MAPR Academy

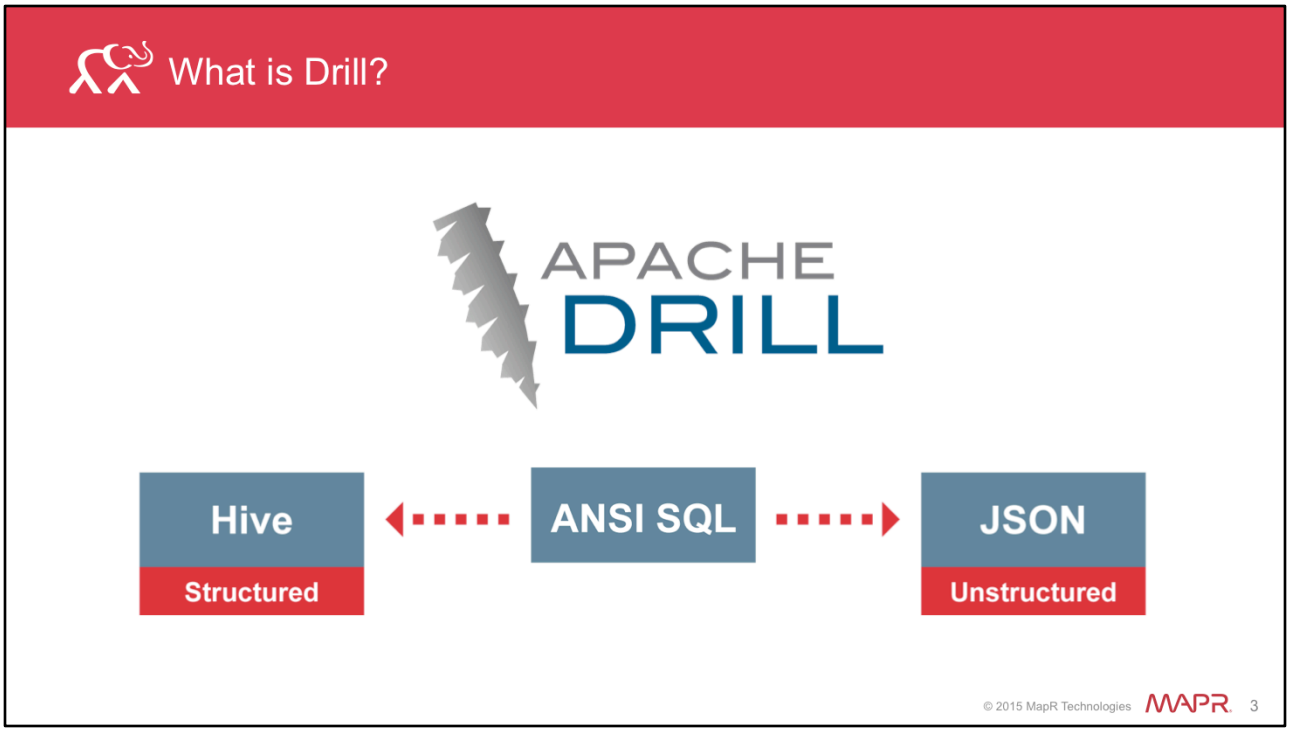
Data Analysis with Apache Drill

Lesson 1: SQL Queries with Drill

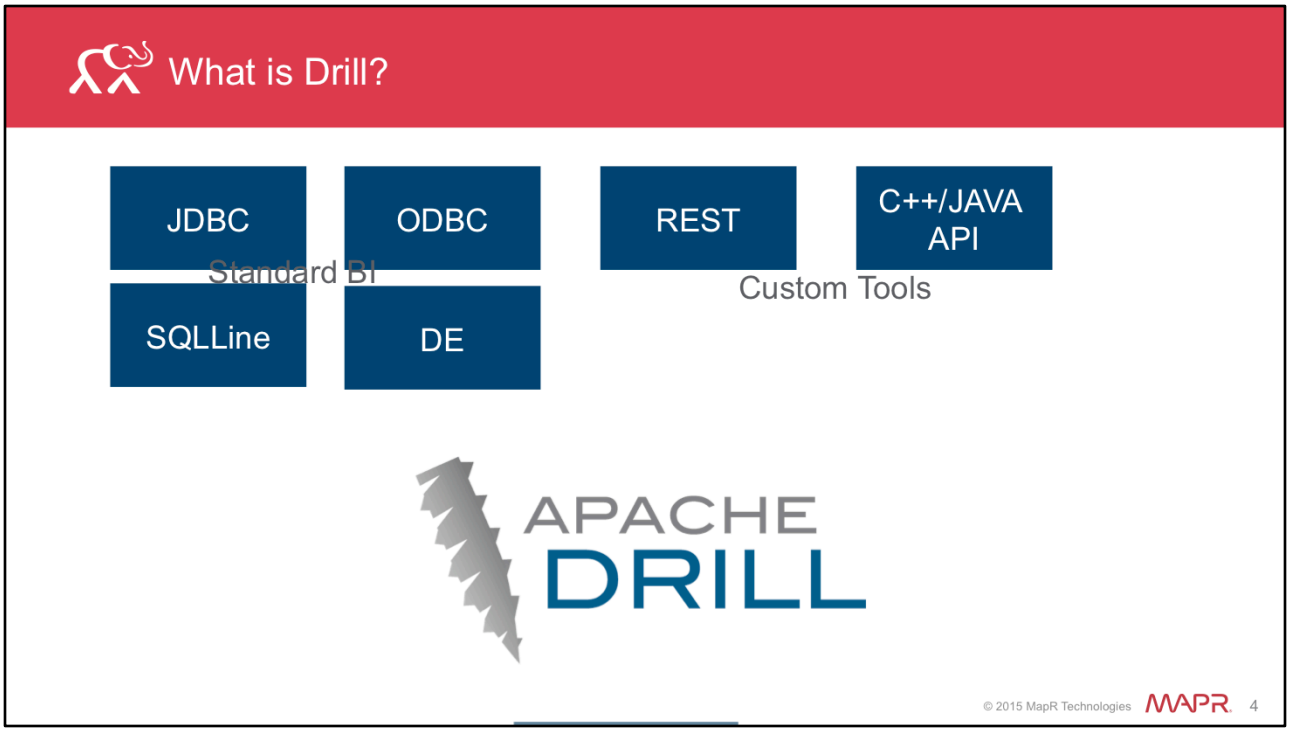
 MAPR

What is Drill?

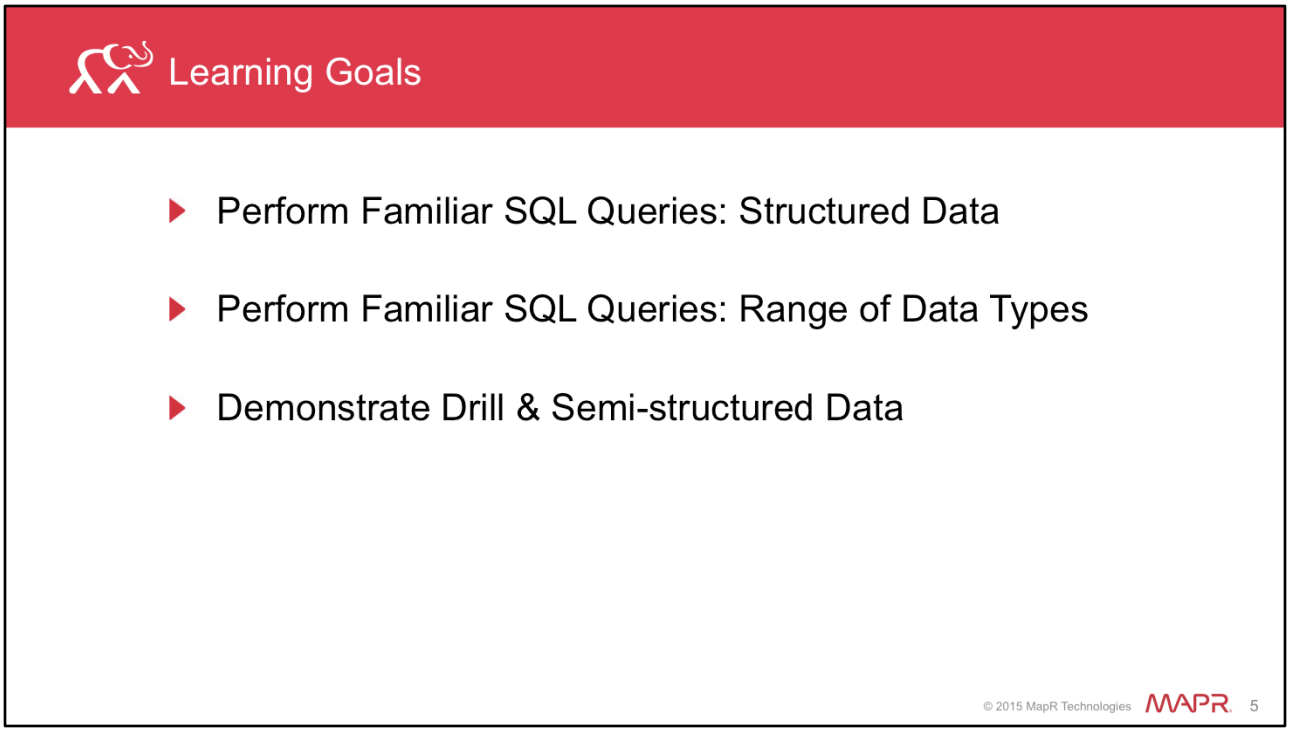
- Apache Drill is a query engine for Big Data exploration.



