talend | Connecting the Data-Driven Enterprise

Participant Guide

Big Data Basics

Version 6.3 edition 2

Talend Inc.
800 Bridge Parkway, Suite 200
Redwood City, CA 94065
United States
+1 (650) 539 3200

Welcome to Talend Training

Congratulations on choosing a Talend training course.

Working through the course

You will develop your skills by working through use cases and practice exercises using live software. Completing the exercises is critical to learning!

If you are following a self-paced, on-demand training (ODT) module, and you need an answer to proceed with a particular exercise, use the help suggestions on your image desktop. If you can't access your image, contact customercare@talend.com.

Exploring

You will be working in actual Talend software, not a simulation. We hope you have fun and get lots of practice using the software! However, if you work on tasks beyond the scope of the training, you could run out of time with the environment, or you could mess up data or Jobs needed for subsequent exercises. We suggest finishing the course first, and if you have remaining time, explore as you wish. Keep in mind that our technical support team can't assist with your exploring beyond the course materials.

For more information

Talend product documentation (help.talend.com)

Talend Community (community.talend.com)

Sharing

Intentionally blank

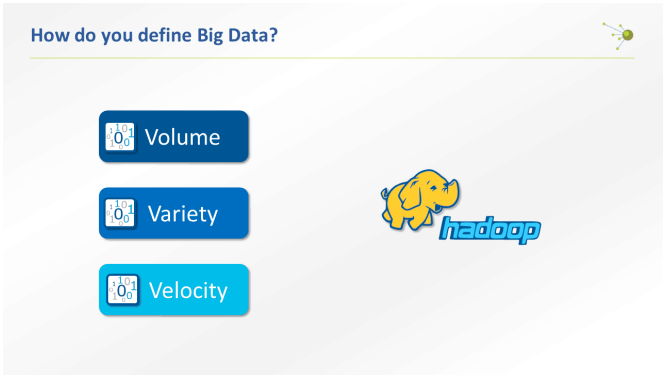
CONTENTS

LESSON 1

Big Data in Context

This chapter discusses:

Concepts



What is Big Data? How can we define it?

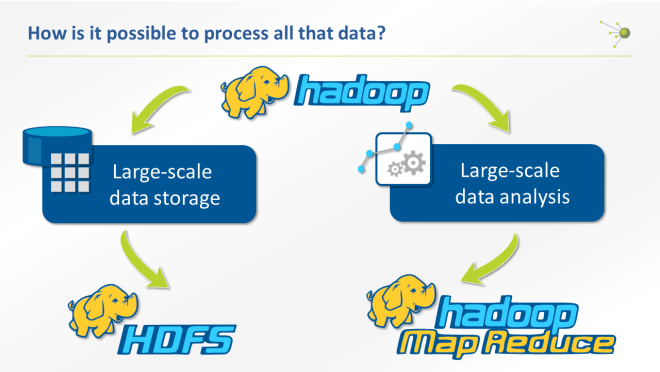
One simple definition is provided by the three Vs.

The first V corresponds to volume. Nowadays, the amount of data that's stored and processed is huge. This is a primary characteristic of Big Data: How big it is. The second V stands for variety. You can collect different types of data from different sources. This presents a challenge: How do you deal with these different types of data? The last V stands for velocity. Data arrives from different sources at different speeds. For example, social media, such as Twitter or Facebook, generates data at increasing speeds. Big Data comes with many challenges that Hadoop tries to solve.

Today we're generating more data than ever: from financial transactions, sensor networks, social media, and server logs, just to name a few. We're generating data faster than ever because of automation and user-generated content. This data has many valuable applications for marketing analysis, product recommendations, and even fraud detection. And these are only a few examples. To extract this valuable information, the data must be processed.

Fortunately, while storage capacity has increased, the cost of storage has decreased. Disk performance has also improved in recent years so from a business perspective, it's more valuable to store and process the data than throw it away. Unfortunately, transfer rates have not improved as fast as storage capacity. Even if we can process data more quickly, accessing it is slow.

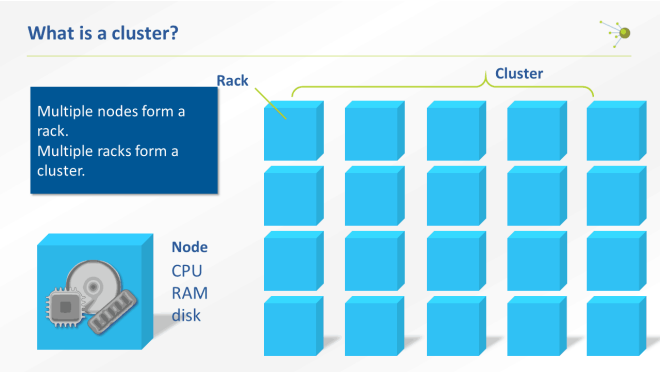
Technically speaking, there are two main problems to solve: Large-scale data storage, and large-scale data analysis.



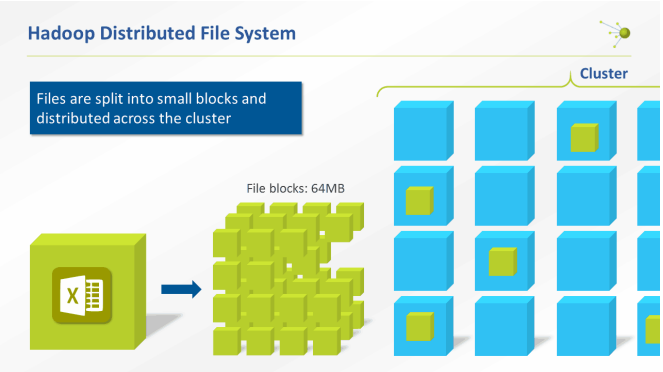
Hadoop offers a solution to these problems by helping you store and process your data. HDFS is the Hadoop Distributed File System, where you can store your data.

MapReduce will help you analyze your data.

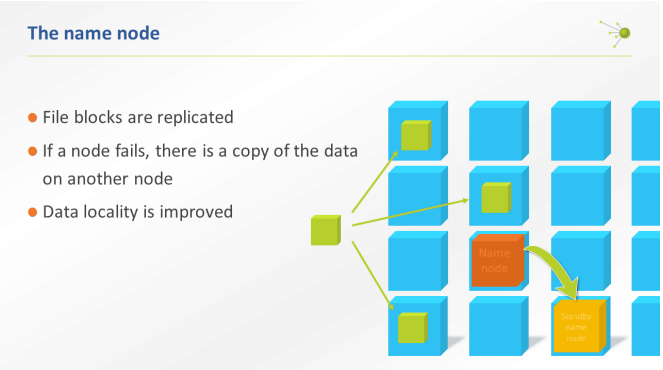
Hadoop runs on a cluster.



A cluster is made of different elements. The most basic element is a node. A node is simple, cheap dedicated hardware, composed of CPU, RAM, and disk. Multiple nodes form a rack. And multiple racks form a cluster. This cluster is used to store and process your data.

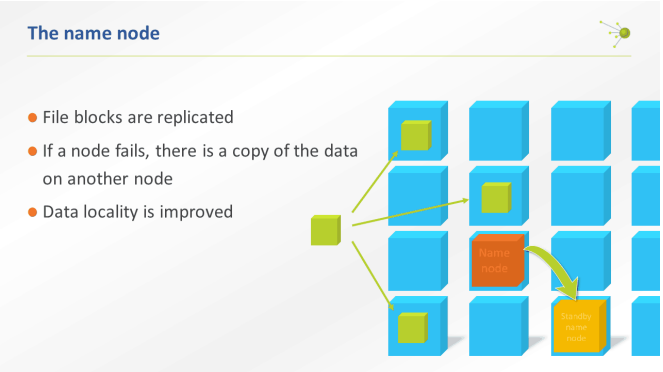


HDFS is the Hadoop distributed file system. Imagine you have a large file to store on HDFS. First it'll be split into small pieces of 64 or 128 megabytes. This size is configurable. Then each piece will be distributed across the cluster. This enables faster processing because multiple nodes can operate on your large file simultaneously.

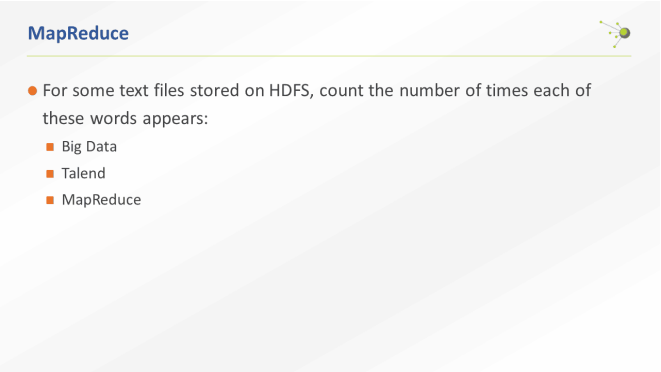


Your file blocks are replicated three times by default and distributed in different data nodes. Then the address of the different blocks is stored in a special node called a name node. If one node fails, it's still possible to find a copy on another node.

Another benefit of duplicating and distributing file blocks is that you increase the data locality. That means you'll have a better chance to find your blocks quickly, which will improve your computational performance.



Usually, high availability is configured with a standby name node. This standby name node is synchronized with the main name node while it's available. If the main name node fails, the standby takes over. The key concept here is the name node. In order to work on your cluster, many Big Data components within Talend Studio require your name node address.

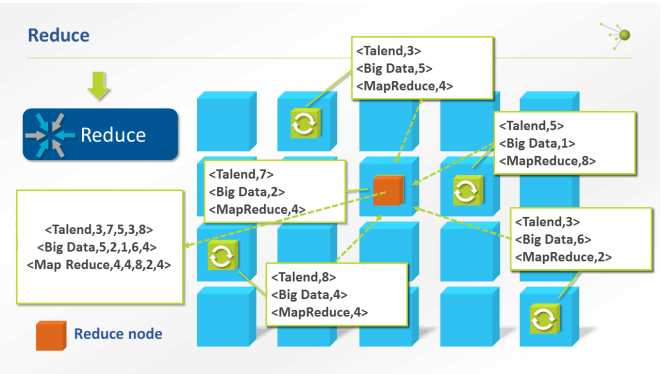


To illustrate the use of the MapReduce framework we take a basic word count use case.

Given some text files stored on HDFS, we want to count the number of times that certain words are mentioned. For example, suppose we want to know how many times the words Big Data, Talend, and MapReduce appear in the text files.

The MapReduce framework processes your data in two stages. Stage one, map, and stage two, reduce. First, the mapper processes each file block independently. The result is given as key-value pairs.

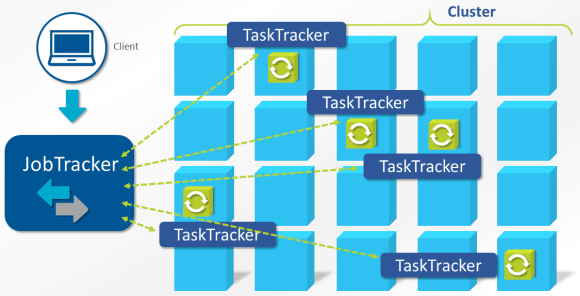
As the file blocks are distributed, the computation is done in parallel by nodes, each hosting a file block. This allows you to get your results faster than if you were doing this computation in a single node.



A reduce node is designated. Then the results are sent to the reducer.

The results are shuffled and sorted before reduction. And finally, the reducer aggregates the result.

JobTracker and TaskTracker



The JobTracker splits our job into tasks, and dispatches them to a TaskTracker running on each node of your cluster. Each task can be either a map or a reduce task.

Each TaskTracker gives updates on its task to the JobTracker, which will send information back to the client on the overall job progress.

The JobTracker is one of Talend Studio's entry points into your cluster. Just as for the name node, many components require that you provide the JobTracker address.

MapReduce limitations

MapReduce has an inflexible, slot-based, memory management model

- Each TaskTracker is configured at startup to have a certain numer of slots
- A task is executed in a single slot
- Slots are configured with maximum memory on cluster startup
- The model is likely to cause over- and underutilization issues

MapReduce also has scalability issues (approximately 4,000 machines)

To summarize, MapReduce version 1 handles processing and resource management and uses a static master-slave model.

This solves some Big Data issues, but the resource management is not optimal.

MapReduce version 1 has some drawbacks. It has an inflexible slot-based memory management model.

Each TaskTracker is configured at startup to have a certain number of slots.

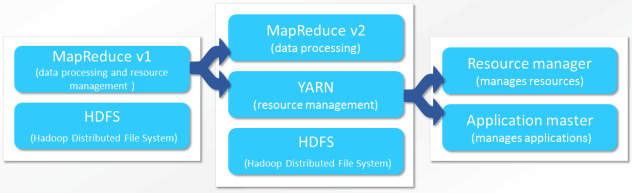
And a task, map or reduce, is executed in a single slot.

Each slot is configured with a maximum memory at startup of the cluster.

YARN & MapReduce v2

YARN (Yet Another Resource Negotiator)

MapReduce 1 was reworked to run on top of YARN

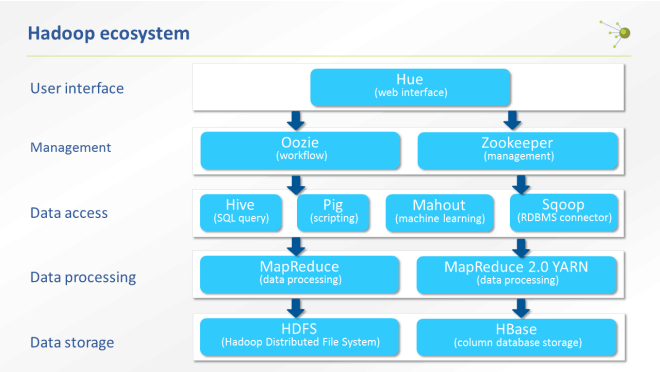


Due to the limitations of MapReduce v1, a new framework has been created called YARN.YARN is designed to manage only resources.It facilitates the development of distributed applications of any kind and is not limited to MapReduce.For example, Spark can run on top of YARN. Spark is covered in the Big Data Advanced course.

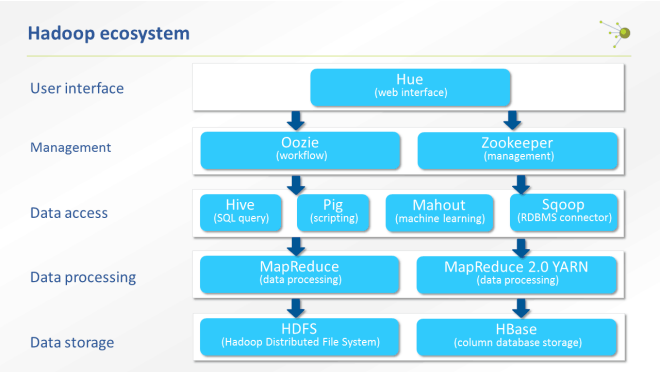
YARN provides daemons just as MapReduce does. The JobTracker daemon was introduced with MapReduce.In YARN, the JobTracker is split into two daemons. The first one is the ResourceManager, which administers resources on the cluster.The second daemon is the Application Master. It manages applications such as MapReduce applications.

Yet Another Resource Negotiator (YARN)

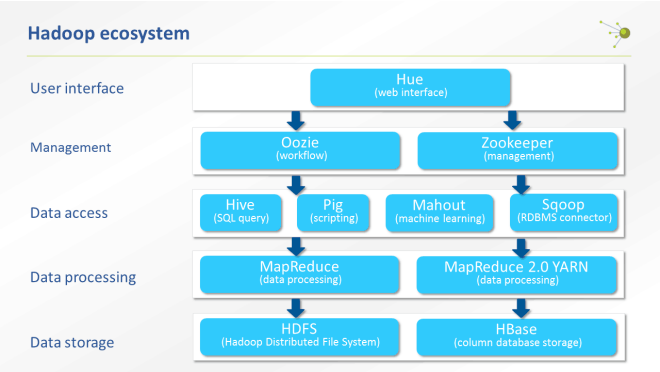
- **YARN** is a framework for **managing resources**
- Facilitates development of all types of distributed applications (not limited to MapReduce)
- Handles resource management
- Provides daemons. The JobTracker is split into two daemons:
  - **Resource Manager**, which administers resources on the cluster
  - **ApplicationMaster**, which manages applications such as MapReduce



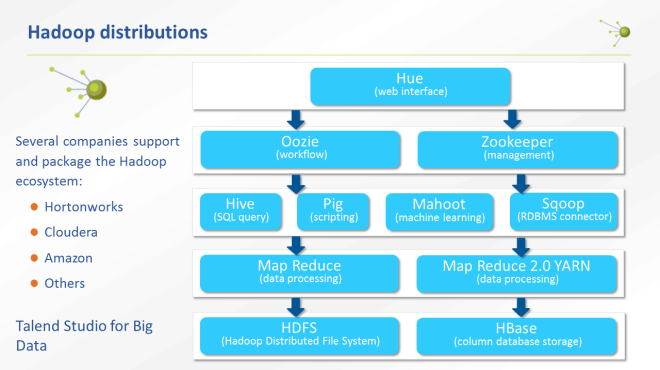
Hadoop is an open-source project from Apache. It's not actually a single product but instead is a collection of several components. Some of them are illustrated here. The first one is HDFS which stores your data files. An alternative data storage method is HBase, a No SQL database optimized for sparse data. Sparse data sets have a lot of NULL values. On top of HDFS sits YARN. To process your data using YARN, you can use MapReduce v2. Alternatively, you can write your own Java code, or use other options.



With Hive, you can create tables on your cluster and then process them using a SQL-like query language. You can also use Pig, which is a scripting language. It is a higher-level language than Java, which makes it much easier to write your processing. Your Pig code will be automatically translated into mappers and reducers and then run on your cluster. Another option is to use Mahout, which is a machine learning library.

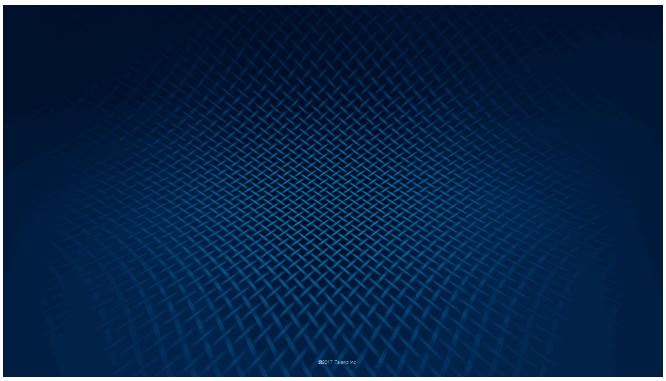


Sqoop is a tool designed for efficiently transferring bulk data between Apache Hadoop and structured data stores, such as relational databases. Oozie is an application used to schedule Hadoop jobs. It combines multiple jobs sequentially into one logical unit of work. Oozie supports Hadoop jobs for MapReduce, Pig, Hive, and Sqoop. Zookeeper is a centralized service for maintaining configuration information, naming, providing distributed synchronization, and providing group services. All these kinds of services are used by distributed applications. Hue is a set of web applications that allow you to interact with your cluster. Hue applications help you browse your files stored on HDFS, track your MapReduce Jobs, work with Hive, or use Oozie workflows.



Those separate projects can be installed independently, or you can use a packaged version of Hadoop. Several companies offer a prepackaged version of Hadoop, for example Hortonworks, Cloudera, and Amazon.

Talend Studio for Big Data supports all of these distributions and much more.



Intentionally blank

LESSON 2

Basic Concepts

This chapter discusses:

Concepts

Outline

- Overview
- Lesson objectives
- Monitoring the Hadoop cluster
- Creating the Hadoop cluster metadata
- Wrap-up

Lesson objectives

After completing this lesson, you will be able to:

- Open a project in the Talend Studio
- Connect to Cloudera Manager
- Connect to Hue
- Create cluster metadata

Overview

During this training you will have access to two virtual machines:

- A Windows instance hosting the **Talend Studio 6.3.1**
- A Linux instance hosting a **pseudo-distributed Cloudera CDH 5.8** cluster (a demo cluster with only one node)

This lesson will help you set up your training environment by connecting to these environments:

- **BDBasics Project** in the Studio
- **Cloudera Manager**
- **Hue**

You will also create your **cluster metadata**.

Monitoring the Hadoop cluster

Overview

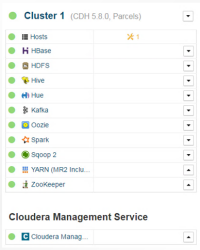
For this training, you will use a Cloudera CDH5.8 cluster running necessary Hadoop functionalities:

- HDFS (file system)
- YARN (processing engine)
- HBase (file system)
- Hive (table storage and processing)
- Sqoop (transfer between RDBMS and HDFS)
- Cloudera Manager (administration)
- Hue (files, tables, and Jobs browser)

Monitoring the Hadoop cluster

Connect to Cloudera Manager and check services' health

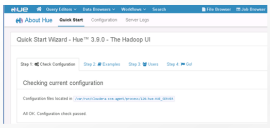
- Cloudera Manager is a web UI that performs administration tasks on the cluster
- In a web browser, navigate to **http://hadoopcluster:7180**
- Log in with:
  - Username: **admin**
  - Password: **admin**



Monitoring the Hadoop cluster

Connect to Hue

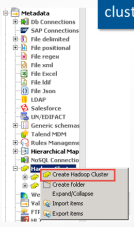
- Navigate to **http://hadoopcluster:8888**
- Log in with:
  - User name: **student**
  - Password: **training**

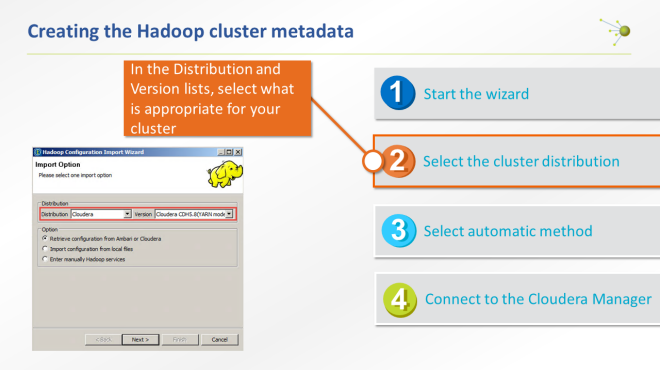


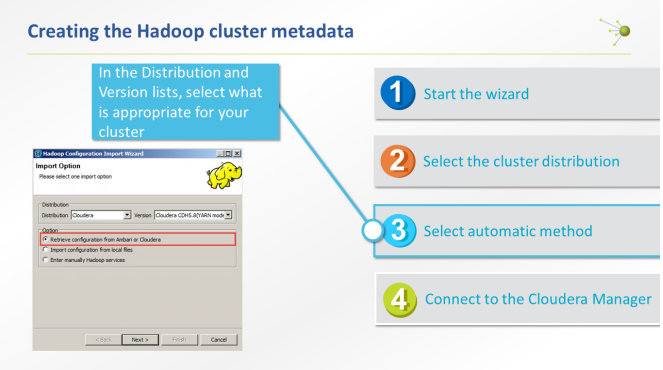
Creating the Hadoop cluster metadata

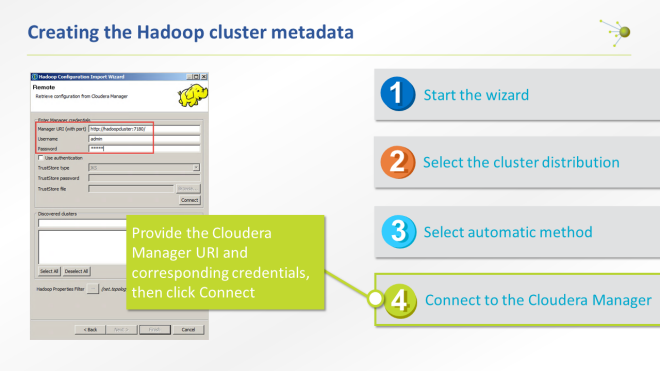
From the Repository, create a new Hadoop cluster metadata.

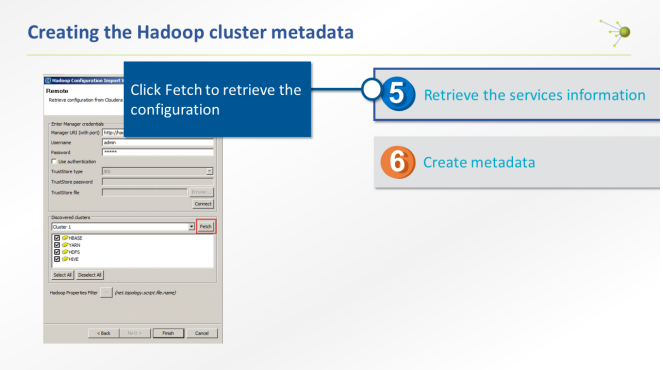
1. Start the wizard
2. Select the cluster distribution
3. Select automatic method
4. Connect to the Cloudera Manager




Creating the Hadoop cluster metadata
In the Distribution and Version lists, select what is appropriate for your cluster
1 Start the wizard
2 Select the cluster distribution
3 Select automatic method
4 Connect to the Cloudera Manager

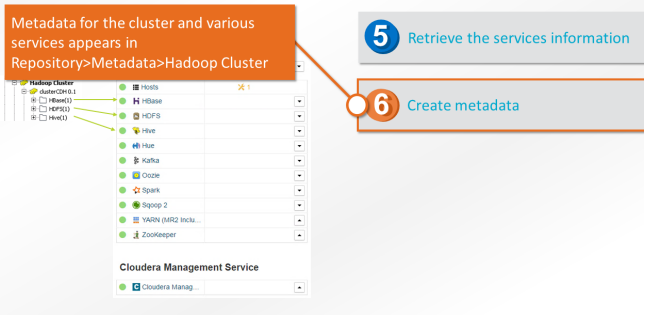

Creating the Hadoop cluster metadata
In the Distribution and Version lists, select what is appropriate for your cluster
1 Start the wizard
2 Select the cluster distribution
3 Select automatic method
4 Connect to the Cloudera Manager


Creating the Hadoop cluster metadata
Provide the Cloudera Manager URI and corresponding credentials, then click Connect
1 Start the wizard
2 Select the cluster distribution
3 Select automatic method
4 Connect to the Cloudera Manager


Creating the Hadoop cluster metadata
Click Fetch to retrieve the configuration
5 Retrieve the services information
6 Create metadata

Creating the Hadoop cluster metadata

Creating the Hadoop cluster metadata



Creating the Hadoop cluster metadata

Hadoop configuration files

You can also use the Hadoop configuration files to create cluster metadata

- In C:\StudentFiles\HadoopConf, find the configuration files:

Creating the Hadoop cluster metadata

Lab overview

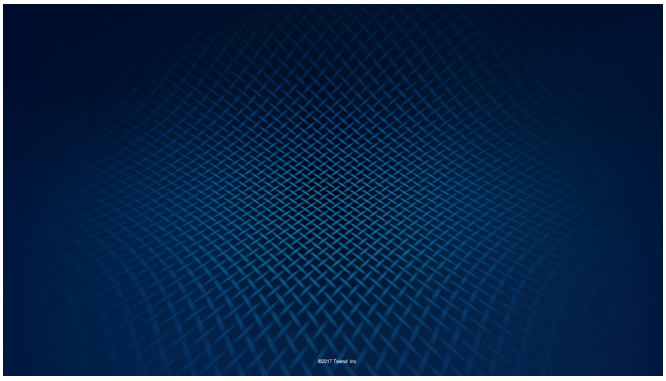
To run Big Data Jobs, the Talend Studio requires **information on how to connect** to your cluster. You can either configure the connection information in each component or store the configuration in metadata in the repository, then reuse it in components as needed.

This lab covers the ways to create cluster metadata:

- Manually
- Using the configuration files
- Using the Hadoop Configuration Import Wizard

Wrap-up

- Opening a project
- Monitoring the Hadoop cluster
  - Cluster is monitored through Cloudera Manager web interface
- Creating cluster metadata
  - Metadata can be created manually, using the Hadoop Configuration files, or by automatically connecting to Cloudera Manager



Basic Concepts

Overview

During this course, you will be assigned a preconfigured Hadoop cluster. The Hadoop cluster was built with a Cloudera CDH 5.8 distribution. The purpose of this exercise is to try different functions, not to have a production cluster. So, this training cluster is in pseudo-distributed mode. That means that there is only one node. This is enough to understand the different concepts in this training.

Before starting to create Jobs to read and write data to and from HDFS, there are some prerequisites. First, you will open a new project in the Talend Studio. Then, you will connect to your cluster to monitor it, using Cloudera Manager and Hue. For your Jobs to succeed, the cluster must be up and running. So, if a service fails you need to know how to restart it.

Finally, you will create your cluster Metadata. This step will avoid repetitive configuration of components in the Jobs you will create.

Objectives

After completing this lesson, you will be able to:

- Open a new project in Talend Studio
- Connect to the Cloudera Manager
- Connect to Hue
- Create Cluster Metadata manually, using configuration files, and automatically

Before you begin

Be sure that you are working in an environment that contains the following:

- A properly installed copy of Talend Studio
- A properly configured Hadoop cluster
- The supporting files for this lesson

Everything has already been set up in your training environment.

The first step is to open your training project in the Talend Studio.

Opening a Project

Task outline

Before developing Talend Jobs, you will need a project to store them in. In this exercise, you will open a pre-existing project for your Big Data Jobs.

Accessing the training environment

For this course, two virtual machines have been set up for you. The first machine is a Windows machine where the Talend Studio is installed. This is where you will create your Job.

The second machine is a Linux machine hosting a Cloudera CDH5.8 cluster. You do not need to access this machine. In fact, the cluster is monitored from the Windows machine through a Web Browser.

To connect to the Windows machine, in your Skytap environment, run the machine named **TalendStudio**.

To start your training cluster, run the Linux machine.

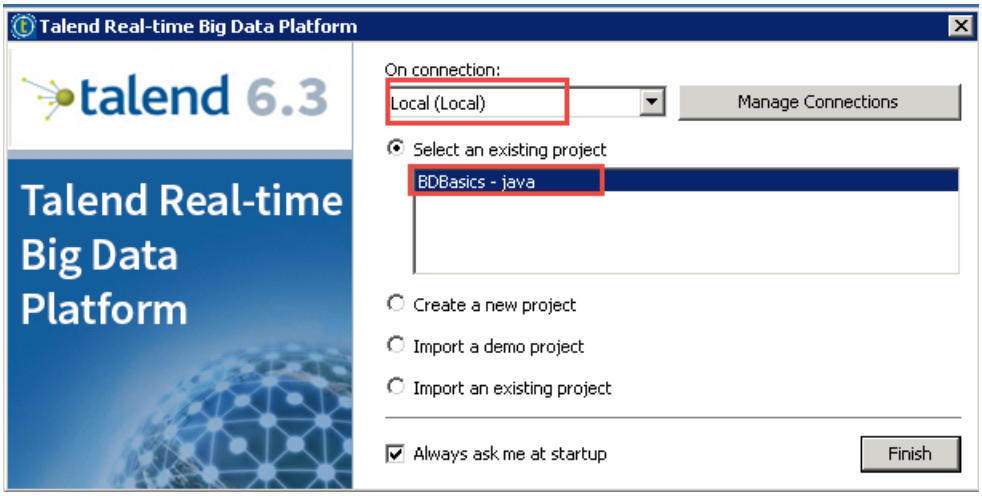
Run

1. START TALEND STUDIO
   Double-click the **Talend Studio shortcut** on your desktop to run Talend Studio.

2. OPEN YOUR TRAINING PROJECT
   Using the Local connection, open the *BDBasics* project.

a. On the **On Connection** list, make sure that the **Local (Local)** connection is selected.

b. Click **Select an existing project** ,then click *BDBasics* in the **Project** list.

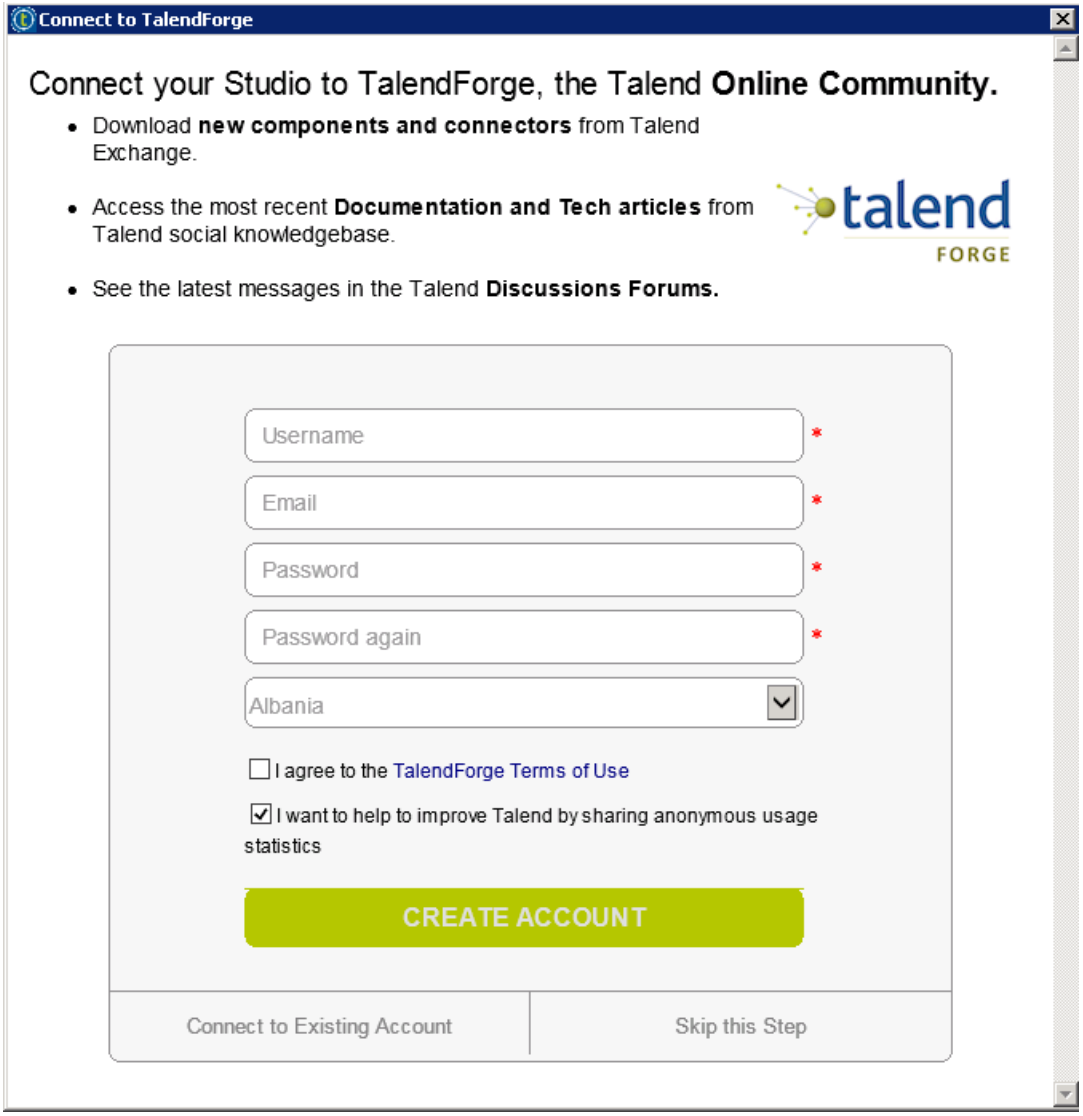


The project has already been created for you.
Note: You may have a different version of the Studio in your training environment. However, you will have the same functions as in the Talend Real-time Big Data Platform.

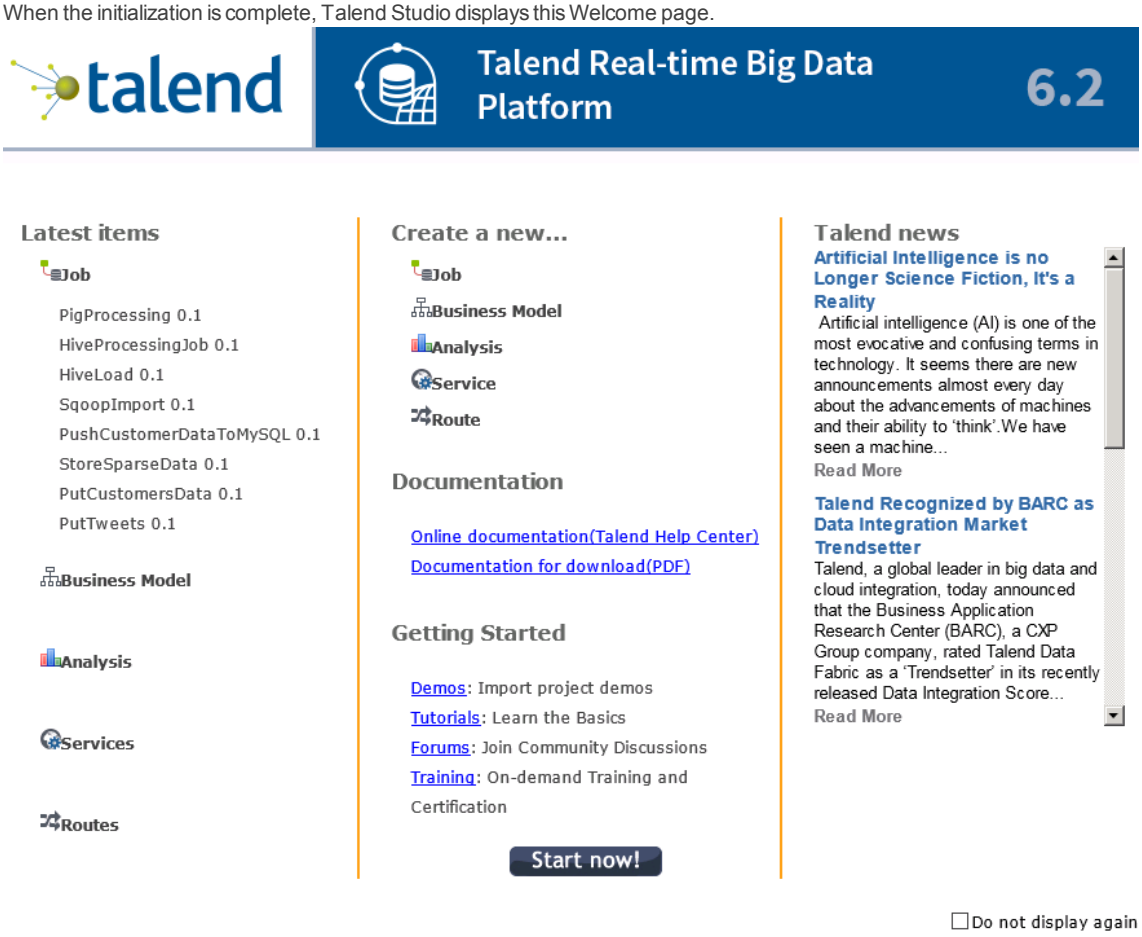
c. Click **Finish** to open the project.

Talend Forge

1. LOG IN
When the **Connect to TalendForge** window appears, log in with your existing Talend account, or create a new one.

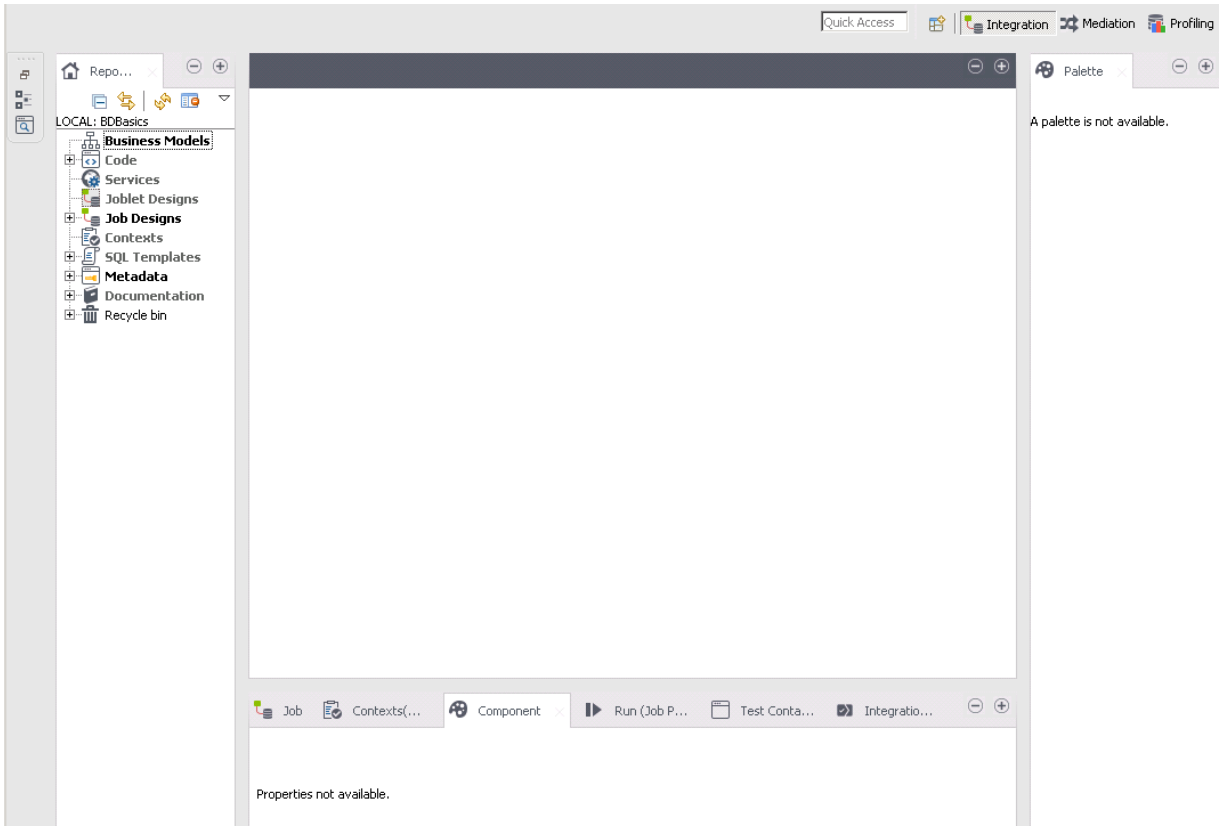

Connect to TalendForge
Connect your Studio to TalendForge, the Talend Online Community.
Download new components and connectors from Talend Exchange.
Access the most recent Documentation and Tech articles from Talend social knowledgebase.
See the latest messages in the Talend Discussions Forums.
talend
FORGE
Username
Email
Password
Password again
Albania
I agree to the TalendForge Terms of Use
I want to help to improve Talend by sharing anonymous usage statistics
CREATE ACCOUNT
Connect to Existing Account
Skip this Step

2. When the initialization is complete, Talend Studio displays this Welcome page.



Start

Click **Start now!**. The Talend Studio main window appears, ready for you to create Jobs:



The next step is to connect to your Hadoop cluster.

Monitoring the Hadoop Cluster

Task outline

In order to develop Jobs using HDFS, HBase, and Hive to store your data, you need a running Hadoop cluster.

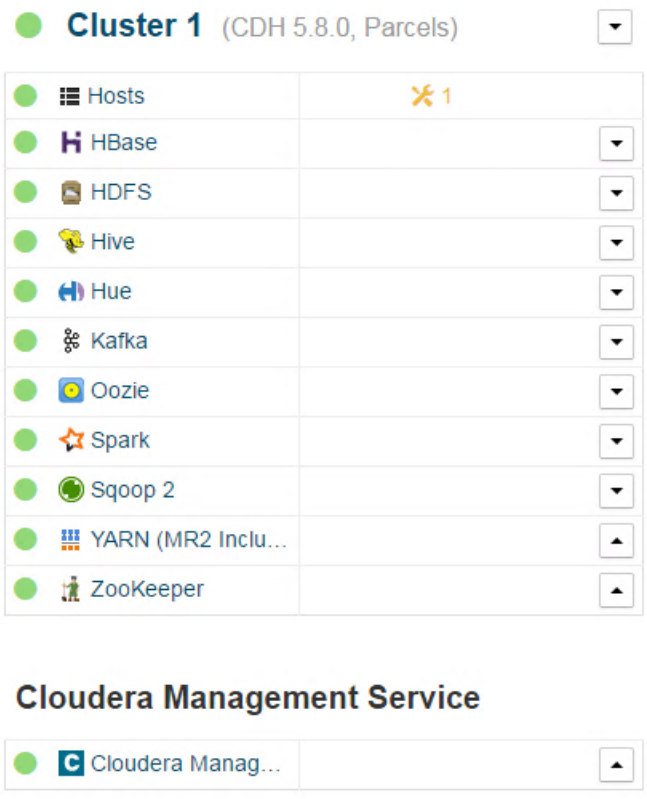
For this training, a Hadoop cluster has been installed using a Cloudera CDH 5.8 distribution. It is running the core Hadoop functions, such as HDFS, YARN, HBase, Hive or Sqoop. It also runs Hue, which will help you to browse your files and tables.

Connect to Cloudera Manager

1. CONNECT TO THE Cloudera Manager
   Click the **Cloudera Manager** shortcut in your web browser.
   Or navigate to the url *http://hadoopcluster:7180*.
   If you can't access the web page, wait a couple of minutes so that the service starts, and try again.
2. LOGIN
   Enter your credentials and log in.
   a. In the **Username** box, enter *admin* then, in the **Password** box, enter *admin*.
   b. Click **Login**.

Check services' health

The Cloudera Manager is a web interface to monitor and perform administration tasks on a cluster. It helps to check services' health and to restart services individually if needed.



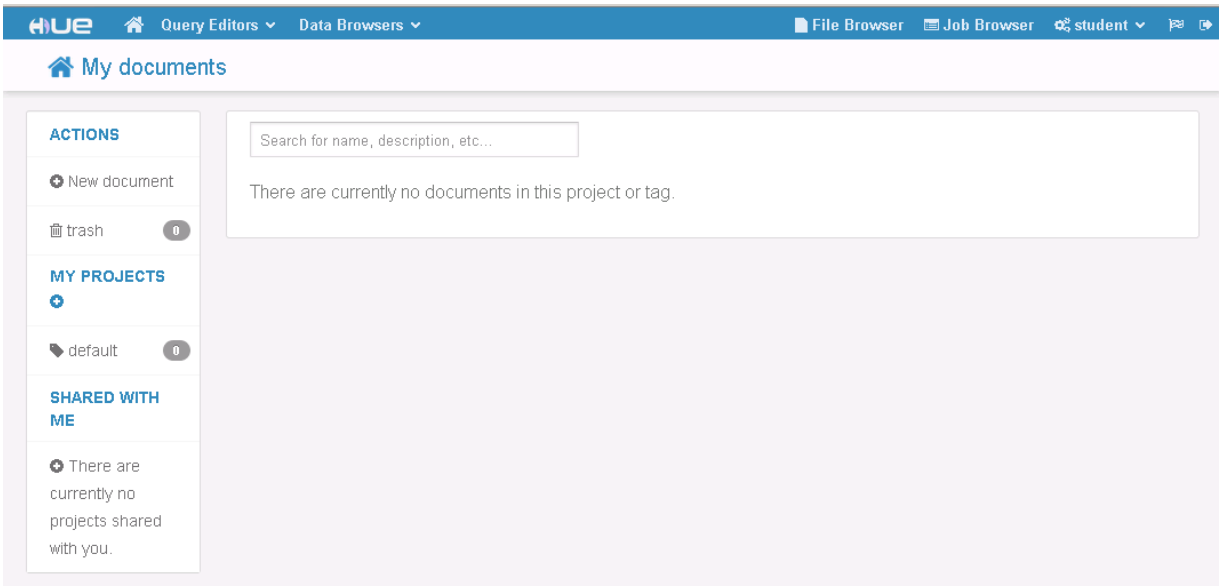
If a service is red flagged, it means it is in bad health. You can restart it individually by clicking the **black arrow** on the right side of the service then clicking **Restart**.

If you can see interrogation points on a white background instead of green or red ligths, restart the Cloudera Management services. This is done by clicking the right arrow next to Cloudera Management services and then clicking Restart. This should fix your issue and you should be able to monitor your cluster as shown above. Otherwise, refer to the troubleshooting guide.

Connect to Hue

Hue is a web interface that helps to check on what is done on your cluster. You can browse your files and tables. It is also possible to track Map Reduce tasks.

1. CONNECT TO Hue
   Navigate to the **Hue** web interface.
   a. Open a new tab in your web browser.
   b. Click the **Hue** shortcut or navigate to *http://hadoopcluster:8888*.
2. LOGIN
   Enter your credentials and log in.
   a. In the **Username** box, enter *student*. In the **Password** box, enter *training*.
   b. Click **Sign in**.
   You are now logged in Hue.



The next step is to create Metadata on your Hadoop Cluster.

Creating Cluster Metadata

Task outline

To be able to run Big Data Jobs, Talend Studio needs to be connected to a running Hadoop Cluster. The connection information can be configured in each component individually, or the configuration can be stored as metadata in the Repository and be reused as needed in the different components.

