive/warehouse/employee': No such file or directory
```

```
[cloudera@quickstart ~]$ hdfs dfs -ls /user/hive/warehouse
Found 3 items
-rw-r--r--   1 cloudera supergroup       1381 2020-10-12 06:05 /user/hive/warehouse/StudentReport.csv
drwxrwxrwx   - cloudera supergroup          0 2020-10-12 07:31 /user/hive/warehouse/employee2
drwxrwxrwx   - cloudera supergroup          0 2020-10-12 05:25 /user/hive/warehouse/newdb.db
```

Hive tutorial

LOAD Data from HDFS to Hive Table

- Let's create some internal table

```
hive> create table student (ID int, Name string,Course string, Age int)
    > row format delimited fields terminated by ',' tblproperties('skip.header.line.count'='1');
OK
Time taken: 0.07 seconds
```

declare this Hive's property to skip the header in Student.csv file

```
hive> describe student;
OK
id                      int
name                    string
course                  string
age                     int
Time taken: 0.04 seconds, Fetched: 4 row(s)
```

Hive tutorial

LOAD Data from HDFS to Hive Table

- Put file Student.csv to HDFS

```
[cloudera@quickstart ~]$ cd /home/cloudera/Desktop
[cloudera@quickstart Desktop]$ ls
Department.csv   Employee2.csv  Employee.csv~       Express.desktop   Parcels.desktop  StudentReport.csv
Eclipse.desktop  Employee.csv   Enterprise.desktop  Kerberos.desktop  Student.csv