- Apache Drill uses standard ANSI SQL to query a variety of structured, semi-structured and unstructured data types
- Drill is built with a modular architecture that can expand its access to any additional data types through custom plugins
- Drill is designed from the ground up to support high-performance analysis on the semi-structured and rapidly evolving data coming from modern Big Data applications, while still providing the familiarity and ecosystem of industry standard ANSI SQL.



- Can access drill through JDBC/ODBC, REST, C++ and JAVA API
- out of box ODBC client tool called Drill explorer (ODBC) and out of box command line tool called SQLLine (JDBC)



After you have finished this lesson, you will be able to use Drill to:

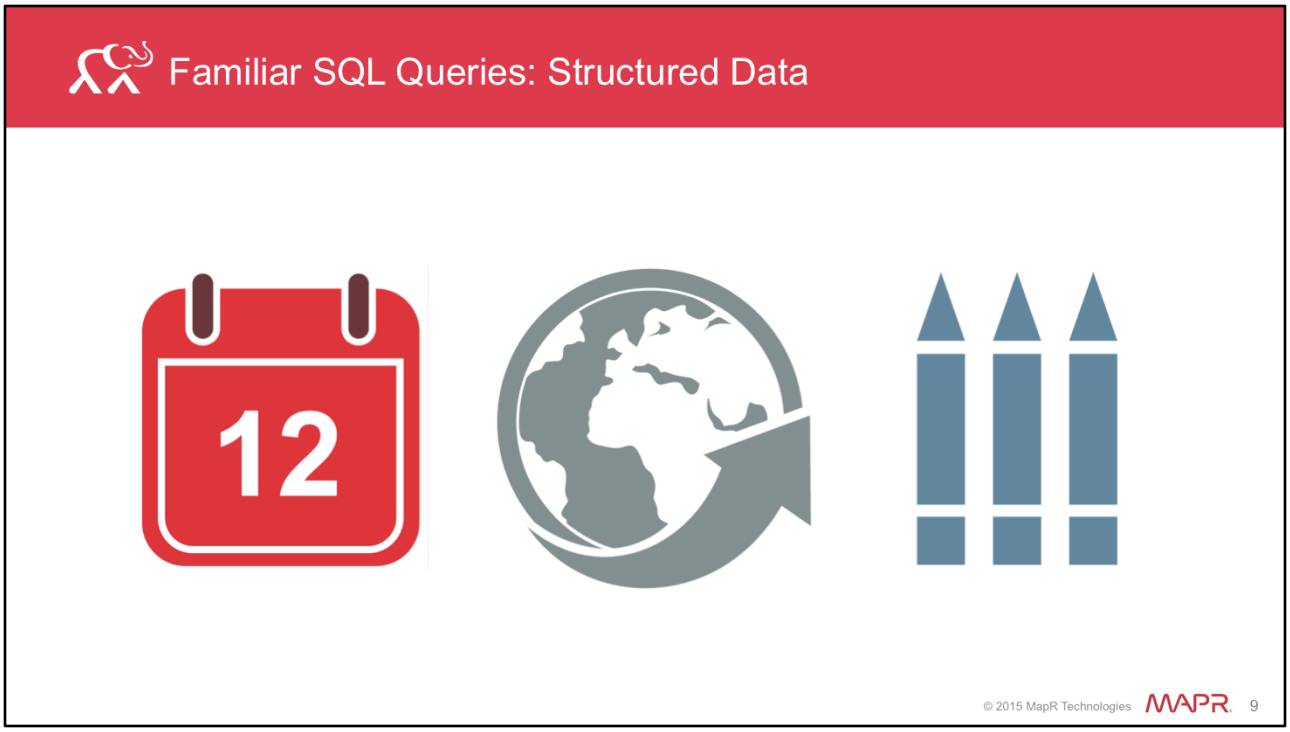
- Perform familiar SQL queries on structured data
- Perform familiar SQL queries on a range of data types
- Demonstrate how Drill can be used to explore semi-structured data

- Big data exploration with Drill is used in a variety of industries.
- Some common uses include improving information security, finding new opportunities for financial investment, analyzing medical information and examining retail data to increase sales and improve the customer experience.

- Large retail businesses, such as the The Big Office Supply Company, often employ data analysts to study their customer activity and purchasing habits.
- From this analysis, they hope to identify such advantages as new sales opportunities and product enhancements based on customer feedback.
- At the Big Office Supply Company, the first thing we want our data analysts to do is to help us expand our sales opportunities.
- We are a business that sells office supplies, after all, and we need to sell more of these supplies to expand our business.

Familiar SQL Queries: Structured Data

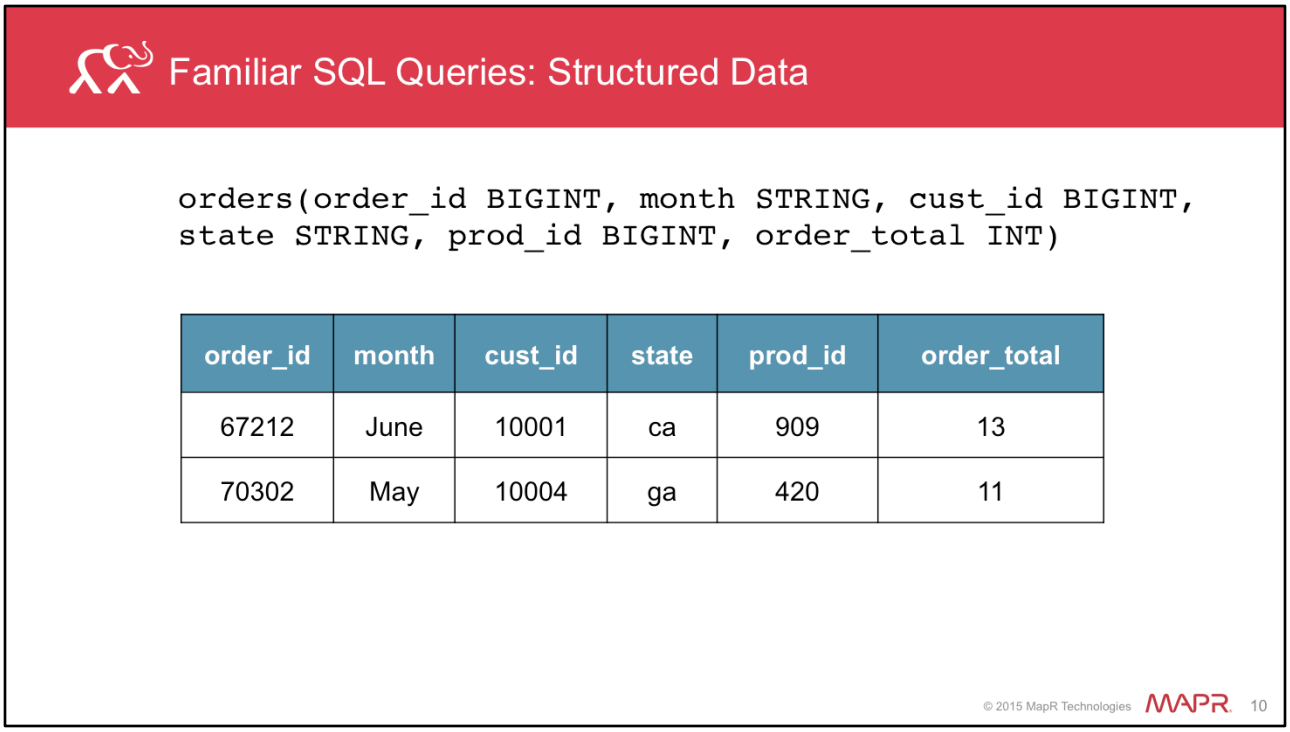
- Our marketing department wants to begin a new sales promotion
- We need to help them decide when and where to focus promotional materials .