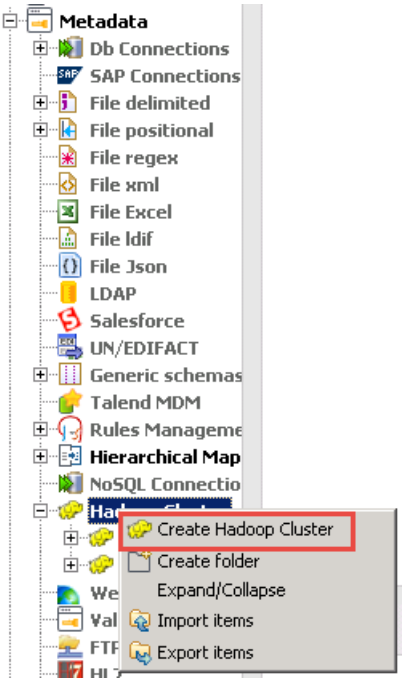
Instead of individual connection configuration of components and Jobs, you will use the second approach.

You will now create your cluster metadata using three different methods. First, you will configure the connection to the cluster manually, next, from the Hadoop configuration files, and last, using a Wizard.

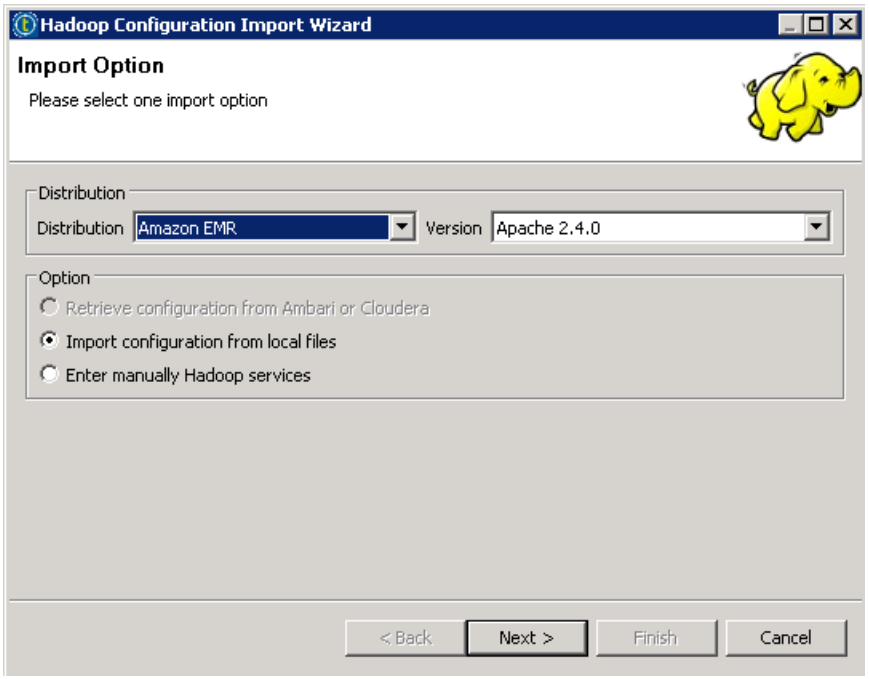
Manual configuration

The first way to create a Hadoop cluster Metadata is to create it manually. This requires that you already have information about your cluster, such as the Namenode URI, and either the Resource Manager address or the Job Tracker URI- depending on if you will use YARN or Map Reduce v1. You may also need other information such as the Job History server, or the Resource Manager Scheduler location.

1. CREATE A NEW HADOOP CLUSTER METADATA
   Create a new Hadoop cluster metadata named *TrainingCluster_manual*.
   a. In the Studio, in the **Repository**, under **Metadata**, locate **Hadoop Cluster**.
   b. Right-click **Hadoop Cluster** then, click **Create Hadoop Cluster**:

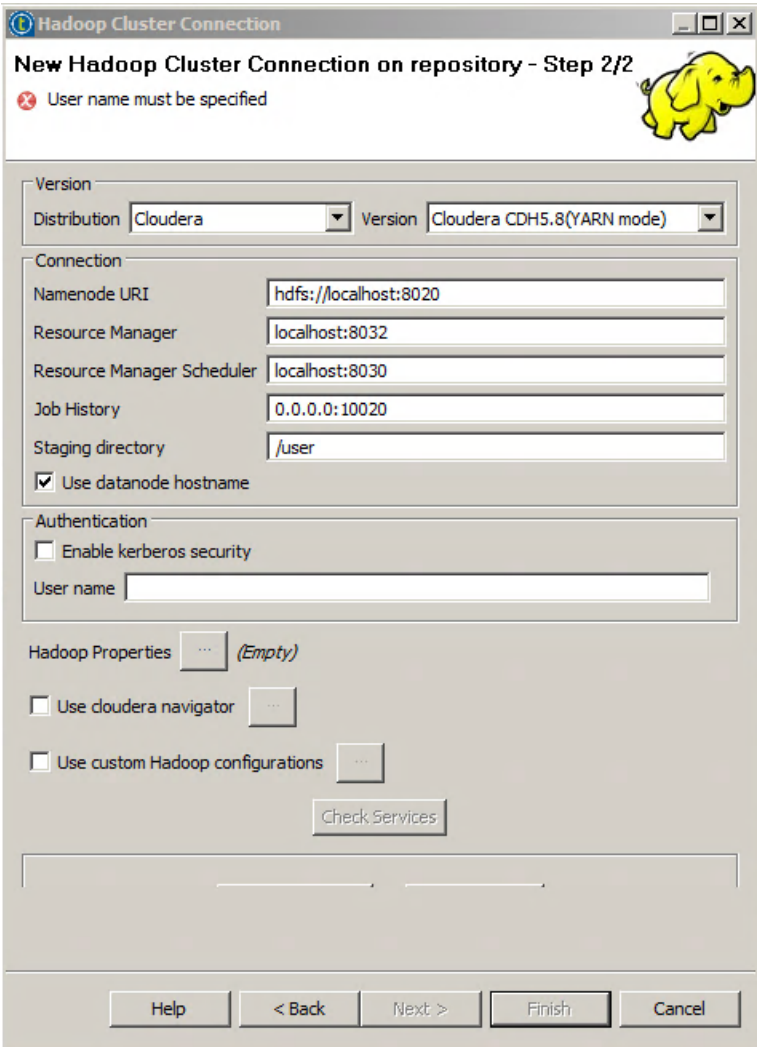


c. In the **Name** box, enter *TrainingCluster_manual*, then click **Next**.
The **Hadoop Configuration Import Wizard** opens:



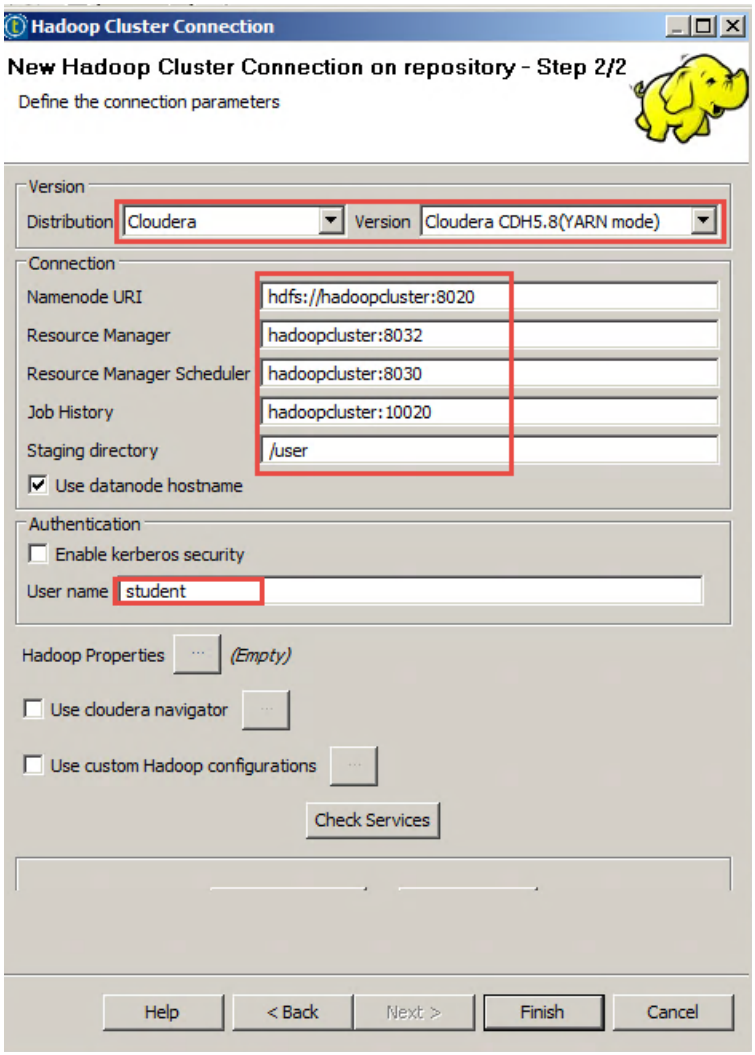
2. SELECT THE DISTRIBUTION AND THE VERSION
In the **Distribution** list, select *Cloudera*, and in the **Version** list, select *Cloudera CDH5.8(YARN mode)*.
3. SELECT THE MANUAL CONFIGURATION
Select **Enter manually Hadoop services**, then, click **Finish**.

The **Hadoop Cluster Connection** window opens.



4. OBSERVE THE DEFAULT CONFIGURATION
   Check that the Distribution information is correct.
   There are a few values preconfigured, such as the Namenode URI and the Resource Manager address.
   The localhost value and the 0.0.0.0 value must be changed to the IP address or to the DNS name of your cluster. If the cluster was configured keeping the default port values, then 8020 and 8032 are the host port for the Namenode and the Resource Manager respectively.
   The Hadoop cluster has already been configured for you with the name **hadoopcluster**.
5. CONFIGURE THE CONNECTION
   Replace the localhost and 0.0.0.0 values by the hostname of the cluster.
   a. Configure the connection as follows:
      - **Namenode URI**: *hdfs://hadoopcluster:8020*
      - **Resource Manager**: *hadoopcluster:8032*
      - **Resource Manager Scheduler**: *hadoopcluster:8030*
      - **Job History**: *hadoopcluster:10020*
      - **Staging directory**: */user*
      - **User name**: *student*

b. Check your configuration.



c. Click **Finish**. Your cluster metadata appears under Repository/Metadata/Hadoop Cluster.

Discovering Hadoop configuration files

You can also create your Metadata using the Hadoop configuration files. The configuration files have been copied in the **HadoopConf** folder under **C:\StudentFiles\BDBasics**.

The Hadoop configuration files are .xml files that describe each parameter value of your Hadoop cluster. In the HadoopConf folder, you will find four files: core-site.xml, hdfs-site.xml, mapred-site.xml and yarn-site.xml. The files were copied from a Hadoop cluster, installed with the same distribution and version as your training cluster.

1. EXAMINE core-site.xml
   Open **core-site.xml** with **Notepad++**:

```
<!--Autogenerated by Cloudera Manager-->
<configuration>
  <property>
    <name>fs.defaultFS</name>
    <value>hdfs://hadoopcluster:8020</value>
  </property>
```

The first property that appears in this file is the location of the Namenode: hdfs://hadoopcluster:8020.

2. EXAMINE yarn-site.xml
   Open **yarn-site.xml** with **Notepad++**:

```
<property>
  <name>yarn.resourcemanager.address</name>
  <value>hadoopcluster:8032</value>
</property>
<property>
  <name>yarn.resourcemanager.admin.address</name>
  <value>hadoopcluster:8033</value>
</property>
<property>
  <name>yarn.resourcemanager.scheduler.address</name>
  <value>hadoopcluster:8030</value>
</property>
<property>
  <name>yarn.resourcemanager.resource-tracker.address</name>
  <value>hadoopcluster:8031</value>
</property>
<property>
  <name>yarn.resourcemanager.webapp.address</name>
  <value>hadoopcluster:8088</value>
</property>
```

   In this file, you will find the Resource Manager address and the Resource Manager Scheduler address.

3. EXAMINE mapred-site.xml
   Open **mapred-site.xml** with **Notepad++**:

```
<property>
  <name>mapreduce.job.reduce.slowstart.completedmaps</name>
  <value>0.8</value>
</property>
<property>
  <name>mapreduce.jobhistory.address</name>
  <value>hadoopcluster:10020</value>
</property>
<property>
  <name>mapreduce.jobhistory.webapp.address</name>
  <value>hadoopcluster:19888</value>
</property>
```

   In this file, you will find the Job History address.

You need all this information to create your Cluster Metadata. If you choose to create the cluster Metadata using the configuration files, the files will be parsed to find all these values.

Configuration with the Hadoop configuration files

1. CREATE A NEW HADOOP CLUSTER METADATA
   Create a new Hadoop cluster metadata and name it *TrainingCluster_files*.

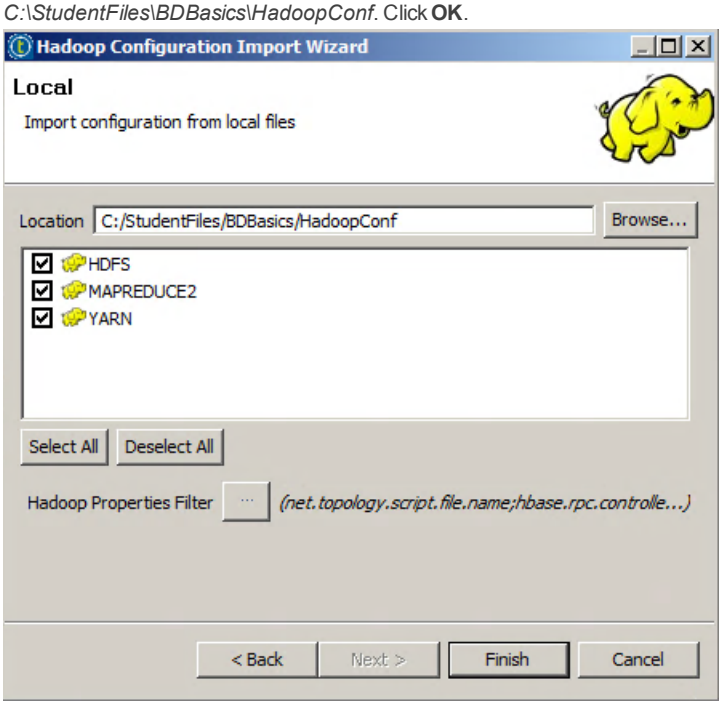
a. Right-click **Hadoop Cluster**, then click **Create Hadoop Cluster**.

b. In the **Name** box, enter *TrainingCluster_files* then, click **Next**.

2. SELECT THE DISTRIBUTION AND THE VERSION
In the **Distribution** list, select *Cloudera* and in the **Version** list, select *Cloudera CDH5.8(YARN mode)*.

3. SELECT A CONFIGURATION FROM FILES
Select **Import configuration from local files**,then click **Next**.

4. LOCATE THE CONFIGURATION FILES
Click **Browse...** then locate the configuration files under *C:\StudentFiles\BDBasics\HadoopConf*. Click **OK**.



5. FINALIZE THE CONFIGURATION
Set the user name to *student* and finalize the metadata creation.

a. Click **Finish**.

b. The configuration is almost done, except for the user name.
In the **User name** box, enter *student*, then click **Finish**.
Your cluster's metadata appears under Repository/Metadata/Hadoop Cluster.

Automatic configuration

The last way to configure your metadata is to connect to the Cloudera Manager, so that all of the connection information will be retrieved automatically to create the cluster's metadata.

1. CREATE A NEW HADOOP CLUSTER METADATA
Create a new Hadoop cluster metadata and name it *TrainingCluster*

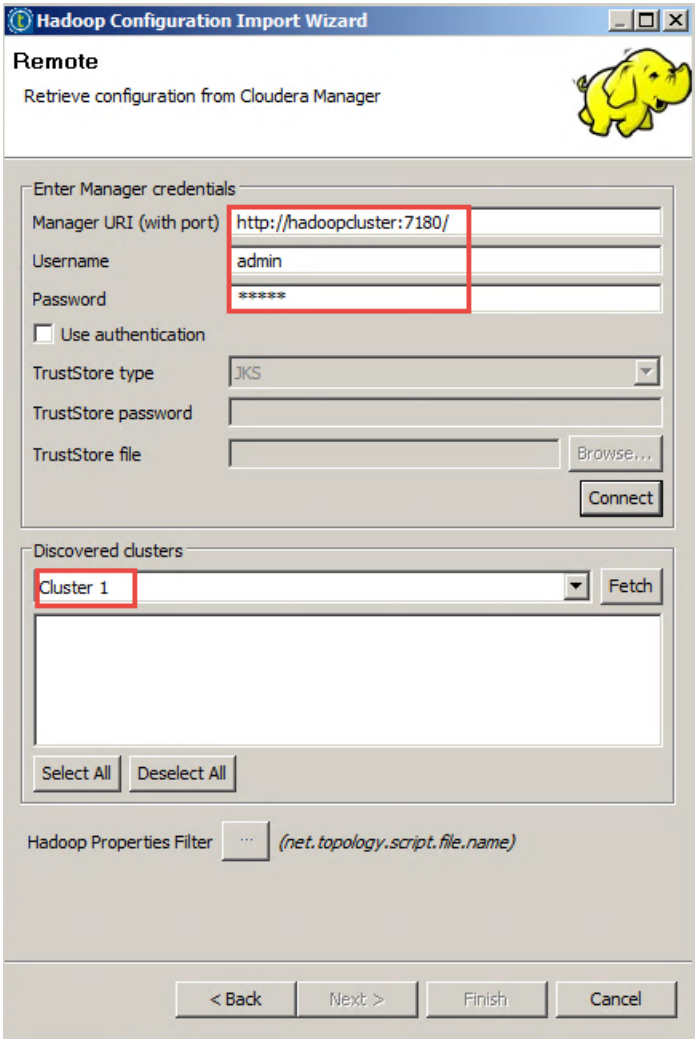
a. Right-click **Hadoop Cluster**, then click **Create Hadoop Cluster**.

b. In the **Name** box, enter *TrainingCluster*, then, click **Next**.

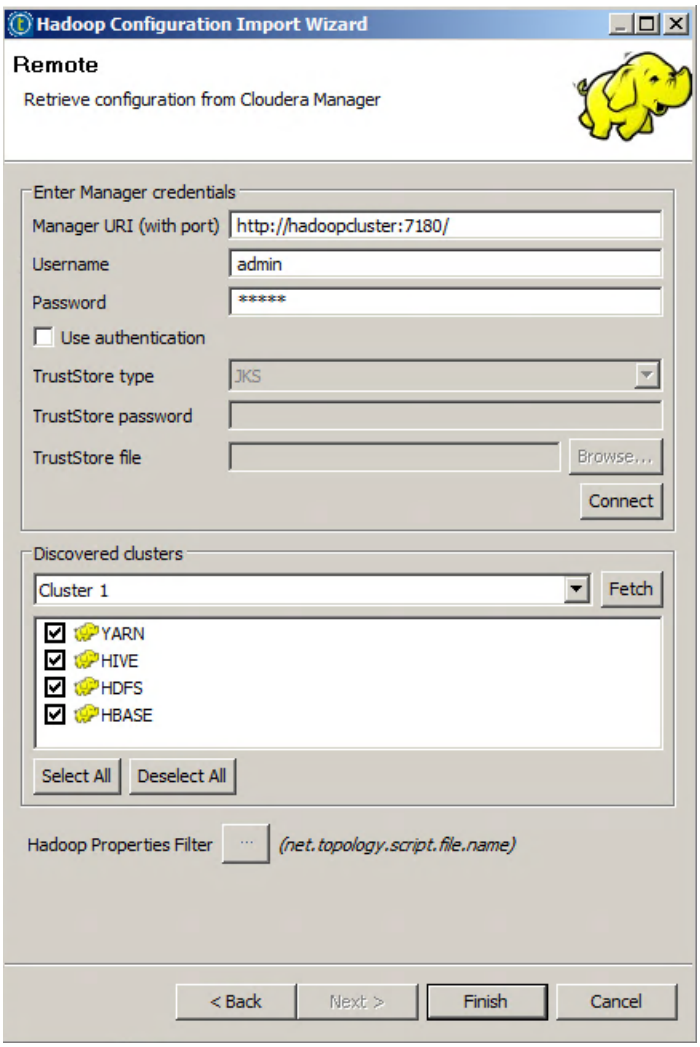
2. SELECT THE DISTRIBUTION AND THE VERSION
In the **Distribution** list, select *Cloudera* and in the **Version** list, select *Cloudera CDH5.8(YARN mode)*.

3. SELECT AN AUTOMATIC CONFIGURATION
Select **Retrieve configuration from Ambari or Cloudera** then, click **Next**.

4. SET THE Cloudera Manager URI
   In the **Manager URI (with port)** box, enter *http://hadoopcluster:7180*.
5. ENTER CREDENTIALS
   In the **Username** and **Password** boxes, enter *admin*.
6. CONNECT TO THE Cloudera Manager
   Click **Connect**. This will list all the clusters administered by the Cloudera Manager:



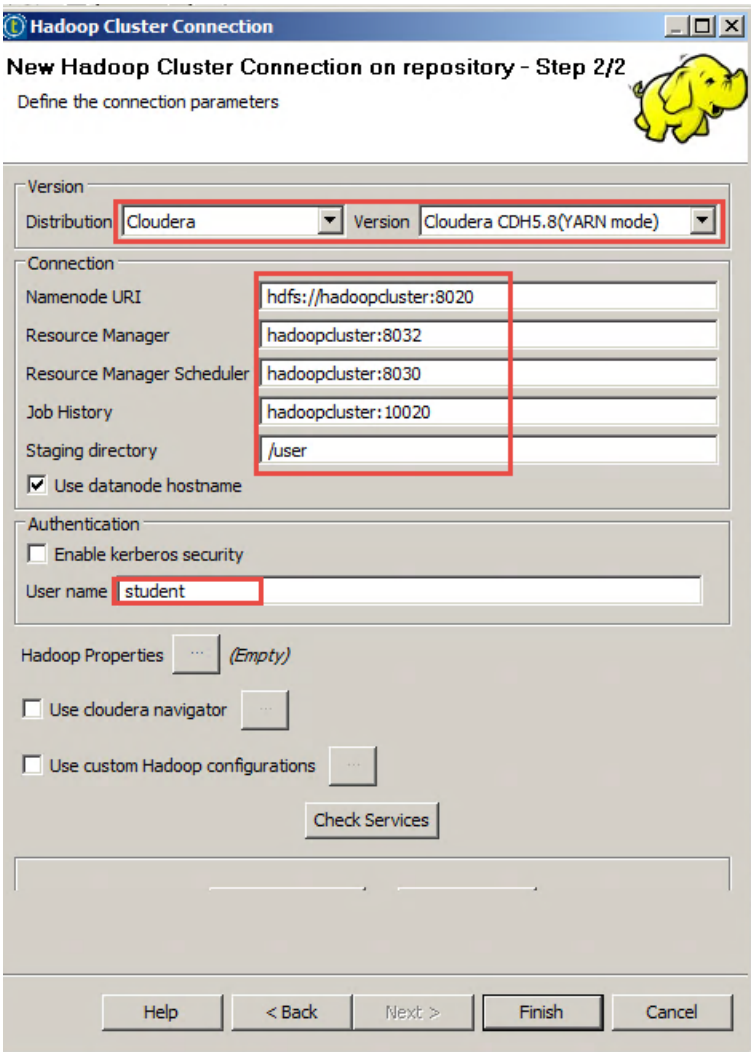
7. FETCH SERVICES
   Click **Fetch**. The wizard will retrieve the configurations files of all running services in your cluster:



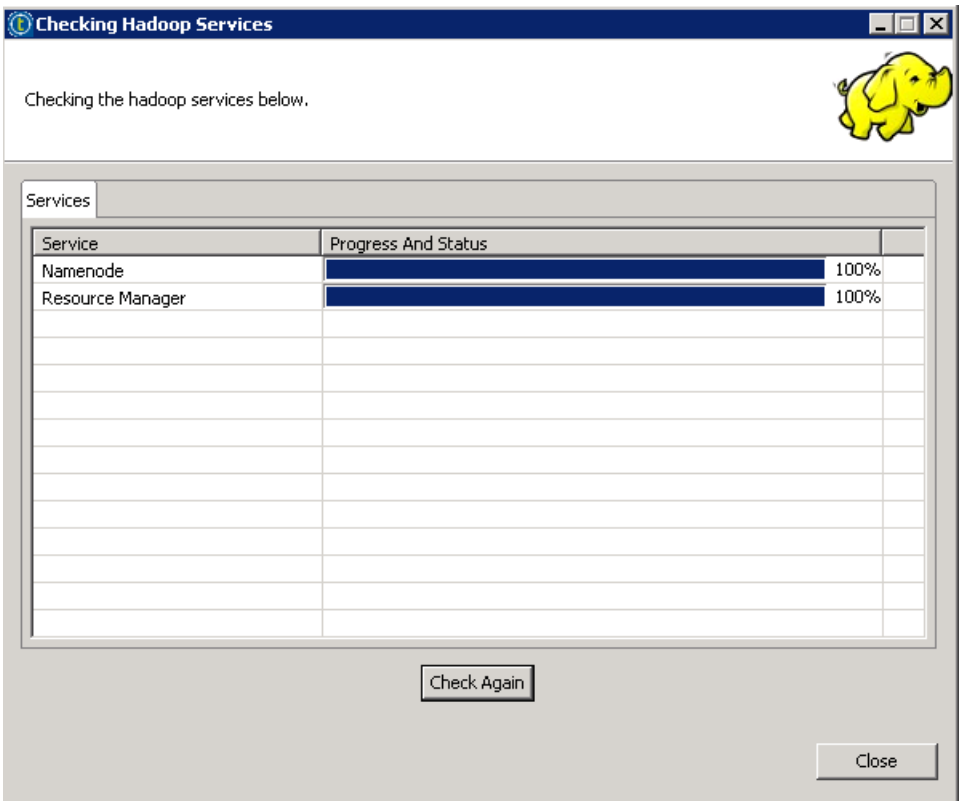
For each service listed, the wizard can create the corresponding metadata in the Repository.
For this lab, you will create metadata only for YARN. You will investigate later how to create metadata for HDFS, Hive and HBase.

8. CREATE METADATA ONLY FOR YARN
   You will create metadata only for the YARN service.
   a. Click **Deselect All** then, select **YARN**.
   b. Click **Finish**.

9. SET THE USERNAME
Enter *student* in the **User name** box.



10. CHECK SERVICES
Click **Check Services**, to check if you succeeded in connecting to the Namenode and the Resource Manager.



If the progress bars go up to 100%, and you have no error message, then your connection was successful.

11. FINALIZE THE HADOOP CLUSTER METADATA CREATION
Now, you have to complete the last steps to create the metadata in the Repository.
    a. Click **Close**.
    b. Click **Finish** to finalize the metadata creation. It will appears in the Repository under Metadata/Hadoop Cluster.

You have now created the same metadata three times using different techniques. For the next lessons, the last metadata, named TrainingCluster, will be used.

Next step

You have almost finished this section. Time for a quick review.

Review

Recap

In this lesson, you covered the key base knowledge required to be efficient in building and running Big Data Jobs.

First, you opened your project, then, you connected to Cloudera Manager and Hue. Cloudera Manager will help you to restart a service if needed. Hue will help you to browse your files and tables, and track the execution of your Map Reduce Jobs.

You also learned how to create Metadata on your cluster. You created the same Metadata three times using different techniques, so you can use any of them for your Jobs.

For the next lessons, the TrainingCluster Metadata will be used. Feel free to try any of the three Metadata. The results will be the same. If not, that means that a metadata is not well configured.

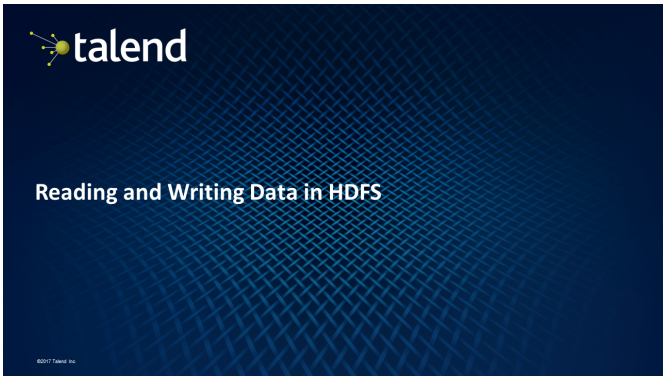
Intentionally blank

LESSON 3

Reading and Writing Data in HDFS

This chapter discusses:

Concepts



Outline

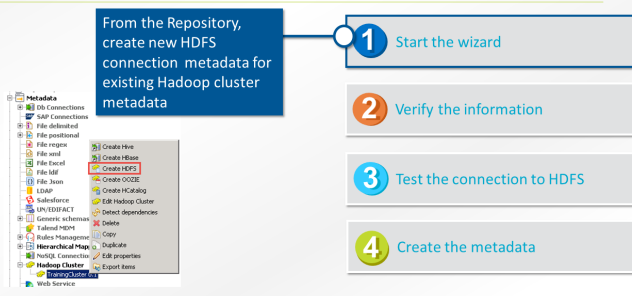
- Lesson objectives
- Creating HDFS connection metadata
- Using HBase to store sparse data sets on HDFS
- Lab overview
- Challenge
- Wrap-up

Lesson objectives

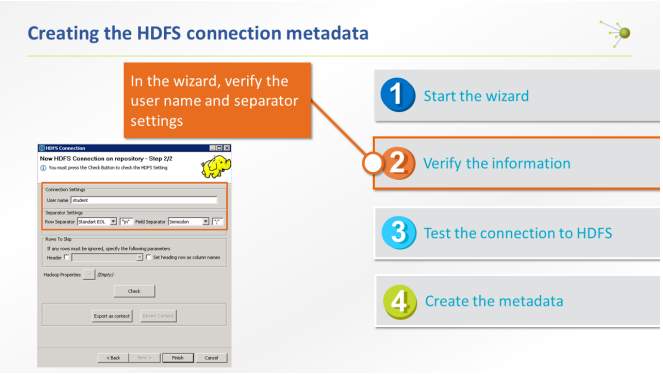
After completing this lesson, you will be able to:

- Write text files to HDFS
- Read text files from HDFS
- Use HBase to store sparse data on HDFS
- Configure connections to HDFS and HBase

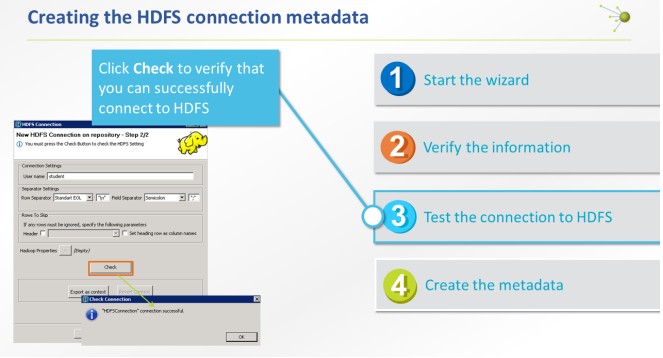
Creating the HDFS connection metadata



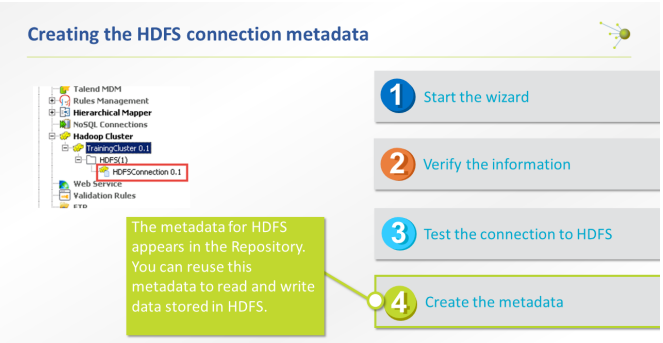
Creating the HDFS connection metadata



Creating the HDFS connection metadata



Creating the HDFS connection metadata



Using HBase to store sparse data sets on HDFS

Apache HBase is the **Hadoop column-oriented database**. It's an open-source, non-relational database modeled after Google BigTable. It provides the same functionality on top of Hadoop and HDFS.

HBase is useful when you need **random, real-time read/write access** to your big data. Furthermore, HBase can host very large tables—with billions of rows and millions of columns--and it's particularly **well suited for sparse data sets**.

Using HBase to store sparse data sets on HDFS

If your relational table looks like this (data missing in columns), it's considered "sparse" and a good candidate for HBase

	Col A	Col B	Col C	Col D	Col E
Row 01	Val1A				
Row 02	Val2A	Val2B	Val2C	Val2D	Val2E
Row 03	Val3A		Val3C		Val3E

Using HBase to store sparse data sets on HDFS

- HBase tables are organized by column. The columns are organized in groups called **column families**.
- When creating an HBase table, you must define the column families *before* inserting any data.
- Each column family can contain many columns.

Lab overview

The Hadoop Distributed File System (**HDFS**)—scales to hold petabytes of data. This lesson covers using Talend Big Data components to read and write data to HDFS. You will:

1. **Read a text file** storing customer information from your local system and **write it to HDFS**
2. **Read a collection of customer Twitter messages,** stored in separate files, and **write them individually** to HDFS
3. Develop a Job to **read a designated subset** of the Twitter files **back from HDFS**
4. Simulate a **sparse data set** and then **write it to HDFS** using **HBase-**dedicated components

Challenge

Put support file

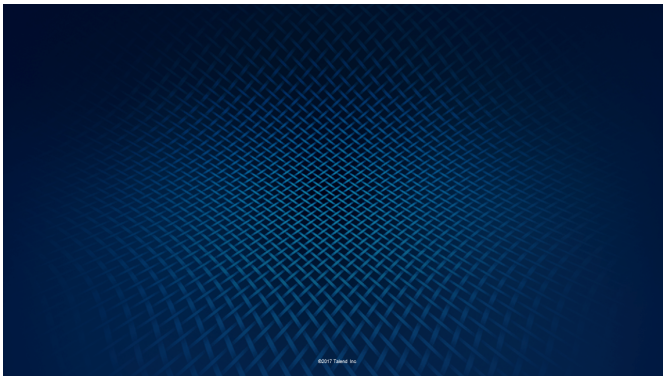
- Create a Job to write a file on HDFS:
  - Source file: C:/StudentFiles/support/support.xml
  - Target file: /user/student/support/support.xml

Double-up orders

- Create a Job to write a file of duplicated orders to HDFS
  - Source file: C:/StudentFiles/duplicated_orders
  - Target file: /user/student/support/support.xml
  - Sort the file before saving it in HDFS

Wrap-up

- Creating HDFS connection metadata
  - Connection to HDFS information stored in metadata, reusable in components
- Using HBase to store sparse data on HDFS
  - HBase allows random, real-time read/write access to sparse data sets



Reading and Writing Data in HDFS

Overview

Hadoop's file system—HDFS—scales to hold petabytes of data. In this lesson you will use Talend Big Data components to read and write data to HDFS.

First, you will read a text file that stores customer information from your local system and write it to HDFS. Then, you will read a collection of customer Twitter messages, stored in separate files, and write them individually to HDFS. Next, you will develop a Job to read a designated subset of the Twitter files back from HDFS.

Finally, you will simulate a sparse data set write it to HDFS using HBase dedicated components.

Objectives

After completing this lesson, you will be able to:

- Write text files to HDFS
- Read text files from HDFS
- Use HBase to store sparse data on HDFS
- Configure connections to HDFS and HBase

Before you begin

Be sure that you are in a working environment that contains the following:

- A properly installed copy of the Talend Studio
- A properly configured Hadoop cluster
- The supporting files for this lesson

The first step is to create a new Job to read a local text file and write it to HDFS.

Storing a File on HDFS

Task outline

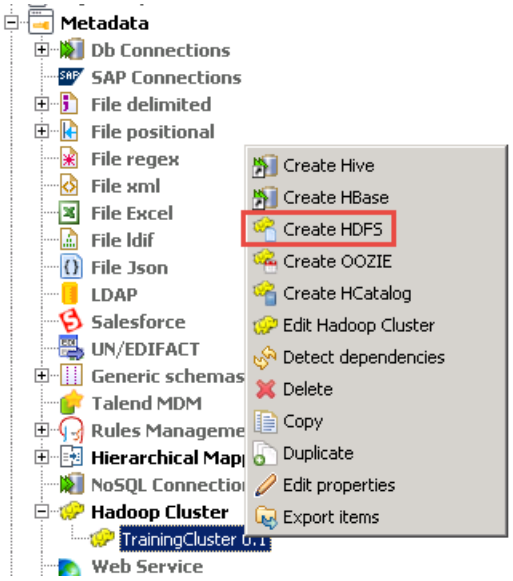
Your first step is to store text data on HDFS. In this lab, you are saving a file containing customer information such as first name, last name, city, state, etc...

As you did previously for your cluster connection, it is possible to create metadata to get connected to HDFS. Once connected to HDFS, you will be able to read and write files on HDFS.

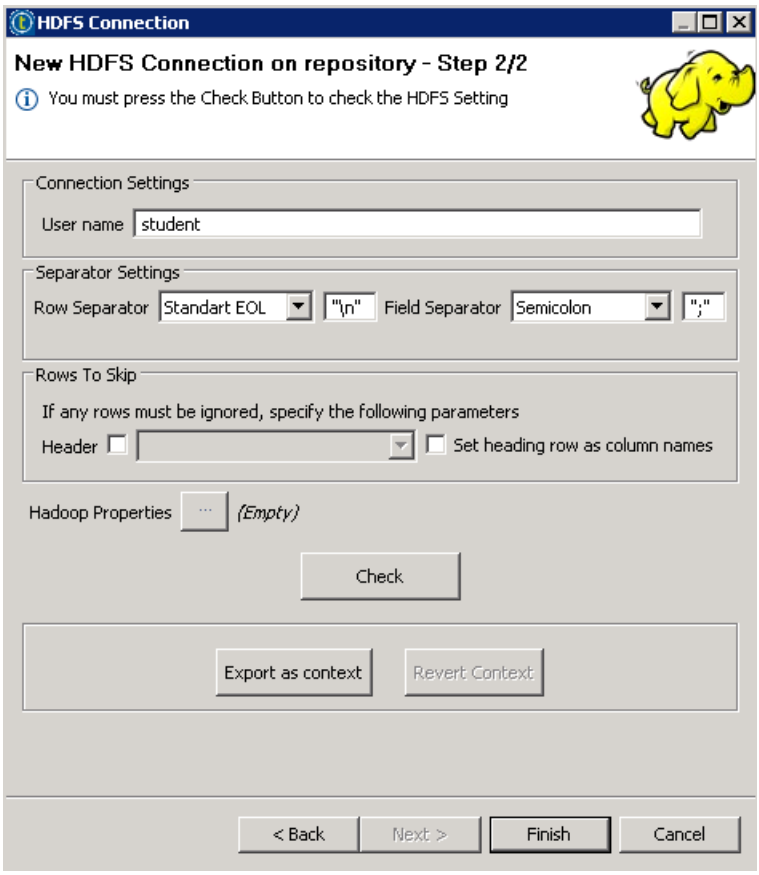
Configure HDFS connection

Now you will create metadata for an HDFS connection.

1. CREATE HDFS CONNECTION METADATA
   Right-click **TrainingCluster** under **Repository>Metadata>Hadoop Cluster**, then click **Create HDFS**:

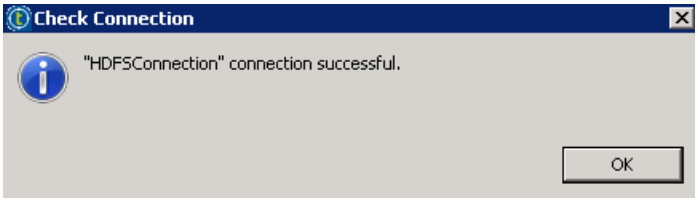


2. NAME THE METADATA
   In the **Name** box, enter *HDFSConnection* , then click **Next**:

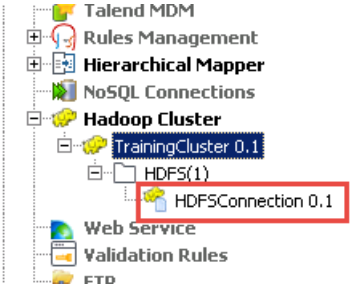


A default configuration is proposed, but you can adjust it to your needs.
For this training, you will keep the default values. This means that your username to connect to HDFS will be *student*, *"\n"* will be the row separator, and *";"* will be the field separator.

3. CHECK THE CONNECTION
Click **Check**. You should quickly be connected to your cluster, and you will have the following success message:



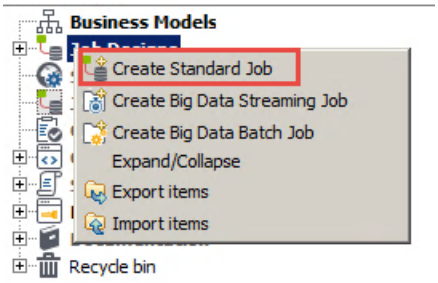
4. FINALIZE THE METADATA CREATION
Click **OK** , then click **Finish** to create your HDFS connection Metadata. It will appear in the repository, below the TrainingCluster metadata:



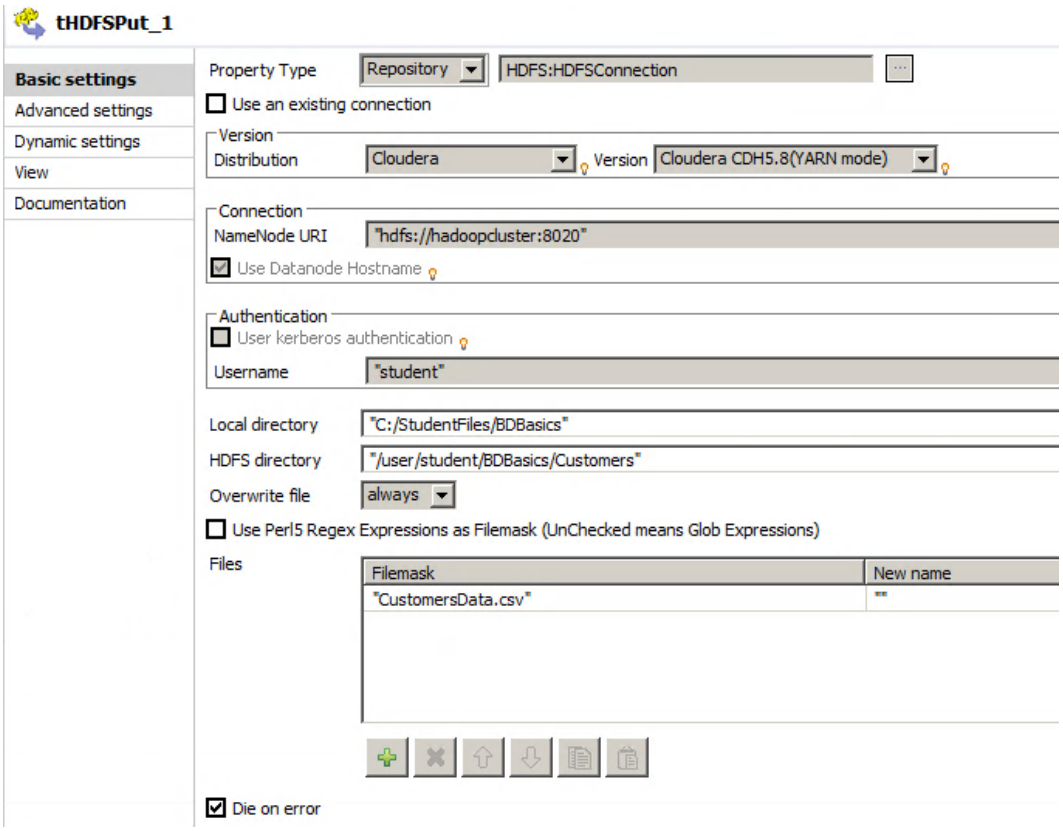
Create a Job to write data to HDFS

In the C:\StudentFiles\BDBasics folder, you will find the CustomersData.csv file. You will create a Job that will read this file and write it to HDFS.

1. CREATE A NEW STANDARD JOB
   Create a Standard Job named *PutCustomersData*.
   a. In the Repository, right-click **Job Designs** , then click **Create Standard Job**:



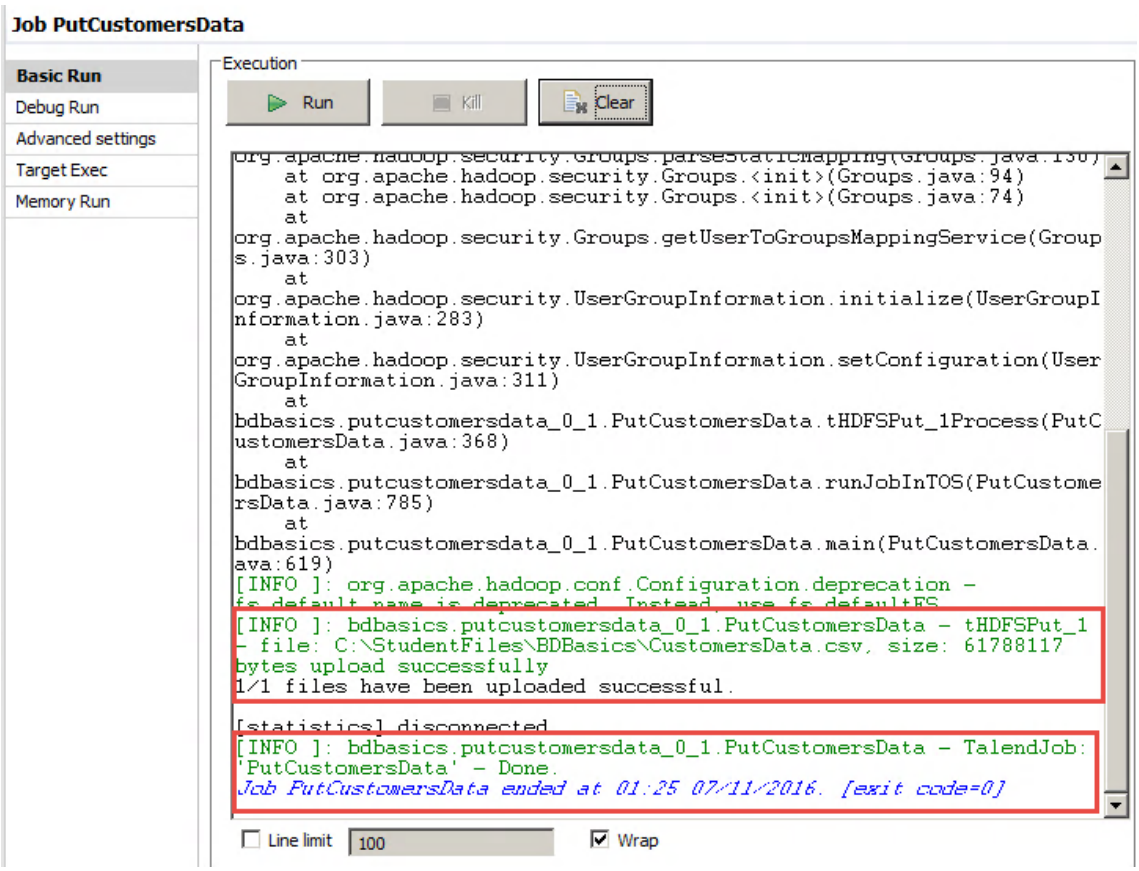
   b. Name your Job *PutCustomersData*.
2. ADD A tHDFSPut COMPONENT
   Place a **tHDFSPut** component on the design workspace. You use this component to move files from a local file system to HDFS.
3. CONFIGURE THE COMPONENT TO USE THE HDFSConnection METADATA
   Configure the **tHDFSPut** component to use the **HDFSConnection** metadata created previously.
   a. Double-click to open the **Component** view.
   b. In the **Property Type** list, select *Repository*.
   c. Click **(...)** , then locate the HDFSConnection metadata that you previously created.
4. CONFIGURE THE LOCAL FOLDER
   To configure the **Local directory**, click **(...)** , then navigate to "C:\StudentFiles\BDBasics".
5. CONFIGURE THE TARGET FOLDER
   Configure tHDFSPut to overwrite the */user/student/BDBasics/Customers* folder.
   a. In the **HDFS directory** box, enter */user/student/BDBasics/Customers*.
   b. In the **Overwrite file** list, select *always*. This tells the component to always replace existing files when a new file of the same name is being saved.
6. READ THE CUSTOMERS DATA FILE
   Configure the **Files** table to read the **CustomersData.csv** file.
   a. Click the **green plus sign** below the **Files table.**
   b. In the **Filemask** column, enter *"CustomersData.csv"*.
   c. Verify your configuration.



Next, you will run your Job and check the result in Hue.

