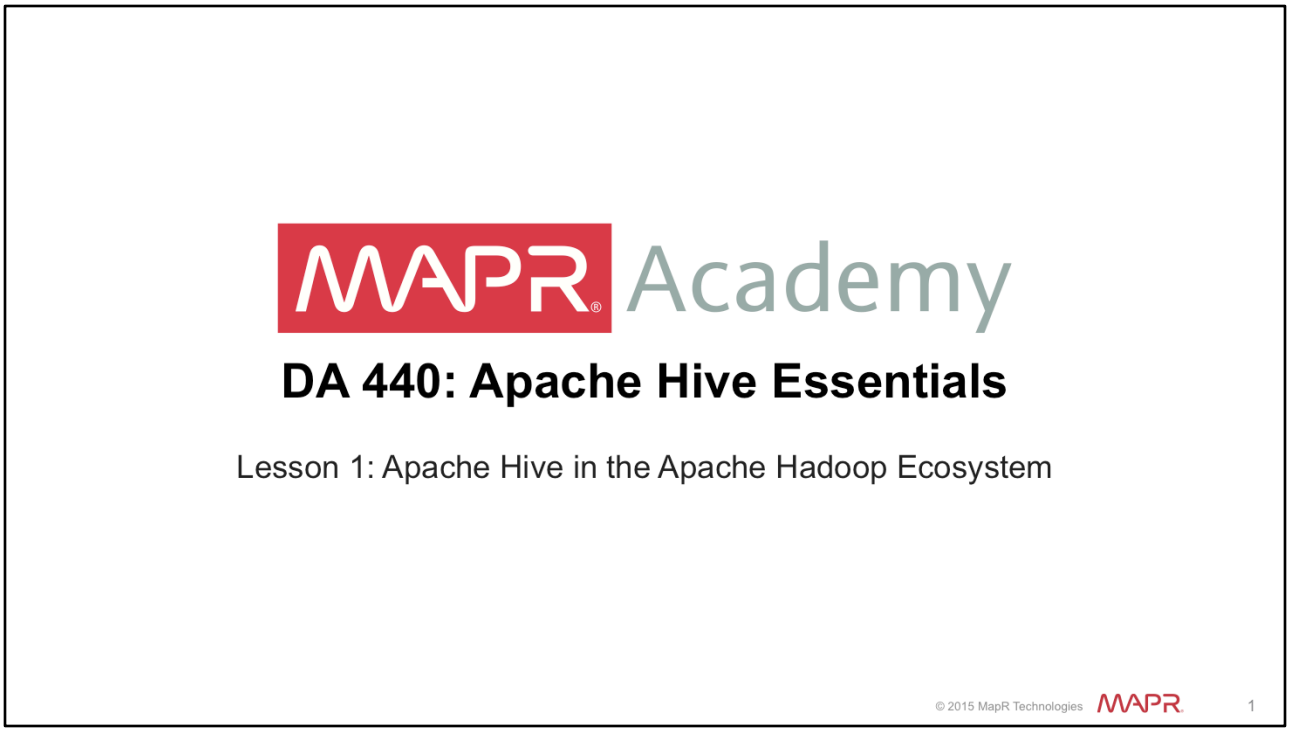
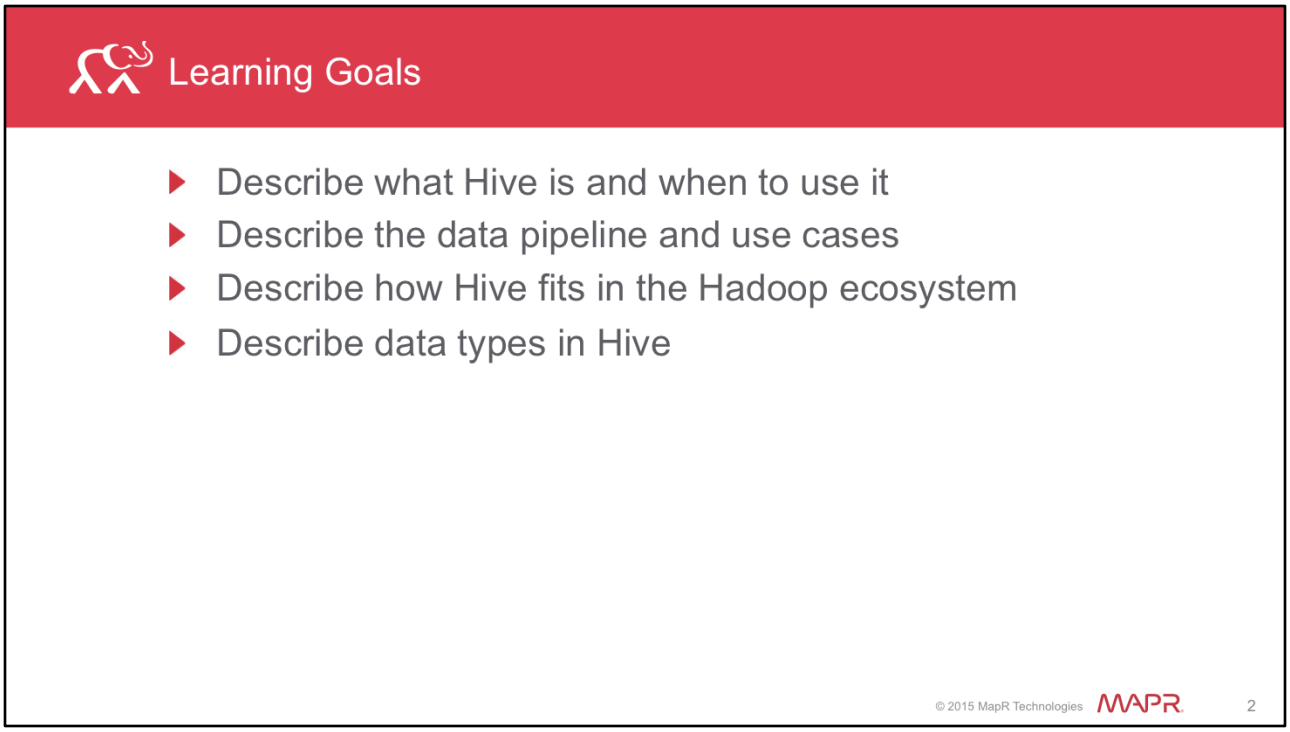
DA 440 – Apache Hive Essentials Slide Guide

Fall 2015 – Version 5.0.0



Welcome to DA 440: Apache Hive Essentials – Lesson 1: Apache Hive in the Hadoop Ecosystem. In this lesson, you will learn what Apache Hive is and how it fits into the Hadoop ecosystem.

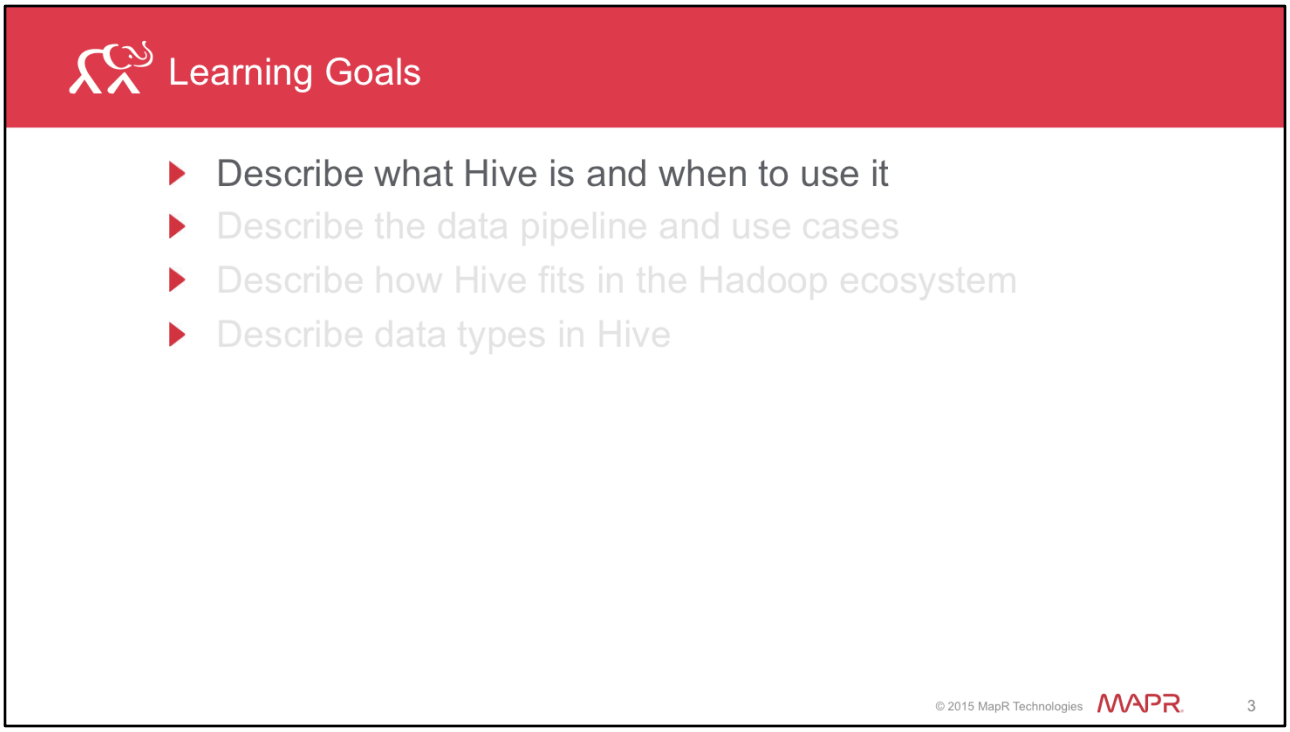


When you have finished this lesson, you will be describe what Hive is as well as when to use it.

You will also be able to describe the data pipeline, how Hive fits into the data pipeline, and what the pros and cons of using Hive are.

You will learn how Hive fits into the broader Hadoop ecosystem, including how Hive uses MapReduce, how Hive interfaces with other Hadoop tools, and some of the differences between Hive versions.

Finally, we will discuss data types and casting in Hive.



Let's start with a brief overview of Hive, as well as other SQL-on-Hadoop tools.

Too Much Data?

You work in a large climate research center. You

- already know SQL
- have accumulated petabytes of weather measurements
- migrated data to HDFS

How do you process this data?

 MAPR 4

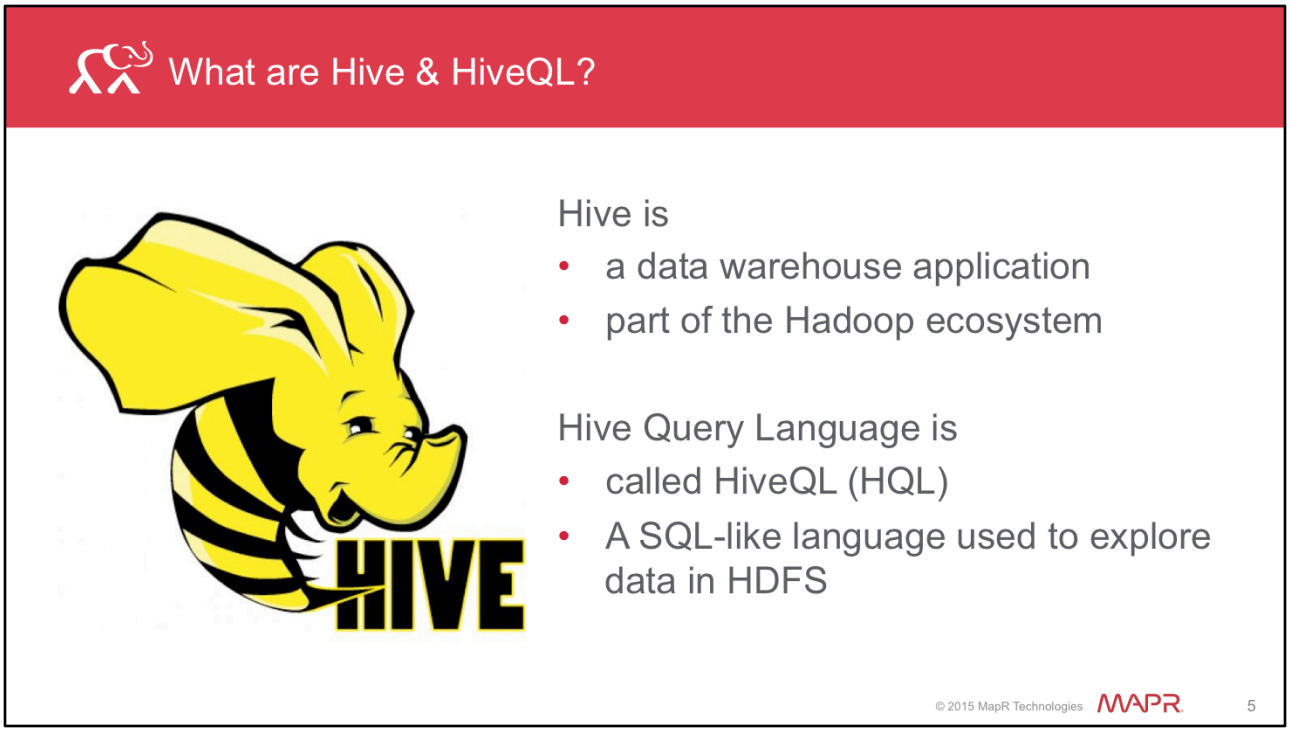
We are working at an international Climate Research Institute, and you are a data scientist. This research center manages thousands of weather stations around the world. Each weather station provides a number of data points: temperature, wind speed, or a number of other measurements. This data is sent in aggregated batches every month.

This data is organized in tables. Each row contains columns representing the station name, the date the measurements were taken, the temperature, and the wind speed.

You, as a data scientist at the Climate Research Institute who is already familiar with SQL, might want to query this data with SQL.

However, the research institute has accumulated hundreds of petabytes of data. These massive tables are too large for a traditional Relational Database Management System to handle. The Climate Research Institute has decided to migrate all its data to a Hadoop Distributed File System. The HDFS can store, access, and backup petabytes of data easily.

However, traditional SQL does not work on the HDFS. What are your options as a data scientist? One option is Apache Hive.



What is Hive? Apache Hive was initially developed at Facebook for the purpose of summarizing, querying, and analyzing large amounts of data stored on a Hadoop Distributed File System.

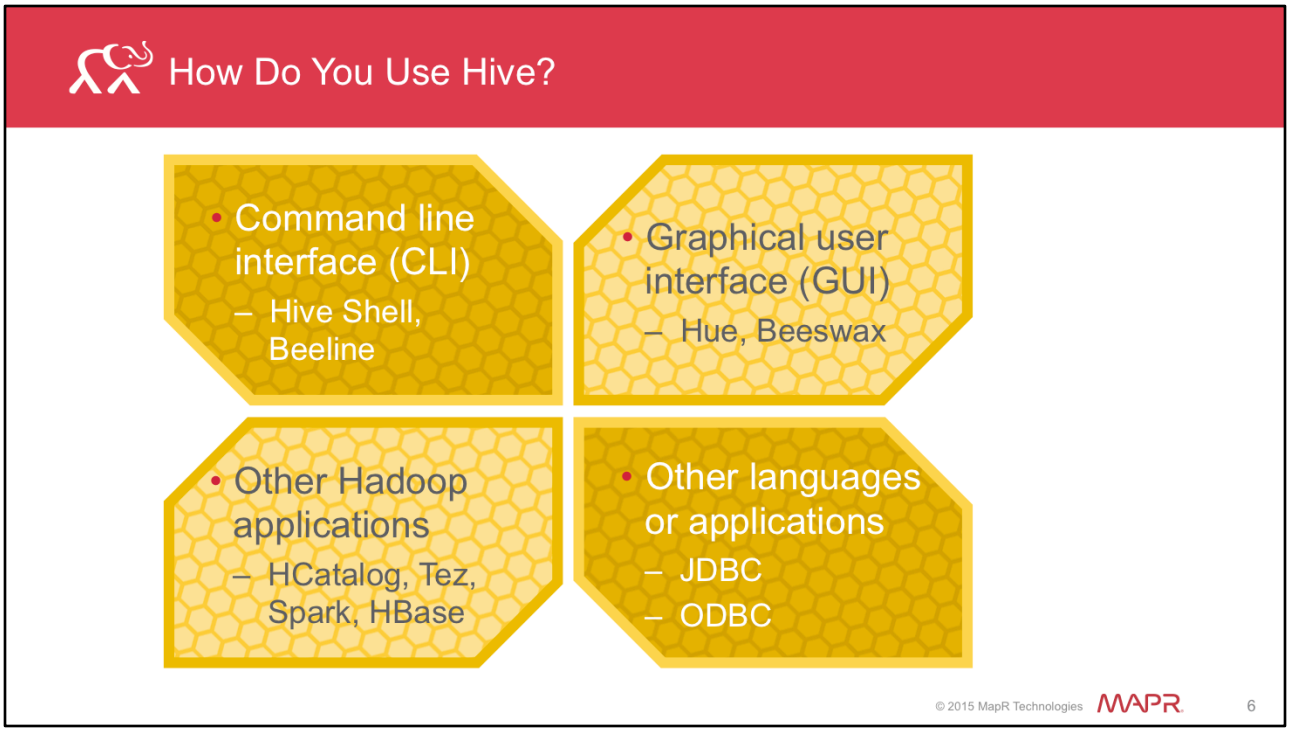
Hive is often used to analyze large databases, such as customer data for online retailers, social media user accounts, or research data like that used at our Climate Research Institute.

Hive is a data warehouse application built on top of Hadoop. Hive can interact with many parts of the Hadoop ecosystem using a query language called HiveQL.

HQL is an SQL-like language that can query data in an HDFS. Hive combines the power of lower level MapReduce code with the ease of a higher level language like SQL.

Since HQL is similar to SQL, it's easy for analysts who are already familiar with SQL to learn HQL. However, keep in mind that HQL is not ANSI-SQL, and this course will discuss some of the differences between HQL and SQL.

These are some of the reasons why Hive is ideal for a situation where your data scientists already know SQL, but your database has become too large for traditional relational database management systems.



Now that we have decided Hive is the right tool for us, how can we use Hive?

Early versions of Hive, this command line interface is sometimes referred to as the Hive Shell. However, in version 0.11 a newer command line interface, called Beeline, was introduced. Depending on the version of Hive, HiveServer, and Hadoop you are running, the CLI may be called the Hive Shell or Beeline. There are only minor differences between the two; consult the Hive documentation for more details about these.

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

Hive can also be used in conjunction with other Hadoop applications such as Hcatelog, Tez, Spark, and Hbase. Hive is often used in conjunction with Pig as part of an Extract, Transform, Load data pipeline.

Finally, Hive can be used with other programming languages and other applications outside of the Hadoop ecosystem. JDBC allows users to use languages like Java, Python, and Ruby with Hive. ODBC allows users to connect Hive to applications like Microsoft Excel or Tableau.

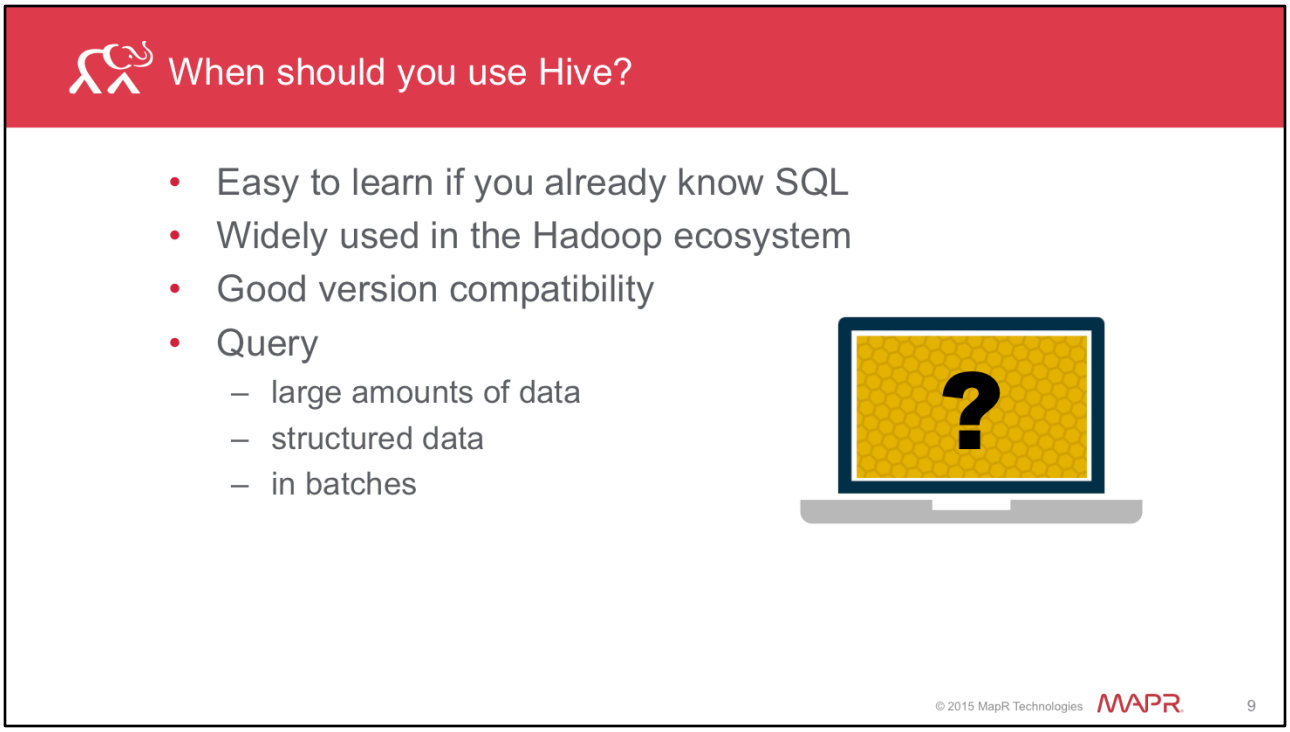
Here we see the same query, SELECT * FROM rothera; entered into the Hive Command Line Interface, the Beeline Shell, and the HUE Graphical User Interface.

As you can see, the syntax of the queries themselves is the same in all three interfaces. There may be slight differences in how each interface handles errors or accesses the file system as well. For the most part, however, the Hive QL you would use is the same regardless of which interface you choose.

INSTRUCTOR: Now is a good time to demonstrate each of these three interfaces. You should have them open and running off-screen, with some sort of data already loaded, so you can easily access them at this point of the lecture.

You should have already installed and connected to the MapR Sandbox during Lesson 0. Try connecting to the Hive CLI, which is how we will primarily be using Hive in this course. You can also try connecting to the Beeline Shell.

Once you have connected to the Hive CLI, you can continue with lesson 1.



Let's review: How do you know if Hive is the right tool for you?

First, it's easy to learn if you already know SQL. Since many data scientists and business analysts are already familiar with SQL, Hive is a great introduction to Hadoop, allowing users to run MapReduce jobs without learning Java code. Hive is also learned by many developers and administrators who work with data scientists.

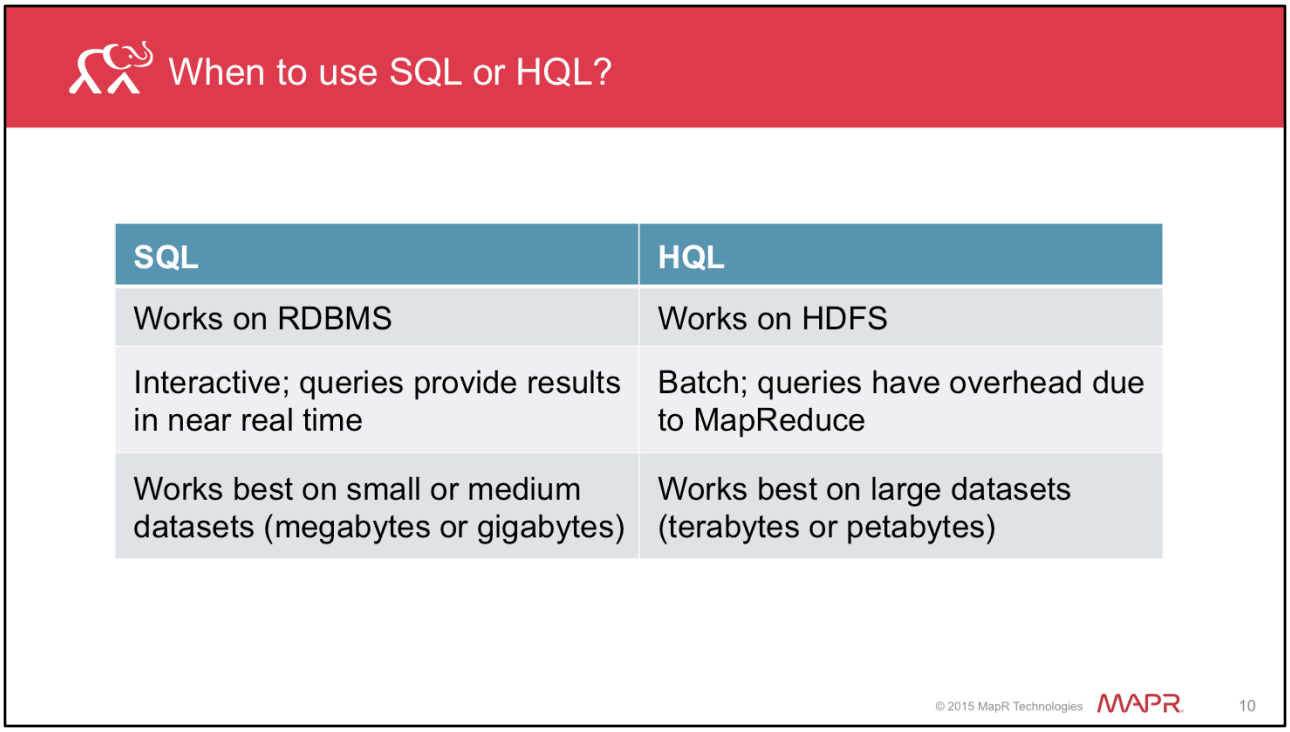
Another benefit of Hive is that it is widely used in the Hadoop ecosystem. There is a large support network available for learners of Hive: many books, forums, and other groups can provide support as you dive deeper into Hive.

Next, Hive has great compatibility with other tools in the Hadoop ecosystem. Hive has been around long enough that it has been tested thoroughly. Hive works, and each new version has fewer bugs and compatibility issues. While newer tools may have additional features, Hive has stood the test of time, much like SQL has.

Unlike SQL, however, Hive is built to handle very large data sets. While SQL may fail on a large dataset, Hive will reliably complete.

Hive is also good for structured data, like tables. Unstructured data, such as JSON, may require some manipulation before it is usable in Hive. If your data is structured, or it can become structured through an ETL process, then Hive might be a good tool for you.

Finally, although you can do exploratory analysis in Hive, it is commonly used for batch queries. If you already know what you want to query, and can write batch scripts for your dataset, Hive might be a good choice for you.



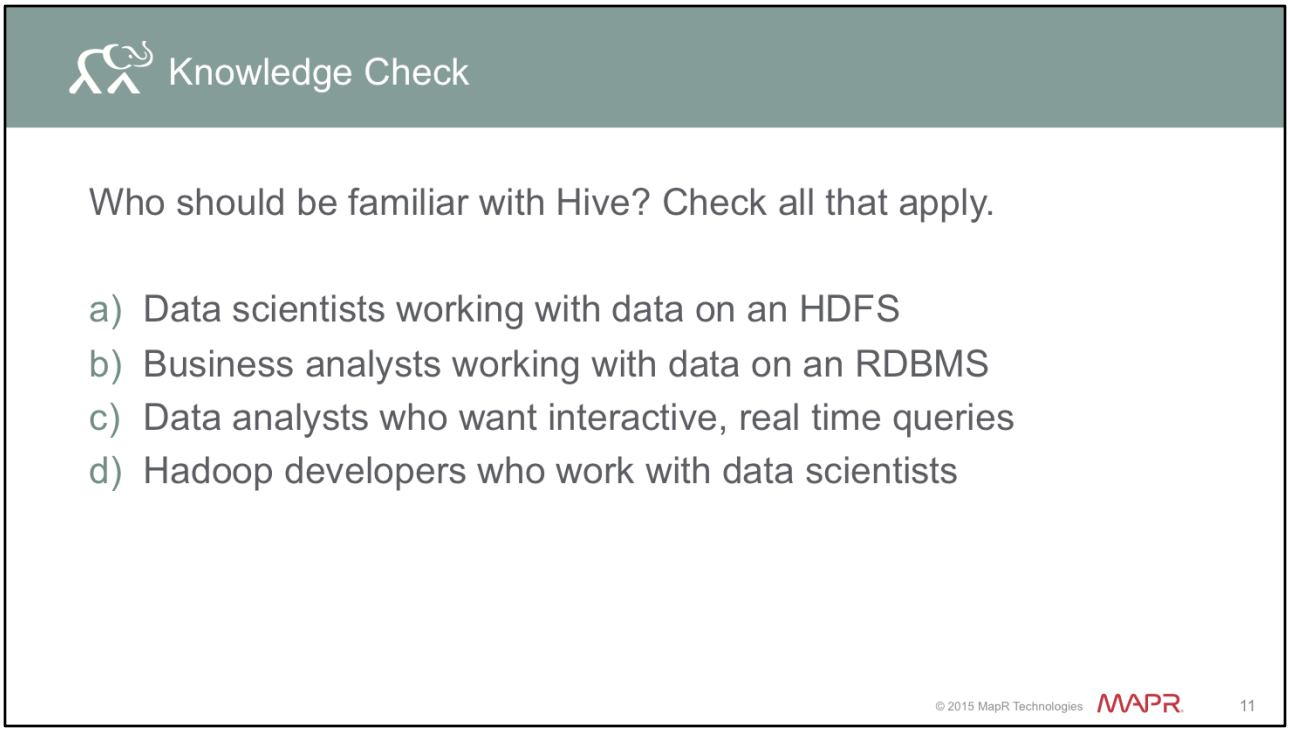
This chart summarizes some common use cases for SQL or Hive.