[cloudera@quickstart Desktop]$ hdfs dfs -put Student.csv
[cloudera@quickstart Desktop]$ hdfs dfs -ls
Found 2 items
-rw-r--r--   1 cloudera cloudera        249 2020-10-12 08:01 Student.csv
drwxr-xr-x   - cloudera cloudera          0 2020-10-12 05:46 emp
```

Hive tutorial

LOAD Data from HDFS to Hive Table

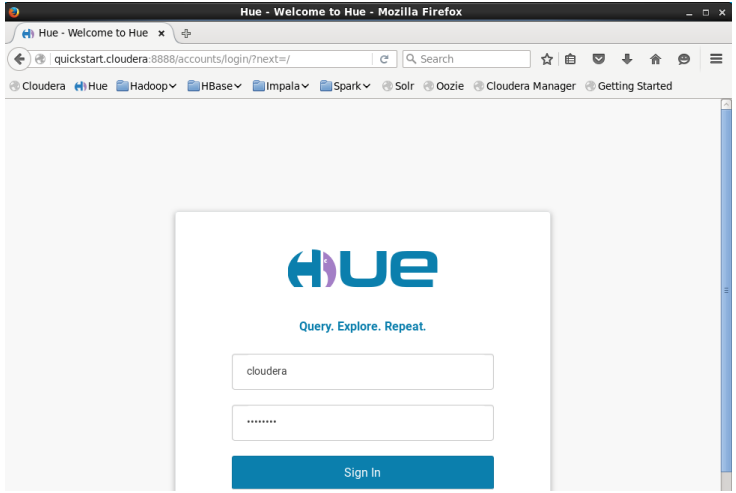
```
hive> load data inpath 'Student.csv' into table student;
Loading data to table default.student
Table default.student stats: [numFiles=1, totalSize=249]
OK
Time taken: 0.217 seconds
```

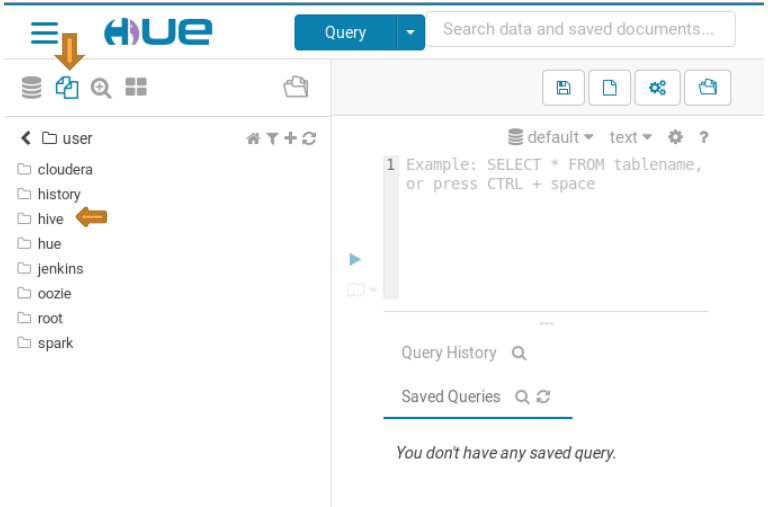
```
hive> select * from student;
OK
123451  Quynh   Hadoop  22
123452  Tai     Java    22
123453  Truong  Python  23
123454  Nghia   Hadoop  24
123455  Thuy    Java    23
123456  Hao     Python  24
123457  Hien    Hadoop  22
123458  Phuong  Java    23
123459  Hai     Python  23
123460  Phuong  Hadoop  24
Time taken: 0.042 seconds, Fetched: 10 row(s)
```

Hive tutorial

Hive command using HUE

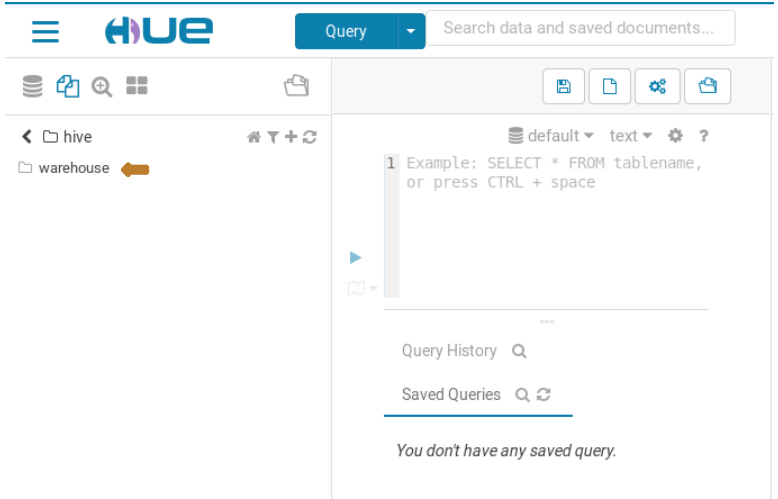
- Login to HUE

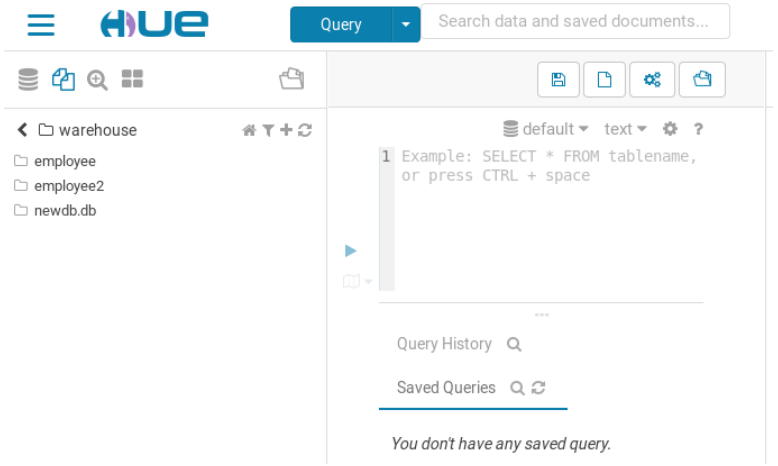




Hive tutorial

Hive command using HUE

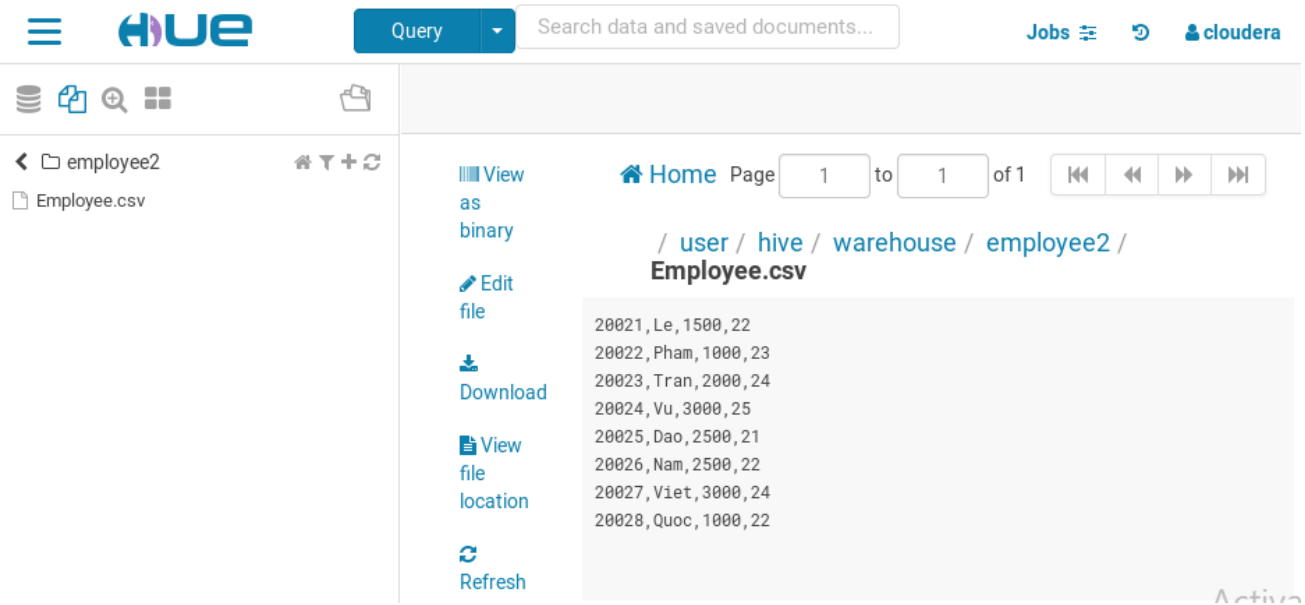




Hive tutorial

Hive command using HUE

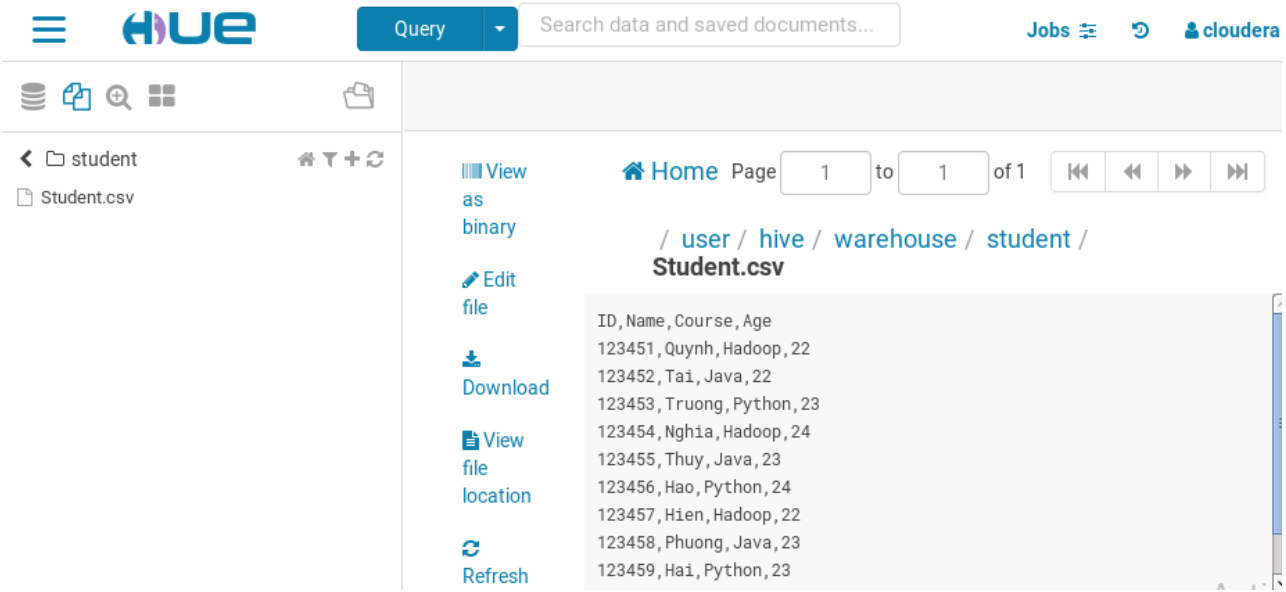
- Let’s check the file Employee.csv



Hive tutorial

Hive command using HUE