Run the Job and verify the results

1. RUN THE JOB AND VERIFY THE RESULTS IN THE CONSOLE
   Run your Job and follow the execution in the **Console**. At the end of execution, you should have an exit code equal to 0 and several other messages that prove that the Job successfully executed:

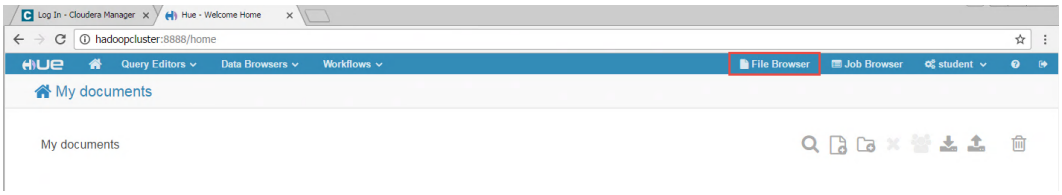


Note: This is not the default Console output for a Standard Job. The BDBasics project has been configured with a Log4j level set to INFO. This allows to have more details about the execution of your Job. For example, you will get the name of the files copied and the number of files uploaded to HDFS.

2. CONNECT TO HUE AND VERIFY THE RESULTS
Check your results using the File Browser of Hue.

   a. In your web browser, the page with Hue should be already opened. Otherwise, navigate to *http://hadoopcluster:8888* and login with student/training.

   b. Click **File Browser**:

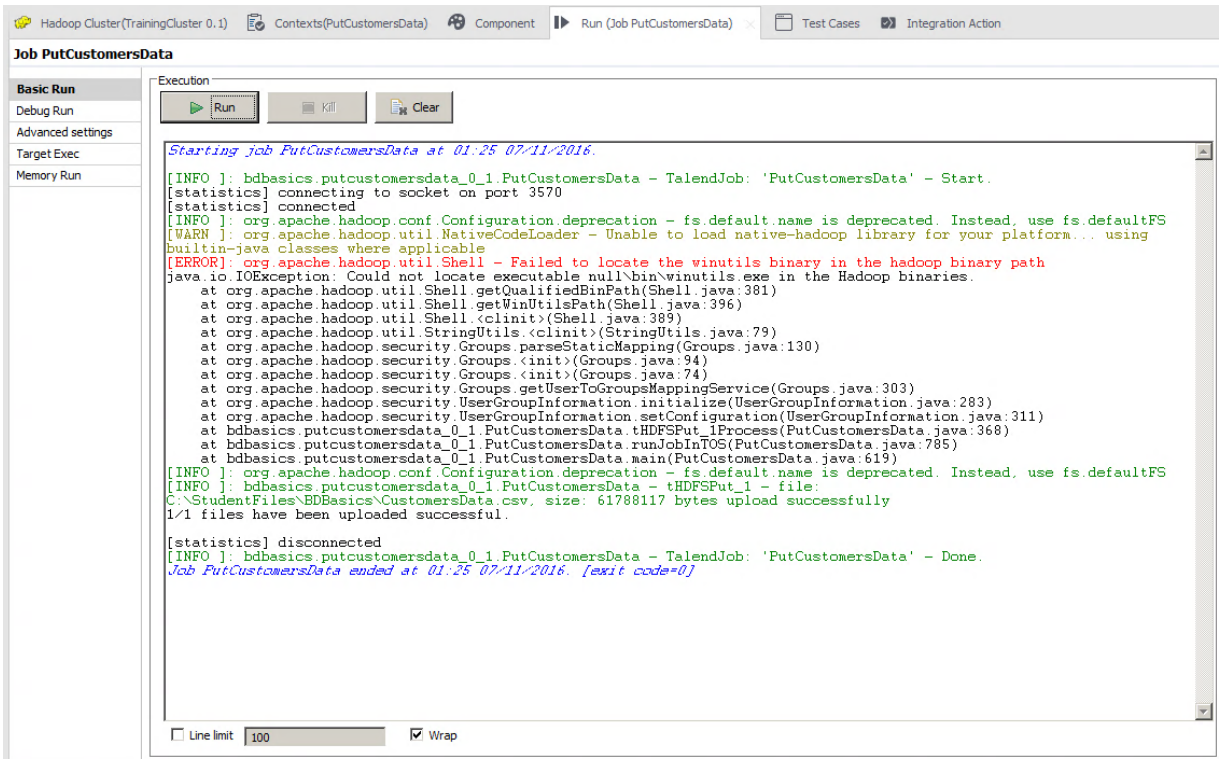


c. Navigate to */user/student/BDBasics/Customers/CustomersData.csv*:

This shows you the content of the CustomersData.csv file. The file was created by your Job in the new directory BDBasics/Customers that you specified.

Examining the Console output - Winutils.exe error

Take a few minutes to read the output in the Console. At the end of the execution of the PutCustomersData Job, you should have the following text in the Console:



In the Console, you will get logs created by your cluster. By default, only logs with WARN or ERROR status are displayed.

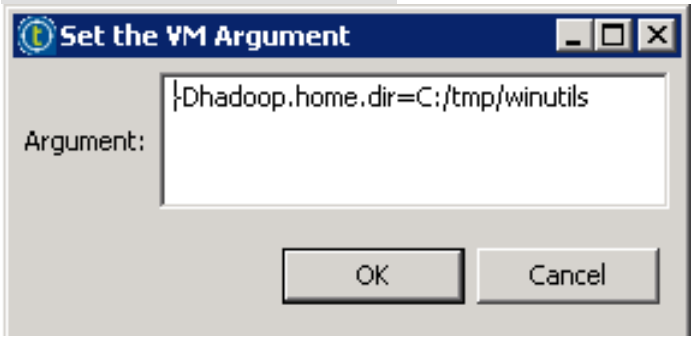
You may have noticed that there is an error displayed in the Console:

```
[WARN ]: org.apache.hadoop.util.NativeCodeLoader - Unable to load native-hadoop
library for your platform... using builtin-java classes where applicable
[ERROR]: org.apache.hadoop.util.Shell - Failed to locate the winutils binary in the
hadoop binary path
java.io.IOException: Could not locate executable null\bin\winutils.exe in the
Hadoop binaries.
    at org.apache.hadoop.util.Shell.getQualifiedBinPath(Shell.java:355)
    at org.apache.hadoop.util.Shell.getWinUtilsPath(Shell.java:370)
    at org.apache.hadoop.util.Shell.<clinit>(Shell.java:363)
    at org.apache.hadoop.util.StringUtils.<clinit>(StringUtils.java:79)
    at org.apache.hadoop.security.Groups.parseStaticMapping(Groups.java:104)
    at org.apache.hadoop.security.Groups.<init>(Groups.java:86)
    at org.apache.hadoop.security.Groups.<init>(Groups.java:66)
    at
org.apache.hadoop.security.Groups.getUserToGroupsMappingService(Groups.java:280)
    at
org.apache.hadoop.security.UserGroupInformation.initialize(UserGroupInformation.jav
a:283)
    at
org.apache.hadoop.security.UserGroupInformation.ensureInitialized(UserGroupInformat
ion.java:260)
```

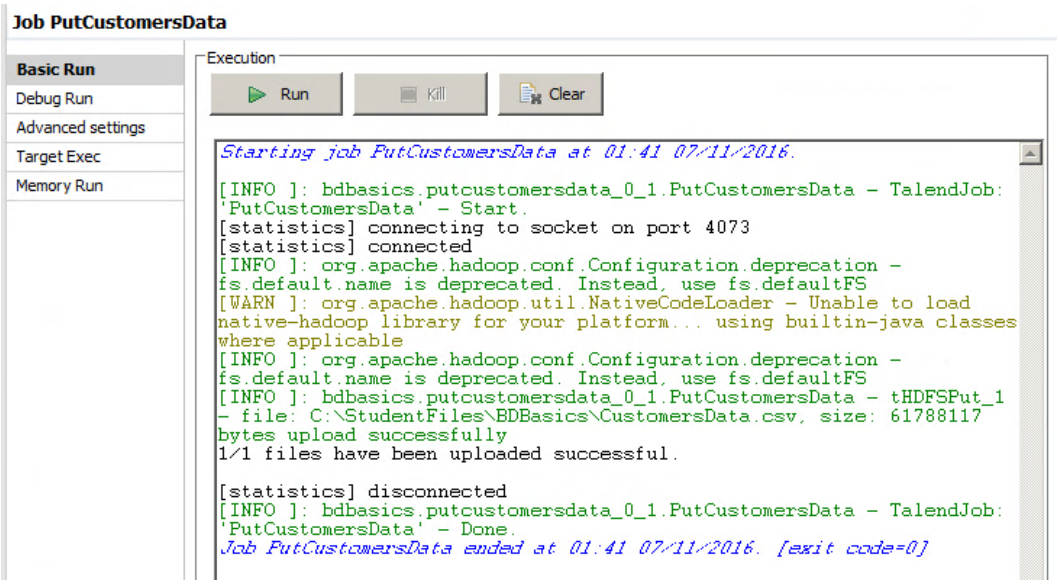
You will see an error message regarding the winutils.exe binary not being available in the Hadoop binaries. In the current Job, this error won't prevent the Job to succeed but it could in some cases.
You can fix this issue by downloading the winutils.exe file and then saving it in a tmp folder. This has been done for you in your training environment.

1. FIX THE ERROR
   Using the Advanced settings tab, set the Hadoop home directory to the location of the winutils.exe file.
   a. In the **Run** view, click **Advanced settings**.
   b. Select the **Use specific JVM arguments** option.
   c. Click **New...**.

d. In the **Argument** box, enter:
`-Dhadoop.home.dir=C:/tmp/winutils`



e. Click **OK** to save the new argument and run your Job again.
The Job should run successfully without error messages:



For the next Jobs built in this training, you can add a JVM argument to set the Hadoop home directory to C:\tmp\winutils, or you can safely ignore this error.

Next, you will put multiple files to HDFS.

Storing Multiple files on HDFS

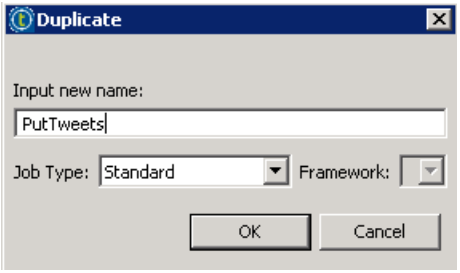
Task outline

Your next step is to store a set of files on HDFS. In this scenario, you are saving a series of Twitter message files for later analysis.

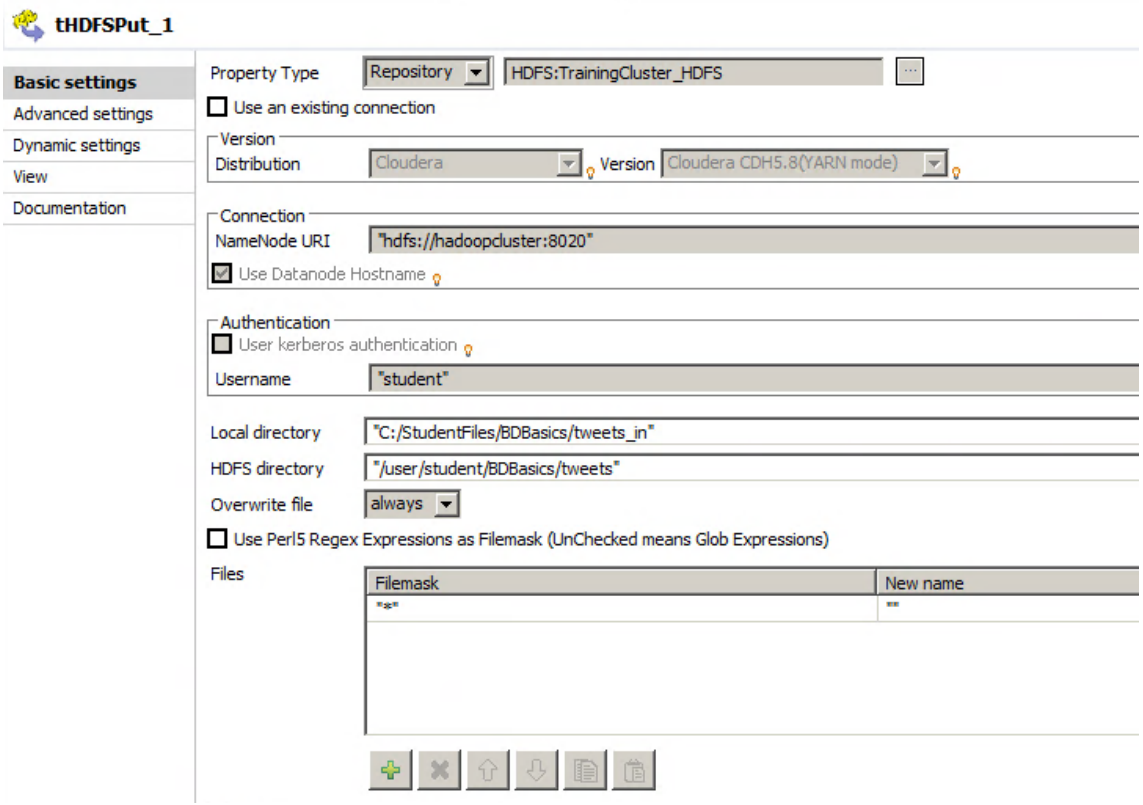
Create the Job

Under the C:\StudentFiles\BDBasics\tweets_in folder, you will find tweets saved as text files. You will put these files on HDFS.

1. DUPLICATE THE JOB
   Duplicate the **PutCustomersData** Job and name the new Job *PutTweets*.
   a. Right-click **PutCustomersData** Job in the Repository, then click **Duplicate**.
   b. In the **Input new name** box, enter *PutTweets* and in the **Job Type** list, select *Standard*:



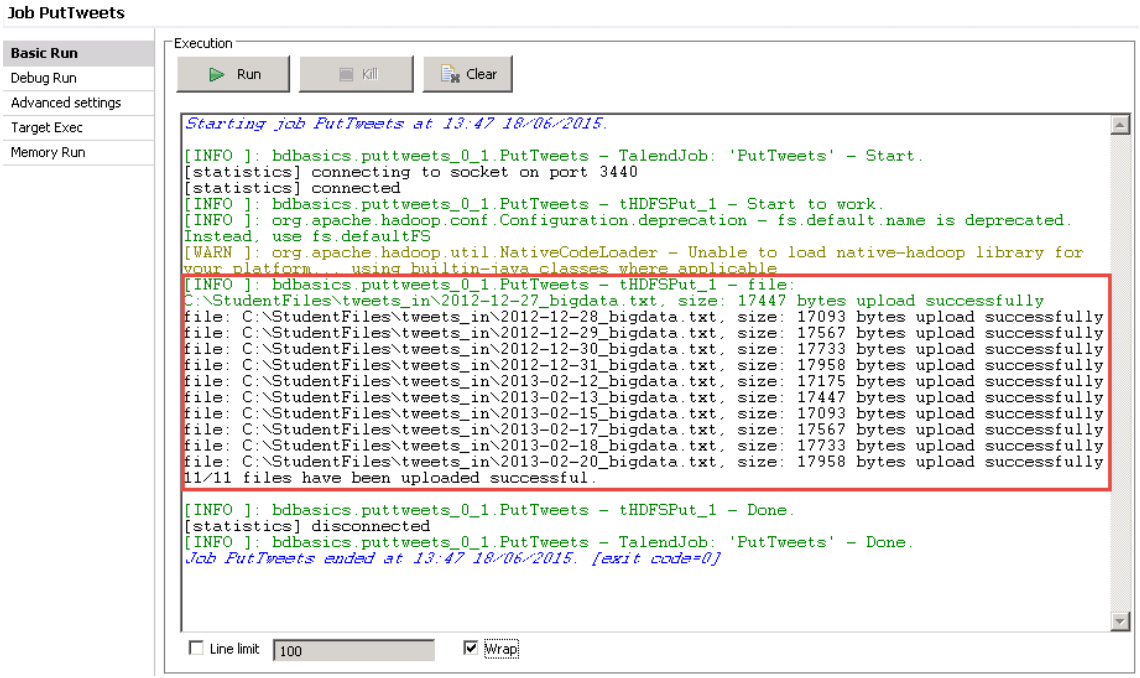
   c. Click **OK** to duplicate the Job. Then, open *PutTweets*.
2. UPDATE THE LOCAL DIRECTORY
   Configure **tHDFSPut** to read from the *C:/StudentFiles/BDBasics/tweets_in* folder in your local file system.
   a. Double-click **tHDFSPut** to open the **Component** view.
   b. In the **Local directory**, locate *C:/StudentFiles/BDBasics/tweets_in*.
   c. In the **Filemask** column, enter *"*"*.
      This means to select all the files under C:\StudenFiles\BDBasics\tweets_in.
3. UPDATE THE TARGET DIRECTORY
   In the **HDFS directory** box, enter */user/student/BDBasics/tweets*.



Next you will run your Job and check the result in Hue.

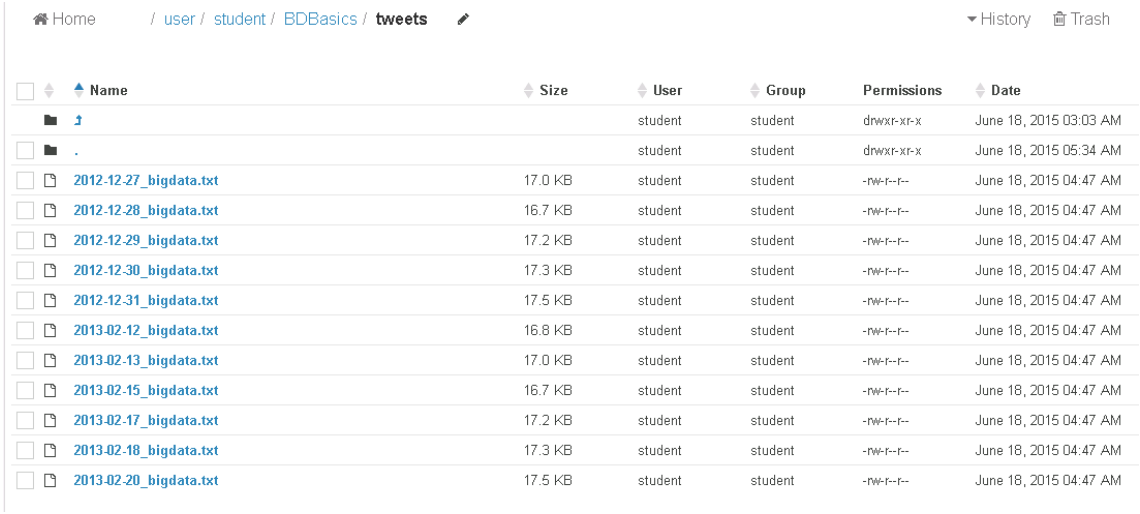
Run

1. RUN YOUR JOB AND VERIFY THE RESULTS**Run** your Job and follow the execution in the **Console**. At the end of execution, you should have an exit code equal to 0 and several other messages that prove that the Job executed successfully:



2. CONNECT TO Hue
   In your web browser, the page with Hue should already be opened. Otherwise, navigate to *http://hadoopcluster:8888* and login with *student/training*.

3. VERIFY THE RESULTS
   Click **File Browser** and navigate to */user/student/BDBasics/tweets*:



Name	Size	User	Group	Permissions	Date
..		student	student	drwxr-xr-x	June 18, 2015 03:03 AM
.		student	student	drwxr-xr-x	June 18, 2015 05:34 AM
2012-12-27_bigdata.txt	17.0 KB	student	student	-rw-r--r--	June 18, 2015 04:47 AM
2012-12-28_bigdata.txt	16.7 KB	student	student	-rw-r--r--	June 18, 2015 04:47 AM
2012-12-29_bigdata.txt	17.2 KB	student	student	-rw-r--r--	June 18, 2015 04:47 AM
2012-12-30_bigdata.txt	17.3 KB	student	student	-rw-r--r--	June 18, 2015 04:47 AM
2012-12-31_bigdata.txt	17.5 KB	student	student	-rw-r--r--	June 18, 2015 04:47 AM
2013-02-12_bigdata.txt	16.8 KB	student	student	-rw-r--r--	June 18, 2015 04:47 AM
2013-02-13_bigdata.txt	17.0 KB	student	student	-rw-r--r--	June 18, 2015 04:47 AM
2013-02-15_bigdata.txt	16.7 KB	student	student	-rw-r--r--	June 18, 2015 04:47 AM
2013-02-17_bigdata.txt	17.2 KB	student	student	-rw-r--r--	June 18, 2015 04:47 AM
2013-02-18_bigdata.txt	17.3 KB	student	student	-rw-r--r--	June 18, 2015 04:47 AM
2013-02-20_bigdata.txt	17.5 KB	student	student	-rw-r--r--	June 18, 2015 04:47 AM

   There should be eleven files, because there are eleven tweet files in your local directory. The **tHDFSPut** component used HDFS to write all of the files in your local directory into the Hadoop file system.
   You used a file mask to write a file set. File masks give you considerable control over what files you want Talend components to operate on.

Now that some data exists on HDFS, you can create a Job to read it.

Reading Data from HDFS

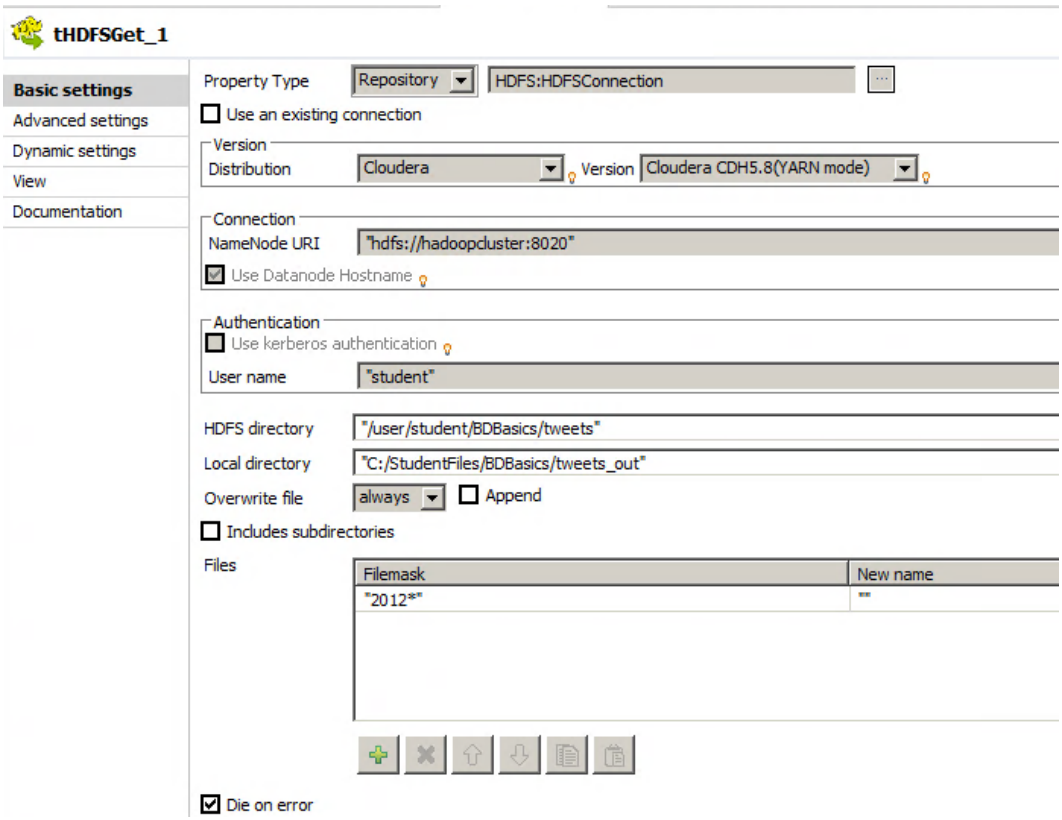
Task outline

Now you will create a Job to read HDFS data. You are going to transfer a subset of the Twitter files from HDFS to your local file system.

Create the Job

If you examine the files in the tweets folder that you just created on HDFS, you will notice that the files were produced in 2012 and 2013. You will read only the 2012 files.

1. CREATE A NEW STANDARD JOB
   Create a new Job, naming it *GetTweets*.
2. ADD A tHDFSGet COMPONENT
   Place a **tHDFSGet** component on the design workspace.
   You use **tHDFSGet** to read files stored on HDFS and make copies of them on your local system.
3. CONFIGURE tHDFSGet TO USE THE HDFSConnection METADATA
   Use the **HDFSConnection** metadata previously created.
   a. Double-click **tHDFSGet** to open the **Component** view.
   b. In the **Property Type** list, select **Repository**.
   c. Click **(...)** , then locate the *HDFSConnection* metadata that you previously created.
4. CONFIGURE THE HDFS DIRECTORY
   To configure the **HDFS directory**, click **(...)** , then navigate to */user/student/BDBasics/tweets*.
5. CONFIGURE THE LOCAL DIRECTORY
   Configure tHDFSGet to overwrite the **C:/StudentFiles/BDBasics/tweets_out** folder.
   a. In the **Local directory**, enter *"C:/StudentFiles/BDBasics/tweets_out"*.
   b. In the **Overwrite file** list, select *always*. This tells the component to always replace existing files when a new file of the same name is being saved.
6. ADD A CONDITION TO READ ALL THE 2012 TWEETS
   Configure the **Files** table to read the tweets corresponding to 2012.
   a. Click the **green plus sign** below the **Files** table.
   b. Replace *"newline"* with *"2012*"*.
   You are using a file mask to request all files in the HDFS directory **tweets** with names that begin with the string "2012":



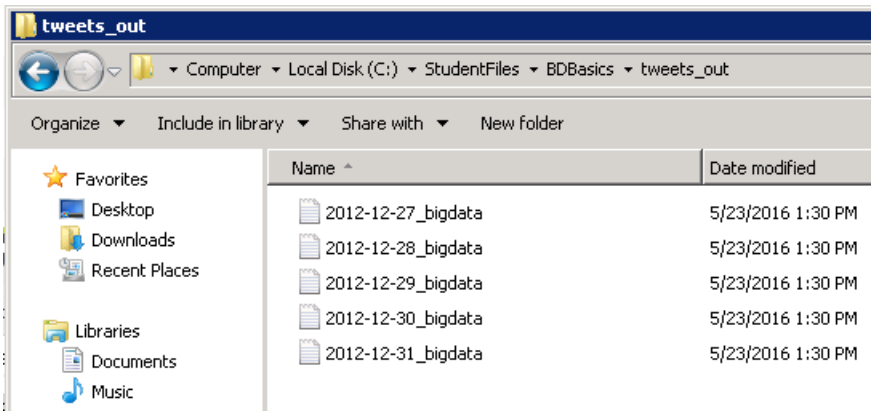
Now, you will run the Job and verify the results in your local file browser.

Run

1. RUN YOUR JOB AND VERIFY THE RESULTS IN THE CONSOLE
   Run your Job and follow the execution in the **Console**. At the end of execution, you should have an exit code equal to 0 and several other messages that prove that the Job successfully executed:

```
[INFO ]: org.apache.hadoop.conf.Configuration.deprecation - fs.default.name is deprecated.
Instead, use fs.defaultFS
[WARN ]: org.apache.hadoop.util.NativeCodeLoader - Unable to load native-hadoop library for your
platform... using builtin-java classes where applicable
[INFO ]: bdbasics.gettweets_0_1.GetTweets - tHDFSGet_1 - file:
hdfs://hadoopcluster:8020/user/student/BDBasics/tweets/2012-12-27_bigdata.txt, size: 17447 bytes
download successfully
file: hdfs://hadoopcluster:8020/user/student/BDBasics/tweets/2012-12-28_bigdata.txt, size: 17093
bytes download successfully
file: hdfs://hadoopcluster:8020/user/student/BDBasics/tweets/2012-12-29_bigdata.txt, size: 17567
bytes download successfully
file: hdfs://hadoopcluster:8020/user/student/BDBasics/tweets/2012-12-30_bigdata.txt, size: 17733
bytes download successfully
file: hdfs://hadoopcluster:8020/user/student/BDBasics/tweets/2012-12-31_bigdata.txt, size: 17958
bytes download successfully
5/5 files have been downloaded.
[statistics] disconnected
[INFO ]: bdbasics.gettweets_0_1.GetTweets - TalendJob: 'GetTweets' - Done.
Job GetTweets ended at 04:54 19/10/2016. [exit code=0]
```

2. VERIFY THE RESULTS IN YOUR LOCAL FILE BROWSER
   In a file browser, navigate to *C:\StudentFiles\BDBasics\tweets_out*:



There are five tweet files that your Job read from HDFS and put on your local system. Because a file mask was used to limit the files selected for the operation, only files with names that begin with "2012" were read from HDFS.

Now that you have worked with text files on HDFS, it is time to work with sparse dataset and experiment how to store them efficiently in HDFS.

Storing Sparse Dataset with HBase

Task outline

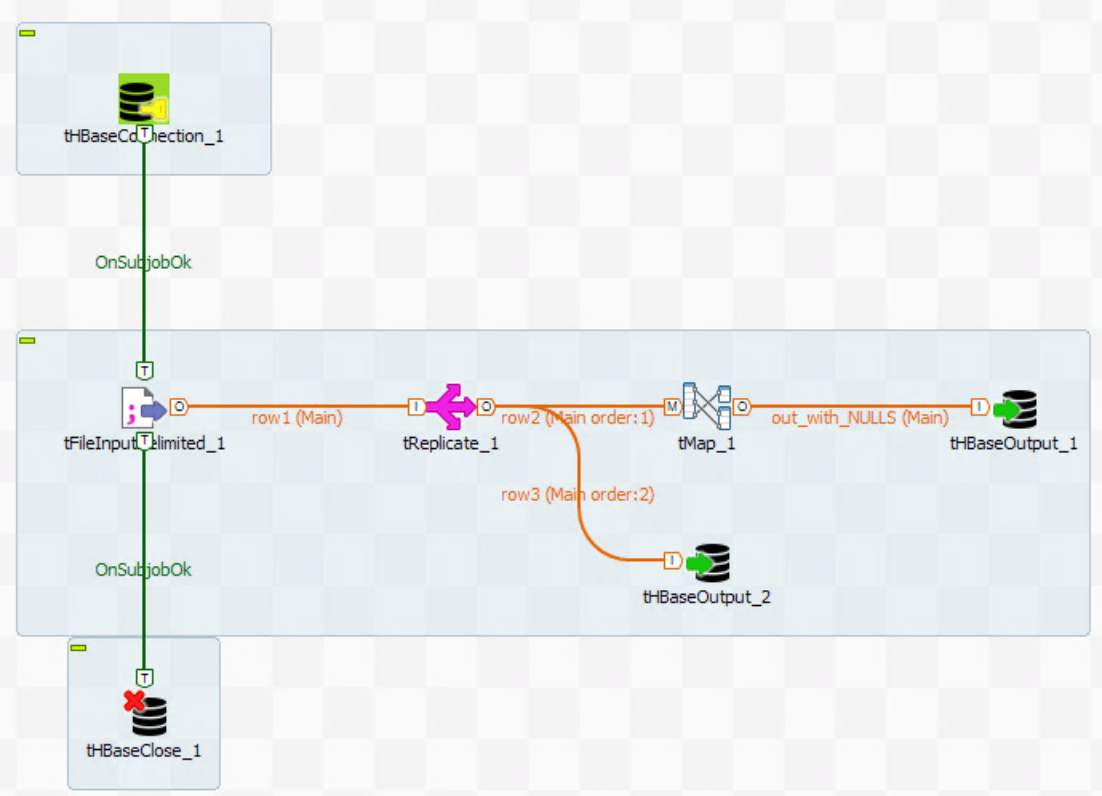
Apache HBase is the Hadoop column-oriented database. It is an open-source, non-relational database modeled after Google's BigTable, and provides the same functionality in top of Hadoop and HDFS.

HBase is useful when you need random, real-time read/write access to your Big Data. Furthermore, HBase can host very large tables -- billions of rows X millions of columns-- and is particularly well suited for sparse data sets.

If your relational table looks like the table below (data missing in columns), it is considered "sparse" and is a good candidate for HBase.

	Col A	Col B	Col C	Col D	Col E
Row 01	Val1A				
Row 02	Val2A	Val2B	Val2C	Val2D	Val2E
Row 03	Val3A		Val3C		Val3E

In this lab, you will generate sparse data and build a Job to store them to HDFS with HBase:

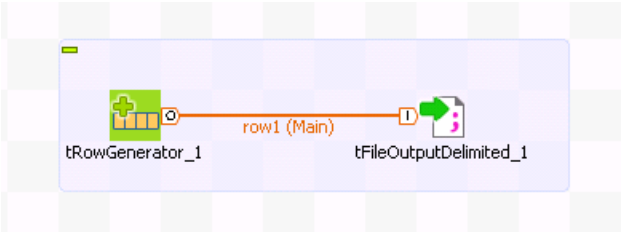


Generate sparse data

First, you will open a Job that will generate a sparse data set.

It will create a dataset that represents the number of connections per month to a website by customers identified by their Id.

1. IMPORT THE GenerateSparseData JOB
   From the JobDesigns.zip archive, import the GenerateSparseData Job.
   a. Under the **C:\StudentFiles\BDBasics** folder, you will find the **JobDesigns.zip** archive file.
   b. Import the Job named *GenerateSparseData*:



   This Job is composed of two components. The **tRowGenerator** component will generate an integer value for the customers ID, and random integer values to simulate the number of connections per month.
   The second component will save the data in a file named *HBase_sample.txt* under the C:\StudentFiles\BDBasics folder.

2. RUN THE JOB AND VERIFY THE RESULTS
   Run the Job, then locate and open *C:\StudentFiles\BDBasics\HBase_sample.txt* with **Notepad++**.

Id	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
100000	1	1	2	5	3	1	0	1	5	9	2	0
100001	0	1	2	2	3	1	0	0	4	2	2	2
100002	0	0	4	2	0	2	0	1	4	8	5	1
100003	0	0	4	4	5	0	2	0	4	6	5	0
100004	0	0	1	3	3	1	1	1	1	7	5	2
100005	1	1	2	5	5	0	2	0	5	6	2	1
100006	0	1	2	3	2	2	1	1	5	7	2	1
100007	0	0	4	3	1	1	2	0	3	0	0	1
100008	1	1	0	5	3	2	2	0	1	4	2	2
100009	0	0	5	4	1	2	0	1	1	5	0	1
100010	1	1	4	5	1	0	2	0	4	1	0	2
100011	0	1	5	1	5	2	0	0	2	9	4	2
100012	1	1	5	0	0	1	1	1	0	5	2	2
100013	1	0	1	4	2	0	1	1	2	7	3	0
100014	1	0	4	1	5	0	1	0	1	4	1	1
100015	1	1	5	0	3	1	2	1	5	4	5	2
100016	0	0	0	1	5	1	2	1	2	2	1	2
100017	0	1	4	5	5	2	0	1	1	7	2	2
100018	0	0	2	3	1	0	0	1	0	3	1	2

   As expected, there are a lot of 0 values in the data.

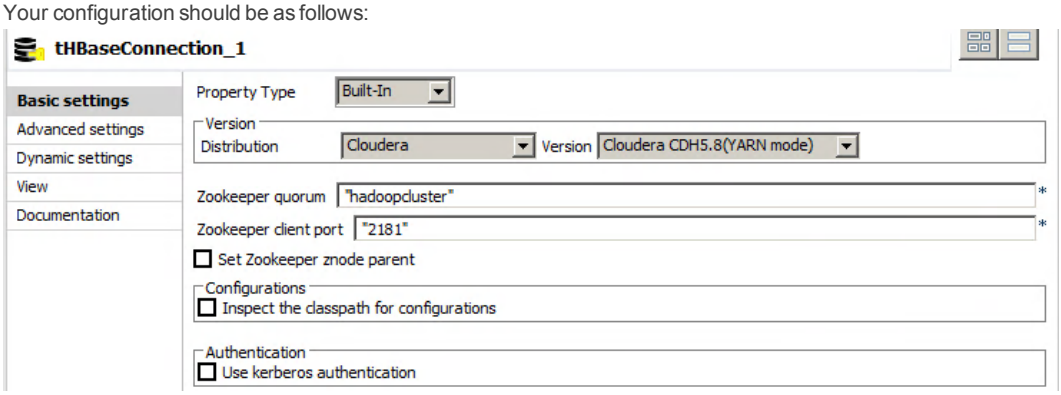
Next, you will configure the connection to HBase.

Configure HBase connection

An alternative way to configure a connection is to use a dedicated component. You will now create a new Job. The first step will be to configure the connection to HBase.

1. CREATE A NEW STANDARD JOB
   Create a new standard Job and name it *StoreSparseData*.
2. ADD A tHBaseConnection COMPONENT
   Add a **tHBaseConnection** component and open the **Component** view.

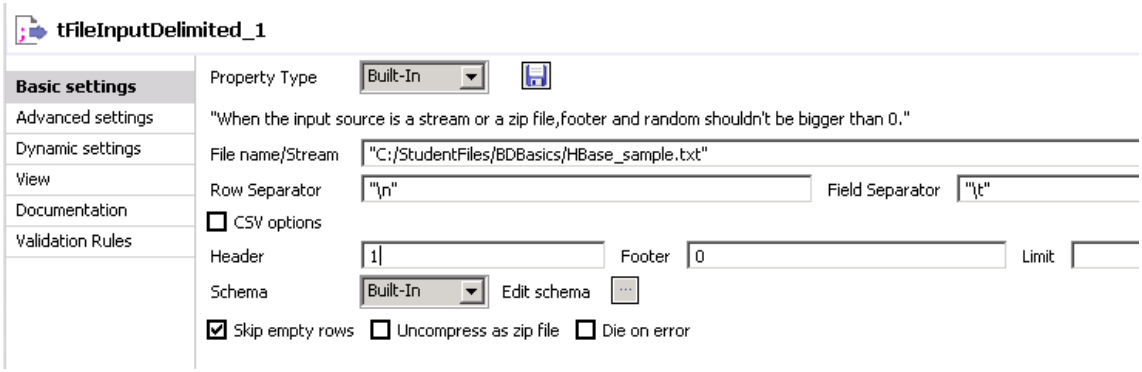
3. CONFIGURE THE DISTRIBUTION AND THE VERSION OF YOUR CLUSTER
   In the **Distribution** list, select *Cloudera* and in the **HBase version** list, select *Cloudera CDH5.8 (YARN mode)*.
4. CONFIGURE THE CONNECTION TO Zookeeper
   Configure the host name and the port of the Zookeeper service.
   a. In the **Zookeeper quorum** box, enter *"hadoopcluster"*.
   b. In the **Zookeeper client port** box, enter *"2181"*.
   Your configuration should be as follows:



Read sparse data

You will now continue to build the Job to store your data with HBase.

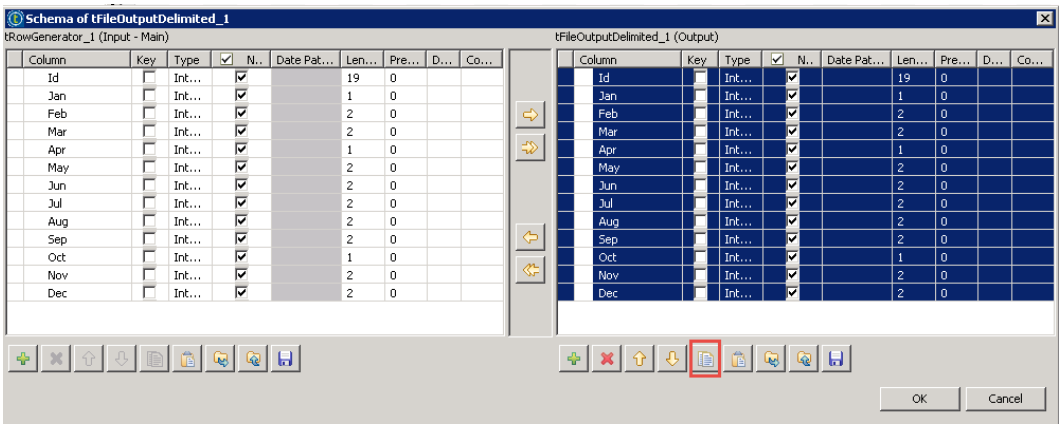
1. ADD A tFileInputDelimited COMPONENT
   Add a **tFileInputDelimited** component below **tHBaseConnection**.
2. CONNECT WITH A TRIGGER
   Connect **tHBaseConnection** to **tFileInputDelimited** with an **OnSubjobOk** trigger.
3. READ THE HBase_sample.txt FILE
   Configure **tFileInputDelimited** to read the *"C:/StudentFiles/BDBasics/HBase_sample.txt"* file, knowing that the **row separator** is *"\n"*, the **field separator** is *"\t"* and that there is *1* **Header row**.



4. CONFIGURE THE SCHEMA
   The schema of the HBase_sample.txt file can be found in the GenerateSparseData Job.

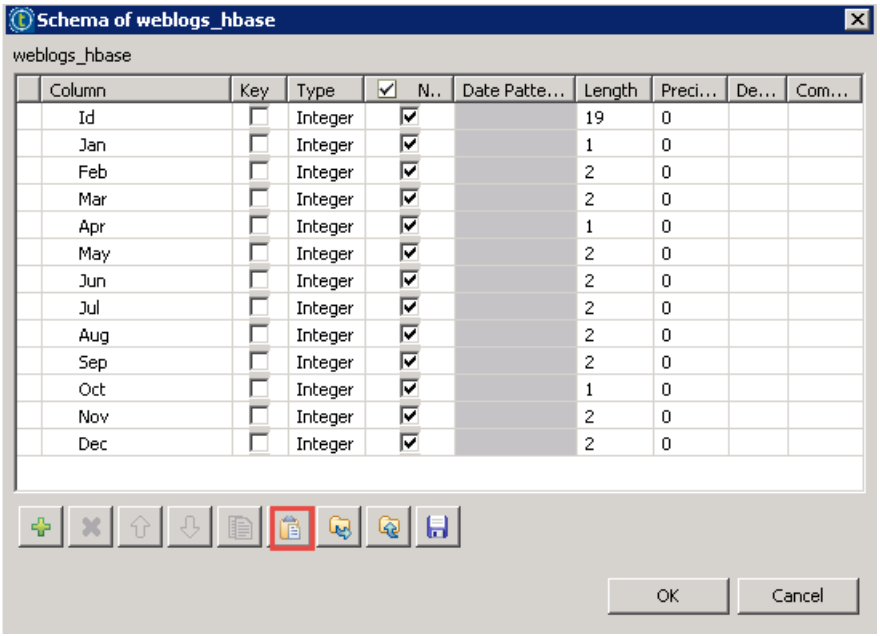
a. In the **GenerateSparseData** Job, double-click **tFileOutputDelimited** to open the **Component** view.

b. Edit the schema, select all the columns and copy the schema:



c. In the **StoreSparseData** Job, double-click **tFileInputDelimited** to open the **Component** view.

d. Edit the schema and paste the schema:



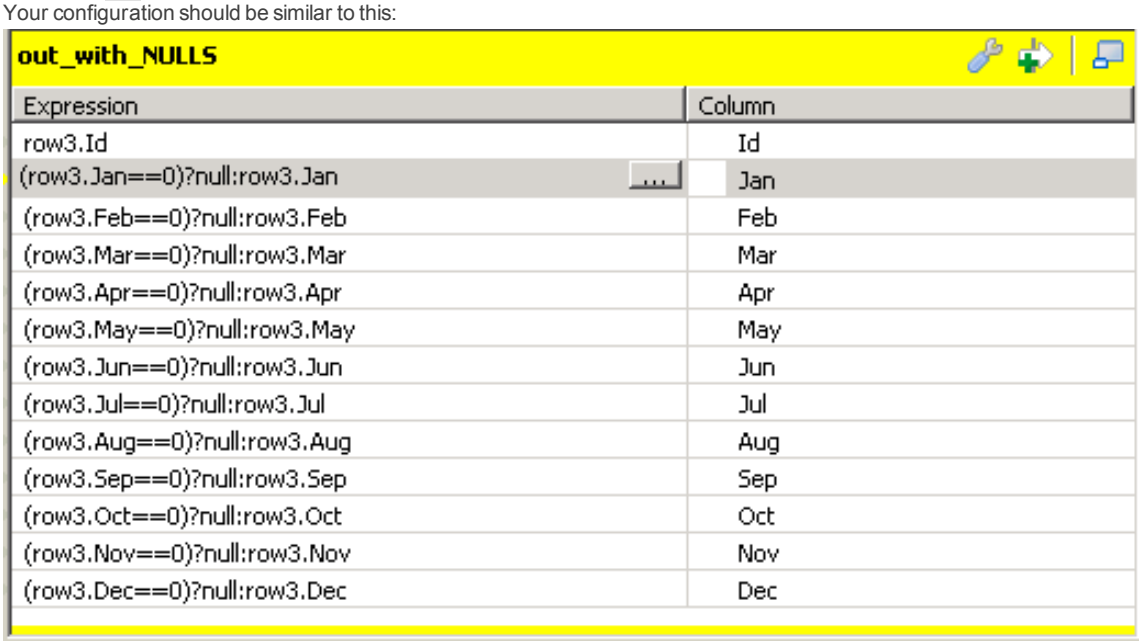
Handle Null values

HBase is well suited for sparse dataset because it does not persist Null values. In our current data set, there are a lot of zeros. You will process the data to find the zeros and replace them with Nulls.

You will store the raw data set in HBase, as well as the processed data.

1. ADD A tReplicate COMPONENT
   Add a **tReplicate** component at the right side of **tFileInputDelimited** and connect it with the **Main** row.
2. ADD A tMap COMPONENT
   Add a **tMap** component at the right side of **tReplicate** and connect it with the **Main** row.
3. OPEN THE MAPPING EDITOR
   Double-click to open the **tMap** editor.
4. CREATE A NEW OUTPUT TABLE
   Add an **output** and name it *out_with_NULLS*.

5. COPY ALL INPUT COLMUNS IN THE OUTPUT TABLE
   Select all the columns in the row table and drag to the **out_with_NULLS** table.
   Note: You may have a different row index.
6. CONFIGURE THE MAPPING TO REPLACE ALL 0 VALUES BY Null VALUES
   Edit the expression for each month and modify it as follows:
   `(row3.Jan==0)?null:row3.Jan`
   This means that all 0 values will be replaced by a null.
   Note: The `Jan` value must be replaced as needed to fit all months.
   Your configuration should be similar to this:



   Click **Ok**.

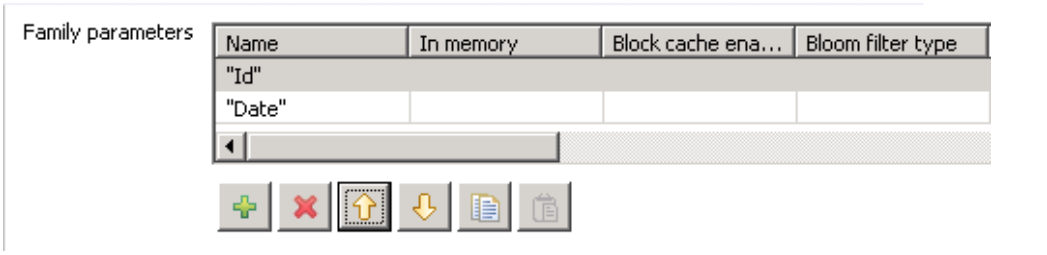
Save data to HBase

Now you will add components to save raw and processed data to HBase.

1. ADD A tHBaseOutput COMPONENT
   At the right side of **tMap**, add a **tHBaseOutput** component and connect it with the **out_with_NULLS** row.Then, open the **Component** view.
2. USE THE EXISTING CONNECTION TO HBASE
   Select **Use an existing connection** and select *tHBaseConnection_1* in the list.
3. Configure the schema
   Click **Sync columns**.
4. GIVE A NAME TO THE HBASE TABLE
   In the **Table name** box, enter *data_withNulls*.
5. SELECT THE APPROPRIATE ACTION
   In the **Action on table** list, select *Drop table if exists and create*.
6. CREATE THE COLUMN FAMILY NAMES
   To create HBase tables, a family name must be associated with each column.

a. To create the family names, click **Advanced settings**.

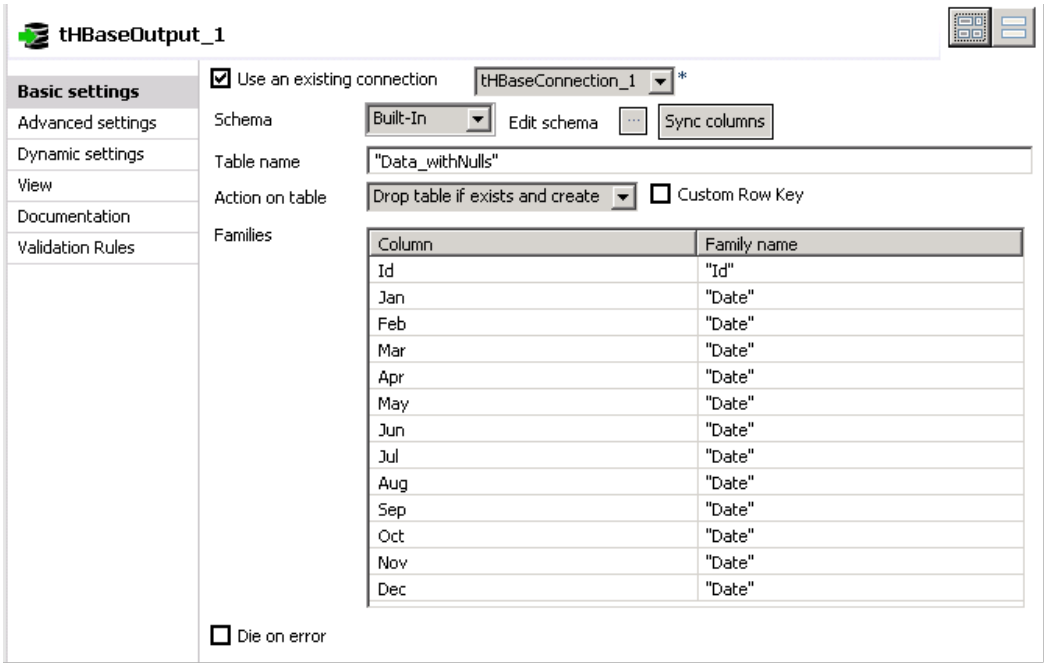
b. In the **Family parameters** table, add 2 lines as follows:



7. CONFIGURE THE Families TABLE
The Id family name should be used for the Id column and the Date family name for all other columns.

a. In the **Basic settings** tab, set the Id column Family Name to *"Id"*.