The type of database you are working on, whether a relational database or a Hadoop database, is an important factor when deciding when to use Hive or SQL.

You should be aware that SQL is interactive, while Hive is batch. SQL queries produce results in real-time, while complex Hive queries may take several hours or even days to run.

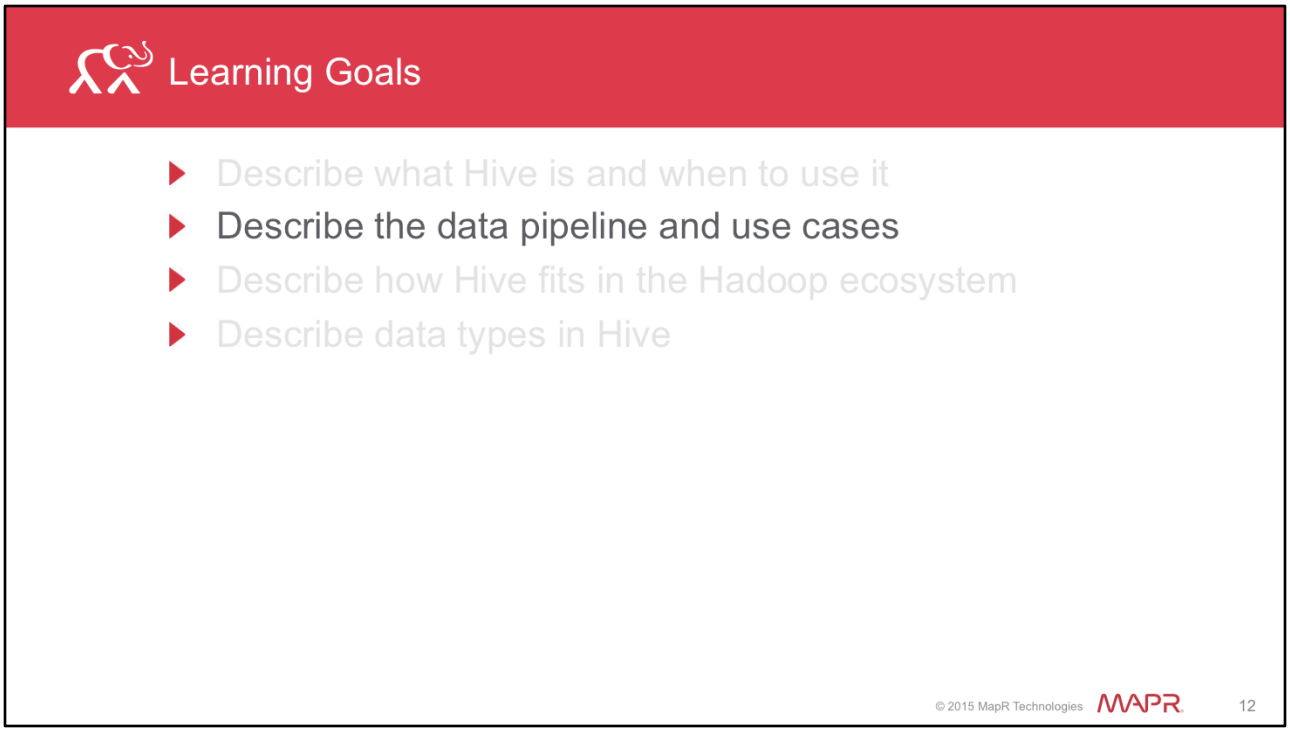
Finally, the size of the database is also important: SQL works best on small or medium datasets, while Hive is optimized for very large datasets.

In the case of our weather data set, our Climate Research Institute is facing an ever-growing database which we recently moved to an HDFS. We are more concerned with trends over time rather than real-time updates. These long term trends can be queried in batches. Therefore, we will choose Hive over SQL.

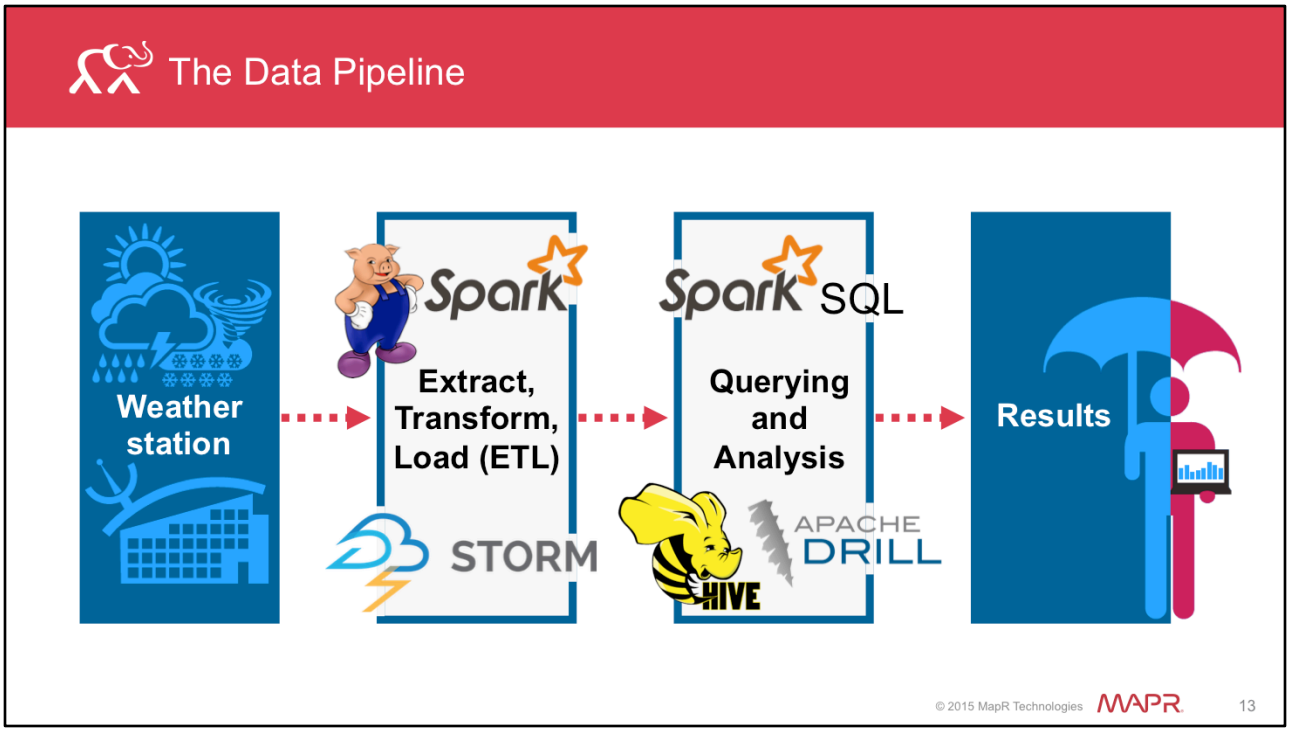


Answers: A and D

INSTRUCTOR: Take some time to have your students introduce themselves. What role(s) do they see themselves preforming with Hive? Are they admins or developers who support data scientists? Are they data scientists or business analysts interested in big data? Are they curious lifelong learners who are taking the course to expand their knowledge of big data technologies?



In this next section, we'll look at other tools in the data pipeline, and where Hive fits in.



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

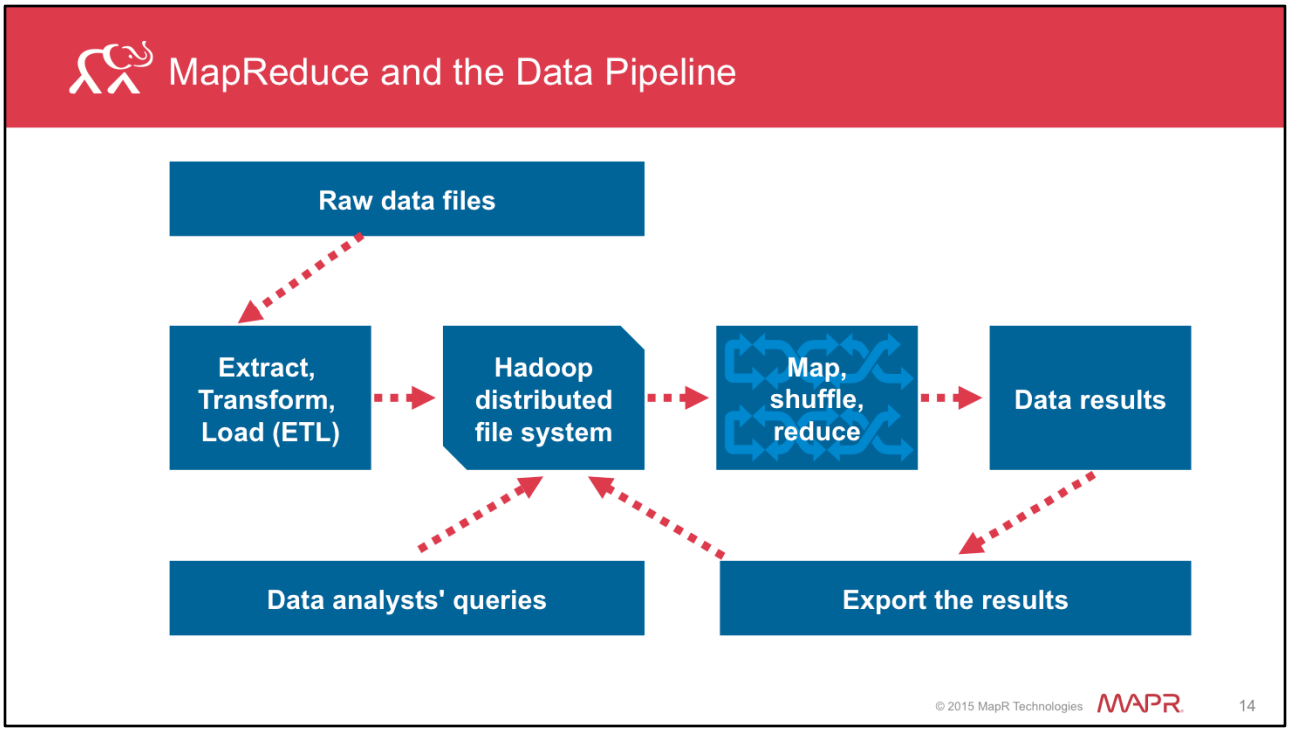
The source of raw data will vary from one project to the next. One example would be the Internet of Things. The Internet of Things refers to networked sensors, such as those found in smart cars or weather stations, that automatically send their data to some central location, such as the Climate Research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

While some parts of the early ETL process can be performed by Hive, other tools such as Pig are better suited for this step of the data pipeline.

Once the data is loaded, a data scientist can query and analyze the transformed data. For example, you might want to find out the average temperature in August at weather station across our entire dataset; this is the type of query you can easily preform in Hive. Depending on the nature of your data and your project, there are again a number of tools to choose from, including other SQL-on-Hadoop tools like Drill or SparkSQL.

Finally, you can export the results of your query to use in a report, which you might create with other programs to generate graphs, write up your analysis, or make slide decks for presentations.



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.

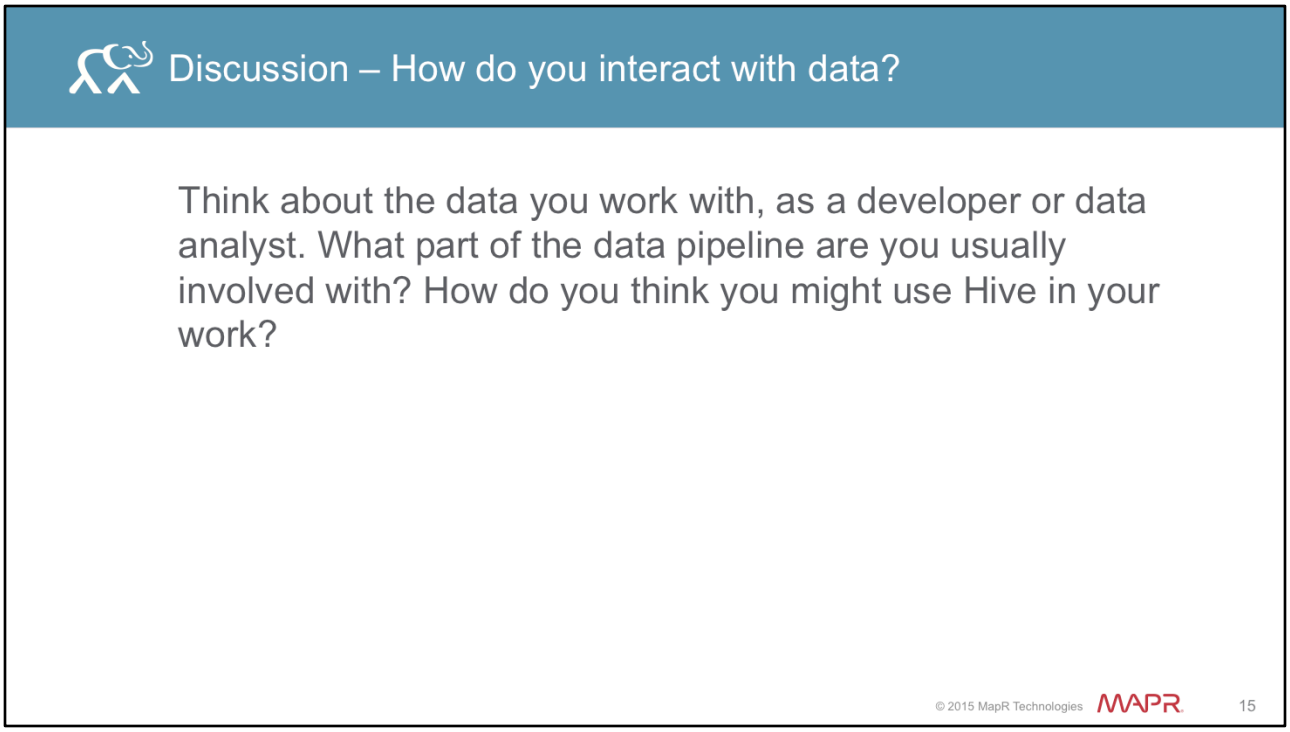
In most cases, a raw data file must be extracted, transformed, and loaded into a Hadoop Distributed File System. This includes any data normalization processes you must perform, such as splitting a string, defining delimiters, or rounding, truncating, or otherwise aggregating data.

Once the data is structured into a table, the data analyst can connect to the HDFS through a Hive client, such as the Hive CLI, the Beeline Shell, a GUI like Hue, or another program such as Spark, Tez, or Thrift.

Once the analyst submits their query, Hive works by converting the SQL-like HiveQL into MapReduce code. Therefore, once the Hive code is run, a MapReduce job is queued. This MapReduce process can take several minutes, hours, or even days for larger datasets, so it is important to be patient. Hive is very reliable, and the process will eventually finish in the majority of cases.

Once the MapReduce process is finished, results are output into the command line or other user interface. The analyst can also choose to store these results as a another table or as .csv file for later use.

These exported results can be loaded back into the HDFS, and the cycle can continue; analysts can continue querying manipulated data in this way.



INSTRUCTORS: Take some time to get to know your students, what their skill levels are, and what their goals for this course are.

Now that we know where in the data pipeline Hive fits, what about the other SQL-on-Hadoop tools that fit in the same place?

Hive is not the only SQL-on-Hadoop tool, but it was the first, and is compatible with most versions of Hadoop. This also means many data analysts already are familiar with Hive, and may have already written scripts in HQL.

Other SQL-on-Hadoop tools like Impala, Drill, and SparkSQL exist as well, but may not be compatible with all versions of Hadoop. Many data scientists first learn SQL, then Hive, then move on to one of these new tools when they find they need speed or a certain functionality which Hive lacks.

If you would like to learn more about Drill, refer to DA 410: Drill Essentials or DEV 360: Spark Essentials in the MapR Academy course catalog.

When to use Hive or Pig?

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

 MAPR 17

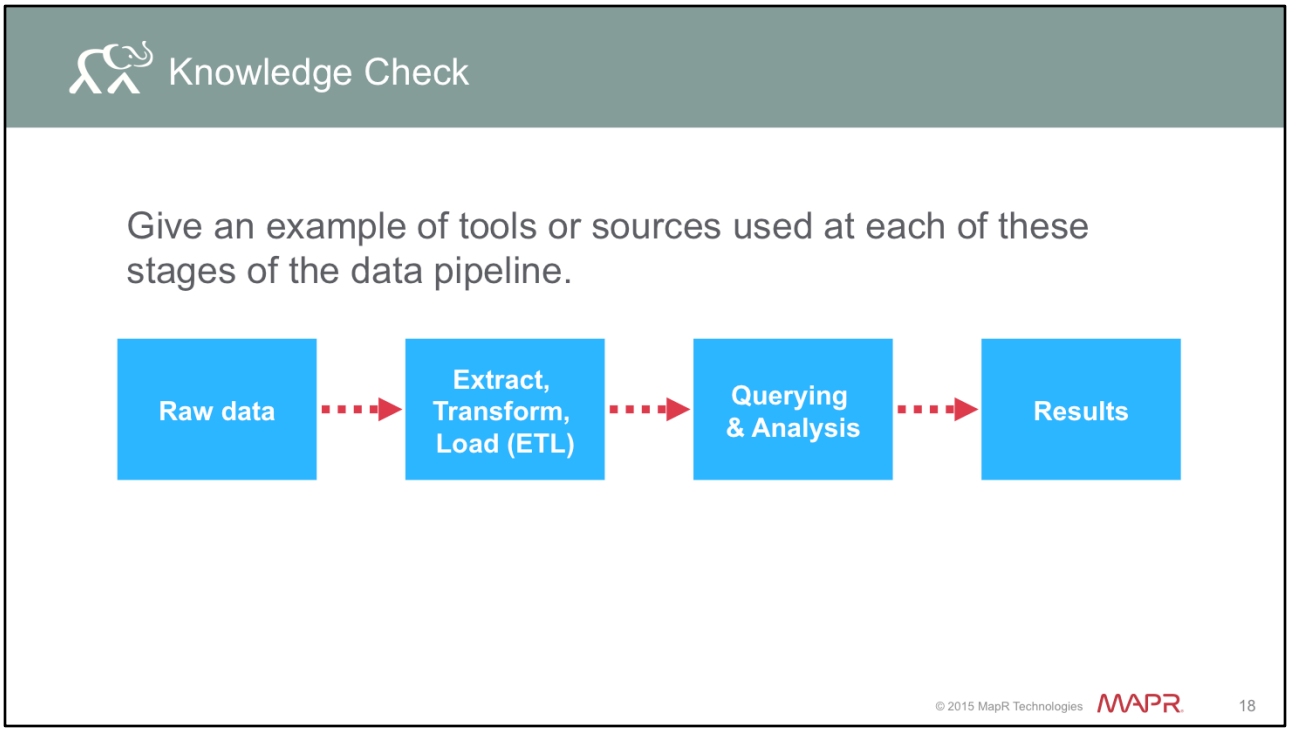
Since Pig and Hive are often used together, and there is some overlap in their capabilities, many users want to know when to use one or the other. It is important to note that both of these tools can preform many of the same tasks, but they are often more suited for one task or another.

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 query in the order you specify.

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.

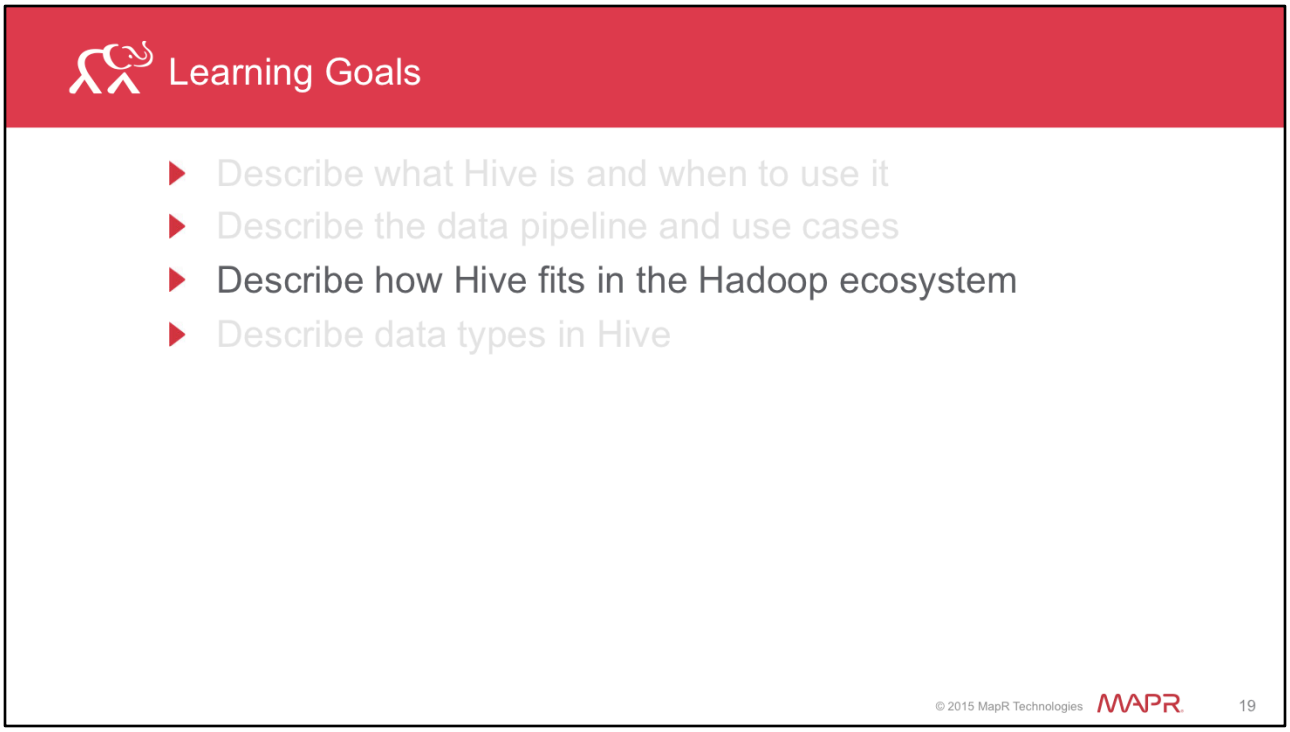
Hive runs batch queries on structured data while Pig can automate ETLs for unstructured data.

In the case of our Climate Research Institute, we will ultimately be using both Hive and Pig. Pig will be used for initial data processing and normalization, while Hive will be used for querying and data analysis.



Some possible responses:

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



In this next section, we'll learn a little bit about the technical specifications in Hive, including features found in different versions of Hive and how Hive interacts with other tools in the Hadoop ecosystem

HiveQL & MapReduce

Hive queries

- work on HDFS
- sent in batches
- Queries sent from Beeline or Hive CL
- Most queries trigger MapReduce

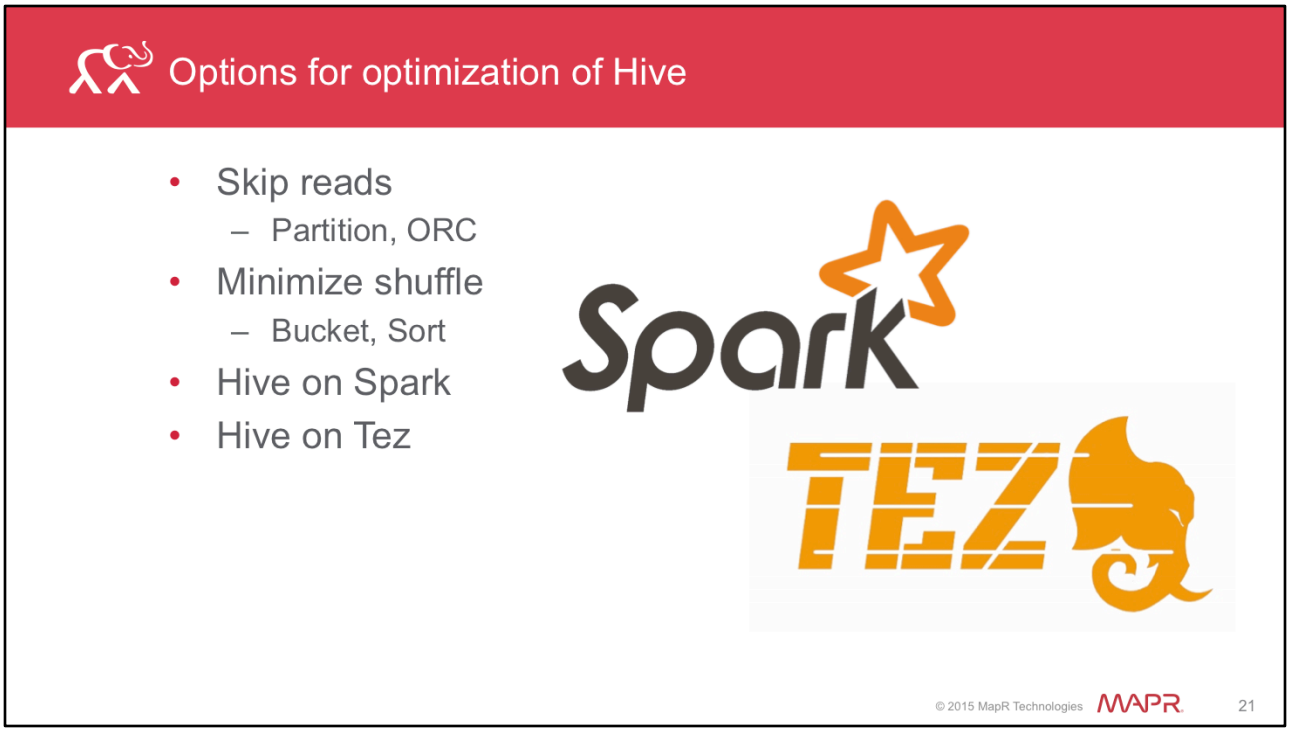
 MAPR 20

It is important to keep in mind that Hive queries are sent in batches.

Because they are sent in batches, and because some queries trigger MapReduce jobs, which requires a lot of overhead and can use a lot of computational resources. Each step in a Hive query, including joins, selects, or any other computational step, may queue up its own MapReduce job, adding additional computational overhead to the process.

An dataset may be several petabytes in size, and a Hive query may take hours or even days to process due to the computational overhead, but it is reliable and will finish eventually.

For users who are accustomed to the near-instantaneous results of SQL, it may appear that the query has failed. This is often not the case, and it is important to be patient when using Hive.



Speed is a major concern for many Hive users. There are several options for optimizing the performance of Hive.

One way to increase the speed of your hive queries is to skip reads, which minimizes the amount of data Hive has to read in order to preform your query. One way to do this is partitioning tables. Telling Hive to only read a part of a table, rather than all your data, can significantly improve performance.

Another way to skip reads is to take advantage of Ordered Record Columnar files. ORC files, when properly sorted, can also minimize the number of rows Hive must read to find the data which matches your query.

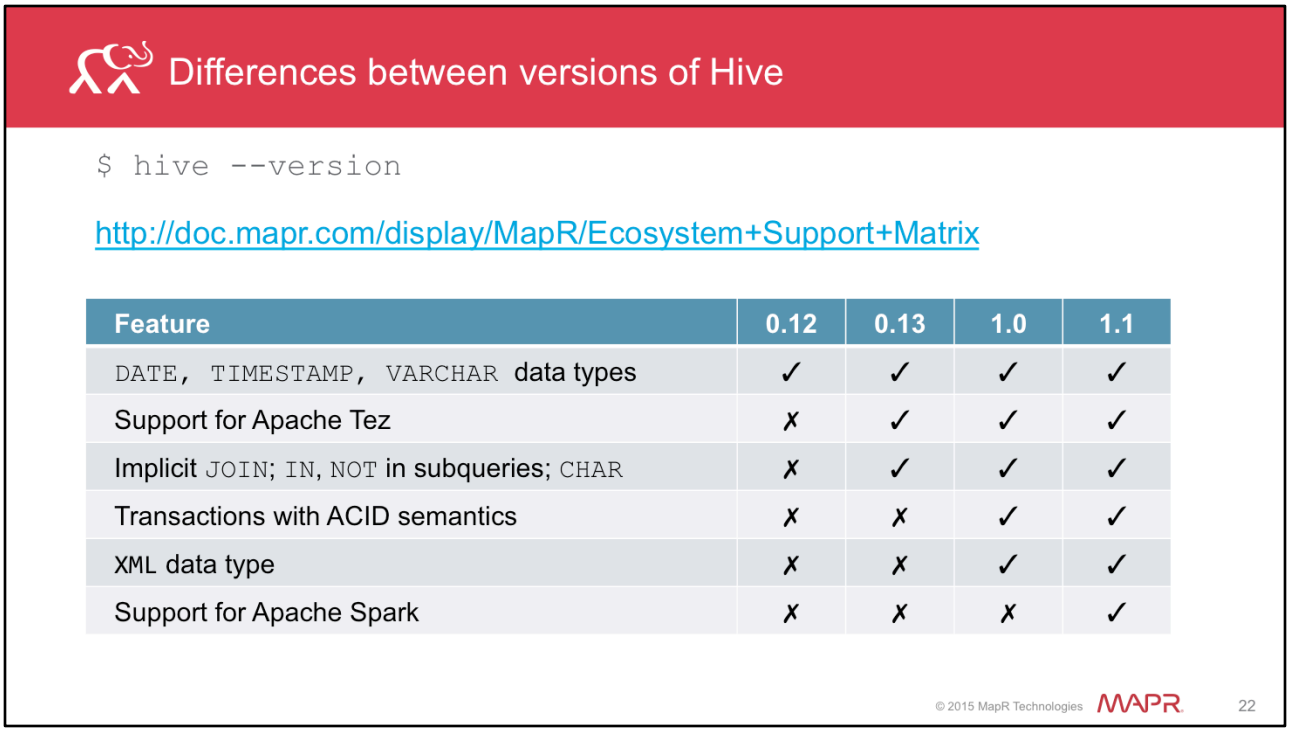
The second common way to optimize Hive queries is to minimize the time spent in the shuffle phase of the MapReduce process. Proper use of bucketing and sorting, especially when joining tables, can help reduce time in the shuffle phase.