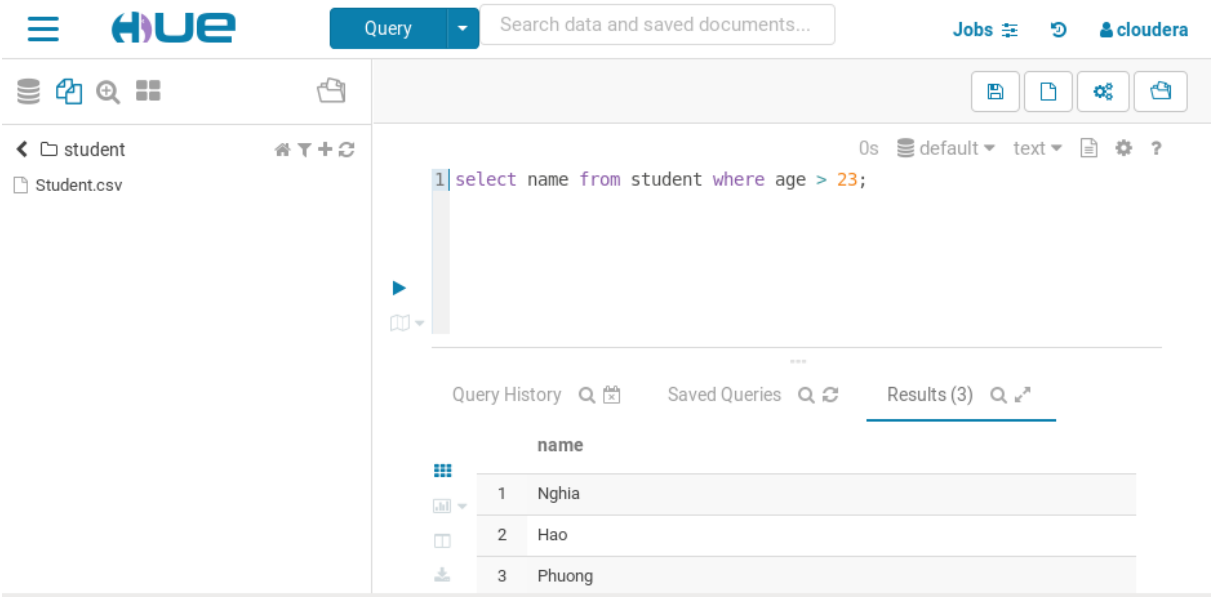
- Let’s check the file Student.csv



Hive tutorial

Hive command using HUE

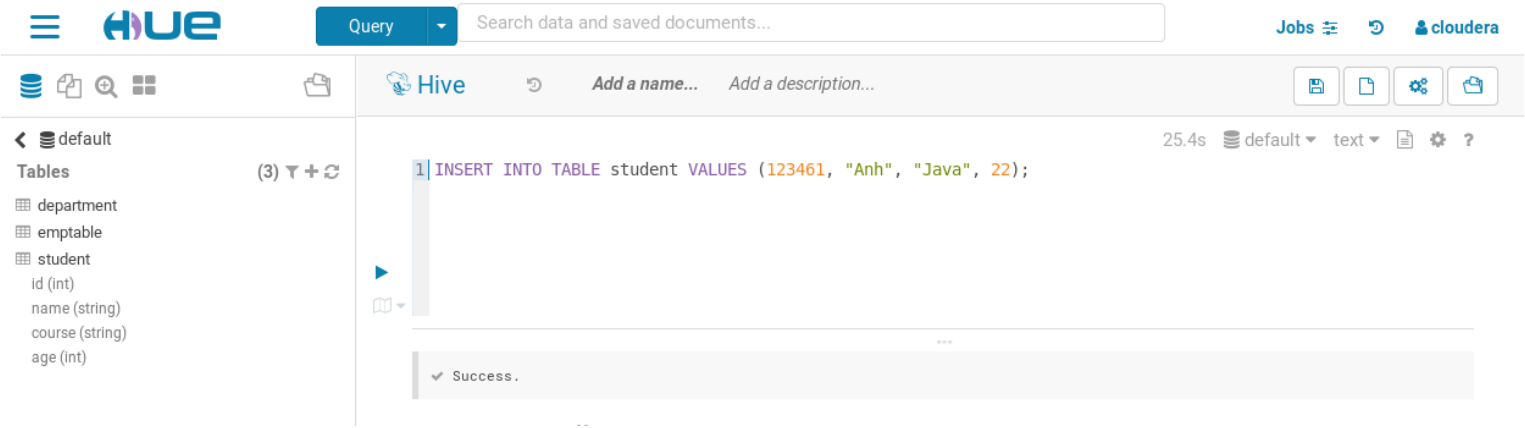
- Let’s make some Hive query



Hive tutorial

Hive command using HUE

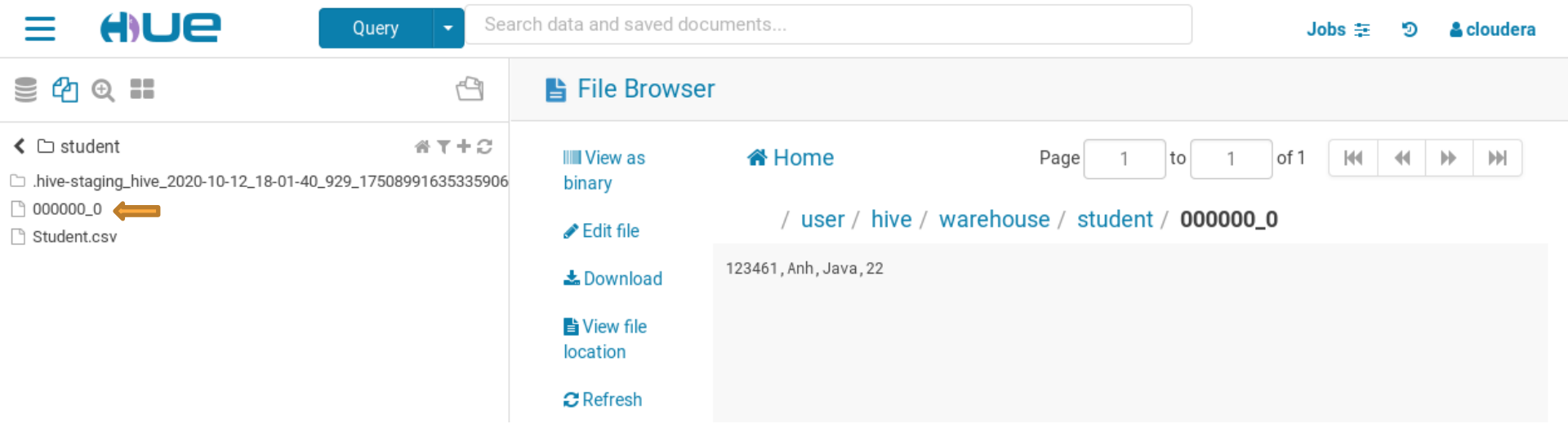
- Let’s make some Hive query



Hive tutorial

Hive command using HUE

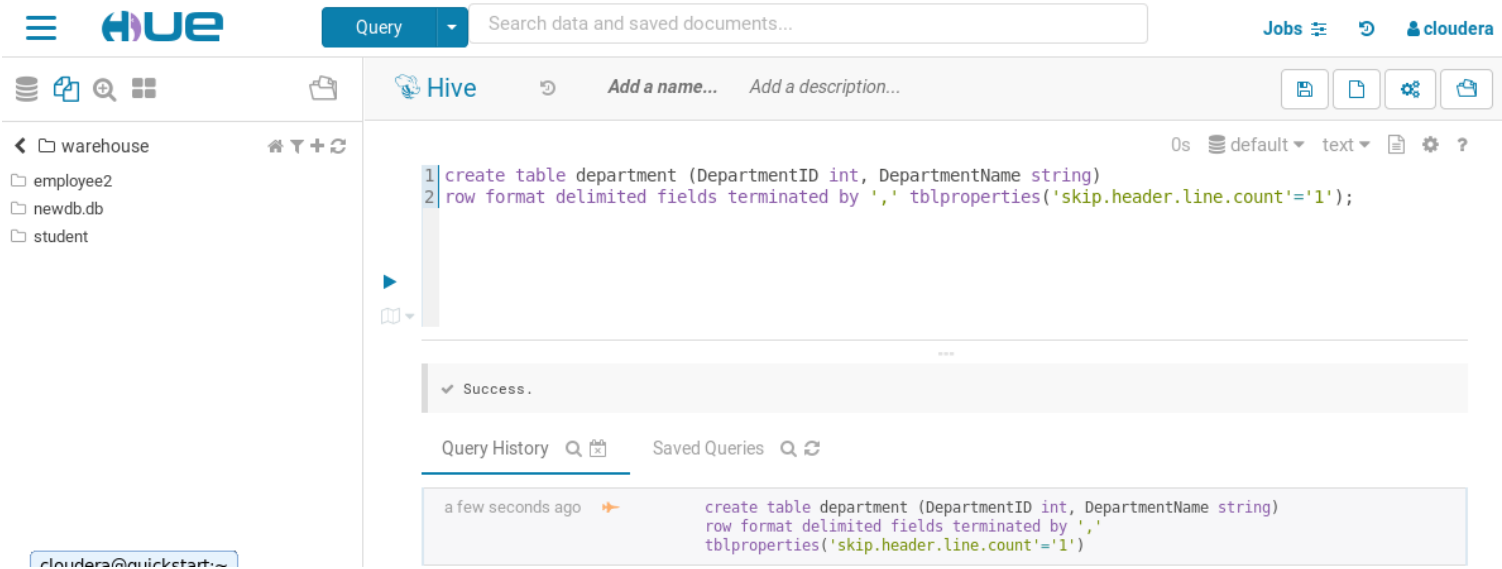
- Check newly created file in HDFS



Hive tutorial

Hive command using HUE

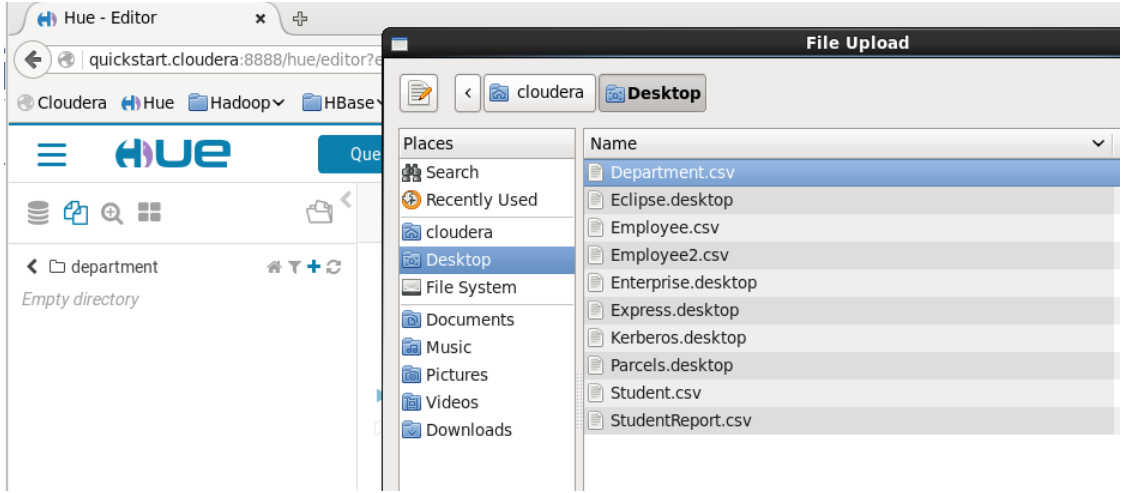
- Let’s create some table in HUE
- Command: **create table department (DepartmentID int, DepartmentName string) row format delimited fields terminated by ',' tblproperties('skip.header.line.count'='1');**



Hive tutorial

Hive command using HUE

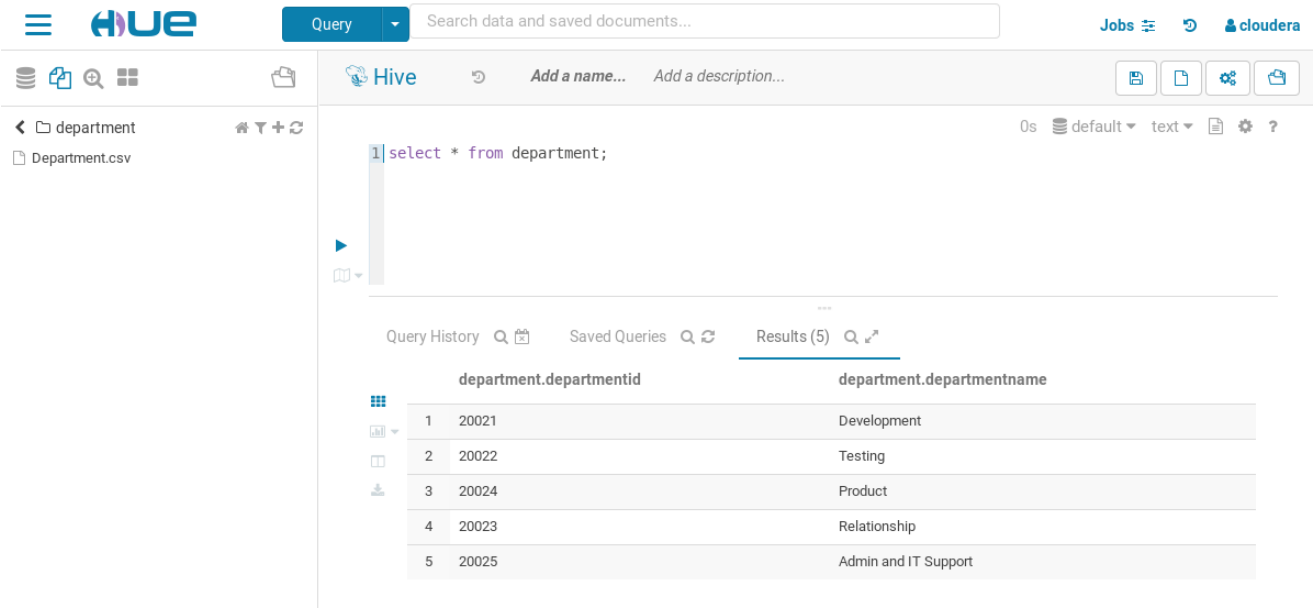
- Upload file Department.csv in to department directory (use + button)



Hive tutorial

Hive command using HUE

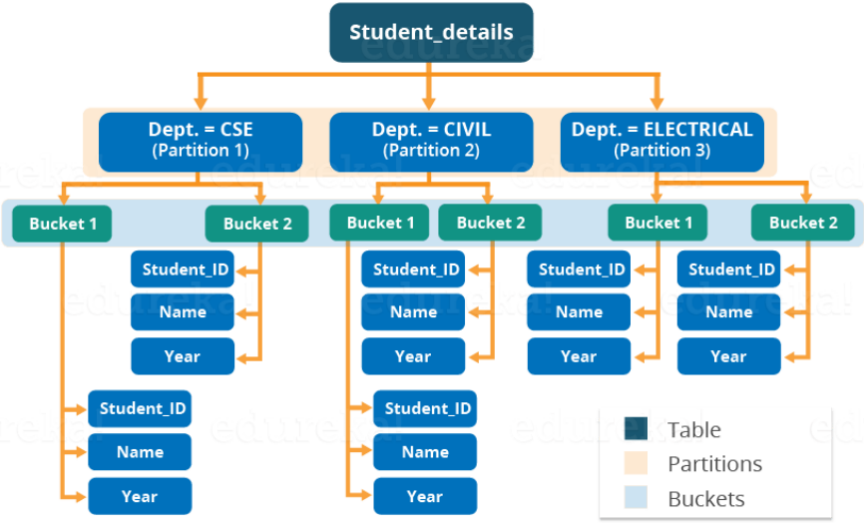
- Query this newly created table



Hive tutorial

Partition in Hive

- Hive organizes tables into partitions for grouping similar type of data together based on a column or partition key. Each Table can have one or more partition keys to identify a particular partition. This allows us to have a faster query on slices of the data.



Hive tutorial

Static partition

- Create new database