We've asked our data analysts to tell us:

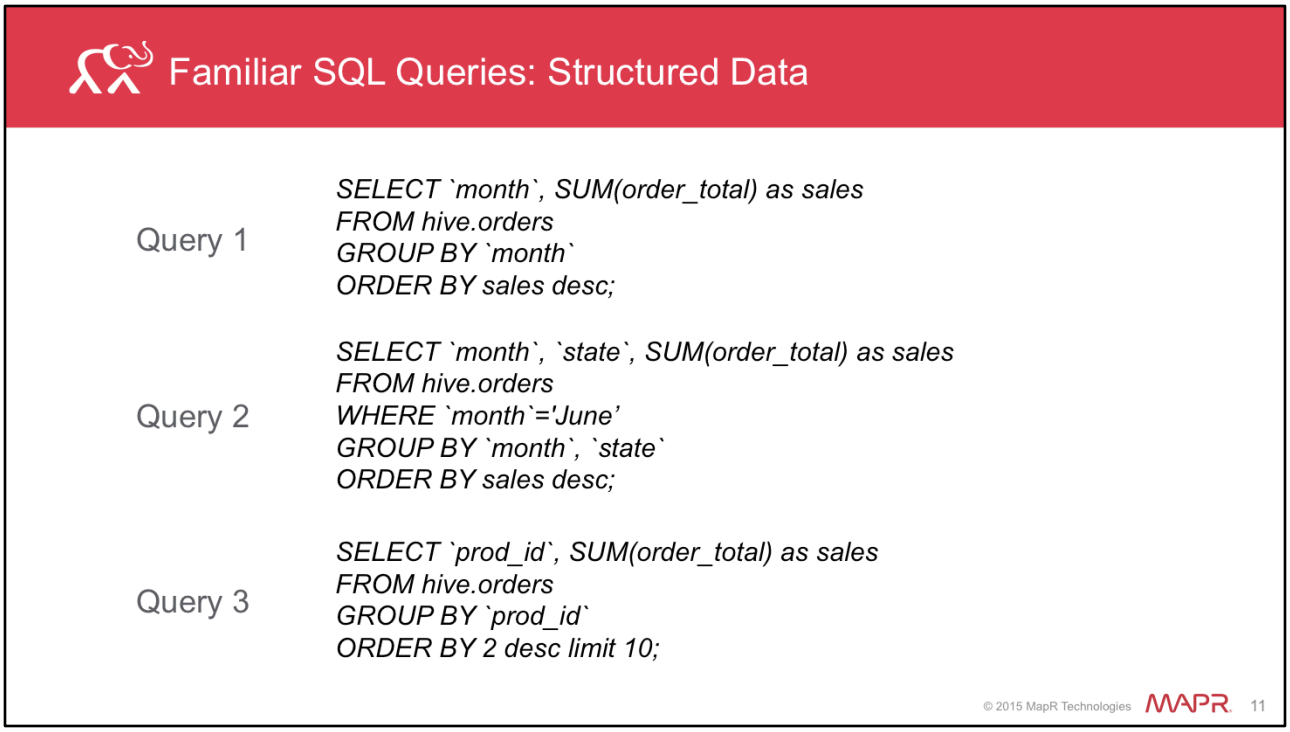
- the top grossing month of sales
- the rank of countries based on their gross sales in that month
- and the top 10 products based on total sales volume

Once we have this information, we can then determine what is the best time to start our promotion, and what areas we should focus on. We can then build a promotion using our top selling products, and target the locations and time to get our best expected ROI



1. Build table TABLE orders(order_id BIGINT, month STRING, cust_id BIGINT, country STRING, prod_id BIGINT, order_total INT)
(Add column header and a couple of rows of data as spoken)

- Our company has offloaded its customer data from a transactional Oracle system into a Hive table on a Hadoop cluster.
- This table, called 'orders,' includes columns for the order ID number, the month the order was placed, the customer ID of the person who placed the order, the country where the customer lives, the product ID of the order and the total amount of the order.
- Our data analysts will use Drill to query this table and find the information that we asked them for



2. Image of CLI query, and then results

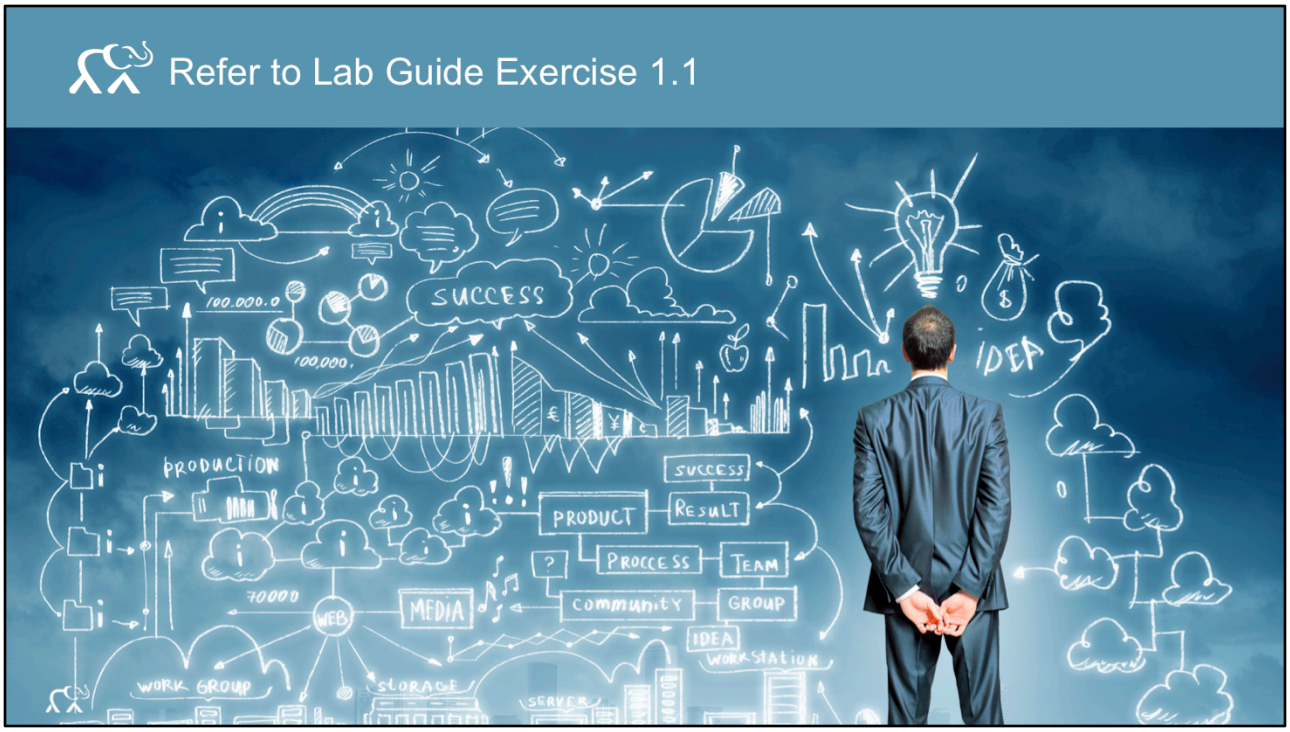
- First, they will make the following query to determine the top month of gross sales …

3. Image of CLI query, and then results

- Then, we will query for the sales total in that month for each state…

4. Image of CLI query, and then results

- And finally they will determine the top 10 products based on volume of sales .

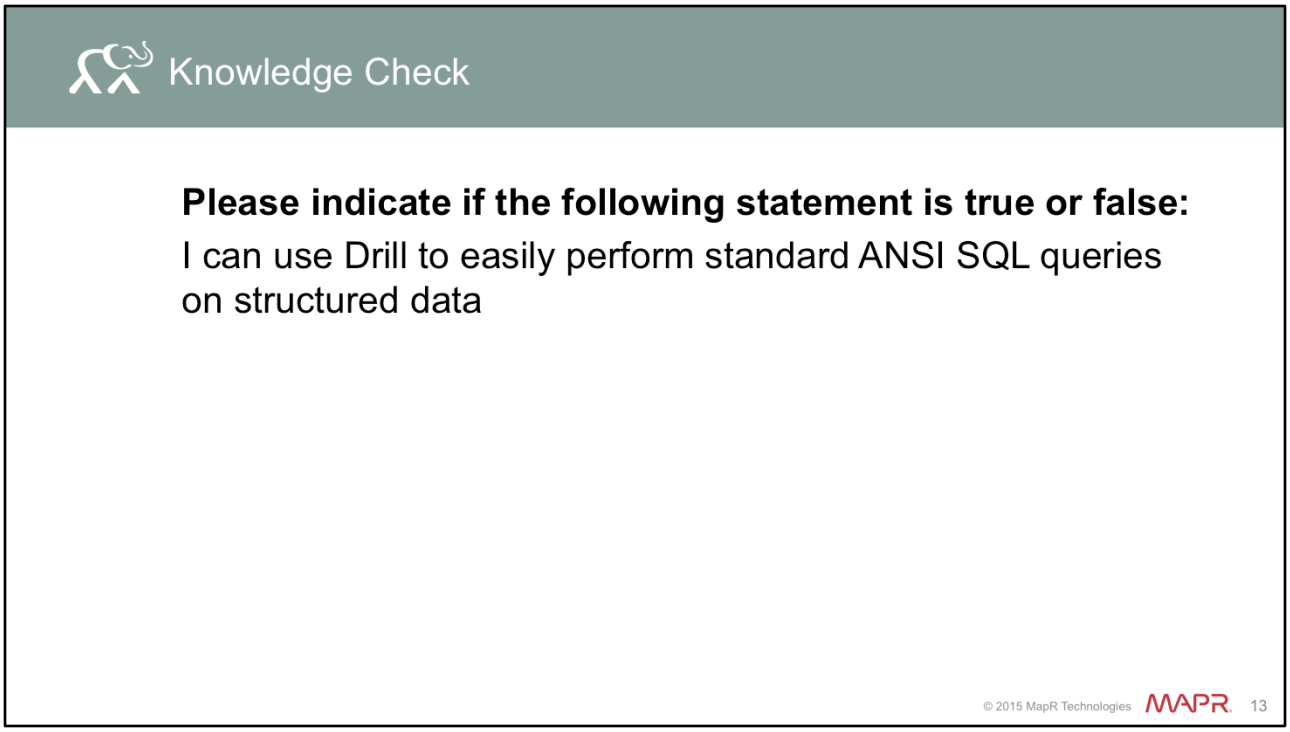


1. Build table TABLE orders(order_id BIGINT, month STRING, cust_id BIGINT, country STRING, prod_id BIGINT, order_total INT)
(Add column header and a couple of rows of data as spoken)