Using Hive in conjunction with other Hadoop tools may also provide a performance boost.

Hive on Spark gives users the option to use Hive queries on data, but instead of triggering a MapReduce job, Hive on Spark uses the Apache Spark engine. This means your queries may be faster.

Likewise, Hive on Tez provides another alternative for Hive users to bypass MapReduce jobs.

While we will not cover using Hive on Spark or Hive on Tez in this course, it is important to be aware that these are options you can consider in the future.



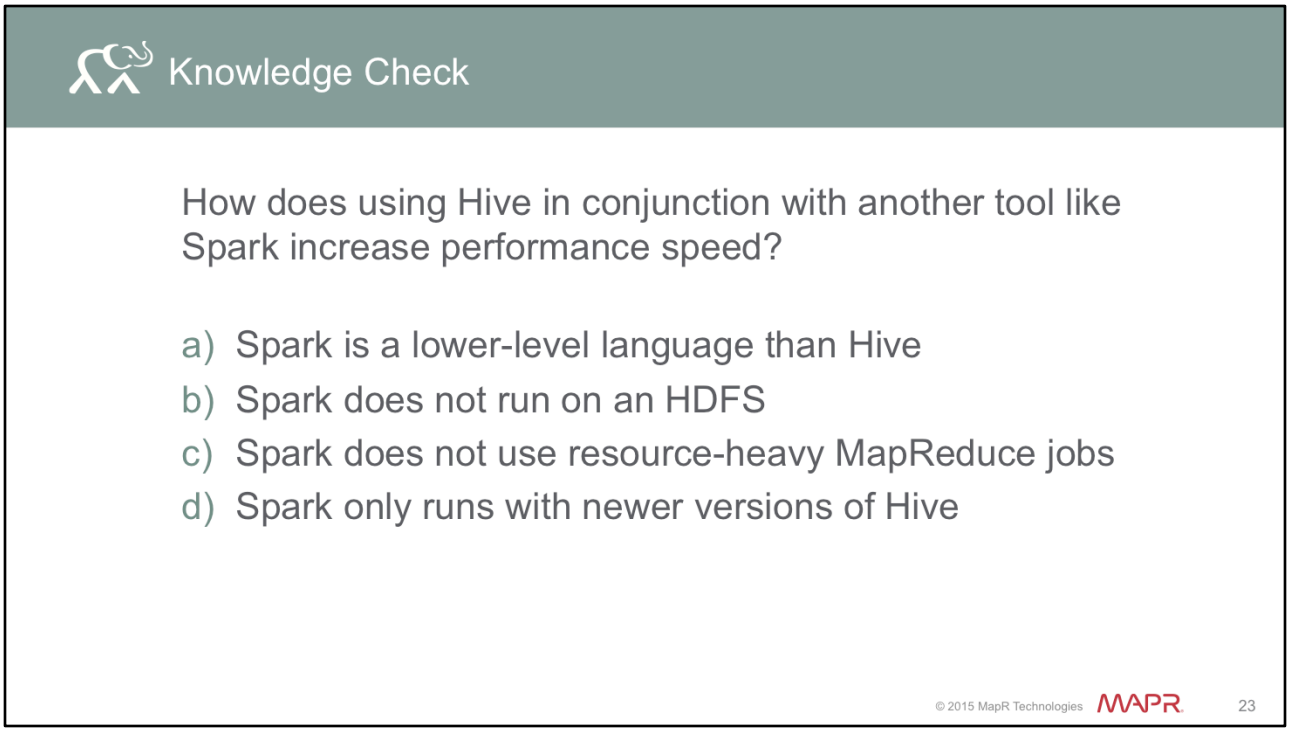
When using Hive, it is important to know what version you are using. You can check this from a bash shell on your MapR Sandbox by typing hive dash dash version. Once you know the version of of Hive you are using, you can refer to the MapR Ecosystem Support Matrix to make sure it is compatible with other tools you might wish to use in your data pipeline.

It is also important to know that Hive is not ANSI-SQL compliant. While each new version of Hive adds more features, certain features of SQL have not been implemented. This table demonstrates some of the different features available.

For example the use of IN or NOT in subqueries, was not available until version 0.13. Likewise, transactions with ACID semantics and advanced datatypes such as XML were not available until version 1.0.

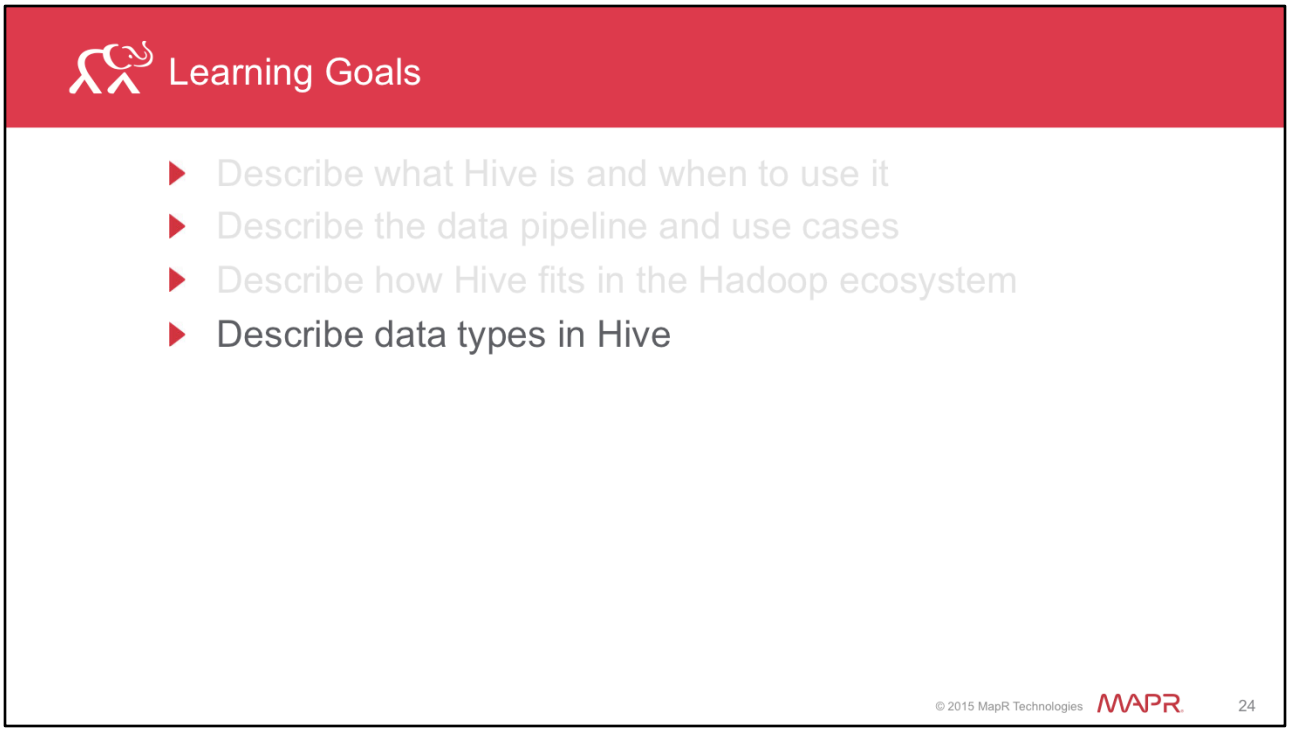
Version 1.1 features additional support for interacting with many other tools in the Hadoop ecosystem, including Apache Spark.

INSTRUCTORS: discuss some of these features with your students; which are necessary for their work?

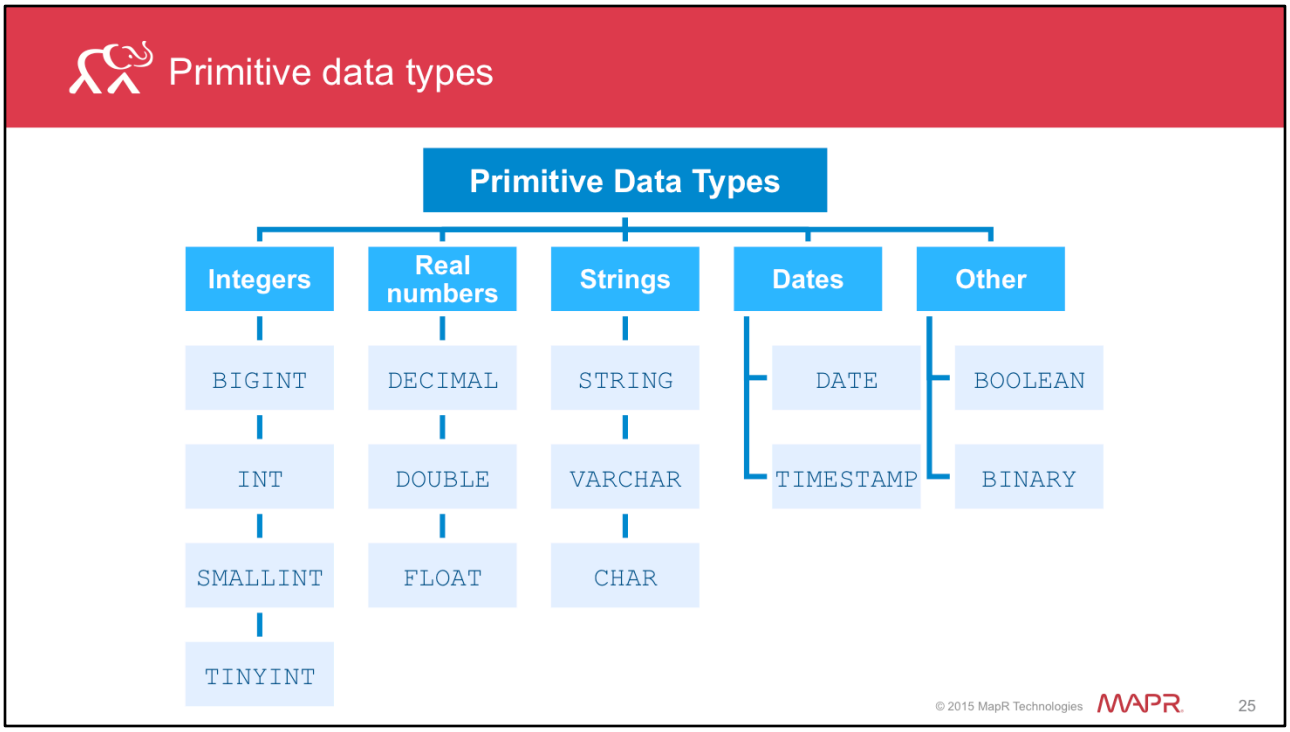


Answer: C

INSTRUCTORS: Discuss other tools your students use, or have heard of. What are the pros and cons of each?



Next, we'll learn about data types in Hive.



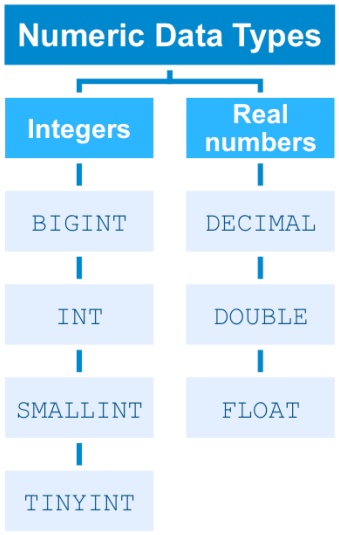
Hive tables can handle a variety of data types, but each column can only have one data type. You must specify whether each column contains an integer, a float, a string, a time stamp, an array, or some other data type whenever you create a table.

In most cases, Hive is able to cast up the data hierarchy, but not down. For example, a tinyint can be recast as an bigint, but a bigint might not be able to be recast as a tiny int. Likewise, a char can be cast as a string, but not vice-versa.

This diagram shows most of the primitive data types currently available in Hive. We will briefly discuss each, and then turn to complex data types.

Numeric data

- Integers are signed whole numbers
  - TINYINT -2^7 to 2^7-1
  - SMALLINT -2^{15} to 2^{15}-1
  - INT -2^{31} to 2^{31}-1
  - BIGINT -2^{63} to 2^{63}-1
- Real numbers include decimals
  - FLOAT 4 bytes
  - DOUBLE 8 bytes
  - DECIMAL 32 bytes
- **NUMERIC**, **MONEY** datatypes from SQL are not available in HiveQL



 MAPR 26

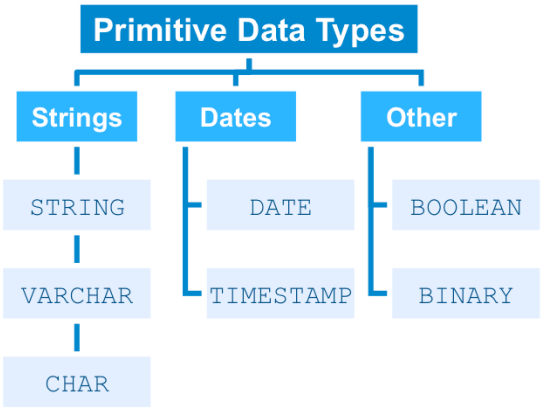
All integers are whole numbers. Integers in Hive are signed, meaning they can be either positive or negative.
The integer data type includes tinyint, smallint, int, and bigint, which correspond to 1, 2, 4, and 8 bytes of information, respectively.

Real numbers includes floats, doubles, and decimals. These can contain decimals of varying precision, with the DECIMAL datatype containing up to 32 digits.

Some commonly used data types in SQL, such as NUMERIC or MONEY, are not available in Hive.

Other primitive data types

- Strings
  - 'Hello world!'
- Dates
  - DATE 1970-01-24
  - TIMESTAMP 1970-01-24 08:52:48.123
- Other
  - TRUE or FALSE
  - Array of bytes



 MAPR 27

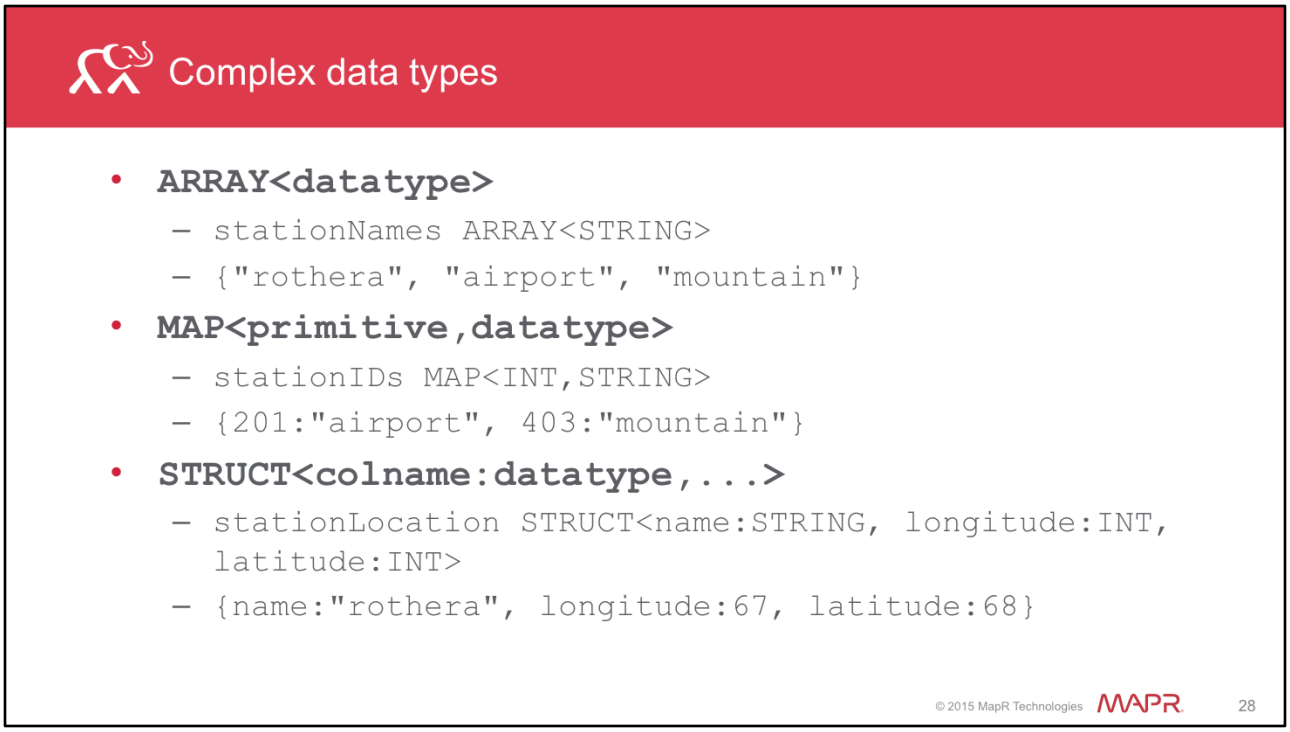
Other common primitive data types include strings, dates, and others.

Strings may be any length, but varchar and char are limited in length. Varchars may be specified as any length between 1 and 65355 characters long. Chars have a maximum of 255 characters.

Dates are useful for ordered data. The DATE data type takes the format YYYY-MM-DD, so that January 24th, 1970 would be expressed as 1970 dash 01 dash 24.

Timestamp on the other hand is the number of seconds since Unix time began. A timestamp includes the date in year-month-day format, followed by the time, in hour-minute-second format, followed by 3 decimal places, measuring thousands of a second.

Other primitive data types include Boolean, which is true/false. The binary data type is an array of bytes that is not interpreted by Hive.



Hive can also handle more complex data types, such as indexed arrays, key-value maps, ordered key-value maps called structs, and uniontypes.

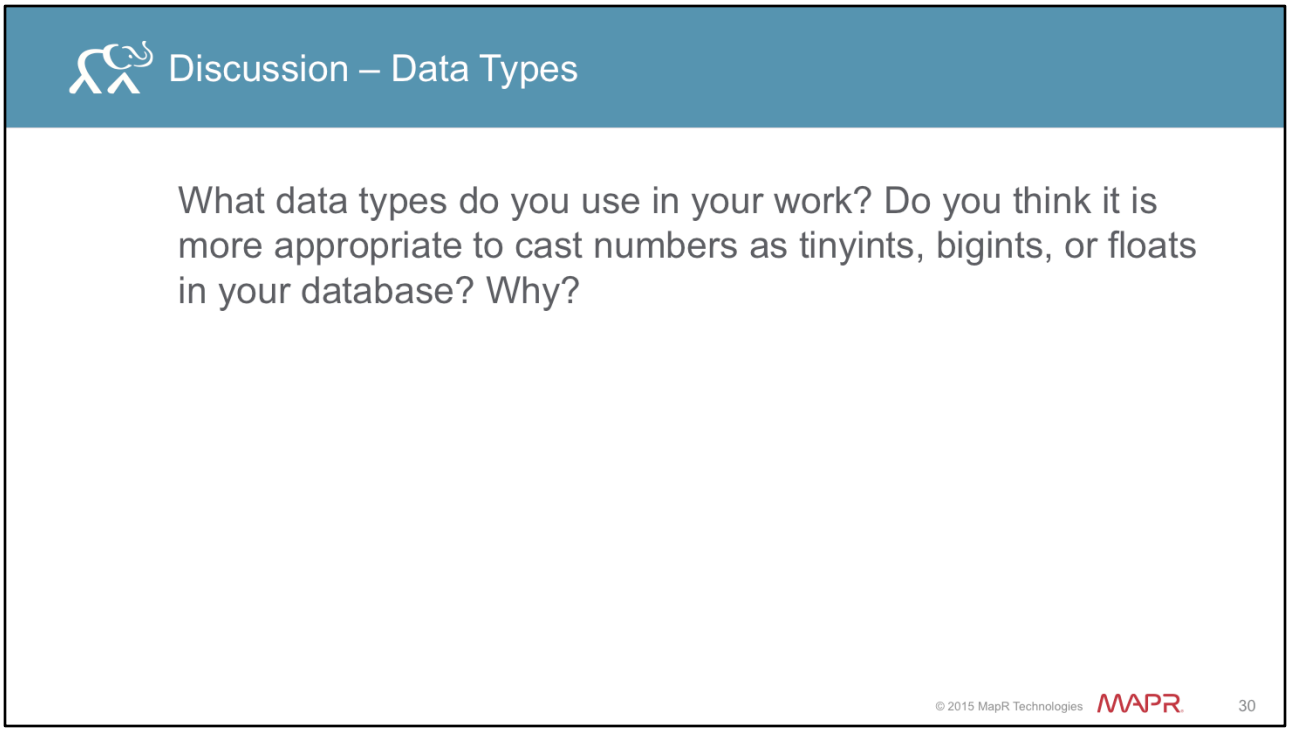
Arrays must contain objects of the same type. You could have an array of strings or an array of ints, but you cannot have an array which contains both strings and ints. For example, you might have an array of strings describing each of your weather stations

A map pairs one primitive datatype with one other datatype. Maps are unordered by default. For example, if each of your weather stations was also assigned a numeric ID number, you might map this ID, as an INT, to the station name, as a STRING.

A struct is a more complex datatype, which may contain several columns of several datatypes. For example, you might have a struct called stationIDs, which maps your station names, which are strings, and their latitude and longitude, which are INTs. In this example, our struct maps the Rothera station with its longitude and latitude, 67 degrees south and 68 degrees west.

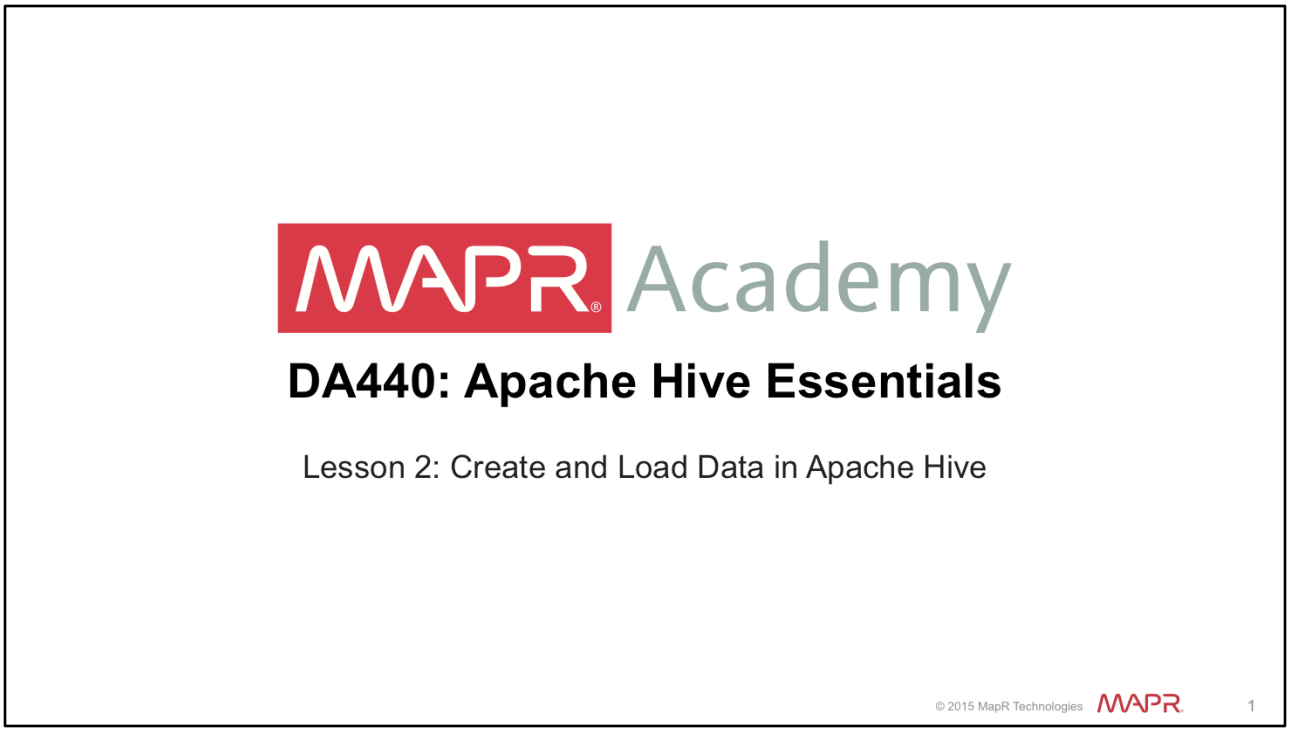
In this lab activity, we will load populate a small table with some sample data to learn how Hive deals with different data types. We will discuss the details of creating and loading tables in the next lesson.

Open your Lab Guide and complete Lab 1.2. When you are finished, you are ready to move on with the lecture.

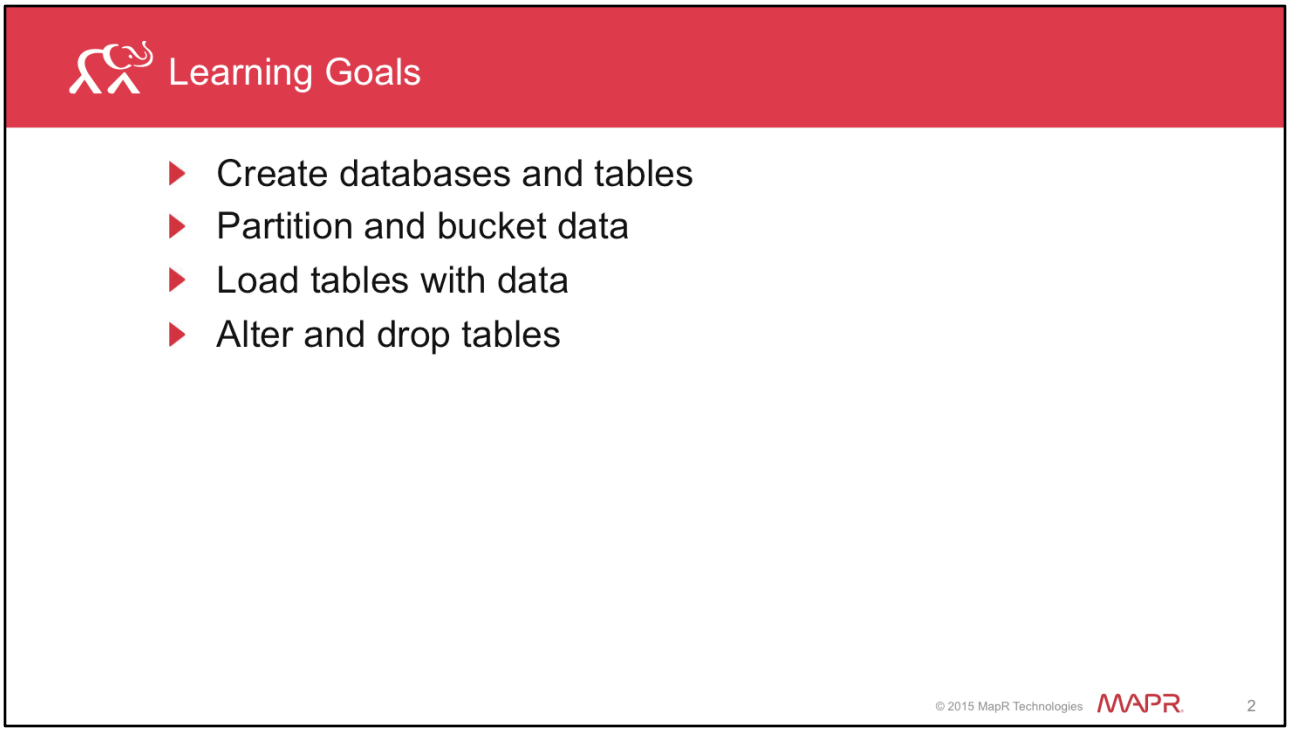


INSTRUCTORS: Have an open-ended discussion with your students about the datatypes found in their different industries. You may want to include a more in-depth discussion of complex datatypes, strings, or numerical data depending on the audience. You might also want to discuss the importance of thinking about the data types you are working with before you create data definitions.

In the next lesson, we will explore how to create tables and load data into Hive.

