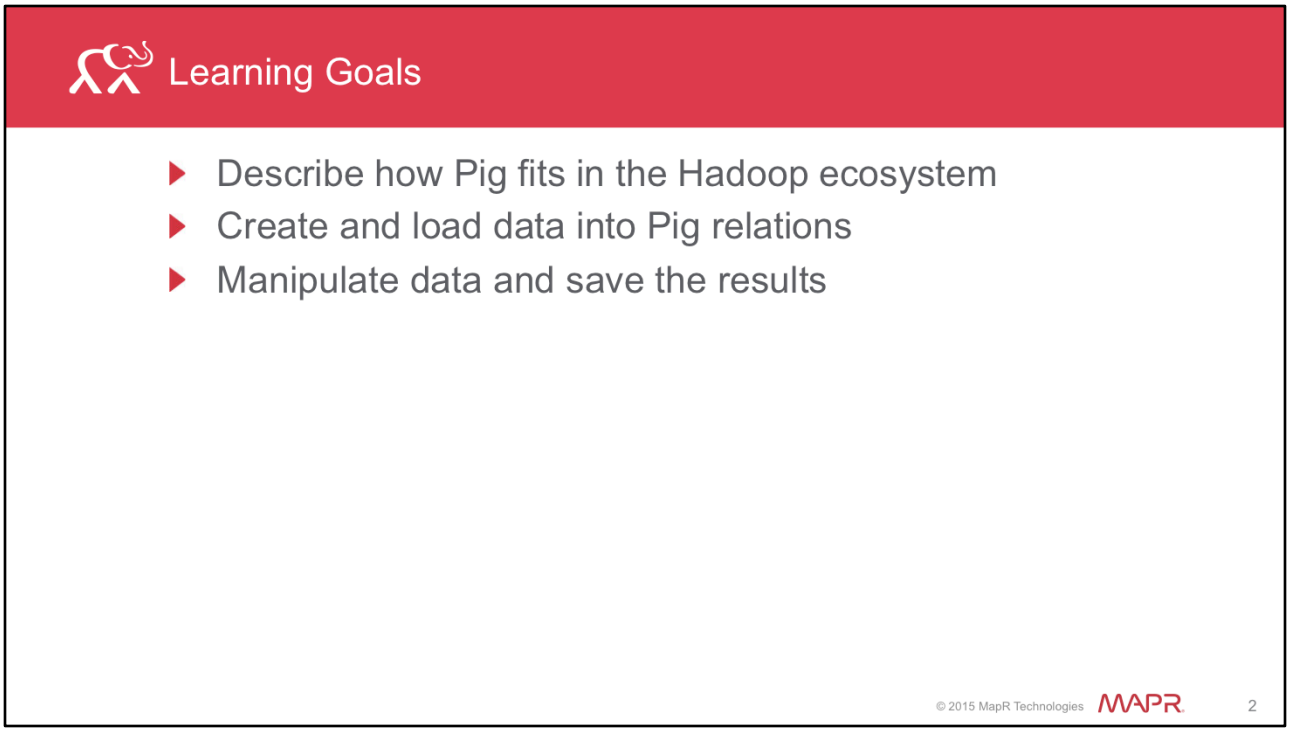
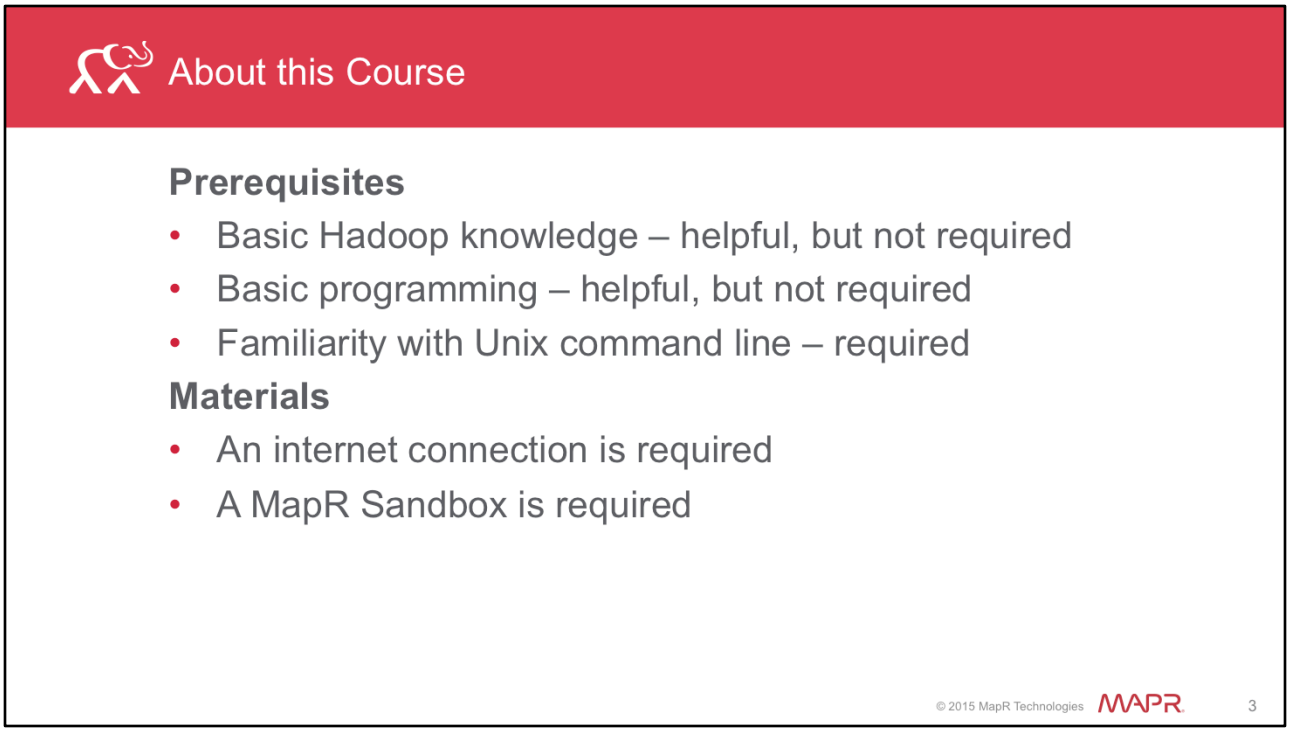
Welcome to MapR Academy. This is DA 450 – Apache Pig Essentials. This course is targeted towards data analysts, data architects, and Apache Hadoop developers. This course is designed to introduce you to Apache Hive and the Hive Query Language.

Use the navigation in the top right hand corner to move through the lesson. You can also jump to specific topics by selecting the table of contents on the left side of the screen.



When you have completed this course, you will be able to:
Describe how Pig fits in the Hadoop ecosystem and the data pipeline
Create and load data into Pig relations
Manipulate data, and save the results



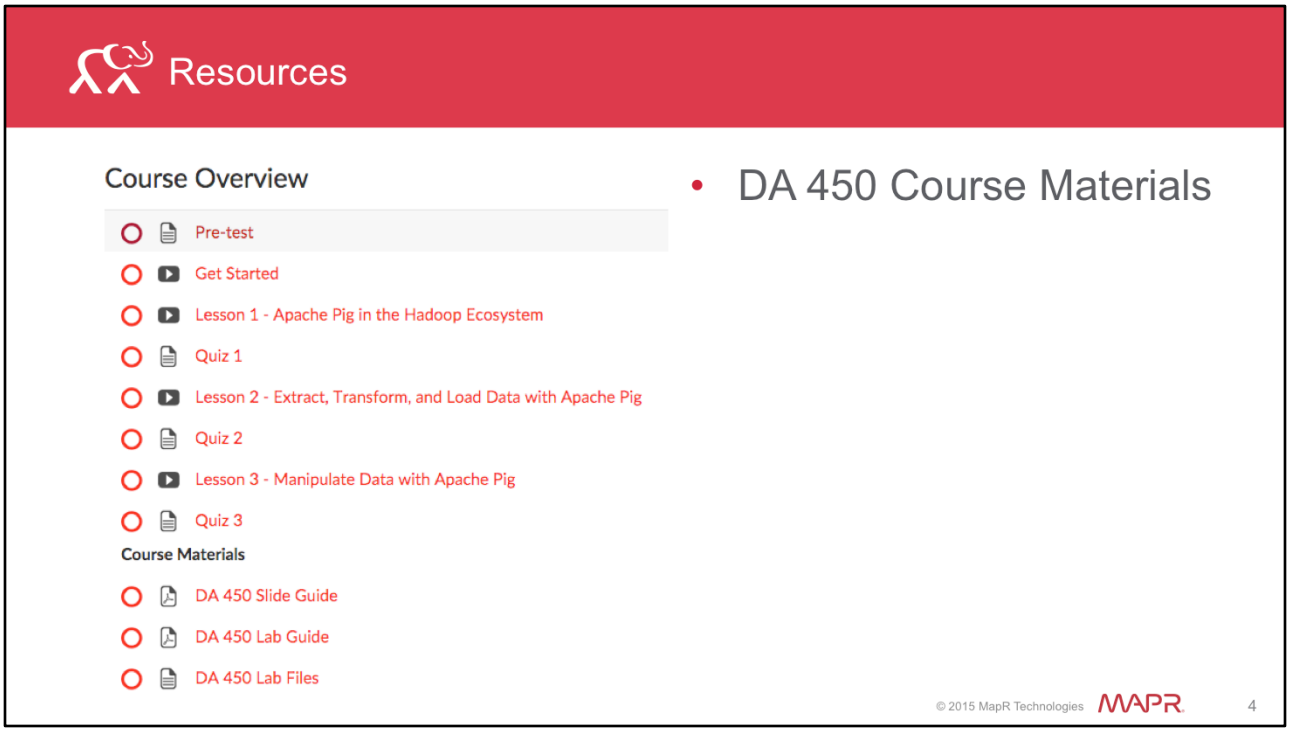
Basic Hadoop knowledge is helpful, but not required. It is helpful to have taken HDE 100 – Hadoop Essentials prior to this course.

Familiarity with SQL, Python, or other programming languages is also helpful, but not required

This course also assumes you are comfortable working in a command line interface, such as a Unix shell.

An internet connection is required to view all the slides, listen to the lectures, complete the quizzes, download the lab guide and the MapR Sandbox.

A virtual machine running the MapR Sandbox is required to complete the labs. Instructions for downloading and installing

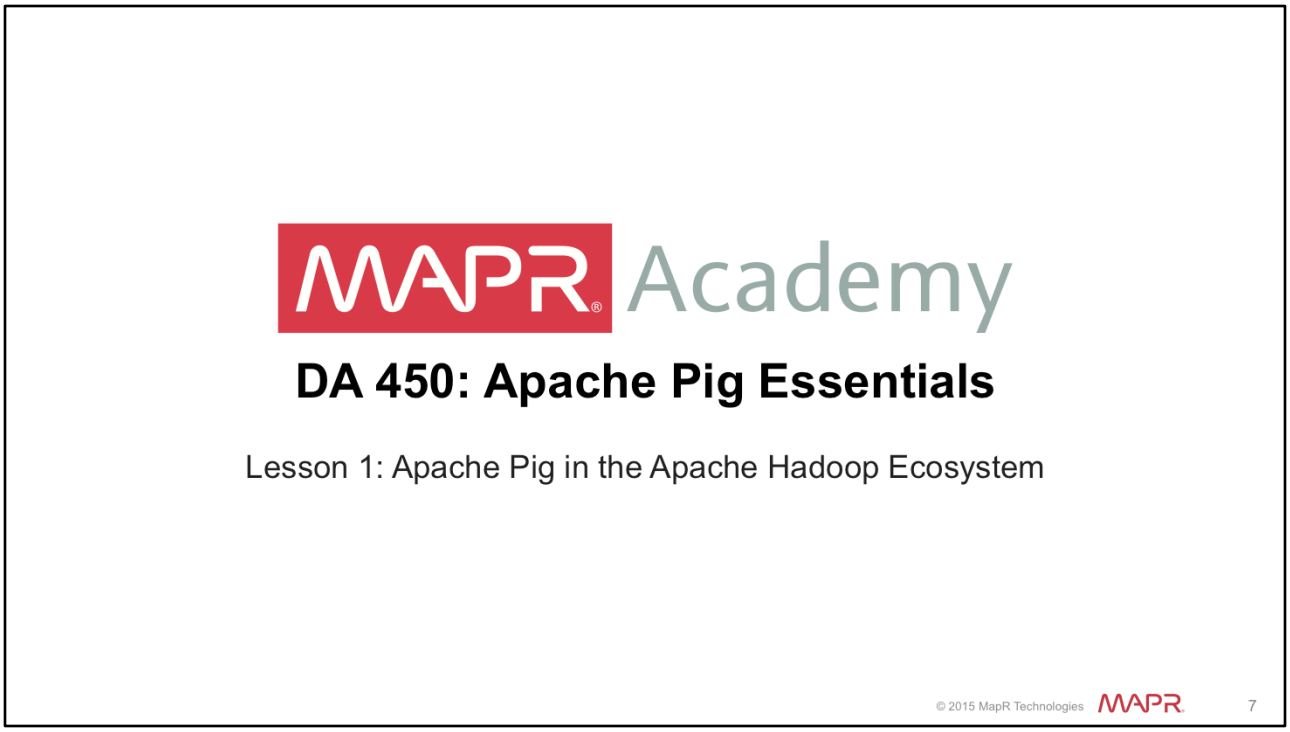


You can perform the exercises demonstrated in this course on the MapR sandbox. Refer to the course materials file – DA 450 LAB GUIDE.pdf – for a list of resources and links that you need for the course. This document is available as an attachment to course in the learning management system as shown here.

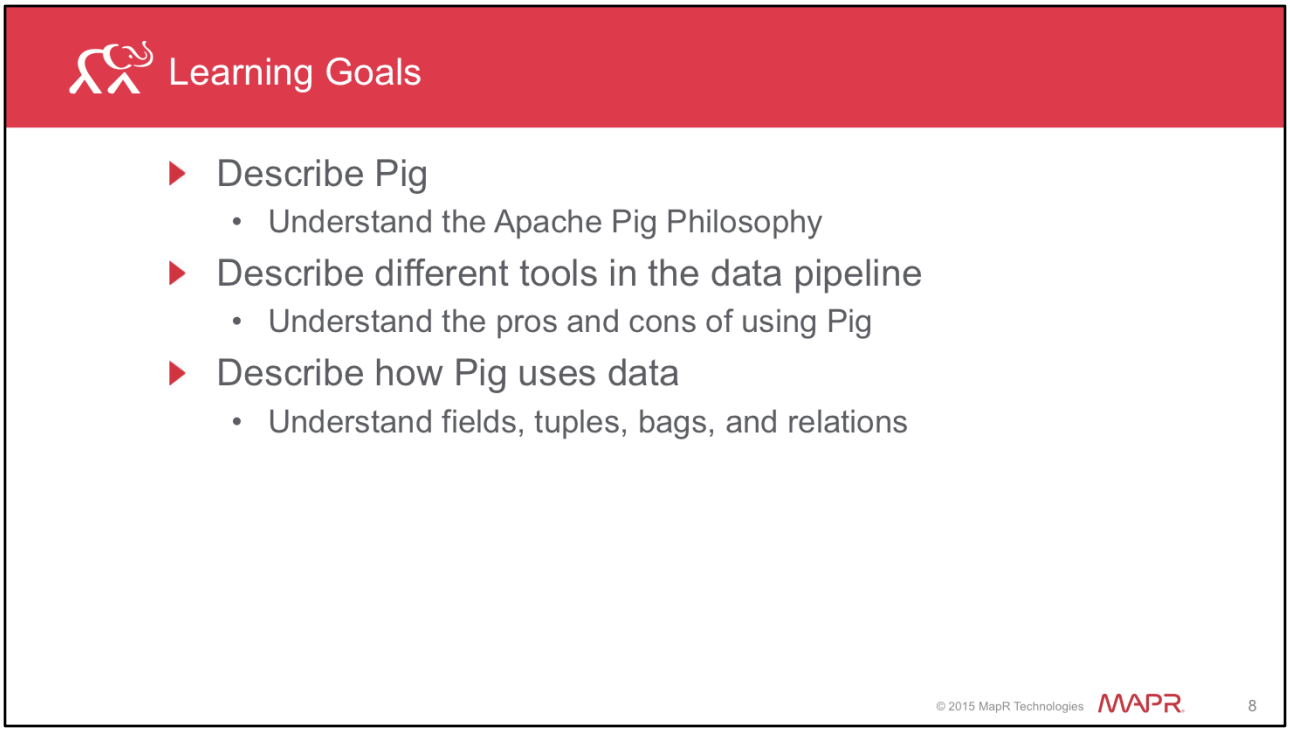
Download and install a virtual machine running the MapR Sandbox before continuing.

Refer to the Get Started section of the Lab Guide to complete this portion of the Lab. In this Lab, you will connect to your Sandbox, AWS cluster, or GCP cluster through a terminal, then use SCP to copy your training files from your machine to your virtual machine. When you are finished, you are ready for Lesson 1.

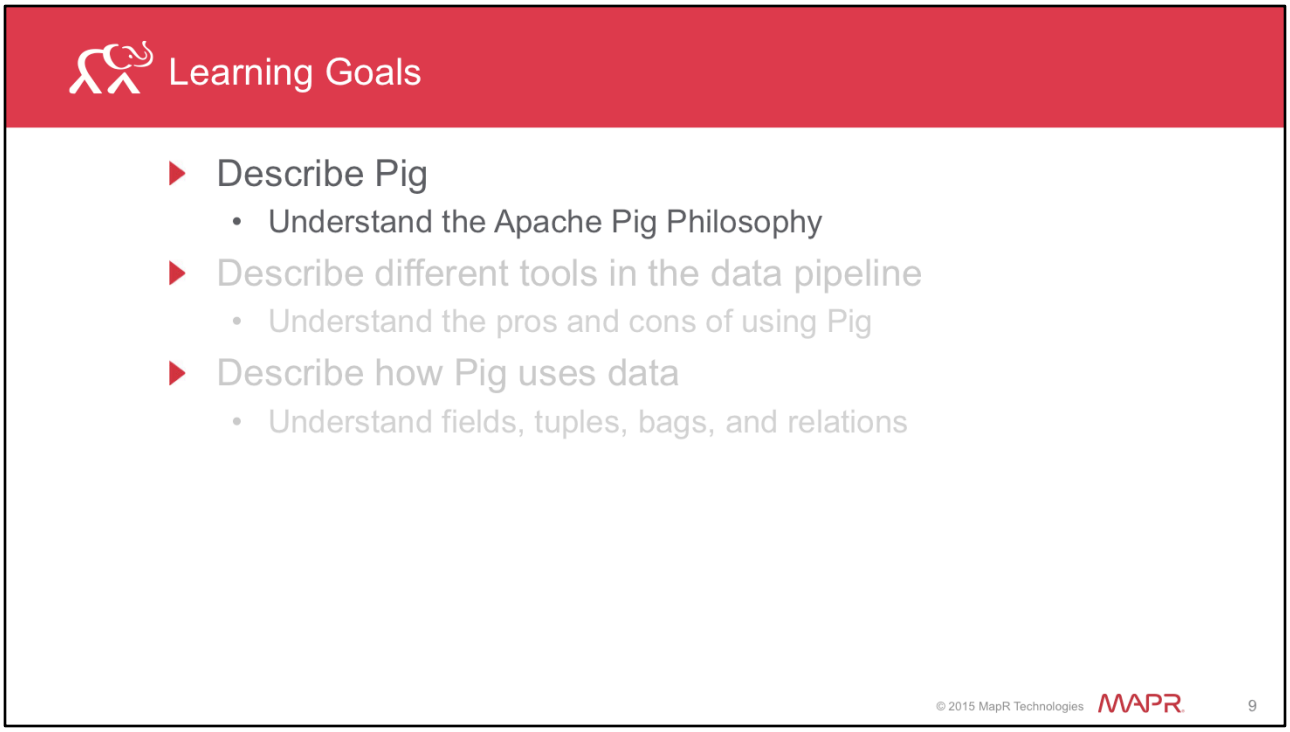
In the next lesson we will take a look at Pig and how it fits in the Hadoop ecosystem.



Welcome to DA 450: Lesson 1 – Apache Pig in the Apache Hadoop Ecosystem. In this lesson, we will learn what Pig is and how it fits into the Hadoop ecosystem.



In this lesson we will learn about Pig, what its use cases are, how it fits in the Hadoop ecosystem, and how it parses data into fields, tuples, bags, and relations.



Let's start with a brief overview of what Pig is, who uses it, and what the Apache Pig Philosophy is.

Too much data?

- You:
  - work in a large climate research center
  - work with data scientists who want to query data with Hive
  - have accumulated petabytes of weather measurements, stored as semi-structured tables
- Your data has moved to HDFS
- How will you handle this?

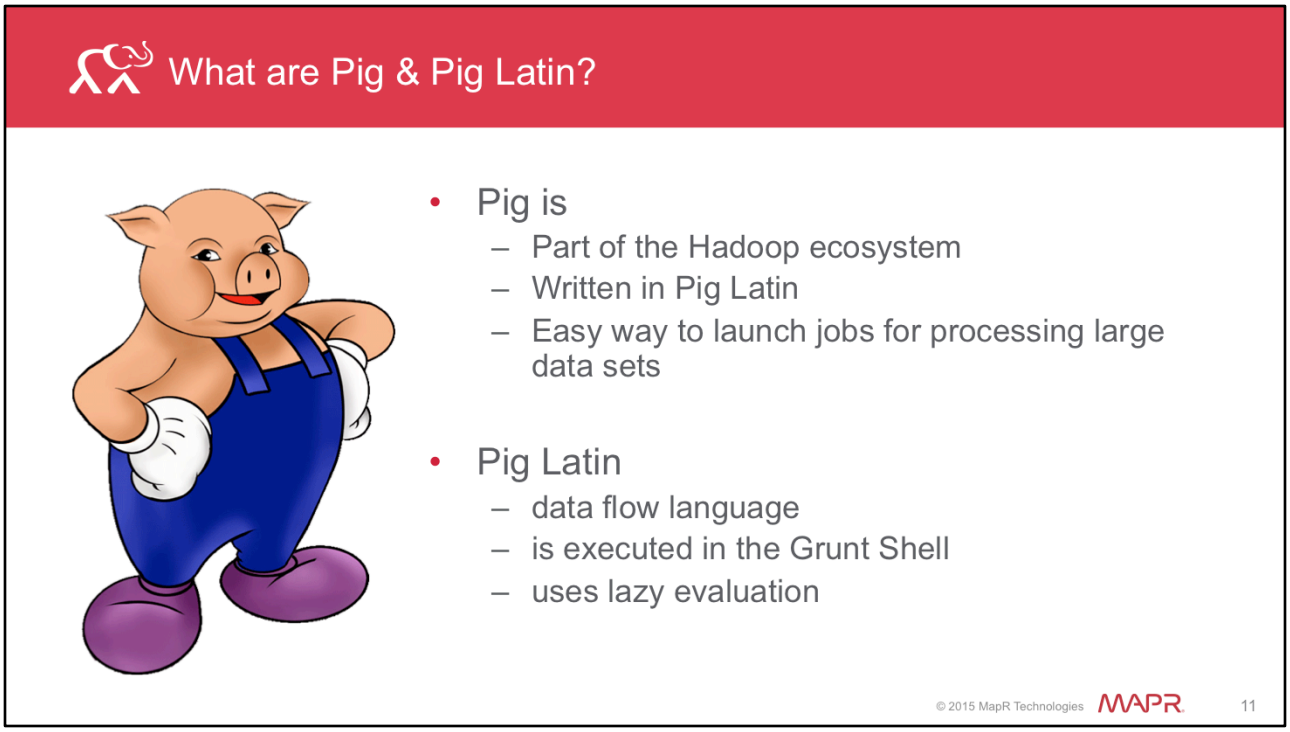
We are working at an international Climate Research Institute, and you are a developer who works closely with data scientists . This research center manages thousands of weather stations all over the globe. Each weather station provides a number of data points such as temperature and wind speed. This data is sent to a central server in batches every month.

This data is semi-structured and is not ideal for our research purposes. Each row contains data points which summarize an entire year's of data from a single weather station on a single metric, such as the temperature or the wind speed.

As a developer, there are several manipulations you would like preformed on this semi-structured data before the data scientists can query it.

However, because you have accumulated several petabytes of data, traditional tools such as SQL or Python are not ideal for this task. Moreover, The Climate Research Institute has decided to migrate all of its data to a Hadoop Distributed File System. The HDFS can store, access, and backup petabytes of data easily.

What are your options as a data scientist or developer who needs to manipulate this massive amount of data? One option is Apache Pig.



What is Pig?

Apache Pig is part of the Hadoop ecosystem, and is often used in conjunction with other tools, such as Apache Hive.

Pig was initially developed at Yahoo as a way to create and execute MapReduce jobs on very large datasets, without the need to write low-level Java or MapReduce code. In addition to MapReduce, Pig is now also used to launch Spark jobs.

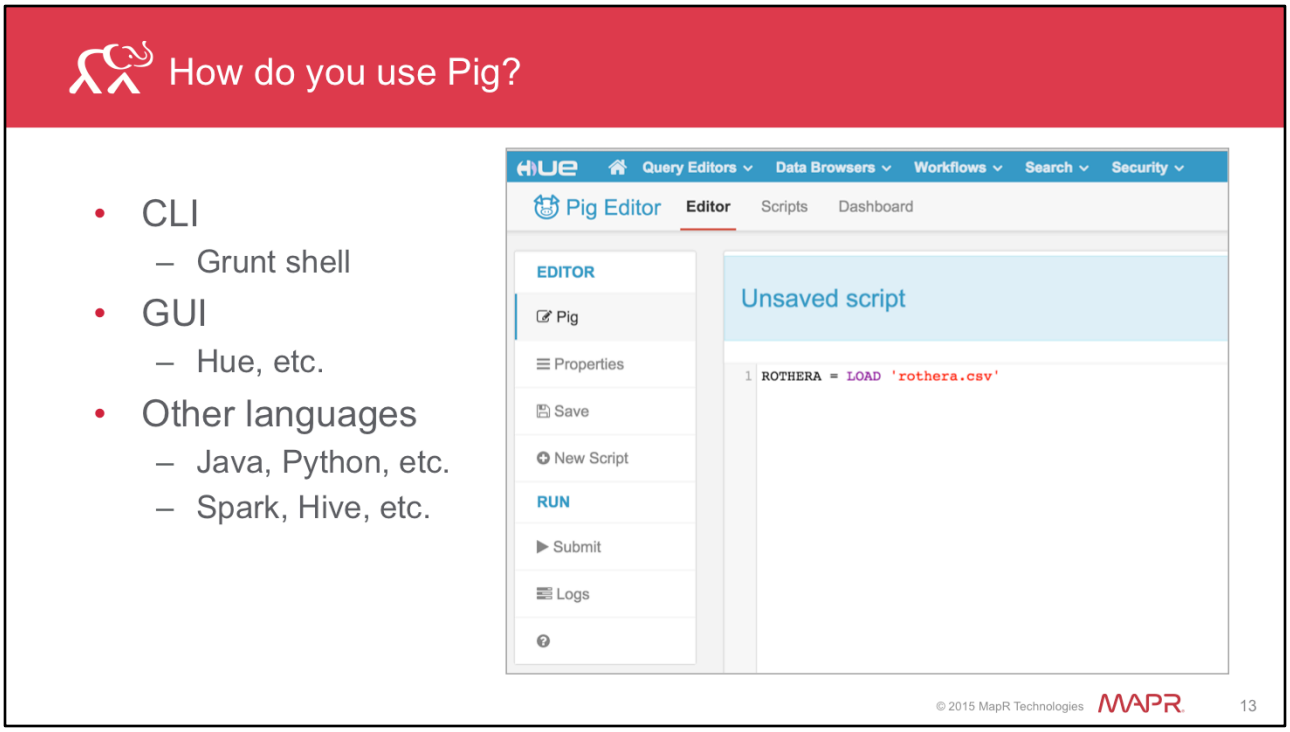
The language of Pig is Pig Latin. Pig Latin is a data flow language, which is optimized for ingesting, processing, and normalizing data. Pig Latin is a higher level language, similar to SQL or Python which makes it easier for data analysts to write MapReduce or Spark jobs in an HDFS.