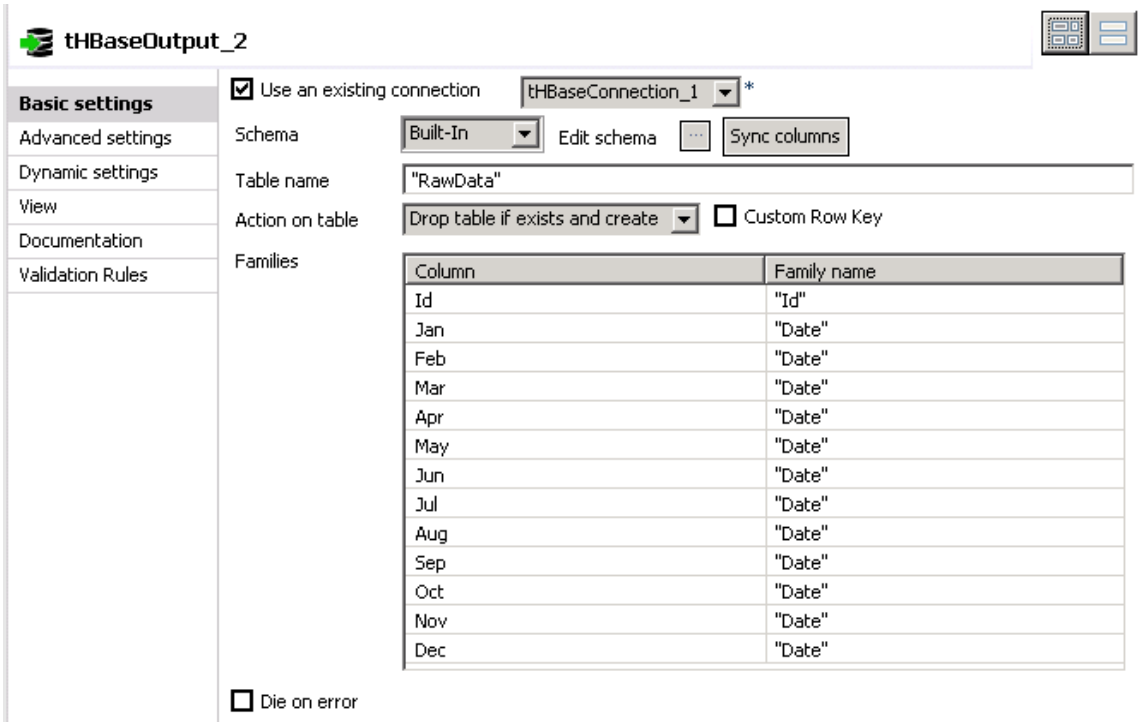
b. Set the other column's Family Name to *"Date"*.
Your configuration should be as follows:



8. ADD ANOTHER tHBaseOutput COMPONENT AND CONFIGURE IT AS THE FIRST ONE
Copy **tHBaseOutput_1** and paste below the **tMap** component.

9. CONNECT IT TO tReplicateE
Connect **tHBaseOutput_2** to **tReplicate** with a **Main** row.

10. NAME THE TABLE RawData
    Open the **Component** view of **tHBaseOutput_2** and change the **Table name** to *"RawData"*:

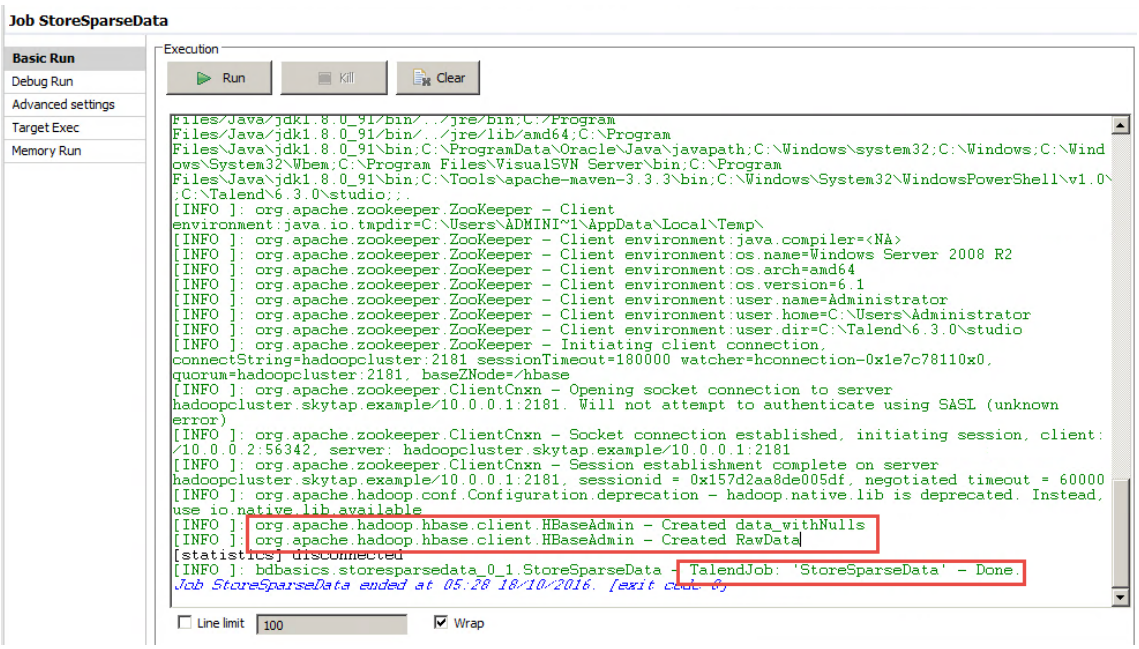


11. ADD A tHBaseClose COMPONENT AND CONNECT IT
    Add a **tHBaseClose** component below the **tFileInputDelimited** component and connect it with an **OnSubjobOk** trigger.

Now you can run your Job and check the results in Hue.

Run the Job and verify the results in Hue

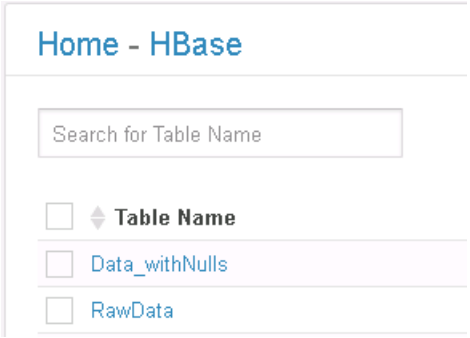
1. RUN YOUR JOB AND VERIFY THE RESULT IN THE CONSOLE
   Run your Job and check the results of the execution in the **Console**.



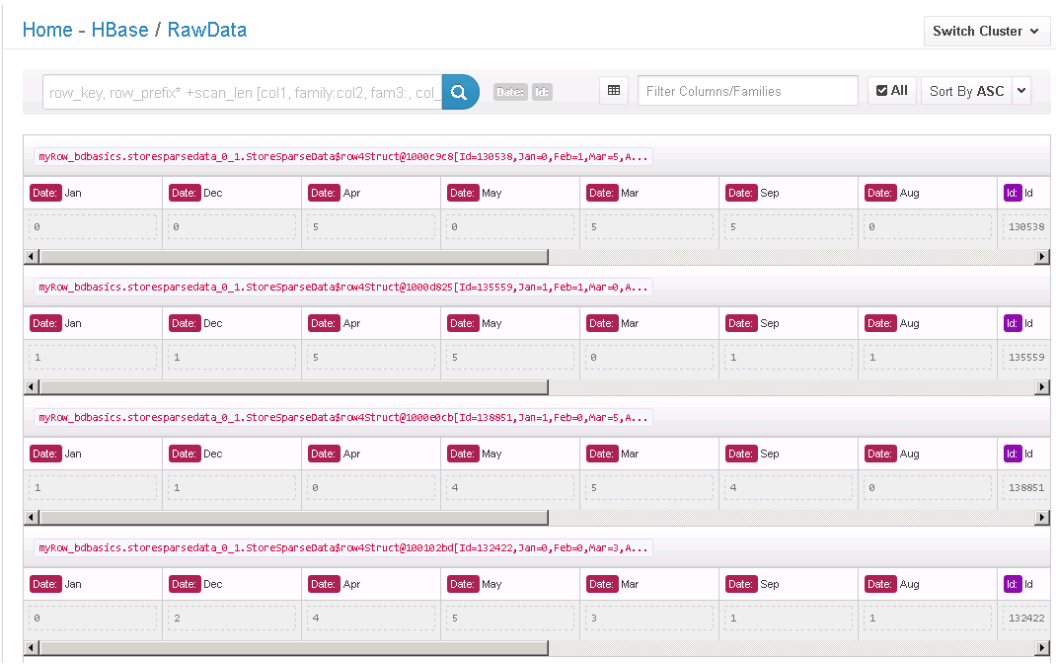
Your execution should be successful. Otherwise, double check your Job and check HBase health in the Cloudera Manager.

2. VERIFY THE RESULTS IN Hue
   Connect to Hue to check your results.

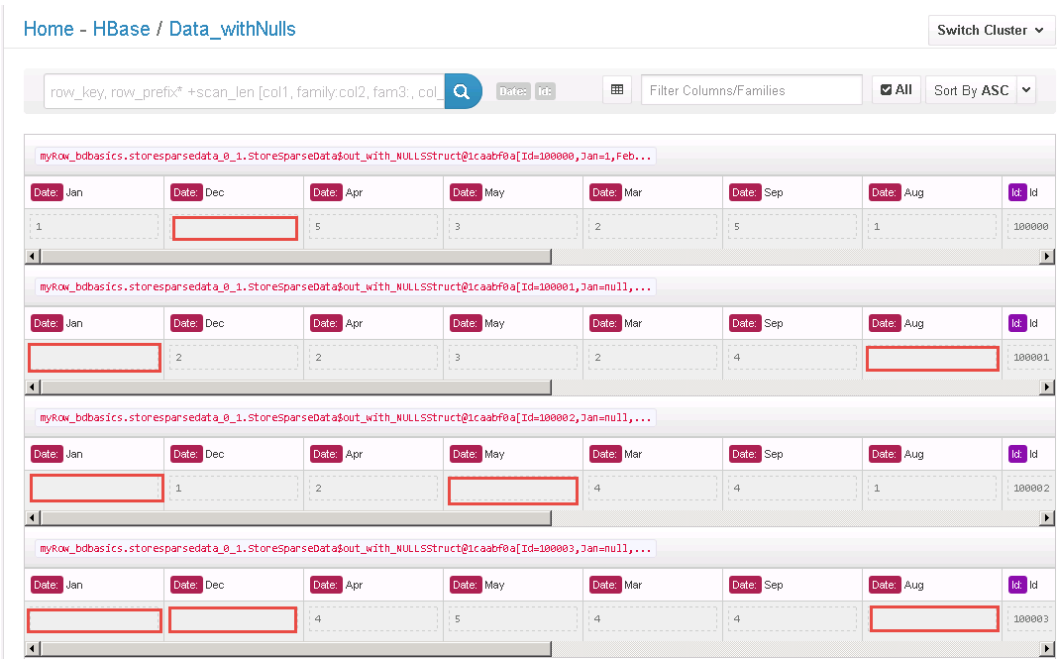
   a. Connect to **Hue** and click **Data Browsers>HBase**.
      This will give you the list of HBase tables. You should see Data_withNulls and RawData in the list:



b. Click **RawData**:



This is an extract of the RawData table. You will still find zeros, but if you compare with the content of **Data_withNulls**:



The Null values are not stored.

Now that you have used HBase to store sparse data, it's time to move to the Challenge to test your knowledge.

Challenges

Task outline

Complete these exercises to further reinforce the skills you learned in the previous lesson. See Solutions for possible solutions to the exercises.

Add Support File

Develop a Job to write an XML file of support requests stored locally to HDFS. Configure the input and output targets as follows:

- Source file: *C:/StudentFiles/BDBasics/support/support.xml*
- Target file: */user/student/BDBasics/support/support.xml*

Double Up Orders

Develop a second Job to write a file of duplicated orders to HDFS. Use the local file *C:/StudentFiles/BDBasics/duplicated_orders* as the source file. Put the file into the HDFS directory */user/student/BDBasics/erp*, keeping the same file name. Use the schema stored in the file *orders.xml* and sort the file on column *id* before saving it.

Hint: use **tSortRow** to sort the file.

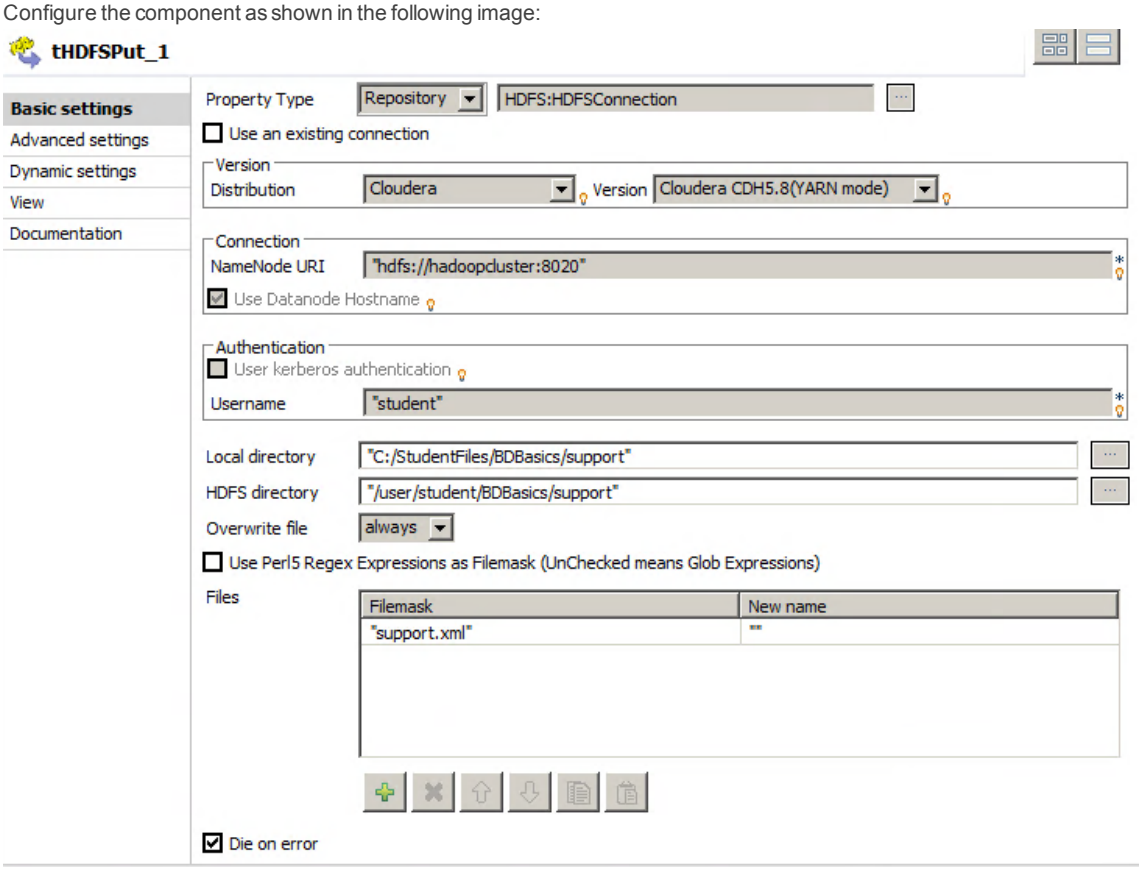
Solutions

Suggested solutions

These are possible solutions to the exercises. Note that your solutions may differ and still be acceptable.

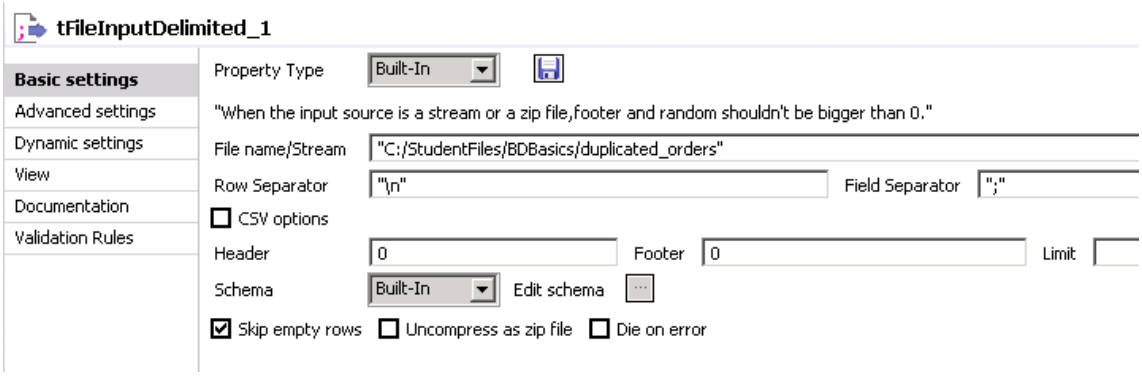
Put Support file

1. Use a **tHDFSPut** component to create the HDFS folder and write the target file to it.
2. Configure the component as shown in the following image:

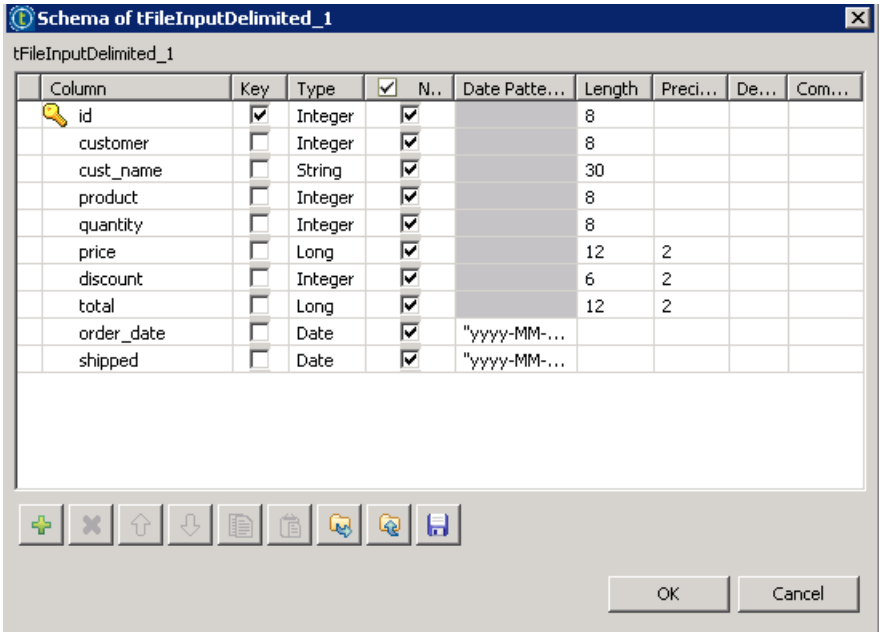


Double-up orders

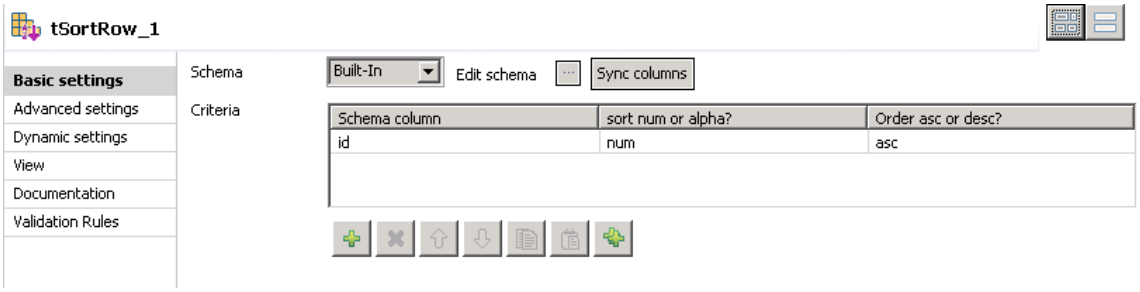
1. Use a **tFileInputDelimited** to read the file *duplicated_orders* from the local file system:



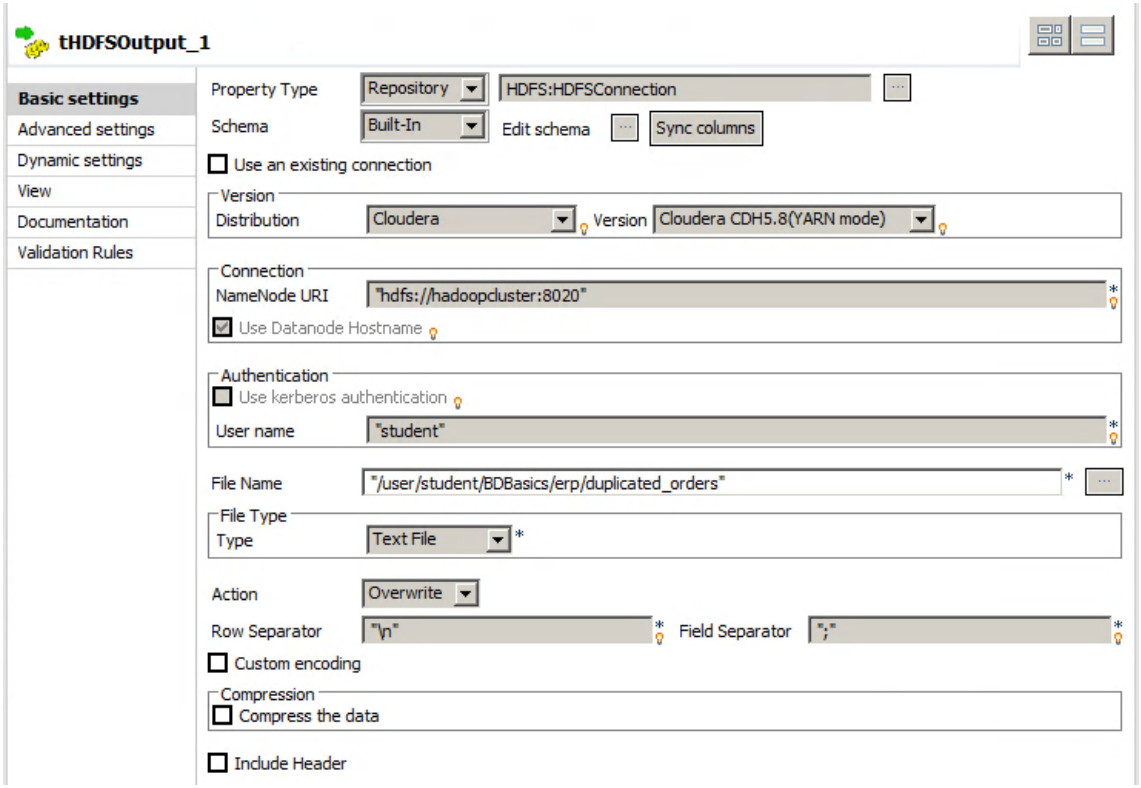
2. Import the schema from the file *orders.xml*:



3. Sort the data in ascending order by the column *id* by using the **tSortRow** component :



4. Use a **tHDFSOutput** component to rewrite the data to HDFS:



5. Click *Overwrite* in the **Action** list.
6. Run your Job and check the results in Hue.

Next step

You have almost finished this section. Time for a quick review.

Review

Recap

In this lesson, you learned the basics of writing files using Talend's Big Data components for Hadoop Distributed File System, HDFS.

You used **tHDFSPut** to write different types of files to HDFS. Then, you used **tHDFSGet** to read a subset of the files back from HDFS.

Last, you used HBase dedicated components to store sparse data on HDFS (**tHBaseConnection**, **tHBaseOutput**, **tHBaseClose**).

Further Reading

For more information about topics covered in this lesson, see the *Talend Data Integration Studio User Guide*, *Talend Big Data Platform User Guide* and the *Talend Components Reference Guide*.

LESSON 4

Working with Tables

This chapter discusses:

Concepts

Outline

- Lesson objectives
- Importing tables with Sqoop
- Creating tables in HDFS with Hive
- Using the Hive table creation wizard
- Lab overview
- Wrap-up

Lesson objectives

After completing this lesson, you will be able to:

- Transfer MySQL tables to HDFS using Sqoop
- Create Hive connection metadata
- Save data as Hive tables

Importing tables with Sqoop

- **Sqoop** is a tool that **transfers data between Hadoop and relational databases**
- You can use Sqoop to import data from a relational database management system (RDBMS), such as MySQL or Oracle, into the Hadoop Distributed File System (HDFS), transform it using MapReduce, and then export it back into an RDBMS
- Sqoop automates most of this process, relying on the database to describe the schema for the data to be imported
- Sqoop creates and runs a **map-only MapReduce Job** to import the data

Importing tables with Sqoop

Import tables

To import a MySQL table into HDFS, you will use a tSqoopImport component.In the tSqoopImport component, the first option is to choose the mode, **CommandLine** or **Java API**:

- If you choose **CommandLine,** the **Sqoop shell** is used to call Sqoop. In this mode, you must **deploy** and **run** the Job **in the host where Sqoop is installed.** This means you must install and use the **Jobserver**.
- If you choose **Use Java API**, the Java API is used to call Sqoop. In this mode, the **Job can be run locally in the Studio,** but you must configure the connection to your cluster.

Creating tables in HDFS with Hive

Hive is a **data warehouse** infrastructure tool for **processing structured data** in Hadoop. It resides on top of Hadoop to summarize Big Data, and it makes querying and analyzing easy.

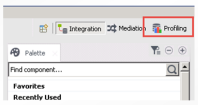
Hive was initially developed by Facebook and then by the Apache Software Foundation.

Hive:

- Stores **schema** in a **database**
- Processes **data** into **HDFS**
- Provides an **SQL-type language** for querying called **HiveQL**
- Is familiar, fast, and scalable

Using the Hive table creation wizard

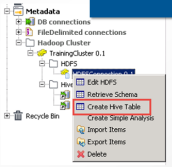
Switch to the Profiling perspective:



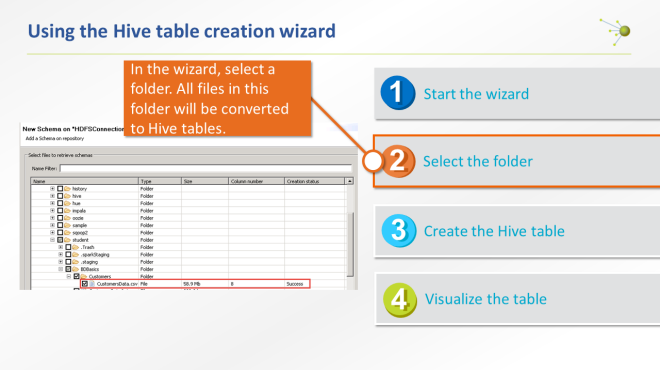
1. Start the wizard
2. Select the folder
3. Create the Hive table
4. Visualize the table

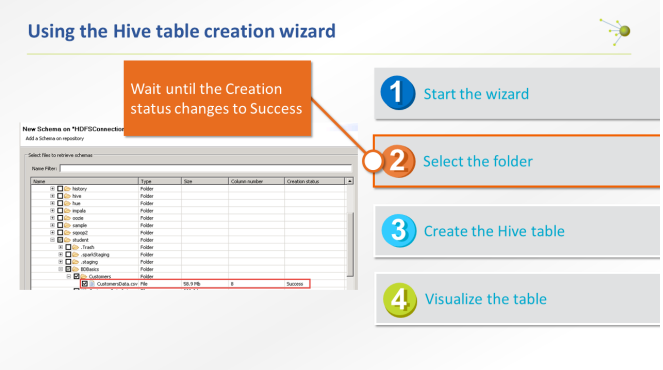
Using the Hive table creation wizard

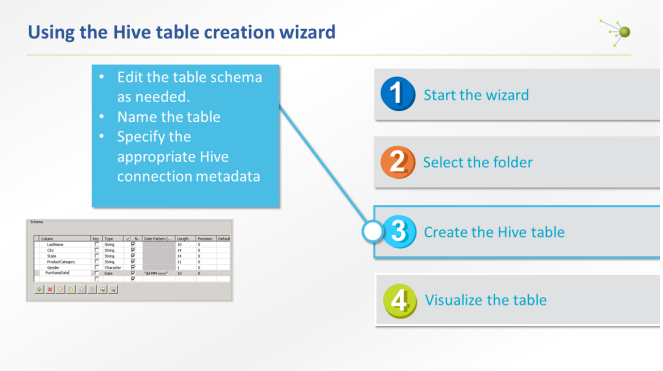
In DQ Repository, right-click the HDFS Connection metadata and click Create Hive Table

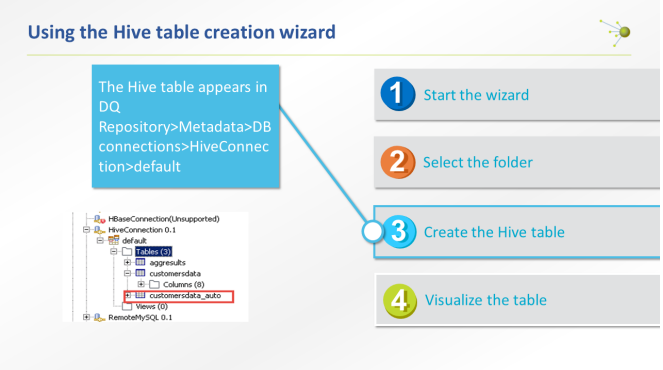


1. Start the wizard
2. Select the folder
3. Create the Hive table
4. Visualize the table

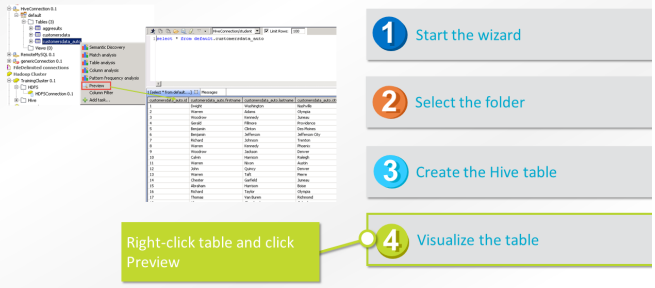

Using the Hive table creation wizard
In the wizard, select a folder. All files in this folder will be converted to Hive tables.
1 Start the wizard
2 Select the folder
3 Create the Hive table
4 Visualize the table


Using the Hive table creation wizard
Wait until the Creation status changes to Success
1 Start the wizard
2 Select the folder
3 Create the Hive table
4 Visualize the table


Using the Hive table creation wizard
Edit the table schema as needed.
Name the table
Specify the appropriate Hive connection metadata
1 Start the wizard
2 Select the folder
3 Create the Hive table
4 Visualize the table


Using the Hive table creation wizard
The Hive table appears in DQ Repository>Metadata>DB connections>HiveConnection>default
HBaseConnection(Unsupported)
HiveConnection 0.1
default
Tables (3)
aggresults
customersdata
Columns (8)
customersdata_auto
Views (0)
RemoteMySQL 0.1
1 Start the wizard
2 Select the folder
3 Create the Hive table
4 Visualize the table

Using the Hive table creation wizard



Lab overview

Importing tables with Sqoop

In this lab, you will transfer MySQL tables into HDFS using Sqoop-dedicated components

- Prepare the MySQL database
- Create database generic metadata
- Import tables
- Run the Job and check the results

Lab overview

Creating tables in HDFS with Hive

In this lab, you will:

- Create Hive connection metadata
- Manually create a Hive table
- Create a Hive table using the Hive table creation wizard

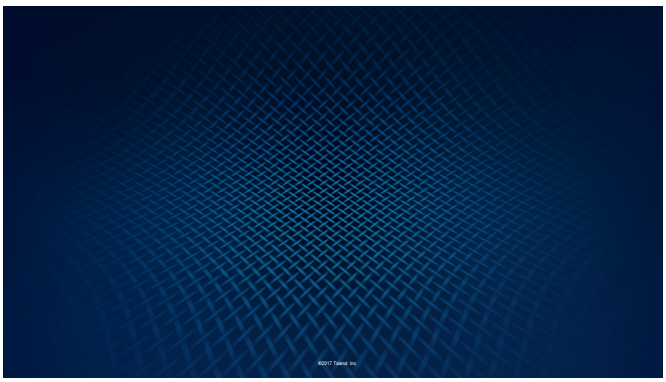
Wrap-up

Importing tables with Sqoop

- Sqoop uses Map-only MapReduce jobs to transfer data between RDBMS and HDFS

Creating tables in HDFS with Hive

- Hive is a data warehouse infrastructure tool for processing structured data in Hadoop
- Manual creation
- Using the Hive table creation wizard


©2017 Talend Inc.

Working With Tables

Overview

In this lesson, you will cover two common use cases.

HDFS can be used to for data warehouse optimization. So, you could decide to move your data from a relational database to HDFS. The first use case will show you how to transfer MySQL tables to HDFS using Sqoop.

The second use case will show you how to create a table using Hive. Then, this table can be processed using Hive QL, which is very similar to SQL.

Objectives

After completing this lesson, you will be able to:

- Transfer MySQL tables to HDFS using Sqoop
- Create Hive connection Metadata
- Save data as Hive tables

Before you begin

Be sure that you are in a working environment that contains the following:

- A properly installed copy of the Talend Studio
- A properly configured Hadoop cluster
- The supporting files for this lesson

First, you will use Sqoop to import a MySQL table to HDFS.

Importing Tables with Sqoop

Task outline

Sqoop is a tool designed to transfer data between Hadoop and relational databases. You can use Sqoop to import data from a relational database management system (RDBMS) such as MySQL or Oracle, into the Hadoop Distributed File System (HDFS), transform the data using Map Reduce, and export that data back into a RDBMS.

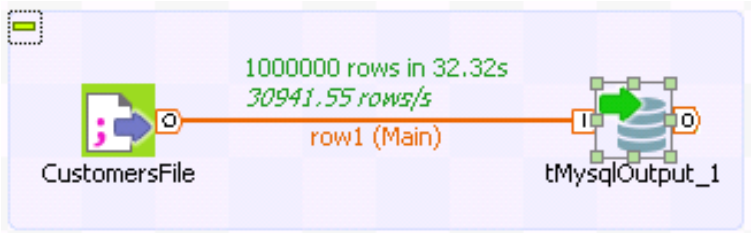
Sqoop automates most of this process, relying on the database to describe the schema for the data to be imported. Sqoop creates and runs a Map-only Map Reduce Job to import the data.

In this exercise, you will transfer MySQL tables into HDFS using Sqoop dedicated components. First, you will push the customers data into a MySQL database.

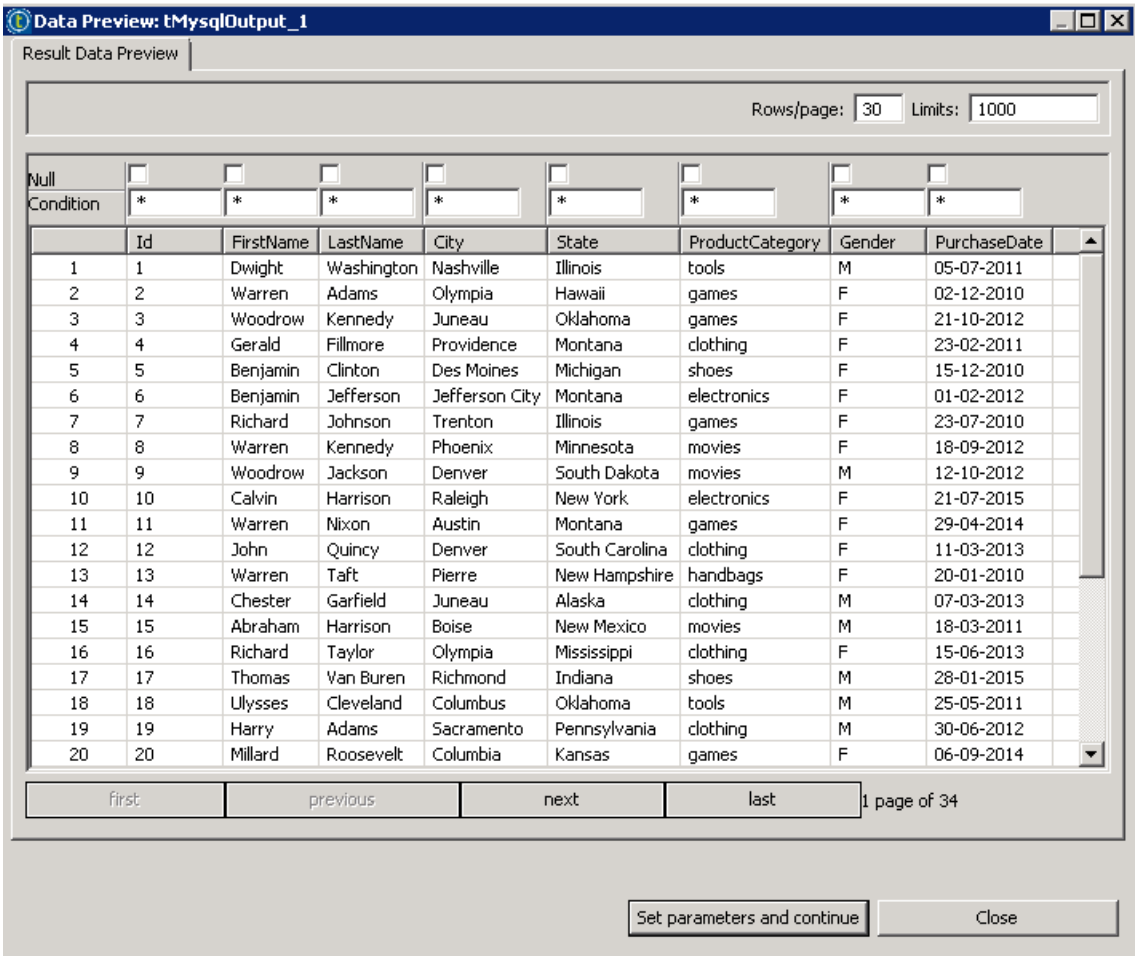
Preparing the MySQL database

The Job to copy the Customers data in MySQL has already been created for you. It is saved in the C:\StudentFiles\BDBasics folder.

1. IMPORT THE PushCustomerDataToMySQL JOB
   From the **C:\StudentFiles\BDBasics\JobDesigns.zip** archive file, import the **PushCustomerDataToMySQL** Job and the associated **RemoteMySQL** database Metadata.
2. RUN THE JOB
   Run **PushCustomerDataToMySQL**.
   This will copy 1 million rows in a remotely hosted MySQL database named CustomersData:



3. USE THE DATA VIEWER TO VALIDATE THE DATA
   To check the data copied in the CustomersData table, right-click the **tMysqlOutput** component, then click **Data Viewer**:



	Id	FirstName	LastName	City	State	ProductCategory	Gender	PurchaseDate
1	1	Dwight	Washington	Nashville	Illinois	tools	M	05-07-2011
2	2	Warren	Adams	Olympia	Hawaii	games	F	02-12-2010
3	3	Woodrow	Kennedy	Juneau	Oklahoma	games	F	21-10-2012
4	4	Gerald	Fillmore	Providence	Montana	clothing	F	23-02-2011
5	5	Benjamin	Clinton	Des Moines	Michigan	shoes	F	15-12-2010
6	6	Benjamin	Jefferson	Jefferson City	Montana	electronics	F	01-02-2012
7	7	Richard	Johnson	Trenton	Illinois	games	F	23-07-2010
8	8	Warren	Kennedy	Phoenix	Minnesota	movies	F	18-09-2012
9	9	Woodrow	Jackson	Denver	South Dakota	movies	M	12-10-2012
10	10	Calvin	Harrison	Raleigh	New York	electronics	F	21-07-2015
11	11	Warren	Nixon	Austin	Montana	games	F	29-04-2014
12	12	John	Quincy	Denver	South Carolina	clothing	F	11-03-2013
13	13	Warren	Taft	Pierre	New Hampshire	handbags	F	20-01-2010
14	14	Chester	Garfield	Juneau	Alaska	clothing	M	07-03-2013
15	15	Abraham	Harrison	Boise	New Mexico	movies	M	18-03-2011
16	16	Richard	Taylor	Olympia	Mississippi	clothing	F	15-06-2013
17	17	Thomas	Van Buren	Richmond	Indiana	shoes	M	28-01-2015
18	18	Ulysses	Cleveland	Columbus	Oklahoma	tools	M	25-05-2011
19	19	Harry	Adams	Sacramento	Pennsylvania	clothing	M	30-06-2012
20	20	Millard	Roosevelt	Columbia	Kansas	games	F	06-09-2014

You will now transfer this table to HDFS using the **tSqoopImport** component.

Create a generic database metadata

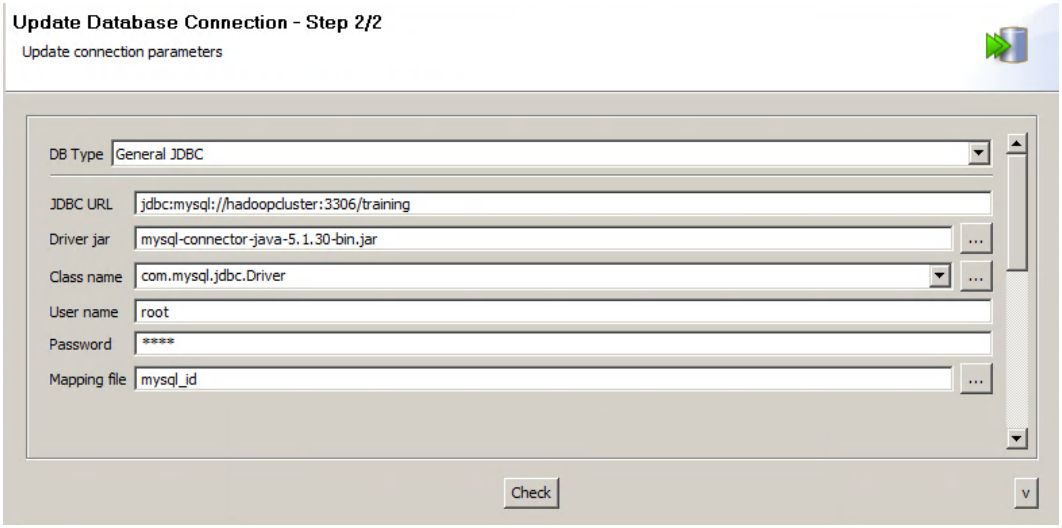
The **tSqoopImport** component calls Sqoop to transfer data from a relational database management system, such as MySQL or Oracle, into the Hadoop Distributed File System.

First, you will create a generic database Metadata, which is required for Sqoop to connect to your MySQL database.

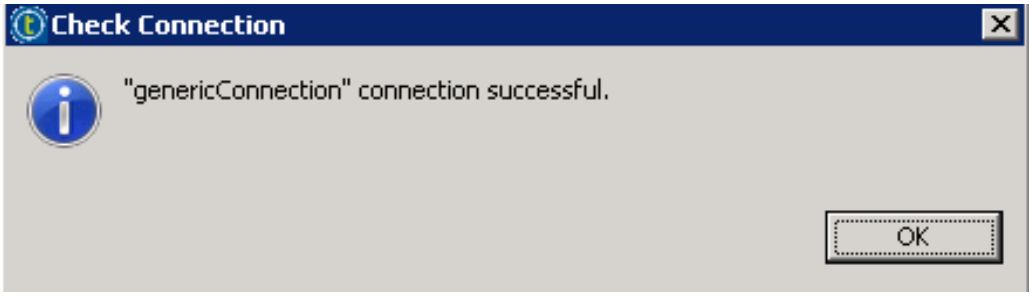
1. CREATE A NEW DATABASE CONNECTION METADATA
   Create a new **Generic JDBC** database connection metadata named *genericConnection*.
   a. In the **Repository**, under **Metadata**, right-click **Db Connections** , then click **Create connection**.
   b. In the **Name** box, enter *genericConnection*. You can also add a **Purpose** and a **Description**. Then, click **Next**.
   c. In the **DB Type** list, select *General JDBC*.
2. CONFIGURE THE METADATA TO CONFIGURE THE CONNECTION TO YOUR MySQL DATABASE
   To configure the genericConnection metadata, add the database URL, driver jar, class name, credentials and mapping file.
   a. In the **JDBC URL** box, enter
   jdbc:mysql://hadoopcluster:3306/training .
   b. Click the **(...)** next to the **Driver jar** box and select
   mysql-connector-java-5.1.30-bin.jar.
   c. Click the **(...)** next to the **Class name** box, then, in the drop-down list, select **com.mysql.jdbc.Driver**.

d. In the **User name** and **Password** boxes, enter *root*.

e. Click the **(...)** next to the **Mapping file** box, then select **mapping_Mysql.xml** in the list. Click **OK**.
Your configuration should be as follows:



3. TEST THE CONNECTIVITY
Click **Check**. Your connection should be successful:



4. FINALIZE THE METADATA CREATION
Click **OK** and **Finish**.
The genericConnection Metadata appears in the Repository.

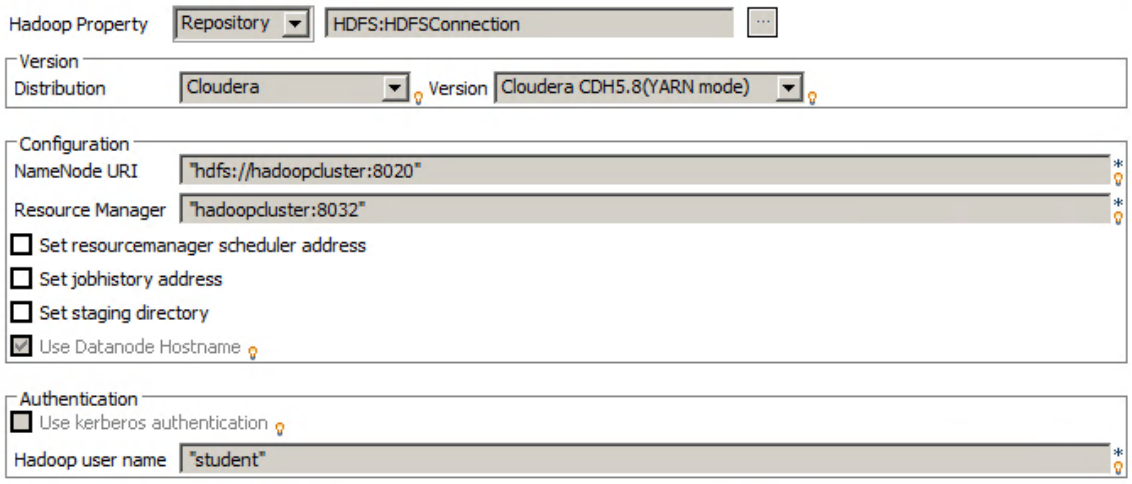
Importing tables

You will create a simple Job to import the CustomersData MySQL table into HDFS using a **tSqoopImport** component.

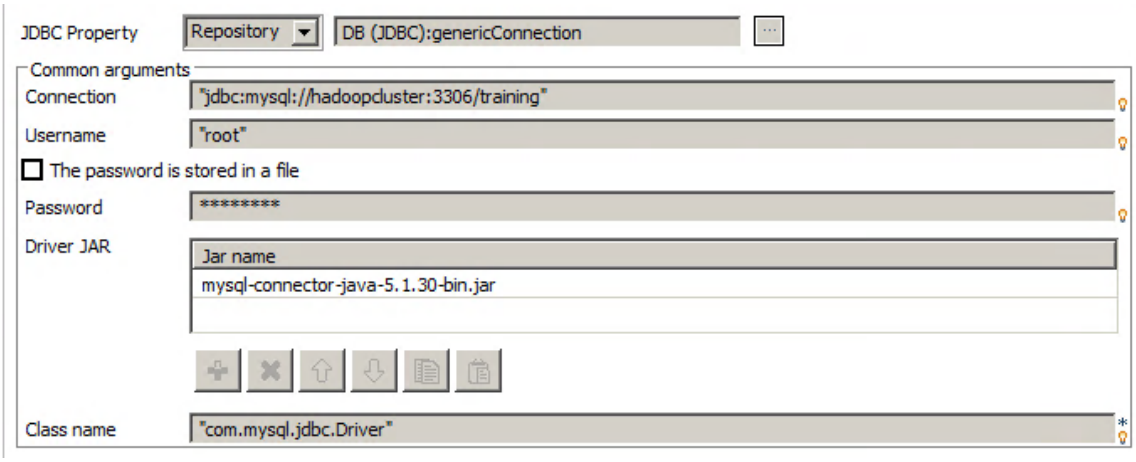
In the **tSqoopImport** component, the first option is to choose the Mode: Commandline or Java API.
If you choose Commandline, the Sqoop shell is used to call Sqoop. In this mode, you have to deploy and run the Job in the host where Sqoop is installed. This means that you have to install and use the Jobserver, as described in the Talend Data Integration Advanced training, or as described in the Talend Installation Guide.
If you select Use Java API, the Java API is used to call Sqoop. In this mode, the Job can be run locally in the Studio, but you have to configure the connection to your cluster.
Note: A JDK is required to execute the Job with the Java API and the JDK versions on both machines must be compatible.