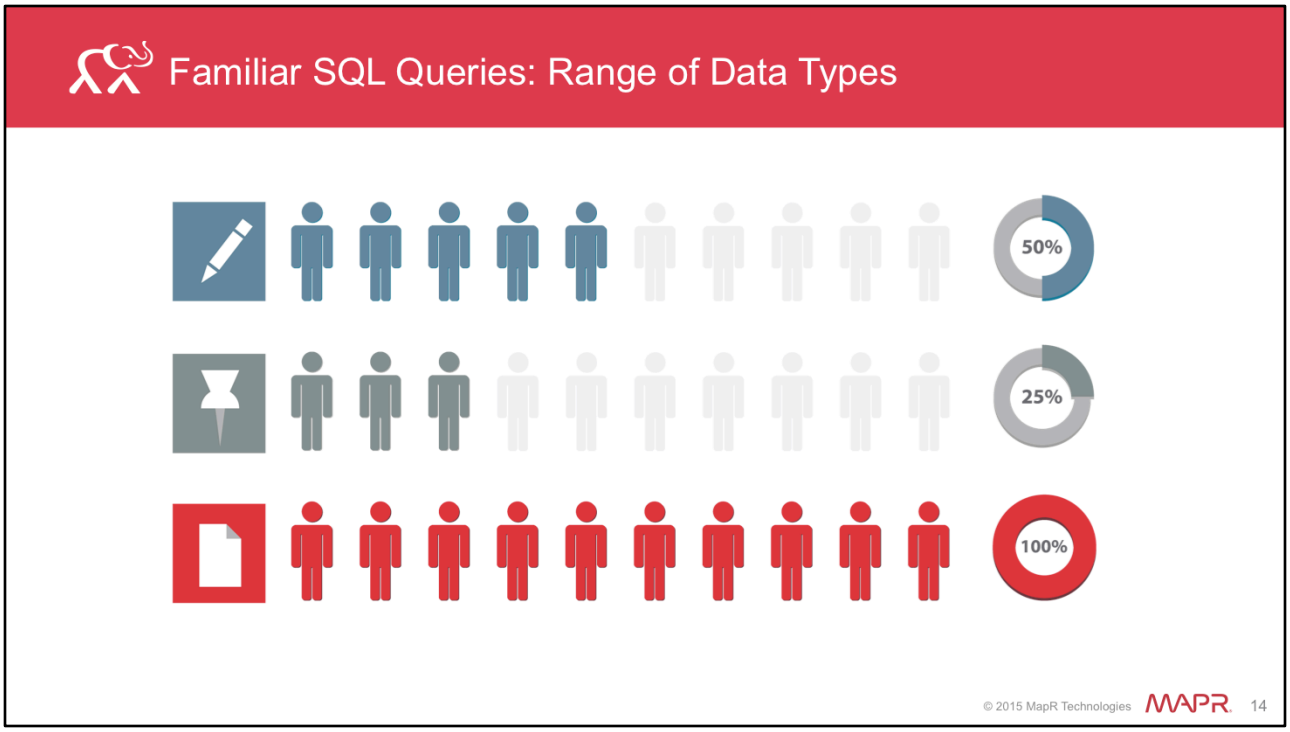
- Our company has offloaded its customer data from a transactional Oracle system into a Hive table on a Hadoop cluster.
- This table, called 'orders,' includes columns for the order ID number, the month the order was placed, the customer ID of the person who placed the order, the country where the customer lives, the product ID of the order and the total amount of the order.
- Our data analysts will use Drill to query this table and find the information that we asked them for

2. Image of CLI query, and then results

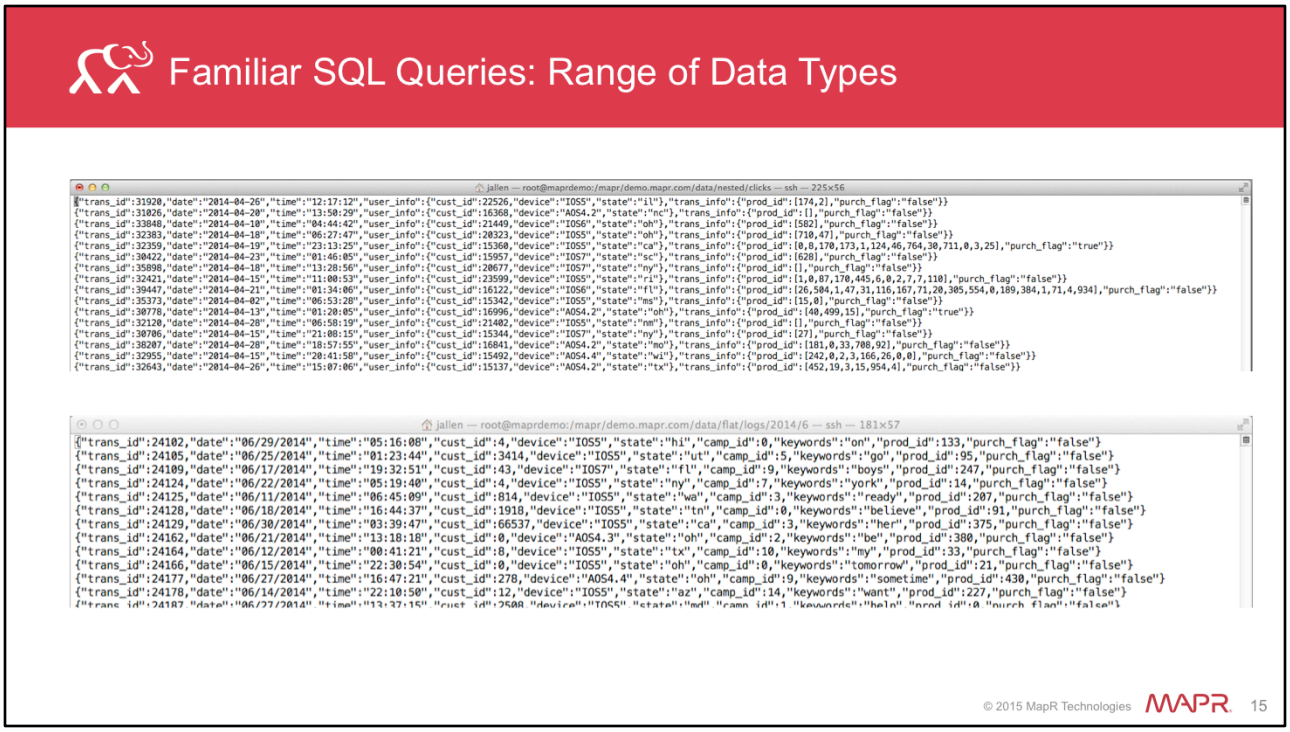
- First, they will make the following query to



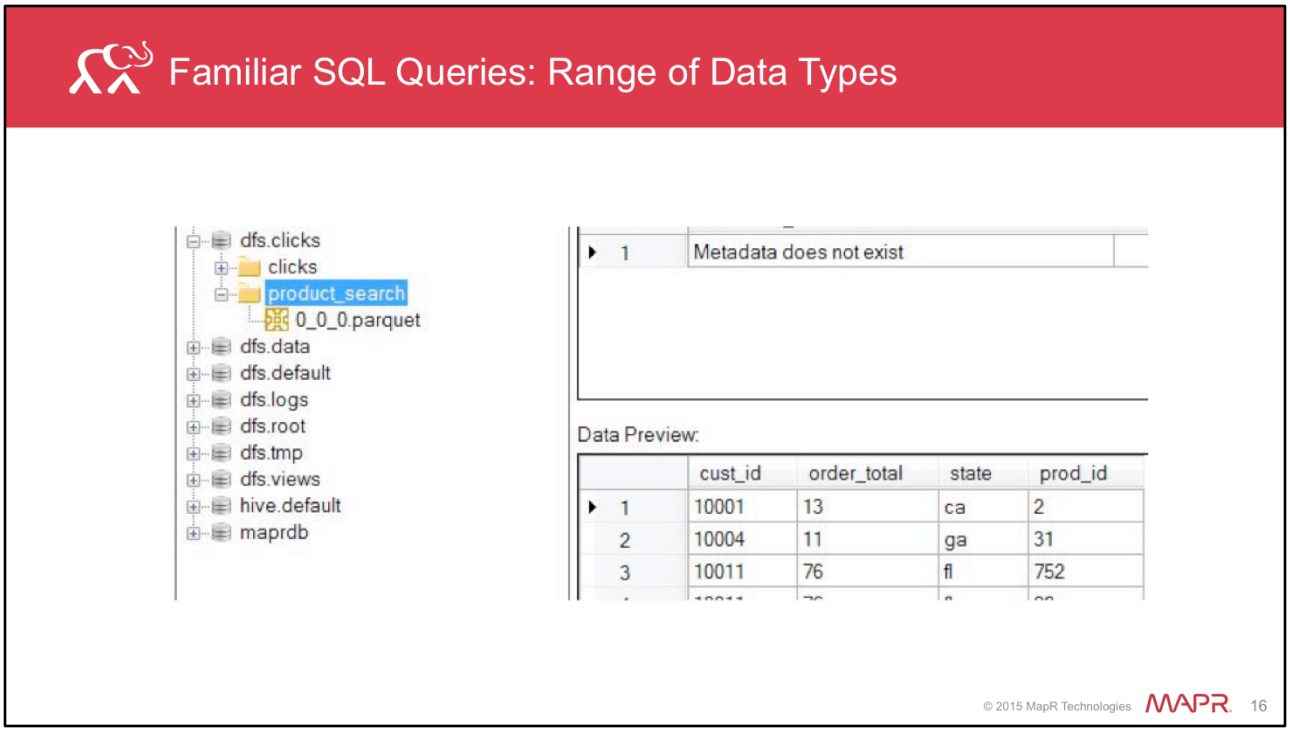
Please indicate if the following statement is true or false:
I can use Drill to easily perform standard ANSI SQL queries on structured data. **(TRUE)**



- Our marketing department is pleased with the results that they received about sales times and locations, but our data analysts are not satisfied.
- They don't feel that just looking at customer purchases gives us the full picture of who is most interested in our products.