Who uses Pig?

- Data scientists & data architects
  - Need to process large batches of data
  - Initial step in extract, transform, load (ETL)
- Hadoop developers & administrators
  - Work with other Hadoop ecosystem apps
  - Install, update, & configure Pig

The most common users of Pig are data scientists and data architects, although many others may use Pig as well.

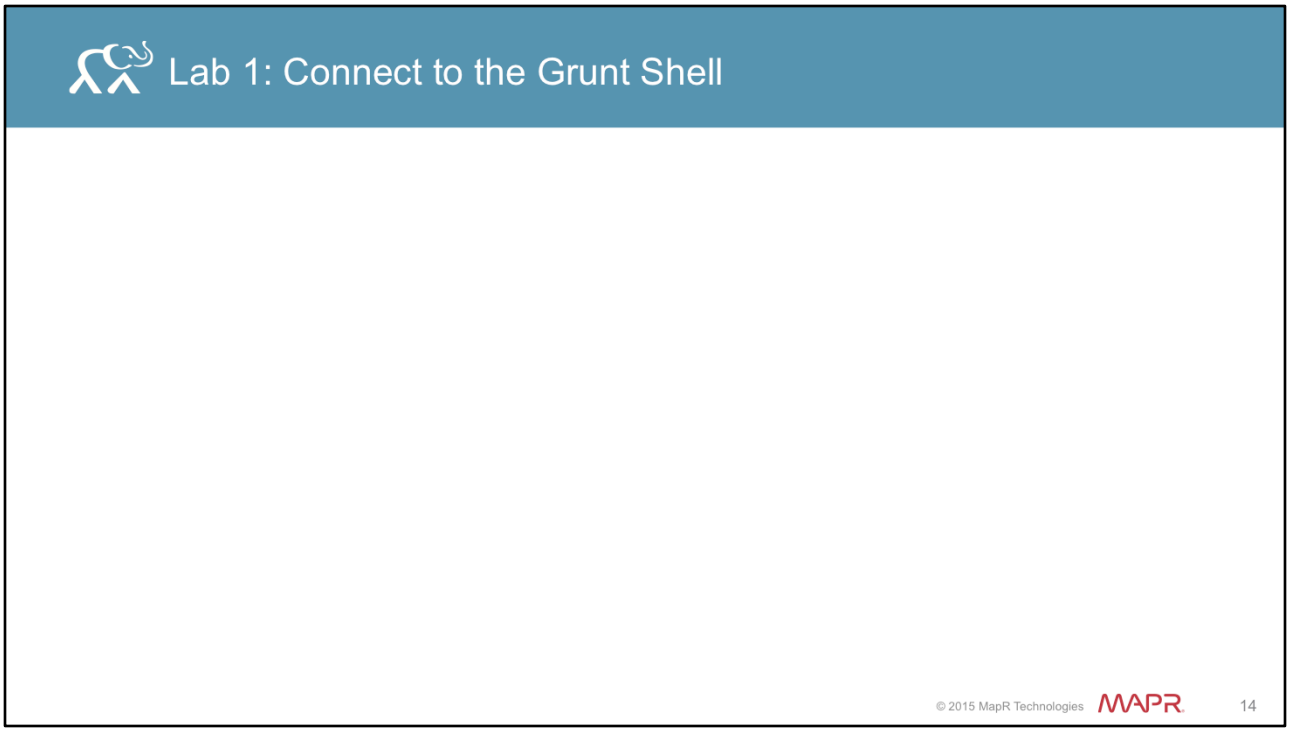
Many Hadoop developers may need to be familiar with Pig in order to work with other applications in the Hadoop ecosystem that also use Pig. Hadoop administrators may need to be familiar with Pig in order to install, update, and configure Pig and related ecosystem tools.



One way to use Pig and Pig Latin is through the grunt shell.

For users not comfortable with the command line, there are graphical user interface options like Hue. We will be using the Grunt Shell in the labs for this course.

Pig can also be used in conjunction with other programming languages. For example, user-defined functions for Pig can be written in Java, Python, Ruby, or JavaScript. Pig can also be used in conjunction with other tools within the Hadoop ecosystem, such as Spark or Hive.



You should have already installed and connected to the MapR Sandbox during Lesson 0. Now connect to the Grunt Shell, which is how we will use Pig in this course.

Once you have connected to the Grunt shell, you can continue with lesson 1.

Knowledge Check

Who should be familiar with Pig? Check all that apply.

a) Data scientists working with data on an HDFS
b) Business analysts working with data on an RDBMS
c) Data analysts who want interactive, real time queries
d) Hadoop developers who work with data scientists

What is the Pig Philosophy?

- Pigs eat anything
- Pigs live anywhere
- Pigs are domestic
- Pigs can fly
- https://pig.apache.org/philosophy.html

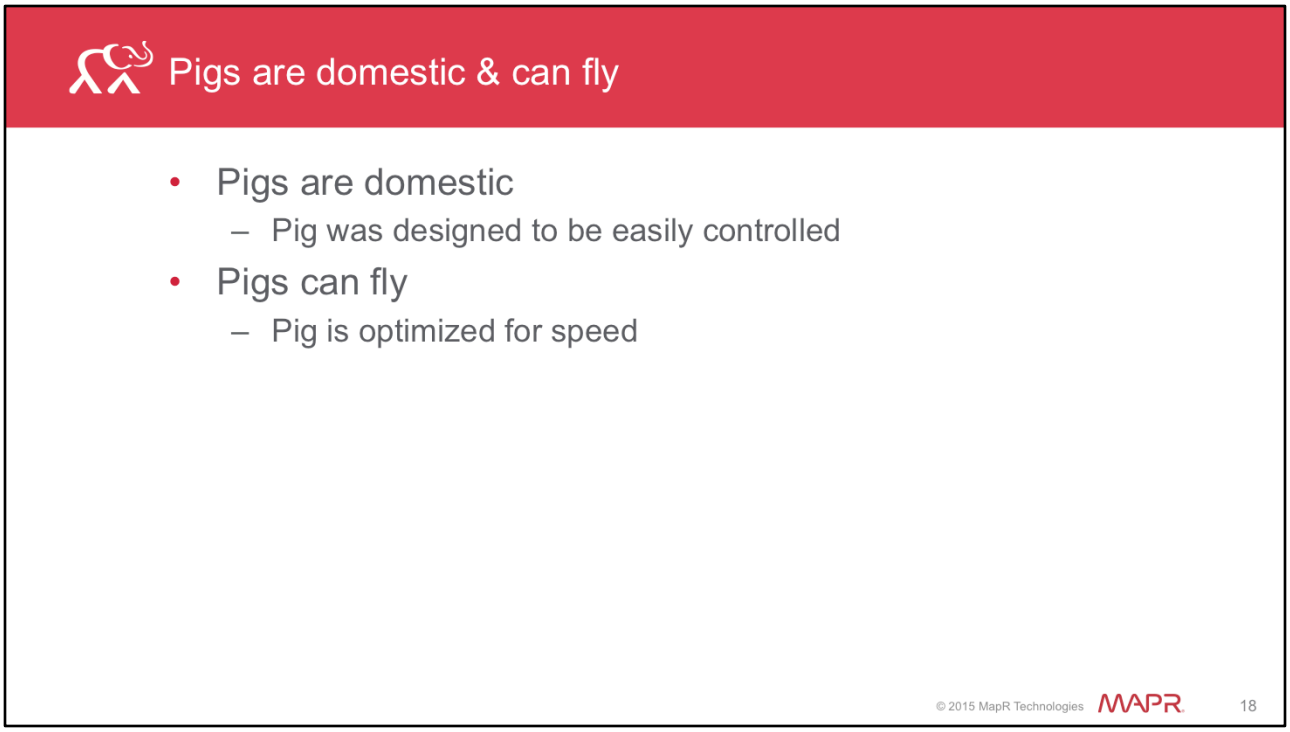
 MAPR 16

One of the best ways to understand what Pig can and cannot do is to understand the Pig Philosophy, which was written by the original developers of Pig.

We'll go over each tenet of the Pig philosophy in a moment. The basic idea, however, is that Apache Pig shares many traits with real pigs. Pigs eat anything, pigs live anywhere, pigs are domestic, and pigs fly!

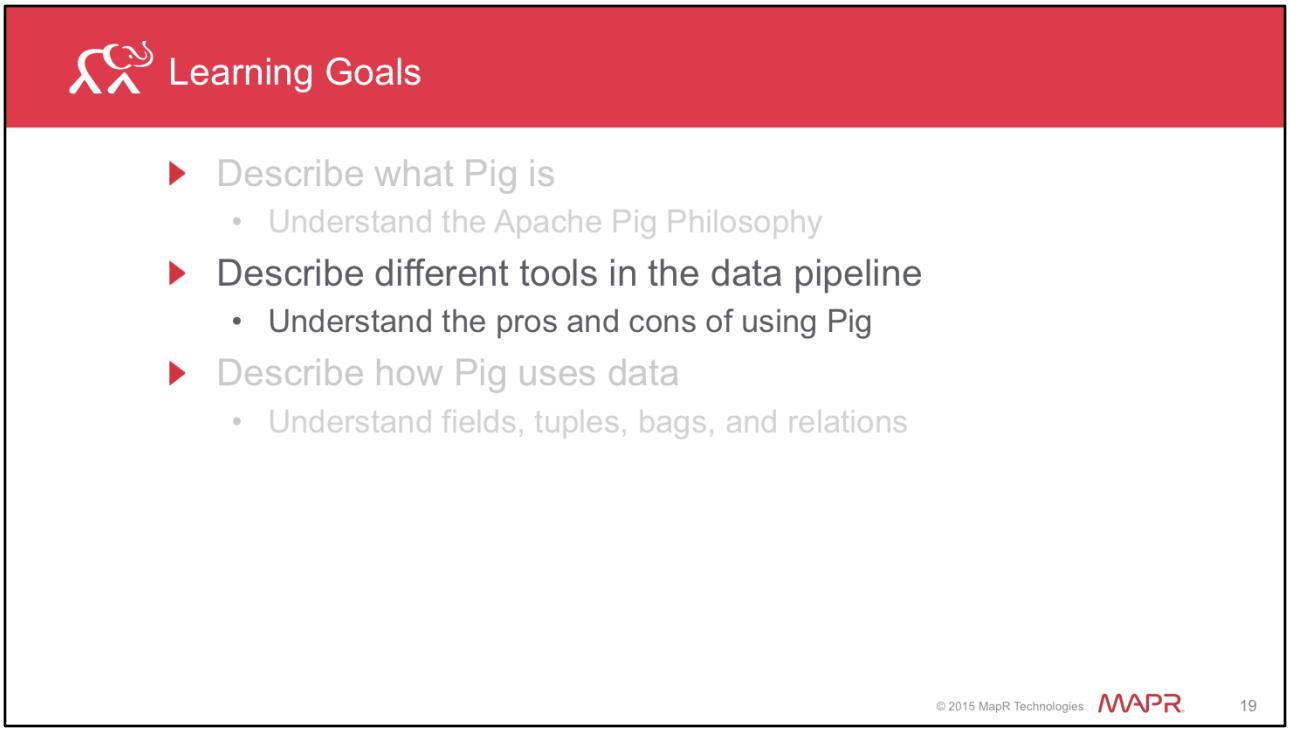
The first tenet of the Pig Philosophy is that Pigs eat anything. What this means is that anything you give Pig, it can handle. Pig can operate on nearly any data type, with or without metadata. It can be structured or unstructured or even nested. Pig can also be extended to operate on data beyond files, including key-value stores or entire databases.

The second tenet of the Pig Philosophy is that Pigs live anywhere. This means that even though it was designed to run on Hadoop, it does not have to work on Hadoop. Pig Latin was designed to work on any parallel data processing system.

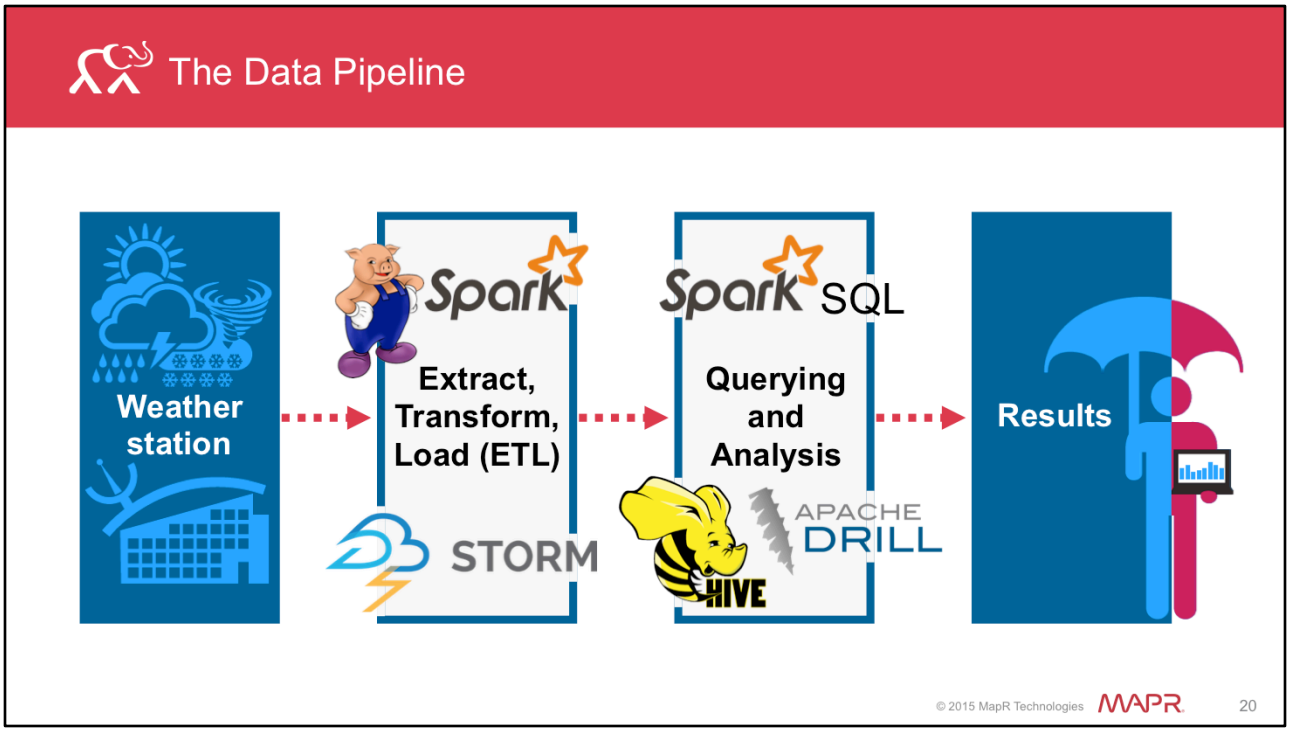


The third tenet of the Pig philosophy is that Pigs are domestic animals. This means that Pig is meant to be easily controlled by its users. Pig is very adaptable and supports a variety of user-defined functions, as well as integration with many other programming languages such as Java and Python.

The last tenet of the Pig philosophy is that Pigs fly. Pig processes data quickly and newer versions are constantly being optimized for performance speed.



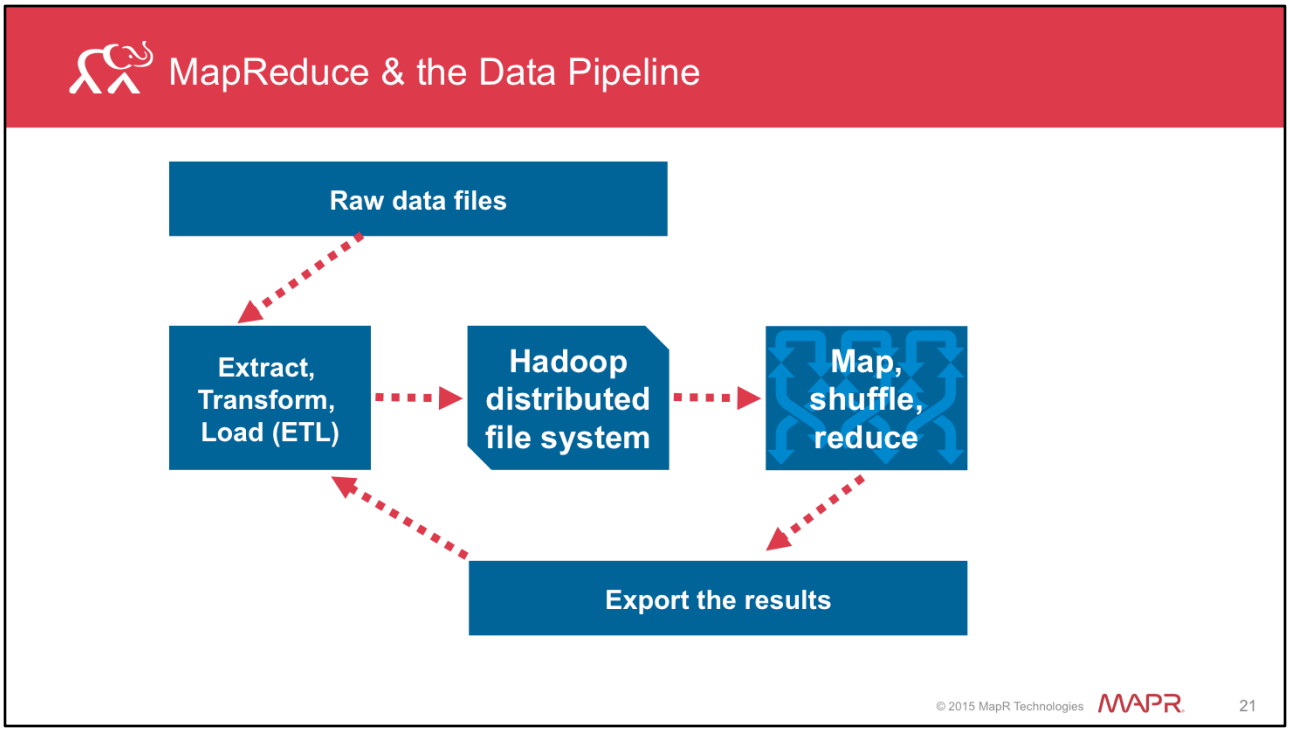
Next, well look at the tools in the data pipeline, and where Pig fits in.



Let's consider the data pipeline from the perspective of a data scientist at our Climate Research Institute.

The source of raw data will vary from one project to the next. One example would be the Internet of Things. The Internet of Things refers to networked sensors, such as those found in smart cars or weather stations, that automatically send their data to some central location, such as the Climate Research Institute servers.

However, this data may not be in a structured table, or it may need to be normalized through rounding or some other algorithm. Tools such as Pig, Storm, and Spark would be good choices for this. These tools can automate the process of data ingestion, normalization, structuring, and exporting into the Hadoop Distributed File System with one or more scripts.

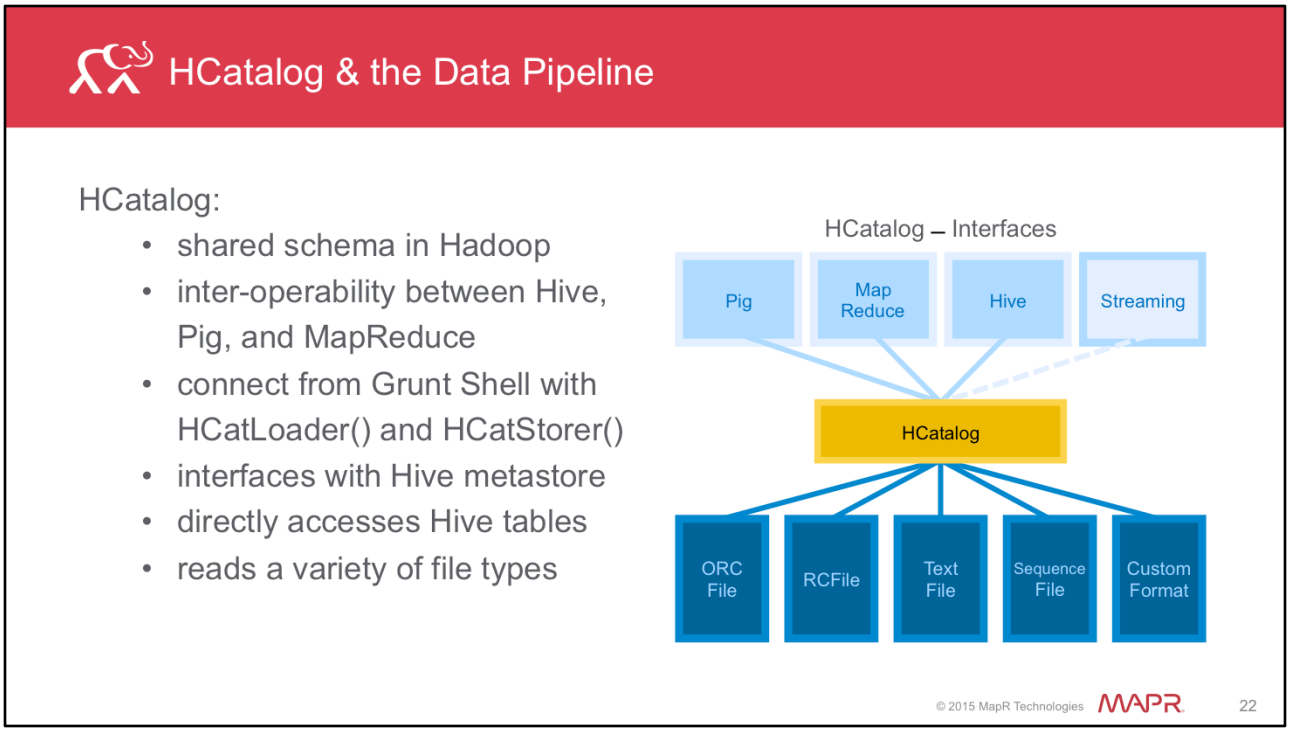


This diagram represents the data pipeline from a more technical perspective.

The data pipeline starts with raw data files. This might be click stream data from a website, log files from a server, JSON files, or any other type of data, including the raw sensor data from our weather stations.

In most cases, a raw data file must be extracted, transformed, and loaded into a Hadoop Distributed File System. This includes any data normalization processes you must perform, such as splitting a string, defining delimiters, or rounding, truncating, or otherwise aggregating data.

This ETL process may be automated through a series of Pig scripts. The exported results can then be loaded back into the



HCatalog provides an interface between the Hive metastore, where Hive table metadata is stored, and other data pipeline tools in the Hadoop ecosystem.

It provides a shared schema in Hadoop, providing interoperability between several Hadoop ecosystem tools such as Pig, Hive, and MapReduce.

You can connect to HCatalog from the Grunt shell with functions like HCatLoader and HCatStorer

You can directly access Hive tables or read and write a variety of file types using these functions.

When to use Hive or Pig?

Pig (and Pig Latin)	Hive (and HiveQL)
Procedural language	Non-procedural, declarative language
Used by both data analysts and developers	Used mostly by data analysts
Used to automate ETL for unstructured data	Used to run batch queries on structured data

Since Pig and Hive are often used together, and there is some overlap in their capabilities, many users want to know when to use one or the other.

Hive is a non-procedural, declarative language while Pig is a procedural language. In other words, Hive will process your query in whatever order is most optimized, while Pig will process your data in the order you specify. Pig's procedural nature makes it more suitable for certain ETL and normalization processes.

Hive is mostly used by data analysts to explore data sets, while Pig is used by both data analysts who want to clean their own data, and by developers who want to process data.

Hive runs batch queries on structured data while Pig can

When to use Pig or Spark?

Pig	Spark
More like Python, SQL	More like Scala
Runs on MapReduce	Runs in memory
Batch processes	Interactive

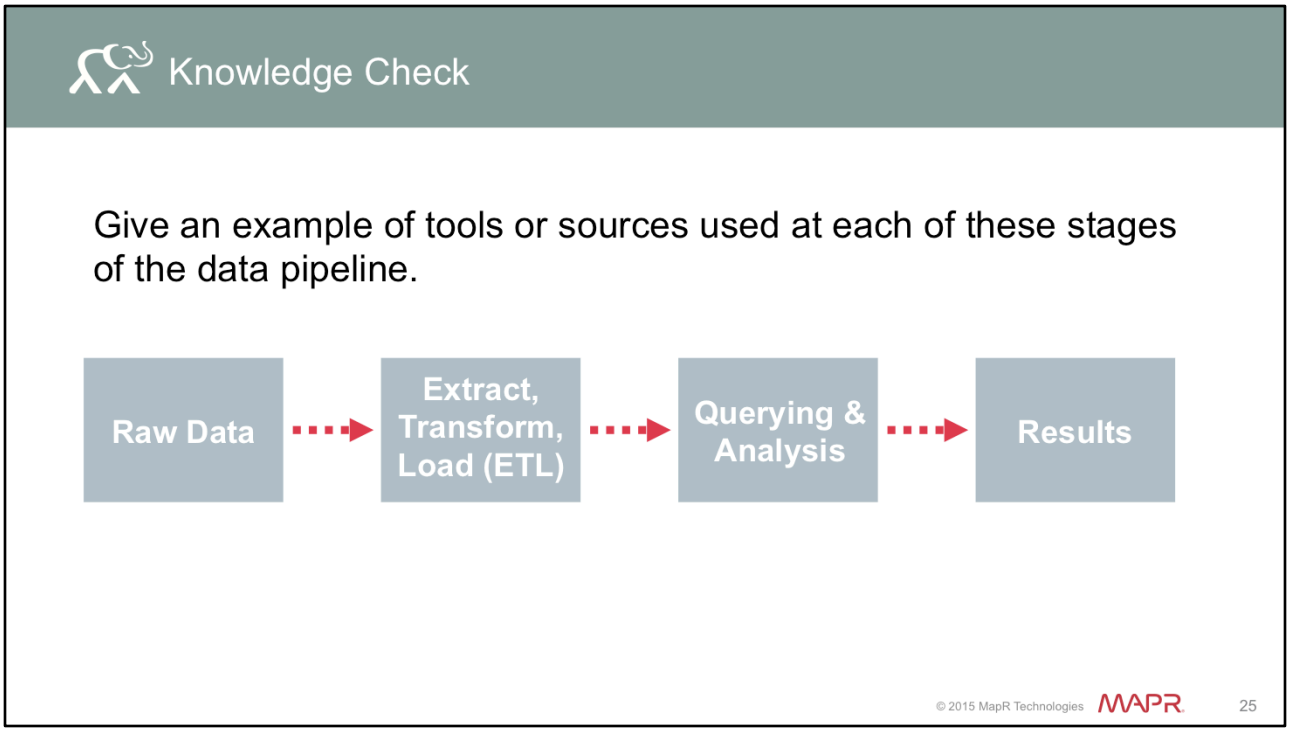
Spark

Pig and Spark are both used for data ingestion and the early steps of the ETL or data analysis pipeline. So when should you use Pig and when should you use Spark?