```
hive> create database studentdb;
OK
Time taken: 0.054 seconds
hive> use studentdb;
OK
Time taken: 0.019 seconds
```

Student2.csv

```
ID,Name,Age,Course
123451,Quynh,22,Hadoop
123452,Tai,22,Java
123453,Truong,23,Python
123454,Nghia,24,Hadoop
123455,Thuy,23,Java
123456,Hao,24,Python
123457,Hien,22,Hadoop
123458,Phuong,23,Java
123459,Hai,23,Python
123460,Phuong,24,Hadoop
```

- Create new table with partition

```
hive> create table student (ID int, Name string, Age int) partitioned by (Course string)
    > row format delimited fields terminated by ',' tblproperties('skip.header.line.count'='1');
OK
Time taken: 0.047 seconds
```

Hive tutorial

Static partition

- Check the format of the table

```
hive> describe student;
OK
id                      int
name                    string
age                     int
course                  string

# Partition Information
# col_name              data_type               comment

course                  string
Time taken: 0.055 seconds, Fetched: 9 row(s)
```

Hive tutorial

Static partition

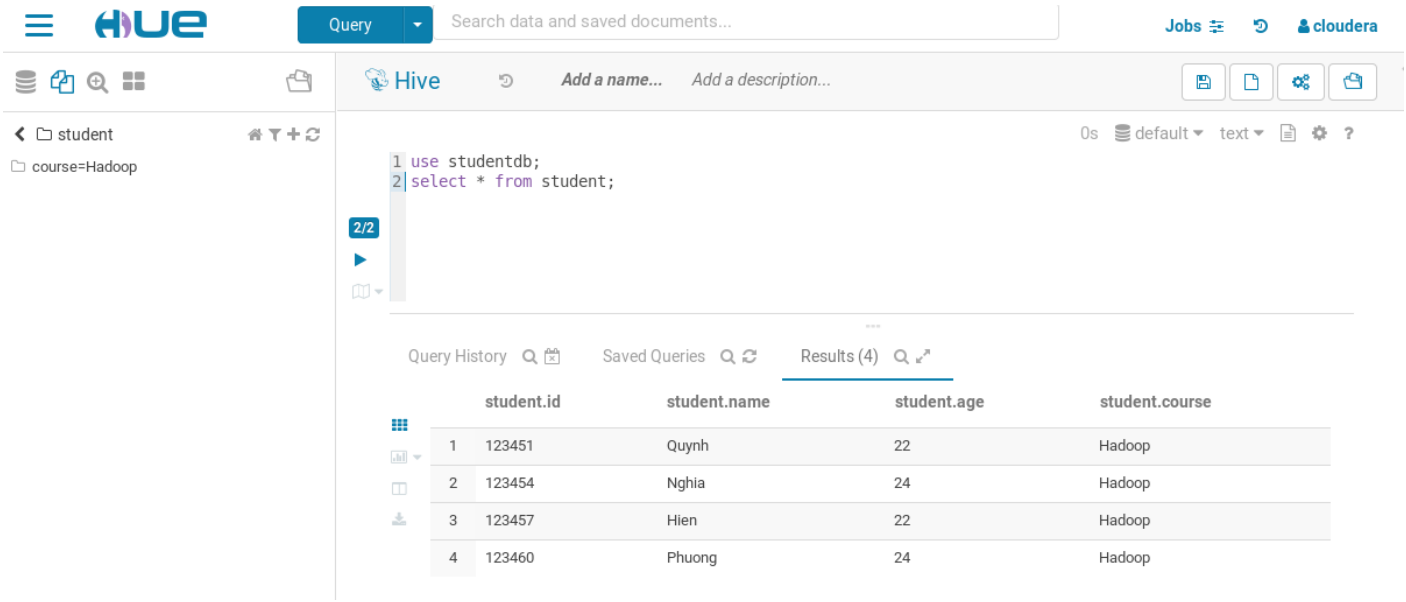
- Load data from file **StudentHadoop.csv** to partition (course=Hadoop)

```
hive> load data local inpath '/home/cloudera/Desktop/StudentHadoop.csv' into table student partition(course= "Hadoop");
Loading data to table studentdb.student partition (course=Hadoop)
Partition studentdb.student{course=Hadoop} stats: [numFiles=1, numRows=0, totalSize=116, rawDataSize=0]
OK
Time taken: 0.163 seconds
```

Hive tutorial

Static partition

- Check new directory **course=Hadoop** in HDFS



Hive tutorial

Static partition

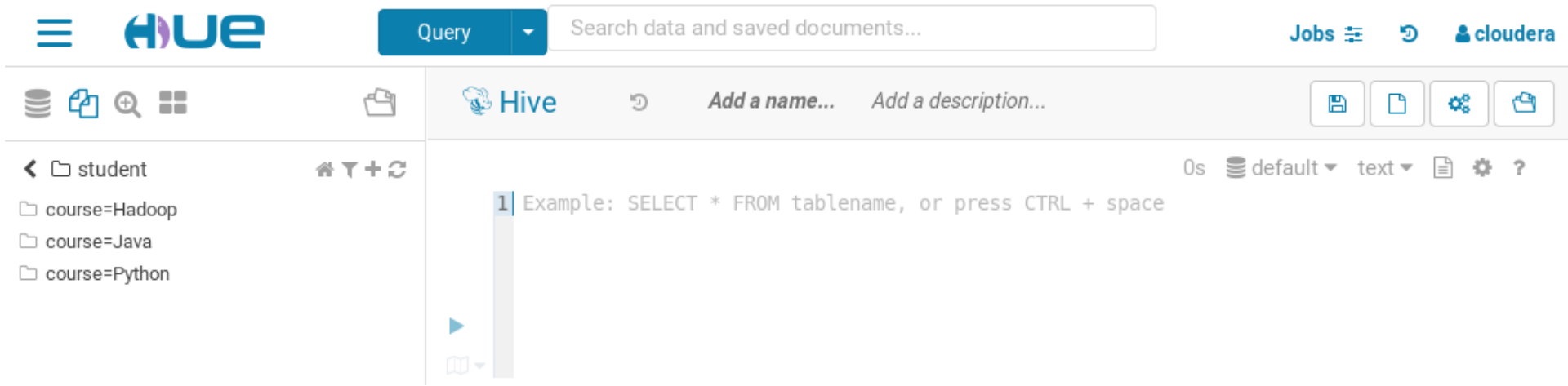
- Continue to load data to other partition

```
hive> load data local inpath '/home/cloudera/Desktop/StudentJava.csv' into table student partition(course= "Java");
Loading data to table studentdb.student partition (course=Java)
Partition studentdb.student{course=Java} stats: [numFiles=1, numRows=0, totalSize=82, rawDataSize=0]
OK
Time taken: 0.227 seconds
hive> load data local inpath '/home/cloudera/Desktop/StudentPython.csv' into table student partition(course= "Python");
Loading data to table studentdb.student partition (course=Python)
Partition studentdb.student{course=Python} stats: [numFiles=1, numRows=0, totalSize=89, rawDataSize=0]
OK
Time taken: 0.27 seconds
```

Hive tutorial

Static partition

- Check all created directories in HDFS



Hive tutorial

Dynamic partition

- Create new database

```
hive> create database newstudentdb;
OK
Time taken: 0.027 seconds
hive> use newstudentdb;
OK
Time taken: 0.009 seconds
hive> set hive.exec.dynamic.partition=true;
hive> set hive.exec.dynamic.partition.mode=nonstrict;
```

Hive tutorial

Dynamic partition

- Create table student (same like before)

```
hive> create table student (ID int, Name string, Course string, Age int) row format delimited fields terminated by ','
tblproperties('skip.header.line.count'='1');
OK
Time taken: 0.054 seconds
hive> describe student;
OK
id                      int
name                    string
course                  string
age                     int
Time taken: 0.034 seconds, Fetched: 4 row(s)
```

Hive tutorial

Dynamic partition

- Load data to table student (same like before)

```
hive> load data local inpath '/home/cloudera/Desktop/Student.csv' into table student;
Loading data to table newstudentdb.student
Table newstudentdb.student stats: [numFiles=1, totalSize=249]
OK
Time taken: 0.127 seconds
```

Hive tutorial

Dynamic partition

- Create table student_partition

```
hive> create table student_partition (ID int, Name string, Age int) partitioned by (Course string) row format delimited fields terminated by ',';
OK
Time taken: 0.036 seconds
hive> describe student_partition;
OK
id                      int
name                    string
age                     int
course                  string