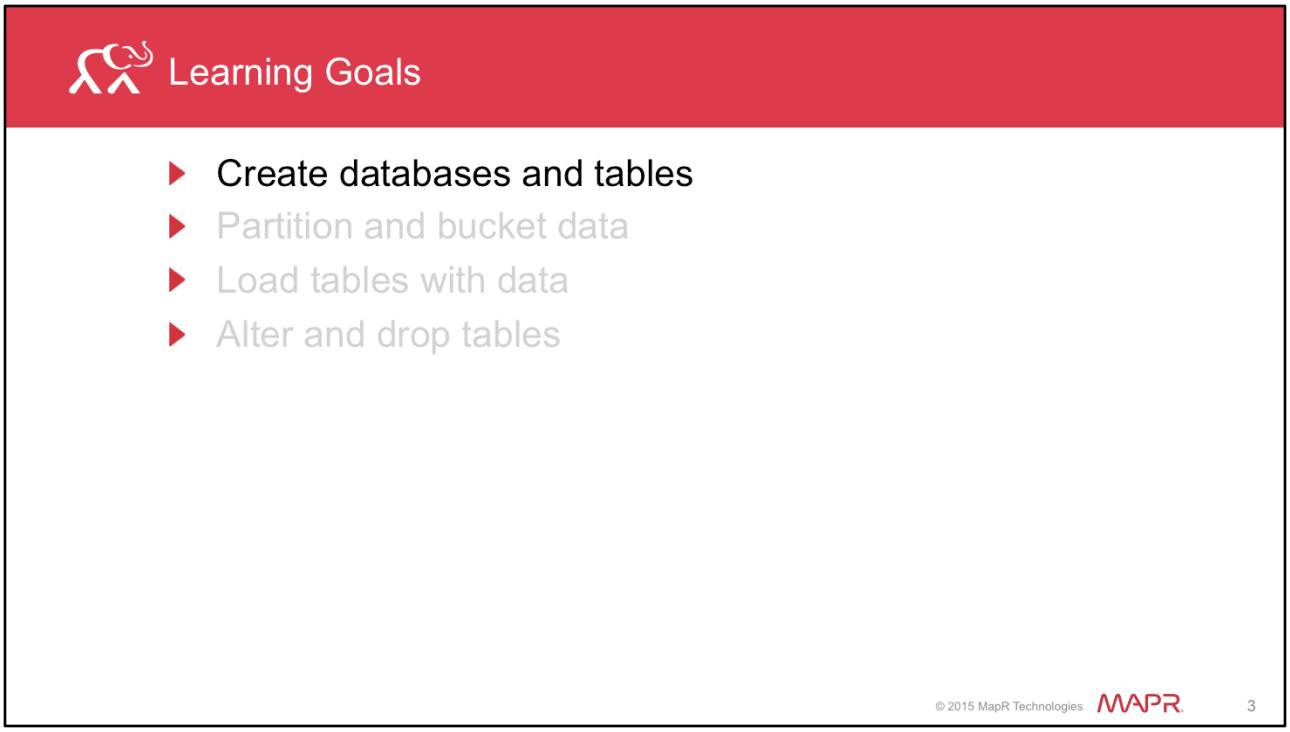


Welcome to DA440 Lesson 2: Create and Load Data in Apache Hive!
In this lesson, we will learn about Hive's data definition language to create tables and load them with data.



By the end of this lesson, you will be able to:

Create databases and simple tables
Create external and partitioned tables
Load tables with data
Alter and drop tables



Let's start by creating databases and simple tables.

Databases in Hive

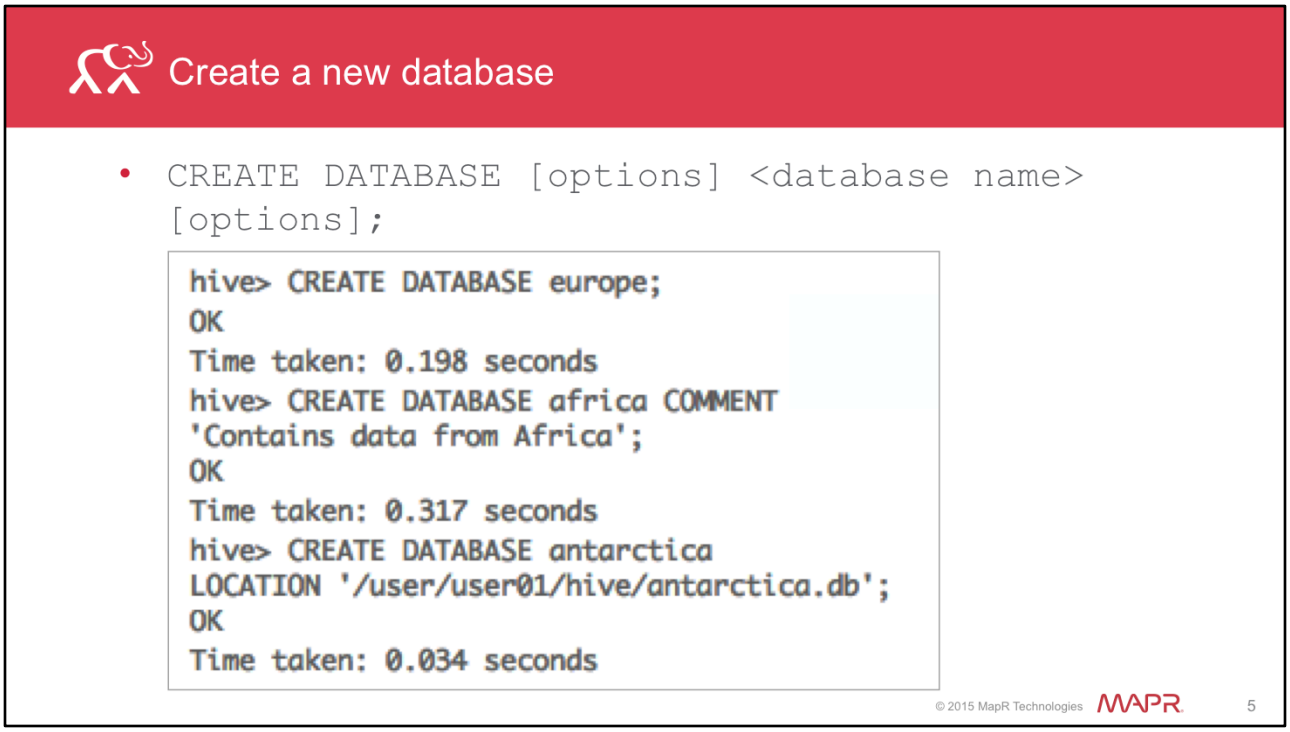
- Databases contain tables
  - `default` is used if you do not specify a database
- Tables contain data
  - HiveQL data definition creates metadata
- Hive itself does not contain data
  - All data is stored on HDFS
  - Hive provides metadata structure for accessing data on the HDFS

 MAPR 4

Databases in Hive contain tables, and tables contain data. Hive databases and tables serve as a structure that can be queried with HiveQL; however, Hive itself does not store any data.

Whenever you create a table in Hive, you must specify the database you create it in. If you do not specify which database your table is in, the “default” database is used.

If there are a large number of users on a cluster, grouping tables together into different databases can help organize your data. In the case of the Climate Research Institute, we might want to group data by year or by region, for example.



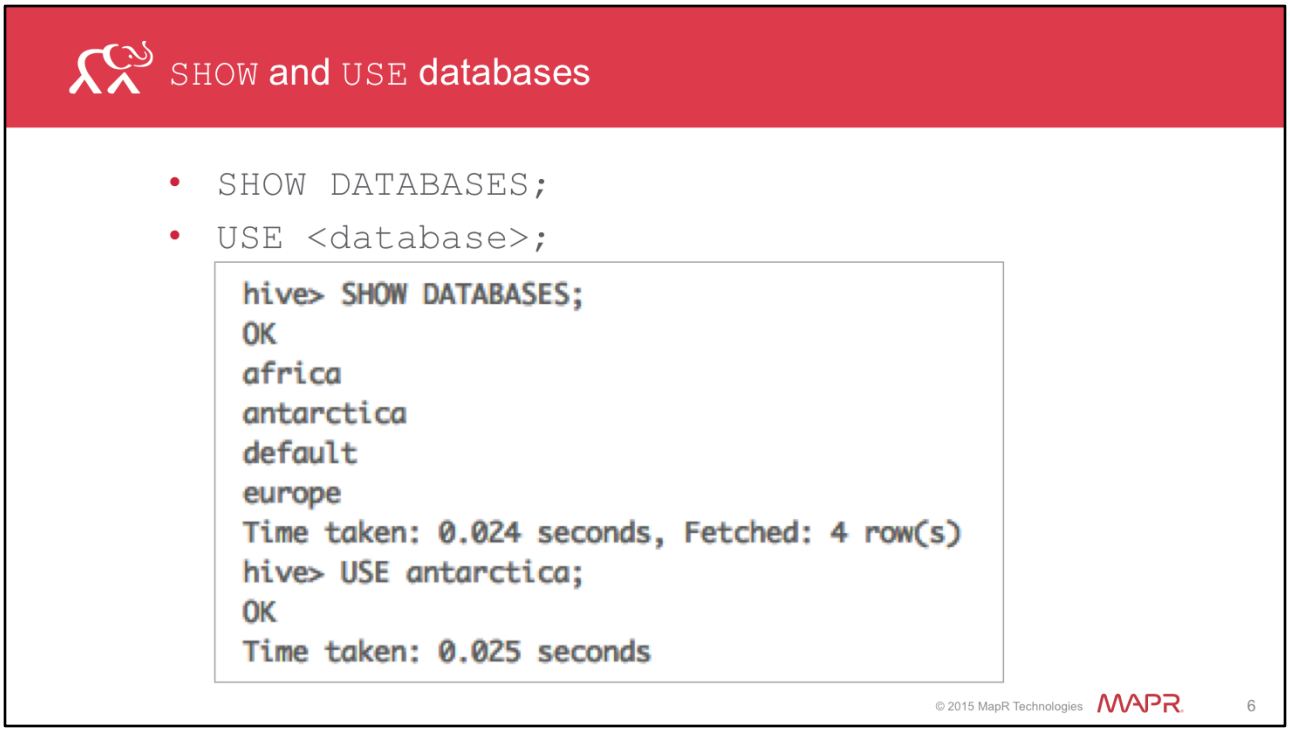
This example shows the syntax for creating new databases from the Hive Command Line Interface. For our Climate Research Center, we want to organize our data logically into several distinct databases. We have decided to make one database for each continent. Here, you can see the creation of the Africa, Antarctica, and Europe databases.

To create a database, at minimum, you must use the CREATE DATABASE command, provide a unique name for your database, and end your statement in a semicolon. We did this to create the Europe database.

There are several options you can add to the CREATE DATABASE command.

For example, you may add a comment about your database. This comment must be inside single quotes, and occurs after the keyword COMMENT following your database name. In this example, we added a comment to the "Africa" database.

You may also specify a location for your database to be stored. To do this, use LOCATION followed by the file path . Here we specified a location for the Antarctica database.

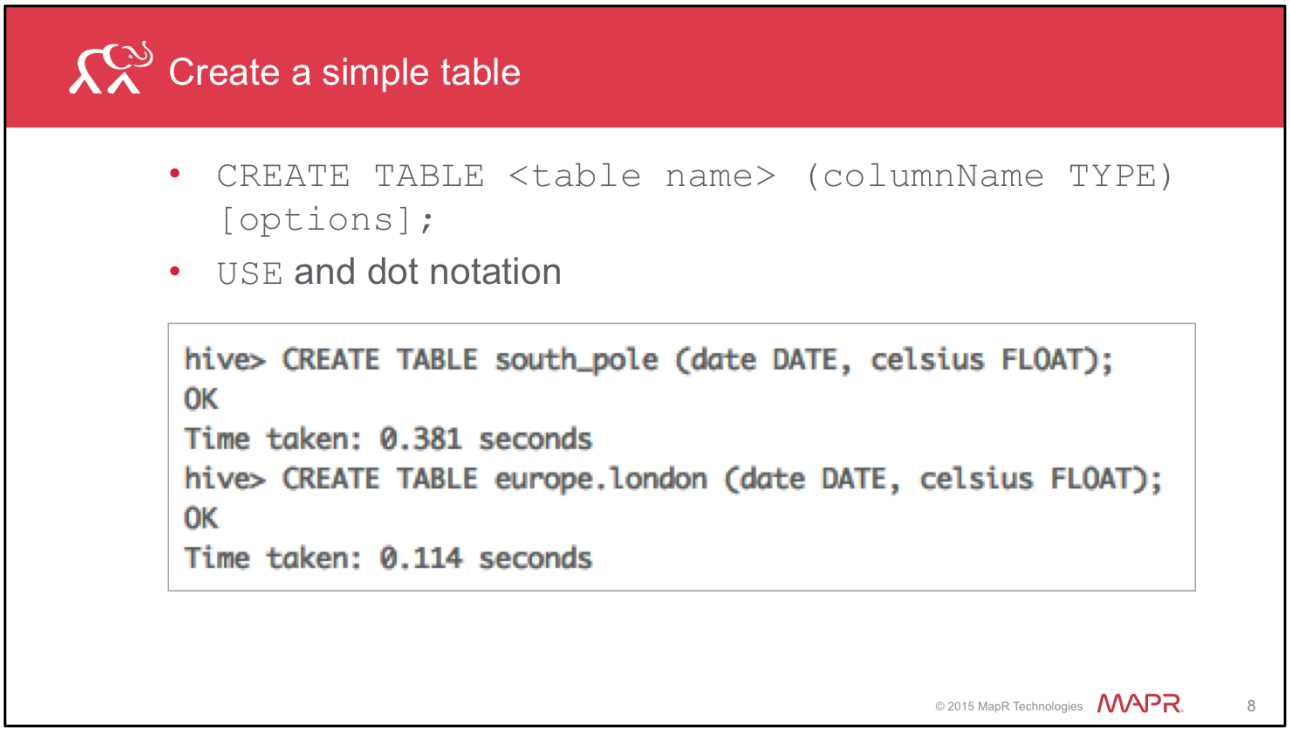


Once you create databases in your cluster, use SHOW DATABASES to list them. Your databases will be listed in alphabetical order. As you can see, we now have the africa, antarctica, and europe databases created in our cluster, in addition to the default database.

To change the working database, use the USE keyword. In this example, we set Antarctica as our working database. For the remainder of the course, examples will be created inside the Antarctica database, unless otherwise specified.

Now you can create your first Hive database. Remember, each database on your cluster needs a unique name. If you are in a classroom environment where several students are sharing the same cluster, it is important to create a unique database name. Each student should create a database associated with their own username; if your assigned username is user02, for example, then your database will be named user02.

Open your Lab Guide, and complete Lab 2.1. When you are finished, you can explore the Hive command line interface by trying familiar SQL commands, or you can continue with the lecture.



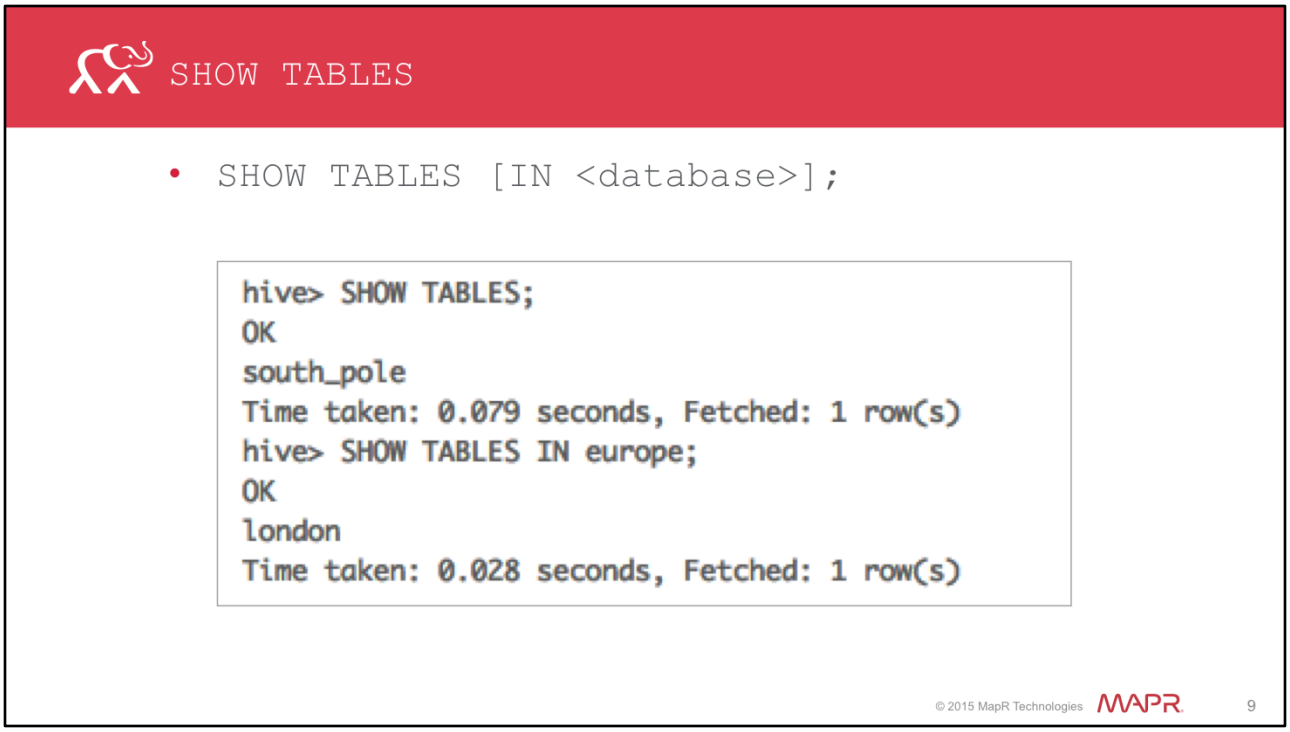
To create a table, use the CREATE TABLE keywords, followed by the name of the table. Like creating a database, Hive will throw an error if the table name already exists.

After the table name, list each column name and its data type in parentheses. We'll discuss data types in Hive later.

As with all Hive commands, end the data definition with a semicolon. By convention, keywords like CREATE and FLOAT are written in allcaps; however, like SQL, Hive is case-insensitive.

Since we already specified we're using the Antarctica database with the USE command, the "south pole" table in this example will be created in the Antarctica database. If we want to add a table called "London" to the "Europe" database, we can either switch databases with the USE command again, or we can use dot notation. By typing "europe.london", we specify the London table is in the Europe database.

To ensure your tables and queries are always made in the correct database, it is a good practice to always use dot notation.



We can see which tables we have already created by using SHOW TABLES.

Since we set our working database to antarctica, using SHOW TABLES right now will only show the south_pole table we just created.

If we want to see the london table, we can either set the working database to europe, or we can use SHOW TABLES IN europe; as shown in this example.

SHOW TABLES is a great way to verify you are working in the correct database, and that your tables were created correctly.

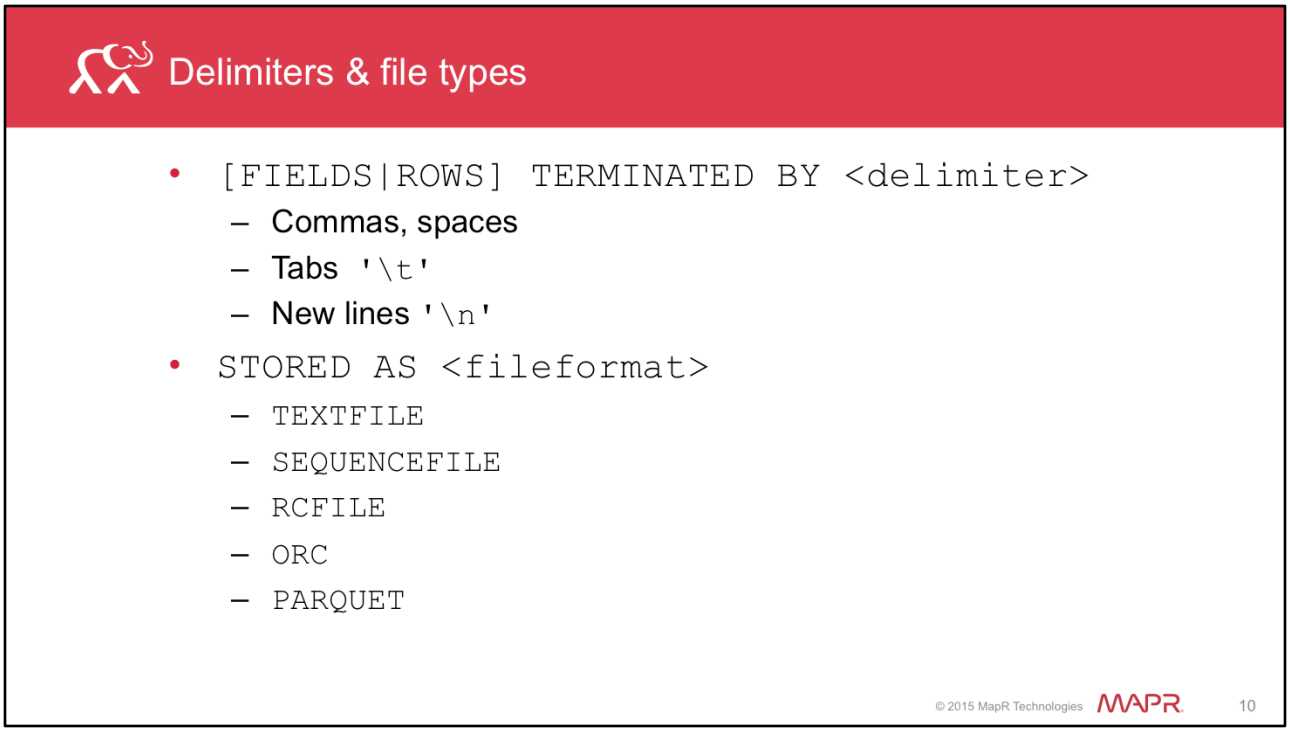


Table delimiters might be commas, spaces, tabs, new lines, or a variety of other characters such as hashes, dollar signs, or semicolons.

You can also specify how your file is stored. The most common file tyles are textfile and parquet files.

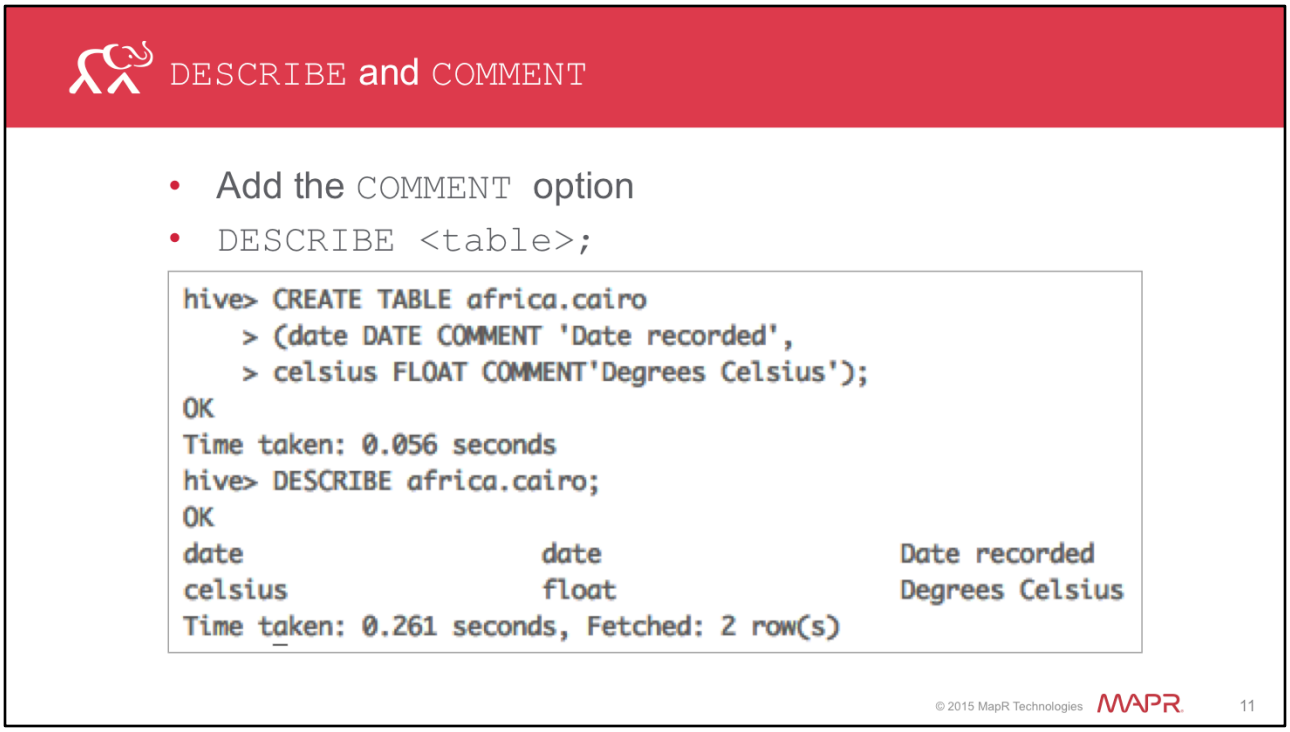
One common file formats is a textfile, which includes .TXT and .CSV.

A sequence file is the native file format of the Hadoop Distributed File System. Sequence files are often split into chunks across mappers.

Record columnar files, or RC files, are a flipped matrix. Columns are stored as rows, and rows are stored as columns. This makes tall narrow data into flat wide data. Depending on the nature of your data, using this file format may help speed processing times.

Ordered record columnar files, abbreviated as ORC, have the same features as RC files, with the addition of headers and footers for metadata and indexing. The ordered nature of ORC files means Hive can sometimes skip reads, again leading to an increase in speed.

Finally, parquet files offer a compressed, efficient columnar file format which works with a variety of tools in the Hadoop Ecosystem, including Hive, Pig, Drill, and Spark. You can read the Apache Parquet documentation for more information about this file format.



Let's create another table, Cairo, in the Africa database. This time, we'll add some comments to our columns so we know what they are.

For ease of readability, we have divided this table creation into several lines. Remember, Hive won't run your query or command until you enter a semicolon to denote your query or command is complete.

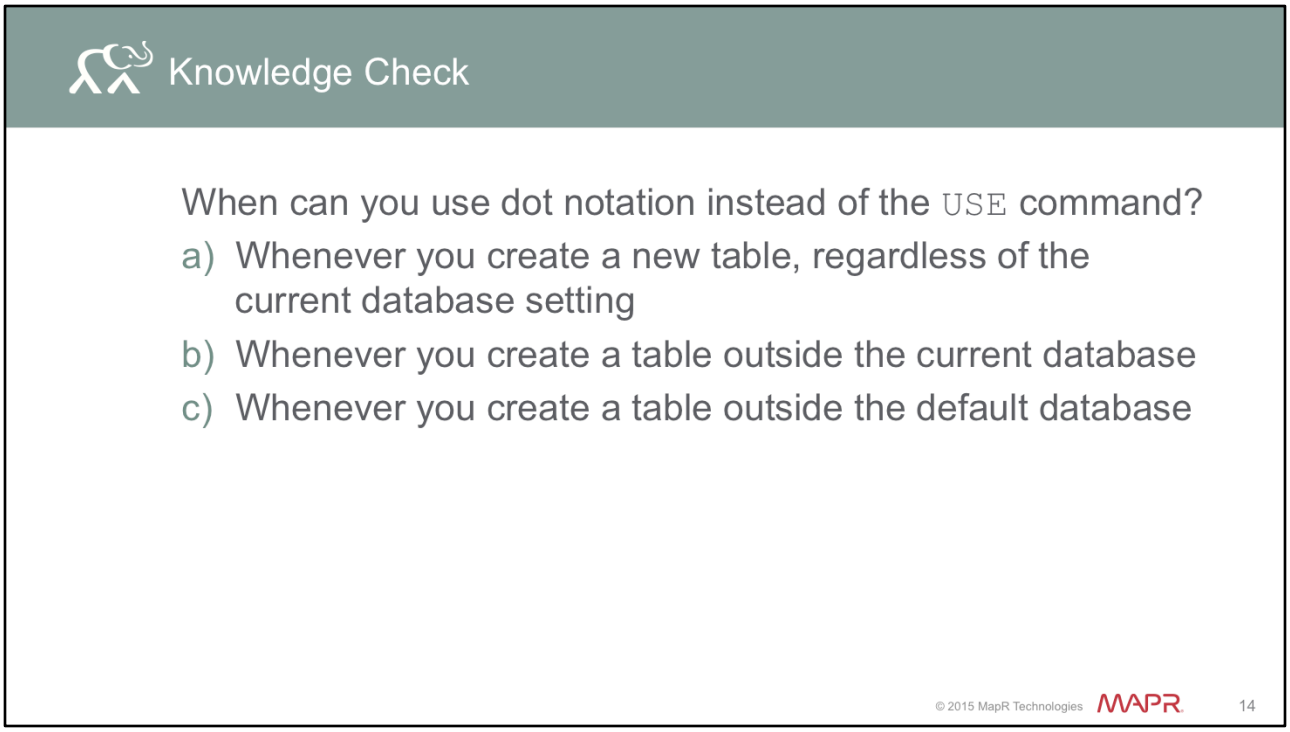
Now that we've created this new table, let's examine it with Describe. Describe will list all the column names, the data type of the column, and any comments we added about the column. In this example we see that the column called "date" contains a data type of "date" and the note that it is the date recorded.

Like SHOW TABLES, Describe is another great way to verify your tables were created successfully.

In addition to DESCRIBE and SHOW TABLES, you can also use SHOW CREATE <table> to verify your data was created correctly.