Pig is written in a higher level language, more similar to Python or SQL, while Spark is written in a much lower level language, similar to Java.

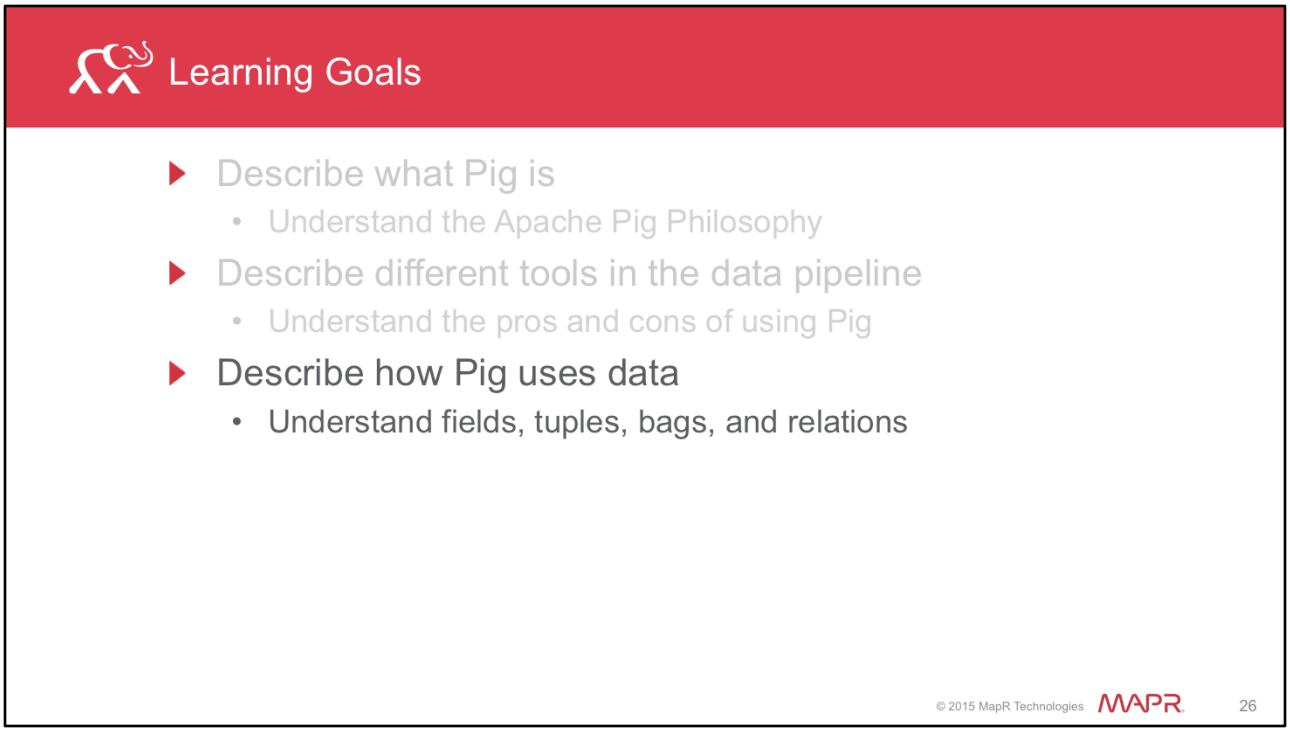
Pig runs batch processes and uses MapReduce. On the other hand, Spark runs in-memory and can therefore be more interactive.

Pig is great if you want to write simple data ingestion scripts. If you find yourself preforming the same ETL process repeatedly, you can run it in batches and automate the process with Pig.

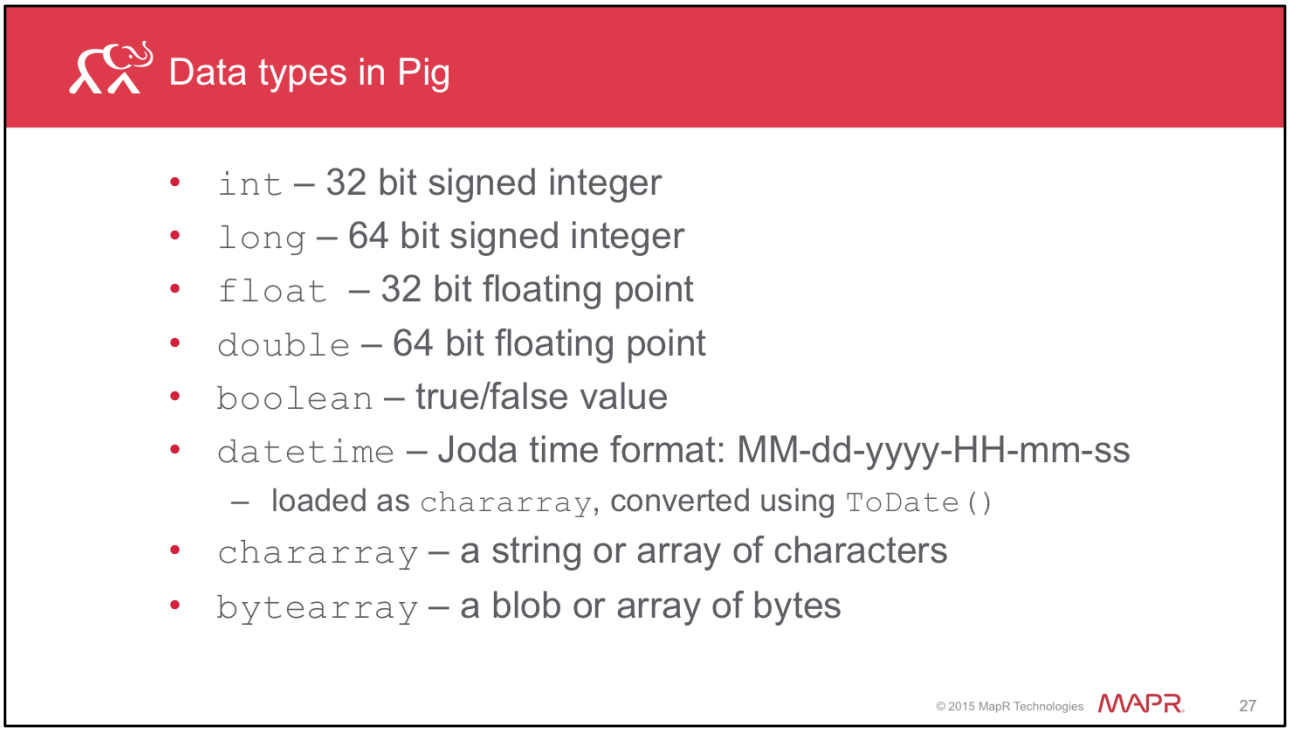


Give an example of tools or sources used at each of these stages of the data pipeline.

Possible Answers
Box 1: Internet of Things
Box 2: Pig, Spark
Box 3: Hive, Drill
Box 4: graph/report



Next, we'll look at how Pig uses data.

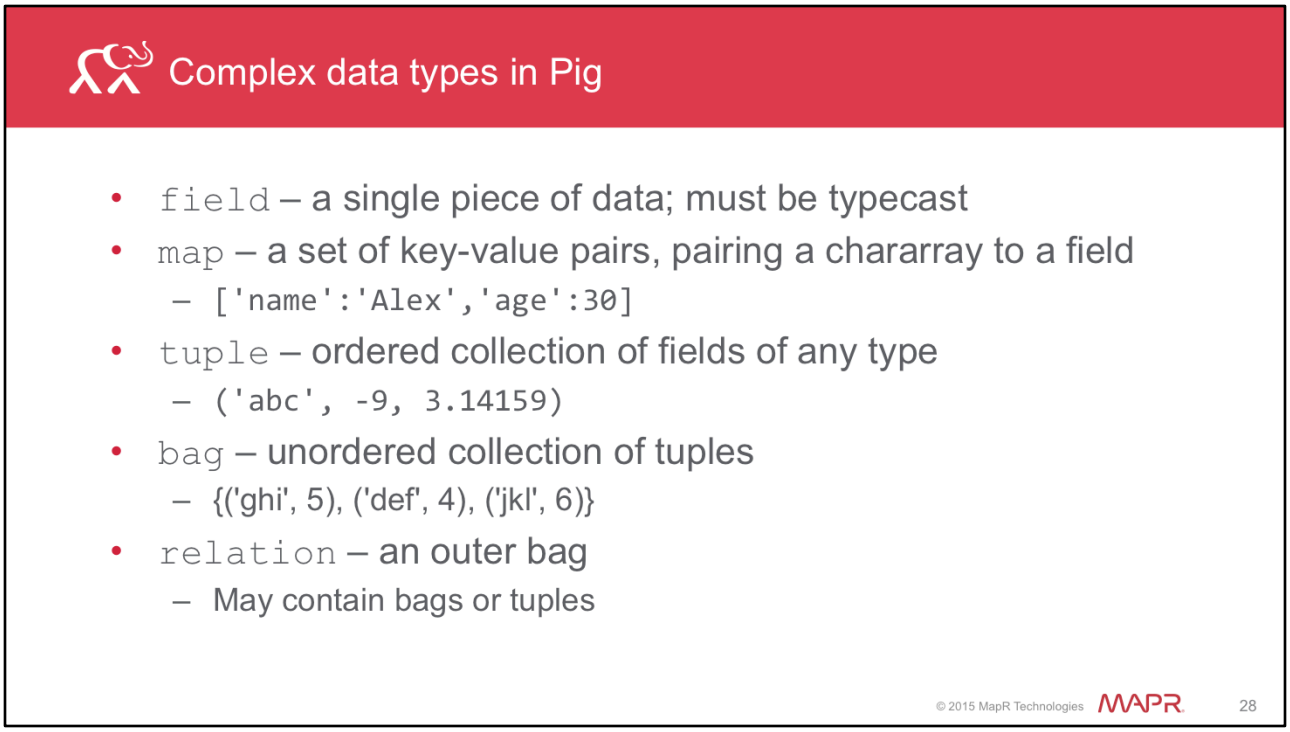


Pig is a somewhat strongly typed language, so you must specify whether each field is an integer, float, chararray, or other data type.

Simple data types in Pig might be familiar to you from other programming languages like Java. These include numeric data types such as INT, LONG, FLOAT, and DOUBLE. These correspond to various sizes of whole numbers and various levels of decimal point precision.

The boolean data type represents a dichotomy, such as 0 and 1, yes or no, or true or false.

The datetime data type uses the Joda time format, which is similar to unix time. Note, however, that dates should be loaded as chararrays, then converted.



every piece of data is loaded into a field; every field's data type must be specified.

a map is a set of key-value pairs, which pairs a character array to another field of any type.

a tuple is an ordered collection of fields. There can be any number of fields of any type in a tuple; you can have a tuple of size 1 or greater, but not zero. There can even be multiple different data types in a single tuple.

a bag is an unordered collection of tuples.

Finally, a relation is an outer bag, which can contain bags and tuples. Relations in Pig work similarly to tables in SQL or Hive. For the most part, we will be working with relations in Pig.

File formats in Pig

- PigStorage
- JSONStorage
- HBaseStorage
- BinStorage
- PigDump
- custom formats

PigStorage is the default storage type in Pig, and will be used if no storage format is specified when loading or storing data.

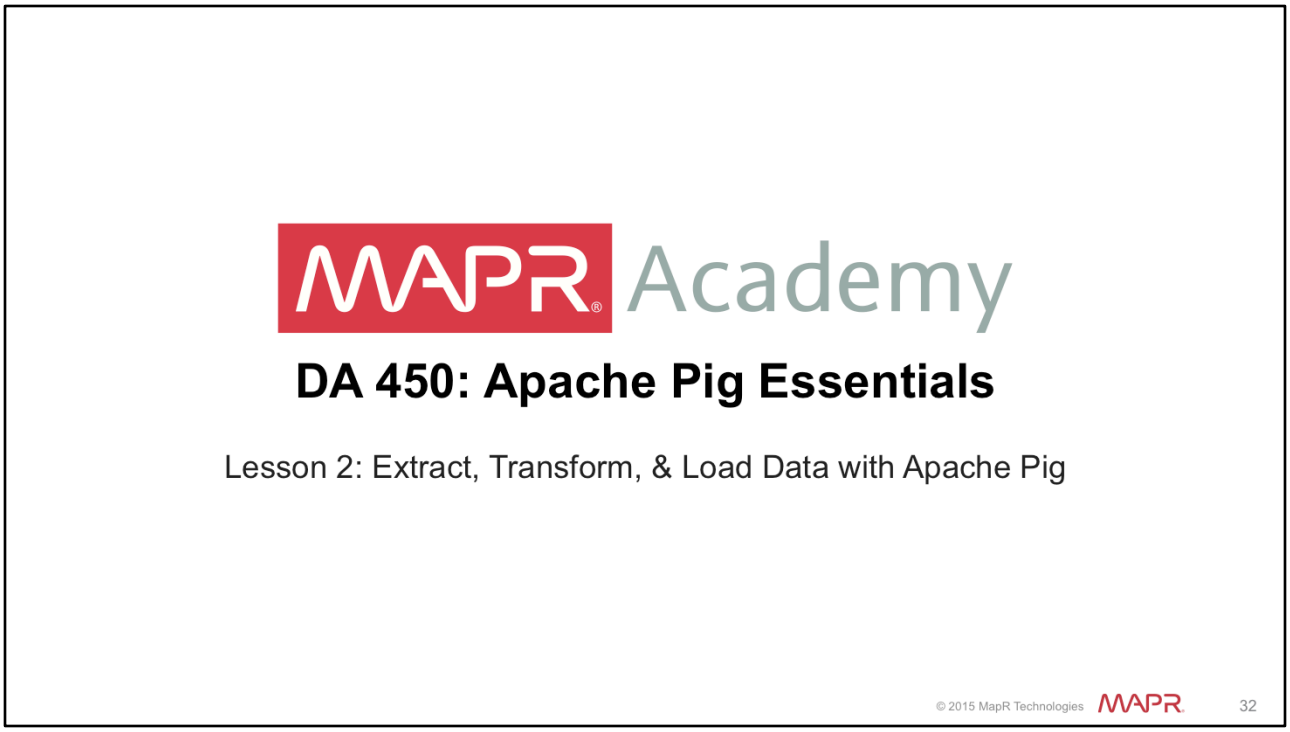
Other commonly used file formats include JSON, HBase, BinStorage, PigDump, as well as custom formats.

Field & record delimiters

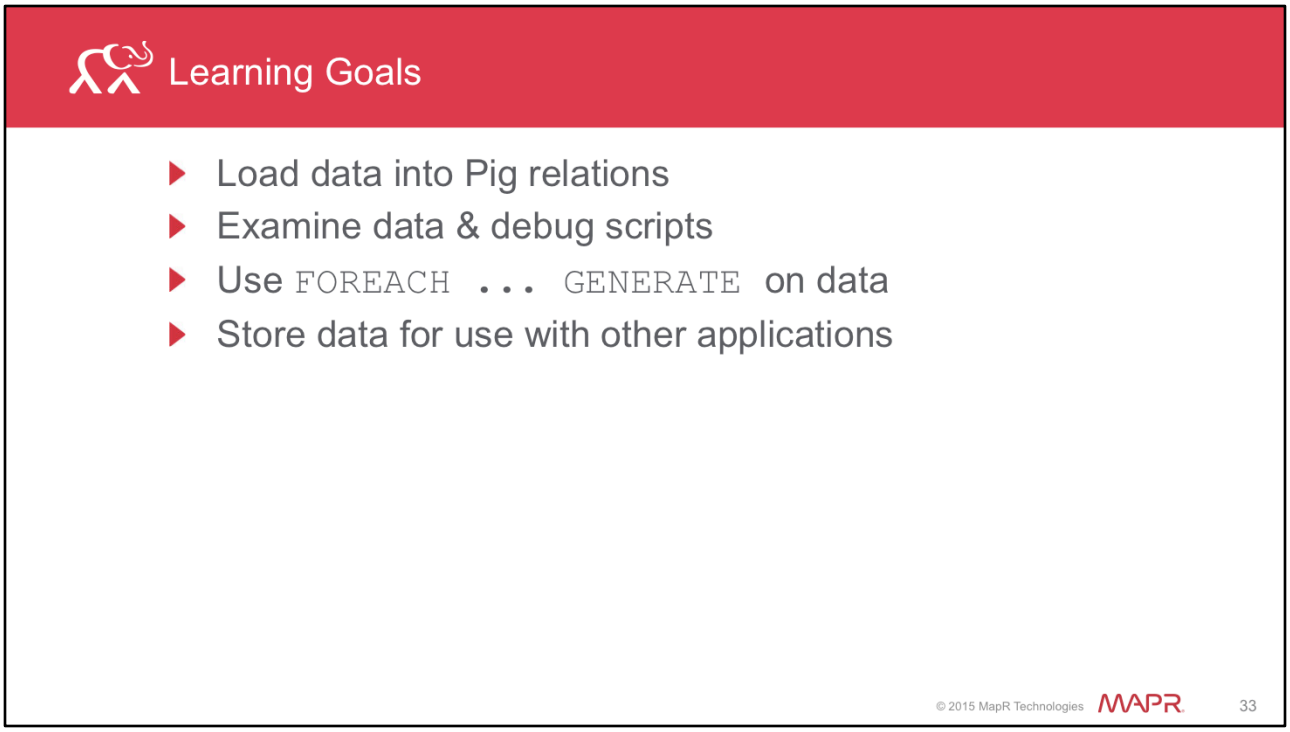
- Field delimiters
  - Tab ('\t')
  - Comma (',')
  - Pipe ('|')
  - Ctrl-A (\u0001)
- Record delimiters
  - Line feed ('\n')
  - Carriage return ('\r')
  - Both ('\n\r')

When loading or storing data, it is also important to know how your data is delimited. For example, fields are commonly delimited by tabs, commas, or pipes, while records are commonly delimited by new lines, carriage returns, or both.

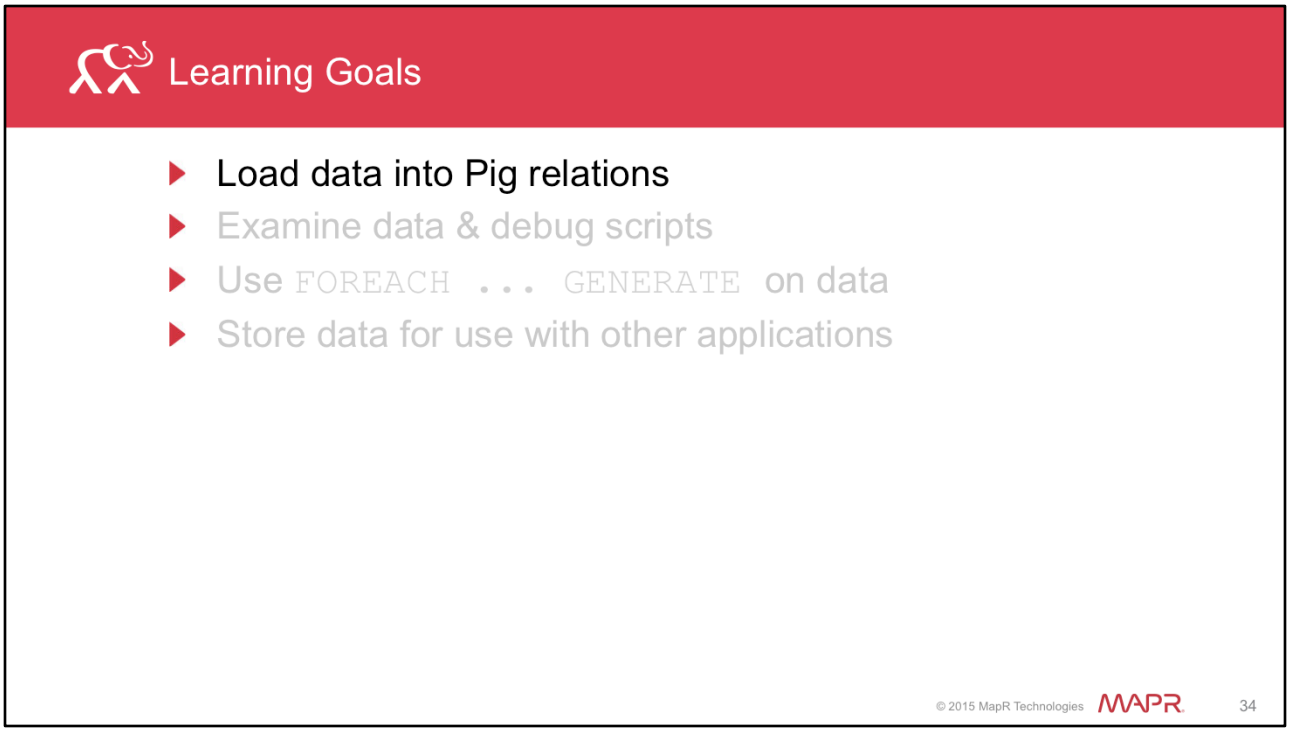
In the next lesson, we will explore how to extract, transform, and load data with Apache Pig



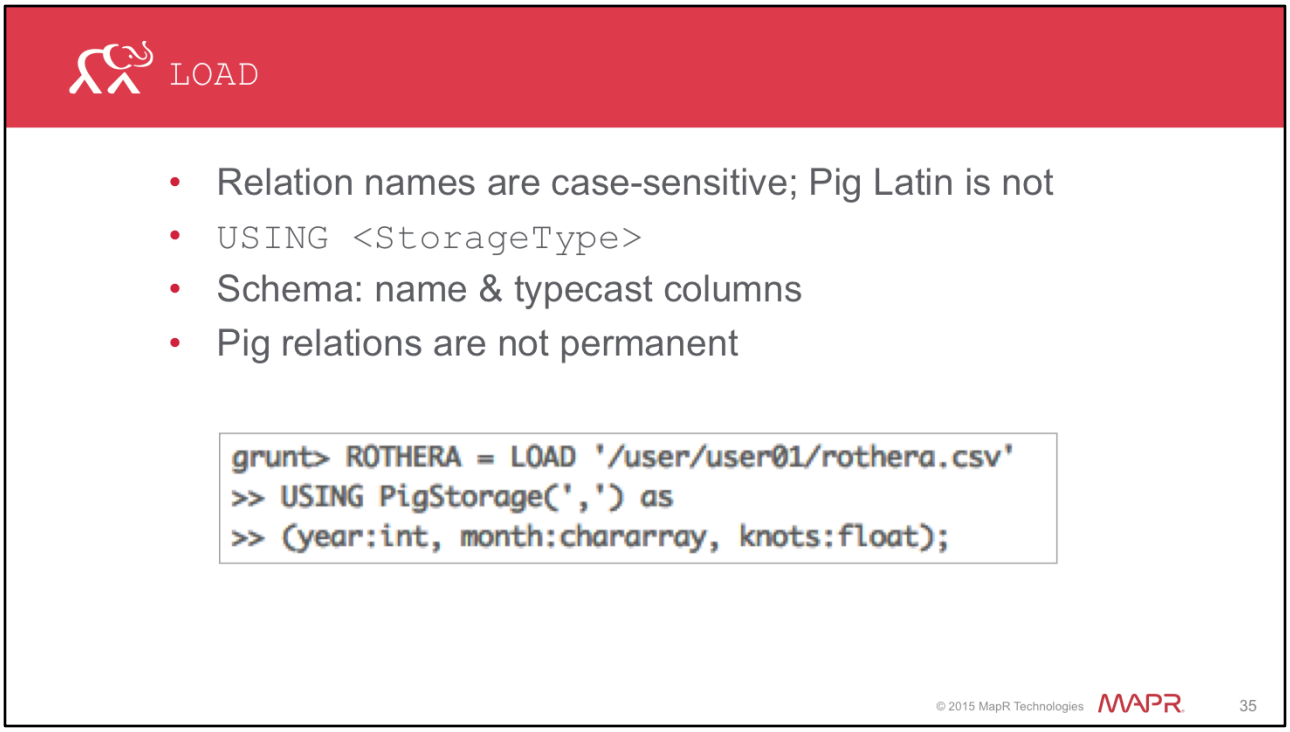
Welcome to DA 450 Lesson 2: Extract Transform and Load data with Apache Pig. In this lesson, we learn about the ETL process in Pig.