# Partition Information
# col_name              data_type               comment

course                  string
Time taken: 0.041 seconds, Fetched: 9 row(s)
```

Hive tutorial

Dynamic partition

- Command: **insert into student_partition partition(Course) select ID, Name, Age, Course from student;**

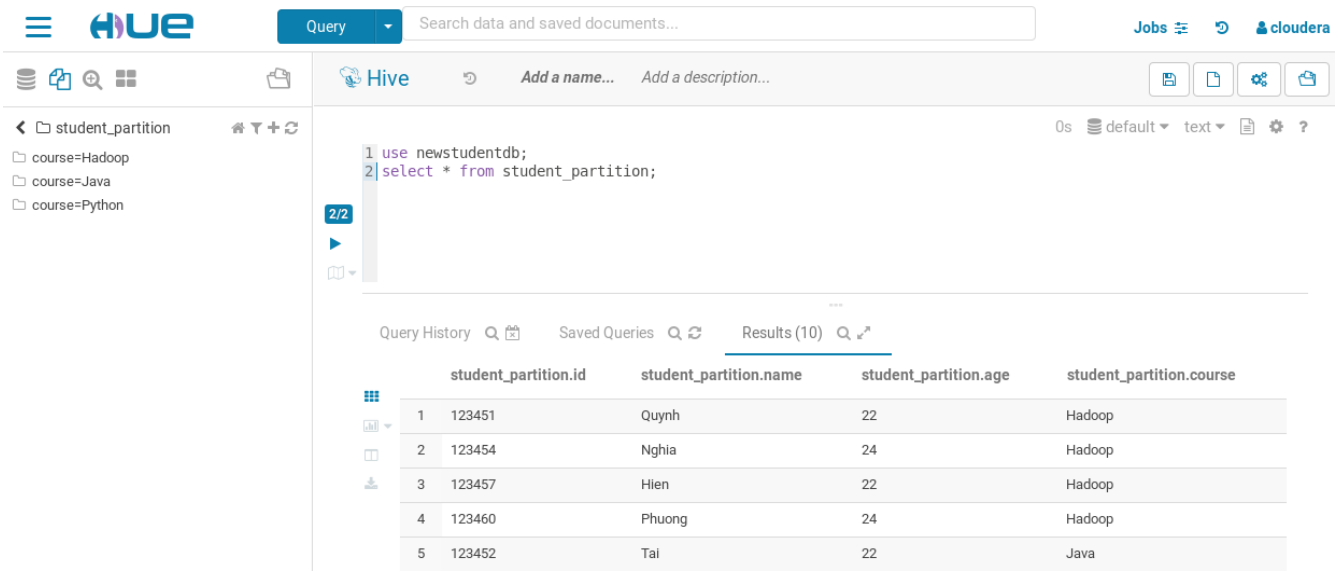
```
hive> insert into student_partition partition(Course) select ID, Name, Age, Course from student;
Query ID = cloudera_20201012112020_58eb9cc8-6a9a-4e6b-ba6f-a96fae6e366b
Total jobs = 3
Launching Job 1 out of 3
Number of reduce tasks is set to 0 since there's no reduce operator
Starting Job = job_1602505280766_0003, Tracking URL = http://quickstart.cloudera:8088/proxy/application_1602505280766_0003/
Kill Command = /usr/lib/hadoop/bin/hadoop job  -kill job_1602505280766_0003
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 0
2020-10-12 11:20:33,403 Stage-1 map = 0%,  reduce = 0%
2020-10-12 11:20:41,979 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 1.05 sec
MapReduce Total cumulative CPU time: 1 seconds 50 msec
Ended Job = job_1602505280766_0003
```

```
Stage-Stage-1: Map: 1   Cumulative CPU: 1.35 sec   HDFS Read: 4800 HDFS Write: 369 SUCCESS
Total MapReduce CPU Time Spent: 1 seconds 350 msec
OK
Time taken: 24.234 seconds
```

Hive tutorial

Dynamic partition

- Check created directories in HDFS



Apache Pig Tutorial

High-Level Data Process Components

Pig

- A **scripting platform** for processing and analyzing large data sets
- Apache Pig allows to write complex **MapReduce programs** using a simple **scripting** language.
- Made of two components:
  - High level language: **Pig Latin** (data flow language).
  - Pig translate Pig Latin script into MapReduce to execute within Hadoop.
- Open source project
- Developed by Yahoo

High-Level Data Process Components

Pig

- Pig Latin example:

```
A = LOAD 'student' USING PigStorage() AS (name:chararray,
age:int, gpa:float);

X = FOREACH A GENERATE name,$2;

DUMP X;
```

Pig tutorial

Pig data model

1.Basic Pig Data Types :

Data Types	Description	How can we use use Pig Data Types?
int	It is a Signed 32-bit integer value	Int can hold any integer value Ex: 23
long	It is a Signed 64-bit integer value	long can hold any Long value i.e bigger than Integer value and it can be displayed as 23L
float	It is a 32-bit floating point value	Float can be represented as 4.5F or 4.5.5f or 4.5e2f or 4.5E2F
double	It is a 64-bit floating point value	We can use this if the value is bigger than float and it can be represented as 08.5 or 08.5e2 or 08.5E2

Pig tutorial

Pig data model

1.Basic Pig Data Types :

Data Types	Description	How can we use use Pig Data Types?
chararray	It is a Character array (string) in Unicode UTF-8 format value	JavaChain.com
bytearray	The default datatype in pig is Byte array	
boolean	boolean represents true or false values.	It could be either true/false and it is case sensitive.
datetime	It displays the datetime	2016-01-14T00:00:00.000+00:00
biginteger	It displays the Biginteger	70409060802
bigdecimal	It displays the bigdecimal	198.78946646131311211

Pig tutorial

Pig data model

2.Complex Data Types

tuple	it is the collections of one or more fields	(Connecticut,Newjersey, Newyork)
bag	it is the collection of one or more tuples	{(Jack, Jill, JavaChain.com), (Connecticut, Newjersey, Newyork)}
map	It has Key and value pair data	[websitename#javachain.com]

Pig tutorial

Tuple and Bag

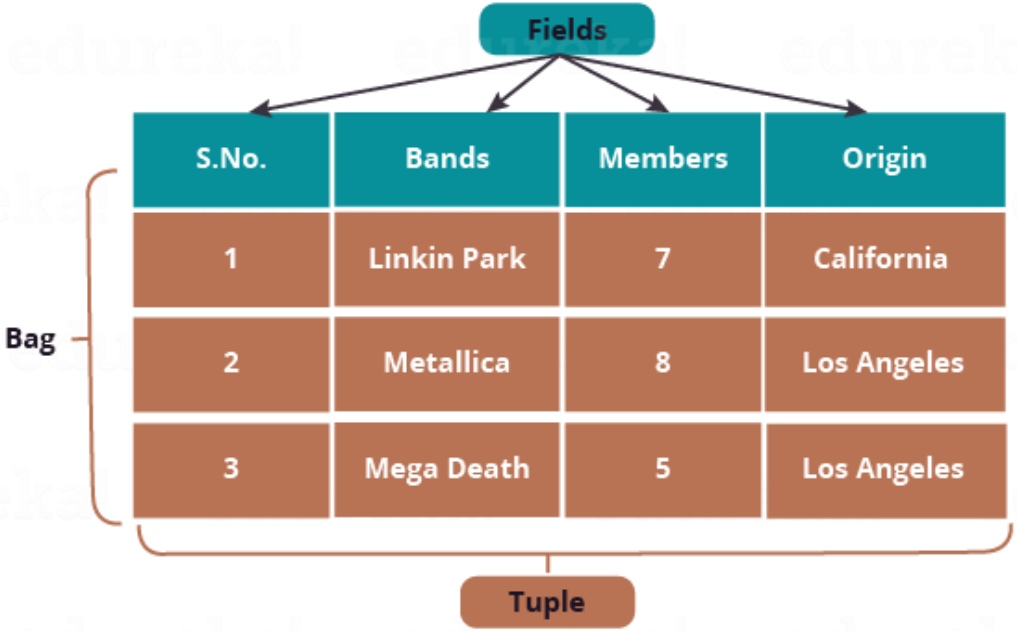


Figure: Apache Pig Data Model

Pig tutorial

Tuple

- Tuple is an ordered set of fields which may contain different data types for each field. You can understand it as the records stored in a row in a relational database. A Tuple is a set of cells from a single row as shown in the above image. The elements inside a tuple does not necessarily need to have a schema attached to it.
- A tuple is represented by '()' symbol.
- Example of tuple – (1, Linkin Park, 7, California)
- Since tuples are ordered, we can access fields in each tuple using indexes of the fields, like $1 form above tuple will return a value 'Linkin Park'. You can notice that above tuple doesn't have any schema attached to it.

Pig tutorial

Bag

- A bag is a collection of a set of tuples and these tuples are subset of rows or entire rows of a table. A bag can contain duplicate tuples, and it is not mandatory that they need to be unique.
- The bag has a flexible schema i.e. tuples within the bag can have different number of fields. A bag can also have tuples with different data types.
- A bag is represented by '{}' symbol.
- Example of a bag – {(Linkin Park, 7, California), (Metallica, 8), (Mega Death, Los Angeles)}

Pig tutorial

Bag

- For Apache Pig to effectively process bags, the fields and their respective data types need to be in the same sequence.
- Set of bags –
- {(Linkin Park, 7, California), (Metallica, 8), (Mega Death, Los Angeles)},
- {(Metallica, 8, Los Angeles), (Mega Death, 8), (Linkin Park, California)}

Pig tutorial

Two types of Bag: Outer Bag and Inner Bag.

- **Outer bag** or **relation** is noting but a bag of tuples. Here relations are similar as relations in relational databases. To understand it better let us take an example:

- {(Linkin Park, California), (Metallica, Los Angeles), (Mega Death, Los Angeles)}

- This above bag explains the relation between the Band and their place of Origin.

Pig tutorial

Pig data model

- **Inner bag** contains a bag inside a tuple. For Example, if we sort Band tuples based on Band's Origin, we will get:

- (Los Angeles, {(Metallica, Los Angeles), (Mega Death, Los Angeles)})
- (California,{(Linkin Park, California)})

- Here, first field type is a string while the second field type is a bag, which is an inner bag within a tuple.

Pig tutorial

Map

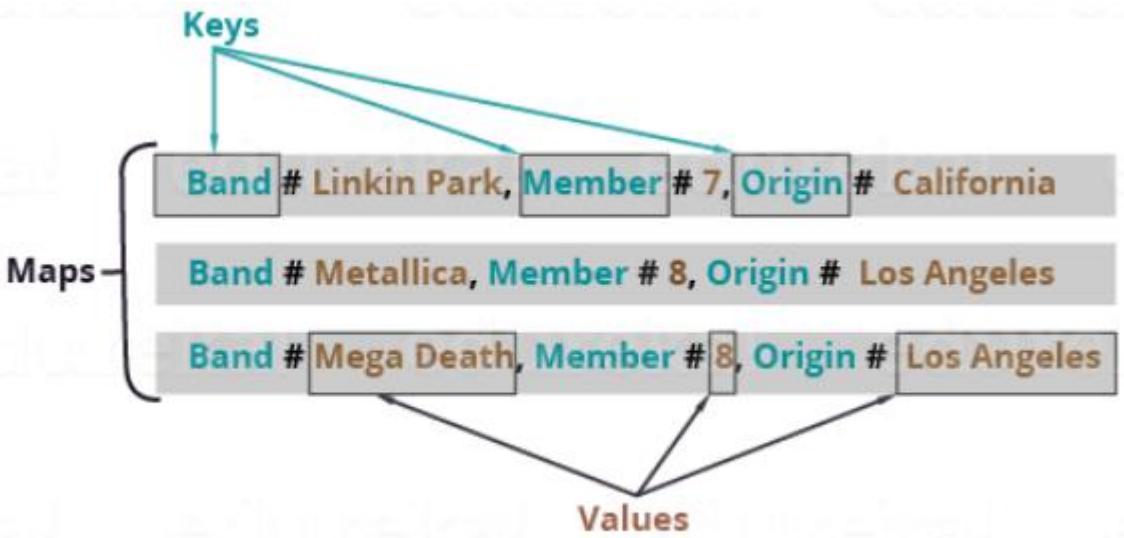


Figure: Map Example

Pig tutorial

Map

- A map is key-value pairs used to represent data elements. The key must be a chararray [] and should be unique like column name, so it can be indexed and value associated with it can be accessed on basis of the keys. The value can be of any data type.
- Maps are represented by '[]' symbol and key-value are separated by '#' symbol, as you can see in the above image.
- Example of maps– [band#Linkin Park, members#7], [band#Metallica, members#8]

Pig tutorial

Schema

- Schema assigns name to the field and declares data type of the field. Schema is optional in Pig Latin but Pig encourage you to use them whenever possible, as the error checking becomes efficient while parsing the script which results in efficient execution of program. Schema can be declared as both simple and complex data types. During LOAD function, if the schema is declared it is also attached with the data.

Pig tutorial

Schema

- Few Points on Schema in Pig:
- If the schema only includes the field name, the data type of field is considered as byte array.
- If you assign a name to the field you can access the field by both, the field name and the positional notation. Whereas if field name is missing we can only access it by the positional notation i.e. $ followed by the index number.
- If you perform any operation which is a combination of relations (like JOIN, COGROUP, etc.) and if any of the relation is missing schema, the resulting relation will have null schema.
- If the schema is null, Pig will consider it as byte array and the real data type of field will be determined dynamically.

Pig tutorial

Open Pig grunt shell

- Command: pig