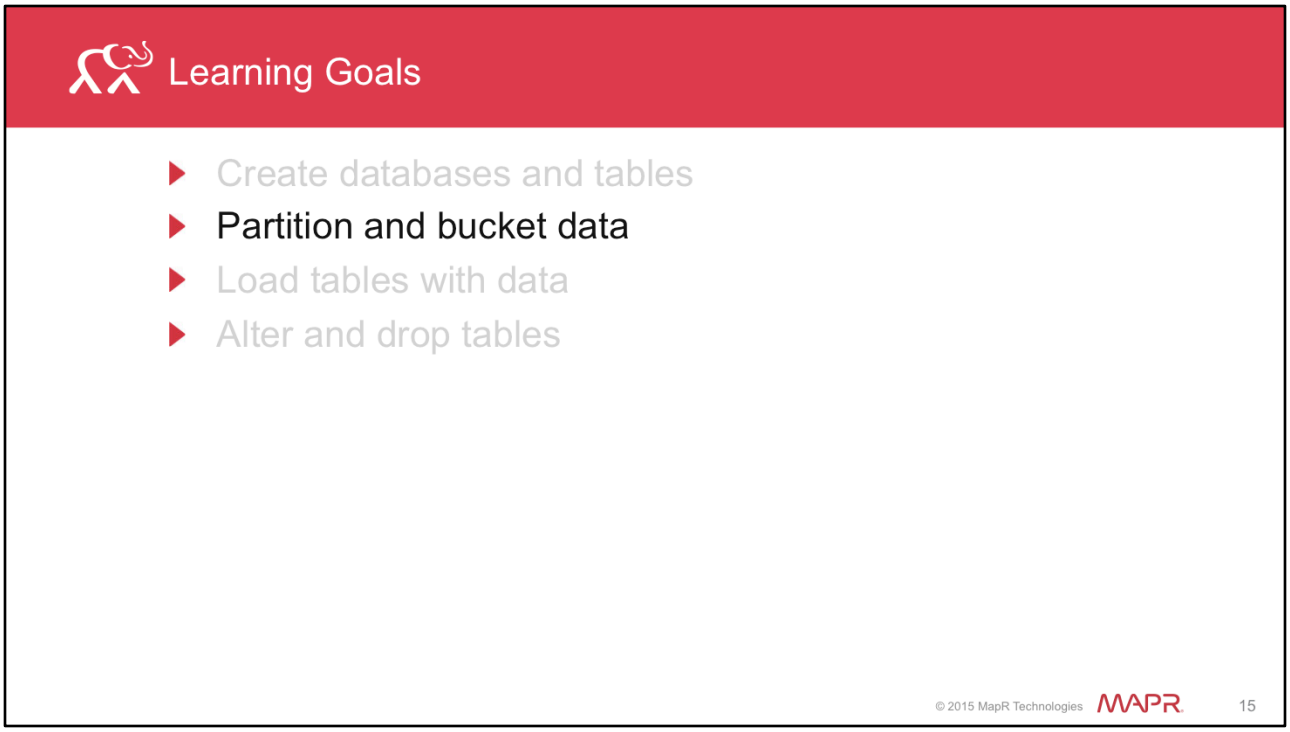
SHOW CREATE TABLE provides details from the data definition of a table.

Now is a great time to create your first Hive table. Open your Lab Guide, and complete Lab 2.2. When you are finished, you can continue to explore the Hive command line by trying familiar SQL commands on your own, or you can continue with the lectures and labs.

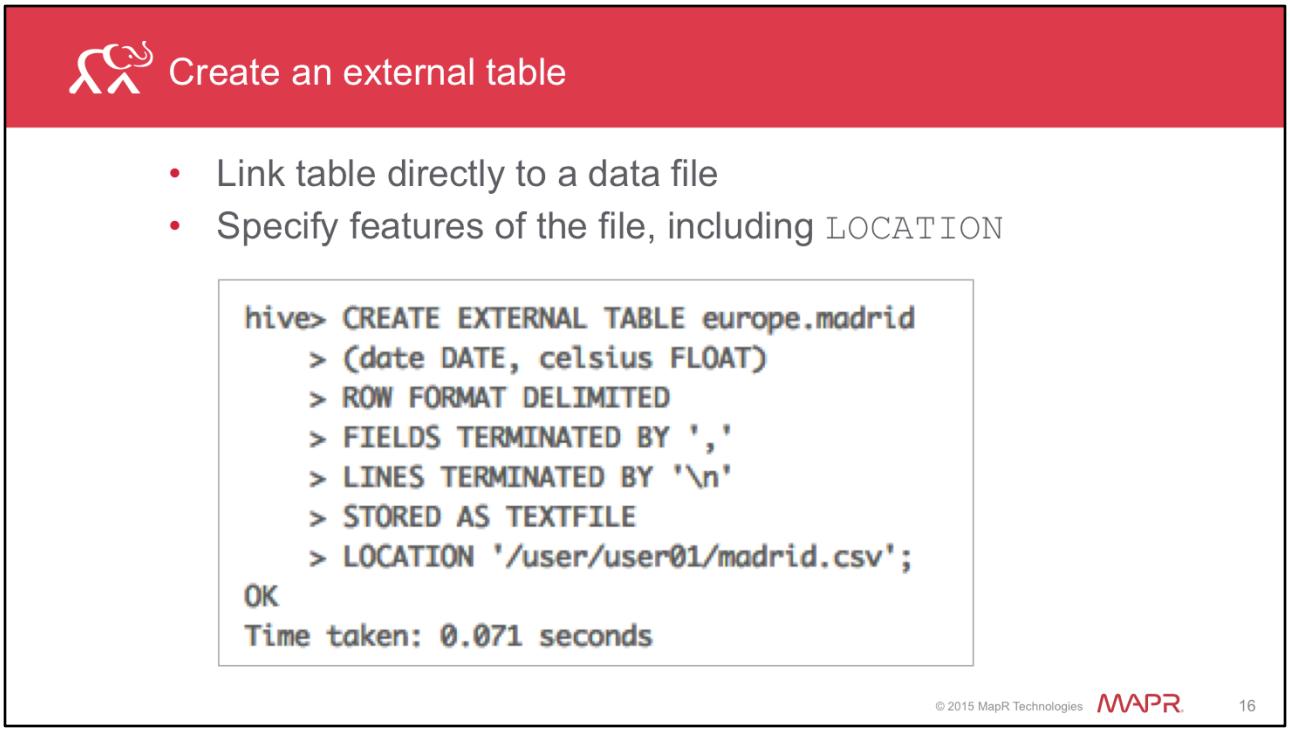


Answer: A (technically all of the above)

To ensure you always create your tables in the correct databases, you can always use the dot notation and not worry about using the USE command.



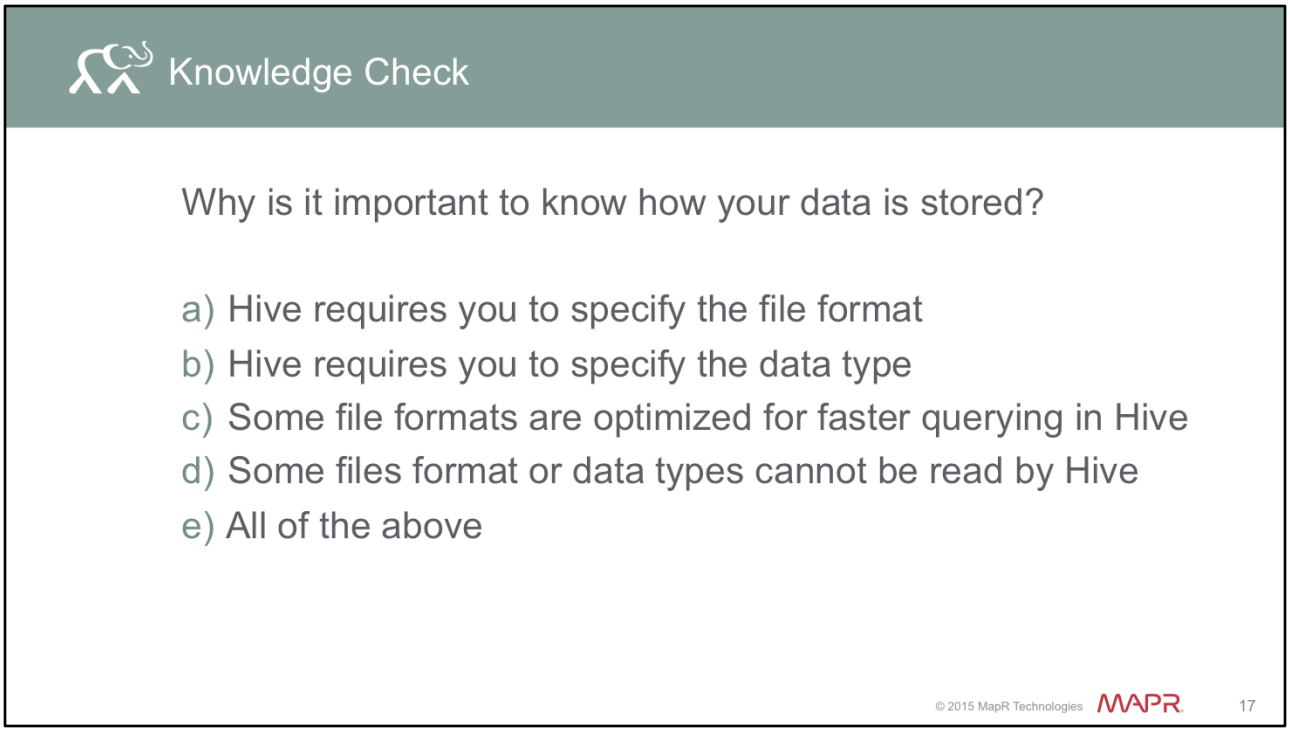
Next, we'll create external tables and partitioned tables.



External tables are linked directly to an external data file. This has several advantages. First, you load the data and create the table simultaneously in the same data definition. Second, external tables are read-only. Read-only is a good option if you do not want your users to accidentally edit or delete data. Users can still create and save views from an external table, but the external table will remain unchanged, as it is linked to an external file.

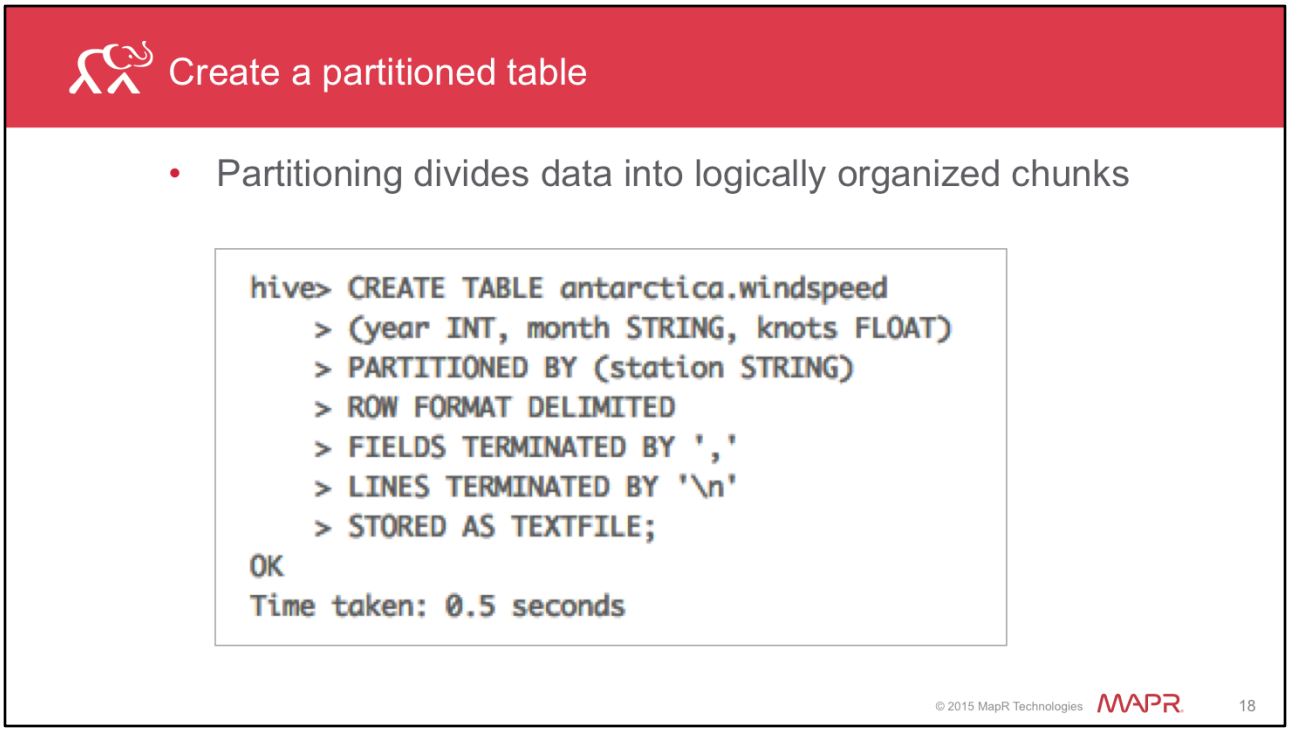
When creating a table, you can specify many features such as the data type of each column, the row format, delimiters, how the file is stored, and its location. In this example, we have specified a row format which is delimited by commas, with lines terminated by a new line character, and is stored as a text file. These settings are common for CSV files.

When you are creating an external table, you must specify a location on your database. This will link your table to data, such as a csv file. In this example, we are linking this table to a file called "madrid.csv".



Answer: all of the above

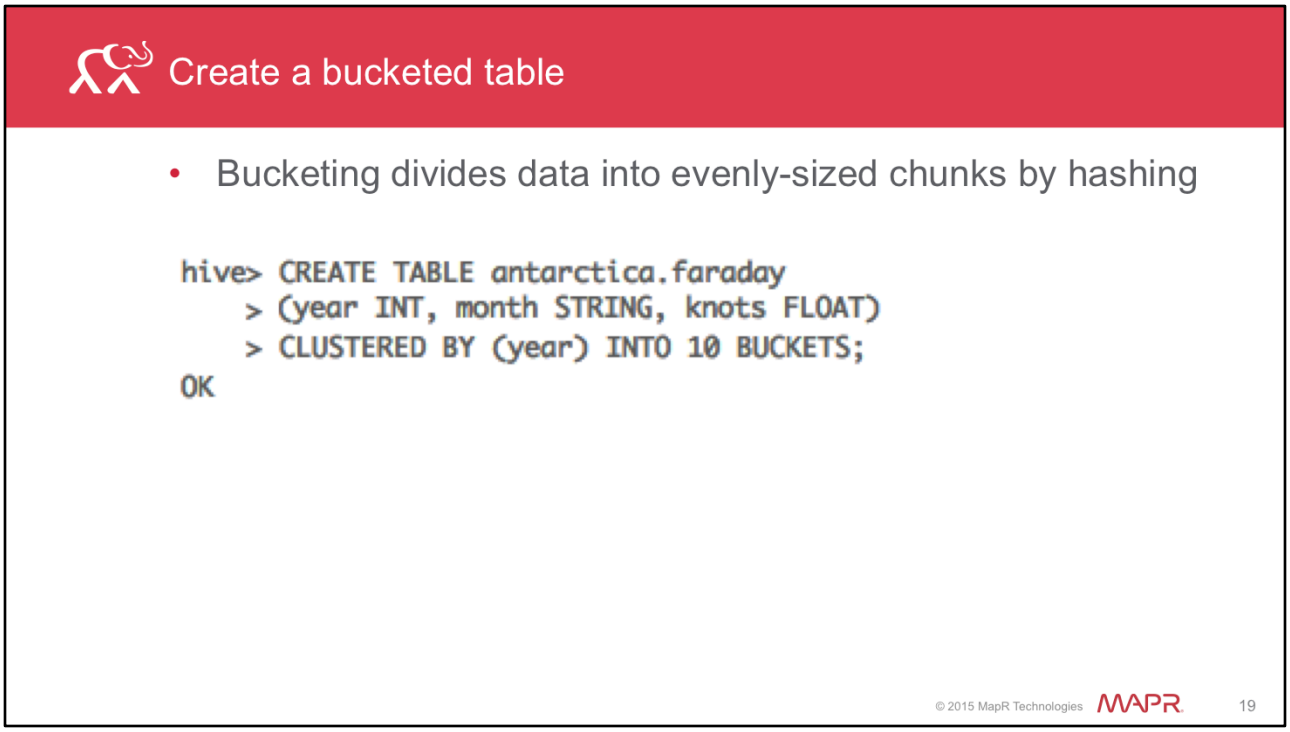
INSTRUCTORS: Take some time now to discuss with your students which file formats they plan to use.



In partitioned tables, the data is divided into smaller chunks. Each chunk of rows will correspond to a partition, which we define logically. These chunks can reduce the number rows Hive must read, which may result in faster processing times.

In this example, we have a large dataset which includes wind speed data, in knots, across several different weather stations. We can partition the data by the station, so that each station can be queried separately without querying the whole table.

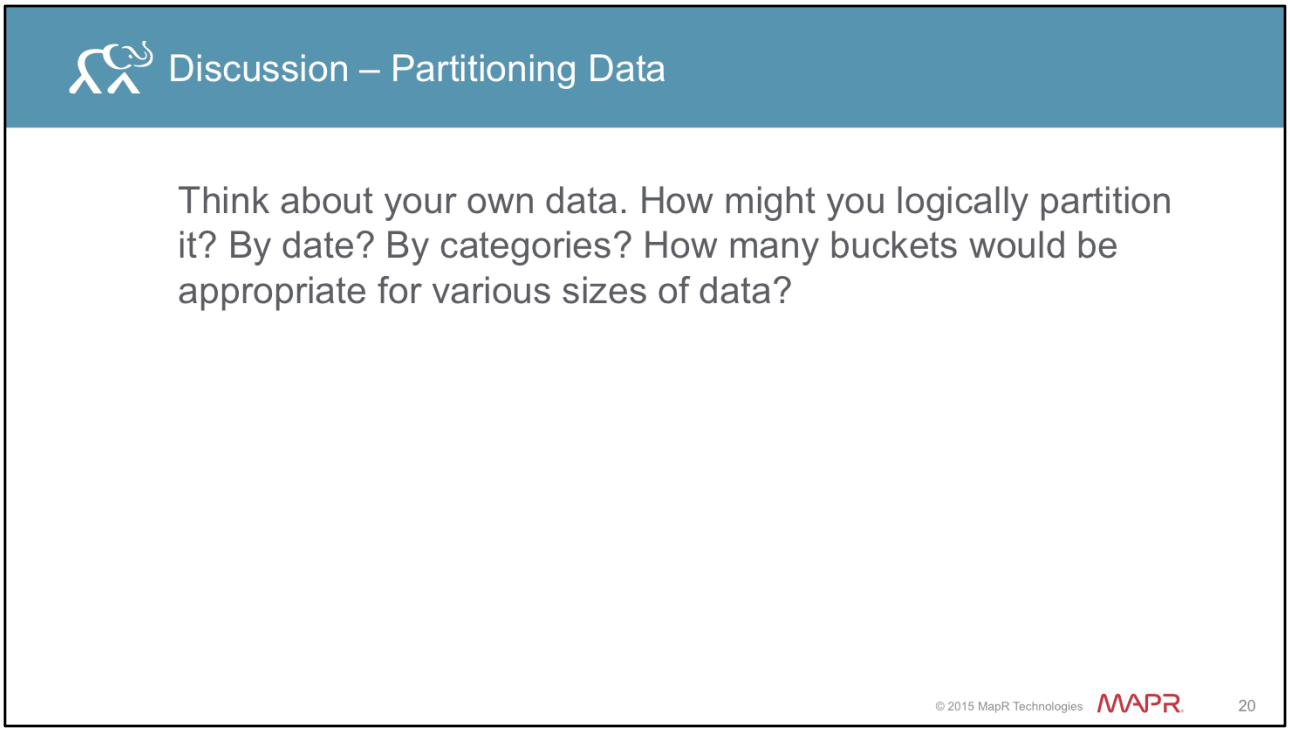
When we load this data later, we will load data corresponding to each station separately.



Like partitioning, bucketing also divides data into smaller chunks. These chunks also reduce the number of reads Hive must make, which may result in faster processing times. Unlike partitions, buckets are not defined by us. Instead, buckets are sampled from the data by a hashing function.

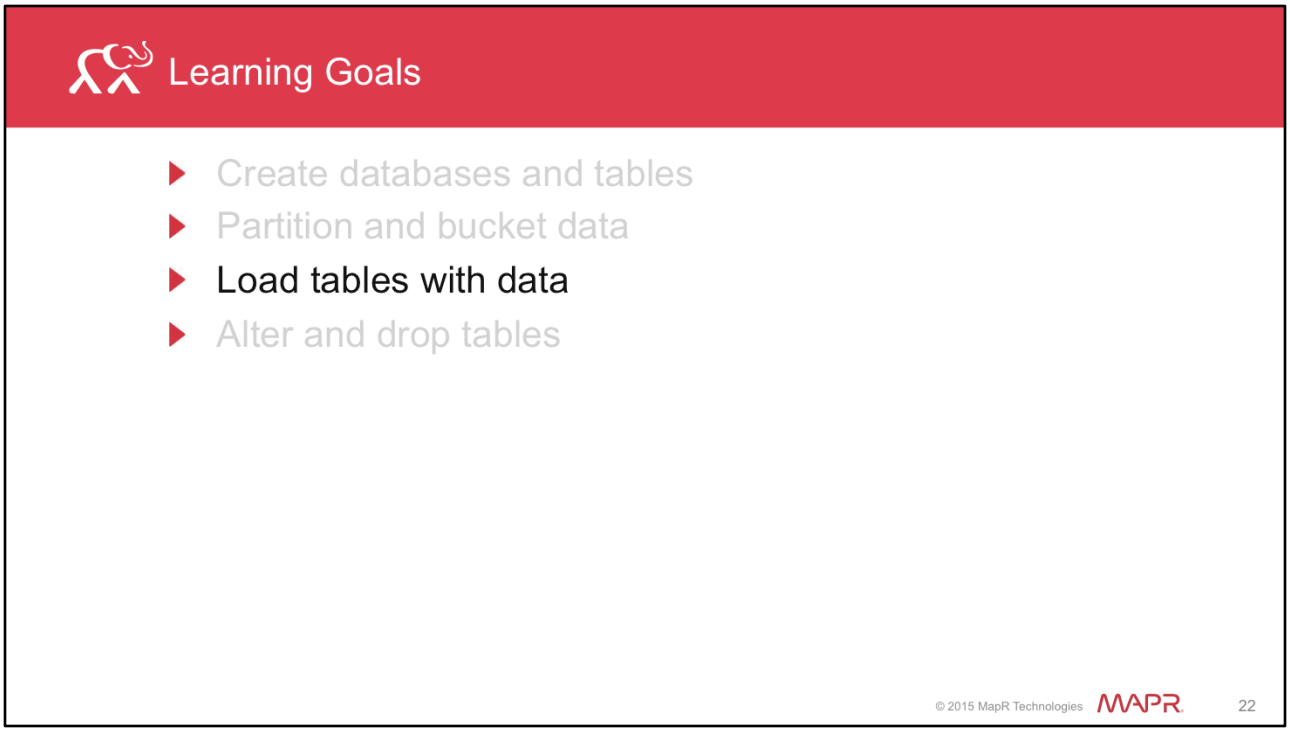
In this example, we bucket the data from the Faraday weather station into 10 buckets. The data will be bucketed by the year. The hash function will create 10 evenly sized samples of the data, and organize each bucket by the year column.

When we load this data later, the data is loaded all at once. The bucketing occurs automatically.

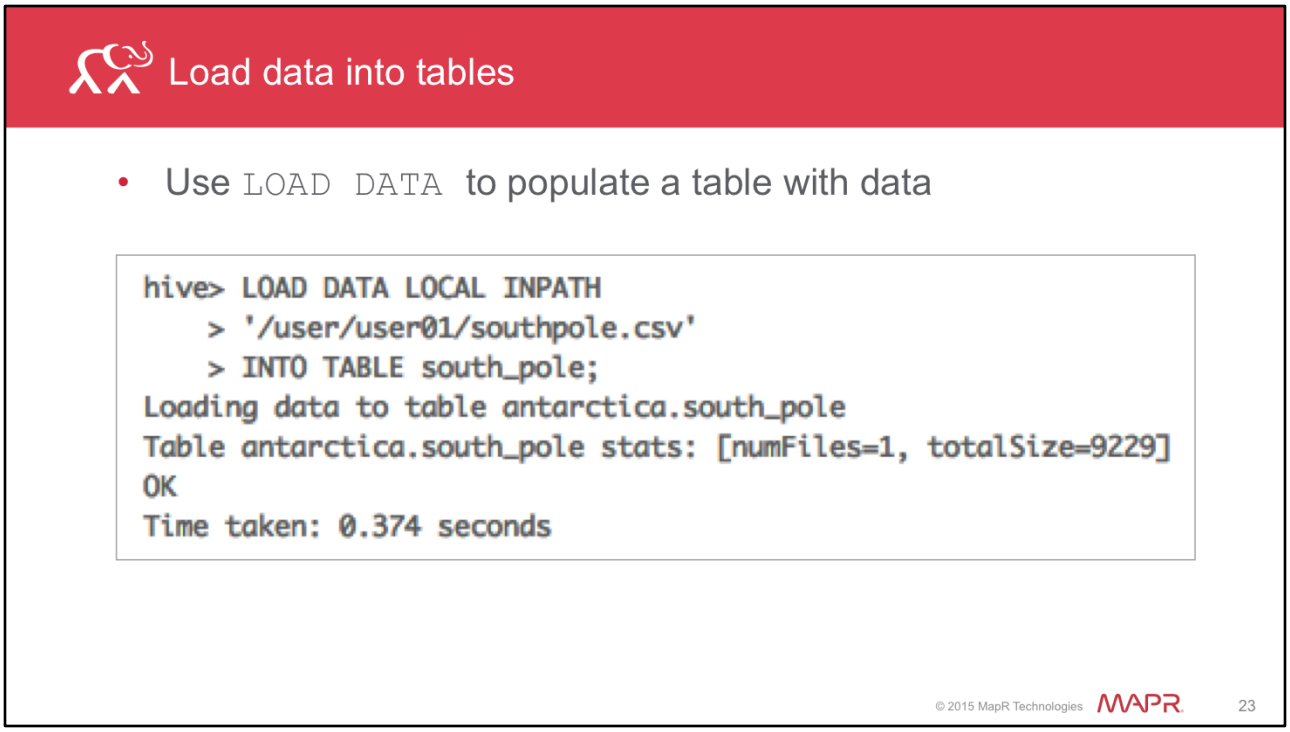


INSTRUCTORS: Take some time to discuss this with your students.

Now you will create external and partitioned tables Hive. Open your Lab Guide, and Lab 2.3. When you are finished, you can continue to explore the Hive command line by trying familiar SQL commands on your own, or you can continue with the lectures and labs.



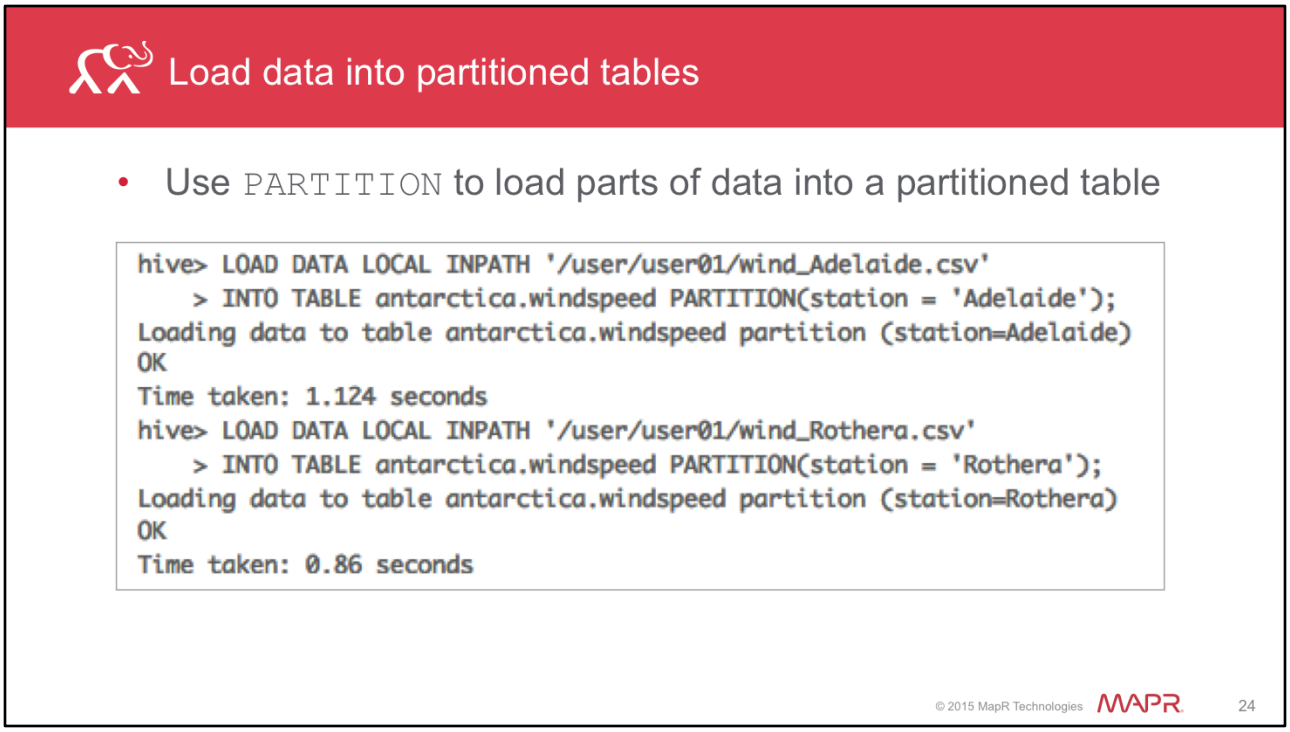
In this next section, we'll learn how to load data into internal and partitioned tables.