1. CREATE A NEW STANDARD JOB
Create a new **standard Job** and name it *SqoopImport*.
2. ADD A tSqoopImport COMPONENT
Add a **tSqoopImport** component and open the **Component** view.
3. CONFIGURE THE tSqoopImport COMPONENT TO USE THE Java API
In the **Mode** box, select **Use Java API**.

4. CONFIGURE THE tSqoopImport COMPONENT TO USE THE HDFSConnection METADATA
   In the **Hadoop Property** list, select **Repository** , then browse to find the **HDFSConnection** you configured earlier in the course.
   This will configure the Distribution, the Hadoop version, the NameNode URI, the Resource Manager, and the Hadoop user name:

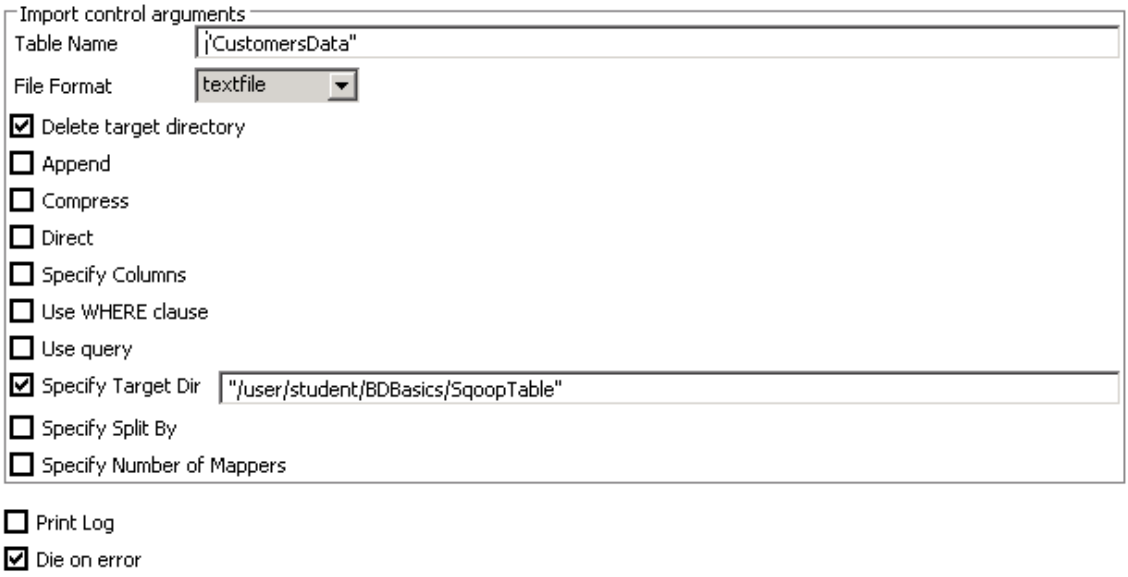


5. USE YOUR DATABASE CONNECTION METADATA
   In the **JDBC Property** list, select **Repository**, then browse to find the **genericConnection** Metadata.
   This configures the Connection, the Username, the Password, and the Driver JAR values:



6. IMPORT THE CustomersData TABLE
   In the **Table Name** box, enter "*CustomersData"*and select **Delete target directory**.
7. CONFIGURE THE TARGET DIRECTORY
   Select **Specify Target Dir** and enter
   "/user/student/BDBasics/SqoopTable".

Your configuration should be as follows:



Run the Job and verify the results

As you did previously, you will run your Job and check the results in the Console and in Hue.

1. RUN YOUR JOB AND VERIFY THE RESULTS IN THE CONSOLE
   Examine the output in the Console.
   a. Run your Job and check the results in the **Console**. The last line should be an exit code equal to 0.
   b. You can investigate what you see in the Console a little bit. For example, you can see the execution of the **Map Reduce Job** generated by the Sqoop import:

```
[INFO ]: org.apache.hadoop.yarn.client.api.impl.YarnClientImpl - Submitted application
application_1476871003091_0006
[INFO ]: org.apache.hadoop.mapreduce.Job - The url to track the job:
http://hadoopcluster:8088/proxy/application_1476871003091_0006/
[INFO ]: org.apache.hadoop.mapreduce.Job - Running job: job_1476871003091_0006
[INFO ]: org.apache.hadoop.mapreduce.Job - Job job_1476871003091_0006 running in uber mode :
false
[INFO ]: org.apache.hadoop.mapreduce.Job -  map 0% reduce 0%
[INFO ]: org.apache.hadoop.mapreduce.Job -  map 75% reduce 0%
[INFO ]: org.apache.hadoop.mapreduce.Job -  map 100% reduce 0%
[INFO ]: org.apache.hadoop.mapreduce.Job - Job job_1476871003091_0006 completed successfully
```

   Note: Your Job Id will be different. The Id is the number following "job_...". In the current example, the Job Id is 1476871003091_0006.

c. Right after, you will find a recap of various counters:

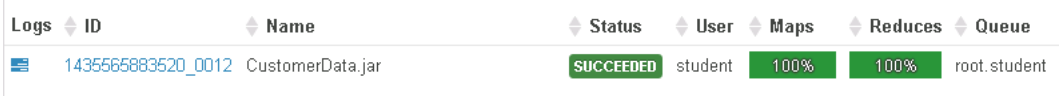
```
[INFO ]: org.apache.hadoop.mapreduce.Job - Counters: 30
    File System Counters
        FILE: Number of bytes read=0
        FILE: Number of bytes written=490328
        FILE: Number of read operations=0
        FILE: Number of large read operations=0
        FILE: Number of write operations=0
        HDFS: Number of bytes read=429
        HDFS: Number of bytes written=72788117
        HDFS: Number of read operations=16
        HDFS: Number of large read operations=0
        HDFS: Number of write operations=8
    Job Counters
        Launched map tasks=4
        Other local map tasks=4
        Total time spent by all maps in occupied slots (ms)=41179
        Total time spent by all reduces in occupied slots (ms)=0
        Total time spent by all map tasks (ms)=41179
        Total vcore-seconds taken by all map tasks=41179
        Total megabyte-seconds taken by all map tasks=42167296
    Map-Reduce Framework
        Map input records=1000000
        Map output records=1000000
        Input split bytes=429
        Spilled Records=0
        Failed Shuffles=0
        Merged Map outputs=0
        GC time elapsed (ms)=467
        CPU time spent (ms)=31230
        Physical memory (bytes) snapshot=1421729792
        Virtual memory (bytes) snapshot=6308982784
        Total committed heap usage (bytes)=2343043072
    File Input Format Counters
        Bytes Read=0
    File Output Format Counters
        Bytes Written=72788117
[INFO ]: org.apache.sqoop.mapreduce.ImportJobBase - Transferred 69.4162 MB in
20.8361 seconds (3.3315 MB/sec)
[INFO ]: org.apache.sqoop.mapreduce.ImportJobBase - Retrieved 1000000 records.
```

Here you can see that 4 map tasks ran on the cluster and that the 1 million records were transferred in approximately 21 seconds.

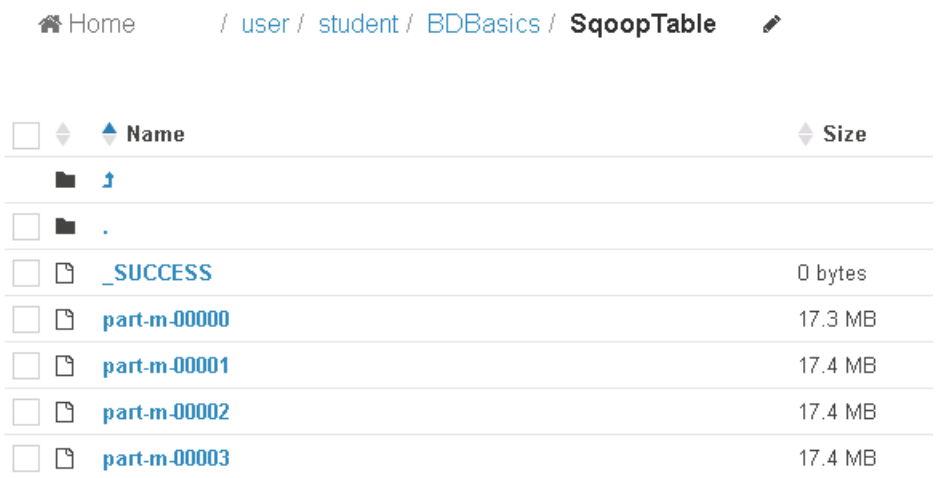
2. CONNECT TO Hue AND VERIFY YOUR RESULTS
   In Hue, use the Job browser to find your jobs and the File browser to find your data.

   a. To see your Job in **Hue**, click **Job Browser**.
   The Job Browser window will open, and you will see the list of all your Jobs.

   b. From the Console, find the **Id** of your Job. Then, in Hue, find your Job Id in the Job list. It should be followed by green boxes, corresponding to a successful execution of Map and Reduce tasks:

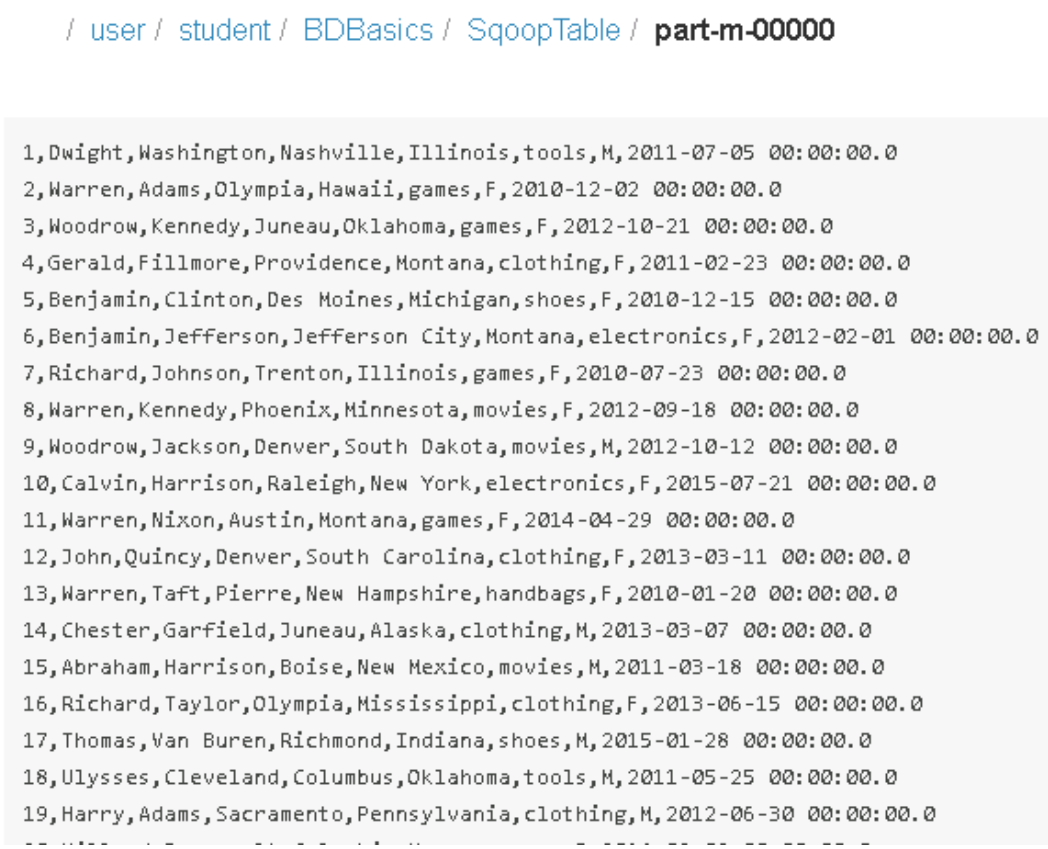


c. In **Hue**, click **File Browser** and navigate to /user/student/BDBasics/SqoopTable:



The data has been split in multiple parts.

d. Click **part-m-00000**:



Here you can check that your data have been imported as expected.

Now that you have imported a MySQL table to HDFS using a **tSqoopImport** component, you can continue to experiment working with tables. The next topic will show you how to create tables in HDFS with Hive.

Creating Tables with Hive

Task outline

Hive is a data warehouse infrastructure tool used to process structured data in Hadoop. It is a database that resides on top of Hadoop to summarize Big Data, and makes querying and analyzing easy.

Hive supports HiveQL, which is similar to SQL, but does not support the complete construct of SQL.

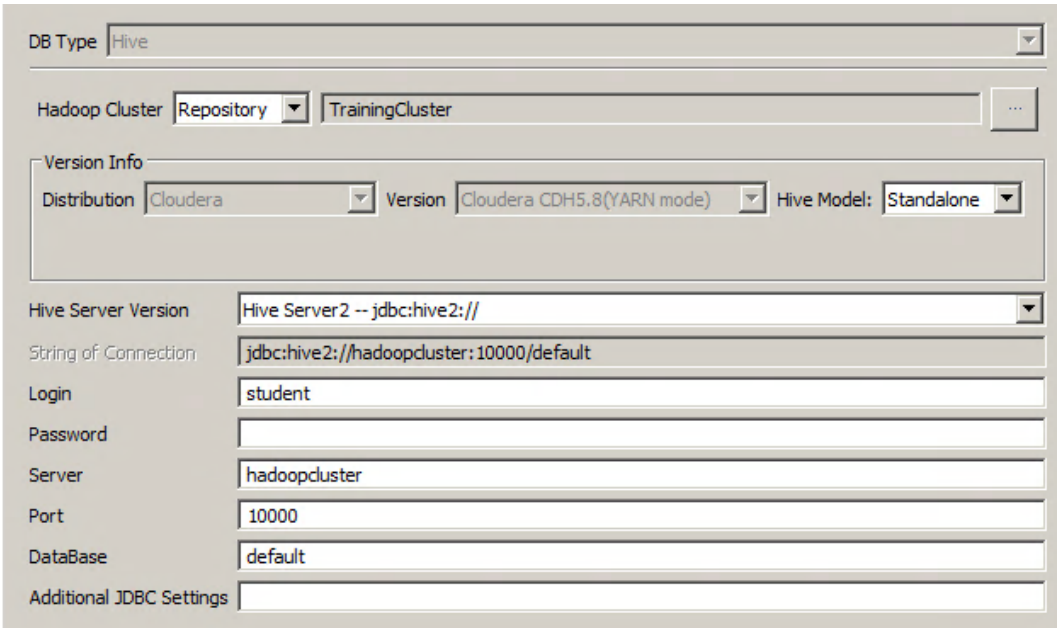
Hive converts the HiveQL query into Map Reduce code and submits it to the Hadoop cluster. Hive, through HiveQL language, provides a higher level of abstraction over Java Map Reduce programming.

First, you will create Hive Metadata in the Repository. Then, you will use various methods to create Hive tables.

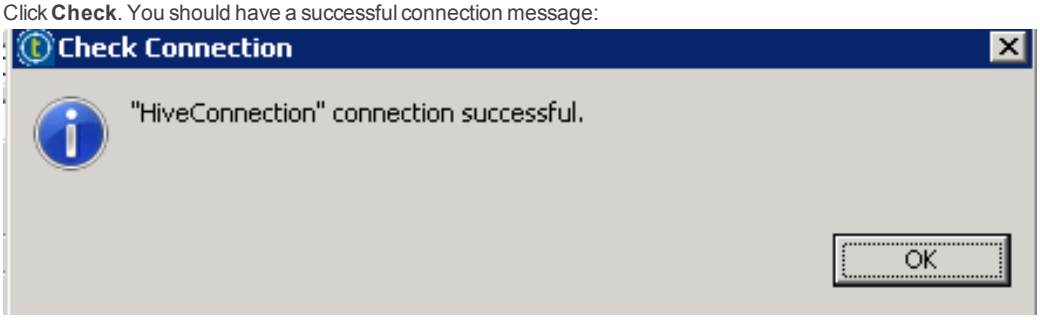
Create Hive connection metadata

As you did previously for the cluster and HDFS connections, you will create a Hive connection metadata in the Repository.

1. CREATE A HIVE CONNECTION METADATA
   Right-click **TrainingCluster** in the **Repository** under Metadata/Hadoop Cluster, , then click **Create Hive**.
2. NAME THE METADATA
   In the **Name** box, enter *HiveConnection*, then click **Next**.
3. CONFIGURE THE CONNECTION TO HIVE
   Select the Hive Model and enter the port number.
   a. In the **Hive Model** list, select **Standalone**.
   b. In the **Port** box, enter *10000*.
   Your configuration should be as follows:



4. TEST THE CONNECTION TO HIVE
   Click **Check**. You should have a successful connection message:

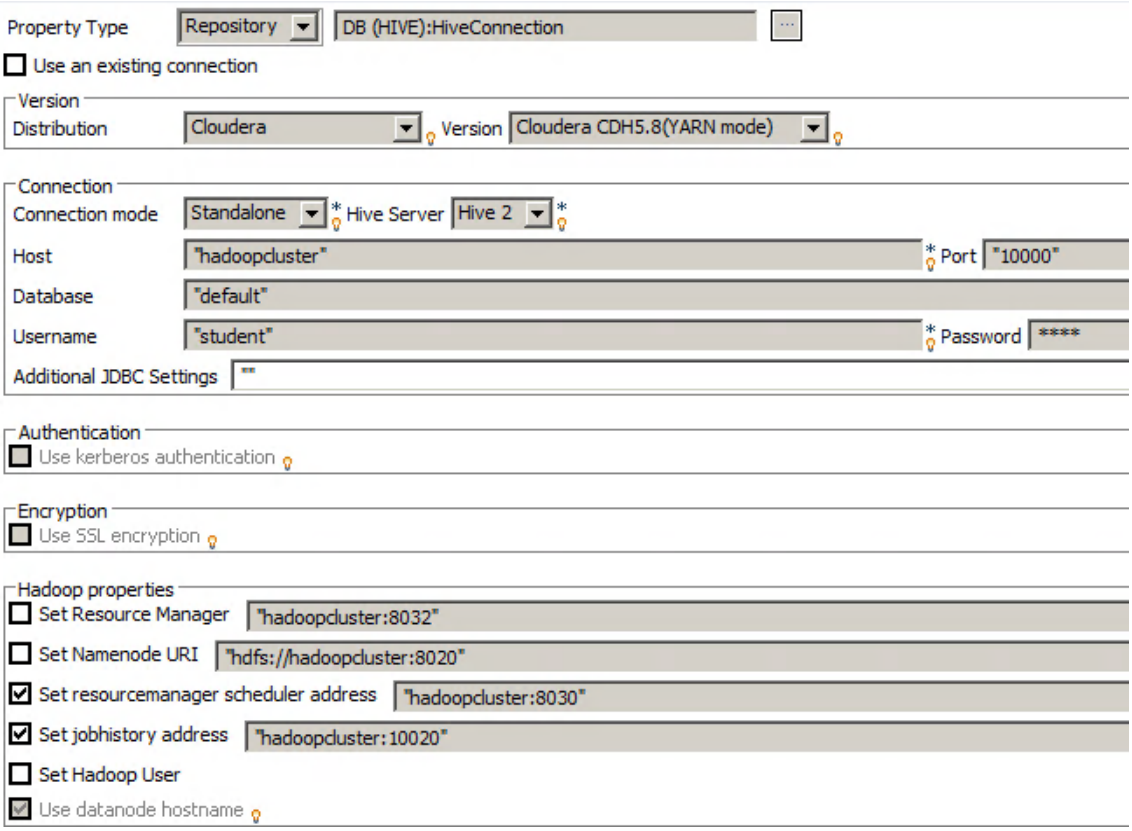


5. FINALIZE THE METADATA CREATION
   Click **Ok** and **Finish** to create the Hive connection Metadata.
   The Hive connection metadata appears in the Repository under Hadoop Cluster/TrainingCluster. The Hive connection metadata is also available in the Repository under Db Connections.

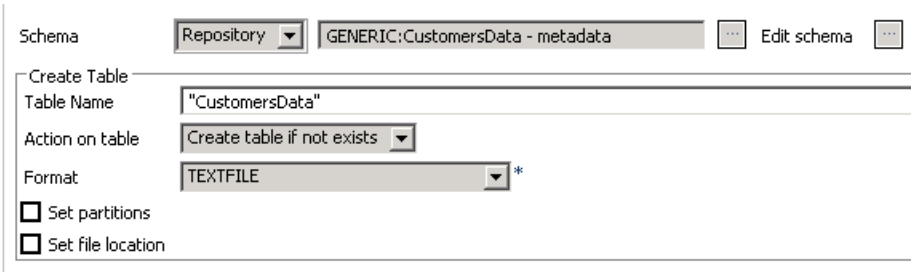
Create a Hive table manually

You will now create a Job that will create a table and populate it with the customer's data.

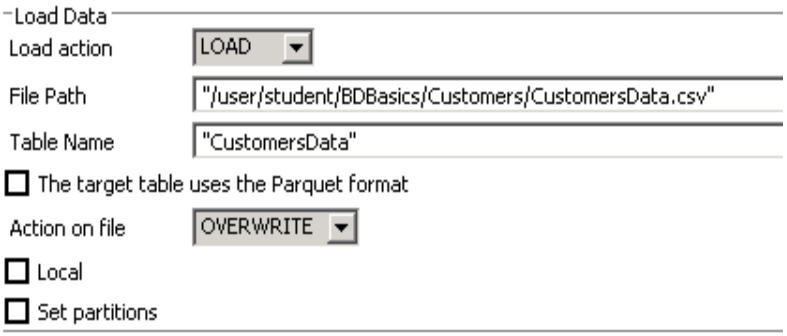
1. CREATE A NEW STANDARD JOB
   Create a new standard Job and name it *HiveLoad*.
2. ADD A tHiveCreateTable COMPONENT
   Add a **tHiveCreateTable** and open the **Component** view.
   The tHiveCreateTable component will create an empty Hive table according to the specified schema.
3. USE THE HiveConnection METADATA
   In the **Property Type** list, select **Repository** , then browse to find the **HiveConnection** Metadata:



4. USE THE CustomersData GENERIC SCHEMA
   In the **Schema** list, select **Repository** then browse to find the **CustomersData generic schema** metadata.
5. READ THE CustomersData TABLE
   In the **Table Name** box, enter *"CustomersData"*.
6. SELECT THE ACTION ON TABLE
   In the **Action on table** list, select **Create table if not exists**:



7. ADD A tHiveLoad COMPONENT
   Add a **tHiveLoad** component.
   The tHiveLoad component will populate the table with data.
8. CONNECT WITH A TRIGGER
   Connect **tHiveLoad** to **tHiveCreateTable** with an **OnSubjobOk** trigger and open the **Component** view.
9. USE THE HiveConnection METADATA
   Set the **Property Type** to **Repository** and use the **HiveConnection** Metadata.
10. LOAD the CustomersData.csv FILE
    Configure **tHiveLoad** to load the *CustomersData.csv* file in a table named *CustomersData*.
    a. In the **Load action** list, select **LOAD**.
    b. In the **File Path** box, enter *"/user/student/BDBasics/Customers/CustomersData.csv"*.
    c. In the **Table Name** box, enter *"CustomersData"*.
    d. In the **Action on file** list, select **OVERWRITE**.



You will now run your Job and check the results in the Console and in Hue.

Run the Job and verify the results

1. RUN YOUR JOB
   Run your Job and check the output in the **Console**:

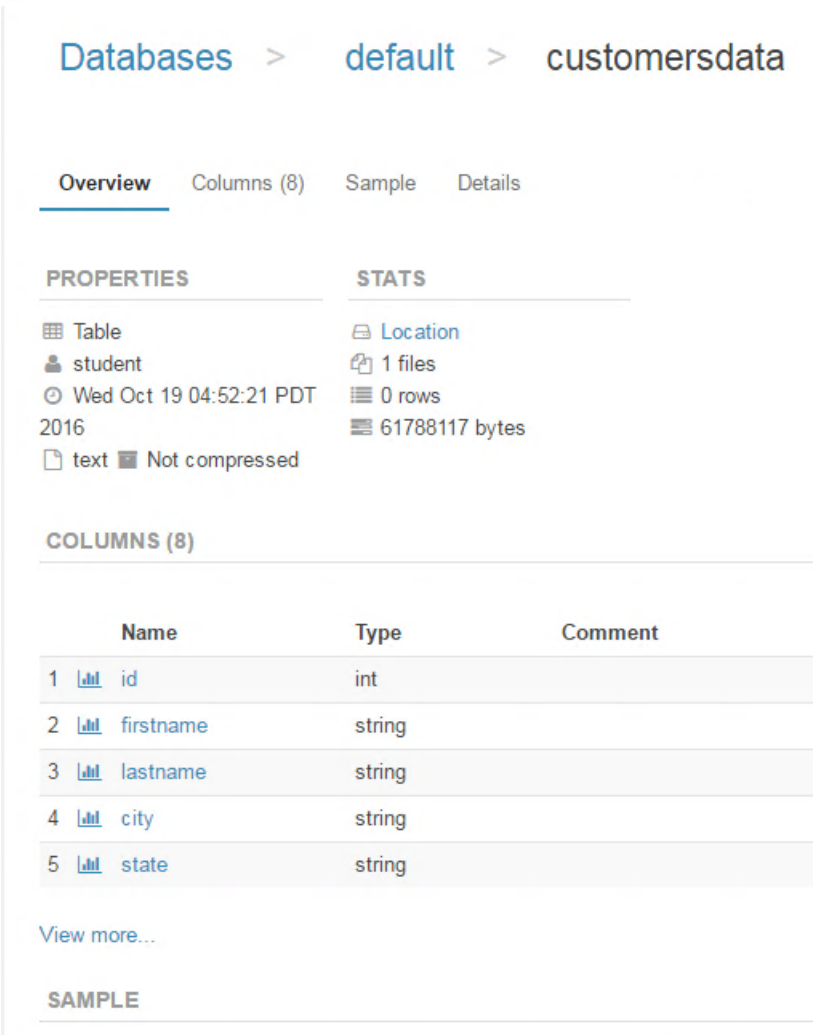
```
Starting job HiveLoad at 04:52 19/10/2016.

[INFO ]: bdbasics.hiveload_0_1.HiveLoad - TalendJob: 'HiveLoad' - Start.
[statistics] connecting to socket on port 3979
[statistics] connected
[INFO ]: bdbasics.hiveload_0_1.HiveLoad - tHiveCreateTable_1 - Connection attempt to
'jdbc:hive2://hadoopcluster:10000/default' with the username 'student'.
[INFO ]: org.apache.hive.jdbc.Utils - Supplied authorities: hadoopcluster:10000
[INFO ]: org.apache.hive.jdbc.Utils - Resolved authority: hadoopcluster:10000
[INFO ]: bdbasics.hiveload_0_1.HiveLoad - tHiveCreateTable_1 - Connection to
'jdbc:hive2://hadoopcluster:10000/default' has succeeded.
[INFO ]: bdbasics.hiveload_0_1.HiveLoad - tHiveLoad_1 - Connection attempt to
'jdbc:hive2://hadoopcluster:10000/default' with the username 'student'.
[INFO ]: org.apache.hive.jdbc.Utils - Supplied authorities: hadoopcluster:10000
[INFO ]: org.apache.hive.jdbc.Utils - Resolved authority: hadoopcluster:10000
[INFO ]: bdbasics.hiveload_0_1.HiveLoad - tHiveLoad_1 - Connection to
'jdbc:hive2://hadoopcluster:10000/default' has succeeded.
[statistics] disconnected
[INFO ]: bdbasics.hiveload_0_1.HiveLoad - TalendJob: 'HiveLoad' - Done.
Job HiveLoad ended at 04:52 19/10/2016. [exit code=0]
```

The Job successfully executed. Now, you can check the results in Hue.

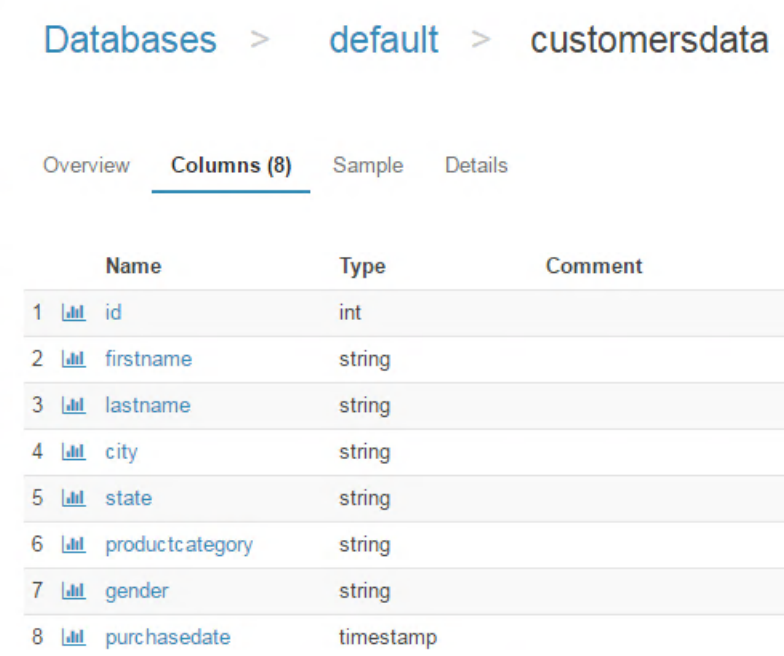
2. CONNECT TO Hue AND CHECK YOUR RESULTS
   In **Hue**, use the **Data Browsers** to examine your CustomersData Hive table.
   a. In **Hue**, click **Data Browsers**.Then, click **Metastore Tables**:

b. Click the **customersdata** table:



This is the overview of your CustomersData table.

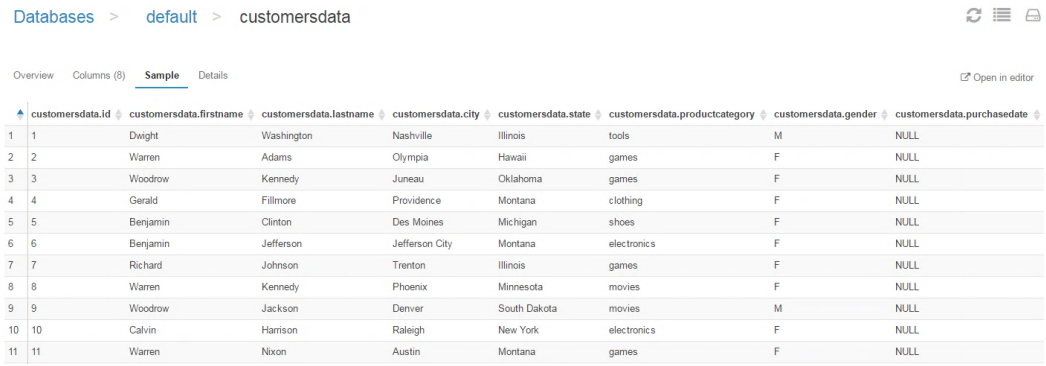
c. Click Columns(8) to inspect the schema of the customersdata table:



	Name	Type	Comment
1	id	int	
2	firstname	string	
3	lastname	string	
4	city	string	
5	state	string	
6	productcategory	string	
7	gender	string	
8	purchasedate	timestamp	

Note: The columns type have been automatically converted.

d. Click **Sample**:



	customersdata.id	customersdata.firstname	customersdata.lastname	customersdata.city	customersdata.state	customersdata.productcategory	customersdata.gender	customersdata.purchasedate
1	1	Dwight	Washington	Nashville	Illinois	tools	M	NULL
2	2	Warren	Adams	Olympia	Hawaii	games	F	NULL
3	3	Woodrow	Kennedy	Juneau	Oklahoma	games	F	NULL
4	4	Gerald	Fillmore	Providence	Montana	clothing	F	NULL
5	5	Benjamin	Clinton	Des Moines	Michigan	shoes	F	NULL
6	6	Benjamin	Jefferson	Jefferson City	Montana	electronics	F	NULL
7	7	Richard	Johnson	Trenton	Illinois	games	F	NULL
8	8	Warren	Kennedy	Phoenix	Minnesota	movies	F	NULL
9	9	Woodrow	Jackson	Denver	South Dakota	movies	M	NULL
10	10	Calvin	Harrison	Raleigh	New York	electronics	F	NULL
11	11	Warren	Nixon	Austin	Montana	games	F	NULL

If you examine the results in the purchasedata column, you will see only NULL values. This is due to the fact that the timestamp format of Hive is not equivalent to the date format in the Talend Studio. This leads to a failure in the data conversion.
A possible workaround is to consider dates as String types.

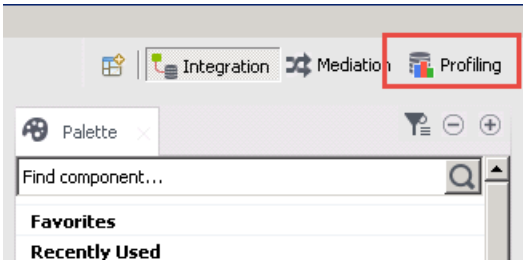
You will now experiment with another way to create a Hive table. You will use a Wizard which will automatically create a Hive table from a file stored in HDFS.

Using the Hive table creation wizard

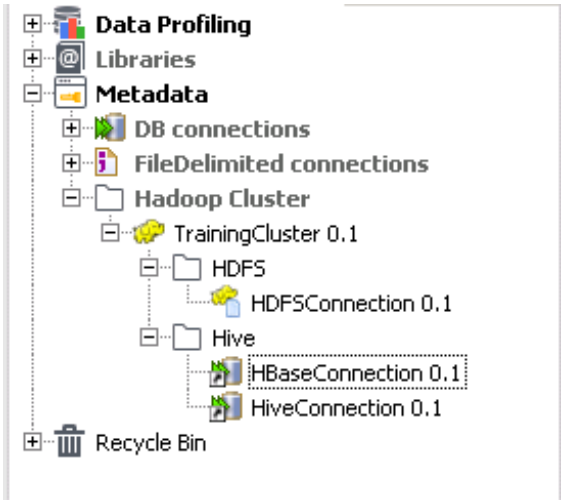
To create a Hive table automatically from the CustomersData.csv file stored on HDFS, you will have to change the perspective of the Studio. You will move to the Profiling perspective, which is dedicated to Data Quality analysis on database or on HDFS, depending on where you stored your data.

1. COPY CustomersData.csv TO HDFS
   To make sure that the CustomersData.csv file is available for the following steps, run the **PutCustomersData** Job again.

2. SWITCH TO THE Profiling PERSPECTIVE
In the upper-right corner of the Studio, click **Profiling** to open the Profiling perspective:



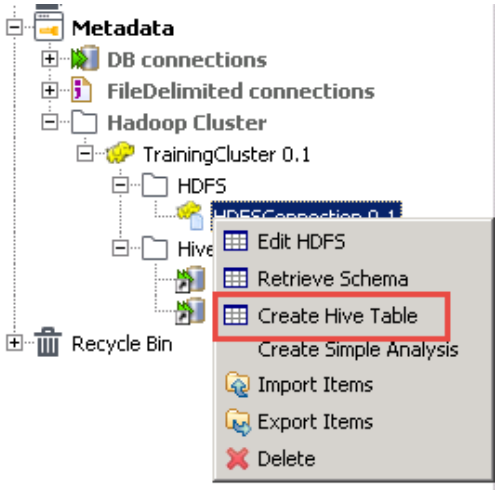
3. LOCATE THE HDFSConnection METADATA
In the **DQ Repository**, expand **Metadata/Hadoop Cluster/TrainingCluster**:



There, you will find the connection metadata you created earlier.

4. CREATE A HIVE TABLE FROM CustomersData.csv
From the **CustomersData.csv** file copied in HDFS, use the wizard to create a Hive table.

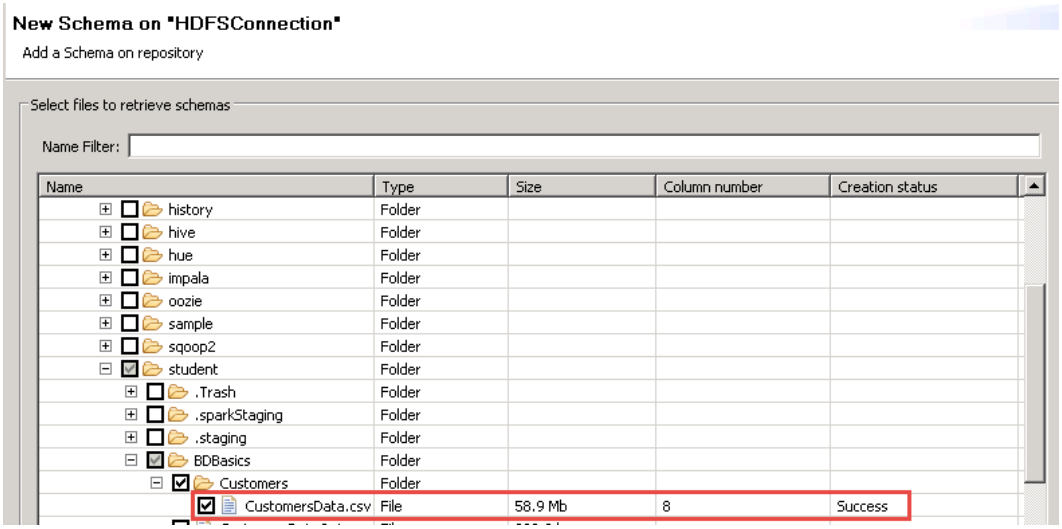
   a. Right-click **HDFSConnection**. Then click **Create Hive Table**:



   The connection to HDFS will be checked and next, the wizard to create a Hive table based on a file stored in HDFS will start.

   b. Browse to find the **CustomersData.csv** file under **/user/student/BDBasics/Customers**.

c. Select **CustomersData.csv** and wait until the **Creation status** changes to **Success**:

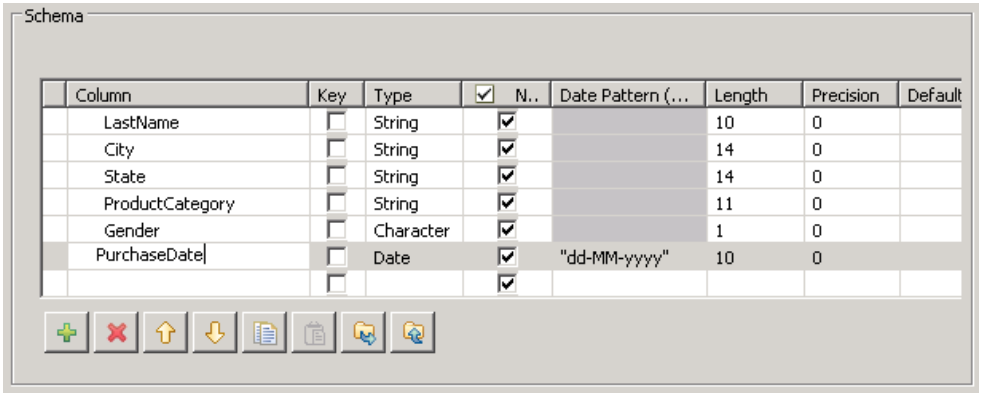


As expected, the wizard detects 8 columns in the CustomersData.csv file.
Note: The Hive table creation wizard will convert **all of the files** under the selected **folder**. So, in the current example, only the CustomersData.csv file will be converted because it is the only file in the Customers folder.

d. Click **Next**.

e. In the **Schema**, edit the columns' names as follows:

- Column0 > *Id*
- Column1 > *FirstName*
- Column2 > *LastName*
- Column3 > *City*
- Column4 > *State*
- Column5 > *ProductCategory*
- Column6 > *Gender*
- Column7 > *PurchaseDate*



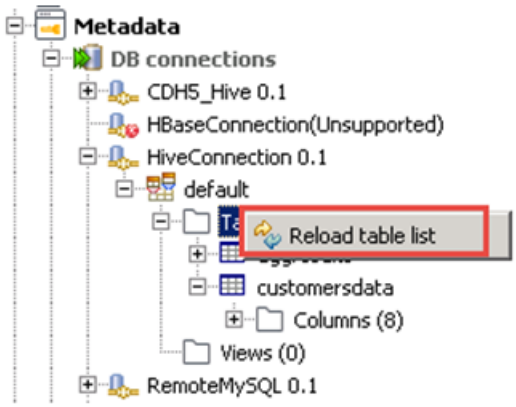
f. Click **Next**.

g. In the **New Table Name** box, enter *CustomersData_auto*.

h. In the **Hive connection** list, select **HiveConnection** , then click **Finish**.
The Hive table is created and then the wizard closes.

5. REFRESH THE HIVE TABLE LIST
From the Repository, reload the Hive table list.

a. Under DQ Repository>Metadata>DB connections>HiveConnection>default, right-click **Tables** and click**Reload table list**:



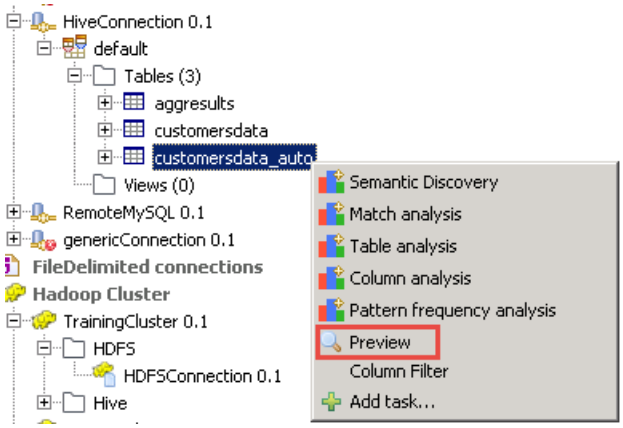
b. Click **Reload** in the Reload pop up message.
The CustomersData_auto table appears in the table list.

Verify the Hive table

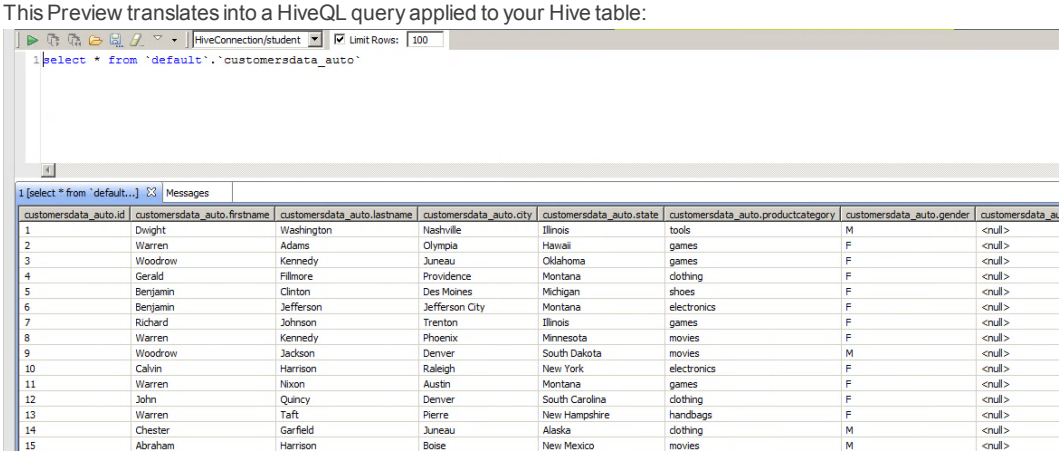
You can check the table in the Studio or in Hue.

1. PREVIEW THE HIVE TABLE
From the Repository, preview the **customersdata_auto** Hive table.

a. In the **DQ Repository**, right-click the **CustomersData_auto** table, then click **Preview**:



b. This Preview translates into a HiveQL query applied to your Hive table:



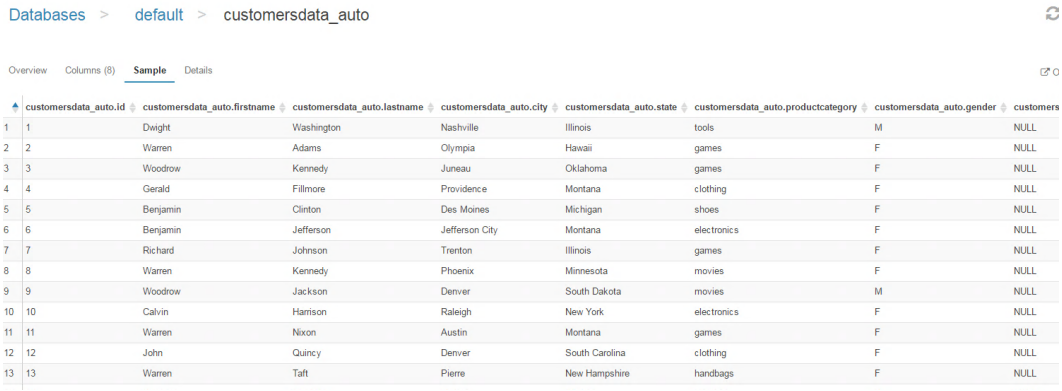
customersdata_auto.id	customersdata_auto.firstname	customersdata_auto.lastname	customersdata_auto.city	customersdata_auto.state	customersdata_auto.productcategory	customersdata_auto.gender	customersdata_a
1	Dwight	Washington	Nashville	Illinois	tools	M	<null>
2	Warren	Adams	Olympia	Hawaii	games	F	<null>
3	Woodrow	Kennedy	Juneau	Oklahoma	games	F	<null>
4	Gerald	Fillmore	Providence	Montana	clothing	F	<null>
5	Benjamin	Clinton	Des Moines	Michigan	shoes	F	<null>
6	Benjamin	Jefferson	Jefferson City	Montana	electronics	F	<null>
7	Richard	Johnson	Trenton	Illinois	games	F	<null>
8	Warren	Kennedy	Phoenix	Minnesota	movies	F	<null>
9	Woodrow	Jackson	Denver	South Dakota	movies	M	<null>
10	Calvin	Harrison	Raleigh	New York	electronics	F	<null>
11	Warren	Nixon	Austin	Montana	games	F	<null>
12	John	Quincy	Denver	South Carolina	clothing	F	<null>
13	Warren	Taft	Pierre	New Hampshire	handbags	F	<null>
14	Chester	Garfield	Juneau	Alaska	clothing	M	<null>
15	Abraham	Harrison	Boise	New Mexico	movies	M	<null>

The query is `"select * from default.customersdata_auto"` and appears in the SQL Editor, at the top of the window.
The result appears in the tab 1, displayed below the SQL editor.

2. VERIFY THE HIVE TABLE FROM Hue
Use the **Data Browsers** in **Hue** to check the **customersdata_auto** Hive table.

a. To check the new table in **Hue**, click **Data Browsers>Metastore Tables**, then click **customersdata_auto** in the table list.

b. Click **Sample**:



	customersdata_auto.id	customersdata_auto.firstname	customersdata_auto.lastname	customersdata_auto.city	customersdata_auto.state	customersdata_auto.productcategory	customersdata_auto.gender	customers
1	1	Dwight	Washington	Nashville	Illinois	tools	M	NULL
2	2	Warren	Adams	Olympia	Hawaii	games	F	NULL
3	3	Woodrow	Kennedy	Juneau	Oklahoma	games	F	NULL
4	4	Gerald	Fillmore	Providence	Montana	clothing	F	NULL
5	5	Benjamin	Clinton	Des Moines	Michigan	shoes	F	NULL
6	6	Benjamin	Jefferson	Jefferson City	Montana	electronics	F	NULL
7	7	Richard	Johnson	Trenton	Illinois	games	F	NULL
8	8	Warren	Kennedy	Phoenix	Minnesota	movies	F	NULL
9	9	Woodrow	Jackson	Denver	South Dakota	movies	M	NULL
10	10	Calvin	Harrison	Raleigh	New York	electronics	F	NULL
11	11	Warren	Nixon	Austin	Montana	games	F	NULL
12	12	John	Quincy	Denver	South Carolina	clothing	F	NULL
13	13	Warren	Taft	Pierre	New Hampshire	handbags	F	NULL

You have covered the various ways to work with Tables with Hive.

Next step

You have almost finished this section. Time for a quick review.

Review

Recap

In this lesson, you learned how to use Talend's Big Data components for Hive and Sqoop.

First, you imported a MySQL table to HDFS using the **tSqoopImport** component. The import was done through a Map-only Map Reduce Job.

Next, you manually created a Hive table with the **tHiveCreateTable** component and populated it with the **tHiveLoad** component.

Last, you used the Hive table creation wizard to automatically create your Hive table from a file stored on HDFS.