```
[cloudera@quickstart ~]$ pig
```

Pig tutorial

Open Pig grunt shell

- Invoke the ls command of Linux shell from the Grunt shell
- Command: sh ls

```
grunt> sh ls
cloudera-manager
cm_api.py
Desktop
Documents
Downloads
eclipse
enterprise-deployment.json
express-deployment.json
kerberos
lib
Music
parcels
Pictures
Public
Templates
Videos
workspace
```

Pig tutorial

Load data into Apache Pig from the file system (HDFS/ Local)

Syntax

The load statement consists of two parts divided by the “=” operator. On the left-hand side, we need to mention the name of the relation **where** we want to store the data, and on the right-hand side, we have to define **how** we store the data. Given below is the syntax of the **Load** operator.

```
Relation_name = LOAD 'Input file path' USING function as schema;
```

Where,

- **relation_name** − We have to mention the relation in which we want to store the data.
- **Input file path** − We have to mention the HDFS directory where the file is stored. (In MapReduce mode)
- **function** − We have to choose a function from the set of load functions provided by Apache Pig (**BinStorage, JsonLoader, PigStorage, TextLoader**).
- **Schema** − We have to define the schema of the data. We can define the required schema as follows −

```
(column1 : data type, column2 : data type, column3 : data type);
```

Note − We load the data without specifying the schema. In that case, the columns will be addressed as $01, $02, etc… (check).

Pig tutorial

Load data from HDFS into Pig relation

- Open gedit and create a txt file and save it in home directory

101 Anto 20000 Architect

102 Bob 7000 SoftwareEngineer

103 Jack 4000 Programmer

104 Bil 3000 ITConsultant

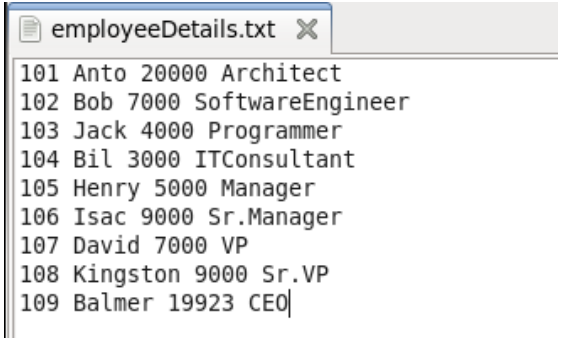
105 Henry 5000 Manager

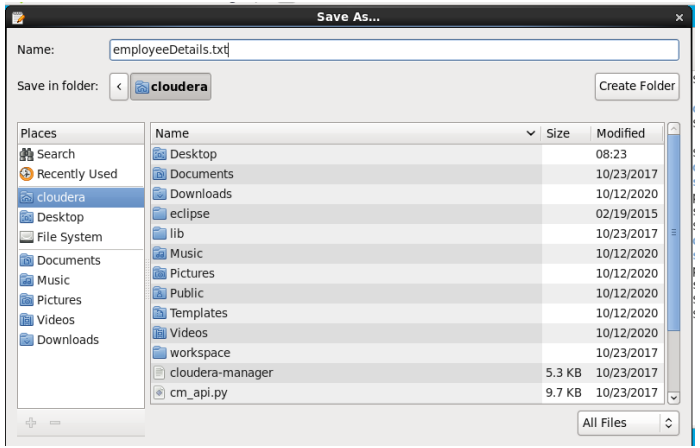
106 Isac 9000 Sr.Manager

107 David 7000 VP

108 Kingston 9000 Sr.VP

109 Balmer 19923 CEO





Pig tutorial

Load data from HDFS into Pig relation

- Put the file to HDFS

```
[cloudera@quickstart ~]$ hdfs dfs -put employeeDetails.txt
```

Pig tutorial

Load data from HDFS into Pig relation

- Let’s load data into Pig Relation using Pig Data Types.
- Command: employee = load 'employeeDetails.txt' using PigStorage(‘ ‘) as (id:int, name:chararray,salary:float,task:chararray);

```
grunt> employee = load 'employeeDetails.txt' using PigStorage(' ') as (id:int, name:chararray,salary:float,task:
chararray);
2020-10-19 09:38:21,715 [main] INFO  hive.metastore - Trying to connect to metastore with URI thrift://127.0.0.1
:9083
```

Pig tutorial

Load data from HDFS into Pig relation

- Let's DESCRIBE the relation to see the Data type names.

```
grunt> describe employee;
2020-10-19 09:48:30,077 [main] INFO  hive.metastore - Trying to connect to metastore with URI thrift://127.0.0.1
:9083
2020-10-19 09:48:30,078 [main] INFO  hive.metastore - Opened a connection to metastore, current connections: 1
2020-10-19 09:48:30,078 [main] INFO  hive.metastore - Connected to metastore.
2020-10-19 09:48:30,098 [main] INFO  org.apache.hadoop.conf.Configuration.deprecation - fs.default.name is depre
cated. Instead, use fs.defaultFS
2020-10-19 09:48:30,098 [main] INFO  org.apache.hadoop.conf.Configuration.deprecation - mapred.job.tracker is de
precated. Instead, use mapreduce.jobtracker.address
employee: {id: int,name: chararray,salary: float,task: chararray}
```

Pig tutorial

Load data from HDFS into Pig relation

- Let's use dump operator to display the result

```
grunt> dump employee;
2020-10-19 09:38:31,176 [main] INFO  org.apache.hadoop.conf.Configuration.deprecation - fs.default.name is depre
cated. Instead, use fs.defaultFS
```

```
2020-10-19 09:38:49,888 [main] INFO  org.apache.pig.backend.hadoop.executionengine.util.MapRedUtil - Total input
 paths to process : 1
(101,Anto,20000.0,Architect)
(102,Bob,7000.0,SoftwareEngineer)
(103,Jack,4000.0,Programmer)
(104,Bil,3000.0,ITConsultant)
(105,Henry,5000.0,Manager)
(106,Isac,9000.0,Sr.Manager)
(107,David,7000.0,VP)
(108,Kingston,9000.0,Sr.VP)
(109,Balmer,19923.0,CEO)
```

Pig tutorial

Load data from Local File System into Pig relation

- Open pig shell in local mode by pig -x local

```
grunt> quit;
[cloudera@quickstart ~]$ pig -x local
```

- Load file

```
grunt> employee2 = load ' /home/cloudera/employeeDetails.txt' using PigStorage(' ') as (id:int, name:chararray,s
alary:float,task:chararray);
grunt>
```

Pig tutorial

Load data from Local File System into Pig relation

- Check the result

```
grunt> dump employee2;
2020-10-19 11:34:30,221 [main] INFO  org.apache.pig.tools.pigstats.ScriptState - Pig features used in the script
: UNKNOWN
```