The LOAD DATA command inserts data from a file into a table. LOAD DATA only works if the table is already defined using the data definition language.

The LOAD DATA command can load data from a local file path. Specify the file path with LOCAL INPATH, then specify the name of the table with INTO TABLE.

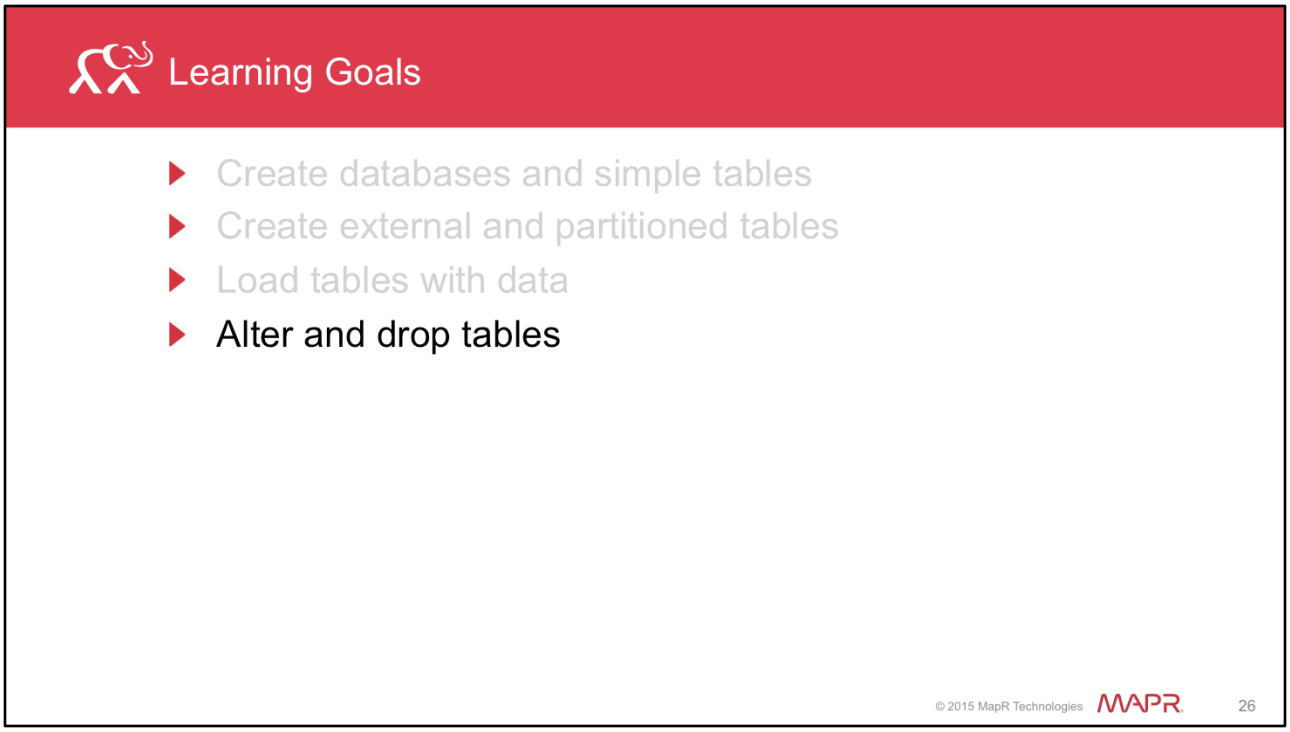
This table will be created in the current working database, which is Antarctica. It is good practice to use dot notation to specify the database.



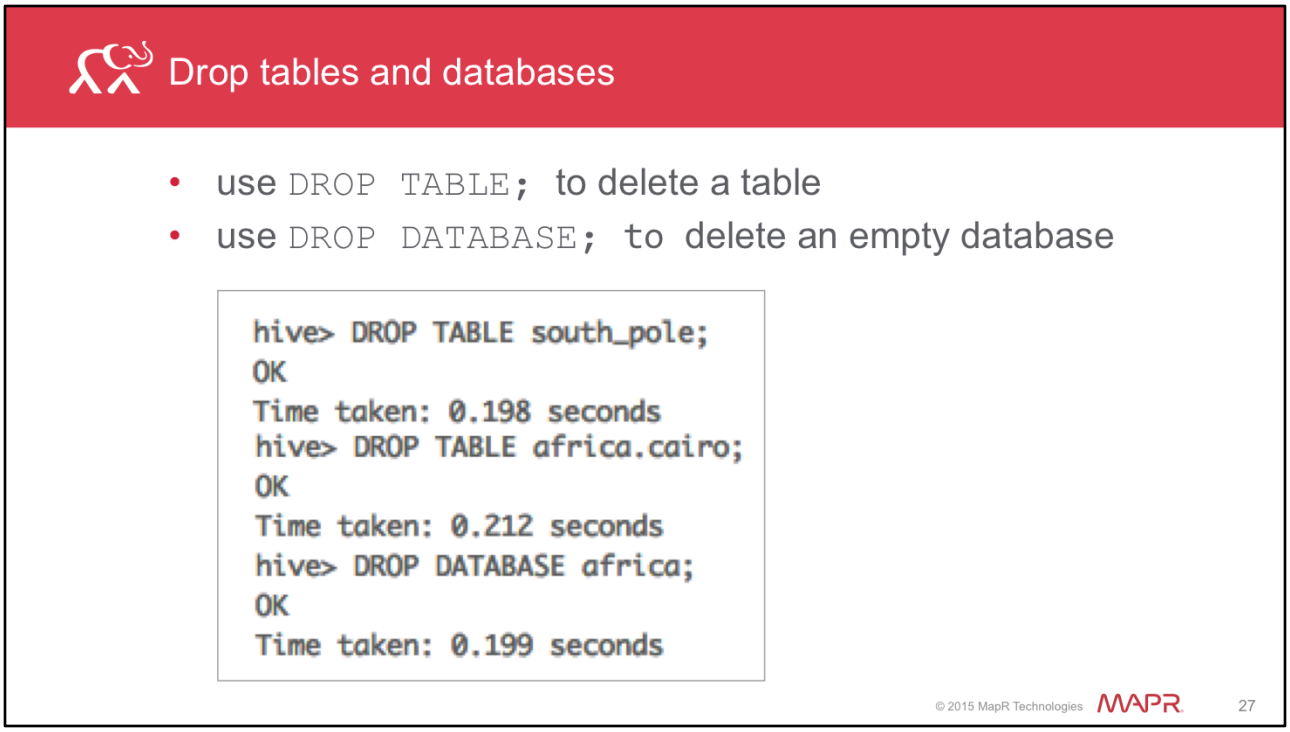
Since our windspeed table is partitioned by the station name, we need to load each partition separately. We specify which station corresponds to each partition when we load it.

The data in wind_adelaide.csv does not have a station column, but when we load the data like this, Hive populates the table with a date column and inserts “Adelaide” for all these data points. This example demonstrates loading two such partitions into the windspeed table.

Now you can try to load data into the tables you have created in Hive. Open your Lab Guide, and complete Lab 2.4. When you are finished, you can continue to explore the Hive command line by trying familiar SQL commands on your own, or you can continue with the lectures and labs.



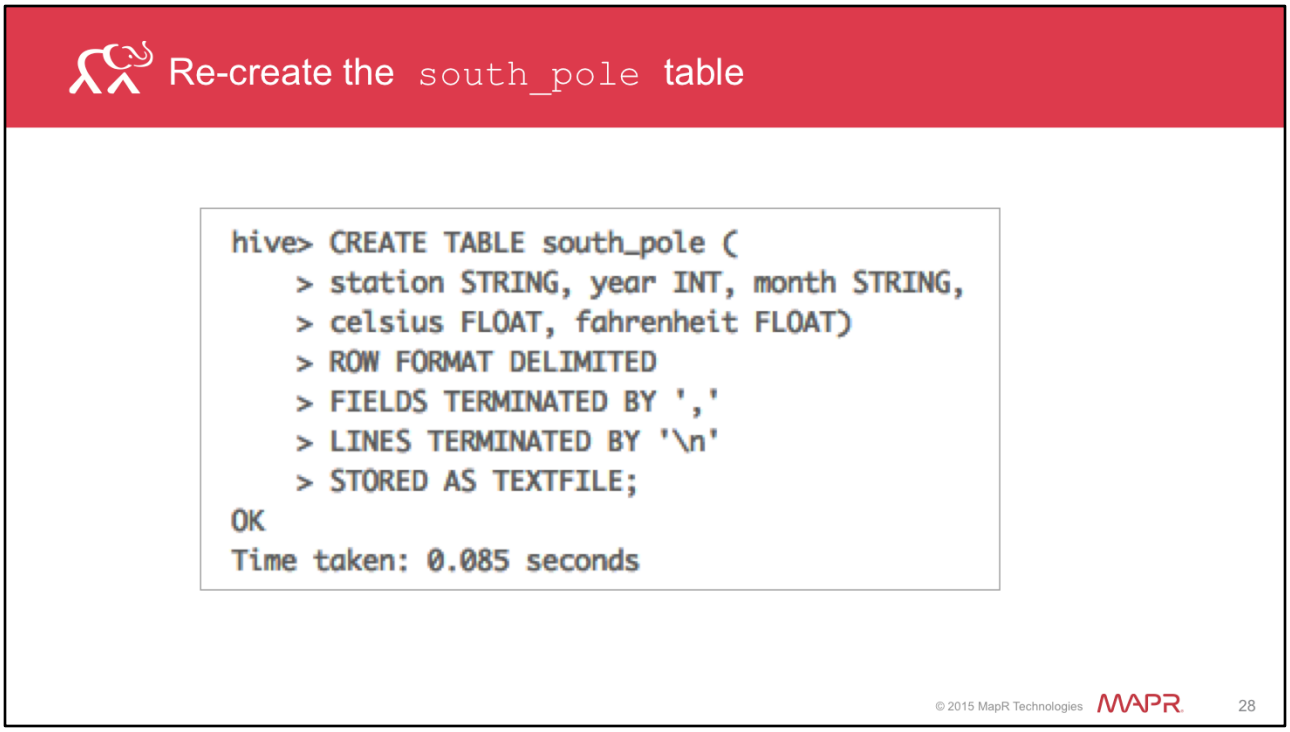
Finally, we'll learn about altering and dropping the tables that we have created.



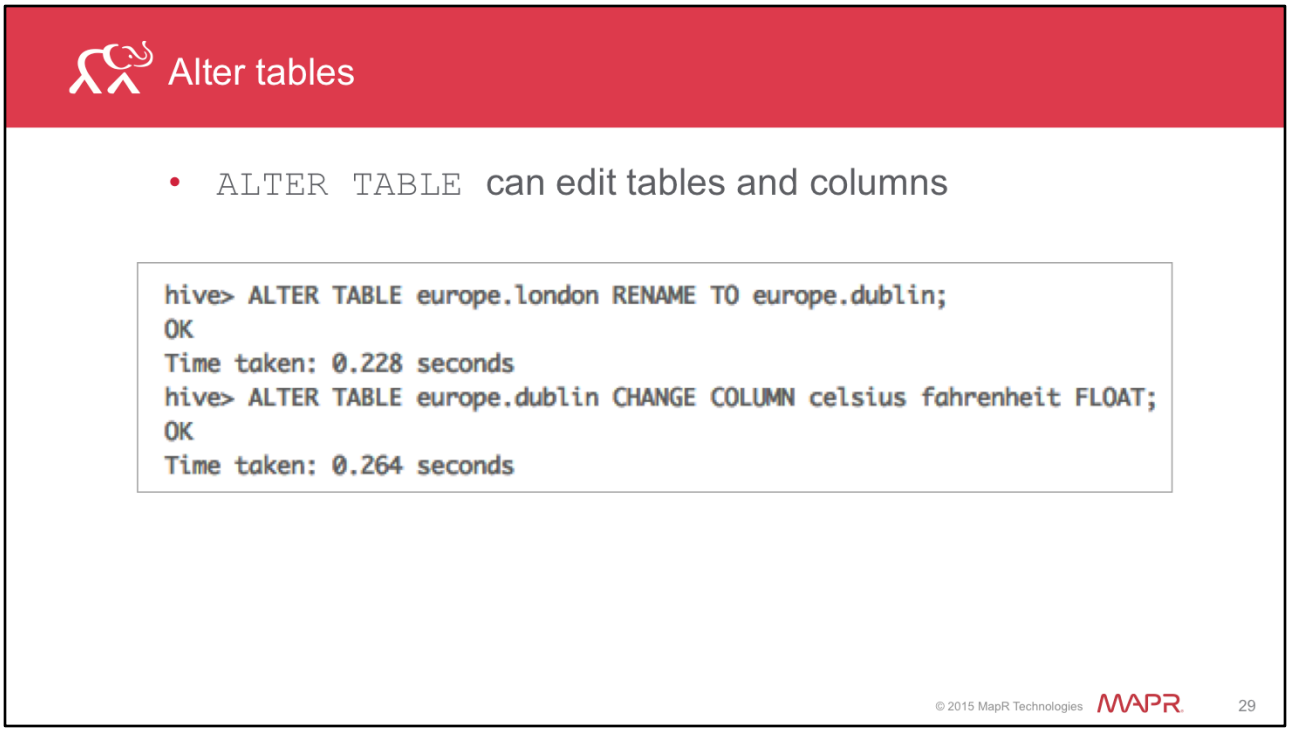
If you have created a table by mistake, or you just no longer need it, you can delete it using DROP.

Note that dropping a table does not necessarily delete your data files. Your data is typically stored on your HDFS, and will remain there unaltered. This is true even if you delete an external table.

You can also delete a database using DROP DATABASE, but only if there are no tables inside that database.



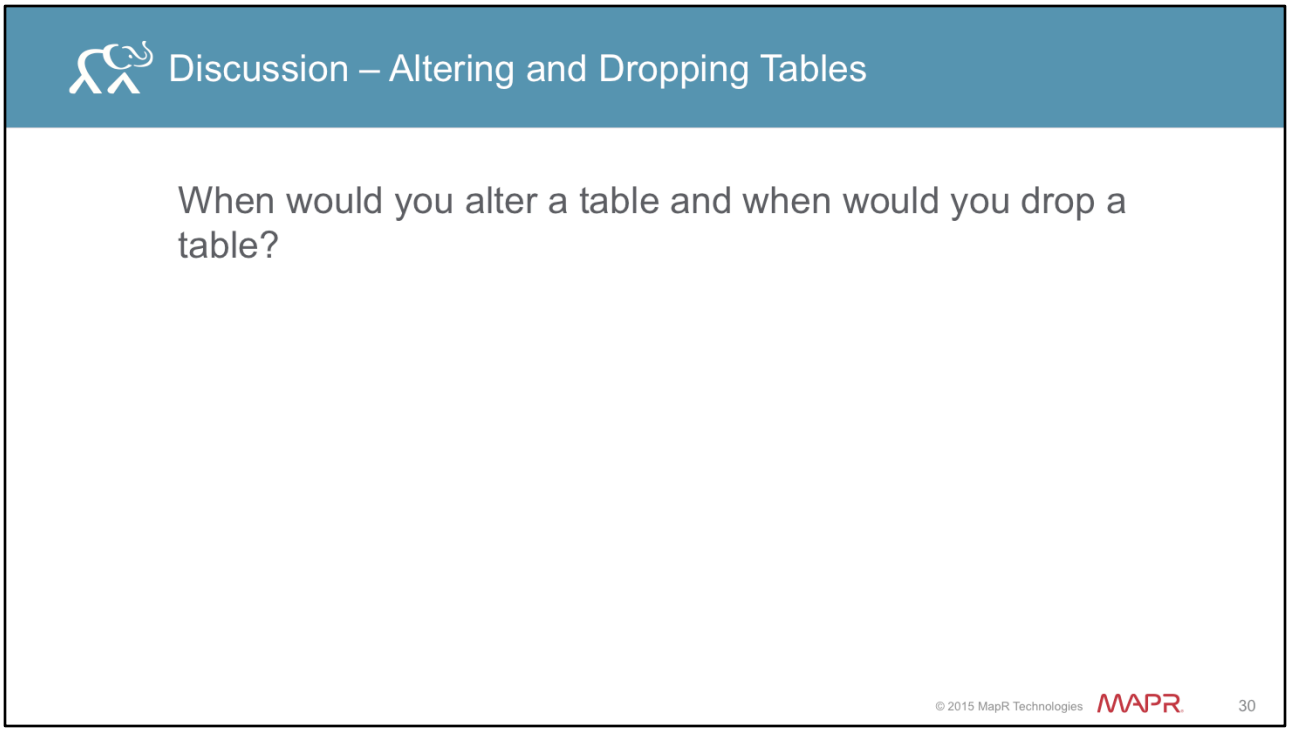
Now that we have dropped the south_pole table, we will need to recreate it if we want to use it again. Let's say we dropped it because it was created with the wrong data definition. Let's now re-create it, this time with different column names and data types.



What if you have created a table, but then realize you need to make a change to it? Rather than dropping the table and recreating it entirely, you can use ALTER TABLE. This is good for smaller changes, such as renaming a column or recasting a data type.

For example, let's say you realized that the london table actually contains data from dublin. You can rename this table.

You then realize that the celsius column should actually be fahrenheit. Remember to specify the data type when altering columns!



INSTRUCTORS: Discuss some of the different use cases of ALTER and DROP.

So far in these labs, you have created a database, created some tables, and loaded them with some data.

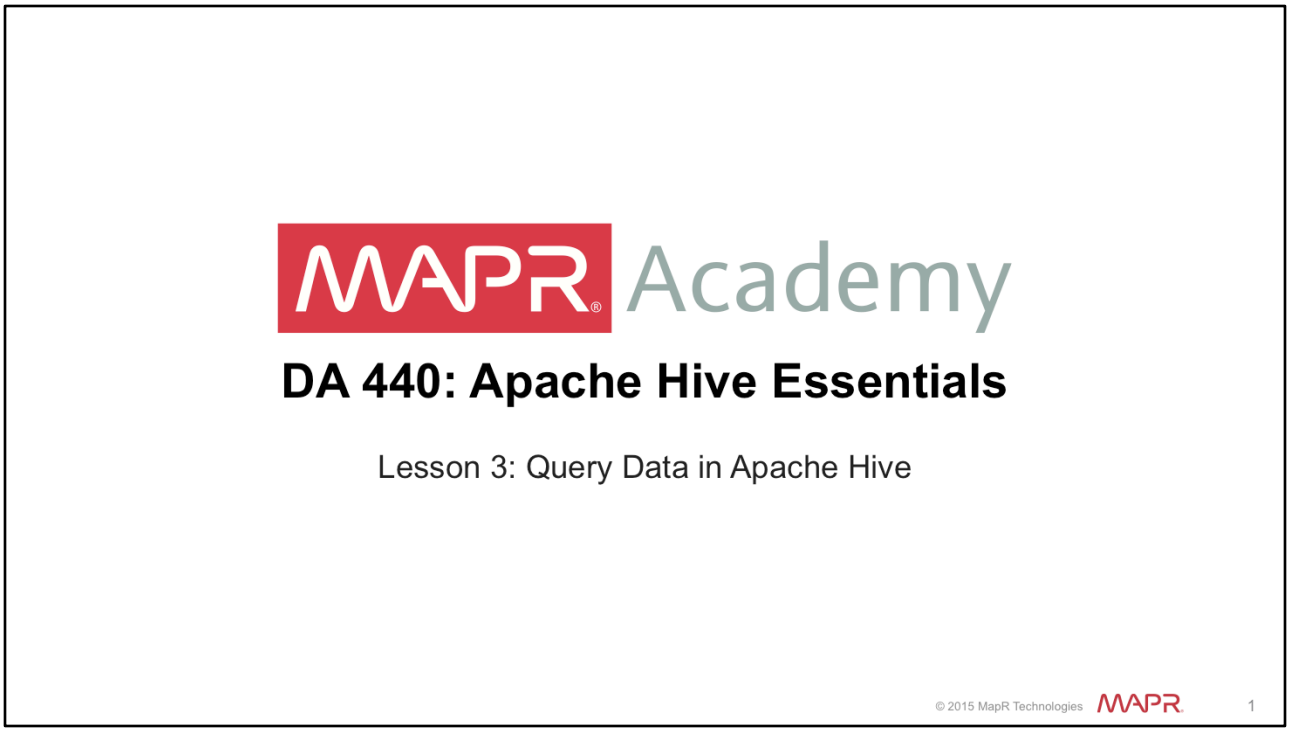
Use the remainder of this lab time to briefly explore these tables by following the directions Lab 2.5.

If you are already familiar with SQL, you might try querying this data using common SQL commands.

When doing the labs, it is recommended that you have a text editor open in order to write queries, rather than writing queries directly in the command line.

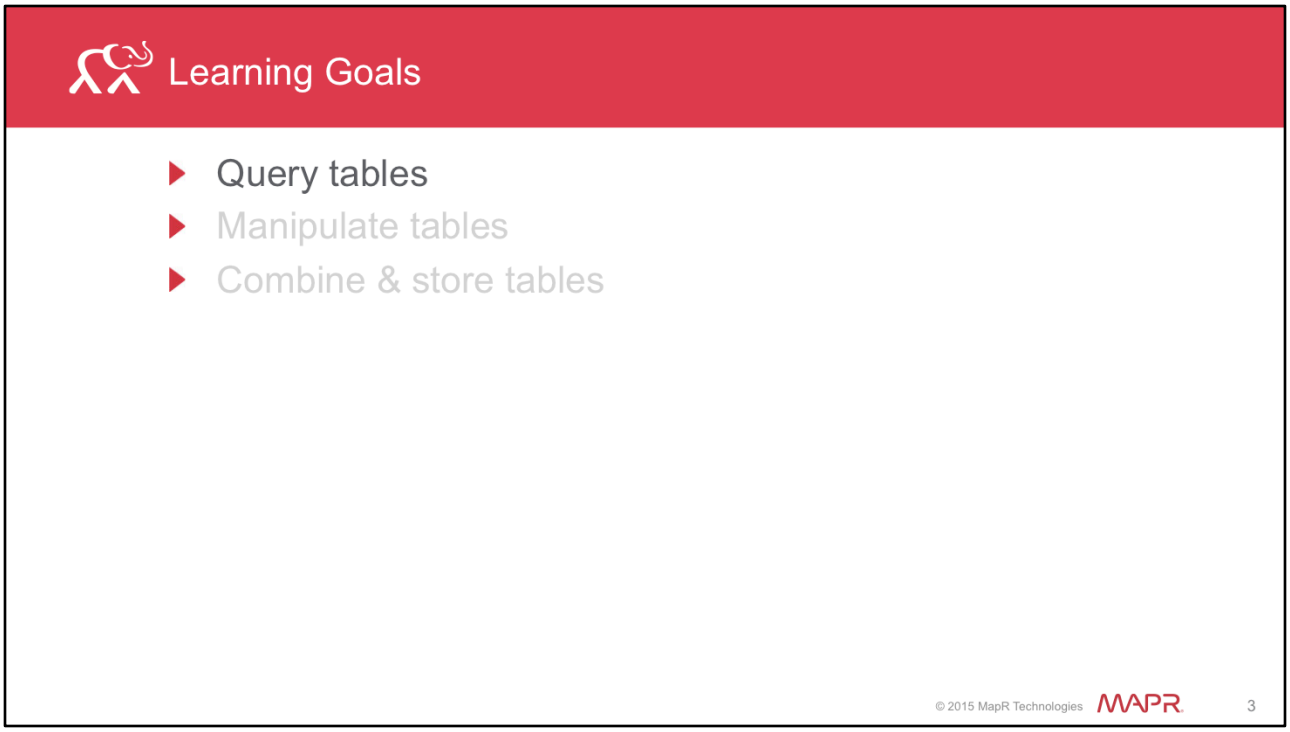
When you have finished Lesson 2 in the Lab Guide, you are ready to move on to Lesson 3.

In the next lesson, we'll learn about how to query and manipulate data in Hive.



Welcome DA 440 Lesson 3 – Query Data in Apache Hive.

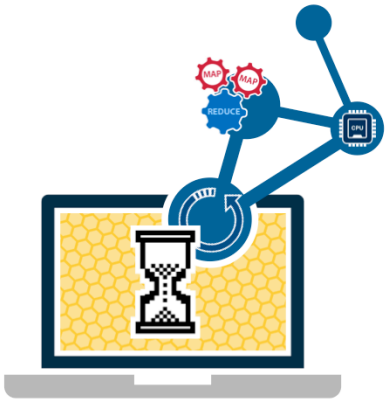
By the end of this lesson, you will know how to query data. You will also learn manipulate data with user defined functions. Finally, we will combine and save our data.



Let's start with some basic queries.

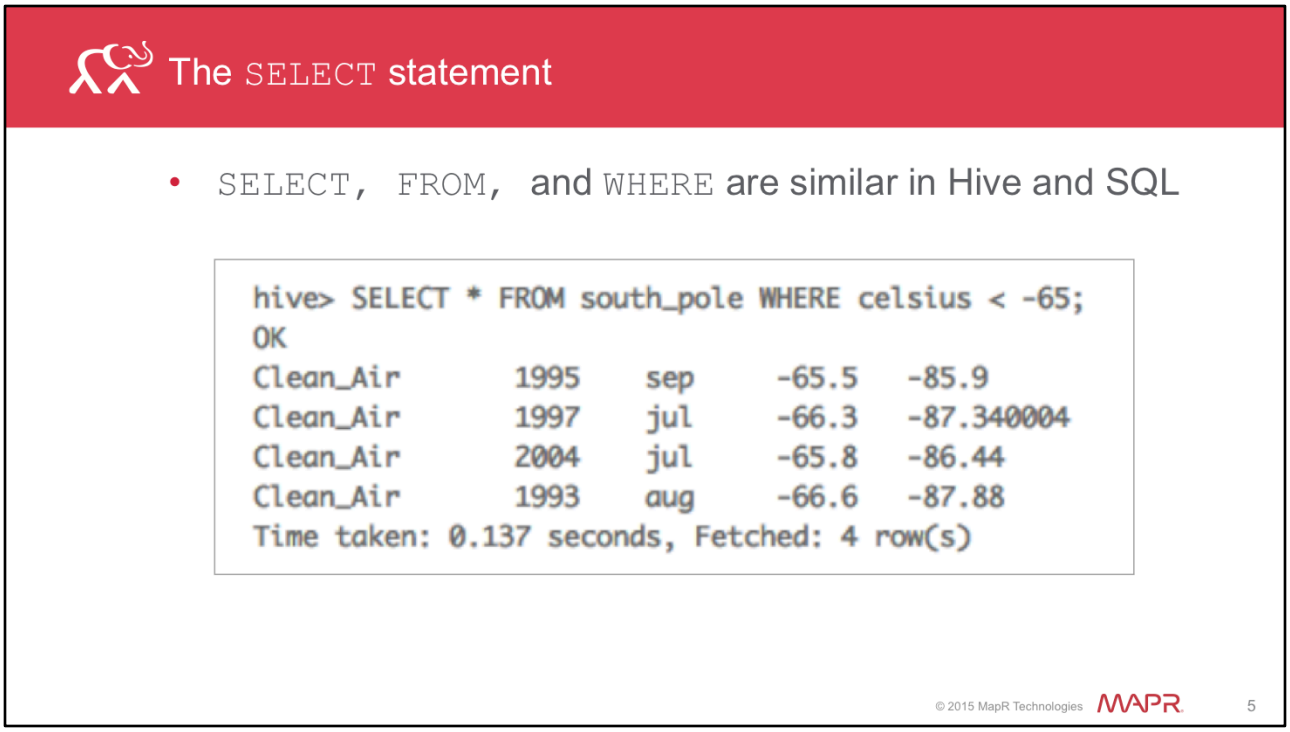
Hive and MapReduce

- Many Hive queries trigger MapReduce
- MapReduce
  - allows Hive to query data on HDFS
  - uses more resources than SQL or Drill



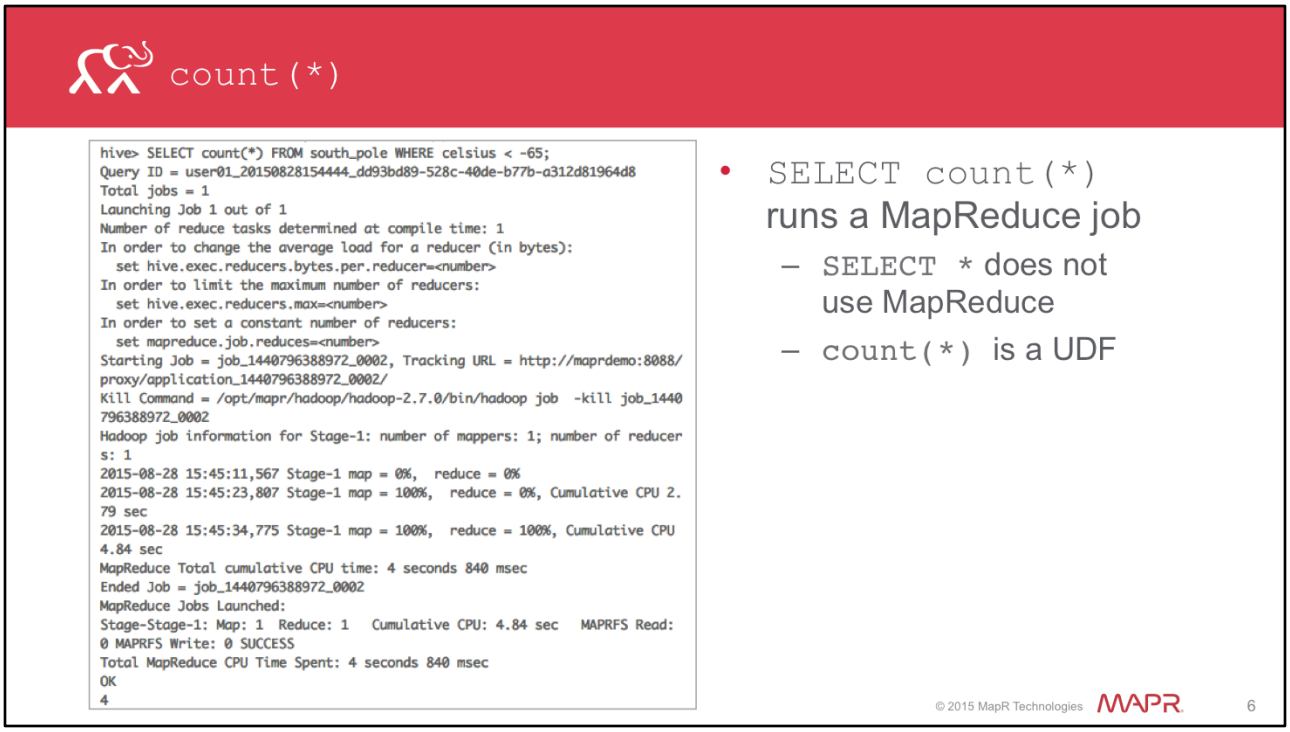
 MAPR 4

Before we begin querying data in Hive, recall how Hive differs from SQL.
Unlike SQL, Hive works on a Hadoop Distributed File System by converting SQL-like code into MapReduce code. This triggers a MapReduce job.
Since MapReduce jobs may be resource-intensive, it is important to try to optimize your queries using some of the techniques discussed in Lesson 1.
Users who are used to the speed of SQL sometimes worry their Hive jobs will not complete. Depending on the speed of your system, you may need to be patient as you wait for the MapReduce job to complete.