Intentionally blank

LESSON 5

Processing Data and Tables in HDFS

This chapter discusses:

Concepts

Outline

- Lesson objectives
- Introduction to Hive QL
- Profiling Hive tables
- Processing data with Pig
- Processing data with a Big Data batch Job
- Lab overview
- Wrap-up

Lesson objectives

After completing this lesson, you will be able to:

- Process Hive tables with a standard Job
- Process Hive tables in the Profiling perspective of the Studio
- Process data with Pig components
- Process data with a Big Data batch Job

Introduction to Hive QL

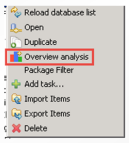
- Hive QL is a high-level programming language similar to SQL
- If your data is stored as Hive tables, you can use Hive QL to process it
- Hive converts the request to MapReduce Jobs that are executed on your cluster

Introduction to Hive QL

Function	MySQL	Hive QL
Retrieve all values	SELECT * FROM table;	SELECT * FROM table;
Select specific columns	SELECT column_name FROM table;	SELECT column_name FROM table;
Select some values	SELECT * FROM table WHERE rec_name= "value";	SELECT * FROM table WHERE rec_name= "value";
Count	SELECT COUNT(*) FROM table;	SELECT COUNT(*) FROM table;
Retrieve information	SELECT from_columns FROM table WHERE conditions;	SELECT from_columns FROM table WHERE conditions;

Profiling Hive tables

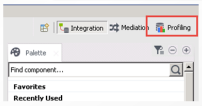
Switch to the Profiling perspective



1. Start the overview analysis wizard
2. Run the overview analysis
3. Start the column analysis wizard
4. Select indicators

Profiling Hive tables

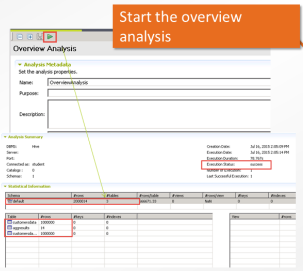
Right-click the Hive connection metadata and click Overview analysis



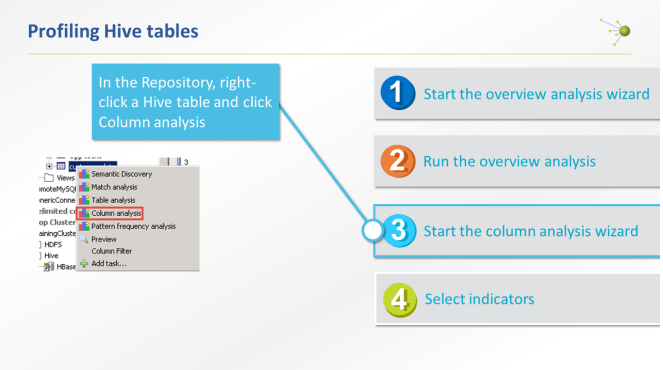
1. Start the overview analysis wizard
2. Run the overview analysis
3. Start the column analysis wizard
4. Select indicators

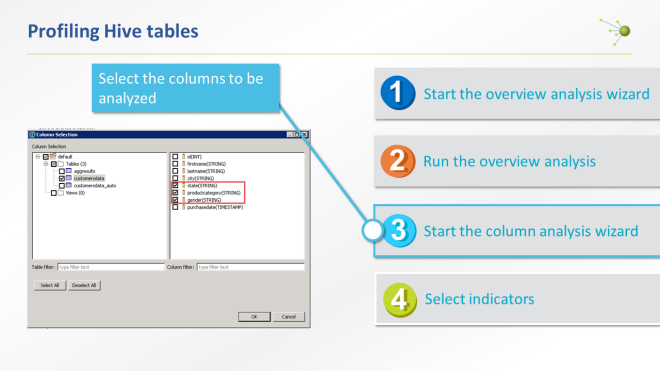
Profiling Hive tables

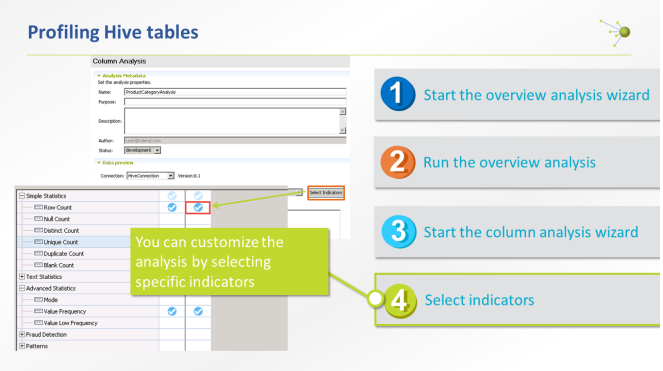
Start the overview analysis

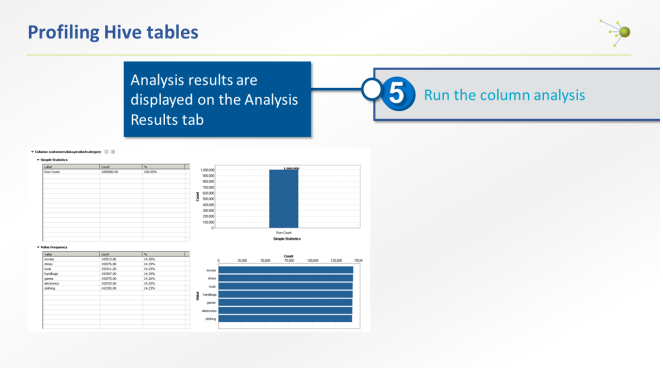


1. Start the overview analysis wizard
2. Run the overview analysis
3. Start the column analysis wizard
4. Select indicators


Profiling Hive tables
In the Repository, right-click a Hive table and click Column analysis
Semantic Discovery
Match analysis
Table analysis
Column analysis
Pattern frequency analysis
Preview
Column Filter
Add task...
1 Start the overview analysis wizard
2 Run the overview analysis
3 Start the column analysis wizard
4 Select indicators


Profiling Hive tables
Select the columns to be analyzed
1 Start the overview analysis wizard
2 Run the overview analysis
3 Start the column analysis wizard
4 Select indicators


Profiling Hive tables
Column Analysis
You can customize the analysis by selecting specific indicators
1 Start the overview analysis wizard
2 Run the overview analysis
3 Start the column analysis wizard
4 Select indicators


Profiling Hive tables
Analysis results are displayed on the Analysis Results tab
5 Run the column analysis

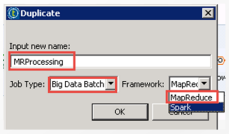
Processing data with Pig

Apache Pig is a platform for **analyzing large data sets**. It consists of a **high-level programming language**, Pig Latin, which opens Hadoop to non-Java programmers. Pig Latin also provides **common operations** to group, filter, join, or sort data.

Pig provides an execution engine on top of Hadoop. The Pig Latin script is converted in **MapReduce code** that is executed on your cluster.

Processing data with a Big Data batch Job

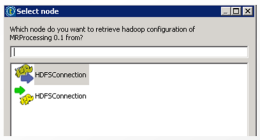
- Duplicate a Standard Job
- In the Job Type list, select Big Data Batch
- In the Framework list, select MapReduce



1. Select the Job type and framework
2. Retrieve the Hadoop configuration
3. Complete the configuration
4. Run the Job

Processing data with a Big Data batch Job

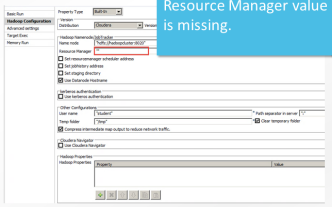
Select the component from which to retrieve the configuration.



1. Select the Job type and framework
2. Retrieve the Hadoop configuration
3. Complete the configuration
4. Run the Job

Processing data with a Big Data batch Job

Before running the Job, confirm that the Hadoop configuration is correct. In this example, the Resource Manager value is missing.



1. Select the Job type and framework
2. Retrieve the Hadoop configuration
3. Complete the configuration
4. Run the Job

Processing data with a Big Data batch Job

1. Select the Job type and framework
2. Retrieve the Hadoop configuration
3. Complete the configuration
4. Run the Job

Each Map or Reduce task to be executed on the cluster appears in the Designer as a progress bar.
When the task is complete, the progress bar changes from red to green

Lab overview

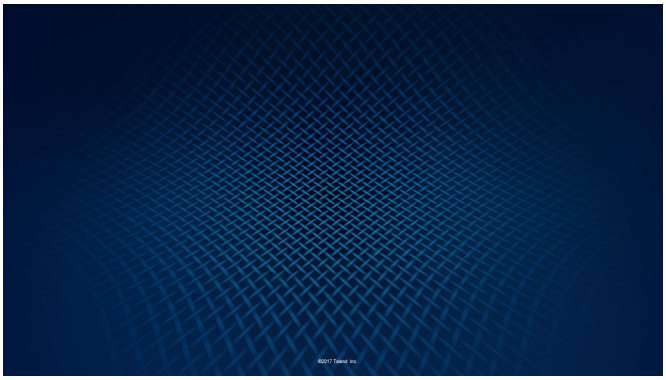
When your tables and data are stored on HDFS, you need to process them to extract useful information

In this lab, you will use different strategies to process your tables and data

- Use HiveQL language in a standard Job
- Use the Profiling perspective to analyze Hive tables
- Process data with Pig components
- Process data with a Big Data batch Job

Wrap-up

- Introduction to Hive QL
  - Hive QL is a high-level programming language similar to SQL that lets you process and query Hive tables
- Profiling Hive tables
  - Using the Profiling perspective, you can interactively extract useful information from your Hive tables
- Processing data with Pig
  - Apache Pig is a platform for analyzing large data sets
  - Pig Latin is a high-level programming language
- Processing data with a Big Data batch job
  - In Big Data batch Jobs, the cluster configuration is at the Job level



Processing Data and Tables in HDFS

Use case

Once stored in HDFS, you will need to process your tables and data to extract useful information.

Depending on your data type, you can adopt various strategies.

If your data is stored as Hive Tables, Hive QL might be the best way to address your needs. Hive QL is a high level programming language similar to SQL. Hive converts the request as Map Reduce Jobs that will be executed on your cluster. In this lesson, you will analyze Hive tables with a Job or with the Profiling view of the Studio.

If your data is stored as text files, one option is to use Pig. Pig Latin is a high-level language providing common operations to group, filter and join data. The Pig Latin script is automatically converted in Java Map Reduce code to be executed on the cluster. Talend provides components to use Pig with minimal programming efforts.

Another way to process your data, covered in this lesson, is to use a Big Data Batch Job. This kind of Job automatically converts the components in Java Map Reduce code that will be run on the Cluster.

Objectives

After completing this lesson, you will be able to:

- Process Hive tables with a standard Job
- Process Hive tables in the Profiling perspective of the Studio
- Process data with Pig components
- Process data with a Big Data Batch Job

Before you begin

Be sure that you are in a working environment that contains the following:

- A properly installed copy of the Talend Studio
- A properly configured Hadoop cluster
- The supporting files for this lesson

First, you will use Hive to process the tables created in the previous lesson.

Processing Hive Tables with Jobs

Task outline

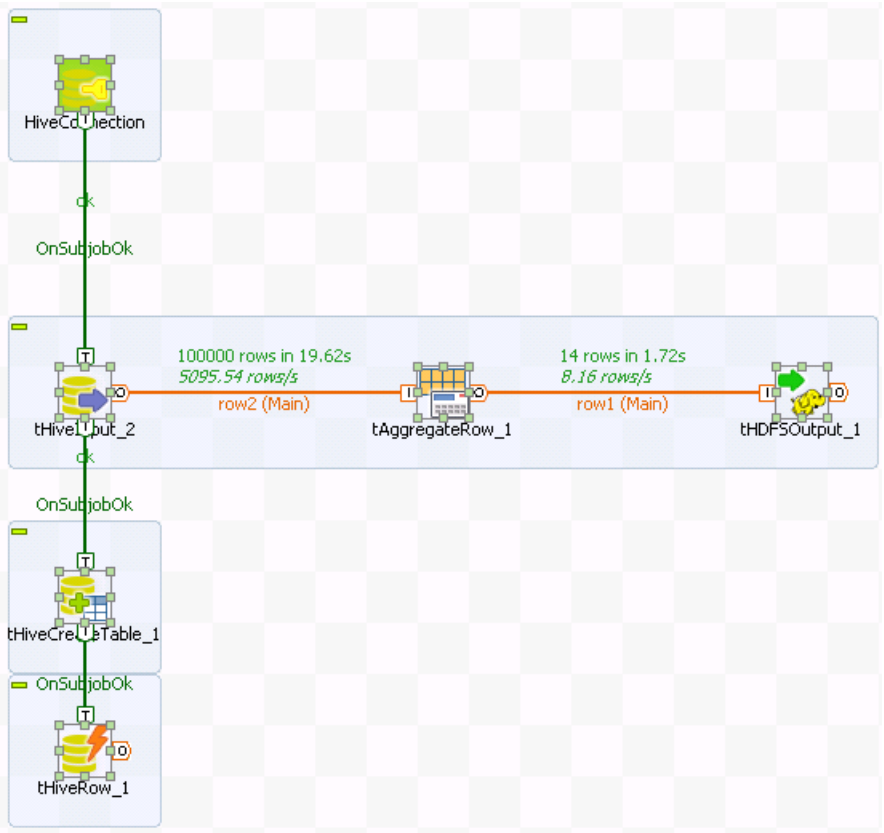
Hive converts the HiveQL query into Map Reduce code and then submits it to the Hadoop cluster. Through HiveQL language, Hive provides a higher level of abstraction compared to Java Map Reduce programming.

You will now analyze the customers data table you created in the Working with Tables lesson.

Using the Studio, you can analyze your data with your own Hive queries or you can use the **Profiling** view and use the Data Quality functions of the Studio over your Hive tables.

You will use various components to extract useful data from a Hive table, process it, and then store the result in another Hive table.

At the end of this lab, your Job will look like the following:



Extracting useful data

The first step is to collect useful data from the CustomersData Hive table you previously created.

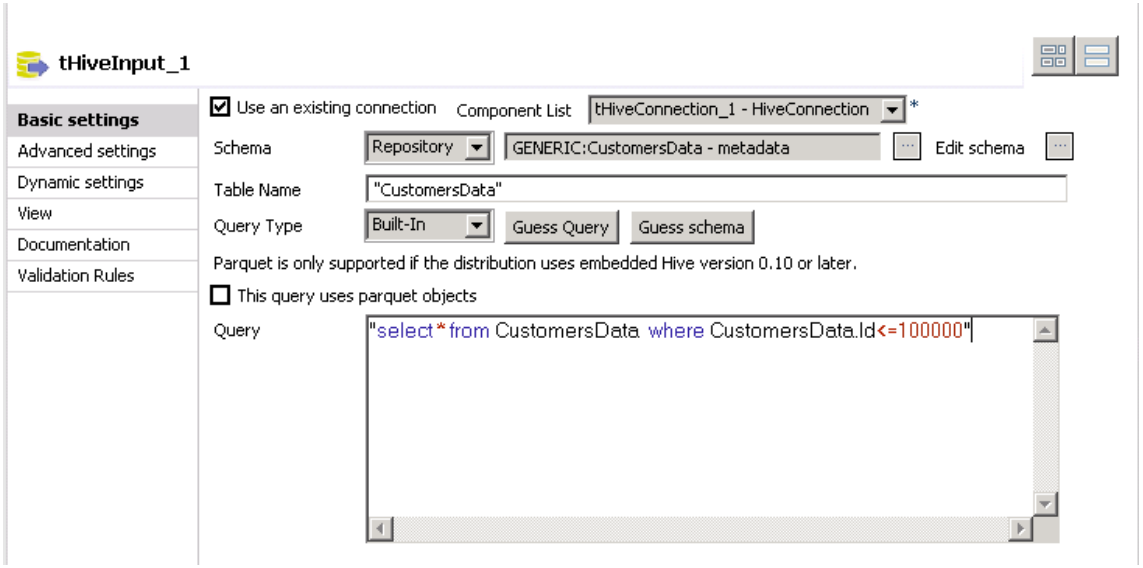
You will limit your investigations to the first 100 000 rows of the table.

1. SWITCH TO THE Integration PERSPECTIVE

   In the upper-right corner of the Studio, click **Integration** to open the Integration perspective.

2. CREATE A NEW STANDARD JOB
   Create a new **Standard Job** and name it *HiveProcessing*.

3. ADD A tHiveConnection COMPONENT
   Add a tHiveConnection component which uses the HiveConnection metadata.
   a. In the **Repository**, click **HiveConnection** under Metadata/Hadoop cluster/TrainingCluster/Hive.
   b. Drag it to the **Designer**.
   c. Select **tHiveConnection** in the **Components** list and click **OK**.
4. ADD A tHiveInput COMPONENT
   Add a tHiveInput component which uses the existing connection to Hive.
   a. Add a **tHiveInput** component and connect it with an **OnSubjobOk** trigger.
   b. Open the **Component** view.
   c. Select the **Use an existing connection** option.
   d. Ensure that **tHiveConnection**_1 is selected on the **Component List**.
5. CONFIGURE THE SCHEMA
   Set the **Schema** to **Repository** and then use the **CustomersData** generic schema metadata.
6. CONFIGURE THE TABLE NAME
   In the **Table Name** box, enter *"CustomersData"*.
7. READ THE FIRST 100,000 ROWS OF THE TABLE
   In the Query box, you will enter the HiveQL query that will be sent to the cluster.
   As mentionned in the Overview, the investigations will be limited to the first 100 000 rows.
   In the **Query** box, enter:
   `"select * from CustomersData where CustomersData.Id<=100000"`
   Your configuration should be as follows:



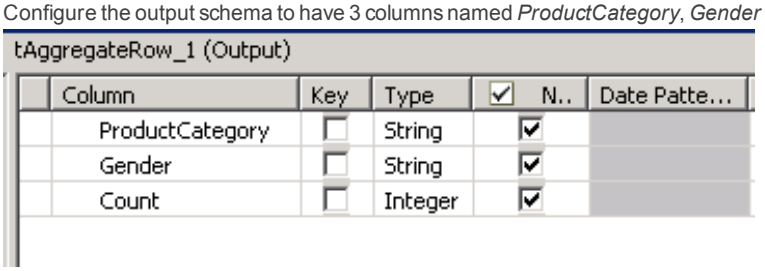
Process data

You will now aggregate the data and store the result in HDFS.

1. ADD A tAggregateRow COMPONENT
   Add a **tAggregateRow** component, connect it with the **Main** row, and then, open the **Component** view.
2. CONFIGURE THE SCHEMA
   Configure the schema to have 3 output columns named *ProductCategory*, *Gender* and *Count*. The first 2 columns are strings and the third one is an Integer.

a. Click **Sync columns** and then click **(...)** to edit the schema.

b. Configure the output schema to have 3 columns named *ProductCategory*, *Gender* and *Count*, as follows:



c. Click **OK** to save the schema.

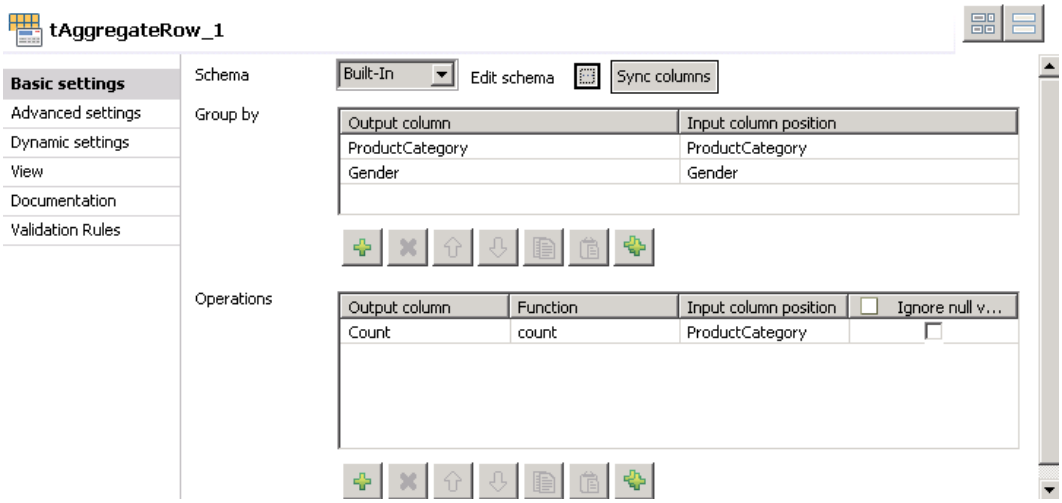
3. CONFIGURE THE AGGREGATION
Configure the **Group by** and **Operations** tables to aggregate your data by **ProductCategory** and **Gender**.

a. Click the **green plus sign** below the **Group by** table to add 2 Output column: *ProductCategory* and *Gender*.

b. Click the **green plus sign** below the **Operations** table to add *Count* to the **Output Column**.

c. In the **Function** column, select **count**.

d. In the **Input column position** column, select **ProductCategory**.
Your configuration should be as follows:



4. ADD A tHDFSOutput COMPONENT AND CONFIGURE THE TARGET FOLDER
Add a **tHDFSOutput** component. Configure it to use the **HDFSConnection** metadata and to write the results in the /user-/student/BDBasics/Hive/agg_results folder.

a. In the **Repository**, click **HDFSConnection** under Metadata/Hadoop cluster/TrainingCluster/HDFS.

b. Drag it to the **Designer**.

c. Select **tHDFSOutput** in the **Components** list, then click **OK**.

d. Connect **tAggregateRow** to **tHDFSOutput** with the **Main** row and then open the **Component** view.

e. In the **File Name** box, enter
"/user/student/BDBasics/Hive/agg_results".

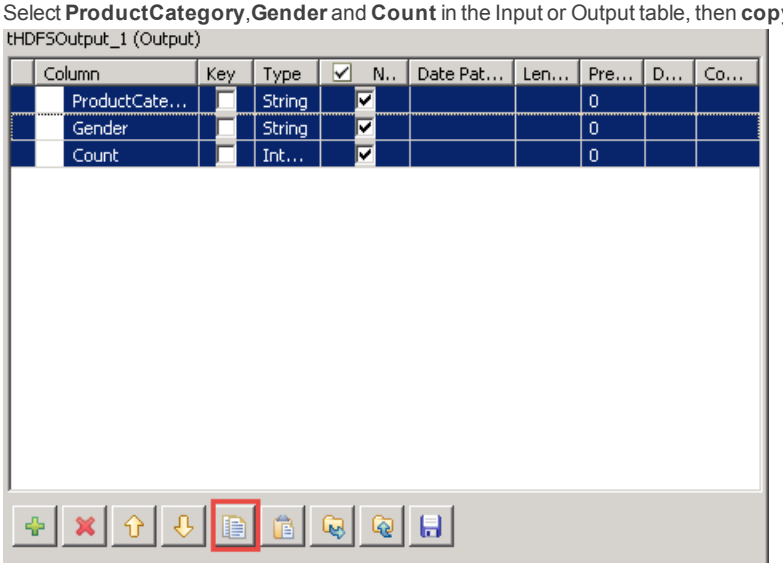
This will save the aggregation results in HDFS. The last step is to transfer the results in a Hive table.

Transfer results to Hive

1. COPY THE OUTPUT SCHEMA
In the tHDFSOuput component, copy the schema. It will be reused in a tHiveCreateTable component.

a. In the **tHDFSOutput Component** view, click **(...)** to edit the schema.

b. Select **ProductCategory**,**Gender** and **Count** in the Input or Output table, then **copy** the schema:



c. Close the **schema** window.

2. ADD A tHiveCreateTable COMPONENT AND CONNECT IT
   Add a **tHiveCreateTable** below **tHiveInput** and connect it with an **OnSubjobOk** trigger.

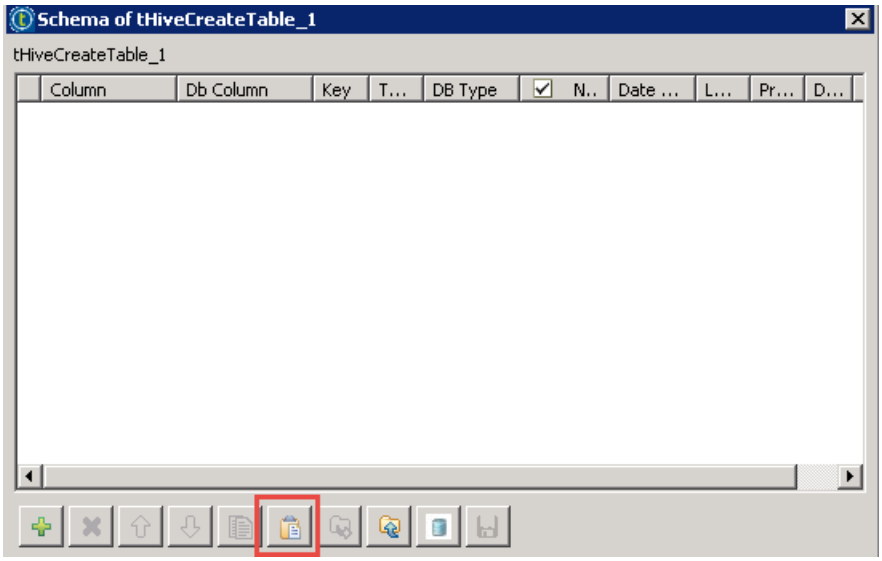
3. CONFIGURE tHiveCreateTableE
   Configure the tHiveCreateTable component to create a table named AggResults with the previously copied schema.

   a. Open the **Component** view.

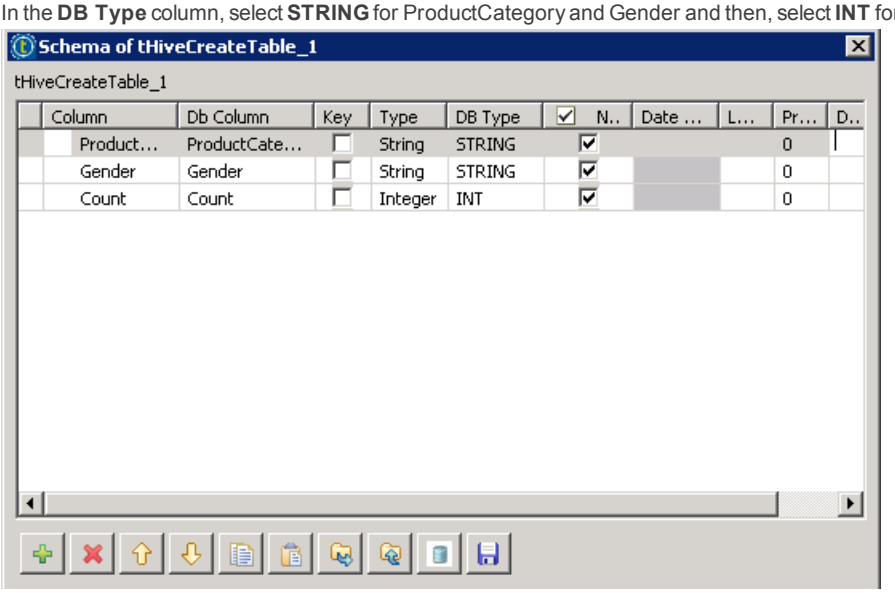
   b. Select the **Use an existing connection** option.

   c. Click **(...)** to edit the schema.

   d. Paste the Schema:



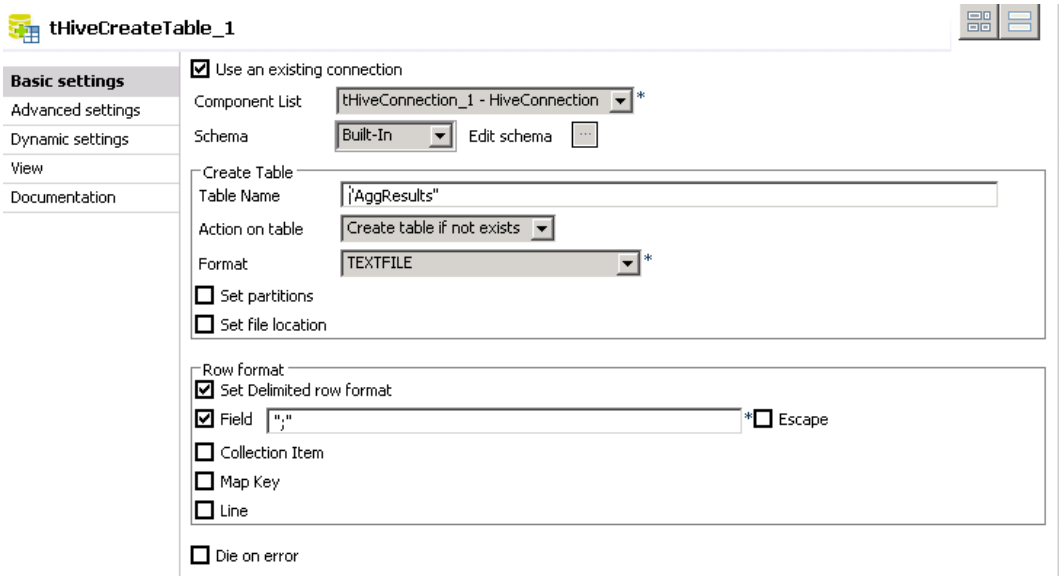
e. In the **DB Type** column, select **STRING** for ProductCategory and Gender and then, select **INT** for Count:



f. Click **OK** to save the schema.

g. In the **Table Name** box, enter *"AggResults"*.

h. In the **Action on table** list, select **Create table if not exists**.
Your configuration should be as follows:



4. ADD A tHiveRow COMPONENT
Add a **tHiveRow** component below **tHiveCreateTable** and connect it with an **OnSubjobOk** trigger.

5. CONFIGURE tHiveRow
In the tHiveRow component, write a query to populate the AggResults Hive table with the agg_results file.

a. In the **Component** view, select the **Use an existing connection** option.

b. Copy the schema in **tHiveCreateTable** and paste it in the schema of the **tHiveRow** component.

c. Click **OK** to save the schema.

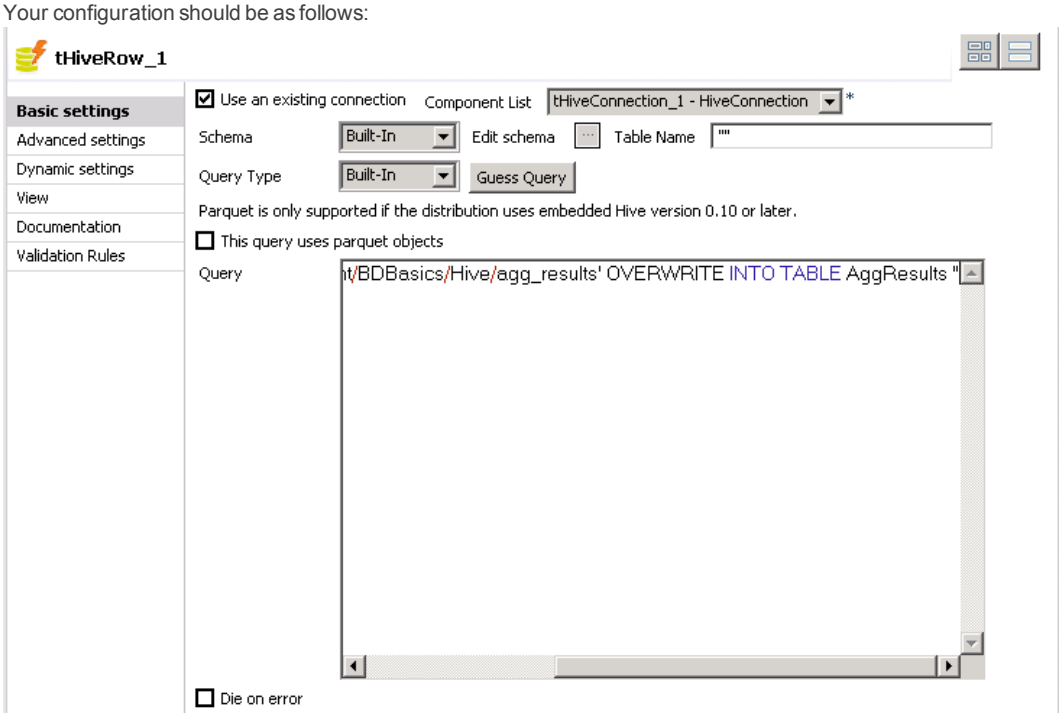
d. In the Query box, you will be able to write your own HiveQL.
The query in a **tHiveRow** component is executed at each flow iteration in your Job. In the current Job, the query will

be executed only once to transfer the data from the HDFS file to the **AggResults** Hive table.
In the **Query** box, enter:

```
"LOAD DATA INPATH
'/user/student/BDBasics/Hive/agg_results' OVERWRITE INTO TABLE AggResults "
```

Note: Copy and paste the Query from the LabCodeToCopy file in the C:\StudentFiles\BDBasics folder.

e. Your configuration should be as follows:



Your Job is now complete. It's time to run it and check the results.

Run the Job and verify the results

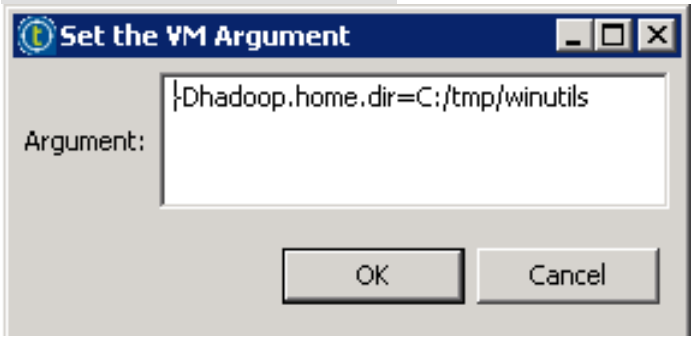
1. RUN YOUR JOB
   Run your Job and check the results in the **Console**:

```
[WARN ]: org.apache.hadoop.util.NativeCodeLoader - Unable to load native-hadoop
library for your platform... using builtin-java classes where applicable
[ERROR]: org.apache.hadoop.util.Shell - Failed to locate the winutils binary in the
hadoop binary path
java.io.IOException: Could not locate executable null\bin\winutils.exe in the
Hadoop binaries.
    at org.apache.hadoop.util.Shell.getQualifiedBinPath(Shell.java:355)
    at org.apache.hadoop.util.Shell.getWinUtilsPath(Shell.java:370)
    at org.apache.hadoop.util.Shell.<clinit>(Shell.java:363)
    at org.apache.hadoop.util.StringUtils.<clinit>(StringUtils.java:79)
    at org.apache.hadoop.security.Groups.parseStaticMapping(Groups.java:104)
    at org.apache.hadoop.security.Groups.<init>(Groups.java:86)
    at org.apache.hadoop.security.Groups.<init>(Groups.java:66)
    at
org.apache.hadoop.security.Groups.getUserToGroupsMappingService(Groups.java:280)
    at
org.apache.hadoop.security.UserGroupInformation.initialize(UserGroupInformation.jav
a:283)
    at
org.apache.hadoop.security.UserGroupInformation.ensureInitialized(UserGroupInformat
ion.java:260)
```

   If you did not include the extra JVM argument as explained in the first lab, you will see an error message regarding the winutils.exe binary not being available in the Hadoop binaries. In the current Job, this error won't prevent the Job to succeed

but it could in some cases.

2. FIX THE winutils.exe ERROR
   Using the Advanced settings tab, set the Hadoop home directory to the location of the winutils.exe file.
   a. In the **Run** view, click **Advanced settings**.
   b. Select the **Use specific JVM arguments** option.
   c. Click **New...**.
   d. In the **Argument** box, enter:
   `-Dhadoop.home.dir=C:/tmp/winutils`

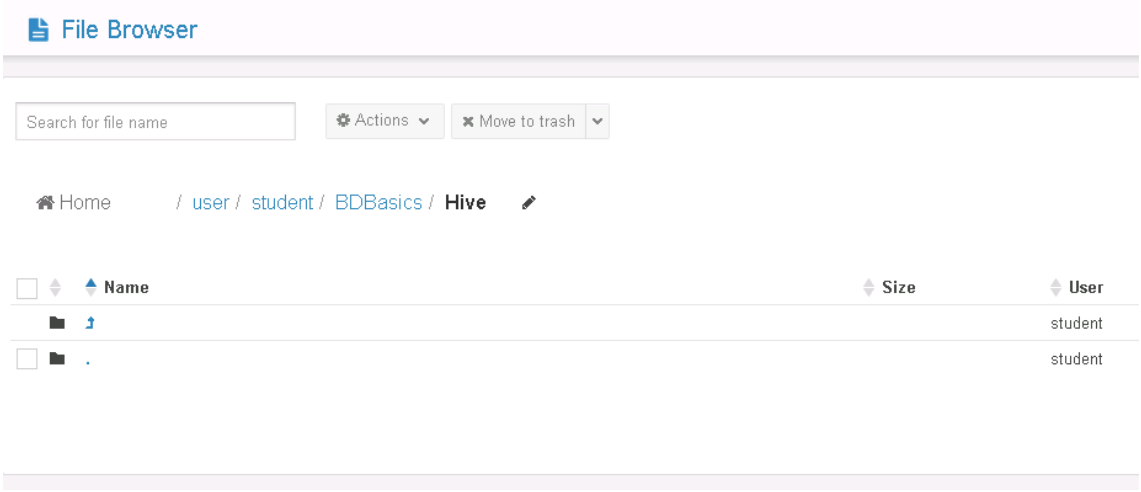


   e. Click **OK** to save the new argument and run your Job again.
   The Job should run successfully without error messages:

```
- Done.
[INFO ]: bdbasics.hiveprocessingjob_0_1.HiveProcessingJob - tHDFSOutput_1 - Start
to work.
[INFO ]: org.apache.hadoop.conf.Configuration.deprecation - fs.default.name is
deprecated. Instead, use fs.defaultFS
[WARN ]: org.apache.hadoop.util.NativeCodeLoader - Unable to load native-hadoop
library for your platform... using builtin-java classes where applicable
[INFO ]: bdbasics.hiveprocessingjob_0_1.HiveProcessingJob - tAggregateRow_1_AGGIN -
Start to work.
[INFO ]: bdbasics.hiveprocessingjob_0_1.HiveProcessingJob - tAggregateRow_1_AGGIN -
Retrieving the aggregation results.
[INFO ]: bdbasics.hiveprocessingjob_0_1.HiveProcessingJob - tAggregateRow_1_AGGIN -
Done.
[INFO ]: bdbasics.hiveprocessingjob_0_1.HiveProcessingJob - tHDFSOutput_1 - Written
records count: 14 .
[INFO ]: bdbasics.hiveprocessingjob_0_1.HiveProcessingJob - tHDFSOutput_1 - Done.
[INFO ]: bdbasics.hiveprocessingjob_0_1.HiveProcessingJob - tHiveCreateTable_1 -
Start to work.
[INFO ]: bdbasics.hiveprocessingjob_0_1.HiveProcessingJob - tHiveCreateTable_1 -
Done.
[INFO ]: bdbasics.hiveprocessingjob_0_1.HiveProcessingJob - tHiveRow_1 - Start to
work.
[INFO ]: bdbasics.hiveprocessingjob_0_1.HiveProcessingJob - tHiveRow_1 - Done.
[statistics] disconnected
[INFO ]: bdbasics.hiveprocessingjob_0_1.HiveProcessingJob - TalendJob:
'HiveProcessingJob' - Done.
Job HiveProcessingJob ended at 17:43 02/07/2015. [exit code=0]
```

3. Even if it's not clearly stated in the Console, the HiveQL query executed on the Cluster in the tHiveInput component,. You can see the Job generated in the **Hue Job Browser**:

4. Using the **Hue File Browser**, navigate to /user/student/BDBasics/Hive:



The folder is empty because the data have been transferred to the Hive table, deleting the file on HDFS.

5. In **Hue**, click **Data Browsers>Metastore Tables**. The table AggResults should be in the table list.
6. Click **AggResults**, then click the **Sample** tab:

Databases > default > aggresults

Columns | Sample | Properties

	productcategory	gender	count
0	clothing	M	6972
1	games	M	7168
2	movies	F	7190
3	electronics	F	7193
4	movies	M	7111
5	games	F	7172
6	shoes	M	7220
7	electronics	M	7162
8	handbags	F	6930
9	tools	F	7201
10	tools	M	7175
11	shoes	F	7170
12	handbags	M	7230
13	clothing	F	7106

You have processed your Hive Table with various components such as tHiveInput and tHiveRow. You will now process your Hive table using the Profiling perspective of the Studio.

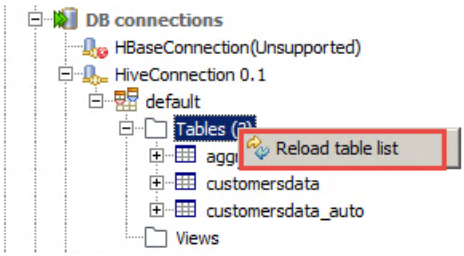
Profiling Hive Tables - Optional

Task outline

Using the Studio, you can run various analysis over your Hive tables. In this exercise, you will use the Profiling view of the Studio to run analysis of your Hive connection, Hive Tables, and columns.

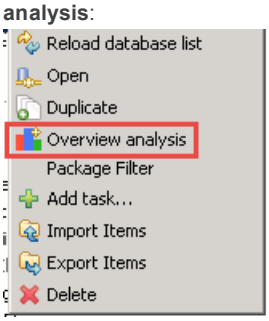
Hive connection analysis

1. SWITCH TO THE Profiling PERSPECTIVE
   In the upper-right corner of the Studio, click **Profiling** to switch to the **Profiling** perspective.
2. REFRESH THE HIVE TABLE LIST
   From the DQ Repository, reload the list of Hive tables.
   a. In the **DQ Repository**, under **Metadata/DB connections**, you will see your **HiveConnection**.
   Right-click **HiveConnection/default/Tables**, then click **Reload table list**:



   b. Click **Reload** in the **Reload pop up message**.
   This will update the table list :

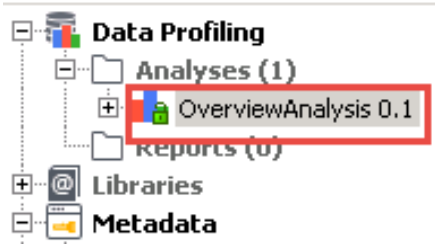
   As expected, you will find the AggResults, CustomersData, and CustomersData_auto Hive tables.
   Now, you will start the Overview Analysis of HiveConnection.
3. CREATE AN OVERVIEW TABLE
   Create an **Overview Analysis** on the **HiveConnection** metadata.
   a. Right-click **HiveConnection** under **DQ Repository/Metadata/DB connections**, then click **Overview analysis**:



   b. In the **Name** box, enter *OverviewAnalysis* and click **Next**.

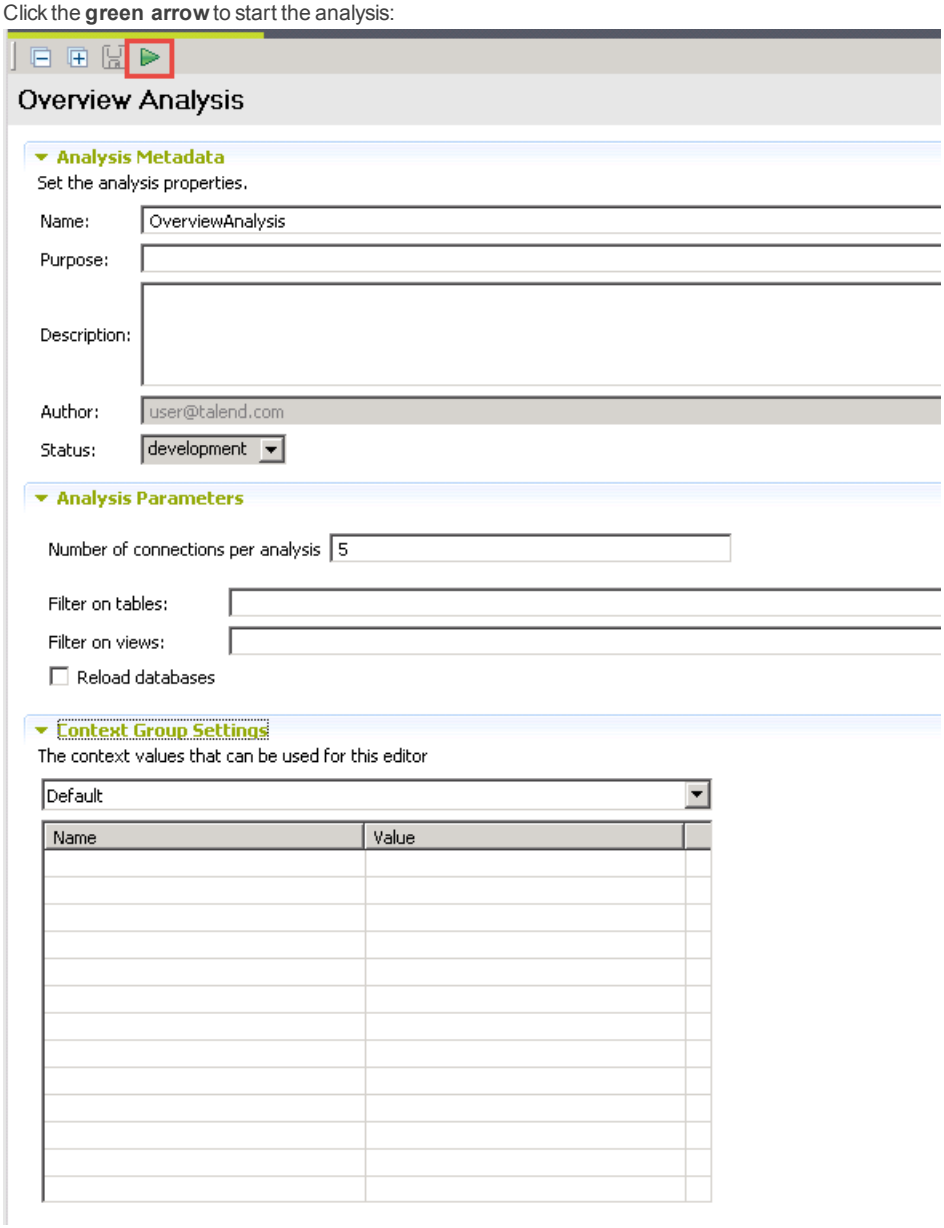
c. It is possible to filter the tables of interest by using the **Table name filter** and **View name filter**boxes. In this lab, you will keep these boxes empty.

d. Click **Finish**.
Your analysis appears in the DQ Repository, under **Data Profiling/Analyses**:

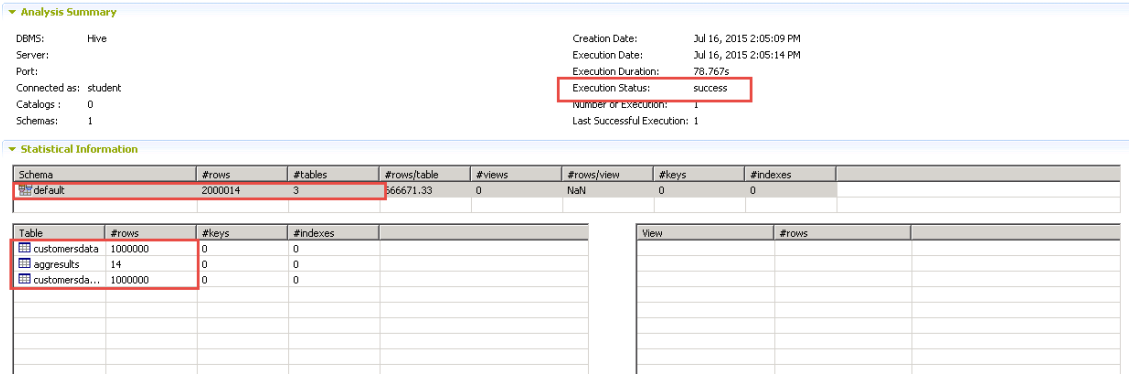


4. START THE ANALYSIS
The **OverviewAnalysis** opens so that you can examine the Analysis settings.

Click the **green arrow** to start the analysis:



5. EXAMINE THE RESULTS
   At the bottom of the **Connection Analysis** window, you will find the **Analysis summary**:



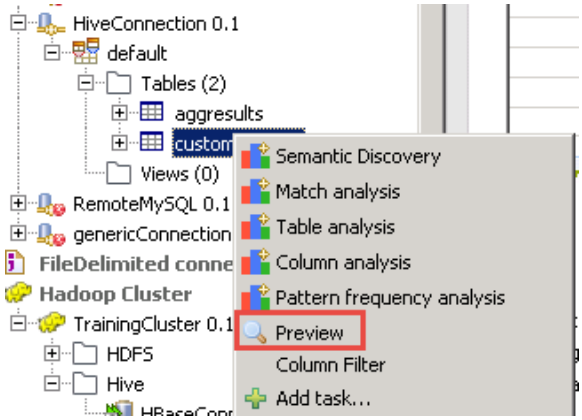
In the Analysis summary, you should see that the analysis was successful. There is 1 schema, named **default**. To get more details, click default. In the table below, you will get 3 tables, named **customersdata**, **customersdata_auto**, and **aggresults**.
As expected, CustomersData and CustomersData_auto have 1 million rows, and AggResults has 14 rows.