```
2020-10-19 11:34:42,816 [main] INFO  org.apache.pig.backend.hadoop.executionengine.util.MapRedUtil - Total input
 paths to process : 1
(101,Anto,20000.0,Architect)
(102,Bob,7000.0,SoftwareEngineer)
(103,Jack,4000.0,Programmer)
(104,Bil,3000.0,ITConsultant)
(105,Henry,5000.0,Manager)
(106,Isac,9000.0,Sr.Manager)
(107,David,7000.0,VP)
(108,Kingston,9000.0,Sr.VP)
(109,Balmer,19923.0,CEO)
```

Pig tutorial

LOAD Data from HIVE Table into PIG Relation.

- Let us consider that we have the Hive table called student with some data in it

```
hive> show tables;
OK
department
emptable
newstuden
student
Time taken: 0.707 seconds, Fetched: 4 row(s)
hive> select * from student;
OK
123451  Quynh   Hadoop  22
123452  Tai     Java    22
123453  Truong  Python  23
123454  Nghia   Hadoop  24
123455  Thuy    Java    23
123456  Hao     Python  24
123457  Hien    Hadoop  22
123458  Phuong  Java    23
123459  Hai     Python  23
123460  Phuong  Hadoop  24
Time taken: 0.706 seconds, Fetched: 10 row(s)
```

Pig tutorial

LOAD Data from HIVE Table into PIG Relation.

- The command below will load the data from HIVE Table into PIG Relation called pigdataStudent

- Command: **pigdataStudent = load 'student' using org.apache.hive.hcatalog.pig.HCatLoader();**

```
grunt> pigdataStudent = load 'student' using org.apache.hive.hcatalog.pig.HCatLoader();
2020-10-19 09:51:23,159 [main] INFO  org.apache.hadoop.conf.Configuration.deprecation - fs.default.name is depre
cated. Instead, use fs.defaultFS
2020-10-19 09:51:23,159 [main] INFO  org.apache.hadoop.conf.Configuration.deprecation - mapred.job.tracker is de
precated. Instead, use mapreduce.jobtracker.address
2020-10-19 09:51:23,194 [main] INFO  hive.metastore - Closed a connection to metastore, current connections: 0
2020-10-19 09:51:23,194 [main] INFO  hive.metastore - Trying to connect to metastore with URI thrift://127.0.0.1
:9083
```

Pig tutorial

LOAD Data from HIVE Table into PIG Relation.

- Check the content of the relation
- Command: **dump pigdataStudent;**

```
grunt> dump pigdataStudent;
2020-10-19 09:52:49,835 [main] INFO  org.apache.hadoop.conf.Configuration.deprecation - fs.default.name is depre
cated. Instead, use fs.defaultFS
```

```
(123451,Quynh,Hadoop,22)
(123452,Tai,Java,22)
(123453,Truong,Python,23)
(123454,Nghia,Hadoop,24)
(123455,Thuy,Java,23)
(123456,Hao,Python,24)
(123457,Hien,Hadoop,22)
(123458,Phuong,Java,23)
(123459,Hai,Python,23)
(123460,Phuong,Hadoop,24)
(123466,Min,Hadoop,25)
(123464,Den,Hadoop,25)
(123461,Anh,Java,22)
(123462,An,Java,24)
```

Pig tutorial

Filter operation

- Now the a1 relation has all the data, Let us try to filter only the values where age > 23.
- Command: **plus23 = filter pigdataStudent by age > 23;**

```
grunt> plus23 = filter pigdataStudent by age > 23;
2020-10-19 09:57:58,418 [main] INFO  hive.metastore - Trying to connect to metastore with URI thrift://127.0.0.1
:9083
2020-10-19 09:57:58,419 [main] INFO  hive.metastore - Opened a connection to metastore, current connections: 1
2020-10-19 09:57:58,419 [main] INFO  hive.metastore - Connected to metastore.
2020-10-19 09:57:58,437 [main] INFO  org.apache.hadoop.conf.Configuration.deprecation - fs.default.name is depre
cated. Instead, use fs.defaultFS
```

Pig tutorial

Filter operation

- **Let's DESCRIBE the relation to see the Data type names**

```
grunt> describe plus23;
2020-10-19 09:58:51,153 [main] INFO  org.apache.hadoop.conf.Configuration.deprecation - fs.default.name is depre
cated. Instead, use fs.defaultFS
2020-10-19 09:58:51,154 [main] INFO  org.apache.hadoop.conf.Configuration.deprecation - mapred.job.tracker is de
precated. Instead, use mapreduce.jobtracker.address
2020-10-19 09:58:51,213 [main] INFO  org.apache.hadoop.conf.Configuration.deprecation - fs.default.name is depre
cated. Instead, use fs.defaultFS
2020-10-19 09:58:51,213 [main] INFO  org.apache.hadoop.conf.Configuration.deprecation - mapred.job.tracker is de
precated. Instead, use mapreduce.jobtracker.address
2020-10-19 09:58:51,300 [main] INFO  org.apache.hadoop.conf.Configuration.deprecation - fs.default.name is depre
cated. Instead, use fs.defaultFS
2020-10-19 09:58:51,300 [main] INFO  org.apache.hadoop.conf.Configuration.deprecation - mapred.job.tracker is de
precated. Instead, use mapreduce.jobtracker.address
plus23: {id: int,name: chararray,course: chararray,age: int}
```

Pig tutorial

Filter operation

- Check the result

```
grunt> dump plus23;
2020-10-19 10:00:48,159 [main] INFO  org.apache.hadoop.conf.Configuration.deprecation - fs.default.name is depre
cated. Instead, use fs.defaultFS
2020-10-19 10:00:48,159 [main] INFO  org.apache.hadoop.conf.Configuration.deprecation - mapred.job.tracker is de

2020-10-19 10:01:12,260 [main] INFO  org.apache.pig.backend.hadoop.executionengine.util.MapRedUtil - Total input
 paths to process : 1
(123454,Nghia,Hadoop,24)
(123456,Hao,Python,24)
(123460,Phuong,Hadoop,24)
(123466,Min,Hadoop,25)
(123464,Den,Hadoop,25)
(123462,An,Java,24)
grunt>
```

Pig tutorial

Storing Data from PIG Relation

Syntax

Given below is the syntax of the Store statement.

```
STORE Relation_name INTO ' required_directory_path ' [USING function];
```

Pig tutorial

Store PIG Relation into HDFS

```
grunt> store plus23 into '/user/cloudera/plus23' using PigStorage(',');
2020-10-19 11:53:51,902 [main] INFO  org.apache.hadoop.conf.Configuration.deprecation - fs.default.name is depre
cated. Instead, use fs.defaultFS
```

```
Counters:
Total records written : 6
Total bytes written : 128
Spillable Memory Manager spill count : 0
Total bags proactively spilled: 0
Total records proactively spilled: 0

Job DAG:
job_1603115446431_0017


2020-10-19 11:54:12,429 [main] INFO  org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.MapReduceLaunc
her - Success!
```

Pig tutorial

Store PIG Relation into HDFS

- **Check if the plus23 directory has been created in HDFS**

```
grunt> ls
hdfs://quickstart.cloudera:8020/user/cloudera/emp       <dir>
hdfs://quickstart.cloudera:8020/user/cloudera/employeeDetails.txt<r 1>  217
hdfs://quickstart.cloudera:8020/user/cloudera/plus23    <dir>
grunt> cd plus23
grunt> ls
hdfs://quickstart.cloudera:8020/user/cloudera/plus23/_SUCCESS<r 1>      0
hdfs://quickstart.cloudera:8020/user/cloudera/plus23/part-m-00000<r 1>  128
```

Pig tutorial

Store PIG Relation into HDFS

- **Check the content of the file**

```
grunt> cat part-m-00000
123454,Nghia,Hadoop,24
123456,Hao,Python,24
123460,Phuong,Hadoop,24
123466,Min,Hadoop,25
123464,Den,Hadoop,25
123462,An,Java,24
```

Pig tutorial

STORE Data from PIG Relation Into HIVE Table

- Create a new Hive table

```
hive> create table plus23student (id int, name string, course string, age int);
OK
Time taken: 1.023 seconds
hive> describe plus23student;
OK
id                      int
name                    string
course                  string
age                     int
Time taken: 0.104 seconds, Fetched: 4 row(s)
```

Pig tutorial

STORE Data from PIG Relation Into HIVE Table

- Store data from Pig relation into the newly created Hive table

```
grunt> store plus23  into 'plus23student' using org.apache.hive.hcatalog.pig.HCatStorer();
2020-10-19 10:04:59,906 [main] INFO  hive.metastore - Trying to connect to metastore with URI thrift://127.0.0.1
:9083


Successfully stored 6 records (128 bytes) in: "plus23student"

Counters:
Total records written : 6
Total bytes written : 128
Spillable Memory Manager spill count : 0
Total bags proactively spilled: 0
Total records proactively spilled: 0

Job DAG:
job_1603115446431_0012


2020-10-19 10:13:46,233 [main] INFO  org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.MapReduceLaunc
her - Success!
```

Pig tutorial

STORE Data from PIG Relation Into HIVE Table

- Check the Hive table

```
hive> select * from plus23student;
OK
123454  Nghia   Hadoop  24
123456  Hao     Python  24
123460  Phuong  Hadoop  24
123466  Min     Hadoop  25
123464  Den     Hadoop  25
123462  An      Java    24
Time taken: 0.325 seconds, Fetched: 6 row(s)
```

Pig tutorial

Create Your First Apache Pig Script

- Create and open an Apache Pig script file in an editor (e.g. gedit)