The SELECT statement in Hive works similarly to the SELECT statement in SQL, which you may already be familiar with.

For example, let's say we wanted to see all the data from the south pole table where the temperature is less than negative 65 degrees celsius. We can use the asterisk to select all the rows, then specify we are selecting FROM south_pole, then restrict the statement with the WHERE clause.

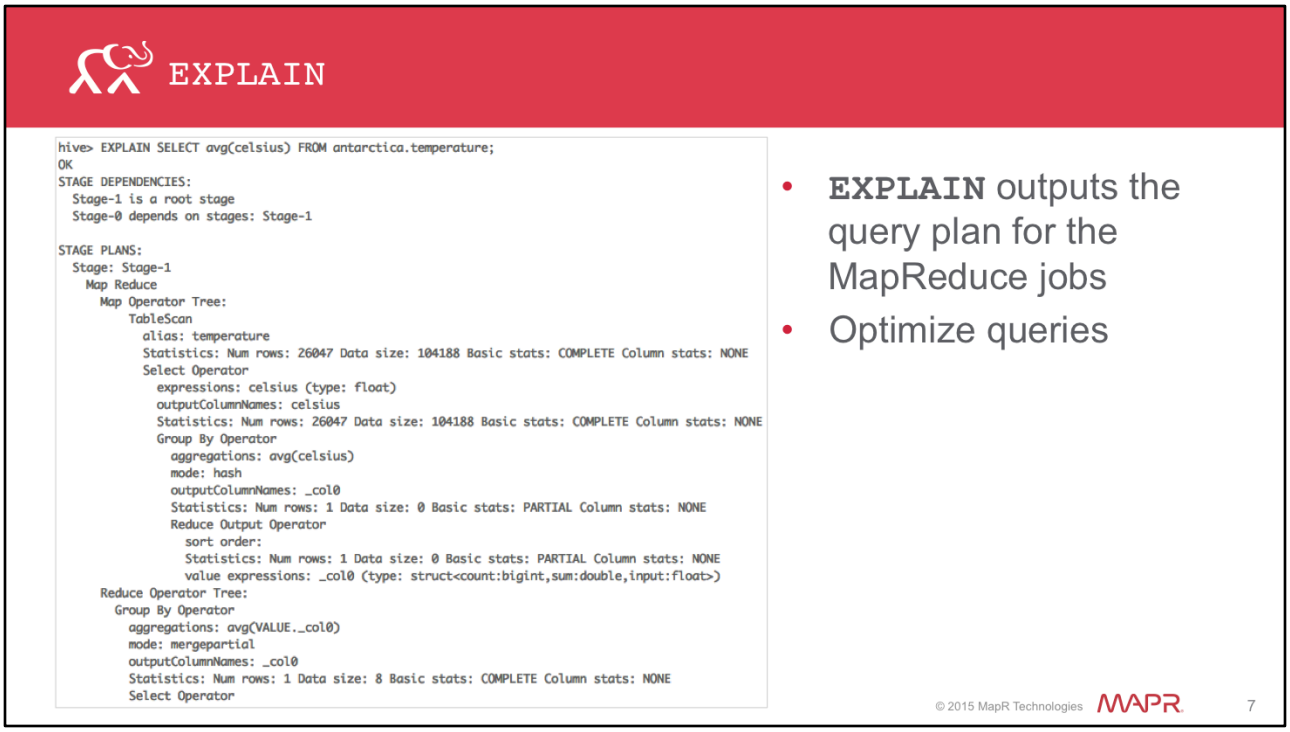


What if you wanted to know how many rows of data match your query, rather than printing all the rows? You might be familiar with SELECT count(*) from SQL

Unlike select * from the previous example, select count(*) triggers a MapReduce job. Here you can see an example of what that looks like in the Hive Command Line Interface.

The Hive CLI will start the job, display options for modifying, tracking, and killing the job, and update you on the progress of the map and reduce phases.

This example shows the output of a MapReduce job. In this example, the MapReduce job took 580 milliseconds, then returned a result of 172. So, there are 172 rows of data where the Celsius column contains a value that is less than negative 65 in our South Pole dataset.

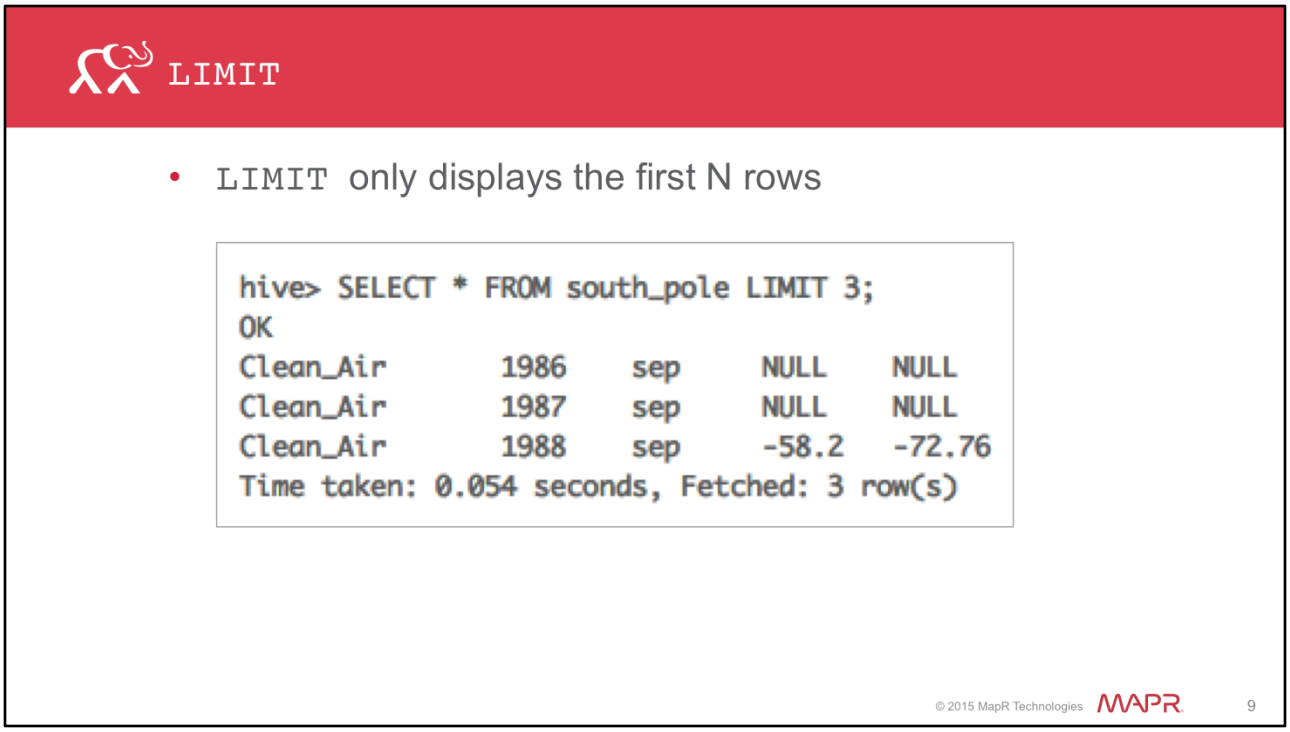


Using EXPLAIN on a Hive query will output the plan for converting HiveQL into a MapReduce job.

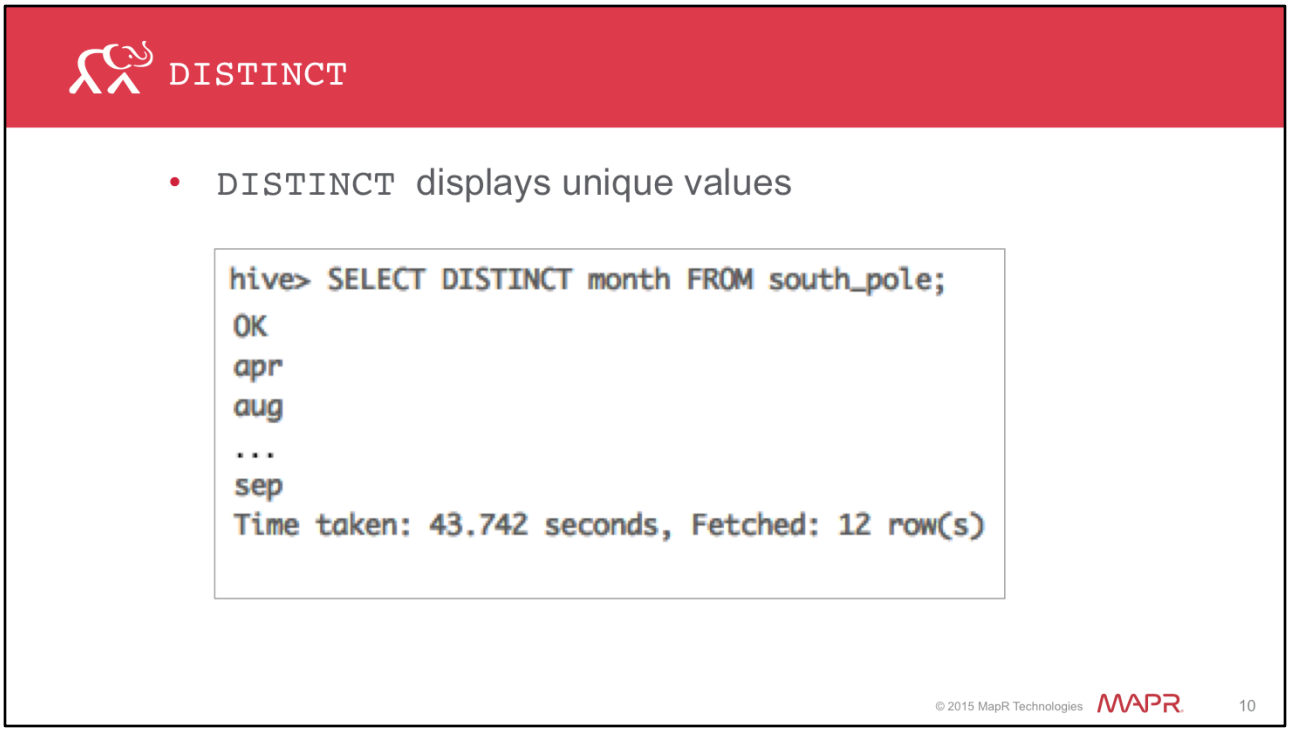
Advanced users of Hive may wish to see how their queries get converted into MapReduce code. This can help them identify inefficiencies in the query, and optimize performance.

This screenshot shows part of the output of EXPLAIN on a query.

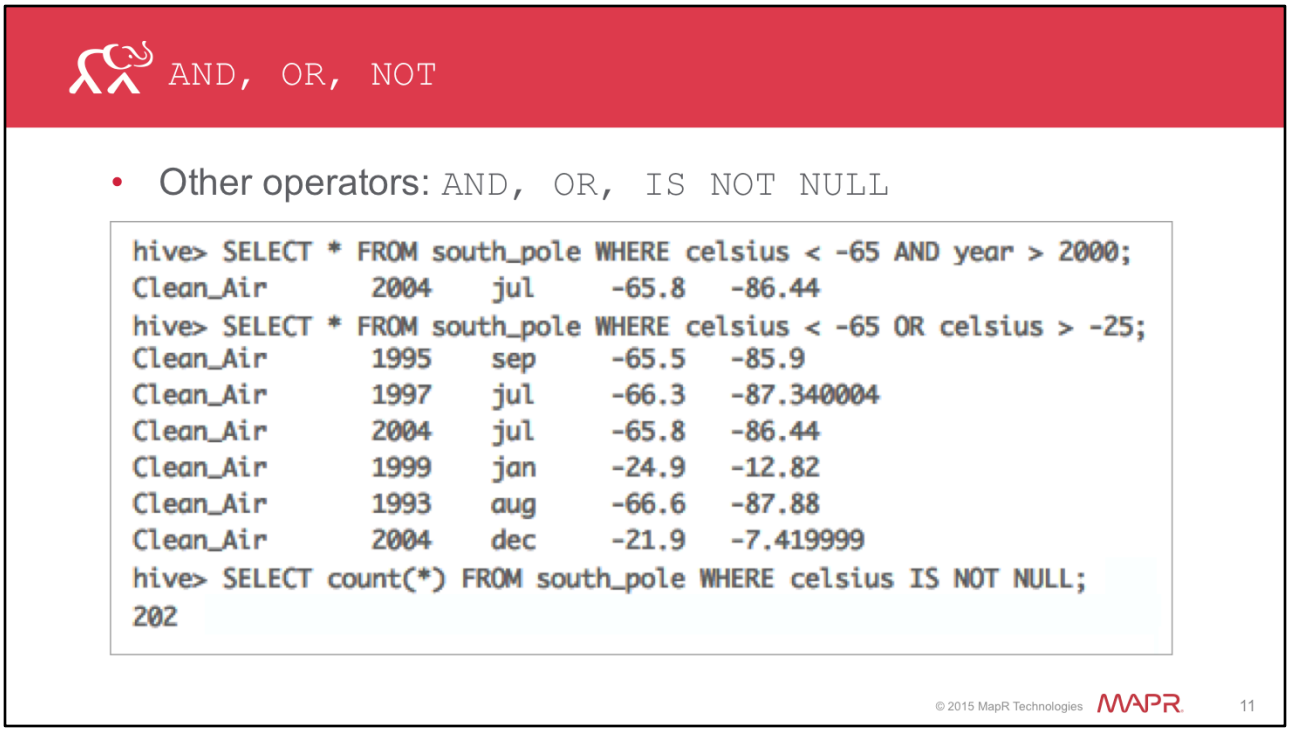
Now you can try your first queries. Complete Lab 3.1 in your Lab Guide before moving on.



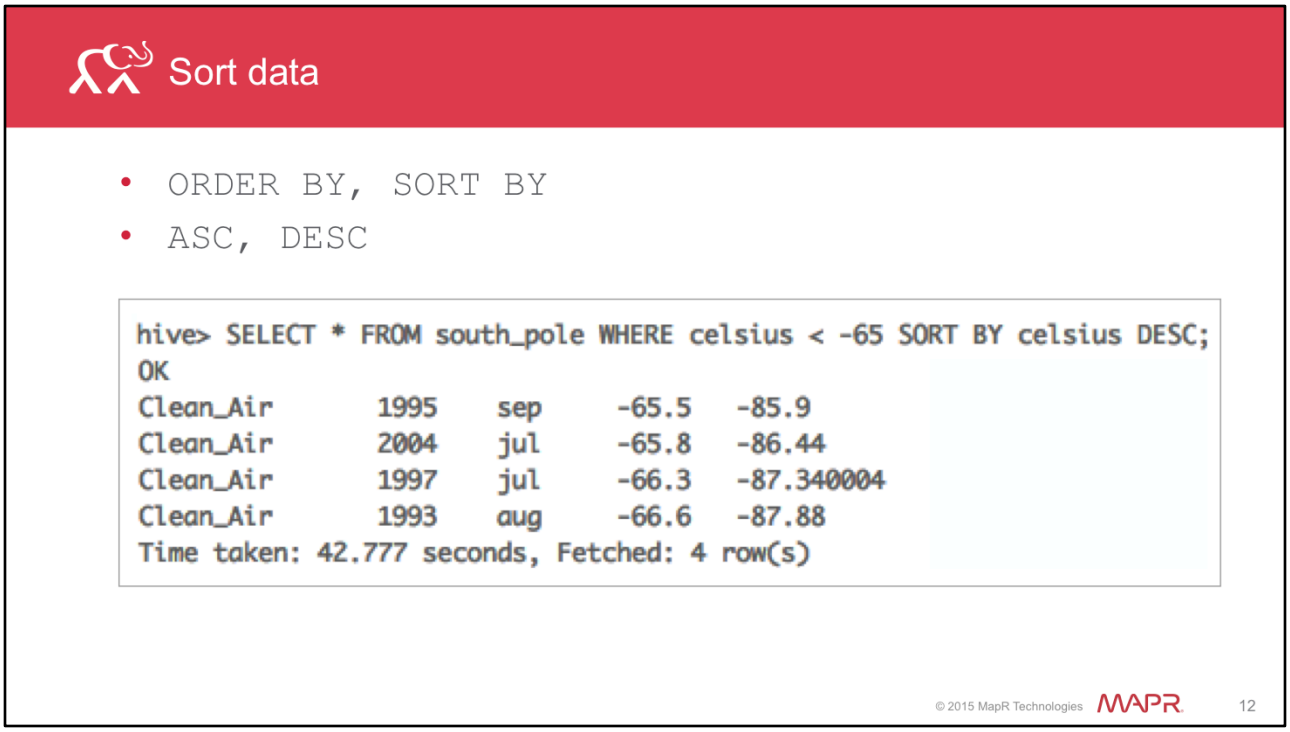
Tables in Hive often have a lot of data, since HDFS is designed to handle large quantities of data. If you use SELECT *, Hive will return the entire dataset, which may be millions or billions of rows. You can use LIMIT to return a smaller subset of rows. You can choose how many rows to display. This is especially useful if you are testing to see if a dataset loaded properly; the first 5 or 10 rows is usually sufficient for this task. In this example, we limited the display to 3 rows.



You can also use SELECT DISTINCT to see a specific subset of the data. For example, if you wanted to know which months you have loaded into your south pole table, you could use SELECT DISTINCT month FROM south_pole;



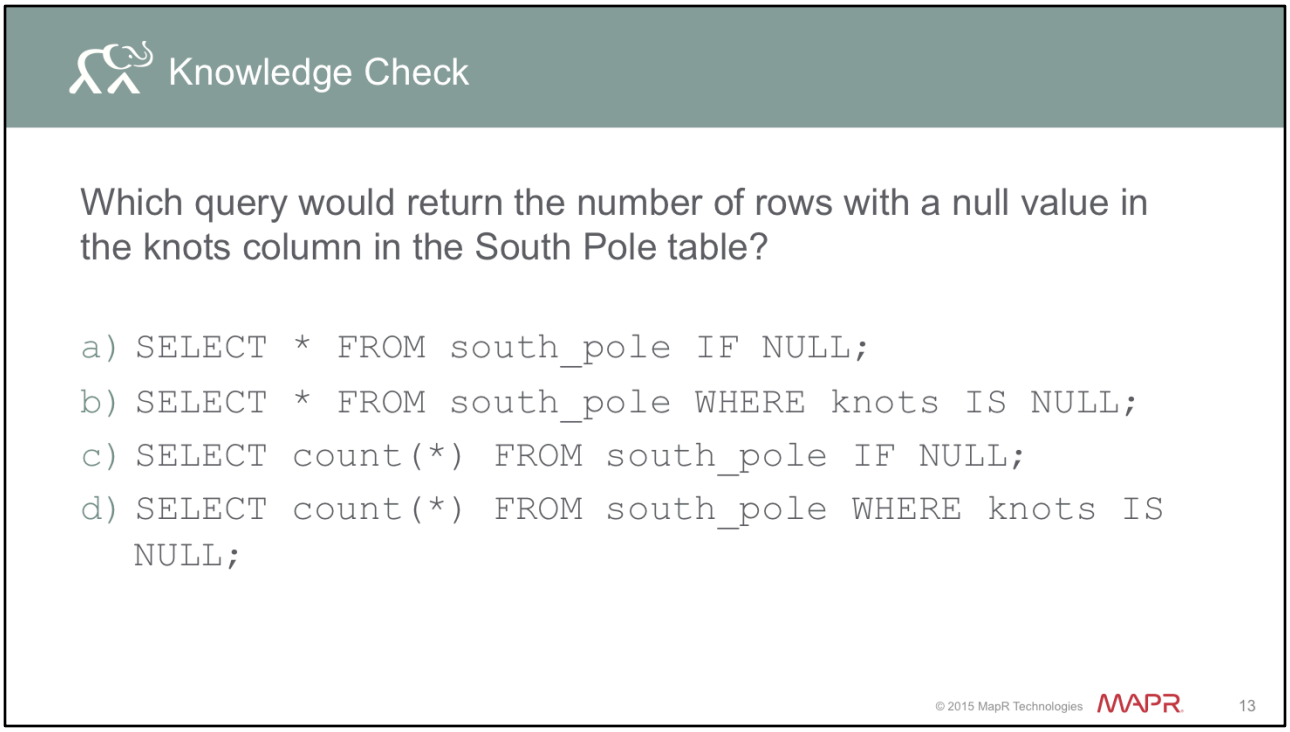
Other querying operators might be familiar from SQL.
For example, you might want to select all the data where the temperature is greater than negative 20 degrees and the year is more recent than 2000.
Alternatively, perhaps you only want extreme weather, regardless of whether it's hot or cold. You might select all the data where celsius is greater than thirty OR celsius is less than zero.
You can also use IS NOT NULL in a where clause. For example, if your dataset has many missing values, you could query all the data where the entry IS NOT NULL



ORDER BY performs a total ordering of the data. You can optionally add the parameters ASC or DESC to sort in ascending or descending order. By default, all sorts are in ascending order.

ORDER BY sorts all of the data. This can result in extremely long run times.
SORT BY works on smaller chunks of the data, and therefore may be better for faster results if you do not need the entire table sorted.

In this example, we selected everything from the south pole table where the celsius column was less than negative 65 degrees, and then sorted it in descending order.

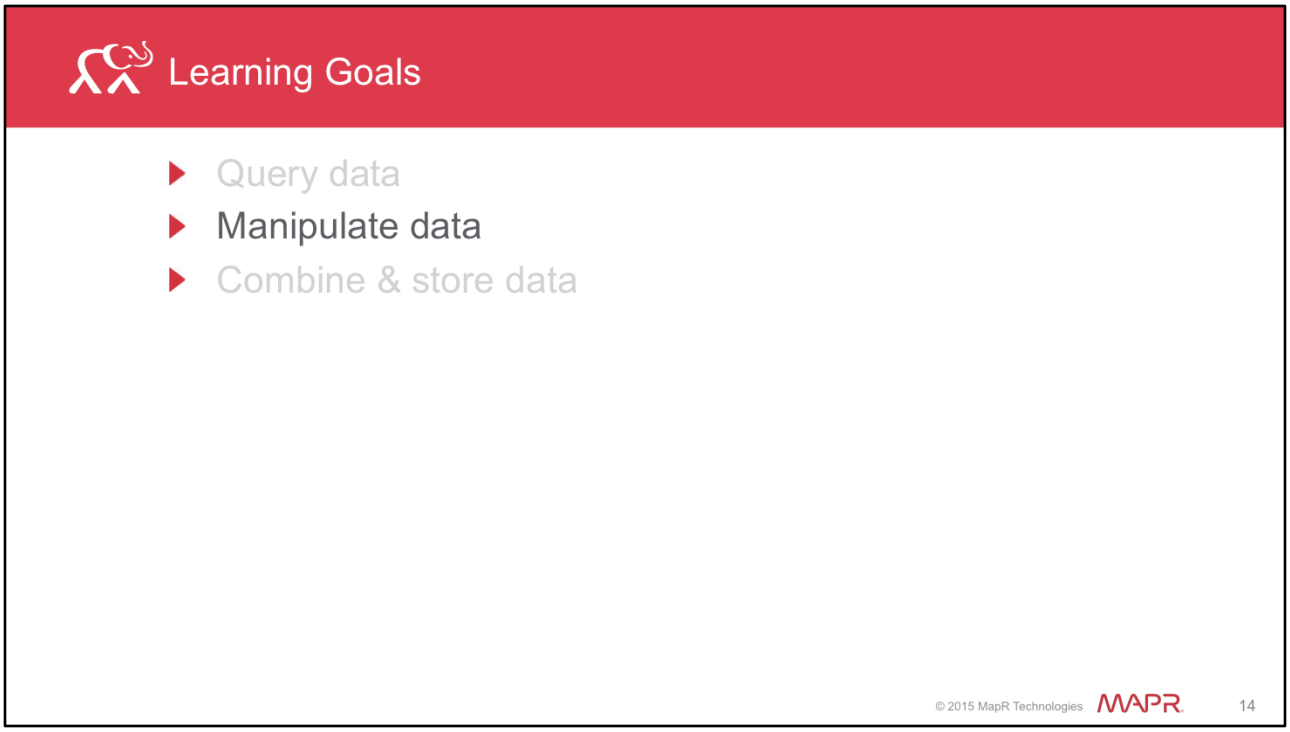


Answer: D

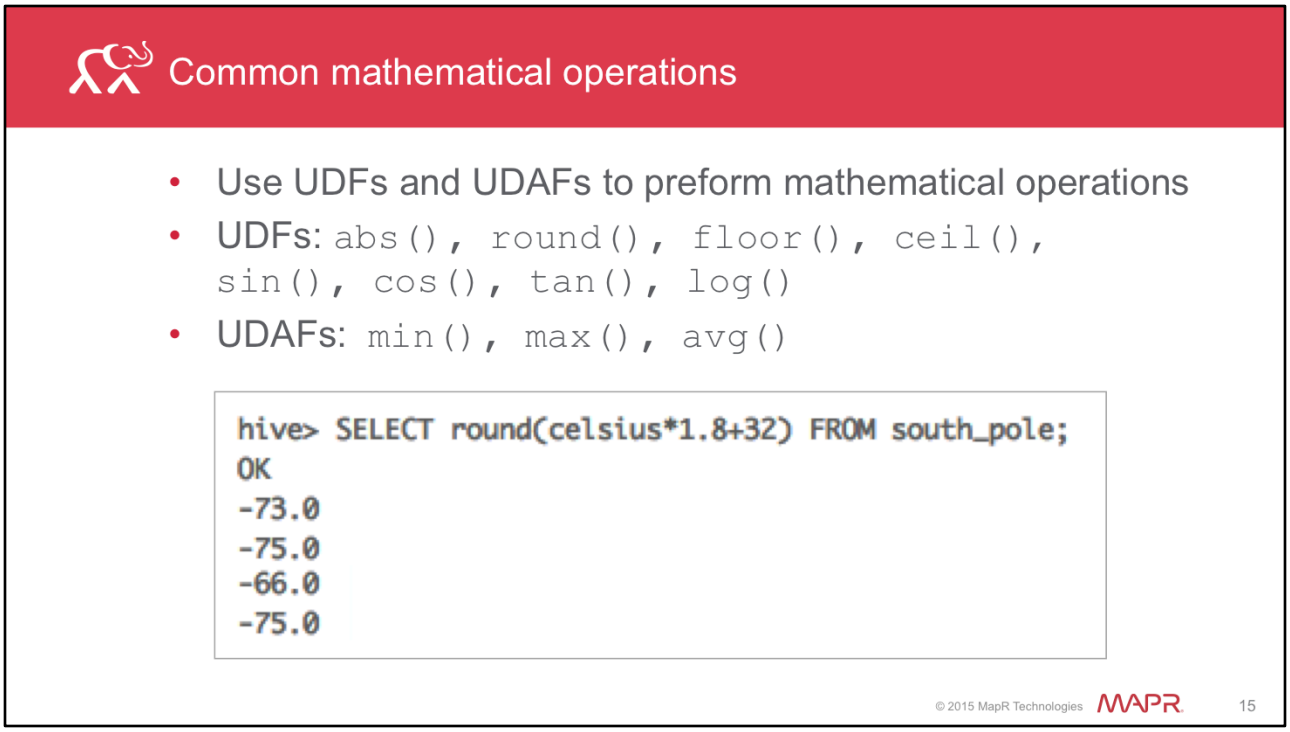
INSTRUCTOR: Ask your students what sort of results each of these queries would return. Alternatively, rather than provide these options, consider collaboratively writing this (and/or other queries) with your students. Discuss a variety of queries you might want to write and try to write them together as a class.