- As users visit our site and view our product pages, the server collects both click data in a nested JSON format and log files as flat text.

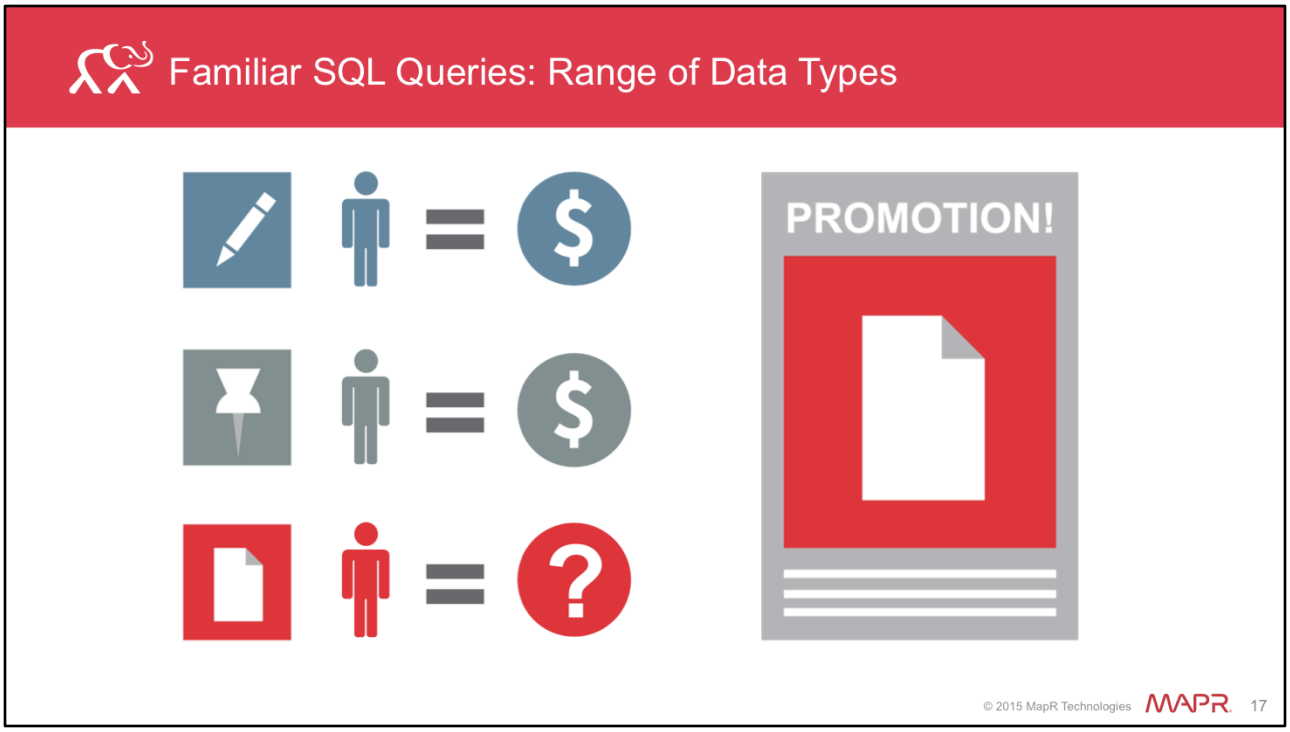


- We also collect all user product search requests, which are regularly converted into parquet format.
- Parquet is a format designed to improve query speed, and will be described in more detail in later lessons.
- Our data analysts want to query all of these files for additional insights

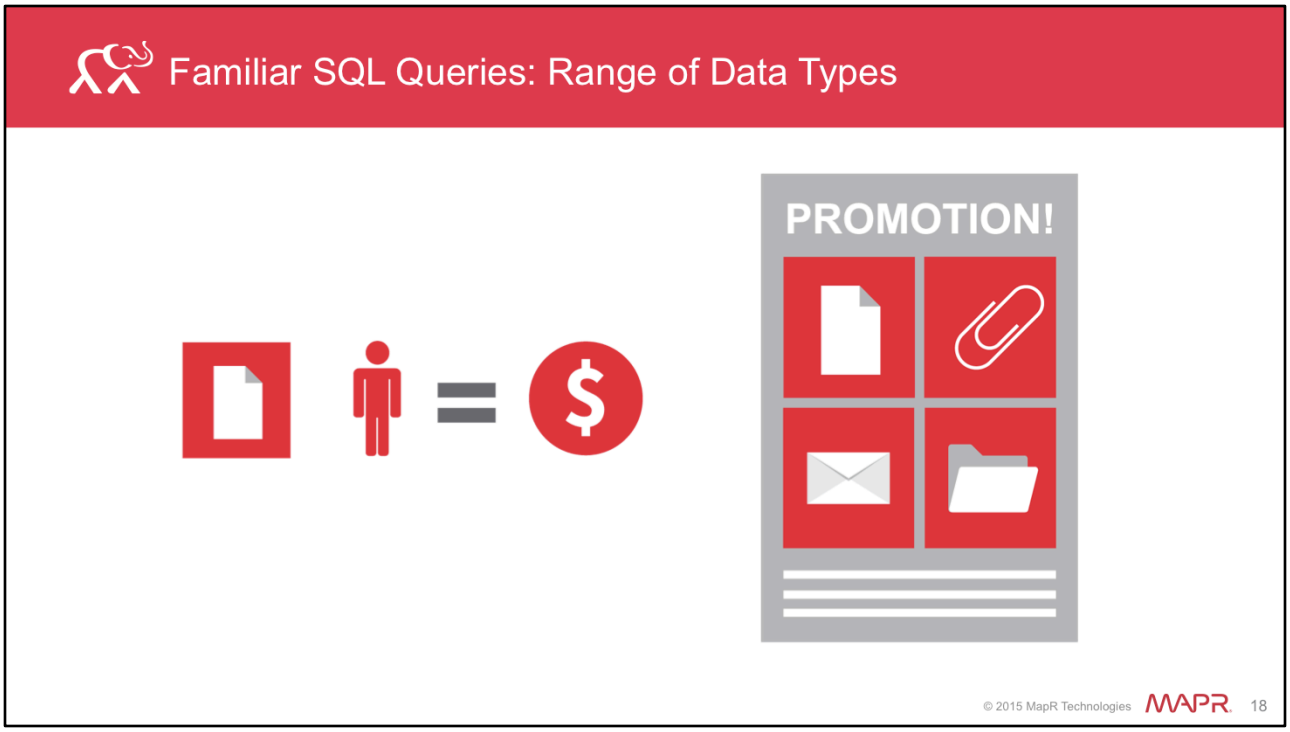
Nested JSON
/mapr/demo.mapr.com/data/nested/clicks/clicks.json

Flat Log
/mapr/demo.mapr.com/data/flat/logs/2014/6/log.json

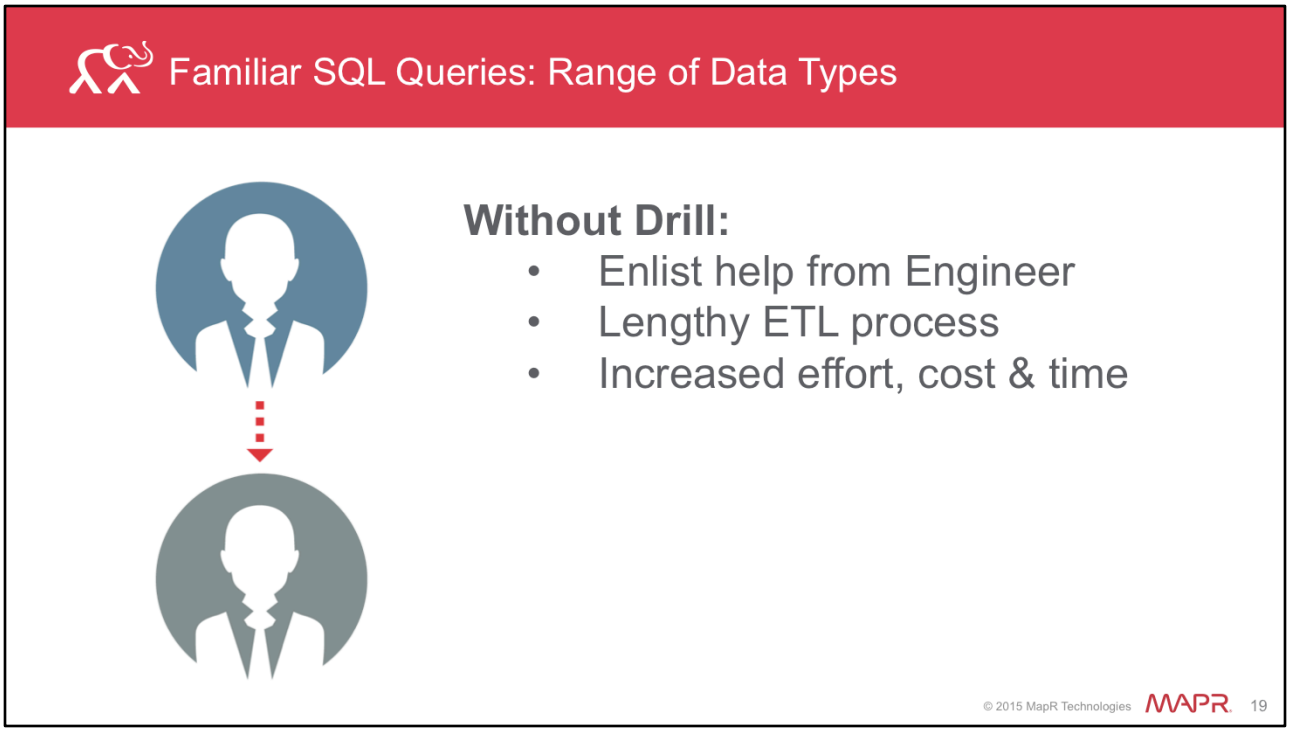
Parquet
/mapr/demo.mapr.com/data/nested/cust_orders/
0_0_0.parquet