```
[cloudera@quickstart ~]$ sudo gedit /home/cloudera/test.pig
```

*test.pig

```
employ = load 'employeeDetails.txt' using PigStorage(' ') as (id:int,
name:chararray,salary:float,task:chararray);

employSalary = FOREACH employ generate id, salary;

dump employSalary;
```

Pig tutorial

Create Your First Apache Pig Script

- Run the script in linux terminal

```
[cloudera@quickstart ~]$ pig test.pig
log4j:WARN No appenders could be found for logger (org.apache.hadoop.util.Shell)
.
log4j:WARN Please initialize the log4j system properly.
log4j:WARN See http://logging.apache.org/log4j/1.2/faq.html#noconfig for more in
fo
```

```
2020-10-19 10:53:21,121 [main] INFO  org.apache.pig.backend.hadoop.executionengi
ne.util.MapRedUtil - Total input paths to process : 1
(101,20000.0)
(102,7000.0)
(103,4000.0)
(104,3000.0)
(105,5000.0)
(106,9000.0)
(107,7000.0)
(108,9000.0)
(109,19923.0)
2020-10-19 10:53:21,248 [main] INFO  org.apache.hadoop.conf.Configuration.deprec
ation - fs.default.name is deprecated. Instead, use fs.defaultFS
```

Pig tutorial

Create Your First Apache Pig Script

- Create file **test2.pig**

test2.pig

```
employ = load 'employeeDetails.txt' using PigStorage(' ') as (id:int,
name:chararray,salary:float,task:chararray);

employName = FOREACH employ generate id, name;
```

Pig tutorial

Create Your First Apache Pig Script

- Run the script in grunt shell

```
grunt> run test2.pig
2020-10-19 11:05:32,010 [main] INFO  org.apache.hadoop.conf.Configuration.deprecation - fs.default.name is depre
cated. Instead, use fs.defaultFS
```

Pig tutorial

Create Your First Apache Pig Script

- Check the result

```
grunt> dump employName;
2020-10-19 11:06:32,619 [main] INFO  org.apache.hadoop.conf.Configuration.deprecation - fs.default.name is depre
cated. Instead, use fs.defaultFS
2020-10-19 11:06:32,619 [main] INFO  org.apache.hadoop.conf.Configuration.deprecation - mapred.job.tracker is de
precated. Instead, use mapreduce.jobtracker.address
```

```
2020-10-19 11:06:51,266 [main] INFO  org.apache.pig.backend.hadoop.executionengine.util.MapRedUtil - Total input
 paths to process : 1
(101,Anto)
(102,Bob)
(103,Jack)
(104,Bil)
(105,Henry)
(106,Isac)
(107,David)
(108,Kingston)
(109,Balmer)
```

Pig tutorial

Positional notation reference

- So far, we fields in Pig Relation are referred by name (e.g. id, name, salary, task, etc.,)
- Names are assigned by you using schemas
- Positional notation is generated by the system. Positional notation is indicated with the dollar sign ($) and begins with zero (0); for example, $0, $1, $2.

	First Field	Second Field	Third Field
Data type	chararray	int	float
Positional notation (generated by system)	$0	$1	$2
Possible name (assigned by you using a schema)	name	age	gpa
Field value (for the first tuple)	John	18	4.0

Pig tutorial

Positional notation reference

- In this example, the field **task** is referenced by position notation **$3**

```
grunt> employee = load 'employeeDetails.txt' using PigStorage(' ') as (id:int, n
ame:chararray, salary:float, task:chararray);
```

```
grunt> describe employee;
employee: {id: int,name: chararray,salary: float,task: chararray}
```

```
grunt> empTask = foreach employee generate id,$3;
grunt> describe empTask;
empTask: {id: int,task: chararray}
```

```
grunt> dump empTask;
2020-10-19 23:20:36,052 [main] INFO  org.apache.pig.tools.pigstats.ScriptState -
```

Pig tutorial

Positional notation reference

- Check the result

```
grunt> dump empTask;
2020-10-19 23:20:36,052 [main] INFO  org.apache.pig.tools.pigstats.ScriptState -
```

```
2020-10-19 23:21:02,992 [main] INFO  org.apache.pig.backend.hadoop.executionengi
ne.util.MapRedUtil - Total input paths to process : 1
(101,Architect)
(102,SoftwareEngineer)
(103,Programmer)
(104,ITConsultant)
(105,Manager)
(106,Sr.Manager)
(107,VP)
(108,Sr.VP)
(109,CEO)
grunt>
```

Pig tutorial

Schema Handling

- You can define a schema that includes both the field name and field type.
- You can define a schema that includes the field name only; in this case, the field type defaults to bytearray.
- You can choose not to define a schema; in this case, the field is un-named and the field type defaults to bytearray.

Pig tutorial

Schema Handling

- The field data types are not specified (defaults type is bytearray)

```
grunt> employee = load 'employeeDetails.txt' using PigStorage(' ') as (id, name, salary, task);
grunt> describe employee;
employee: {id: bytearray,name: bytearray,salary: bytearray,task: bytearray}
```

Pig tutorial

Schema Handling

- Unknow schema

```
grunt> employee = load 'employeeDetails.txt' using PigStorage(' ');
grunt> describe employee;
Schema for employee unknown.
```

```
grunt> bonusSalary = foreach employee generate $0,$2+500;
2020-10-20 00:04:31,169 [main] WARN  org.apache.pig.PigServer - Encountered Warning IMPLICIT_CAST_TO_INT 3 time(s).
```

```
grunt> describe bonusSalary;
2020-10-20 00:04:47,411 [main] WARN  org.apache.pig.PigServer - Encountered Warning IMPLICIT_CAST_TO_INT 1 time(s).
bonusSalary: {bytearray,int}
```

Pig tutorial

Schema Handling

- Check the result

```
grunt> dump bonusSalary;
2020-10-20 00:03:47,623 [main] WARN  org.apache.pig.PigServer - Encountered Warning IMPLICIT_CAST_TO_INT 1 time(s).
2020-10-20 00:03:47,624 [main] INFO  org.apache.pig.tools.pigstats.ScriptState - Pig features used in the script: UNKNOWN
```

```
2020-10-20 00:04:06,072 [main] INFO  org.apache.pig.backend.hadoop.executionengine.util.MapRedUtil - Total input paths to process
: 1
(101,20500)
(102,7500)
(103,4500)
(104,3500)
(105,5500)
(106,9500)
(107,7500)
(108,9500)
(109,20423)
```

Pig tutorial

Schema Handling

- Declare the schema of the result

```
grunt> bonusSalary = foreach employee generate $0,$2+500 as salary:float;
2020-10-20 00:03:30,144 [main] WARN  org.apache.pig.PigServer - Encountered Warning IMPLICIT_CAST_TO_INT 2 time(s).
grunt> describe bonusSalary;
2020-10-20 00:04:17,802 [main] WARN  org.apache.pig.PigServer - Encountered Warning IMPLICIT_CAST_TO_INT 1 time(s).
bonusSalary: {bytearray,salary: float}
```

```
grunt> bonusSalary = foreach employee generate $0 as id:int, $2+500 as salary:float;
2020-10-20 00:12:55,336 [main] WARN  org.apache.pig.PigServer - Encountered Warning IMPLICIT_CAST_TO_INT 4 time(s).
grunt> describe bonusSalary;
2020-10-20 00:13:07,130 [main] WARN  org.apache.pig.PigServer - Encountered Warning IMPLICIT_CAST_TO_INT 1 time(s).
bonusSalary: {id: int,salary: float}
```

High-Level Data Process Components

Hive & Pig

- Both requires compiler to generate Mapreduce jobs
- Hence high latency queries when used for real time responses to ad-hoc queries
- Both are good for **batch processing** and **ETL** jobs
- Fault tolerant

High-Level Data Process Components

Impala

- Cloudera Impala is a query engine that runs on Apache Hadoop.
- Similar to HiveQL.
- Does not use Map-reduce
- Optimized for low latency queries
- Open source apache project
- Developed by Cloudera
- Much faster than Hive or pig

Hive

1. Create an internal table name **InternalEmployee** with the schema

ID	int
Name	string
Salary	float
Department	string
Age	int

a. Put file **EmployeeInfo.csv** to **HDFS**, then load data from this file to table InternalEmp

b. Login to HUE and query EmpName and EmpAge of all employees with salary-≥ 2000

c. In HUE, insert a row with the content [20029,Quang,1000,Support,20] to the table and then show the content of the newly created file which stores the data of the new row

2. Create a dynamic partition table named **EmployeePartition** to store the data from **EmployeeInfo.csv** file. This table is partitioned by **Department**.

a. Load data to EmployeePartition table

b. Open HUE and query to show all the rows of the table

Sqoop and Pig

1. Create table **Employee** inside database **EmployeeInfo** and load the content from **EmployeeNoheader.csv** to this table. Then query the created database to show the info of employees with **salary >1500**

id	name	salary	department	age
20023	Tran	2000	Product	24
20024	Vu	3000	Product	25
20025	Dao	2500	Testing	21
20026	Nam	2500	Support	22
20027	Viet	3000	Product	24

2. Use Sqoop import to get id, name, and salary of employees in Employee table **with salary > 2000** and save results in **Employee2000plus** directory. Then show the imported results.

3. Load data from file **EmployeeNoheader.csv** to a relation named **PigEmployee** and show the result

4. Create a relation named **PigEmployee1500plus** to store employees with **salary > 1500**

5. Store data in **PigEmployee1500plus** to a Hive table name **HiveEmployee1500plus** and show the result

6. Load data in **EmployeeNoheader.csv** to a relation named **PigEmployeeNoschema** without declaring datatype. Then from this relation, create another relation named **BonusSalary** with all data is same as that in **PigEmployeeNoschema** but **salary = salary + 500**