When you have completed this lesson, you will be able to:
Load data into Pig relations
Examine data and debug scripts
Use FOREACH ... GENERATE on data
Store data for use with other applications



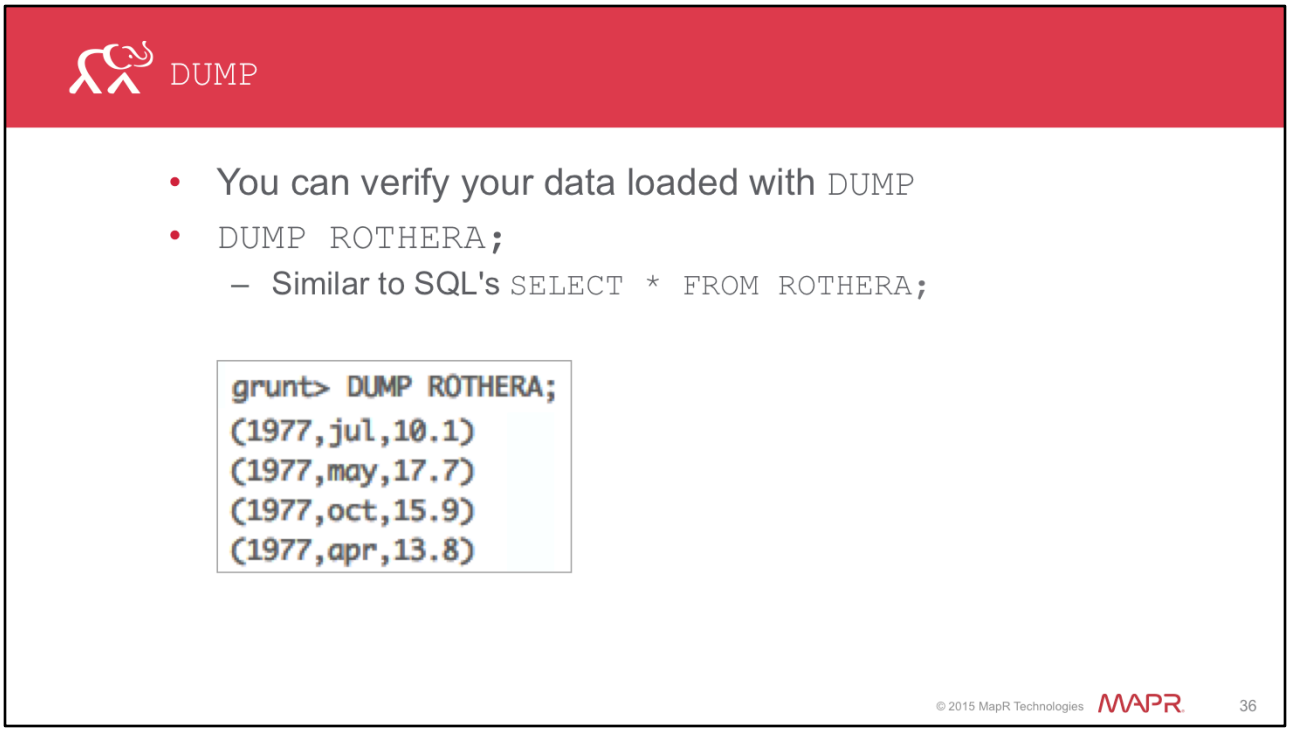
Let's start by loading data into Pig relations



Here we see an example of loading a a CSV file into a Pig relation which we have named ROTHERA. This file contains information about the windspeed, measured in knots, from the Rothera weather station in Antarctica.

Note: that both the Pig relation name and the Pig Latin keywords, such as "LOAD" and "USING" are in all-caps. Like SQL, this is by convention, and is not a requirement. However, note that relation names are case-sensitive, while Pig Latin itself is not.

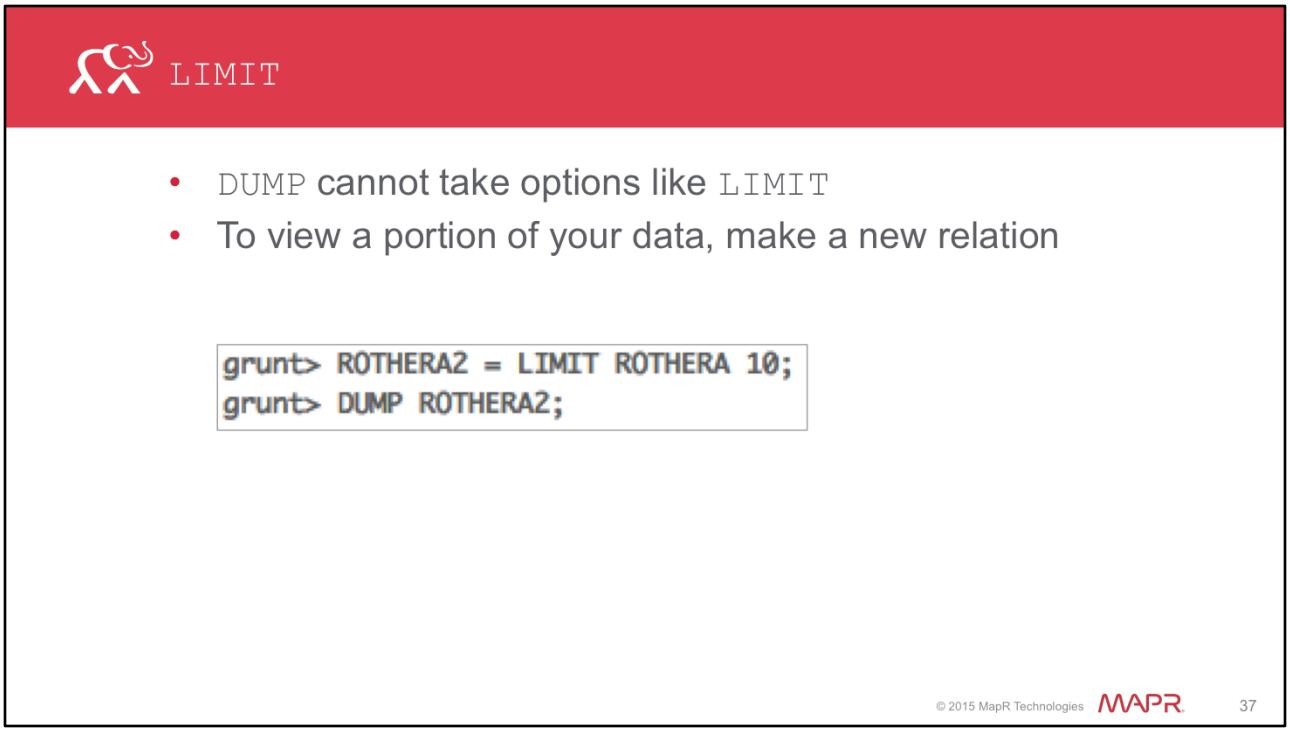
Once you have named your relation and specified the location of the data, you can optionally specify how the file is stored with "USING". This may include delimiters and formats such as PigStorage, HBase, JSON, or others.



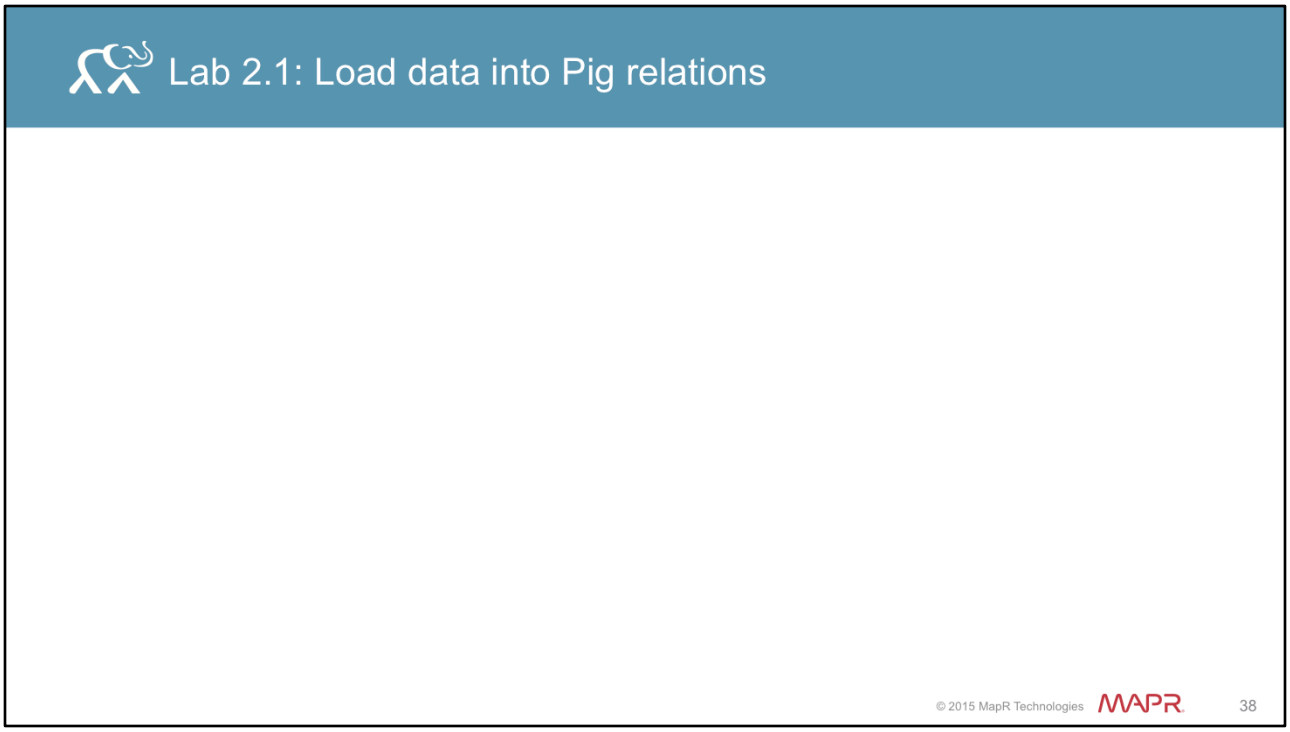
Once you have loaded data, you can verify it was loaded correctly by using DUMP. The DUMP command works similarly to the SELECT * command from SQL.

DUMP will start a MapReduce process, so it may take a moment.

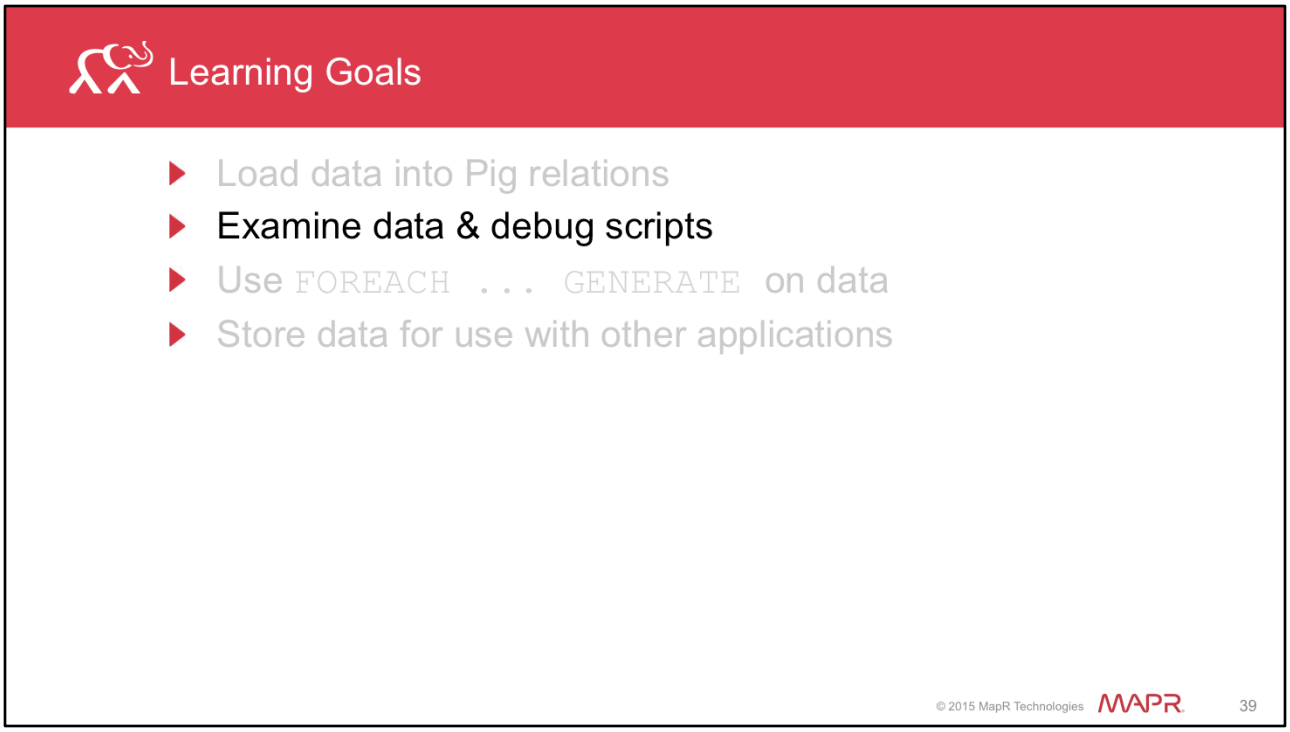
It is important to note that DUMP will display all the data in your relation. Depending on the size of your dataset, this may be quite substantial.



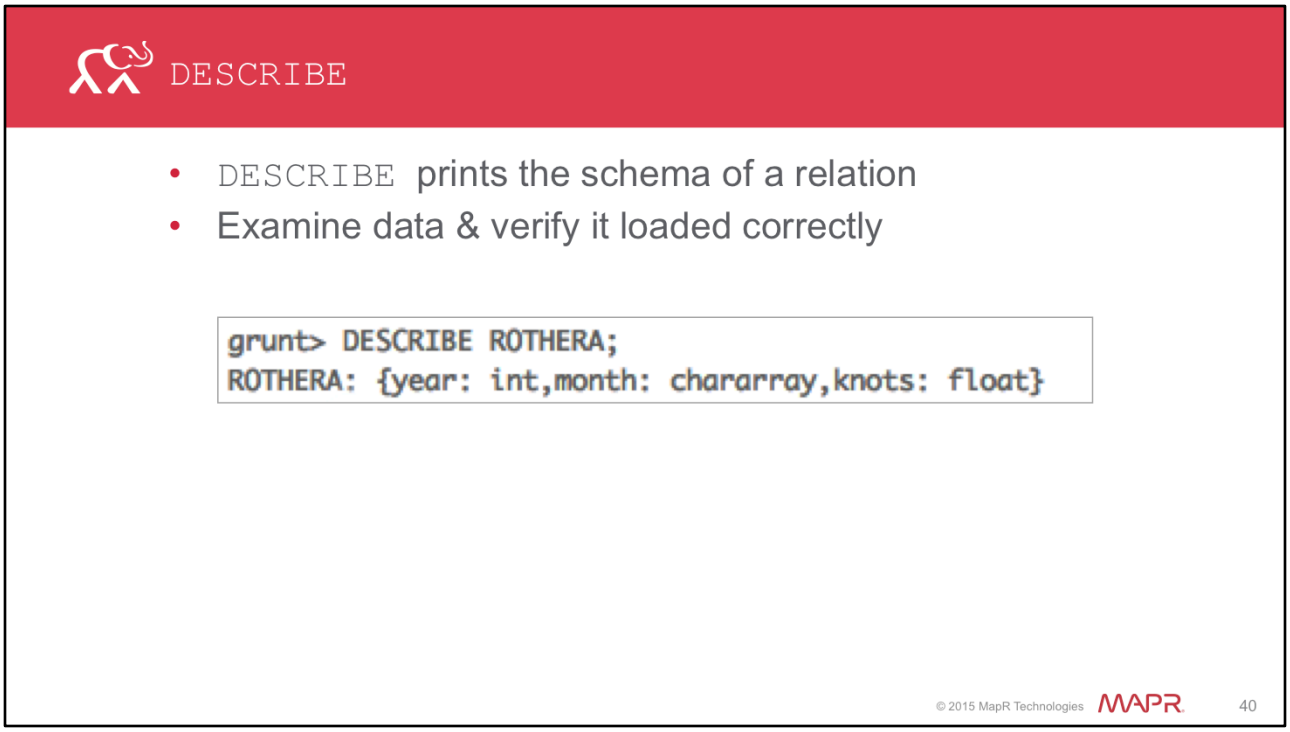
Unlike SELECT * in SQL, DUMP cannot take any options. If you only want to view a portion of your data, you can create a new relation and dump that. For example, you can limit the ROTHERA relation to just 10 rows and save it as ROTHERA2, as shown.



In this lab, you will load some data into your first Pig relation using the Grunt shell. Open your lab guide, and complete Lab 2.1. When you are finished, you are ready to continue with the lecture.

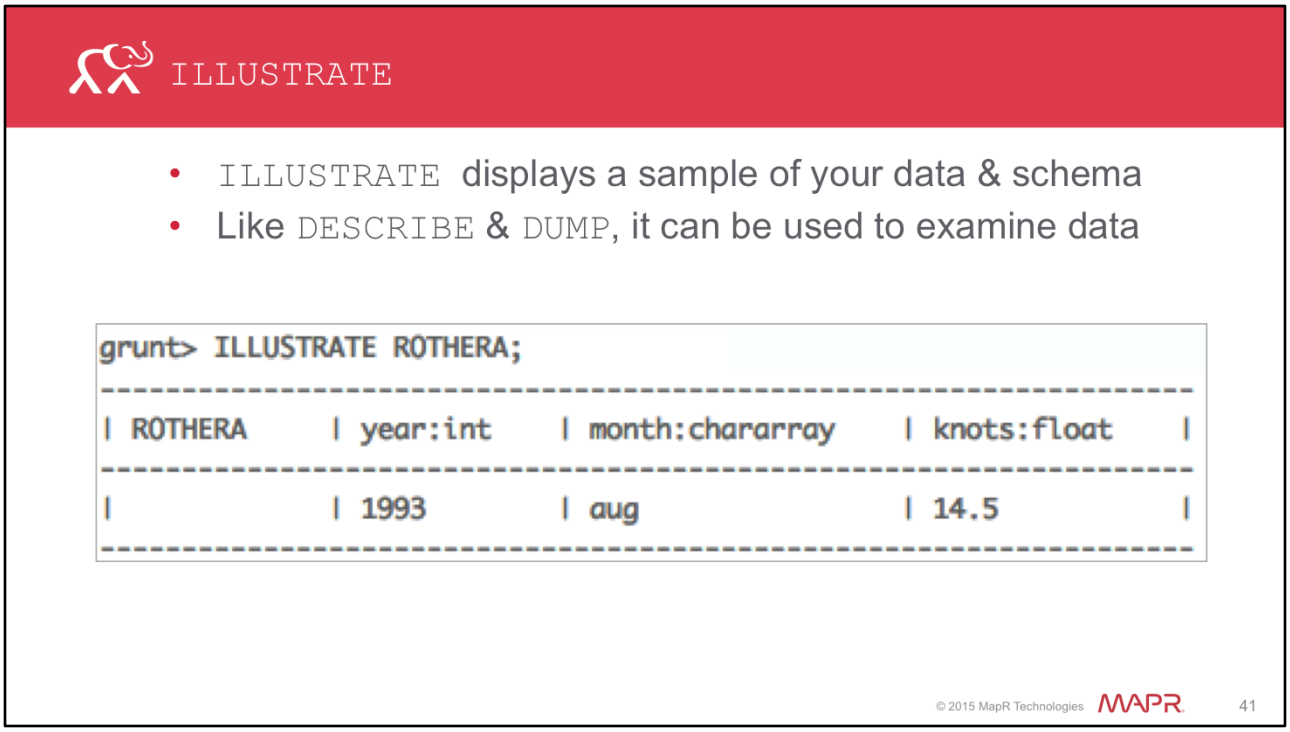


Next we'll learn more ways to examine data that has been loaded into a Pig relation, and how to debug your Pig scripts.



The DESCRIBE keyword prints a very simple schema of your relation. You will notice that this format is similar to the schema we used to load the relation in the previous section.

DESCRIBE does not start a MapReduce job, so it is very fast. This makes it a great tool to verify that your data loaded correctly, especially after several transformations. You can use this simple schema print to check that all the columns you want exist and are typecast correctly.



Like DESCRIBE and DUMP, the ILLUSTRATE keyword can help you verify that your data are loaded correctly.

Like DUMP, but unlike DESCRIBE, ILLUSTRATE will trigger a MapReduce job. Therefore, it may take a moment to return a result.

The advantage of ILLUSTRATE is that it combines some of the features of both DESCRIBE and DUMP. It displays, in a visual way, the columns in your Pig relation, including their names, data types, and a one-row sample of the data. This way you can see the actual data as well as the schema in your relation.

EXPLAIN

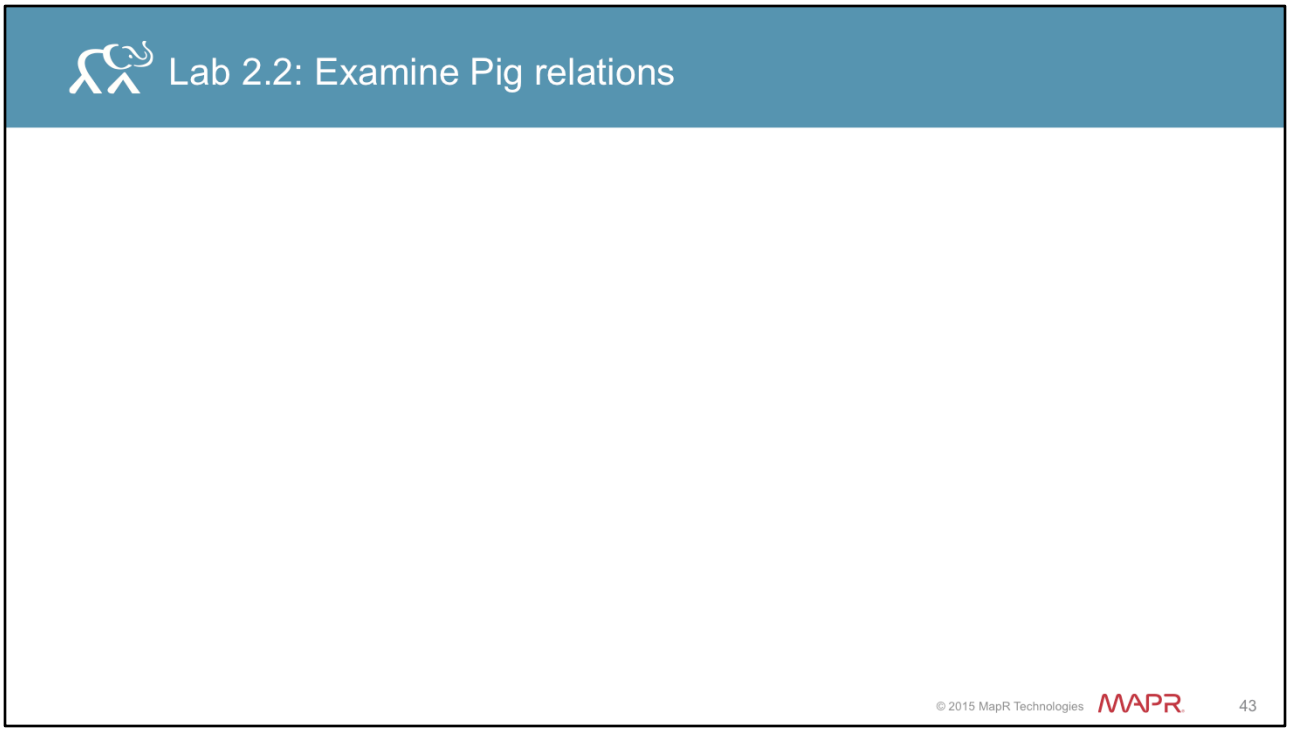
- DUMP, DESCRIBE or ILLUSTRATE relations
- EXPLAIN relations or Pig Latin scripts
- EXPLAIN is an advanced debugging tool that explains the dataflow, including MapReduce jobs

```
#-----------------------------------------------
# Map Reduce Plan
#-----------------------------------------------
MapReduce node scope-64
Map Plan
ROTHERA: Store(fakefile:org.apache.pig.builtin.PigStorage) - scope-63
|
|---ROTHERA: New For Each(false,false,false)[bag] - scope-62
    |   |
    |   Cast[int] - scope-54
    |   |
    |   |---Project[bytearray][0] - scope-53
    |   |
    |   Cast[chararray] - scope-57
    |   |
    |   |---Project[bytearray][1] - scope-56
    |   |
    |   Cast[float] - scope-60
    |   |
    |   |---Project[bytearray][2] - scope-59
    |
    |---ROTHERA: Load(/user/user01/rothera.csv:PigStorage(',')) - scope-52
Global sort: false
```

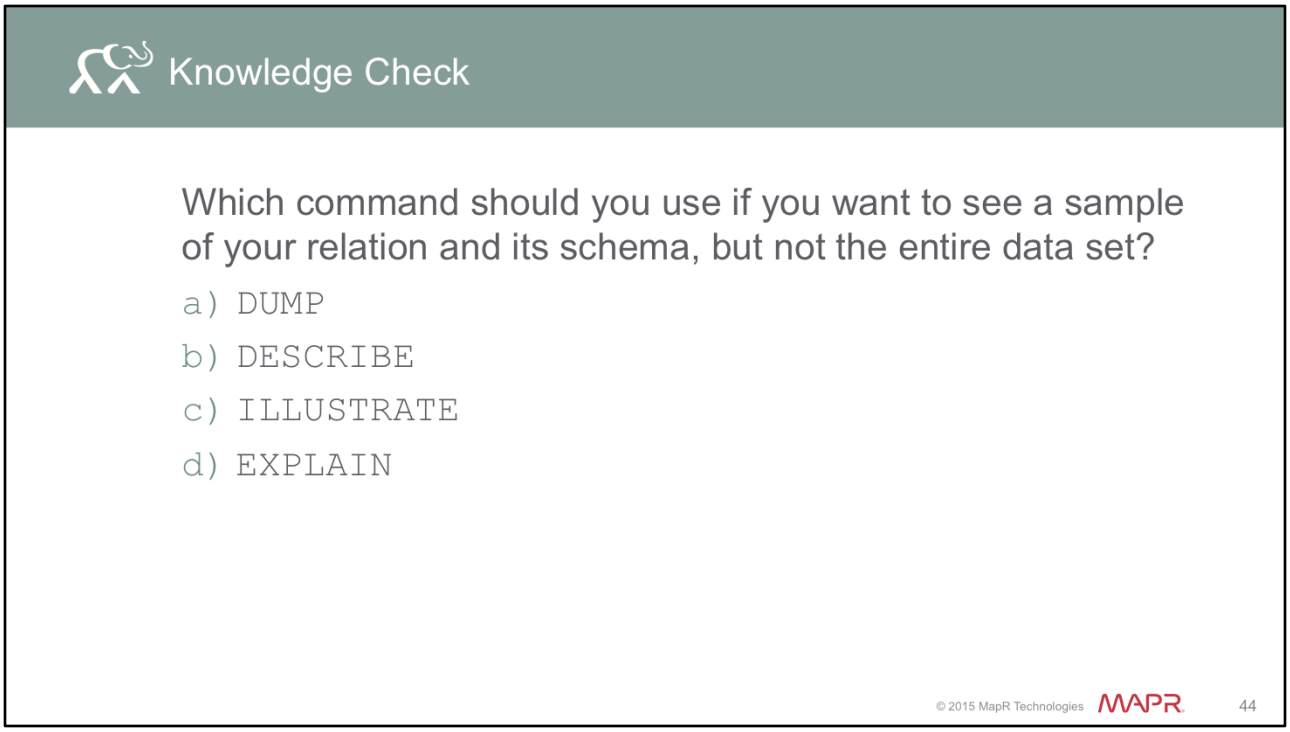

While you can only use DUMP, DESCRIBE, or ILLUSTRATE on a pig relation, you use EXPLAIN on relations or an entire Pig Latin script.

EXPLAIN displays the logical, physical and MapReduce execution plans for a relation or Pig Latin script.

Here you can see a portion of the EXPLAIN output for the ROTHERA relation. As you can see, this is a more detailed and technical printout than ILLUSTRATE or DESCRIBE. We won't be using EXPLAIN in this course, but it is good to be aware of as you automate ETL processes and try to optimize your dataflow for speed.