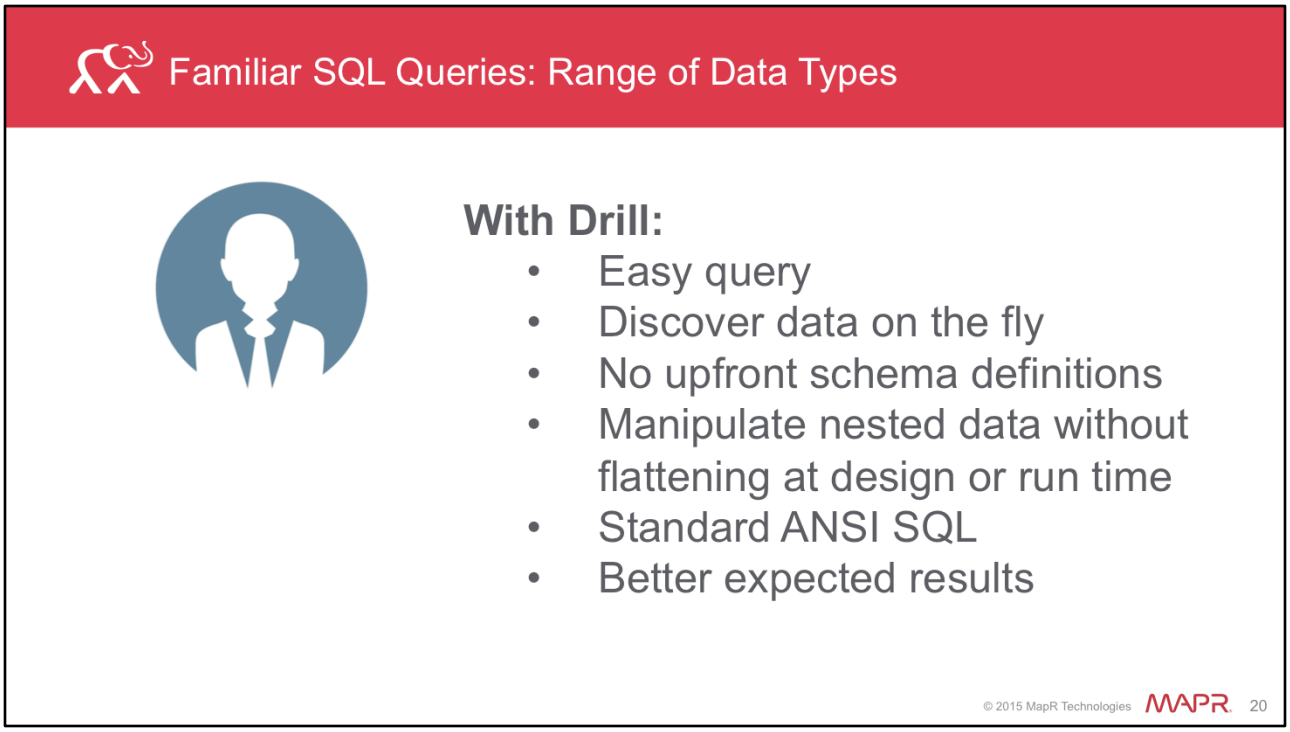
- First, they will look at the search data to see what are the most popular items searched over the last year, and target those items in our promotion
- Next, they use the click and log data to look at who is viewing our site, but did not buy the product at which they were looking.
- Since we know they have interest in that specific product, we can target our promotion to that item and expect a higher rate of return



- Also, they want to look at exactly what different people are purchasing
- When we couple this with information from our products database, we can then provide specific, targeted coupons for upgrades or consumables to match those products

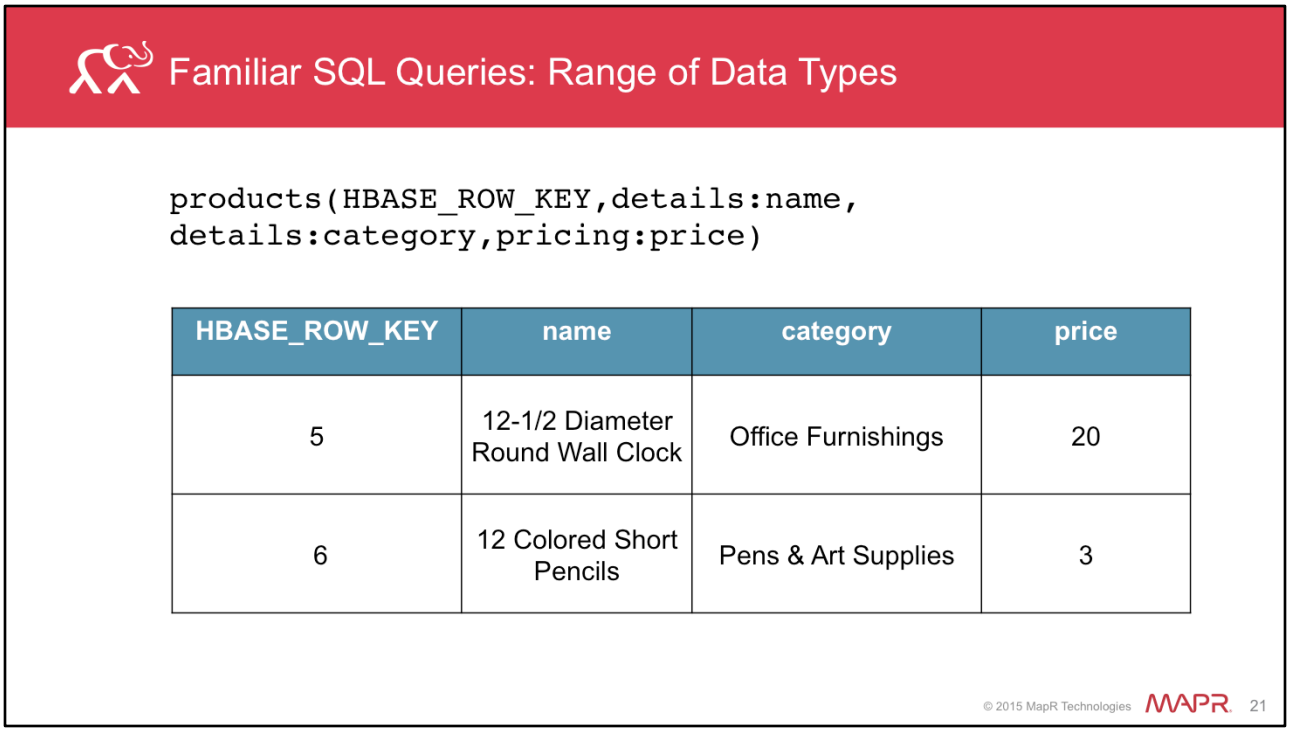


- Normally, our data analysts would be out of luck trying to use SQL on these types of files.
- To query data of this type, they would either need to enlist a data engineer to help them write a MapReduce program, or perform a lengthy ETL process to flatten the data and create a centralize schema for it.