Now that you have an overview of your Hive Tables, you can move to the next step, which is to analyze each table.

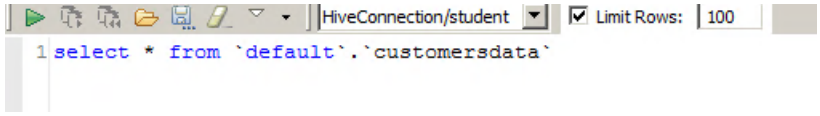
Hive tables analysis

In the Profiling view, you can easily refine the kind of analysis needed to suit your needs. You will now focus on the CustomersData table.

1. VISUALIZE THE customersdata HIVE TABLE
   From the DQ Repository, display the **customersdata** Hive table.
   a. Right-click **customersdata** under DQ Repository/Metadata/DB connections/HiveConnection/default/Tables, then click **Preview**:



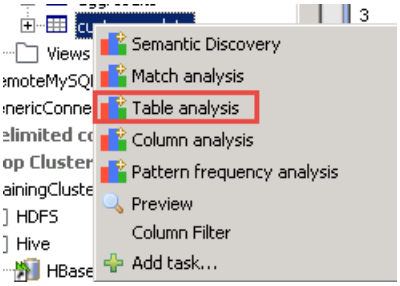
   b. This will open a **SQL Editor** window which is split in two parts.
   The upper part is the request submitted to preview your table:



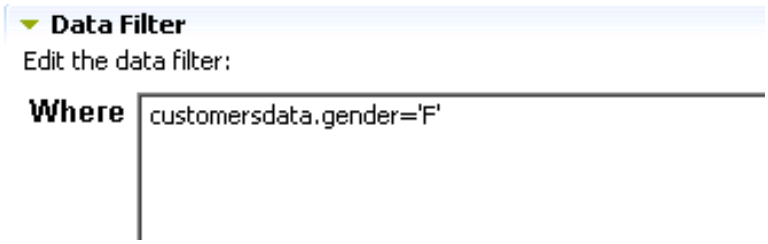
The bottom part is a preview of the CustomersData table:

customersdata.id	customersdata.firstname	customersdata.lastname	customersdata.city	customersdata.state	customersdata.productcategory	customersdata.gender	customersdata.purchas
1	Dwight	Washington	Nashville	Illinois	tools	M	<null>
2	Warren	Adams	Olympia	Hawaii	games	F	<null>
3	Woodrow	Kennedy	Juneau	Oklahoma	games	F	<null>
4	Gerald	Fillmore	Providence	Montana	clothing	F	<null>
5	Benjamin	Clinton	Des Moines	Michigan	shoes	F	<null>
6	Benjamin	Jefferson	Jefferson City	Montana	electronics	F	<null>
7	Richard	Johnson	Trenton	Illinois	games	F	<null>
8	Warren	Kennedy	Phoenix	Minnesota	movies	F	<null>
9	Woodrow	Jackson	Denver	South Dakota	movies	M	<null>
10	Calvin	Harrison	Raleigh	New York	electronics	F	<null>
11	Warren	Nixon	Austin	Montana	games	F	<null>
12	John	Quincy	Denver	South Carolina	clothing	F	<null>
13	Warren	Taft	Pierre	New Hampshire	handbags	F	<null>
14	Chester	Garfield	Juneau	Alaska	clothing	M	<null>
15	Abraham	Harrison	Boise	New Mexico	movies	M	<null>
16	Richard	Taylor	Olympia	Mississippi	clothing	F	<null>
17	Thomas	Van Buren	Richmond	Indiana	shoes	M	<null>
18	Ulysses	Cleveland	Columbus	Oklahoma	tools	M	<null>
19	Harry	Adams	Sacramento	Pennsylvania	clothing	M	<null>
20	Millard	Roosevelt	Columbia	Kansas	games	F	<null>
21	Harry	Kennedy	Charleston	New Hampshire	shoes	M	<null>
22	Theodore	McKinley	Annapolis	Delaware	clothing	F	<null>
23	Herbert	Kennedy	Pierre	Missouri	games	F	<null>
24	Millard	Eisenhower	Lincoln	Indiana	movies	F	<null>
25	Millard	Roosevelt	Sacramento	Michigan	clothing	M	<null>
26	Jimmy	Eisenhower	Albany	Kansas	movies	M	<null>
27	Millard	Adams	Topeka	New York	electronics	M	<null>

2. CREATE A TABLE ANALYSIS
   From the DQ Repository, create a Table analysis on the customersdata table. Add a filter to examine data corresponding to women.
   a. Right-click **customersdata** and click **Table analysis**.

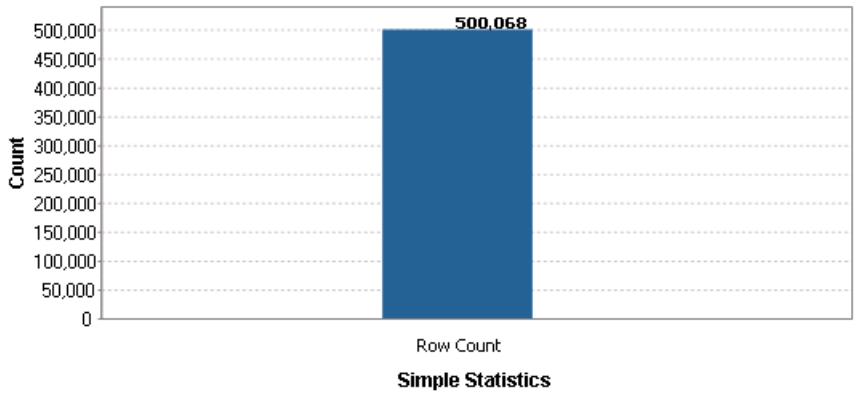


   b. In the **Name** box, enter *TableAnalysis* and click **Finish**.
   c. The **TableAnalysis** window opens. By default, a row count operation is proposed. You will add a filter to count the number of rows where the customer is a woman.
      In the **Where** box, in the **Data Filter** tab, enter *customersdata.gender='F'*:



   d. Click the **green arrow** icon to start the analysis.

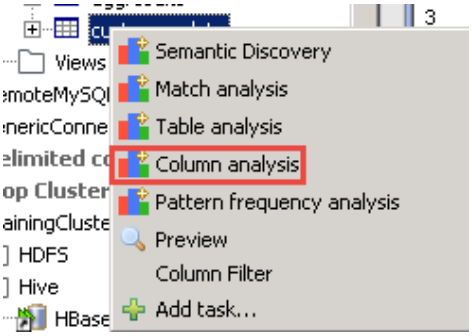
e. On the right side of the Table Analysis, you will find the result of your request:



You will continue to investigate your data, running analysis on some columns of the CustomersData table.

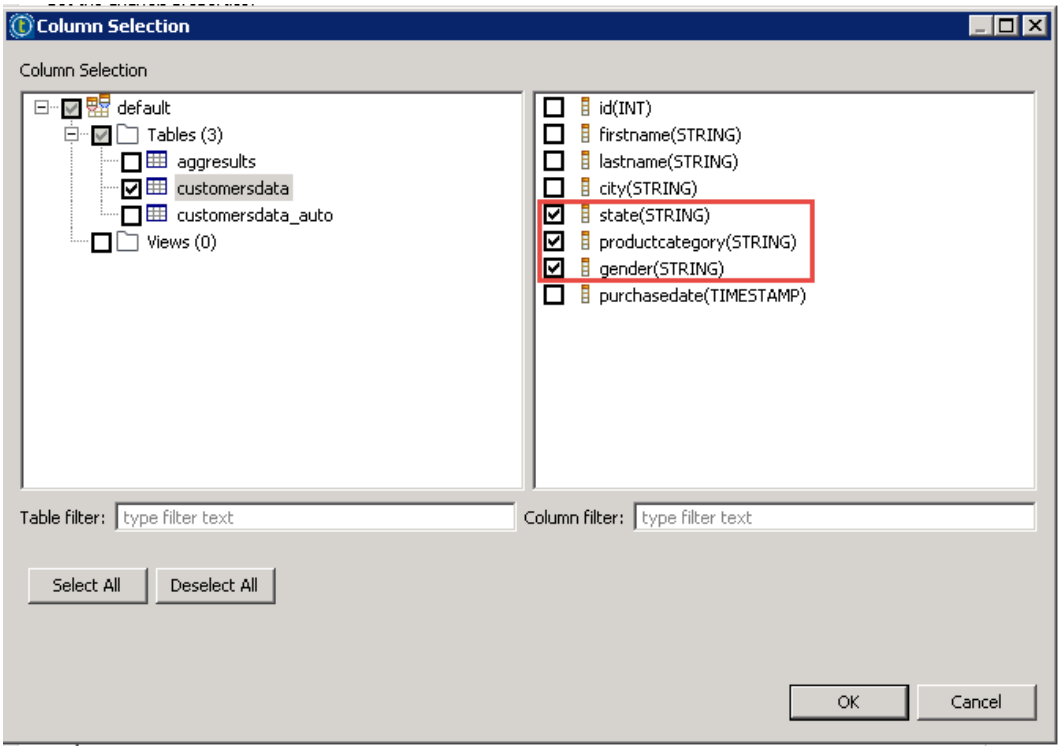
Column analysis (optional)

1. CREATE A COLUMN ANALYSIS
   From the DQ Repository, create a Column Analysis named *ColumnAnalysis*.
   a. Right-click **customersdata** under DQ Repository/Metadata/DB connections/HiveConnection/default/Tables, then click **Column Analysis**:



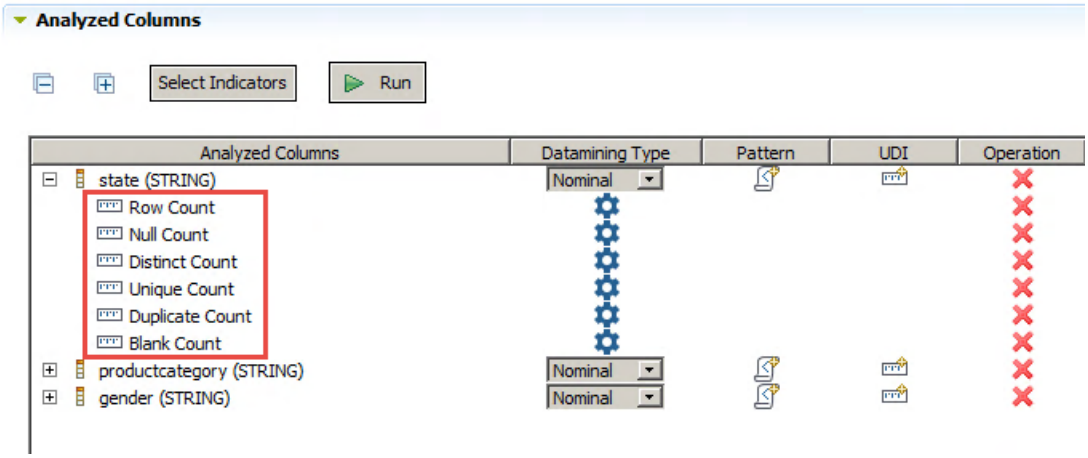
   b. In the **Name** box, enter *ColumnAnalysis*, then click **Finish**.
   The **Column Analysis** page opens.
2. CONFIGURE THE COLUMN ANALYSIS
   As the analysis can be time consuming, you will need to reduce the scope of the analysis. You will exclude the id, firstname, lastname, city, and purchasedate columns.

a. Under **Data preview**, click **Select Columns**, then select the **state**, **productcategory** and **gender** columns:



b. Click **OK** to save the selection.

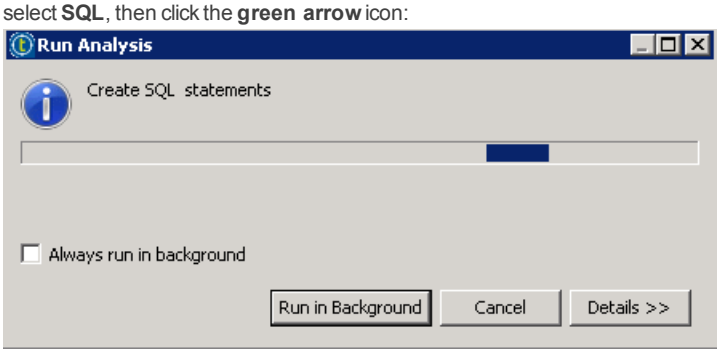
3. EXAMINE THE ANALYSIS CONFIGURATION
Under **Analyzed Columns** you will find state, productcategory and gender columns.
Click the **plus sign** next to **state** to view the details of what will be analyzed in the state column:



You will see basic information about your columns, such as the number of rows, the number of blank or null values, the number of distinct count.

4. CONFIGURE THE EXECUTION ENGINE
In the **Analysis Parameters** tab, you can choose the execution engine: SQL or Java. To send your requests to the cluster,

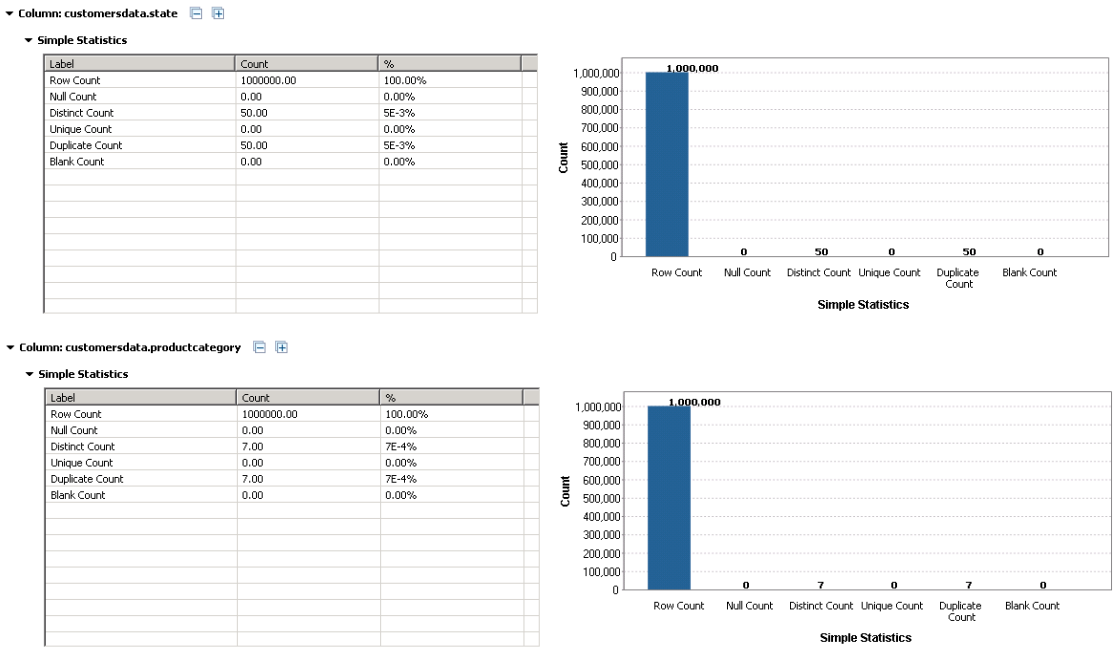
select **SQL**, then click the **green arrow** icon:

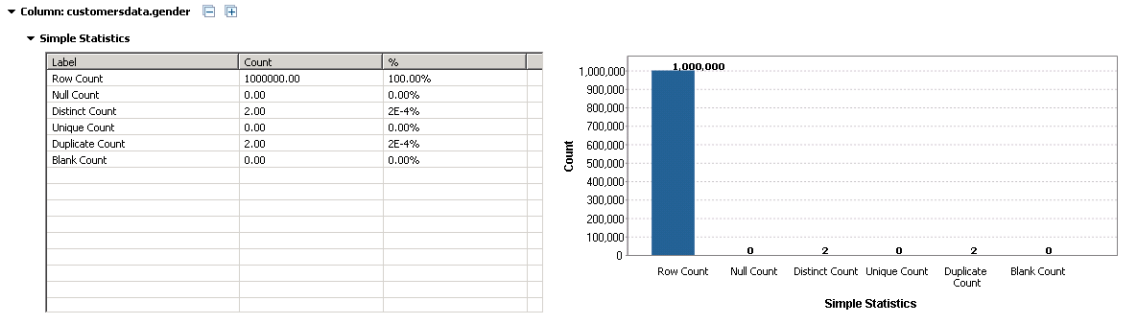


5. CONNECT TO Hue
While the analysis is running, you can go in to the **Hue Job Browser** and follow the execution of the Map and Reduce tasks launched by the Column Analysis:

Name	Status	User	Maps	Reduces	Queue
SELECT COUNT(state) FROM default.custom..."(Stage-1)	RUNNING	student	6%	6%	root.student
SELECT COUNT(*) FROM (SELECT state...myquery(Stage-2)	SUCCEEDED	student	100%	100%	root.student
SELECT COUNT(*) FROM (SELECT state...myquery(Stage-1)	SUCCEEDED	student	100%	100%	root.student
SELECT COUNT(*) FROM (SELECT state...myquery(Stage-2)	SUCCEEDED	student	100%	100%	root.student
SELECT COUNT(*) FROM (SELECT state...myquery(Stage-1)	SUCCEEDED	student	100%	100%	root.student
SELECT COUNT(*) FROM (SELECT DISTINCT st...A(Stage-2)	SUCCEEDED	student	100%	100%	root.student
SELECT COUNT(*) FROM (SELECT DISTINCT st...A(Stage-1)	SUCCEEDED	student	100%	100%	root.student

6. EXAMINE THE ANALYSIS RESULTS
At the end of the execution, open the **Analysis Results** tab. There you will find the results of each analysis of the **state**, **productcategory** and **gender** columns.



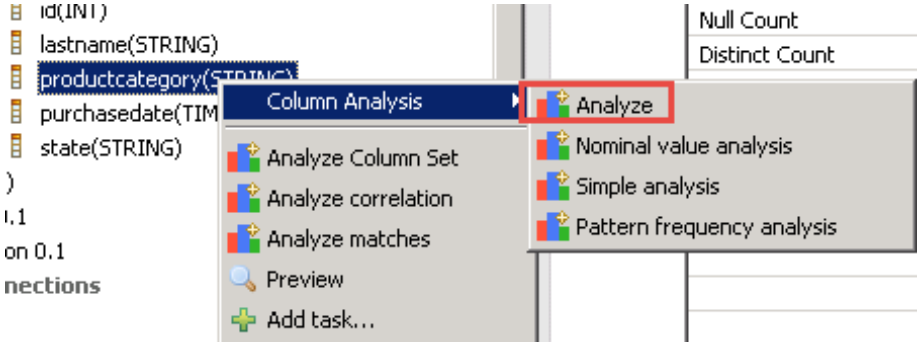


There are 7 distinct product categories. You will now run an analysis to list these values.

Product Category column analysis

1. CREATE A COLUMN ANALYSIS
   From the DQ Repository, create a Column Analysis over the **productcategory** column and name it *ProductCategoryAnalysis*.

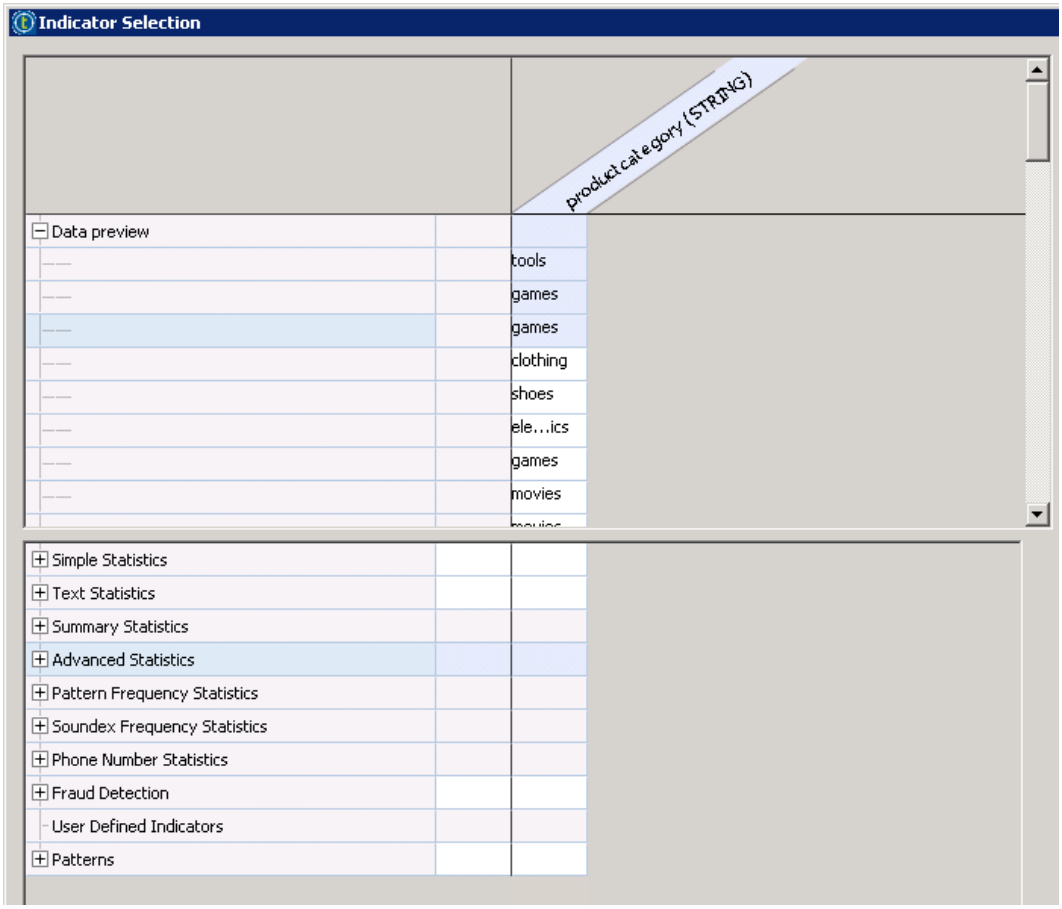
   a. Right-click **productcategory** under DQ Repository/Metadata/DB connections/HiveConnection/default/Tables/customersdata/Columns, and click**Column Analysis>Analyze**:



   b. In the **Name** box, enter *ProductCategoryAnalysis* and click **Finish**.

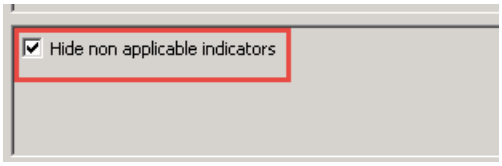
2. SELECT INDICATORS
   Select the Row Count and Value Frequency indicators.

a. In the **Column Analysis** window, under **Data preview**, click **Select Indicators**:

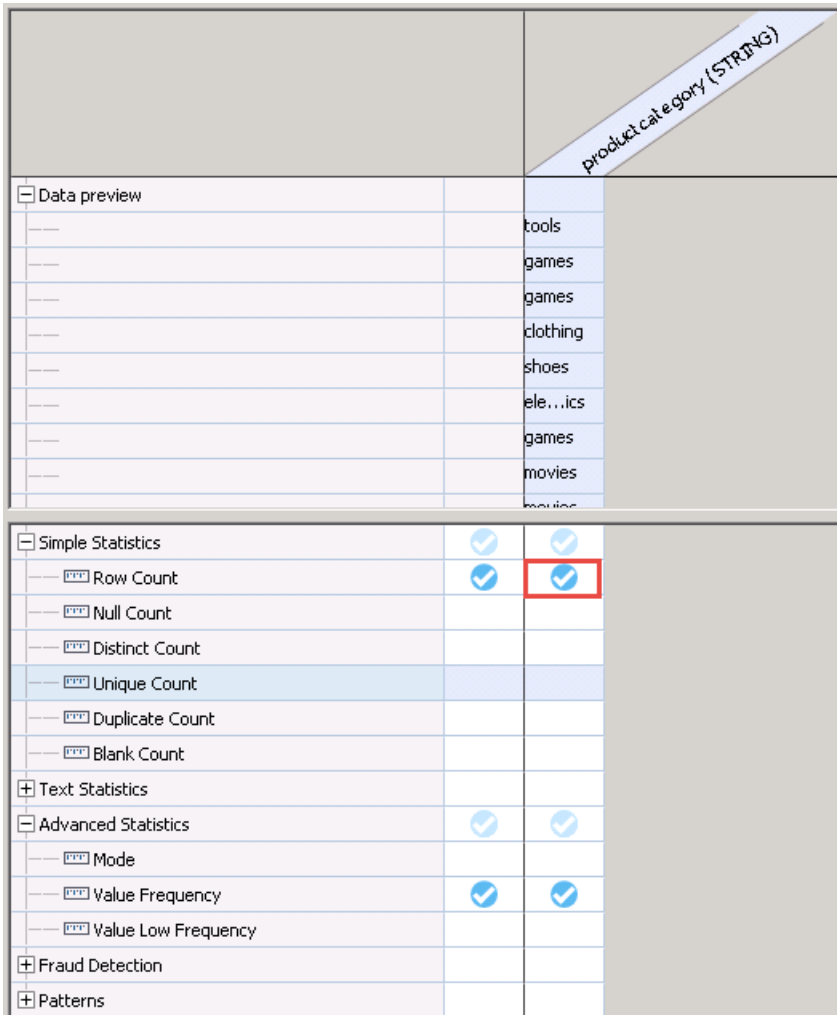


This is were you will specify which statistics you are interested in.

b. If you scroll down, you will see an option named **Hide non applicable indicators**. Select this option. This will simplify the indicators selection:

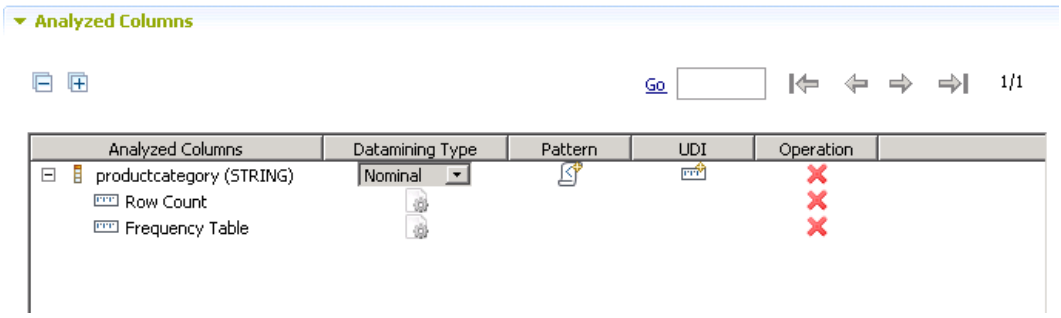


c. Expand **Simple Statistics** and select **Row Count** in the **productcategory** column:

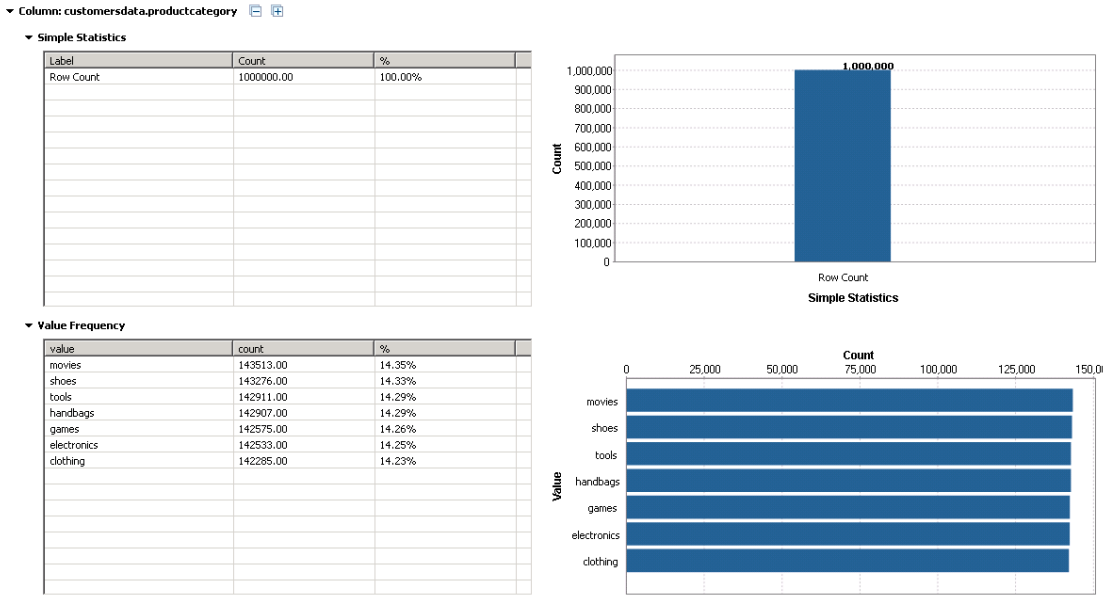


d. Expand **Advanced Statistics** and select **Value Frequency**.

e. Click **OK** to save your selection:



3. RUN THE ANALYSIS
   Click the **green arrow** to start the analysis.
4. EXAMINE THE ANALYSIS RESULTS
   Open the **Analysis Results** tab:



The frequency analysis lists the product categories values and the count for each value. The numbers are very close to each others because this data set has been randomly generated.

You have covered the last analysis for this exercise. Now it's time to move to the next exercise, where you will cover how to process data on HDFS using Pig.

Processing Data with Pig

Task outline

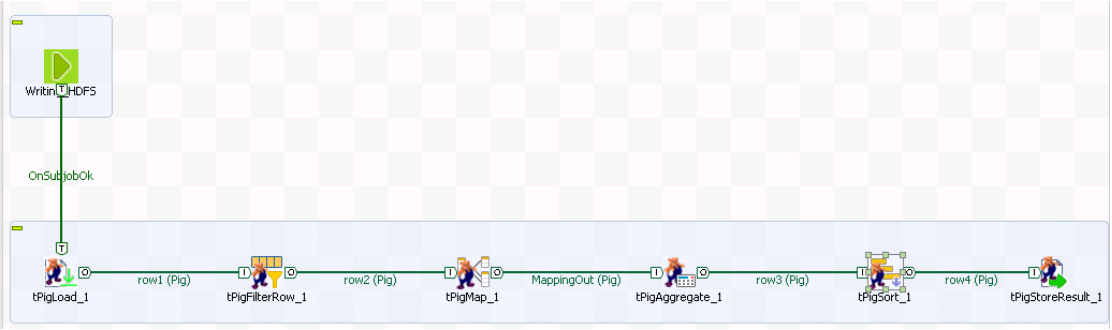
Map Reduce is very powerful but it requires a Java programmer, and the programmer may have to re-invent common functions such as joining or filtering. This is the motivation behind Pig.

Pig is a platform for analyzing large data sets. It consists of a high-level programming language, Pig Latin, that opens Hadoop to non-Java programmers. Pig Latin also provides common operations to group, filter, join, or sort data.

Pig provides an execution engine on top of Hadoop.
The Pig Latin script is converted to Map Reduce code, which will be executed in your cluster.

In this lab, you will process the Customers data stored in HDFS. You will perform basic tasks such as filtering rows, mapping, aggregating and sorting data, and storing your results in HDFS.

At the end of this lab, your Job will look like the following:



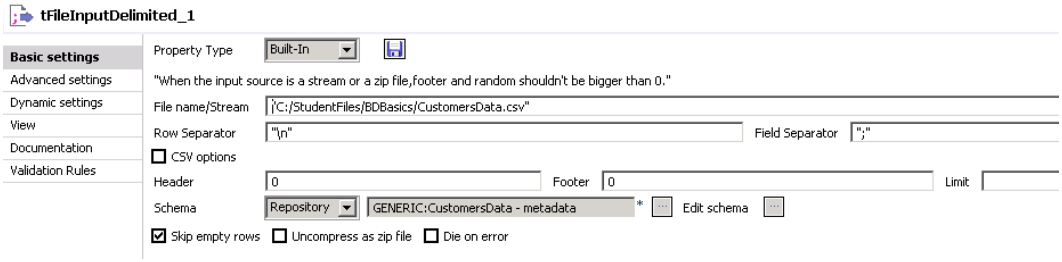
Writing data to HDFS

Earlier in the course, you used the **tHDFSPut** component to copy a file from your local file system and paste it in HDFS. Another way to write data to HDFS is to use the **tHDFSOutput** component, which writes a data flow in HDFS.

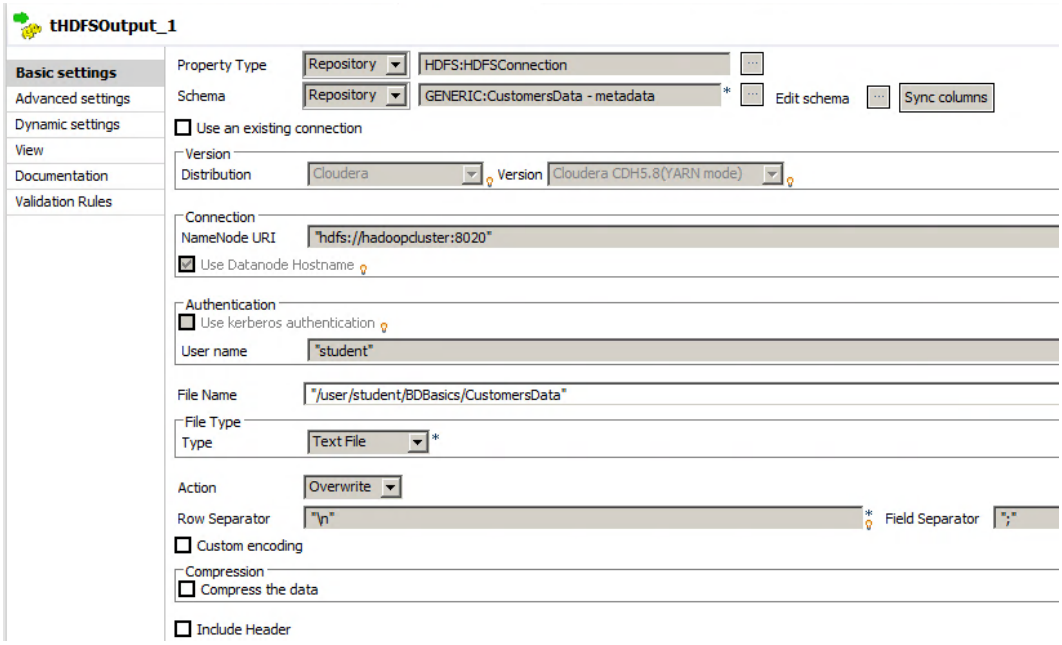
You will create a Job that reads the CustomersData.csv file and writes it to HDFS using the **tHDFSOutput** component.

1. SWITCH TO THE Integration PERSPECTIVE
   In the upper-right corner of the Studio, click **Integration** to switch your Studio to the **Integration** perspective.
2. CREATE A NEW STANDARD JOB
   Create a new **Standard Job** and name it *WritingHDFS*.
3. ADD A tFileInputDelimited COMPONENT
   Add a **tFileInputDelimited** component and open the **Component** view.
4. READ THE INPUT FILE
   Configure the tFileInputDelimited component to read the CustomersData.csv file.
   a. Next to the**File name** box, click**(...)**and navigate to *"C:\StudentFiles\BDBasics\CustomersData.csv"*.
   b. Set the **Row Separator** to *"\n"* and the **Field Separator** to *";"*.

c. Set the **Schema** type to **Repository** and browse to find the **CustomersData** generic schema metadata.
Your configuration should be as follows:



5. ADD A tHDFSOutput COMPONENT
Add a **tHDFSOutput** component and connect it with the **Main** row.
6. CONFIGURE THE OUTPUT FOLDER
Configure tHDFSOutput to write in the */user/student/BDBasics/CustomersData* folder.
    a. Double-click the **tHDFSOutput** component to open the **Component** view.
    b. In the **Property Type** list, select **Repository**. Then, find **HDFSConnection**.
    c. In the **Schema** list, select **Repository**. Then, find the **CustomersData** generic schema metadata.
    d. In the **File Name** box, enter
    "/user/student/BDBasics/CustomersData".
    Your configuration should be as follows:



    e. **Save** your Job.
    This Job will be executed later from another Job, using a tRunJob component.

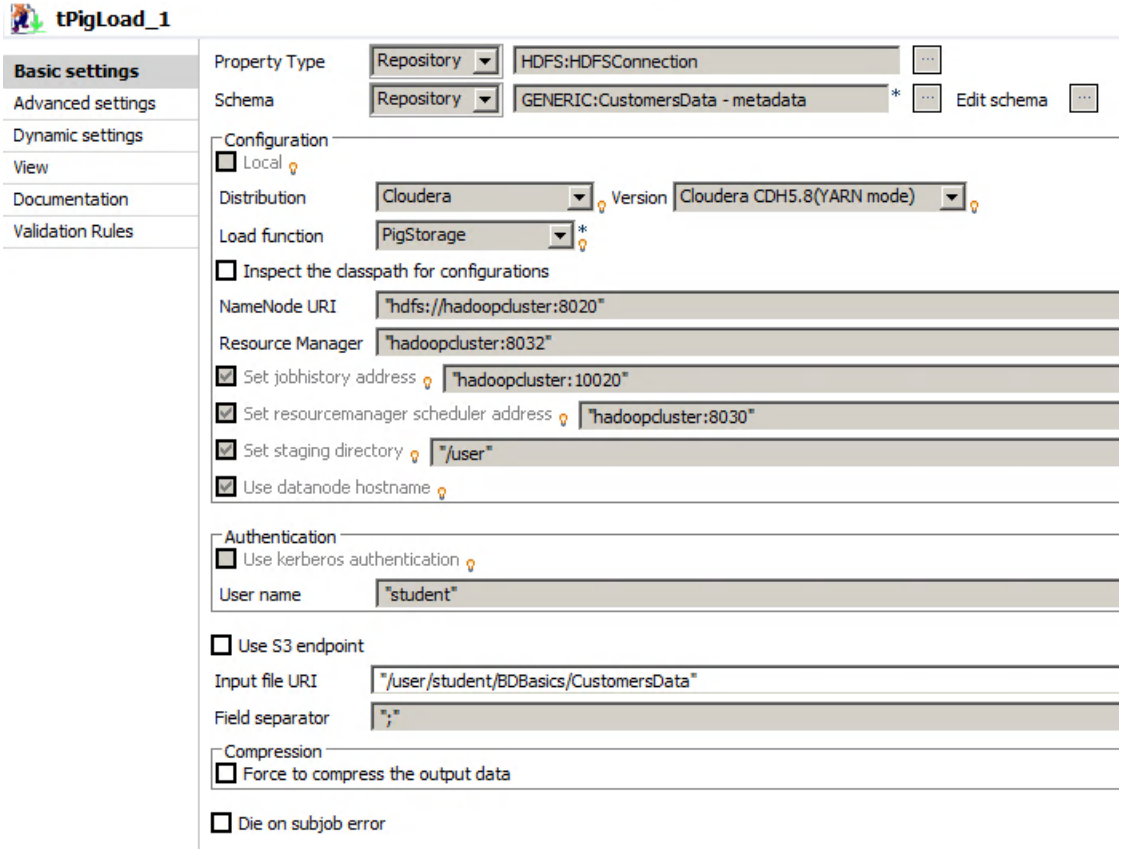
Load data

You will create a new Job named **PigProcessing**, which will process the customers' data.

1. CREATE A NEW STANDARD JOB
Create a a new **Standard Job** and name it *PigProcessing*.
2. ADD THE WritingHDFS JOB
Drag the **WritingHDFS** Job from the **Repository** and drop it in the **Designer**. It will appear as a **tRunJob** component

labelled WritingHDFS:

3. ADD A tPigLoad COMPONENT
   Add a **tPigLoad** component and Connect the **WritingHDFS** component to **tPigLoad** with an **OnSubjobOk** trigger
4. SELECT THE EXECUTION MODE
   Double-click **tPigLoad** to open the **Component** view.
   Pig Jobs can be executed in local or Map/Reduce mode. This is configured in the **tPigLoad** component. For this lab, you will use the Map/Reduce mode.
5. CONFIGURE THE CONNECTION TO HDFS
   As you did for the **tHDFSOutput** component, set the **Property Type** to **Repository** using the **HDFSConnection** metadata.
6. CONFIGURE THE SCHEMA
   Set the **Schema** type to **Repository** using the **CustomersData** generic schema metadata.
7. CONFIGURE THE INPUT FILE
   In the **Input file URI** box, enter *"/user/student/BDBasics/CustomersData"*.
   Your configuration should be as follows:

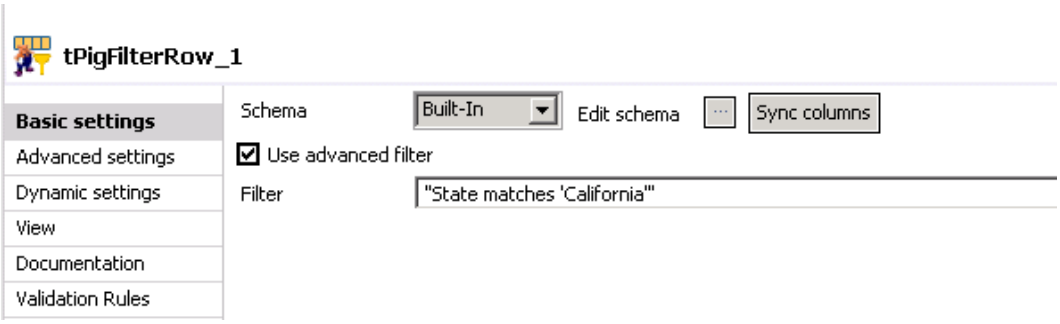


Filter and map Data

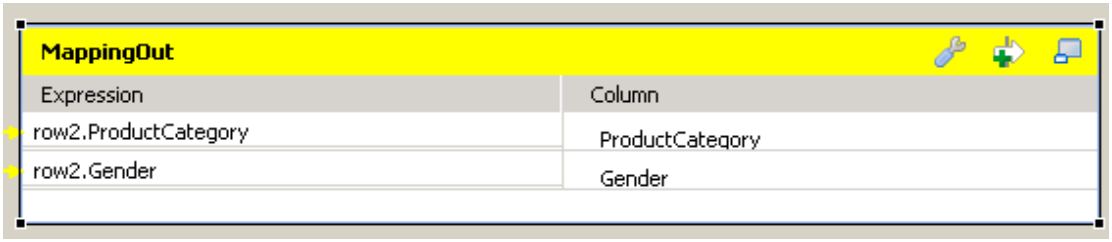
You will continue to build your Job with the next two components, in order to filter and map your data. The goal here is to extract customers living in California and get the corresponding gender and product category.

You will use the **tPigFilterRow** component to filter the State. Then, you will use the **tPigMap** component to extract the data from the Gender and ProductCategory columns.

1. ADD A tPigFilterRow COMPONENT
   Add a **tPigFilterRow** and connect with a **Pig Combine** row.
2. CONFIGURE THE FILTERING
   Add a filter to select customers living in California.
   a. Open the **Component** view.
   b. Select the **Use advanced filter** option.
   c. In the **Filter** box, enter *"State matches 'California'"*:



3. ADD A tPigMap COMPONENT AND CONNECT IT
   Add a **tPigMap** component, connect it with the **Pig Combine** row, and then open the **Component** view.
4. CREATE THE MAPPING OUTPUT
   Create a new output named *MappingOut*.
5. CONFIGURE THE MAPPING
   Select **ProductCategory** and **Gender** columns in the **row2** table and drag in the **MappingOut** table.
   Your mapping should be as follows:



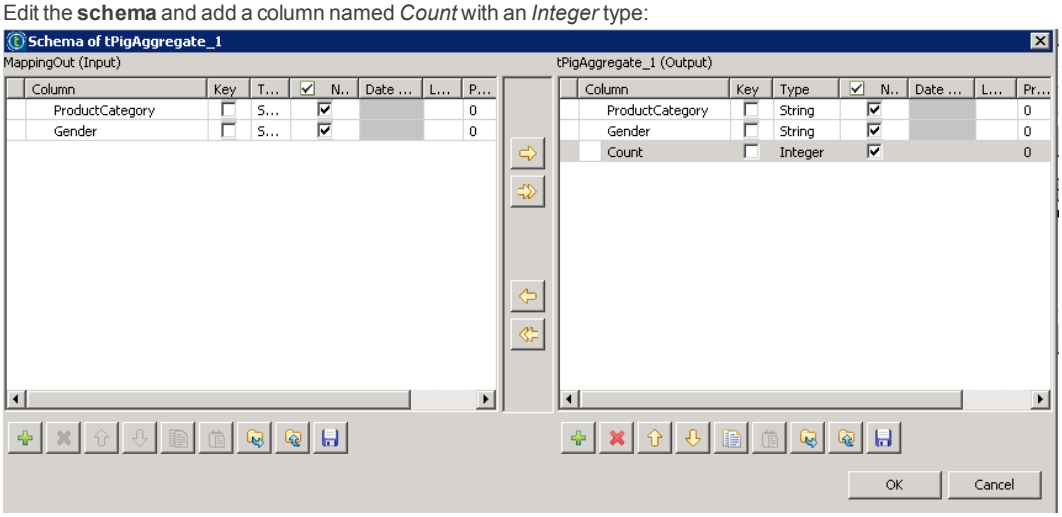
6. SAVE THE MAPPING
   Click **OK** to save the mapping.

Aggregate and sort data

Now that you have extracted the data, you will aggregate and sort it. The goal here is to have the number of men and women per ProductCategory, and then sort them by alphabetical order.

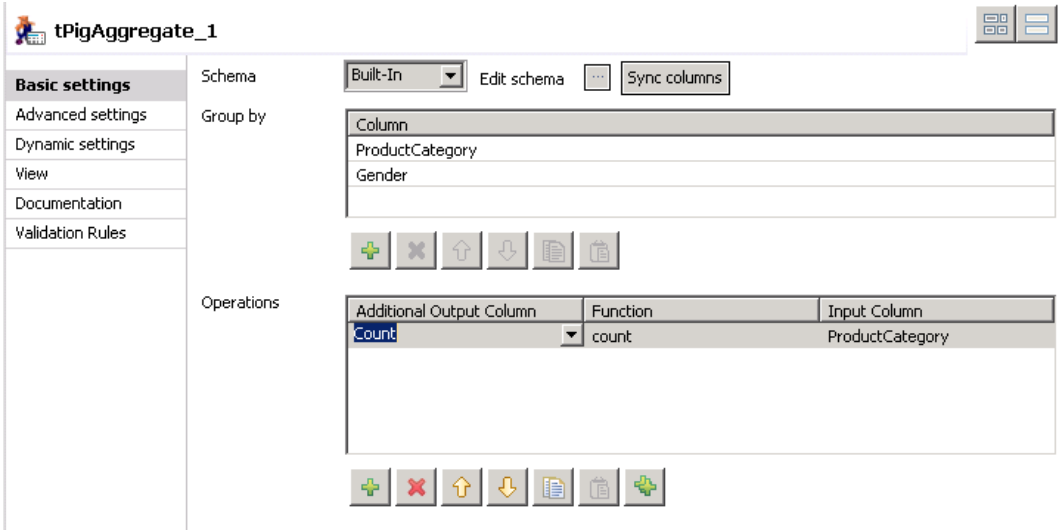
1. ADD A TPIGAGGREGATE COMPONENT
   Add a **tPigAggregate** component, and connect it with the **MappingOut** row. Then, open the **Component** view.
2. CONFIGURE THE OUTPUT SCHEMA
   Add a new column to the output table, named Count which is an Integer.

a. Edit the **schema** and add a column named *Count* with an *Integer* type:



b. Click **OK** to save the schema.

3. CONFIGURE THE AGGREGATION
Configure the **Group by** and **Operations** tables to aggregate your data by **ProductCategory** and **Gender**.
   a. Click the **green plus sign** below the **Group by** table to add the **ProductCategory** and **Gender** columns.
   b. Click the **green plus sign** below the **Operations** table.
   c. In the **Additional Output Column** list, select **Count**.
   d. In the **Function** list, select **count**.
   e. In the **Input Column** list, select **ProductCategory**.
   Your configuration should be as follows:

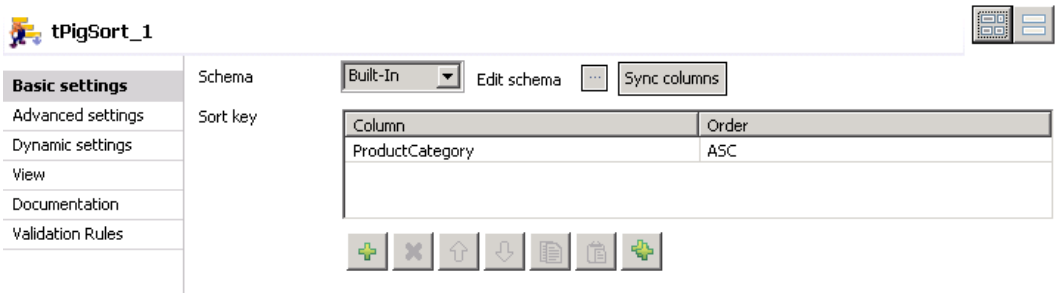


4. ADD A tPigSort COMPONENT
Add a **tPigSort** component, connect it with the **Pig Combine** row, and then, open the **Component** view.