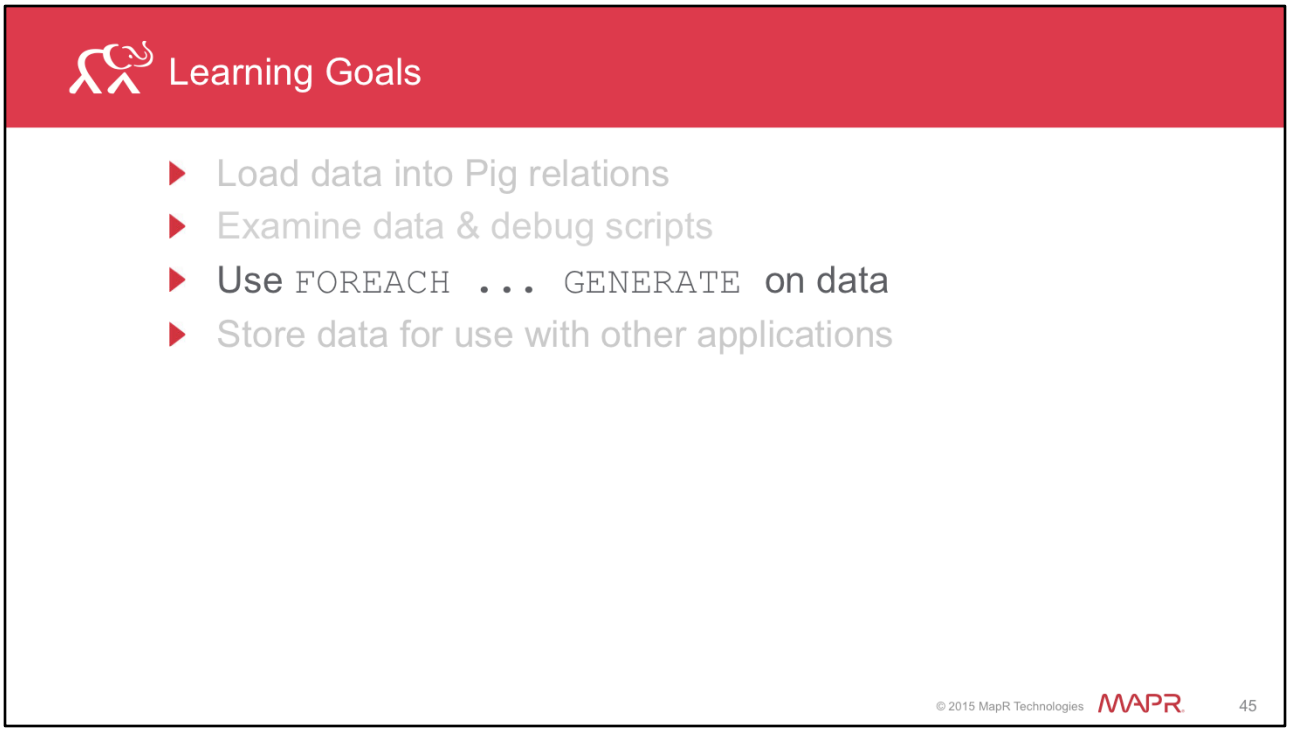


Open your lab guide, and complete Lab 2.2. When you are finished, you are ready to continue with the lecture.

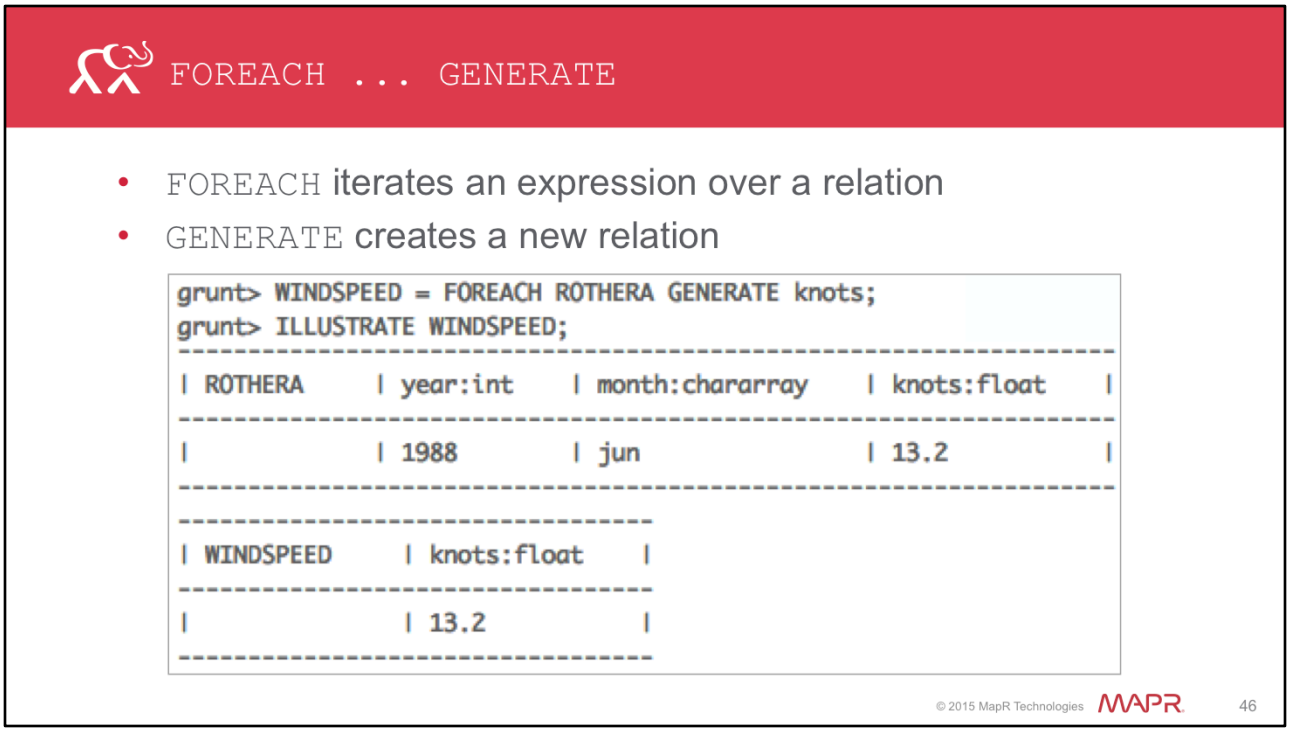


Answer: C

INSTRUCTORS: briefly discuss some use cases of each of these commands with your students



Next, we'll begin manipulating data with FOREACH ... GENERATE



The FOREACH keyword iterates an expression over an entire relation. FOREACH is often used in conjunction with GENERATE. GENERATE creates a new relation from the manipulations preformed by FOREACH.

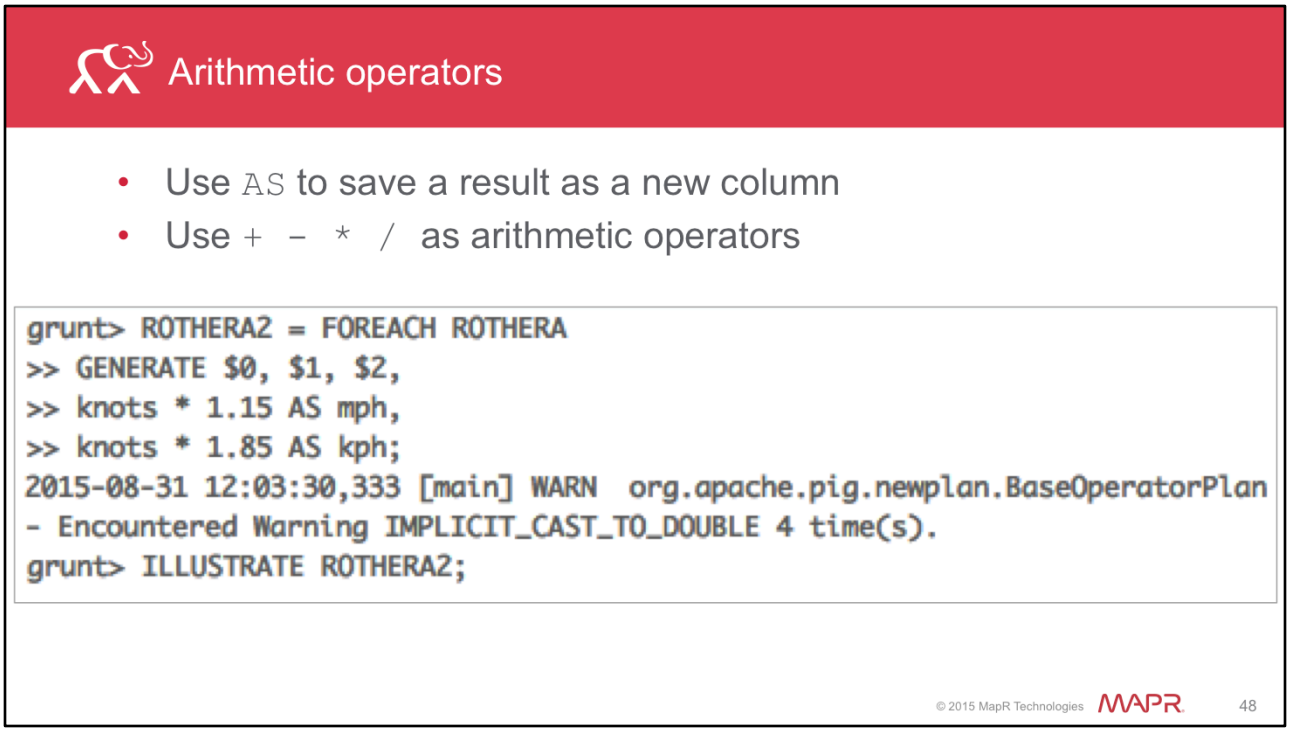
Let's say we wanted to extract just the knots column from the ROTHERA relation, and rename it as WINDSPEED. In this example, we use WINDSPEED equals FOREACH ROTHERA GENERATE knots. When we use ILLUSTRATE on the new WINDSPEED relation, Pig prints out the schema for the new relation, and the original ROTHERA relation as well. This way, you can see how the two relations are related. This can be useful when multiple transformations have been preformed on a relation.

Column numbers

- Use column numbers, as in `$2` (count from zero)

```
grunt> WINDSPEED2 = FOREACH ROTHERA GENERATE $2;
grunt> ILLUSTRATE WINDSPEED2;
-------------------------------------------------------------------------
| ROTHERA     | year:int    | month:chararray    | knots:float    |
-------------------------------------------------------------------------
|             | 1999        | jan                | 12.5           |
-------------------------------------------------------------------------
-----------------------------------
| WINDSPEED2     | knots:float    |
-----------------------------------
|                | 12.5           |
-----------------------------------
```

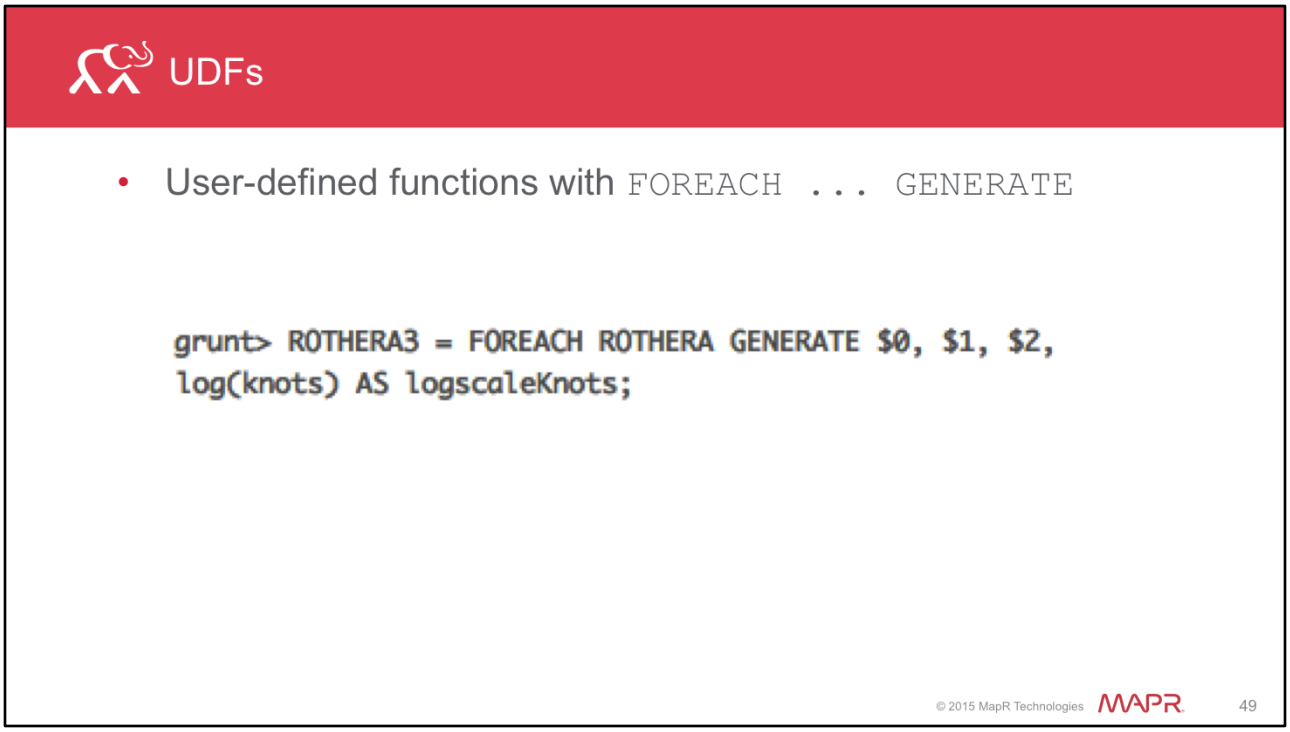

In the last example, we called the column by its name, knots. We can also call columns by their number, counting from zero, as in the example shown here. Since the knots column is the third in our dataset, counting from zero, we can call it by using dollar-two.



In addition to selecting columns from a relation with FOREACH, there are a number of other operations you might preform on your data as part of an ETL process.

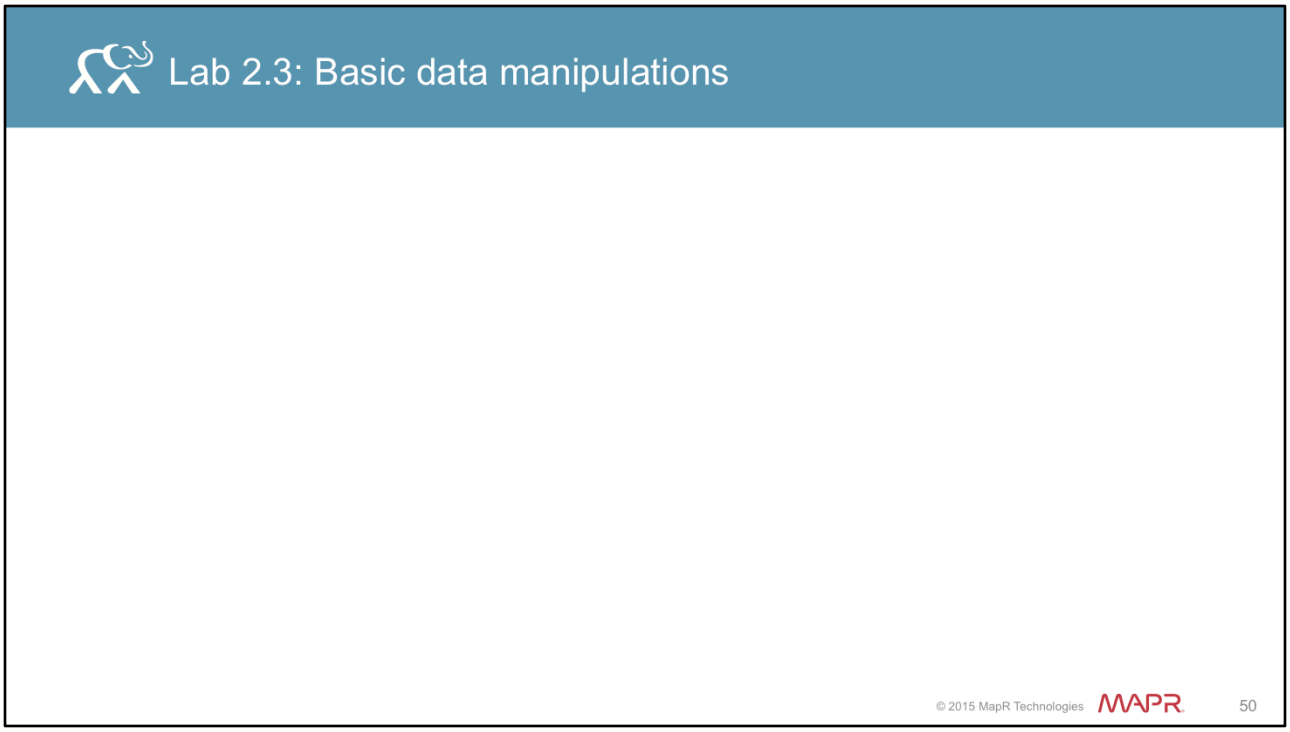
One example might be to add a column with some sort of manipulated data. In this example, we make a new relation called ROTHERA2. This contains all the same information as the original ROTHERA relation, with two new columns. The first new column is MPH, which includes the knots column converted into miles per hour. The second new column is KPH, which is the knots column converted into kilometers per hour.

Since we did not explicitly state what type of our new columns, Pig implicitly casts them as doubles. We can see this, as Pig provides a warning message after we generate the new relation. We can verify this when we use illustrate to examine

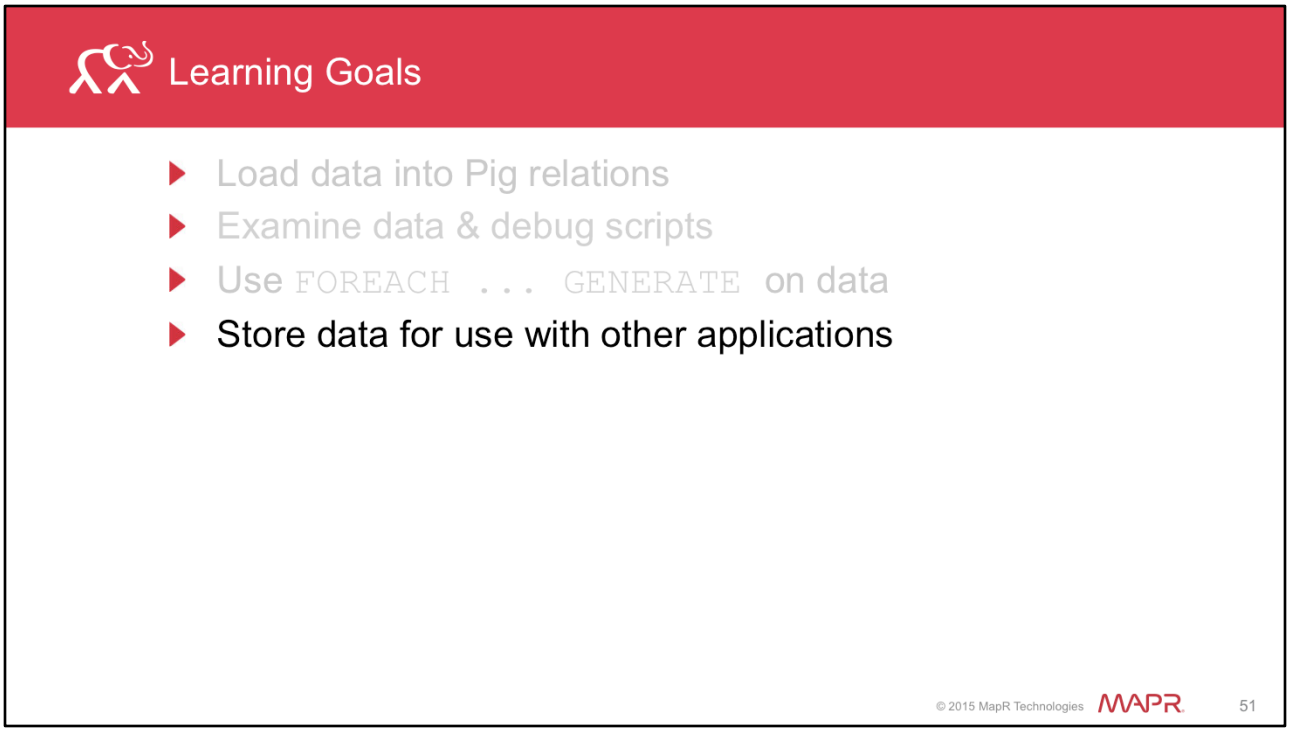


In addition to arithmetic operators, you can also use User-defined functions with FOREACH GENERATE.

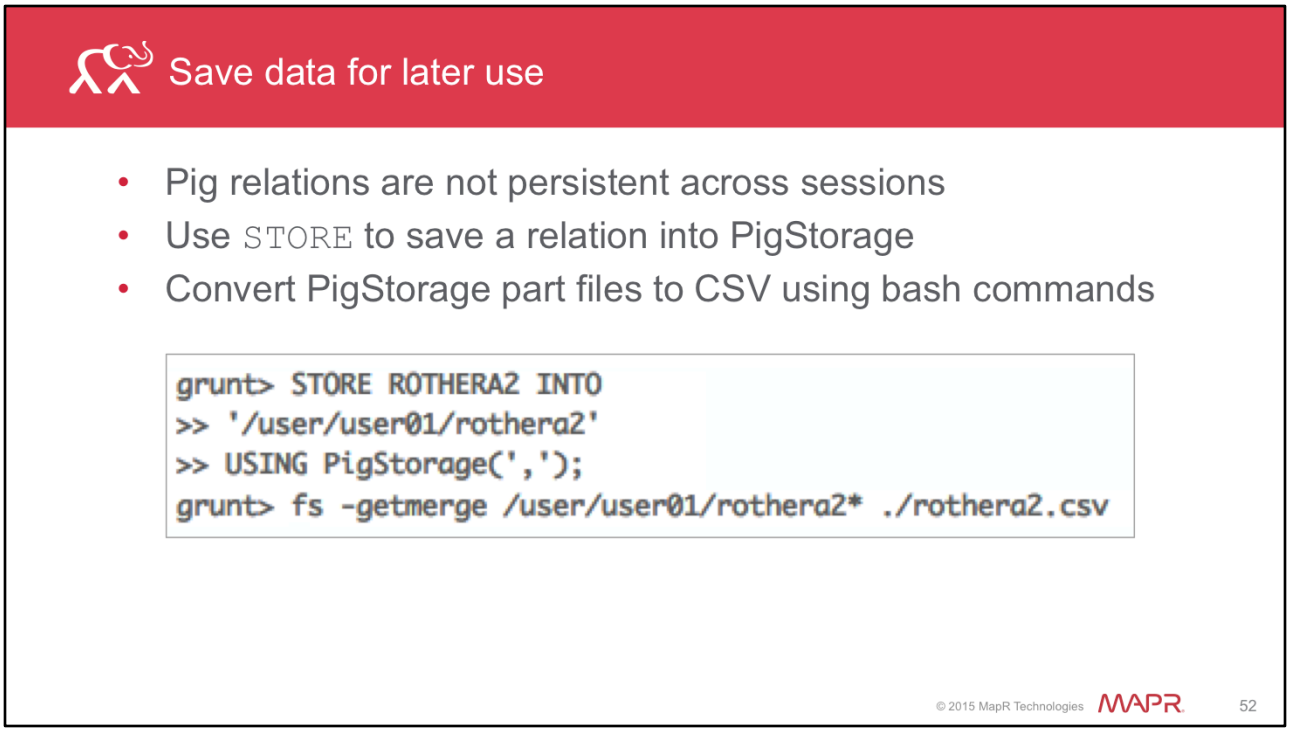
In this example, we use the log() UDF to create a new column called logscaleKnots.



Open your lab guide, and complete Lab 2.3. When you are finished, you are ready to continue with the lecture.



Lastly, we will learn how to store data in Pig.



Recall that pig relations are not persistent across sessions. Therefore, it is important to store data as a final step of an ETL process.

Since we don't know whether we'll ultimately need to report our climate data in knots, miles per hour, or kilometers per hour, we would like store the ROTHERA2 relation for later use.

We use STORE to chunk our relation into several parts, which get saved together in a single folder we create.

Like the LOAD command, there are several options for STORE, such as USING PigStorage or USING HBaseStorage. By default, Pig will store data using PigStorage and as a tab-delimited file.

It is possible to use some bash commands directly from the

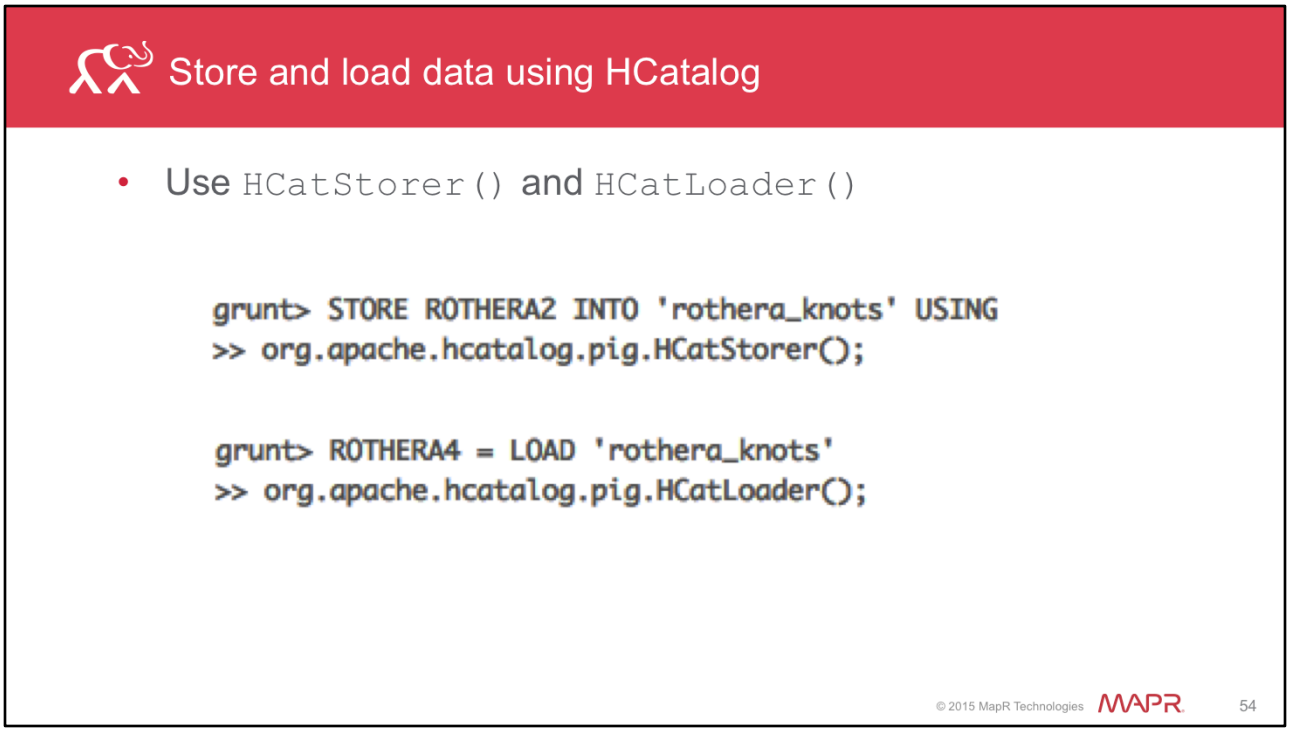
Reload stored data

- Use `LOAD` & `ILLUSTRATE` to load the new file

```
grunt> ROTHERA3 = LOAD '/user/user01/rothera2.csv' USING PigStorage(',') as
>> (year:int, month:chararray, knots:float, mph:float, kph:float);
grunt> ILLUSTRATE ROTHERA3;
-------------------------------------------------------------------------------------------------------
| ROTHERA3     | year:int     | month:chararray     | knots:float     | mph:float          | kph:float     |
-------------------------------------------------------------------------------------------------------
|              | 2010         | apr                 | 13.5            | 15.524999999999999 | 24.975        |
-------------------------------------------------------------------------------------------------------
```


Now we can make a new relation, ROTHERA3, and load our new file, rothera2.csv.