- Drill makes querying semi-structured data extremely easy.
- Drill can discover the schema of the data on the fly, allowing our data analysts to query the data without creating any upfront schema definitions.
- Drill's flexible data model makes it easy to query and manipulate nested data without any flattening at design time nor at run time.
- Our data analysts can use Drill to query our JSON and parquet files using the standard ANSI SQL that they are familiar with, just like they did with our structured database.
- Because of this, Drill allows our data analysts to quickly and easily gain new insights into our data, and get much better expected results from our new promotion

We will discuss more about how Drill discovers the schema of data on the fly in the next lesson

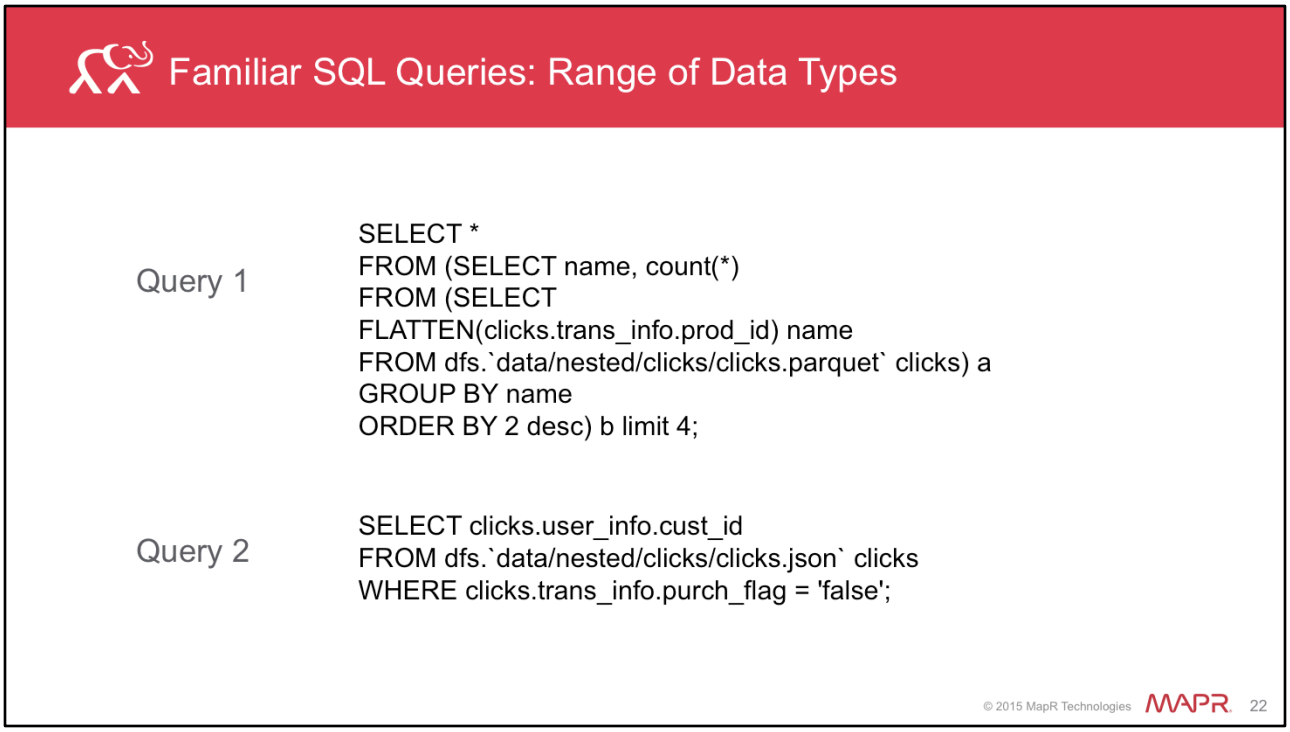


1. *show JSON file and flat file directory tree, zoomed out from server icon*

- The web click and log are stored on our servers. The JSON click files are large files of nested data, while the log files are flat text and are stored in folders saved by year, month and day. Drill can access these files directly, without the need to move them to into a data warehouse or even onto our Hadoop cluster\

2. The parquet files are a nested columnar format, and are offloaded into our Hadoop file system daily.

- The parquet files are a nested columnar format, and are offloaded into our Hadoop file system daily.



Our Data Analysts will use Drill to perform similar SQL queries as before, this time using the nested JSON data, the flat text files, the parquet files, the HBase table, and also combining these different data types into a single query, on the fly.

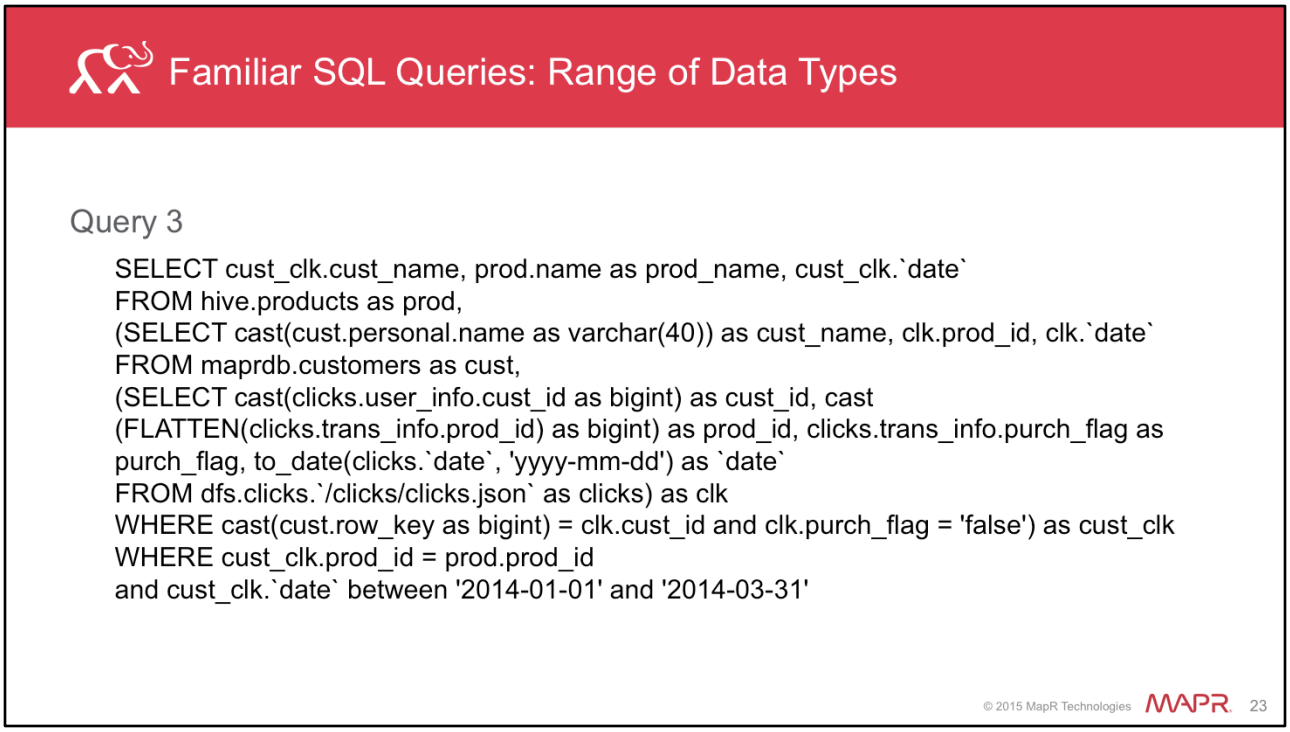
Query 4. Image of CLI query, and then results

First, they will use the following query to look at the parquet files, and determine the top 4 searched for items over the last year.

Query 5. Image of CLI query, and then results

- They will then make the following query on nested JSON and flat text files to see who visited our site, but did not buy the product or products that they were viewing

Query 6. Image of CLI query, and then results



Our Data Analysts will use Drill to perform similar SQL queries as before, this time using the nested JSON data, the flat text files, the parquet files, the HBase table, and also combining these different data types into a single query, on the fly.

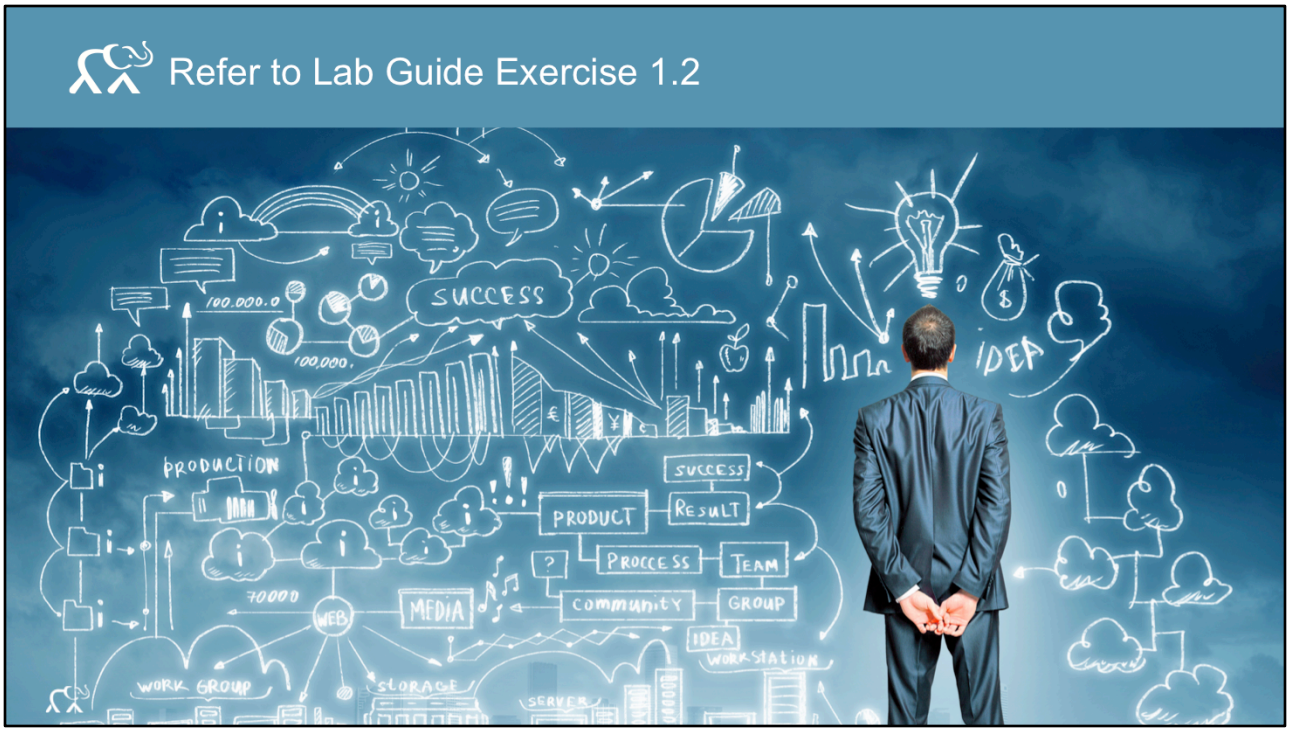
Query 4. Image of CLI query, and then results

First, they will use the following query to look at the parquet files, and determine the top 4 searched for items over the last year.