DISCUSS: When have you used DISTINCT, NOT, OR, AND, etc.? What are the use cases of these operators in your own experience as a data scientist? How are these similar or different from what you can do in SQL, Hive, or other languages or programs like Drill?

DISCUSS: Recall the version chart from Lesson 1. Are there issues with older versions and queries you might want to use?



In this next section, we'll learn about a few common user defined functions.

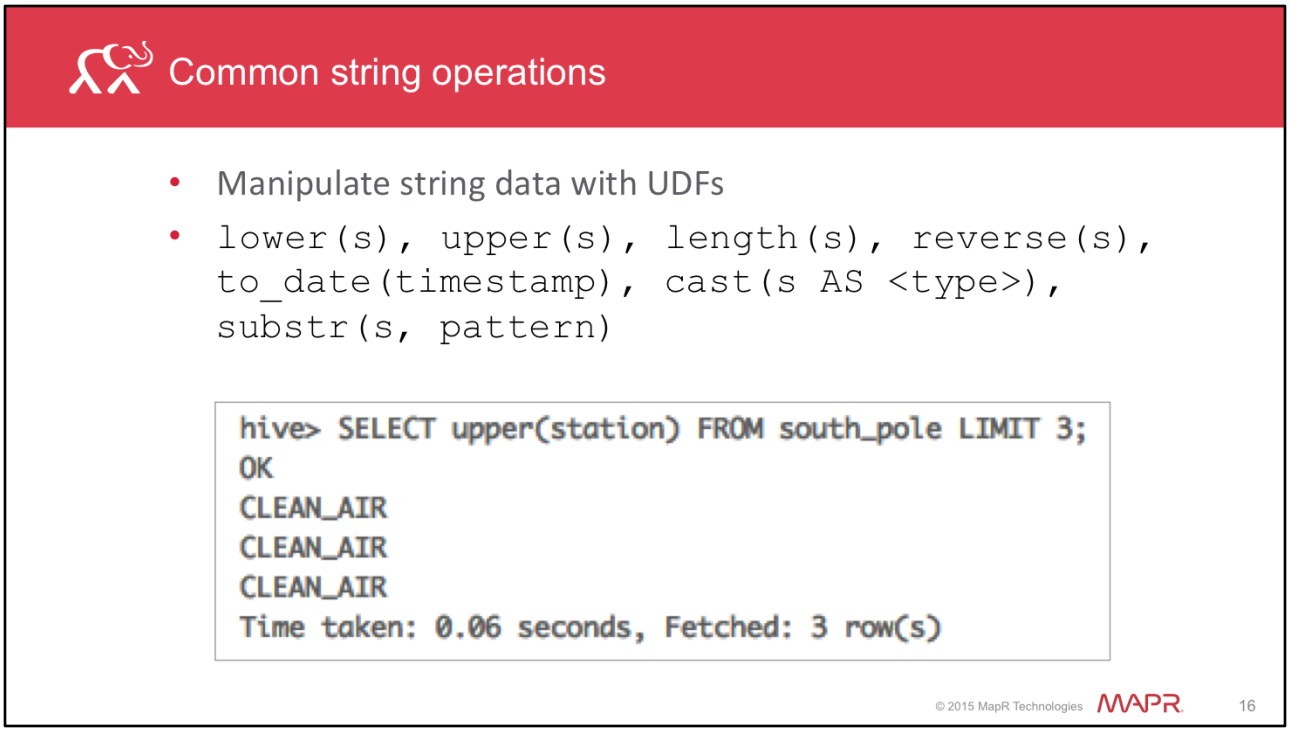


You can use arithmetic operators when querying data. For example, the south pole table only has data in the Celsius column. If you'd like to see this data in degrees Fahrenheit, you can convert it using multiplication and addition, as shown in this example here. We also use the round() UDF.

In addition to arithmetic operators, you can use user defined functions and user defined aggregate functions to preform many other common mathematical operations on numerical data.

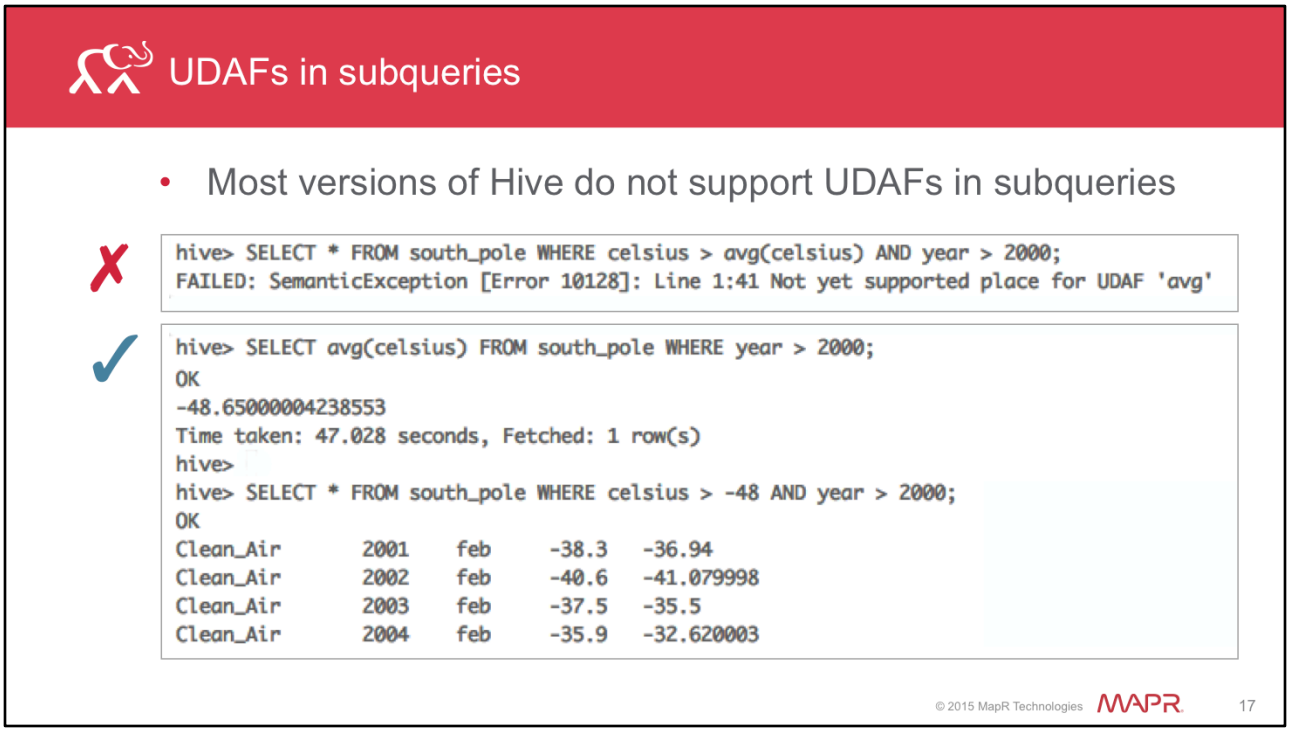
INSTRUCTORS: Now is a great time to demo some of these functions. Try these queries:

```
SELECT min(celsius), max(celsius) FROM south_pole;
SELECT avg(celsius) FROM south_pole WHERE year > 2000;
SELECT round(fahrenheit) FROM south_pole LIMIT 10;
```



In addition to numerical operations, Hive supports many string operations. The upper() function is demonstrated in the screenshot shown. This function takes the station name string, which was saved in mixed-case, and converts everything to all caps. The lower() function would do the opposite.

A short list of some common string UDFs are shown here. However, this list is not exhaustive, and you should consult your Hive documentation for a complete list.



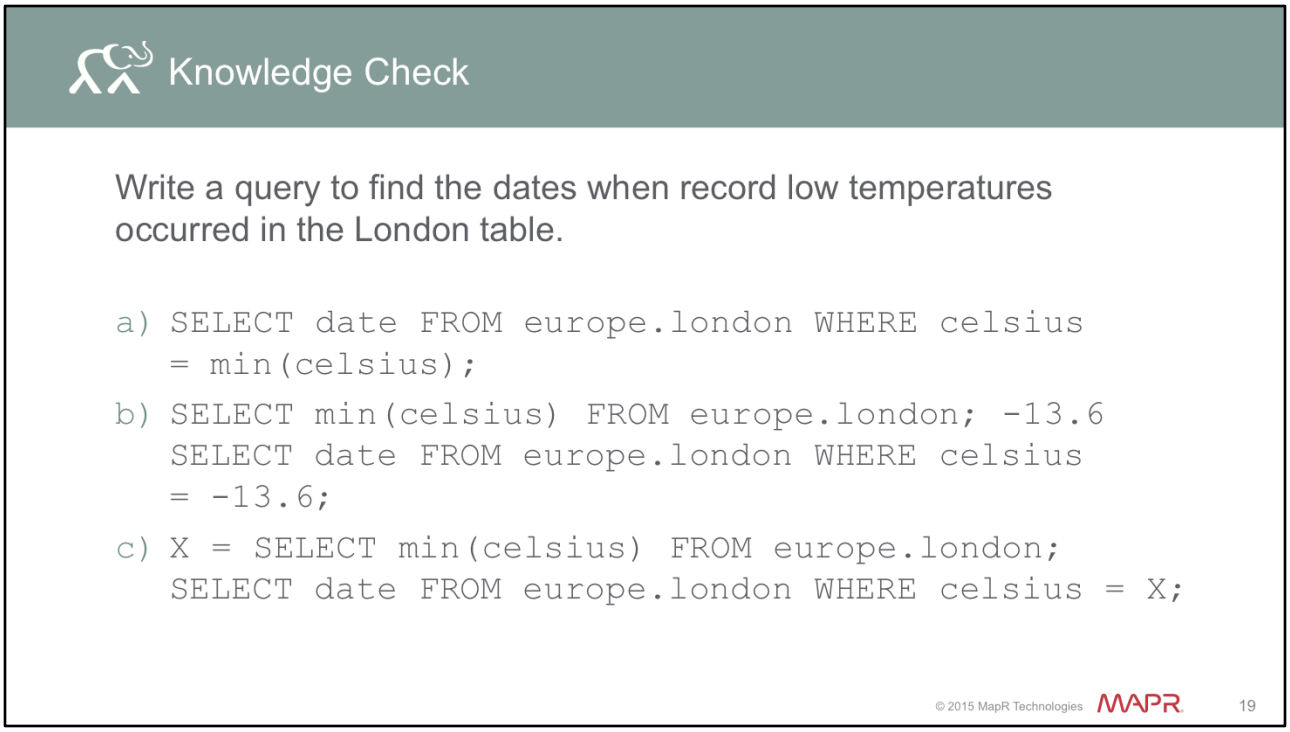
It is important to remember that Hive is not ANSI-SQL compliant. Although the HQL queries you have seen so far are similar to SQL, there are some important differences.

For example, currently Hive does not support UDFs inside sub queries, although it is possible that future versions of Hive may support this feature.

This means that if you wanted to select all the data from your south pole weather station where the temperature was greater than average, as in the first query shown, you could not.

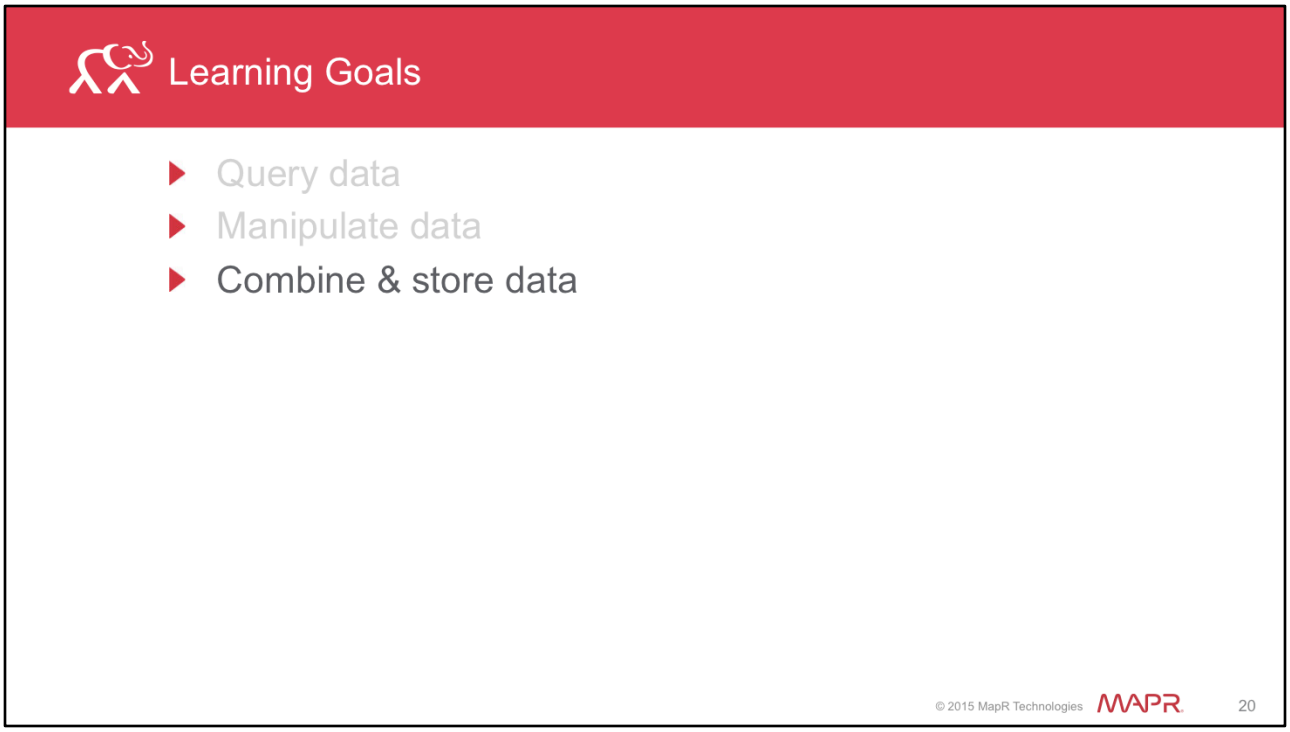
If you wanted to preform a similar query, you would first have to determine what the average temperature in the dataset was, and then write a query selecting data which is greater than that number. In this example, we find that the average is about -48. We can then query the data again, based on this number.

Now you can try more queries. Complete Lab 3.2 in your Lab Guide before moving on. Feel free to explore some of the other user defined functions we have discussed.

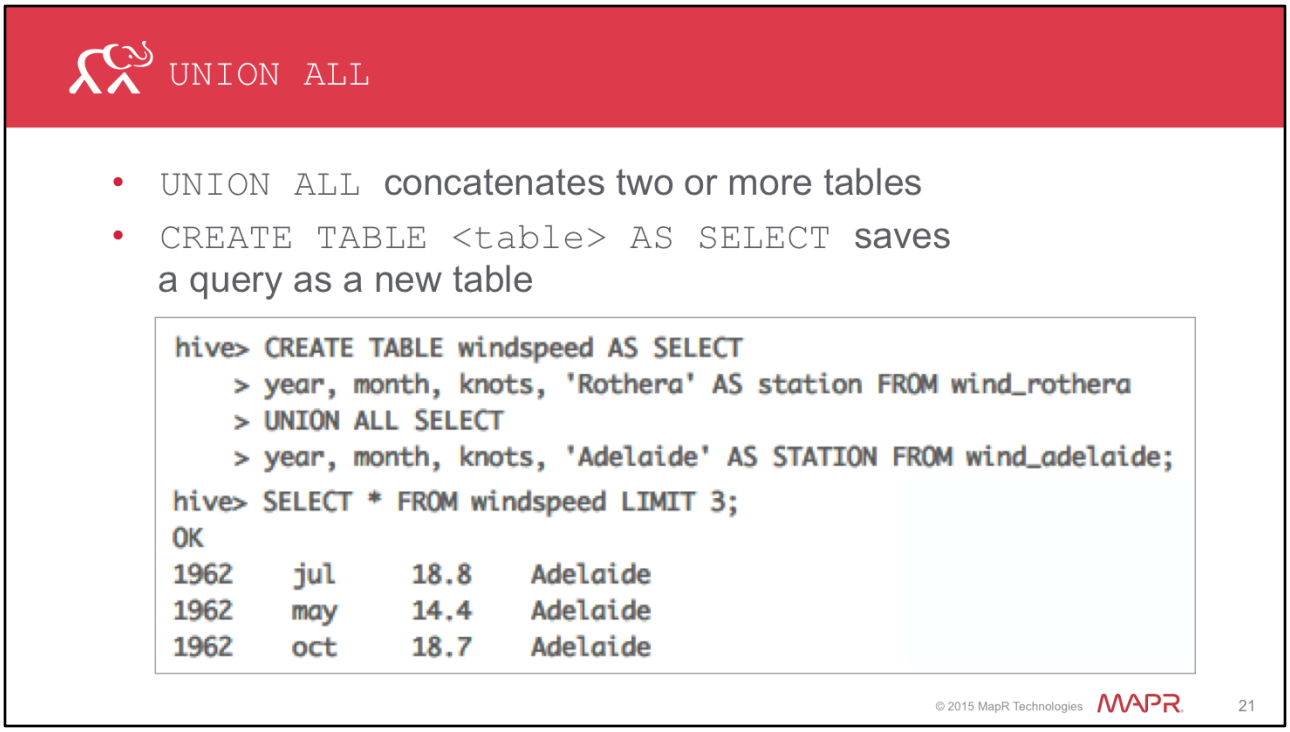


Answer: B

INSTRUCTOR: Rather than provide these options, collaboratively write this (and/or other queries) with your students. Discuss a variety of queries you might want to write and try to write them together as a class.



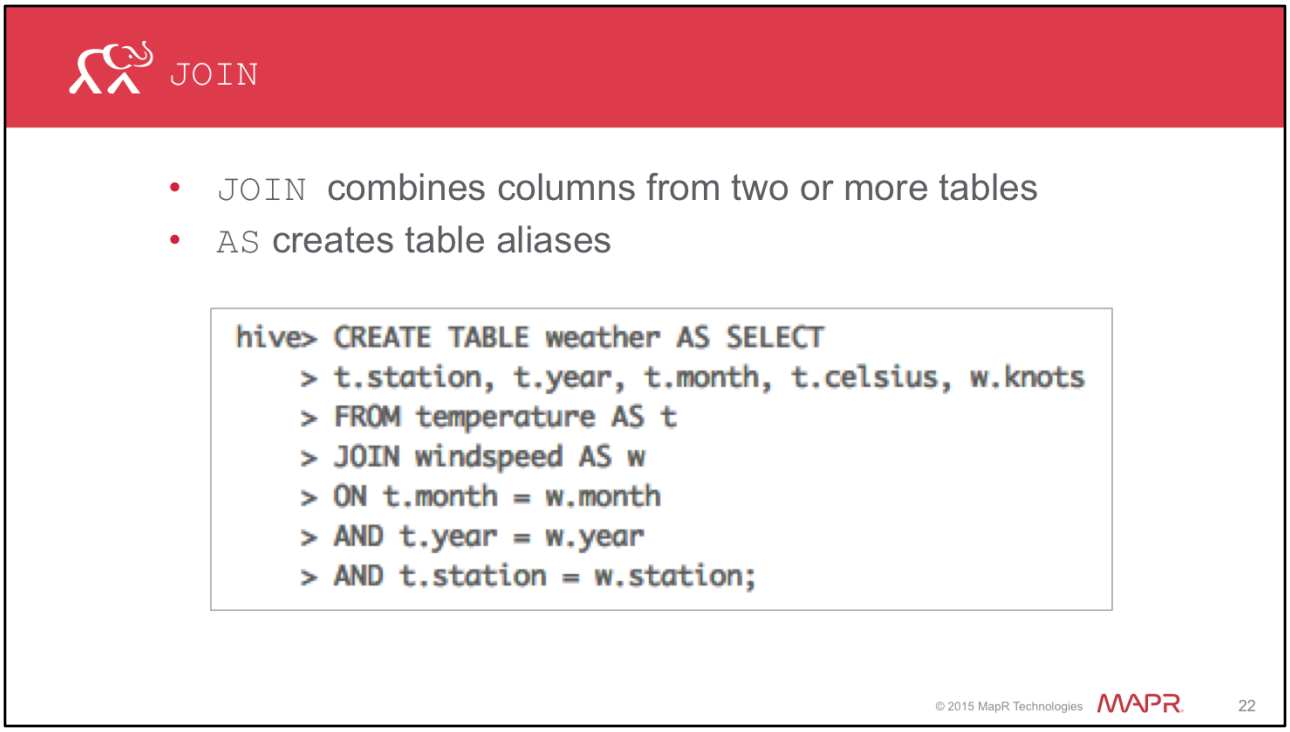
In this last section, we'll learn how to combine datasets and save them as new tables or files



Union all allows you to concatenate two datasets. For example, we can create a new table, windspeed, using AS SELECT.
In this table, we will combine some data from several different weather stations.

In order to keep the data in this new table organized, we will add a new column called "station", which we will populate with a STRING so we know which station the data comes from. This new column did not exist in either original table.

the UNION ALL command then concatenates the data from Rothera with the data from Adelaide. In this example, a new column, station name, is created with a string we define.



JOIN add columns of data from two or more tables by matching values which are common to each table.

The temperature and windspeed tables each contain data from multiple weather stations.

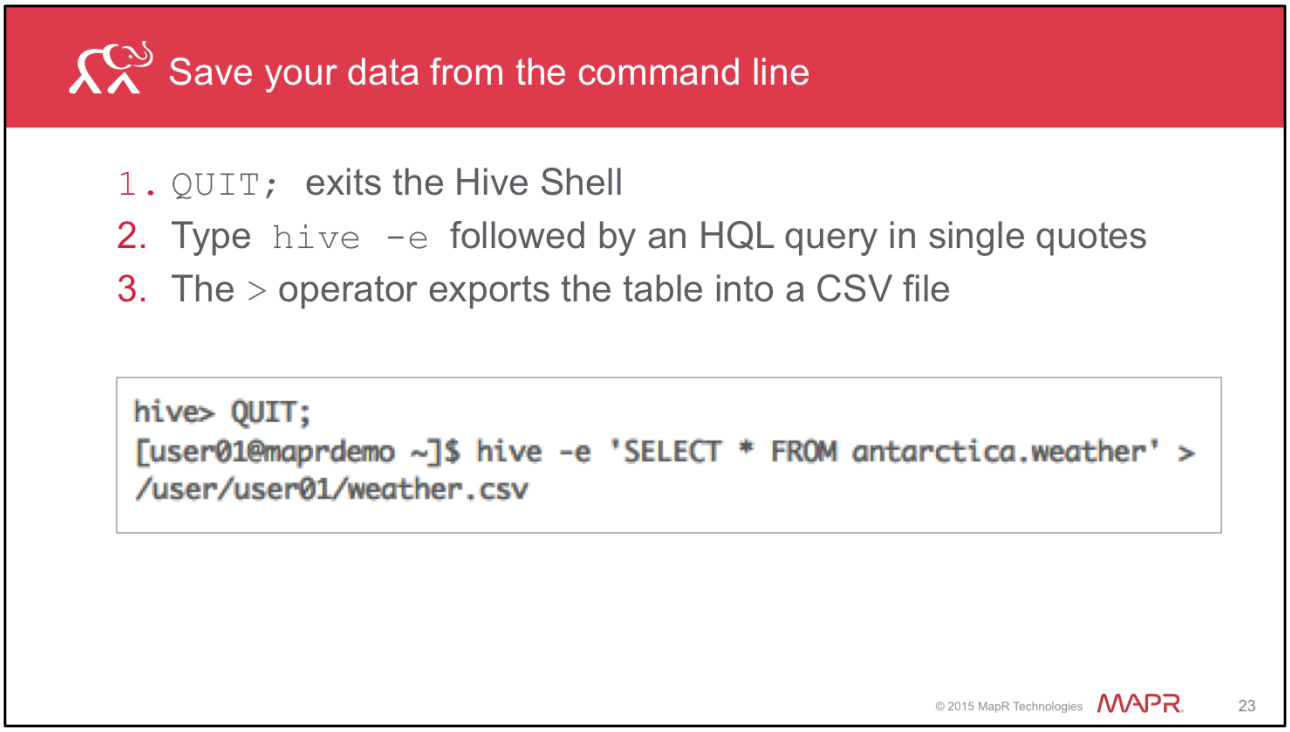
Let's create a new table called weather. We'll join these two tables on the date and station name. This way, both windspeed and temperature from the same station on the same month will be in the same row.

We'll abbreviate the names of our temperature table AS t and our windspeed table AS w. This abbreviation will make using dot notation easier.

We want to select the station, date, and celsius column from the temperature table, and add the knots column from the windspeed table.

When we join the tables, we have to specify which columns of data should match. We join ON the date column from both temperature and windspeed, AND we also join on the station name from each table.

With this new table, we can make some much more interesting queries. For example, we can find out what the windspeed was on the days with the record lows.



After you have manipulated data, you might want to save it in a file. You can save your data as a file from the bash command line interface.

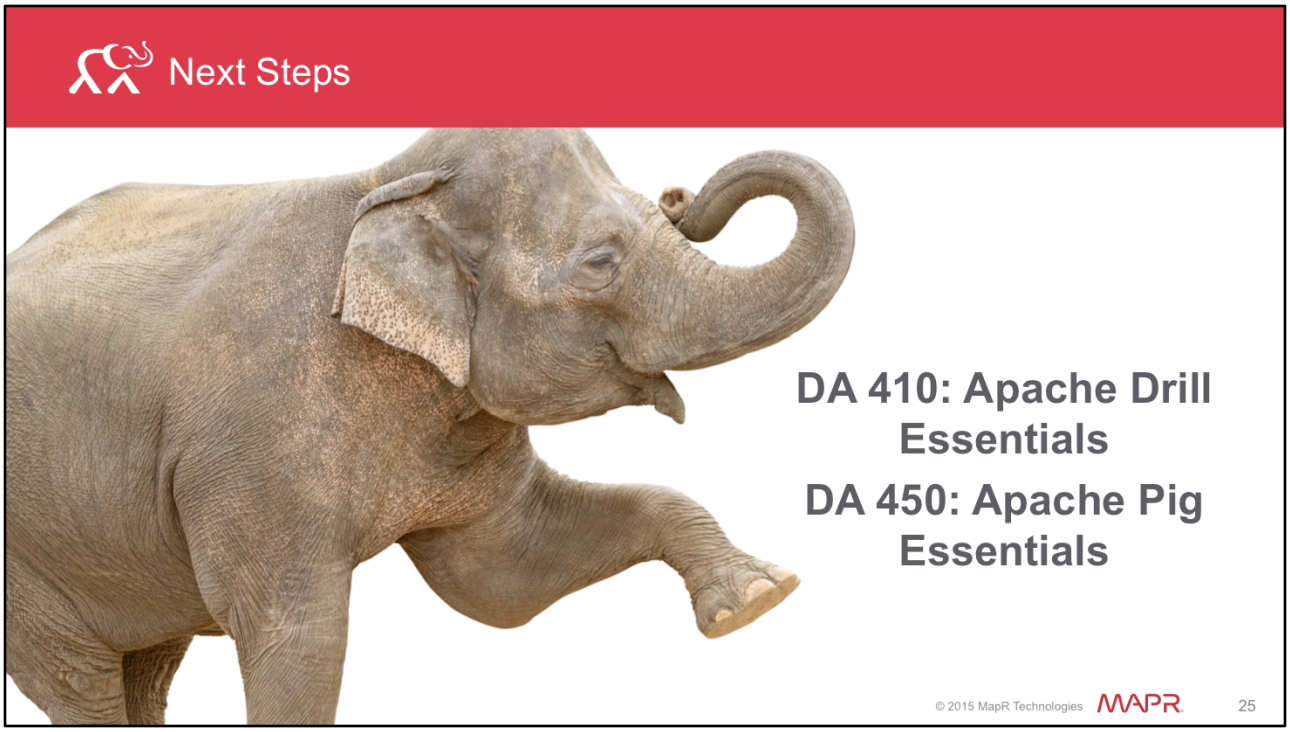
To use hive from the bash command line, exit the hive command line interfact with QUIT. Then, type hive dash e and select everything from the weather table in the antarctica database. Place this hive query inside single quotes.

using the arrow operator, we'll save these queries to a new csv file called weather.csv. You can then verify your file was created by using LS, and view your new file directly from the bash shell using an editor such as VI.

You can now use this csv file to load a new table if you would like. This will save you from having to join or create tables from the datasets you loaded previously.

Complete Lab 3.3 in your Lab Guide before moving on. When you are finished, you may explore the weather data set as a data analyst. Be creative and try to come up with your own research questions and queries.

Take a moment to make sure you understand everything from the labs and lectures. Use this time to ask your instructor any clarifying questions.



Thank you! You have now completed DA440: Apache Hive Essentials. Visit the MapR Academy website for more courses, including DA410: Drill Essentials or DA450: Pig Essentials