This time, we specify that MPH and KPH should be cast as floats. As we can see when we use ILLUSTRATE, they are recast as floats, even though they were originally saved as doubles.

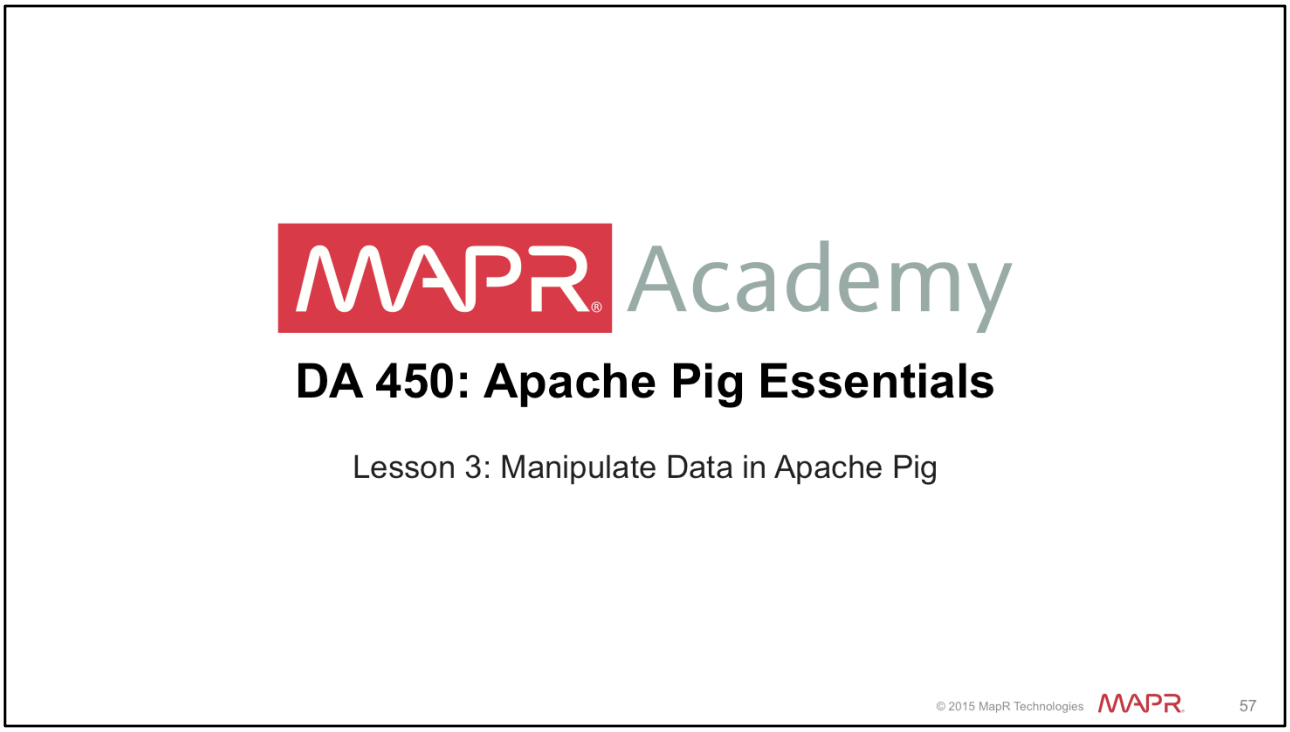


In addition to loading and storing data as files, you can also load and store data using HCatelog, as demonstrated here.

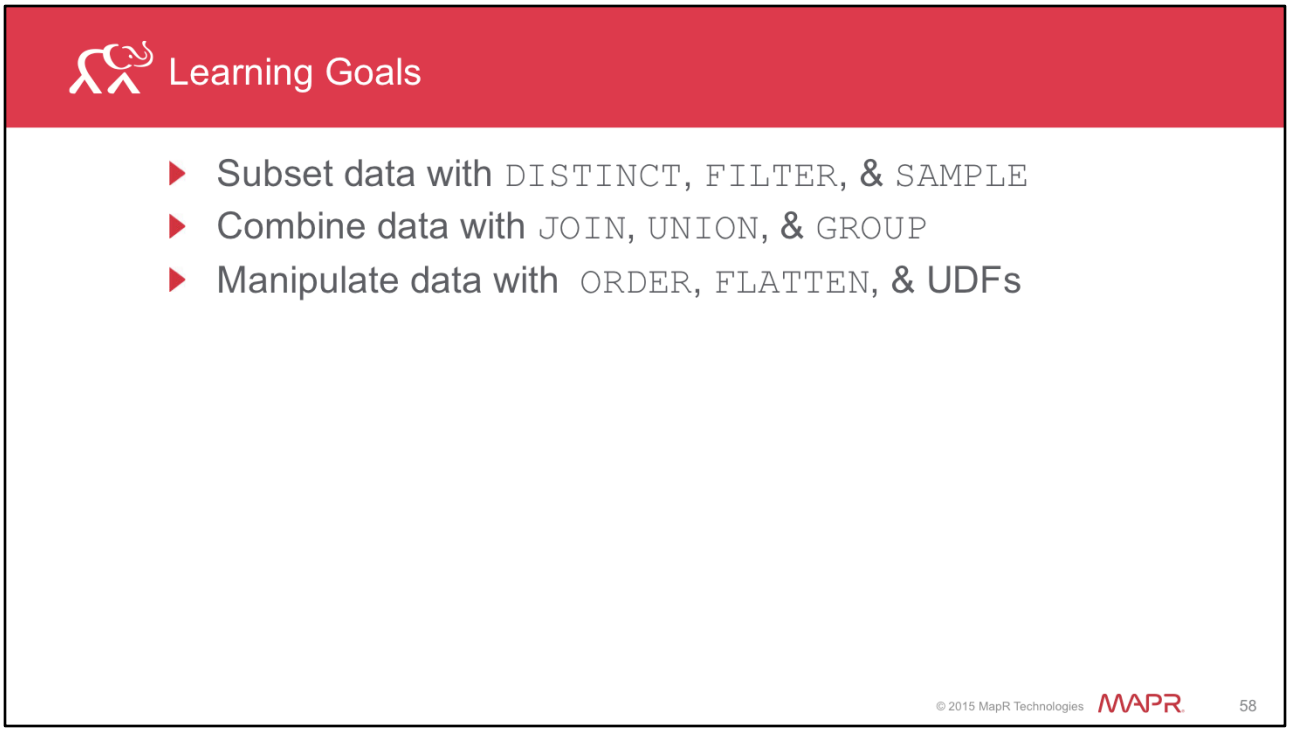
If you haven't already, complete the remainder of the Lesson 2 labs. If you finish early, you might try the more advanced ETL described at the end of the Lesson 2 lab guide, which involves combining multiple raw data files into a single relation. You might also like to use FOREACH ... GENERATE to freely explore the dataset provided.

When you are finished with the Lesson 2 lab exercises, you are ready to move on to Lesson 3.

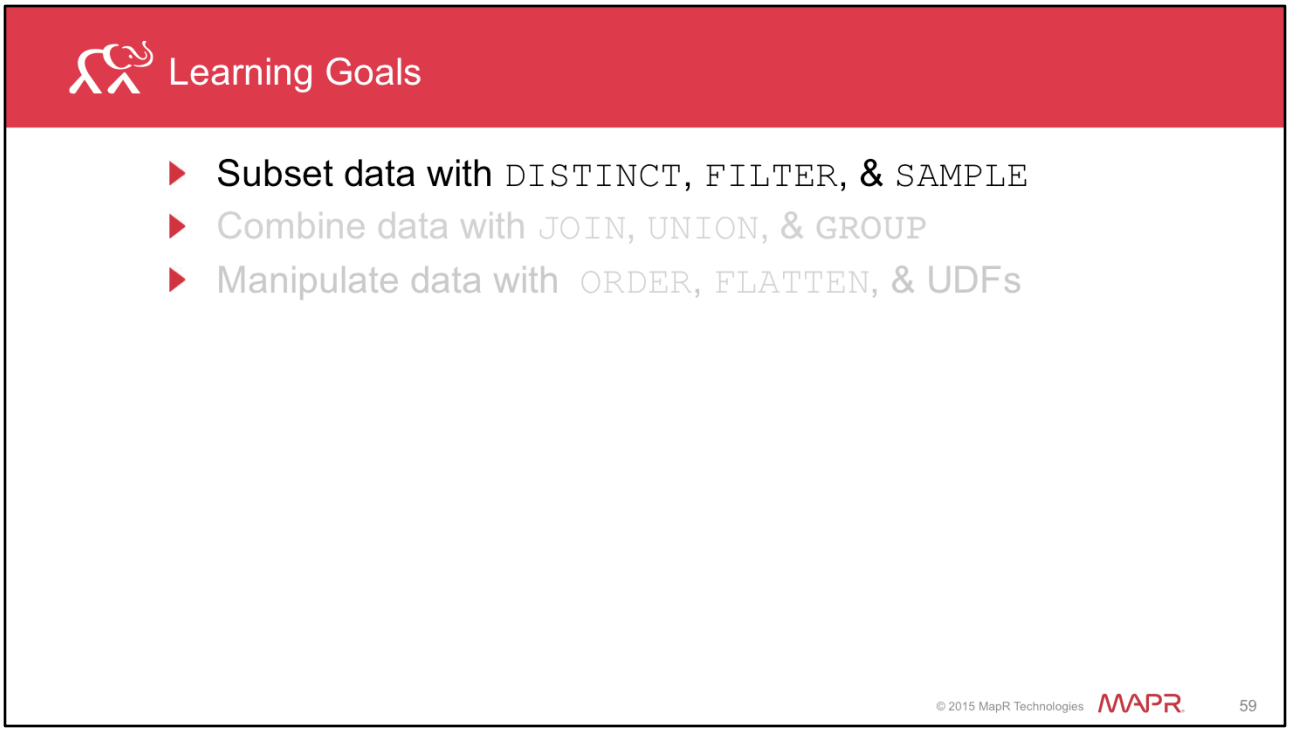
In the next lesson, we'll learn how to manipulate data in Apache Pig.



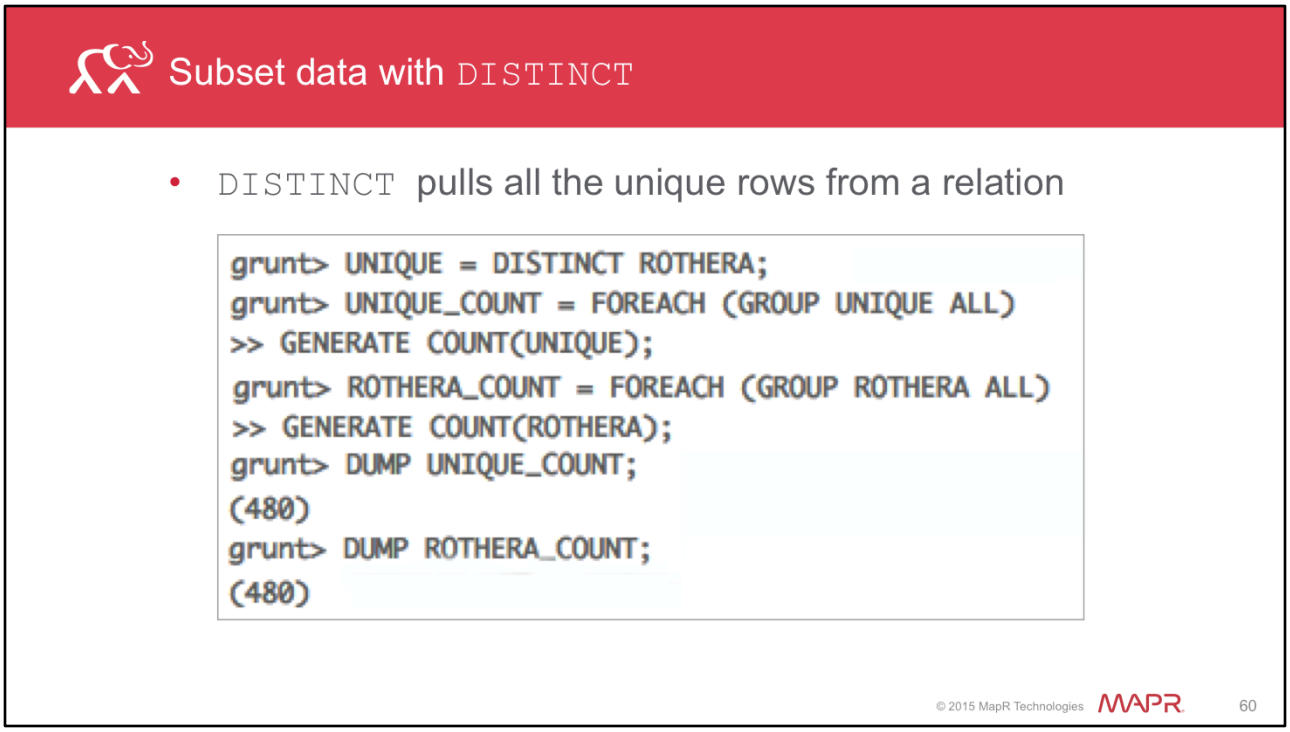
Welcome to DA 450 Lesson 3: Manipulate Data in Apache Pig. In this lesson, we will cover more advanced data manipulations in Apache Pig.



By the end of this lesson, you will be able to:
subset data with DISTINCT, FILTER, and SAMPLE
combine data with JOIN and UNION
manipulate data with ORDER, FLATTEN and UDFs

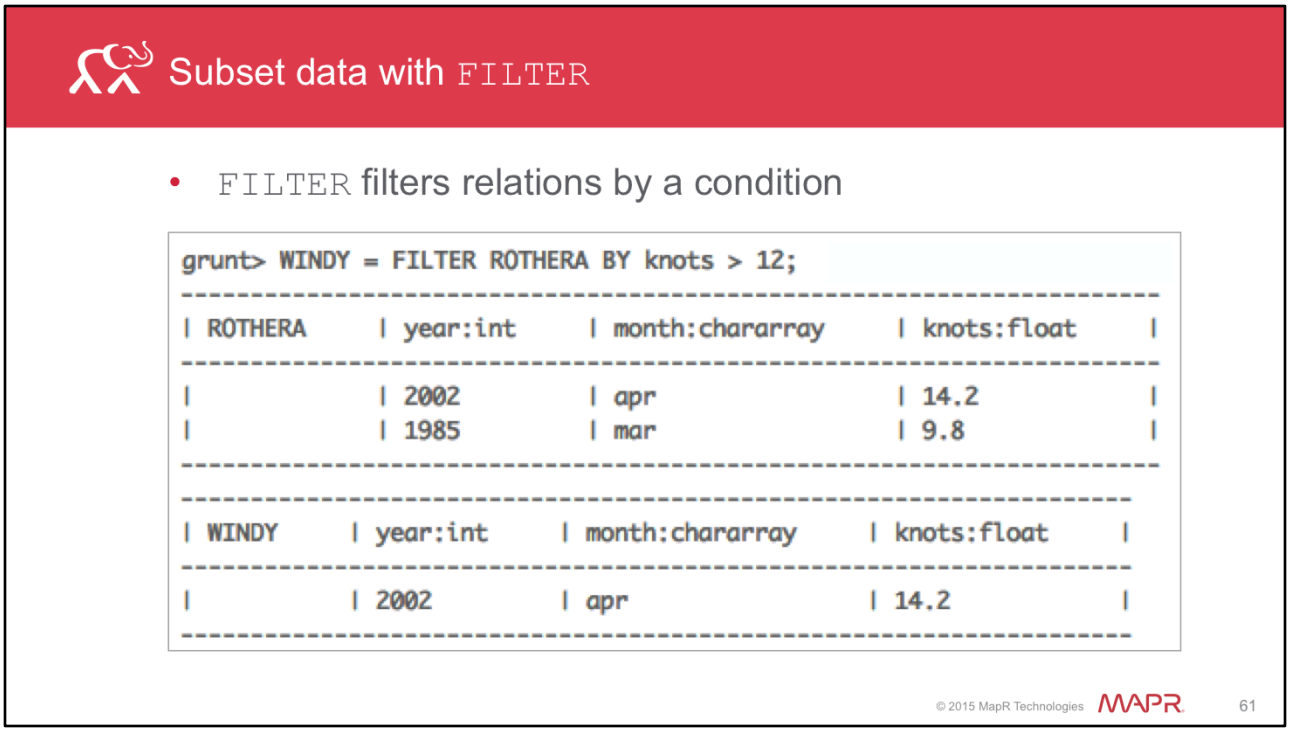


Let's start by subsetting data.



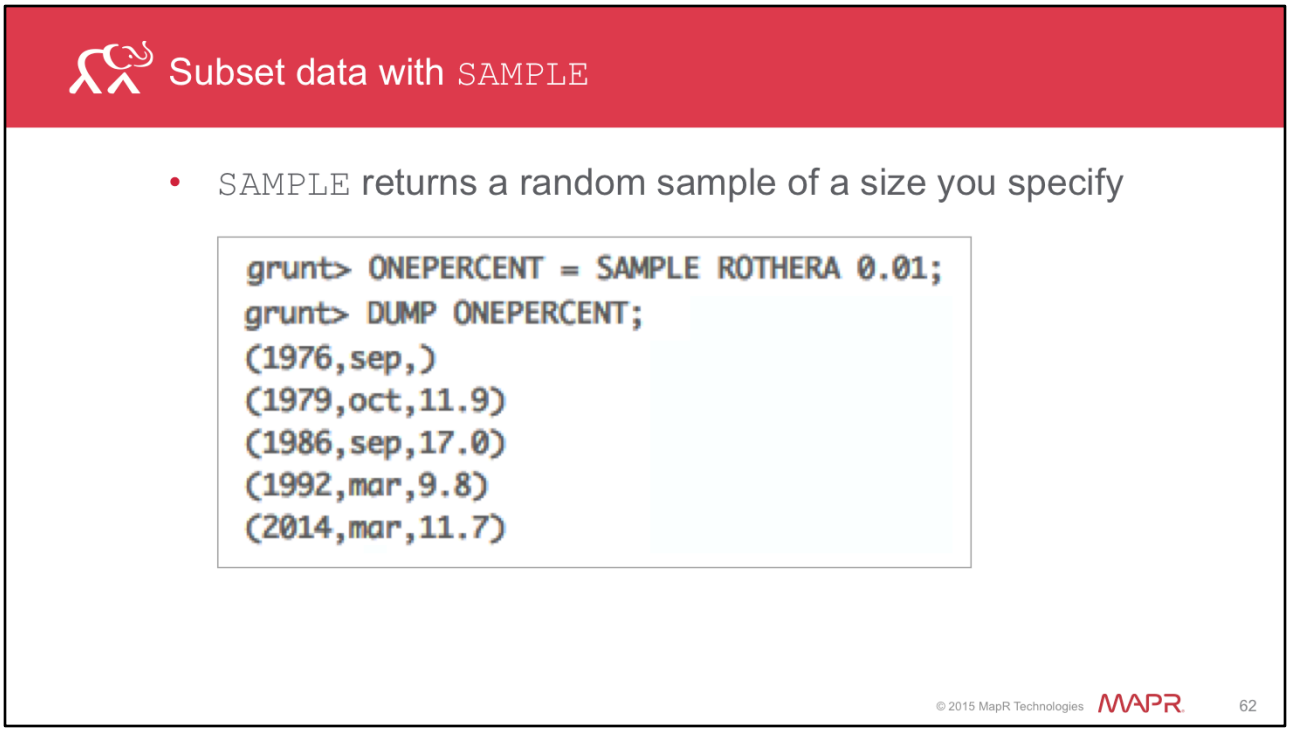
The DISTINCT keyword pulls all the unique rows from a relation DISTINCT is useful for eliminating duplicate data.

In this example, we make a new relation called UNIQUE which contains all the distinct data points in the ROTHERA relation we created in the previous lesson. We can then generate a count of all the rows in both the ROTHERA and the UNIQUE relations. We then DUMP these counts, and see that they both contain 480 rows. In this way, we are able to verify that there are no duplicate rows in our data.

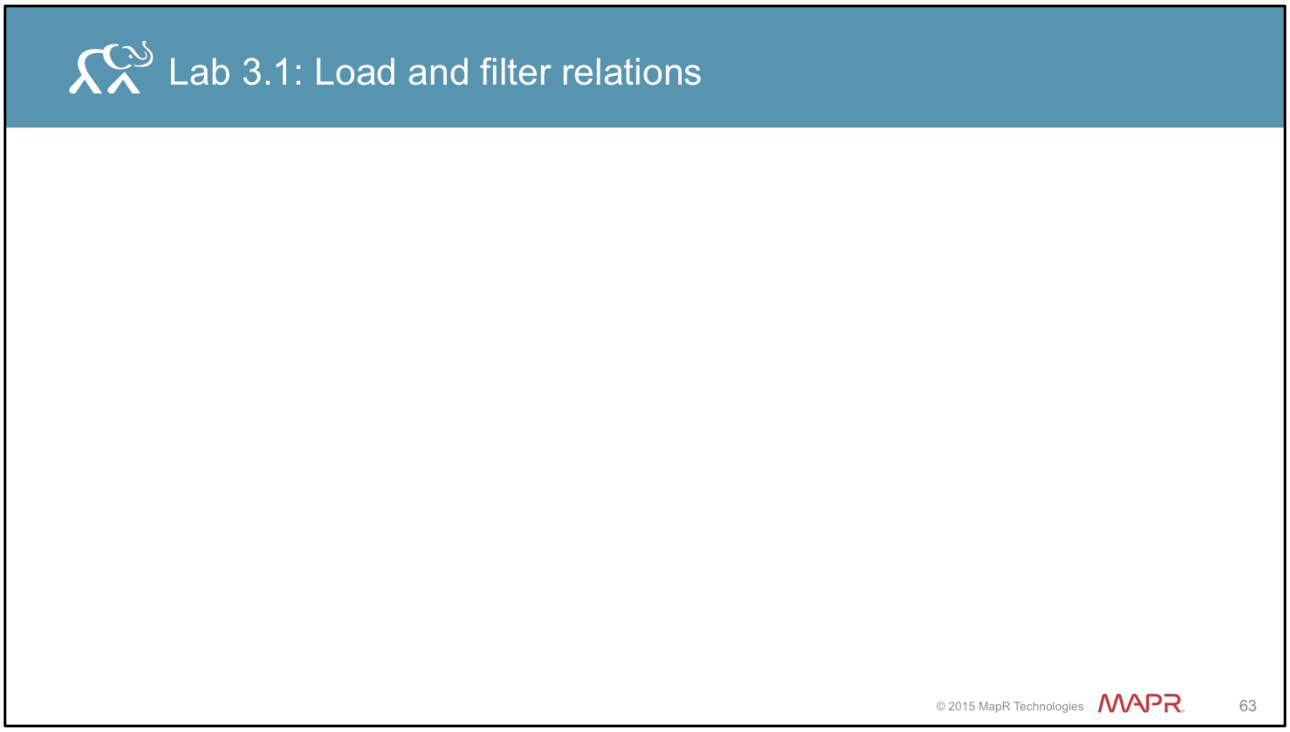


You can also subset by using FILTER. Here we create a new relation called "WINDY" which filters the ROTHERA relation by the knots column. The WINDY relation will only contain data where the knots measurement is greater than 12.

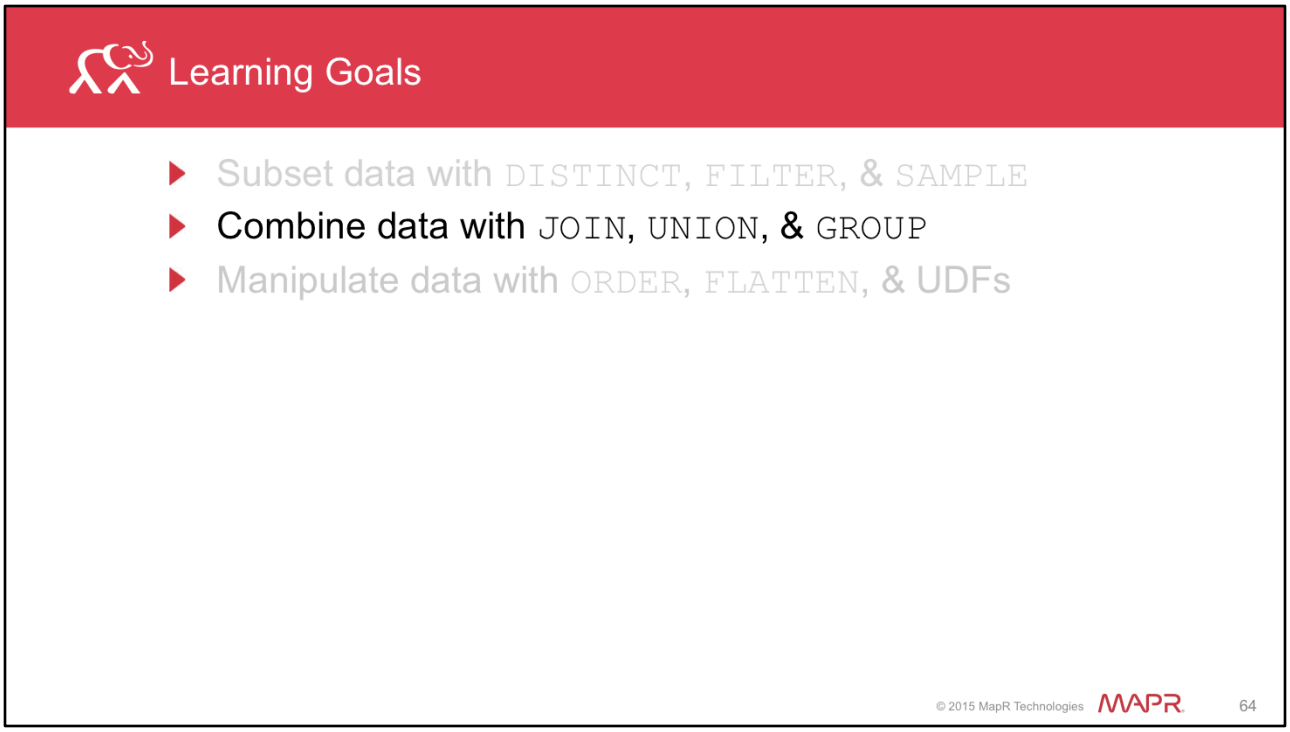
FILTER can use any type of boolean comparison, including string comparisons.