5. CONFIGURE THE SORTING
Configure the Sort key table to sort ProductCategory in Ascending order.

a. Click the **green plus sign** below the **Sort key** table.

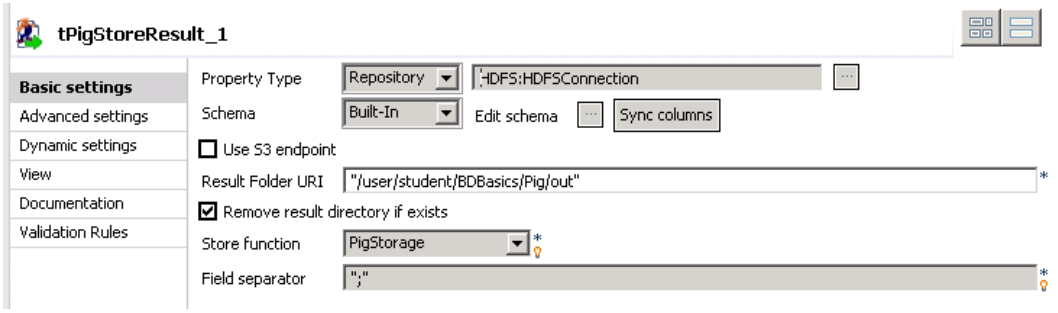
b. Configure to sort the **ProductCategory** column by ascending order, as shown below:



Store results

Once processed, the last step is to store your results on HDFS.

1. ADD A tPigStoreResult COMPONENT
   Add a **tPigStoreResult** component, connect it with the **Pig Combine** row, and then, open the **Component** view.
2. USE THE HDFSConnection METADATA
   Set the **Property Type** to **Repository** and then select **HDFSConnection**.
3. CONFIGURE THE OUTPUT FOLDER
   Write the results in the */user/student/BDBasics/Pig/out* folder.
   a. In the **Result Folder URI**, enter *"/user/student/BDBasics/Pig/out"*.
   b. Select the **Remove directory if exists** option.
   This will allow you to run the Job again as needed.
   c. Your configuration should be as follows:



Run the job and verify the results

1. RUN YOUR JOB AND VERIFY THE RESULTS
   Run the Job and inspect the results in the **Console**.
   a. Scrolling down the Console, you will find information about the WritingHDFS Job:

```
[INFO ]: bdbasics.writinghdfs_0_1.WritingHDFS - tFileInputDelimited_1 - Start to work.
[INFO ]: bdbasics.writinghdfs_0_1.WritingHDFS - tFileInputDelimited_1 - Retrieving records from the
datasource.
[INFO ]: bdbasics.writinghdfs_0_1.WritingHDFS - tFileInputDelimited_1 - Retrieved records count: 1000000.
[INFO ]: bdbasics.writinghdfs_0_1.WritingHDFS - tFileInputDelimited_1 - Done.
[INFO ]: bdbasics.writinghdfs_0_1.WritingHDFS - tHDFSOutput_1 - Written records count: 1000000 .
[INFO ]: bdbasics.writinghdfs_0_1.WritingHDFS - tHDFSOutput_1 - Done.
[INFO ]: bdbasics.pigprocessing_0_1.PigProcessing - tRunJob_1 - The child job
bdbasics.writinghdfs_0_1.WritingHDFS' is done.
[INFO ]: bdbasics.pigprocessing_0_1.PigProcessing - tRunJob_1 - Done.
[INFO ]: bdbasics.pigprocessing_0_1.PigProcessing - tPigStoreResult_1 - Start to work.
```

   The WritingHDFS Job succesfully executes and then the Pig components will start to work.

b. If you continue to investigate in the logs, you will find the Pig requests equivalent to each Pig component. The first Pig component is the **tPigLoad**:

```
[INFO ]: org.apache.hadoop.conf.Configuration.deprecation - fs.default.name is deprecated. Instead, use
fs.defaultFS
[INFO ]: bdbasics.pigprocessing_0_1.PigProcessing - tPigLoad_1 - register query : tPigLoad_1_row1_RESULT =
LOAD '/user/student/BDBasics/CustomersData' using PigStorage(';') AS (Id:int, FirstName:chararray,
LastName:chararray, City:chararray, State:chararray, ProductCategory:chararray, Gender:chararray,
PurchaseDate:chararray);
[INFO ]: bdbasics.pigprocessing_0_1.PigProcessing - tPigFilterRow_1 - register query :
tPigFilterRow_1_row2_RESULT = FILTER tPigLoad_1_row1_RESULT BY State matches 'California';
[INFO ]: bdbasics.pigprocessing_0_1.PigProcessing - tPigMap_1 - register query : tPigMap_1_row2_RESULT =
FOREACH tPigFilterRow_1_row2_RESULT GENERATE *;
[INFO ]: bdbasics.pigprocessing_0_1.PigProcessing - tPigMap_1 - register query : tPigMap_1_RESULT =
```

c. Right after, you will find Pig requests for **tPigFilterRow**, **tPigMap**, **tPigAggregate**, **tPigSort** and **tPigStoreResult**:

```
[INFO ]: bdbasics.pigprocessing_0_1.PigProcessing - tPigLoad_1 - register query : tPigLoad_1_row1_RESULT
LOAD '/user/student/BDBasics/CustomersData' using PigStorage(';') AS (Id:int, FirstName:chararray,
LastName:chararray, City:chararray, State:chararray, ProductCategory:chararray, Gender:chararray,
PurchaseDate:chararray);
[INFO ]: bdbasics.pigprocessing_0_1.PigProcessing - tPigFilterRow_1 - register query :
tPigFilterRow_1_row2_RESULT = FILTER tPigLoad_1_row1_RESULT BY State matches 'California';
[INFO ]: bdbasics.pigprocessing_0_1.PigProcessing - tPigMap_1 - register query : tPigMap_1_row2_RESULT =
FOREACH tPigFilterRow_1_row2_RESULT GENERATE *;
[INFO ]: bdbasics.pigprocessing_0_1.PigProcessing - tPigMap_1 - register query : tPigMap_1_RESULT =
FOREACH tPigMap_1_row2_RESULT GENERATE *;
[INFO ]: bdbasics.pigprocessing_0_1.PigProcessing - tPigMap_1 - register query :
tPigMap_1_MappingOut_RESULT = FOREACH tPigMap_1_RESULT GENERATE *;
[INFO ]: bdbasics.pigprocessing_0_1.PigProcessing - tPigMap_1 - register query :
tPigMap_1_MappingOut_RESULT = FOREACH tPigMap_1_MappingOut_RESULT GENERATE $5 AS ProductCategory,$6 AS
Gender;
[INFO ]: bdbasics.pigprocessing_0_1.PigProcessing - tPigAggregate_1 - register query :
tPigAggregate_1_GROUP = GROUP tPigMap_1_MappingOut_RESULT BY (ProductCategory,Gender);
[INFO ]: bdbasics.pigprocessing_0_1.PigProcessing - tPigAggregate_1 - register query :
tPigAggregate_1_row3_RESULT = FOREACH tPigAggregate_1_GROUP GENERATE group.ProductCategory AS
ProductCategory,group.Gender AS Gender, COUNT(tPigMap_1_MappingOut_RESULT.ProductCategory) AS Count;
[INFO ]: bdbasics.pigprocessing_0_1.PigProcessing - tPigSort_1 - register query : tPigSort_1_row4_RESULT
ORDER tPigAggregate_1_row3_RESULT BY ProductCategory ASC;
[INFO ]: bdbasics.pigprocessing_0_1.PigProcessing - tPigStoreResult_1 - register query : STORE
tPigSort_1_row4_RESULT INTO '/user/student/BDBasics/Pig/out' using PigStorage(';');
[INFO ]: org.apache.pig.tools.pigstats.ScriptState - Pig features used in the script:
GROUP_BY,ORDER_BY,FILTER
[INFO ]: org.apache.pig.newplan.logical.optimizer.LogicalPlanOptimizer - {RULES_ENABLED=[AddForEach,
ColumnMapKeyPrune, DuplicateForEachColumnRewrite, FilterLogicExpressionSimplifier,
GroupByConstParallelSetter, ImplicitSplitInserter, LimitOptimizer, LoadTypeCastInserter, MergeFilter,
```

d. If you continue to scroll down, you will see several MapReduce Jobs submitted, as well as their statistics. This is the final report of all Map Reduce Jobs execution:

```
HadoopVersion   PigVersion  UserId  StartedAt   FinishedAt  Features
2.6.0-cdh5.8.1  0.12.0-cdh5.8.1 Administrator   2017-01-09 02:30:32 2017-01-09 02:31:46 GROUP_BY,ORDER_BY,FILTER

Success!

Job Stats (time in seconds):
JobId   Maps    Reduces MaxMapTime  MinMapTIme  AvgMapTime  MedianMapTime   MaxReduceTime   MinReduceTime   AvgReduceTime   Med:
ure Outputs
job_1480946808137_0141  1   1   4   4   4   2   2   2   2   tPigAggregate_1_GROUP,tPigAggregate_1_row3_RESULT,tPigFilterRow_
_RESULT,tPigMap_1_MappingOut_RESULT GROUP_BY,COMBINER
job_1480946808137_0142  1   1   2   2   2   2   2   2   2   tPigSort_1_row4_RESULT  SAMPLER
job_1480946808137_0143  1   1   2   2   2   2   2   2   2   tPigSort_1_row4_RESULT  ORDER_BY    /user/student/BDBasics/Pig/

Input(s):
Successfully read 1000000 records (61788497 bytes) from: "/user/student/BDBasics/CustomersData"

Output(s):
Successfully stored 14 records (208 bytes) in: "/user/student/BDBasics/Pig/out"

Counters:
Total records written : 14
Total bytes written : 208
Spillable Memory Manager spill count : 0
Total bags proactively spilled: 0
Total records proactively spilled: 0

Job DAG:
job_1480946808137_0141  ->  job_1480946808137_0142,
job_1480946808137_0142  ->  job_1480946808137_0143,
job_1480946808137_0143
```

There, you can see that the execution was successful. First, 1 million rows were read from the CustomersData file. Then, 14 records were written in the /user/student/BDBascis/Pig/out folder.

2. In **Hue**, using the **File Browser**, navigate to */user/student/BDBascis/Pig/out*.

3. Click the **part-r-00000** file to see the result:


```
clothing;M;1464
clothing;F;1410
electronics;F;1416
electronics;M;1486
games;F;1508
games;M;1494
handbags;M;1396
handbags;F;1452
movies;M;1400
movies;F;1475
shoes;M;1460
shoes;F;1475
tools;M;1505
tools;F;1460
```

4. In **Hue**, in the **Job Browser**, you will be able to see all the Jobs submitted by the Studio. The Jobs have different IDs but they have the same name: BDBASICS_PigProcessing_0.1_tPigLoad_1.
All your Jobs have succeeded:

Logs	ID	Name	Status	User	Maps	Reduces
	1435742073748_0015	BDBASICS_PigProcessing_0.1_tPigLoad_1	SUCCEEDED	student	100%	100%
	1435742073748_0014	BDBASICS_PigProcessing_0.1_tPigLoad_1	SUCCEEDED	student	100%	100%
	1435742073748_0013	BDBASICS_PigProcessing_0.1_tPigLoad_1	SUCCEEDED	student	100%	100%
	1435742073748_0012	BDBASICS_PigProcessing_0.1_tPigLoad_1	SUCCEEDED	student	100%	100%
	1435742073748_0011	BDBASICS_PigProcessing_0.1_tPigLoad_1	SUCCEEDED	student	100%	100%
	1435742073748_0010	BDBASICS_PigProcessing_0.1_tPigLoad_1	SUCCEEDED	student	100%	100%
	1435742073748_0009	BDBASICS_PigProcessing_0.1_tPigLoad_1	SUCCEEDED	student	100%	100%
	1435742073748_0008	BDBASICS_PigProcessing_0.1_tPigLoad_1	SUCCEEDED	student	100%	100%

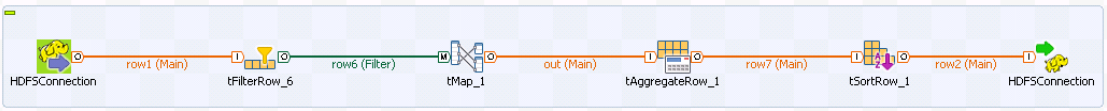
You can now continue investigating processing data on HDFS and move to the next exercise.

Processing Data with Big Data Batch Job

Task outline

The last way to process data covered in this course is to use Big Data Batch Jobs. First, you will create a Standard Job and then convert it to a Big Data Batch Job using the Map Reduce framework.

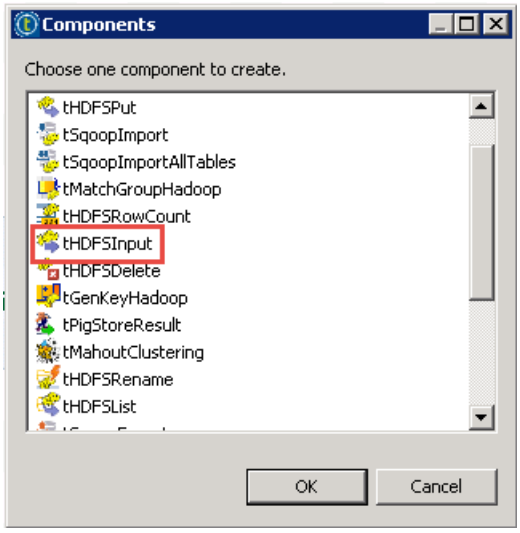
The Job will be very similar to the PigProcessing Job. At the end of this exercise, your Job will look like the following:



The first step will be to read the data using the HDFSConnection metadata.

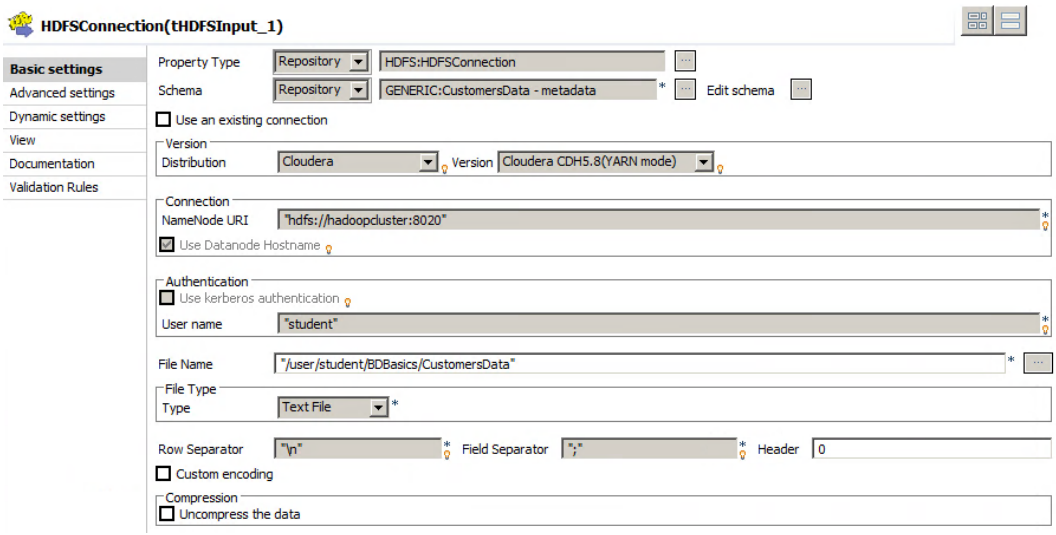
Read and filter data

1. CREATE A NEW STANDARD JOB
   Create a new **Standard Job** and name it *StandardProcessing*.
2. ADD A tHDFSInput
   Add a tHDFSInput component which uses the HDFSConnection metadata.
   a. In the Repository, under Metadata/Hadoop Cluster/TrainingCluster/HDFS, click **HDFSConnection** and drag it to the **Designer**.
   b. Select **tHDFSInput** in the **Components** list and click **OK**:

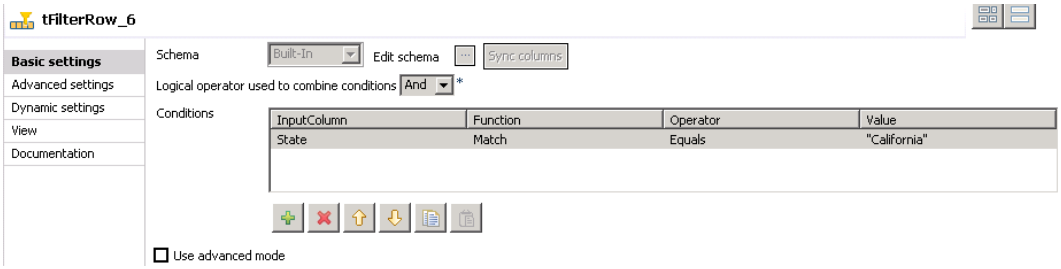


   c. Double-click the **tHDFSInput** component to open the **Component** view.
3. CONFIGURE tHDFSInput
   Configure **tHDFSInput** to read the *CustomersData* file.
   a. In the **Schema** list, select **Repository**. Then, navigate to find the **CustomersData** generic schema metadata.
   b. Next to the **File Name** box, click **(...)** and browse to find */user/student/BDBasics/CustomersData*.

Your configuration should be as follows:



4. ADD A tFilterRow COMPONENT
   Add a **tFilterRow** component and connect it with the **Main** row.
5. FILTER THE DATA
   Configure the Conditions table to filter customers living in California.
   a. In the **Component** view, click the **green plus sign** below the **Conditions** table to add a new line.
   b. In the **InputColumun** column, select **State**.
   c. In the **Function** column, select **Match**.
   d. In the **Operator** column, select **Equals**.
   e. In the **Value** colum, enter *"California"*.
   Your configuration should be as follows:

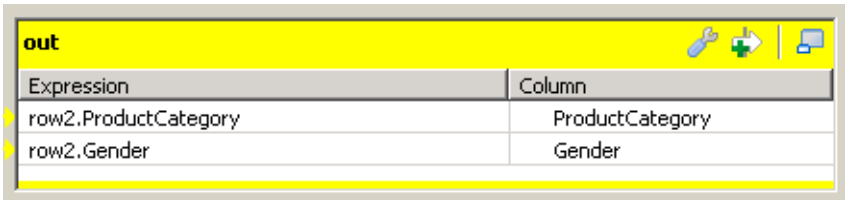


In the next section, you will map and aggregate your data.

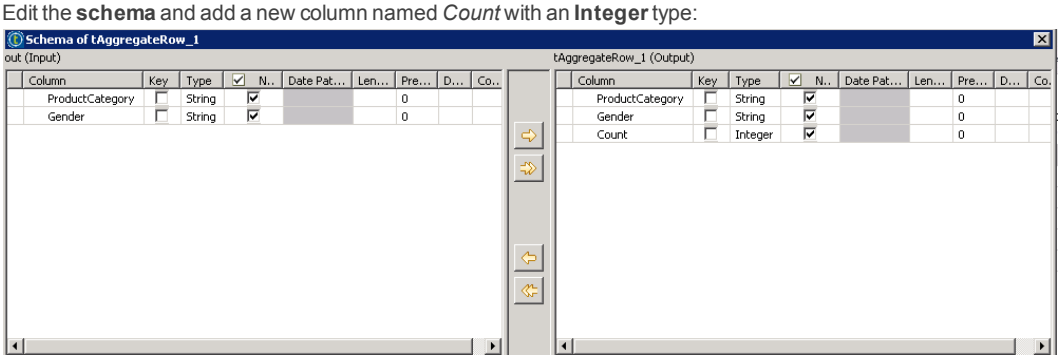
Map and Aggregate Data

1. ADD A TMAP COMPONENT
   Add a **tMap** component and connect it with the **Filter** row.
2. CREATE A MAPPING OUTPUT
   Open the mapping editor and create a new output named *out*.

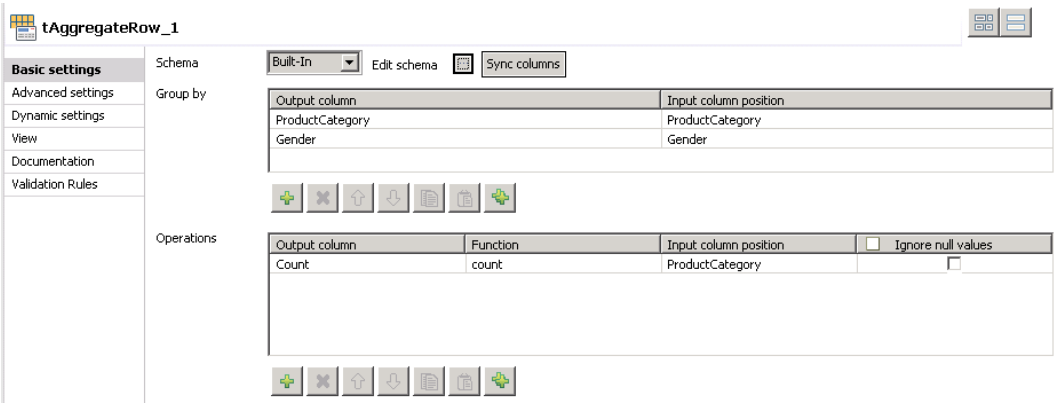
3. CONFIGURE THE MAPPING
   In the **row2** table, select **Gender** and **ProductCategory** and drag to the **out** table, as follows:



4. SAVE THE MAPPING
   Click **Ok** to save your configuration.
5. ADD A tAggregaterow COMPONENT
   Add a **tAggregateRow** component and connect it with the **out** row.
6. CONFIGURE THE AGGREGATION AS PREVIOUSLY
   Configure the **Group by** and **Operations** tables to aggregate your data by **ProductCategory** and **Gender**. The result of the aggregation is saved in a new output column named *Count*.
   a. Open the **Component** view.
   b. Edit the **schema** and add a new column named *Count* with an **Integer** type:



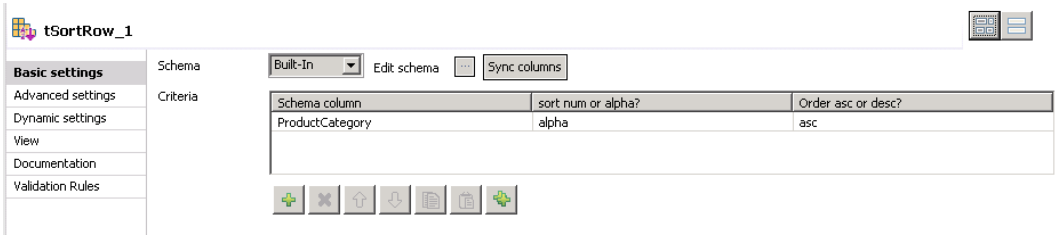
   c. Click **OK** to save the schema.
   d. Click the **green plus sign** below the **Group by** table and add the **ProductCategory** and **Gender** columns.
   e. Click the **green plus sign** below the **Operations** table.
   f. In the **Output column** list, select **Count**.
   g. In the **Function** list, select **count**.
   h. In the **Input colum position** list, select **ProductCategory**:



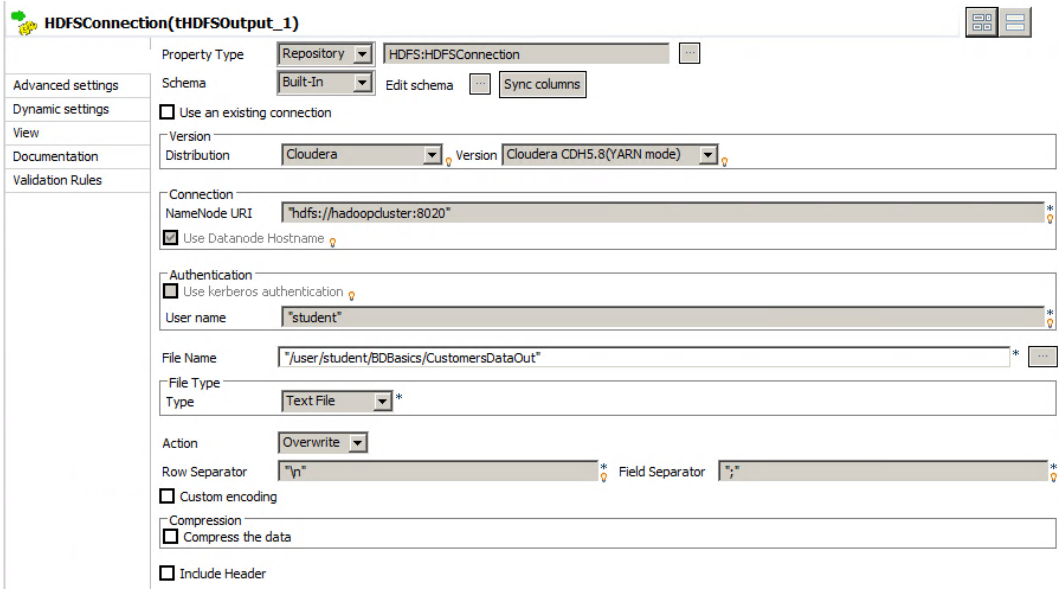
The last steps are to sort the results and then to save them on HDFS.

Sort and save Data

1. ADD A tSortRow COMPONENT
   Add a **tSortRow** component, connect it with the **Main** row, and then, open the **Component** view.
2. CONFIGURE THE SORTING
   Configure the Criteria table to sort ProductCategory in ascending order.
   a. Click the **green plus sign** below the **Criteria** table.
   b. In the **Schema** column, select **ProductCategory**.
   c. In the **sort num or alpha?** column, select **alpha**.
   d. In the **Order asc or desc** column, select **asc**:



3. ADD A tHDFSOutput
   Add a tHDFSOutput component which uses the HDFSConnection metadata.
   a. In the **Repository**, under Metadata/Hadoop Cluster/TrainingCluster/HDFS, click **HDFSConnection** and drag it to the Designer.
   b. In the **Components** list, select **tHDFSOutput**.
   c. Connect it with the **Main** row and open the **Component** view.
4. CONFIGURE THE OUTPUT FILE
   Write the data in the */user/student/BDBasics/CustomersDataOut* folder.
   a. In the **File Name** box, enter *"/user/student/BDBasics/CustomersDataOut"*.
   b. In the **Action** list, select **Overwrite**.
   Your configuration should be as follows:



Your Job is now ready to run.

Run the job and verify the results

1. RUN THE JOB
   Run your Job and check the results in the **Console**:

```
[INFO ]: bdbasics.standardprocessing_0_1.StandardProcessing - tHDFSInput_2 - Retrieving
records from the datasource.
[INFO ]: bdbasics.standardprocessing_0_1.StandardProcessing - tHDFSInput_2 - Retrieved
records count: 1000000 .
[INFO ]: bdbasics.standardprocessing_0_1.StandardProcessing - tHDFSInput_2 - Done.
[INFO ]: bdbasics.standardprocessing_0_1.StandardProcessing - tFilterRow_6 - Processed
records count:1000000. Matched records count:20401. Rejected records count:979599.
[INFO ]: bdbasics.standardprocessing_0_1.StandardProcessing - tFilterRow_6 - Done.
[INFO ]: bdbasics.standardprocessing_0_1.StandardProcessing - tMap_1 - Done.
[INFO ]: bdbasics.standardprocessing_0_1.StandardProcessing - tAggregateRow_1_AGGOUT -
Done.
[INFO ]: bdbasics.standardprocessing_0_1.StandardProcessing - tSortRow_1_SortOut - Start
to work.
[INFO ]: bdbasics.standardprocessing_0_1.StandardProcessing - tAggregateRow_1_AGGIN -
Start to work.
[INFO ]: bdbasics.standardprocessing_0_1.StandardProcessing - tAggregateRow_1_AGGIN -
Retrieving the aggregation results.
[INFO ]: bdbasics.standardprocessing_0_1.StandardProcessing - tAggregateRow_1_AGGIN -
Done.
[INFO ]: bdbasics.standardprocessing_0_1.StandardProcessing - tSortRow_1_SortOut - Done.
[INFO ]: bdbasics.standardprocessing_0_1.StandardProcessing - tHDFSOutput_2 - Start to
work.
[INFO ]: org.apache.hadoop.conf.Configuration.deprecation - fs.default.name is deprecated.
Instead, use fs.defaultFS
[INFO ]: bdbasics.standardprocessing_0_1.StandardProcessing - tSortRow_1_SortIn - Start to
work.
[INFO ]: bdbasics.standardprocessing_0_1.StandardProcessing - tSortRow_1_SortIn - Done.
[INFO ]: bdbasics.standardprocessing_0_1.StandardProcessing - tHDFSOutput_2 - Written
records count: 14 .
[INFO ]: bdbasics.standardprocessing_0_1.StandardProcessing - tHDFSOutput_2 - Done.
[statistics] disconnected
[INFO ]: bdbasics.standardprocessing_0_1.StandardProcessing - TalendJob:
'StandardProcessing' - Done.
Job StandardProcessing ended at 16:06 01/07/2015. [exit code=0]
```

   Your Job should execute successfully.

2. CONNECT TO Hue AND VERIFY THE RESULTS
   In **Hue**, using the **File Browser**, navigate to /user/student/BDBasics and open **CustomersDataOut**:


```
clothing;M;1464
clothing;F;1410
electronics;F;1416
electronics;M;1486
games;M;1494
games;F;1508
handbags;F;1452
handbags;M;1396
movies;F;1475
movies;M;1400
shoes;M;1460
shoes;F;1475
tools;F;1460
tools;M;1505
```

   The results should be the same as in the previous lab, **Processing Data with Pig**.

You will now convert this Standard Job into a Big Data Batch Job, which will use the Map Reduce framework.

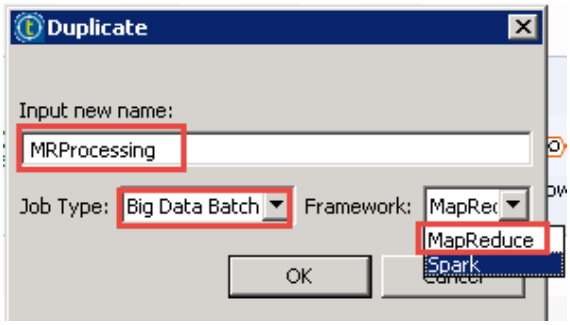
Convert to a Map Reduce batch Job

Using the Studio, you can convert a standard Job to a Big Data Batch Job, and choose between a Map Reduce or Spark framework.

In this lesson, you will focus on Map Reduce Jobs.

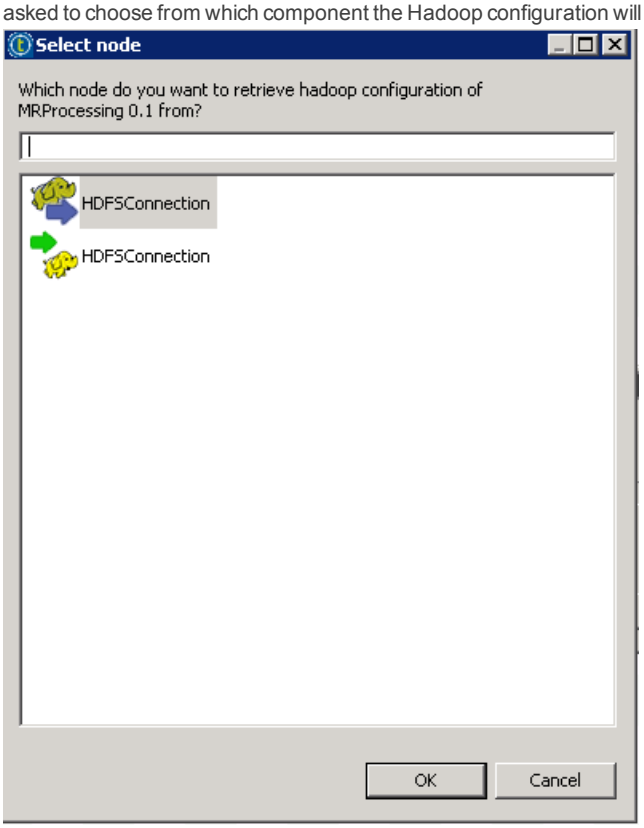
Instead of converting your current Job, you will duplicate it as a Map Reduce based Batch Job.

1. DUPLICATE YOUR JOB
   Duplicate the **StandardProcessing** Job and name the duplicate *MRProcessing*.
   MRProcessing is a Big Data batch Job using the MapReduce framework.
   a. In the **Repository**, right-click the **StandardProcessing** Job, and then, click **Duplicate**.
   b. In the **Input new name** box, enter *MRProcessing*.
   c. In the **Job Type** list, select **Big Data Batch**.
   d. In the **Framework** list, select **MapReduce**:

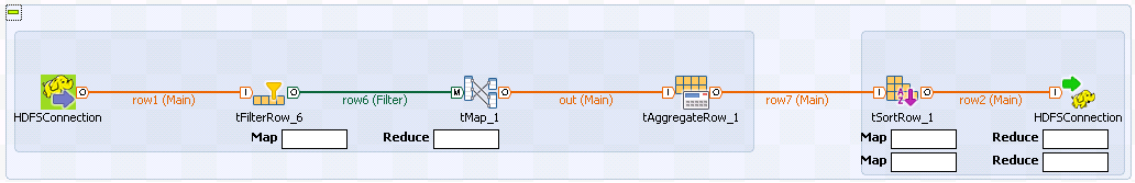


   e. Click **OK**.
2. CONFIGURE THE CONNECTION TO THE HADOOP CLUSTER
   The Hadoop cluster configuration is set at the Job level in Big Data Batch Jobs. So, before duplicating your Job, you will be

asked to choose from which component the Hadoop configuration will be retrieved:



3. SELECT A COMPONENT
   Select the **HDFSConnection** component with the blue arrow and click **OK**.
4. OPEN THE JOB
   In the **Repository**, under **Big Data Batch Jobs**, you will find your **MRProcessing Job**. If you haven't done so already, double-click to open it:



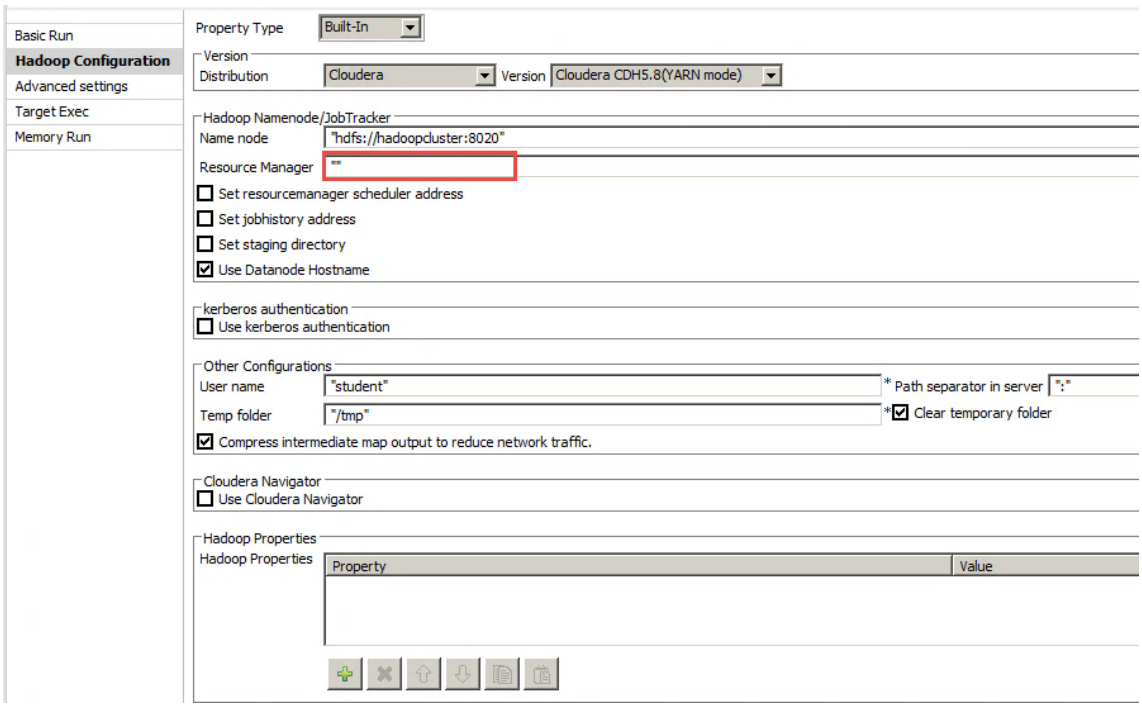
5. CONFIGURE THE OUTPUT FOLDER
   Configure **tHDFSOutput** to write the results in the */user/student/BDBasics/CustomersDataOut_MR* folder.
   a. Double-click the **tHDFSOutput** component to open the **Component** view.
   b. In the **Folder** box, enter
   "/user/student/BDBasics/CustomersDataOut_MR".

Your standard Job is now converted to a Map Reduce Job.

Run the Job and verify the results

Before running your Job, you will check that the Hadoop cluster configuration is correct.

1. OPEN THE RUN VIEW
   Open the **Run** view of the **MRProcessing Job**.
2. VERIFY THE HADOOP CLUSTER CONNECTION INFORMATION
   Click the **Hadoop Configuration** tab:

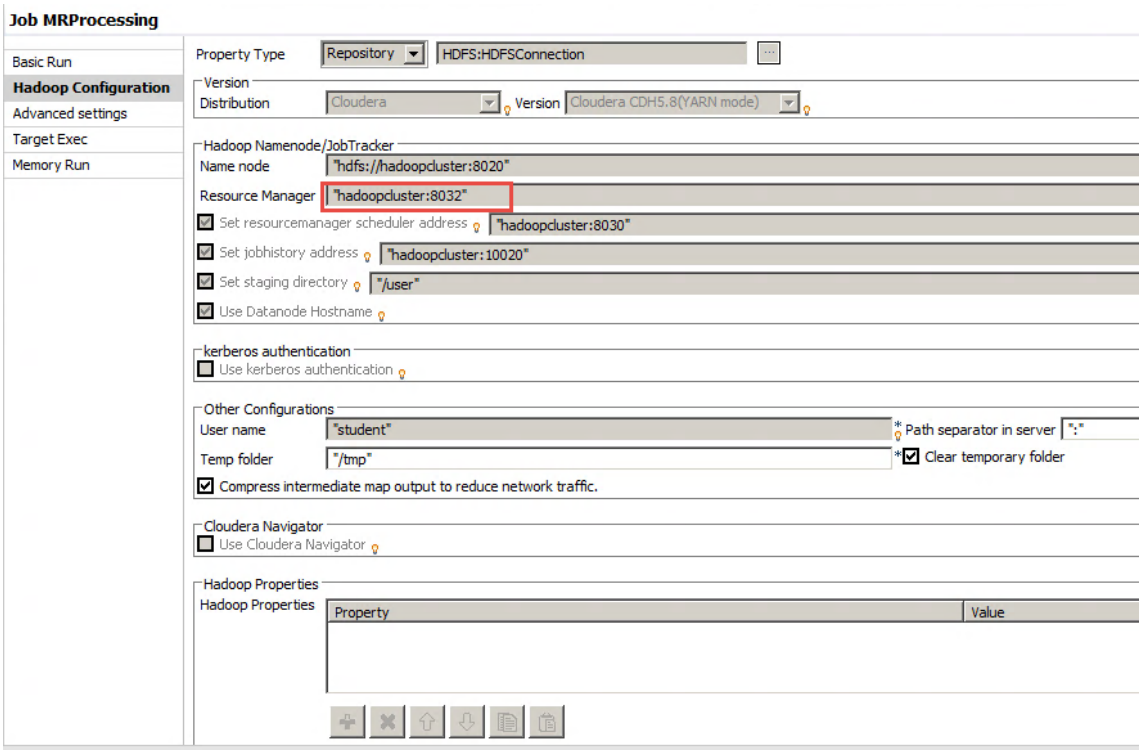


As the configuration has been retrieved from a **tHDFSInput** component, some configurations are missing, because they are not necessary to read/write from HDFS.

The Resource Manager address is necessary to run a Map Reduce Job, and is currently missing. If you run the Job, it will fail with the following error message:

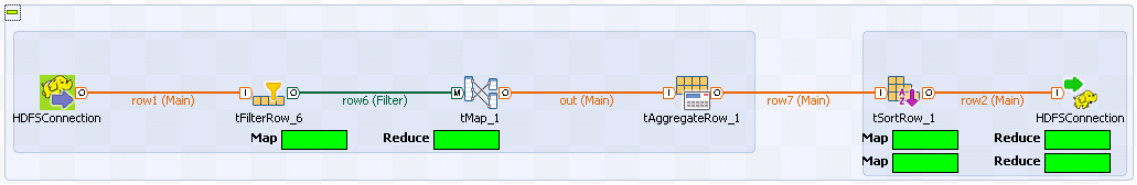
```
java.io.IOException: Cannot initialize Cluster. Please check your configuration for
mapreduce.framework.name and the correspond server addresses.
    at org.apache.hadoop.mapreduce.Cluster.initialize(Cluster.java:120)
    at org.apache.hadoop.mapreduce.Cluster.<init>(Cluster.java:82)
    at org.apache.hadoop.mapreduce.Cluster.<init>(Cluster.java:75)
    at org.apache.hadoop.mapred.JobClient.init(JobClient.java:472)
    at org.apache.hadoop.mapred.JobClient.<init>(JobClient.java:450)
    at
bdbasics.mrprocessing_0_1.MRProcessing.setDefaultMapReduceConfig(MRProcessing.java:5163)
    at bdbasics.mrprocessing_0_1.MRProcessing.initMapReduceJob(MRProcessing.java:5139)
    at bdbasics.mrprocessing_0_1.MRProcessing.run(MRProcessing.java:5071)
    at org.apache.hadoop.util.ToolRunner.run(ToolRunner.java:70)
    at bdbasics.mrprocessing_0_1.MRProcessing.runJobInTOS(MRProcessing.java:5044)
    at bdbasics.mrprocessing_0_1.MRProcessing.main(MRProcessing.java:5029)
[INFO ]: org.apache.hadoop.mapreduce.Cluster - Failed to use
org.apache.hadoop.mapred.YarnClientProtocolProvider due to error: Does not contain a
valid host:port authority:  (configuration property 'yarn.resourcemanager.address')
Job MRProcessing ended at 17:21 01/07/2015. [exit code=1]
```

3. CONFIGURE THE RESOURCE MANAGER ADDRESS
   To fix this, you can either use your **TrainingCluster metadata**, or set the **Resource Manager** to *"hadoopcluster:8032"*:



4. RUN YOUR JOB
   Go back to the **Basic Run** tab and **run** your Job.
5. OBSERVE THE DESIGNER
   You can check that your Job is running in the **Designer** view. As the different Map Reduce Jobs execute, you will see the **progress bars** changing.

   First they are empty, then, when a Map or Reduce task starts to execute, the progress bar becomes red. When a task successfully complete, the progress bar becomes green:



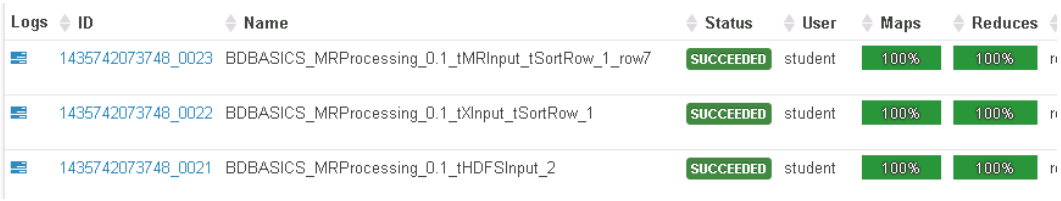
6. OBSERVE THE CONSOLE
   You can also follow the execution in the **Console**:

```
deprecated. Instead, use mapreduce.client.output.filter
Running job: job_1435742073748_0021
 map 0% reduce 0%
 map 50% reduce 0%
 map 100% reduce 0%
 map 100% reduce 100%
Job complete: job_1435742073748_0021
Counters: 49
    File System Counters
        FILE: Number of bytes read=200
        FILE: Number of bytes written=380209
        FILE: Number of read operations=0
        FILE: Number of large read operations=0
        FILE: Number of write operations=0
        HDFS: Number of bytes read=61827598
        HDFS: Number of bytes written=441
        HDFS: Number of read operations=9
        HDFS: Number of large read operations=0
        HDFS: Number of write operations=2
    Job Counters
        Launched map tasks=2
        Launched reduce tasks=1
        Data-local map tasks=2
        Total time spent by all maps in occupied slots (ms)=16661
        Total time spent by all reduces in occupied slots (ms)=3780
        Total time spent by all map tasks (ms)=16661
        Total time spent by all reduce tasks (ms)=3780
        Total vcore-seconds taken by all map tasks=16661
        Total vcore-seconds taken by all reduce tasks=3780
        Total megabyte-seconds taken by all map tasks=17060864
        Total megabyte-seconds taken by all reduce tasks=3870720
```

You should see three reports, one for each Map Reduce Job launched by the Studio.
The execution of Map Reduce tasks is given with a percentage.

7. CONNECT TO Hue
   Use the **Job Browser** and the **File Browser** to check the execution and the results.

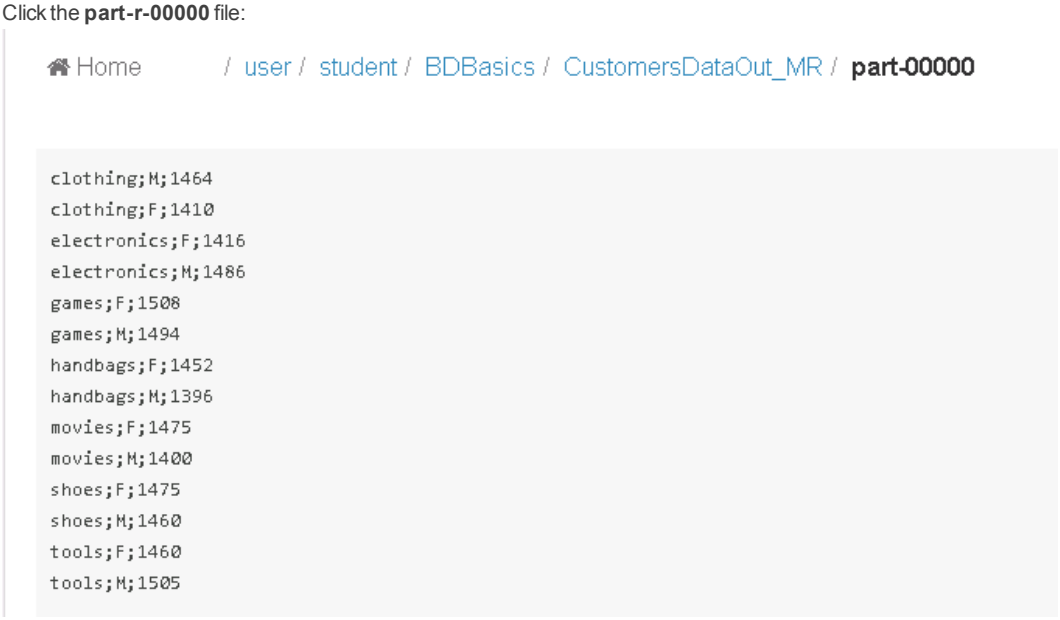
   a. From the Console, you can get your Job ID and find it in the **Hue Job Browser**:



Logs	ID	Name	Status	User	Maps	Reduces
	1435742073748_0023	BDBASICS_MRProcessing_0.1_tMRInput_tSortRow_1_row7	SUCCEEDED	student	100%	100%
	1435742073748_0022	BDBASICS_MRProcessing_0.1_tXInput_tSortRow_1	SUCCEEDED	student	100%	100%
	1435742073748_0021	BDBASICS_MRProcessing_0.1_tHDFSInput_2	SUCCEEDED	student	100%	100%

   b. Using the **File Browser** in **Hue**, check the results saved in the **/user/student/BDBasics/CustomersDataOut_MR folder**.

c. Click the **part-r-00000** file:



You should have the same results as in previous labs.

You have now covered the different ways to process tables and data stored on HDFS.

Next step

You have almost finished this section. Time for a quick review.

Review

Recap

In this chapter, you covered how to process tables and data stored on HDFS.

The first part of the chapter was dedicated to Hive Tables. You built a Job to extract data of interest in your Hive table with a tHiveInput component. You processed the data, saved the result in HDFS with a tHDFSOutput component, and you transferred the result to a Hive table using tHiveCreateTable and tHiveRow.

Next, you used the Profiling view of the Studio to perform different levels of analyses. You started at the connection level, moved to the Tables and column level, and ended with a custom analysis on the ProductCategory column. Each request was run as a Map reduce Job in your cluster.

The second part of the chapter was dedicated to data processing with Pig and Big Data Batch Jobs.

You created a Job using Pig components to process your data. Each Pig request was executed as a Map Reduce Job in your cluster.

Next, you created a standard Job to process your data and reproduced the results obtained with Pig. Then, you duplicated the standard Job and created a Big Data Batch Job using the Map Reduce Framework.

Further reading

If you want to discover more about data profiling, the Talend Data Quality trainings will help you. If you are interested in discovering more about Big Data Batch Jobs, the Talend Big Data Advanced training will give you an overview of real life use cases using Big Data Batch Jobs.

Intentionally blank