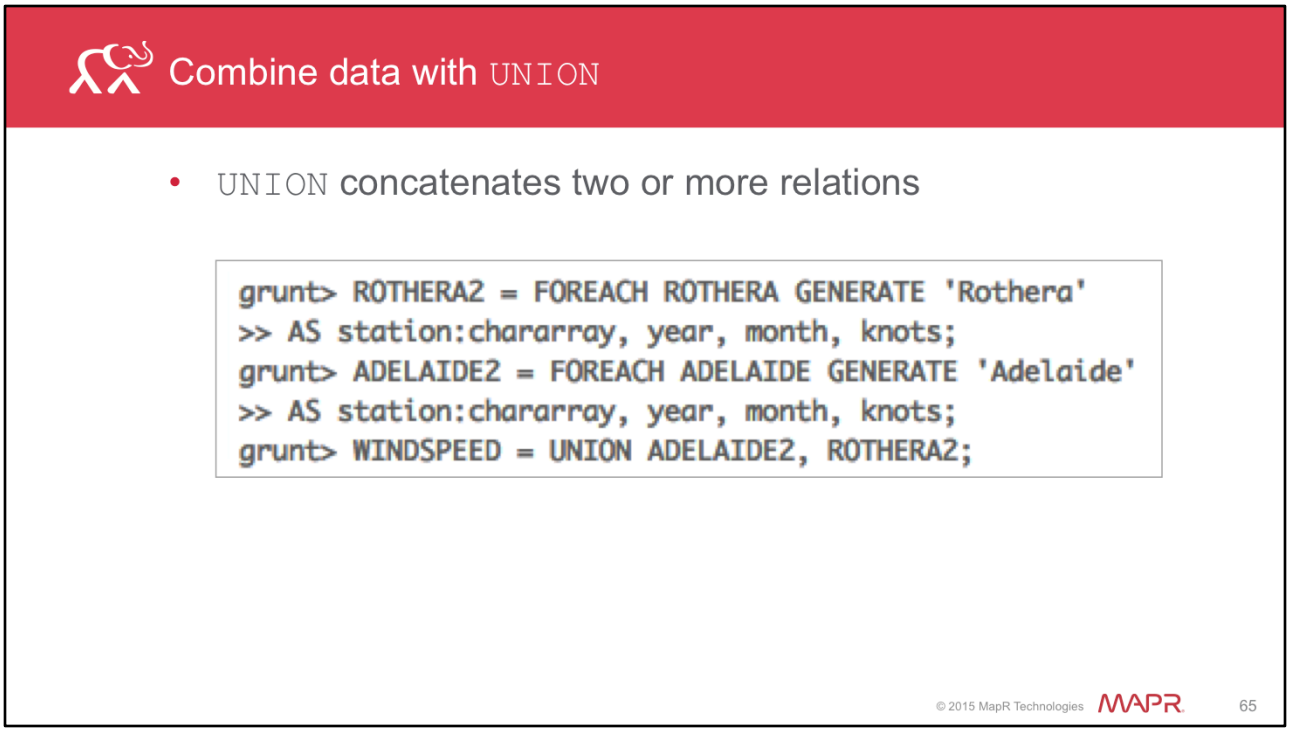
The SAMPLE keyword returns a random sample of a size you specify. In this example, we sample 1% of the ROTHERA relation. As we saw earlier, there are 480 rows of data in the ROTHERA relation, so the 1% sample returns 5 random data points.



In this lab, you will manipulate the data you began to process in the previous lesson's lab. Open your lab guide and complete Lab 3.1. When you are finished, you are ready to move on with the lecture.



In this next section, we'll learn how to combine data with Join, Union, and Group.



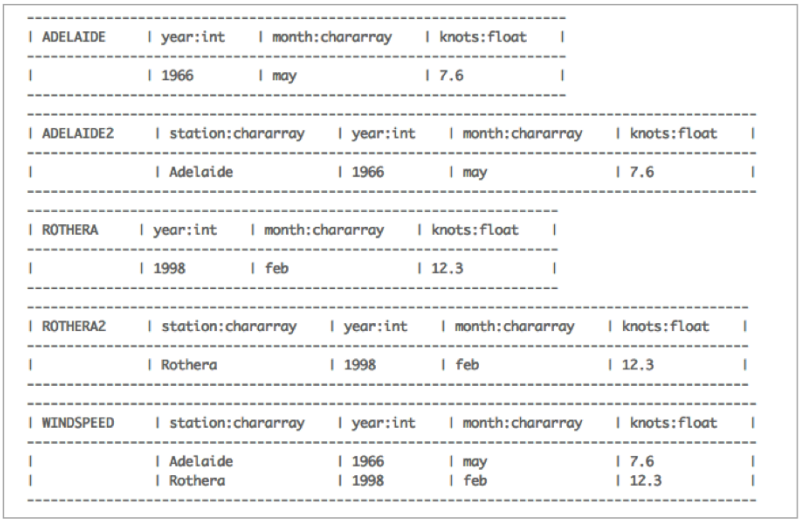
The UNION keyword concatenates two or more relations

In this example, we first add a station name column to the Rothera relation. We do the same to another relation, named ADELAIDE, which contains a similar dataset as ROTHERA but from a different weather station. This way, when we concatenate the two datasets, we know which rows came from which dataset.

Next, we concatenate these two datasets into a single relation called WINDSPEED using the UNION keyword.

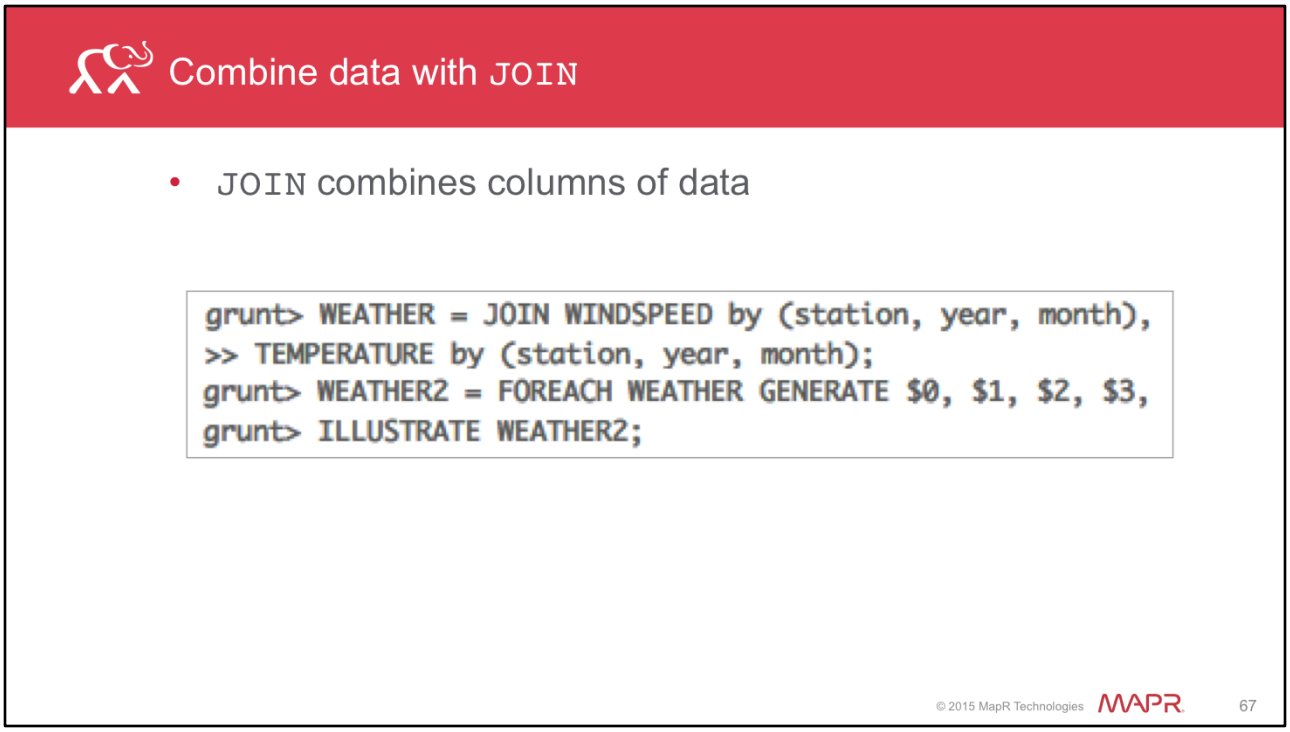
ILLUSTRATE on transformed relations

- ILLUSTRATE on a transformed relation shows all steps



```
-------------------------------------------------------------------
| ADELAIDE     | year:int    | month:chararray    | knots:float    |
-------------------------------------------------------------------
|              | 1966        | may                | 7.6            |
-------------------------------------------------------------------

-------------------------------------------------------------------------------------------
| ADELAIDE2    | station:chararray    | year:int    | month:chararray    | knots:float    |
-------------------------------------------------------------------------------------------
|              | Adelaide             | 1966        | may                | 7.6            |
-------------------------------------------------------------------------------------------

-------------------------------------------------------------------
| ROTHERA      | year:int    | month:chararray    | knots:float    |
-------------------------------------------------------------------
|              | 1998        | feb                | 12.3           |
-------------------------------------------------------------------

-------------------------------------------------------------------------------------------
| ROTHERA2     | station:chararray    | year:int    | month:chararray    | knots:float    |
-------------------------------------------------------------------------------------------
|              | Rothera              | 1998        | feb                | 12.3           |
-------------------------------------------------------------------------------------------

-------------------------------------------------------------------------------------------
| WINDSPEED    | station:chararray    | year:int    | month:chararray    | knots:float    |
-------------------------------------------------------------------------------------------
|              | Adelaide             | 1966        | may                | 7.6            |
|              | Rothera              | 1998        | feb                | 12.3           |
-------------------------------------------------------------------------------------------
```

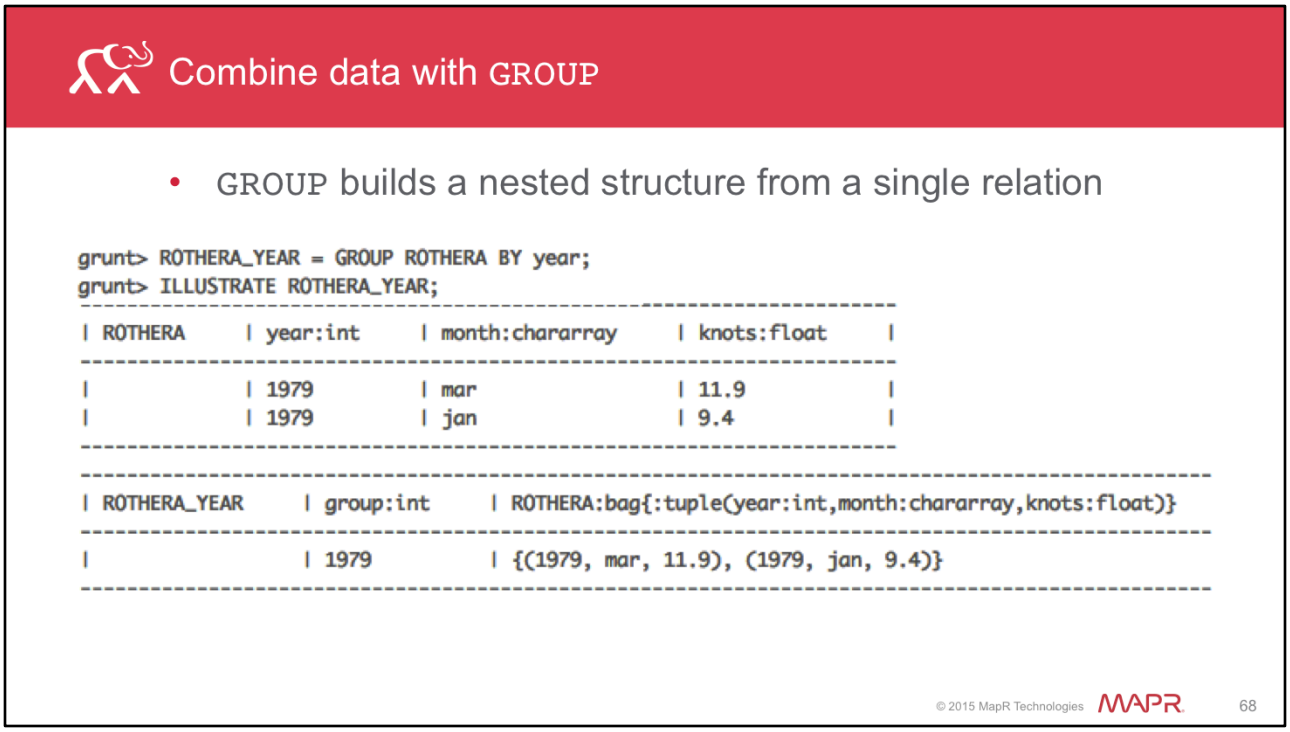

In this example, we check the new WINDSPEED relation with ILLUSTRATE. When we use ILLUSTRATE on manipulated relations like this, Pig displays each step of the transformation process. We can see the original data, the manipulated data, and the unioned data.



Like in SQL, the JOIN keyword combines columns of data from two or more different relations. However, the syntax for JOIN in Pig is somewhat different from that in SQL.

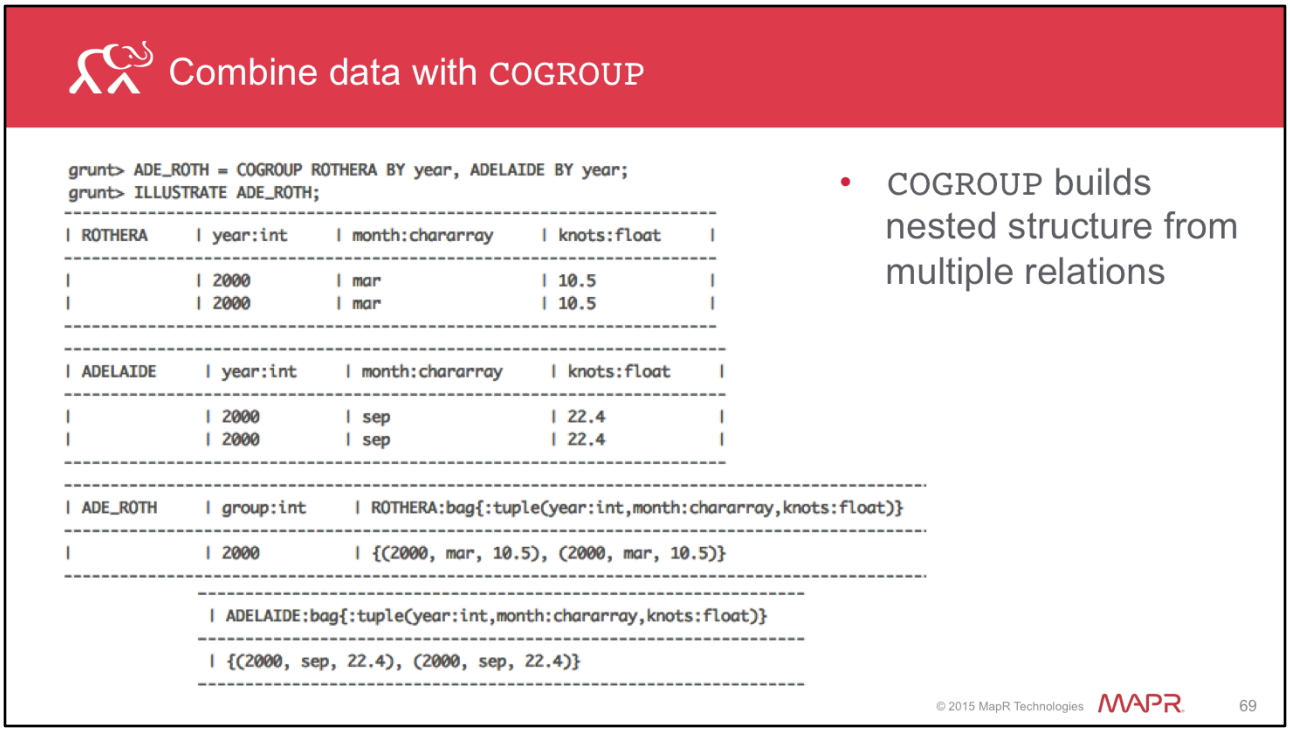
In this example, we join the WINDSPEED relation, which we created previously, with another relation called TEMPERATURE. We want the station names, as well as the dates, to match in each row. Implicitly, all data (celsius from TEMPERATURE, and knots from WINDSPEED) will be joined.

This type of join will join all the columns from both datasets, matching by station, year, and month. We can eliminate the extraneous columns by generating another relation, WEATHER2, with only the columns we want. We can verify this using ILLUSTRATE.

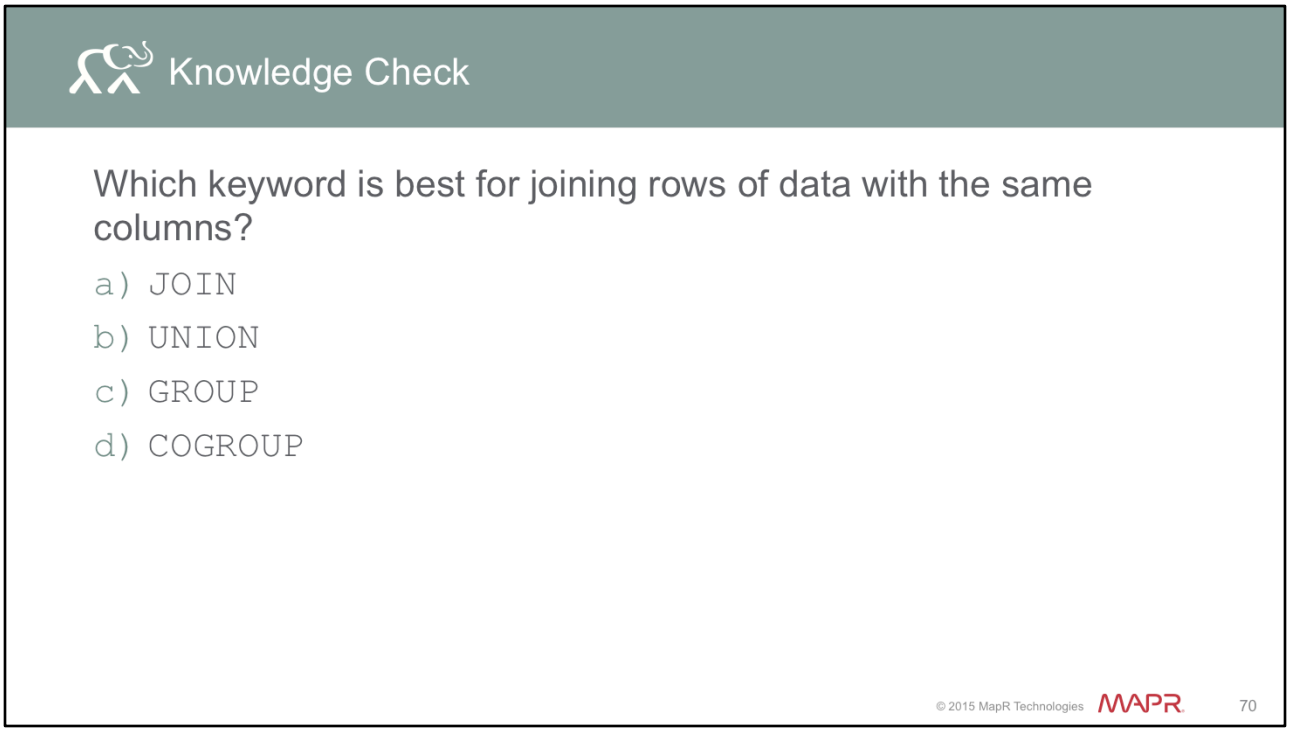


Another way to combine data is with GROUP.

In this example, we group the ROTHERA data by year. This creates a nested structure, with each bag containing data from each year.

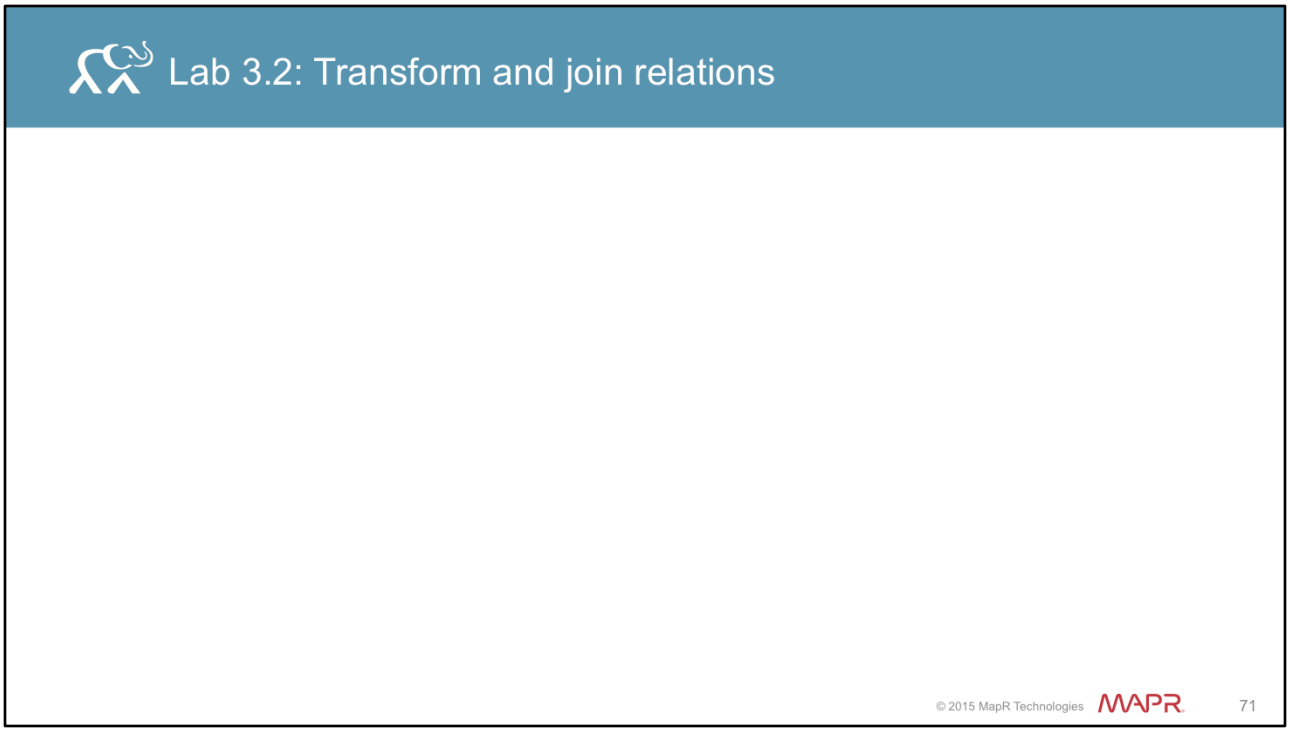


COGROUP also builds nested structures, like GROUP, but from two or more relations, as shown in this example.

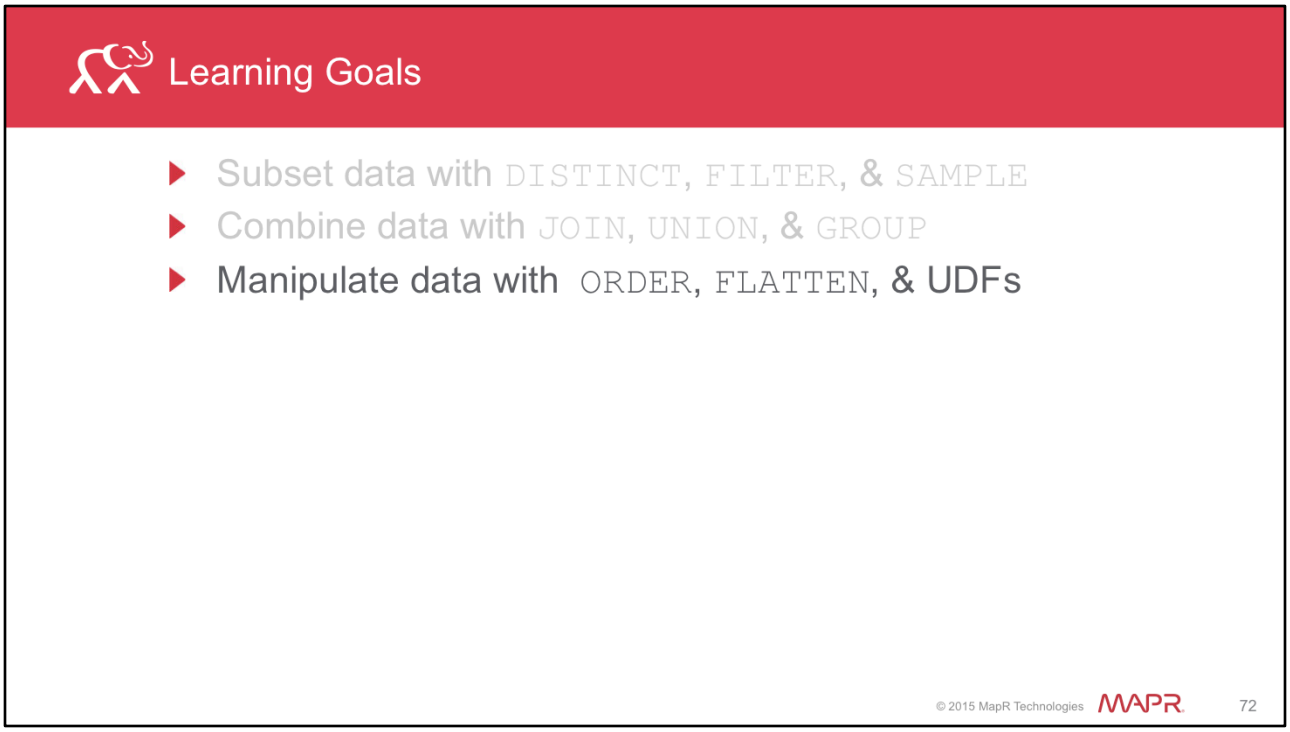


Answer: B

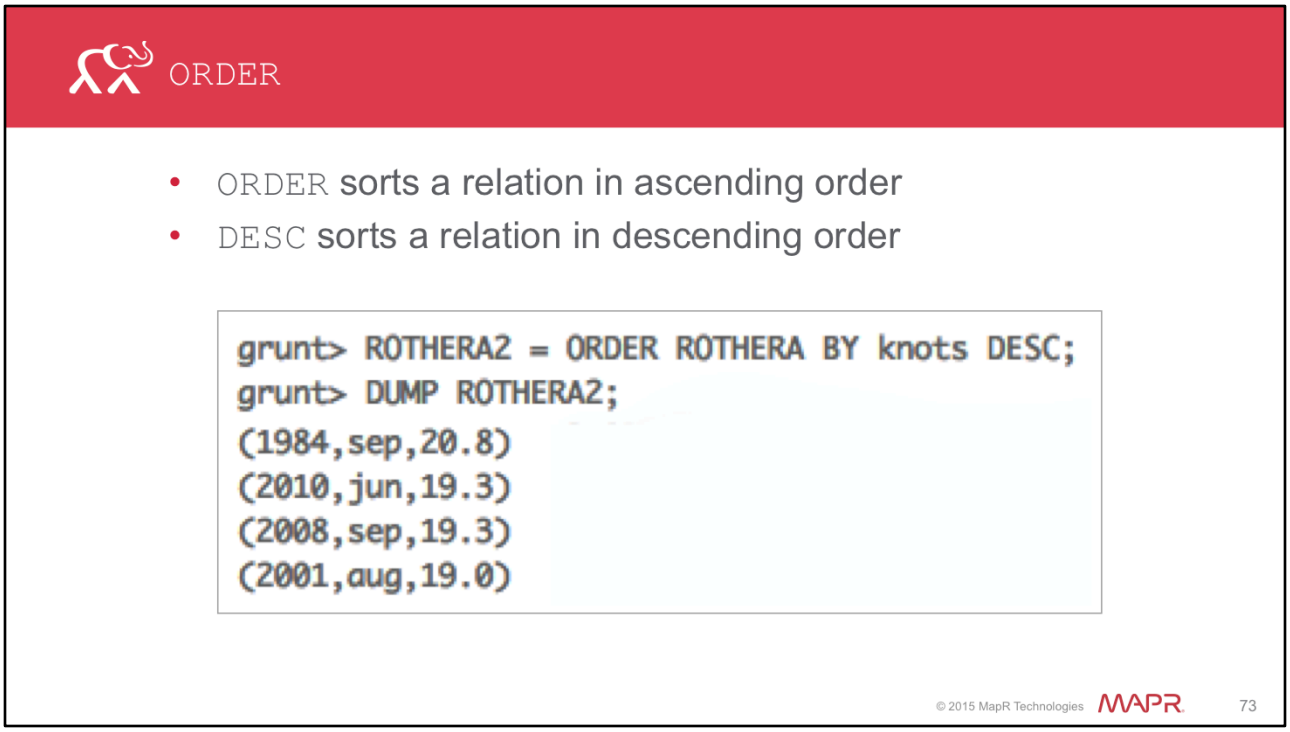
INSTRUCTORS: Take a moment to briefly discuss some different use cases of each of these keywords.



Continue with the lab by completing Lab 3.2.

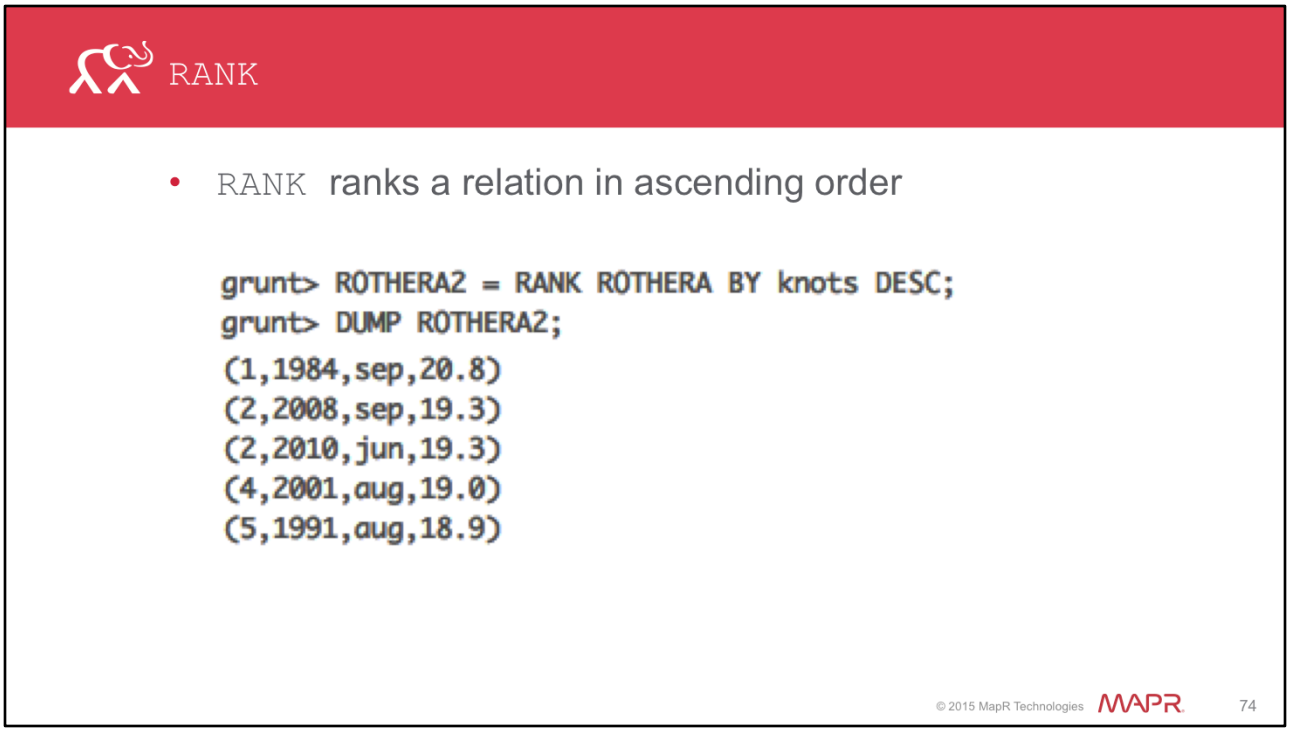


In this section, we'll learn how to manipulate datasets with ORDER and FLATTEN



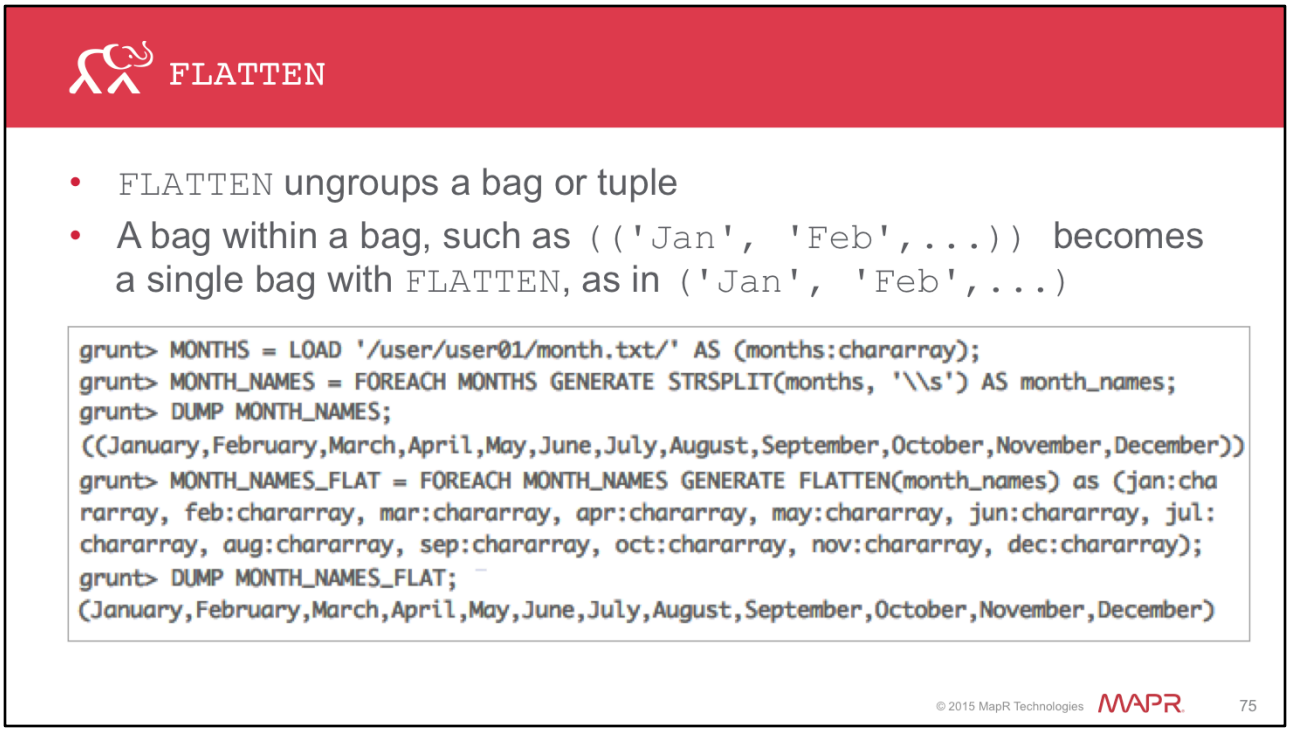
The ORDER keyword sorts a relation in ascending order. You can optionally add DESC to sort in descending order.

In this example, we want to see the entries with the fastest recorded windspeeds at the Rothera station, so we order ROTHERA by the knots column in descending order.
Remember, Pig will not actually sort your data until you request output, as with DUMP.
In this example, the first line does not trigger a MapReduce job, while the second one does.



The RANK keyword also sorts a relation in ascending order, or descending order with the DESC option

In this example, we again order the data in the ROTHERA relation. This time, Pig adds a ranking column as well. Note that in the case of a tie, both are ranked the same.



The FLATTEN keyword is used to ungroup a bag or tuple.

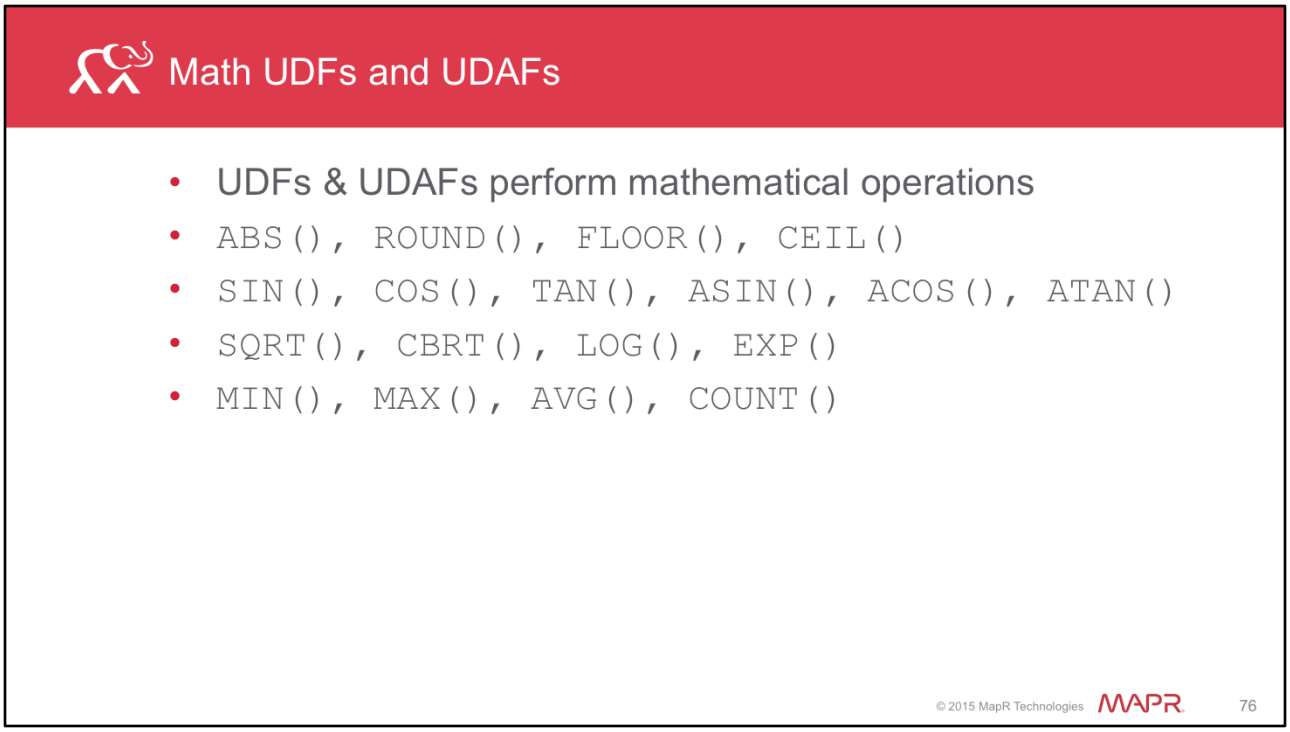
In this example, we have a document called "month.txt" which has the full name of each month. However, the data is saved as a single string. We load the data into a relation called MONTHS.

Then, we generate a tuple called month_names, inside a new relation also called MONTH_NAMES.

When we dump the MONTH_NAMES relation, we see all the names of the months listed inside a single tuple.

We want these to be 12 separate columns, rather than a single tuple. To ungroup this list of month names, we use flatten.

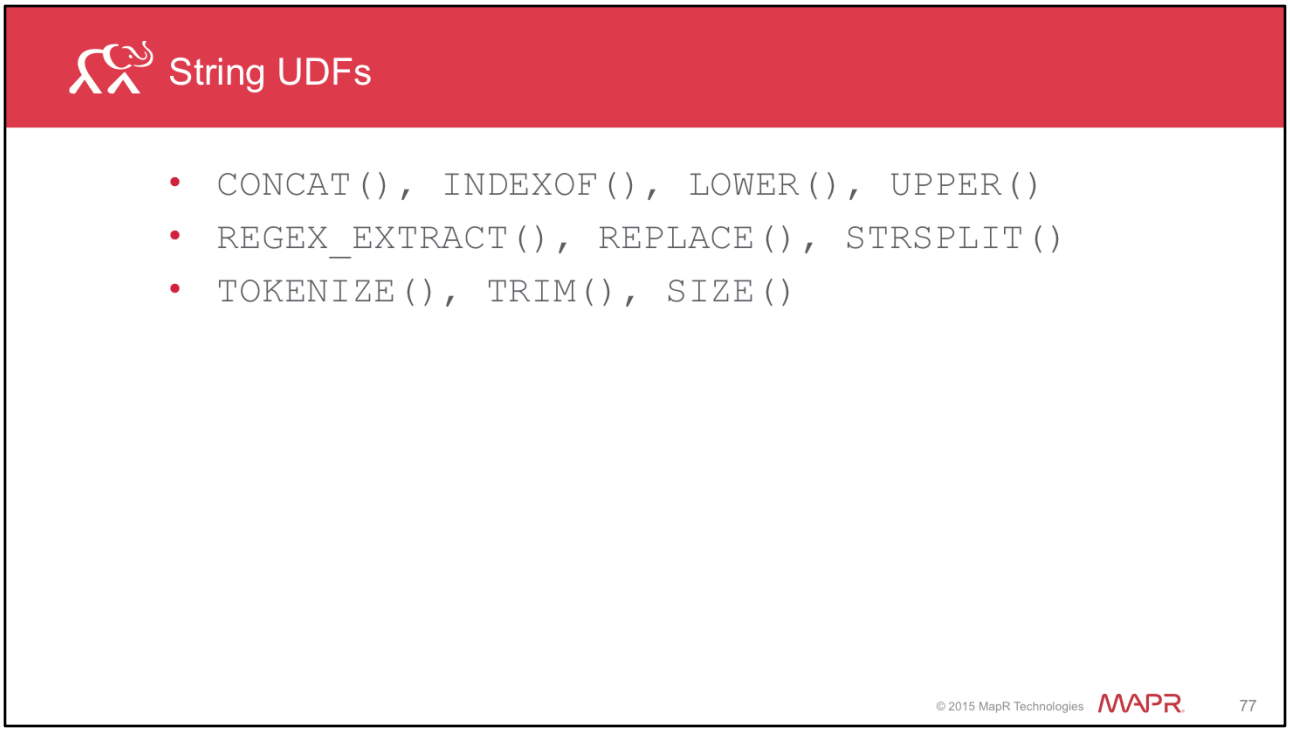
We create another new relation, MONTH_NAMES_FLAT, to



Pig comes with many common built-in user-defined functions, such as absolute value, sine, cosine, tangent, rounding, averages, square roots, and many others.

Here is just a sample of some of the math UDFs and UDAFs available. Consult the Apache Pig documentation for more details on how to use each of these functions.

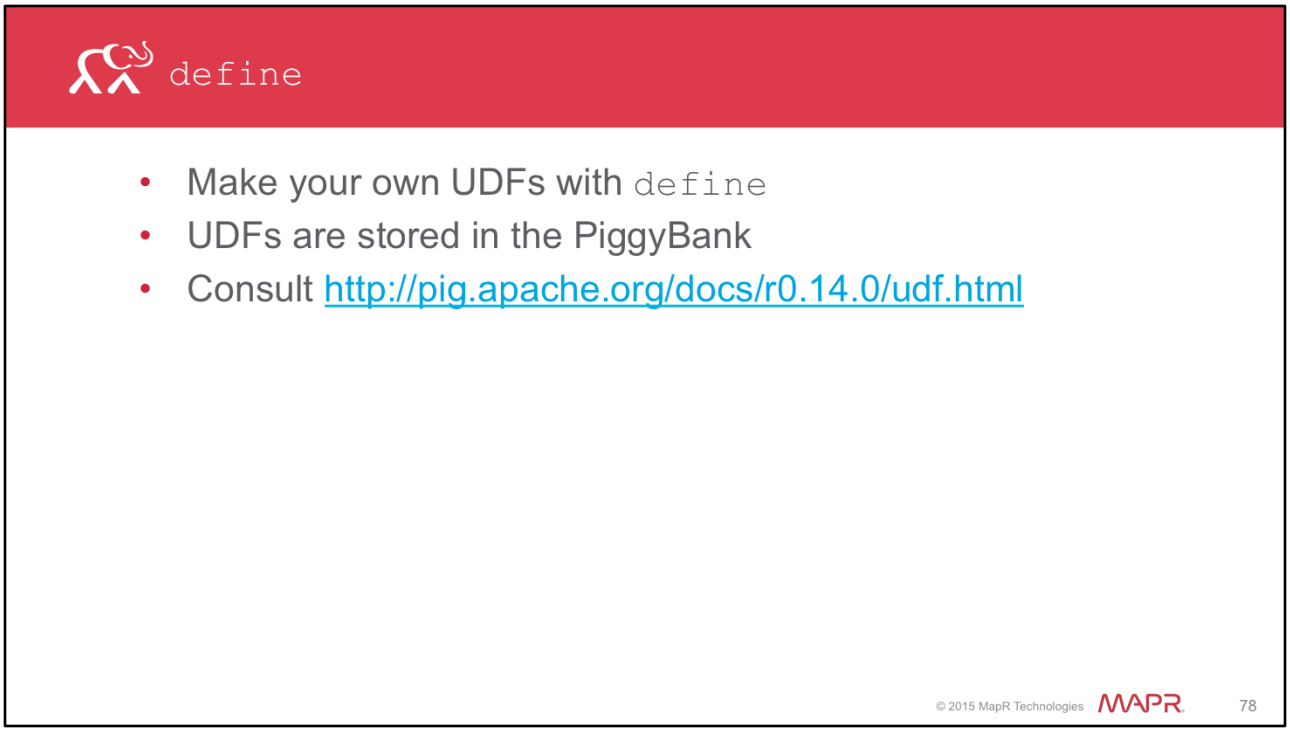
INSTRUCTORS: Take some time to discuss a few of these with your students. Which are they likely to use in their own work? You may wish to demo a few of them.



In addition to mathematical operations, there are many built-in string functions available

Here is just a sample of some of the string UDFs available. Consult your documentation for more details.

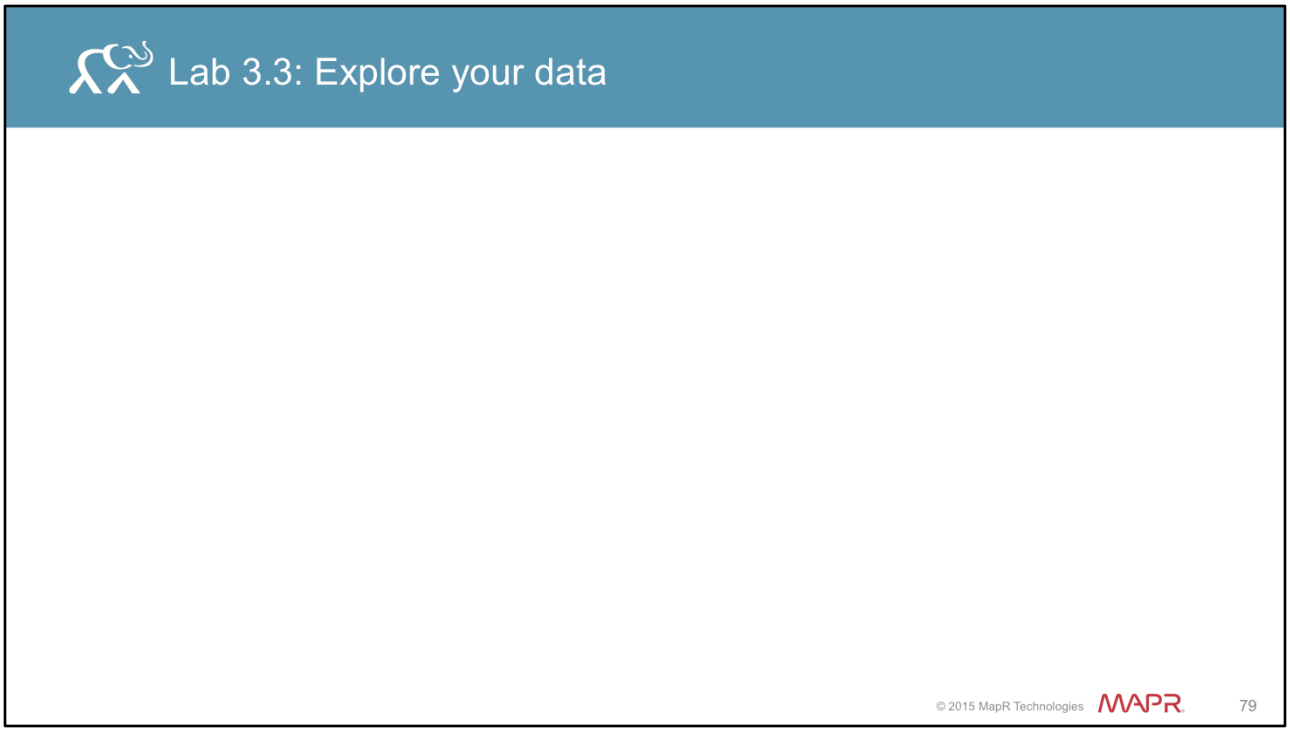
INSTRUCTORS: Take some time to discuss a few of these with your students. Which are they likely to use in their own work? You may wish to demo a few of them.



Pig allows users to create their own UDFs, written in Pig Latin, Python, Java, or other languages.

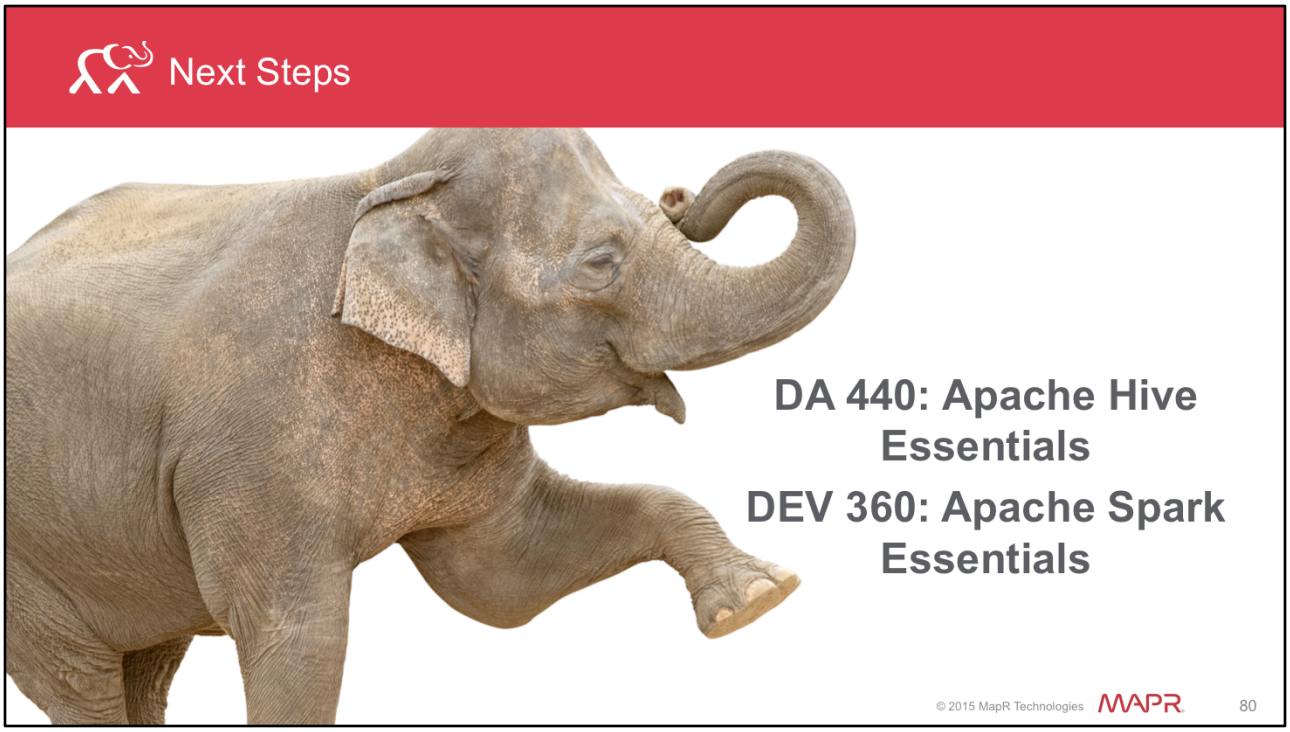
These UDFs are then stored in the PiggyBank, along with pre-installed UDFs.

There are many ways to write and store UDFs. Consult the Apache Pig documentation for more details



Take some time to ask your instructor any final questions you may have about Pig or Pig Latin to ensure you understand the material.

Now that you know a little bit more about data manipulations, return to the "TRY THIS" at the end of the Lesson 2 lab. If you can complete this challenge, try the baseball statistics challenge at the end of of the Lesson 3 lab.



Congratulations! You have now completed DA450: Pig Essentials. Visit training.mapr.com for more courses, including DEV360: Spark Essentials or DA440: Hive Essentials.

DA 440 Hive Essentials will continue the data pipeline you started in this course. In DA 450, you will load the weather data you manipulated in DA 440, and query it using the Hive Query Language.