Query 5. Image of CLI query, and then results

- They will then make the following query on nested JSON and flat text files to see who visited our site, but did not buy the product or products that they were viewing

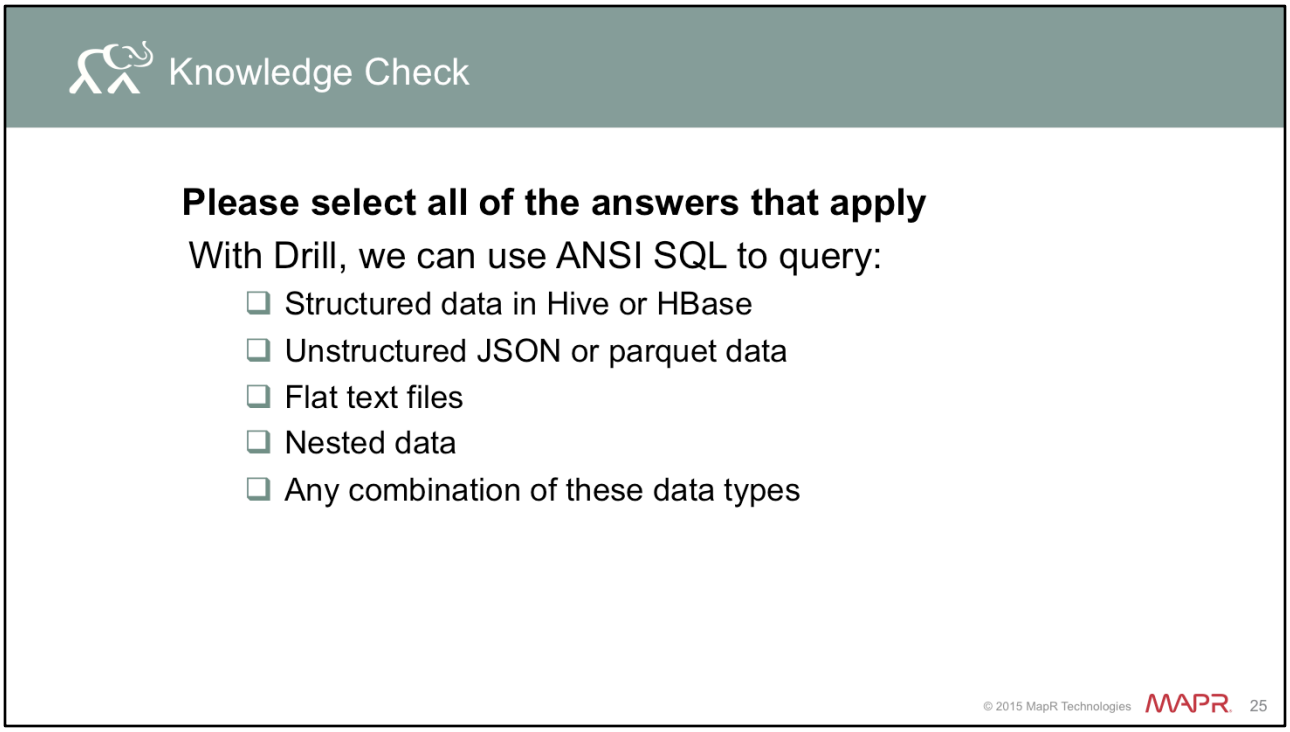
Query 6. Image of CLI query, and then results



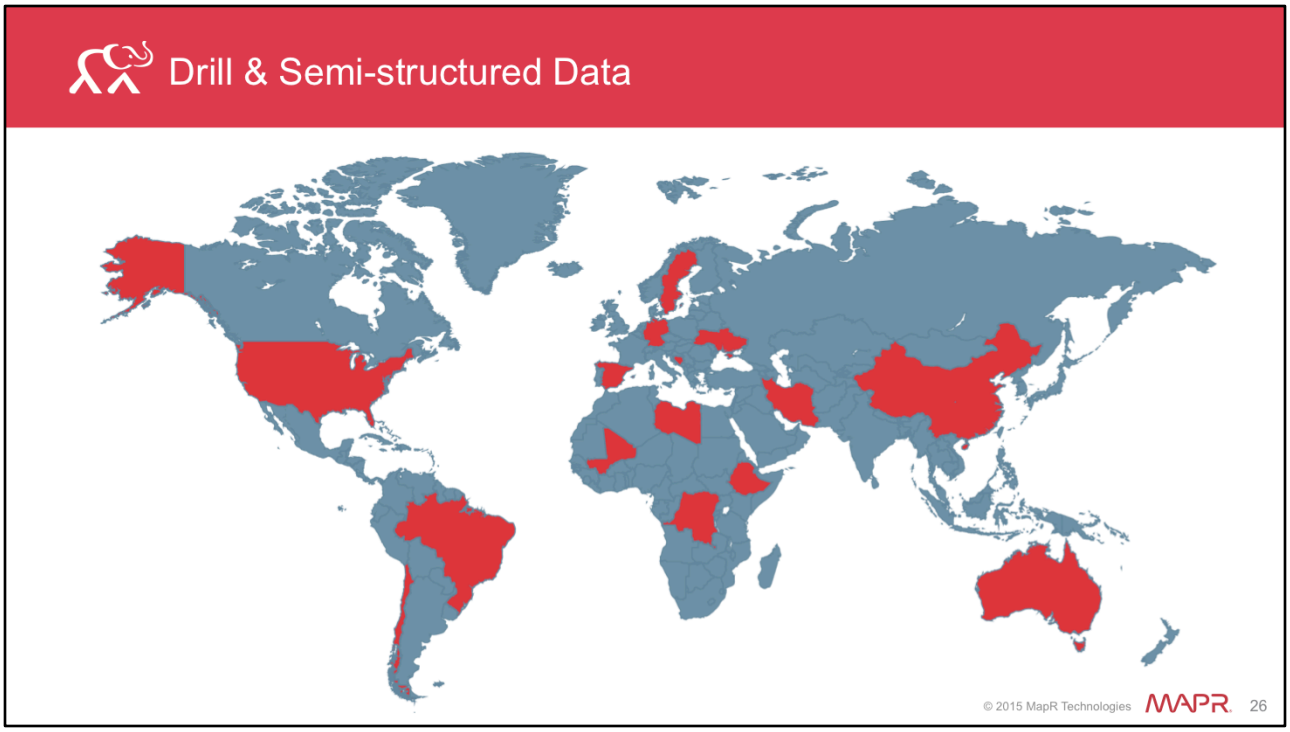
1. **Show JSON file and flat file directory tree, zoomed out from server icon**
   - The web click and log are stored on our servers.
   - The JSON click files are large files of nested data, while the log files are flat text and are stored in folders saved by year, month and day.
   - Drill can access these files directly, without the need to move them to into a data warehouse or even onto our Hadoop cluster .

2. **Show parquet format zoomed out from a cluster icon**
   - The parquet files are a nested columnar format, and are offloaded into our Hadoop file system daily.

3. ***Build table products (HBASE_ROW_KEY,details:name,details:category,pricing:price)***

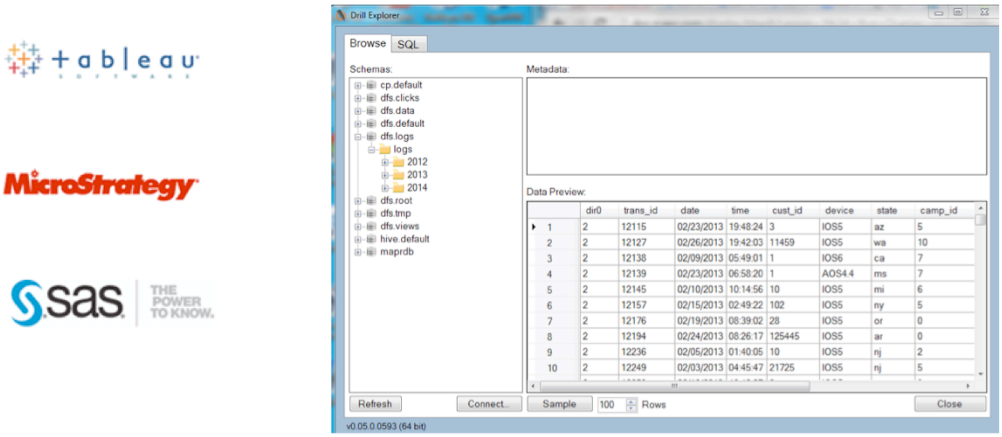


Please select all of the answers that apply
With Drill, we can use ANSI SQL to query:
Structured data in Hive or HBase
Unstructured JSON or parquet data
Flat text files
Nested data
Any combination of these data types
(Check all)



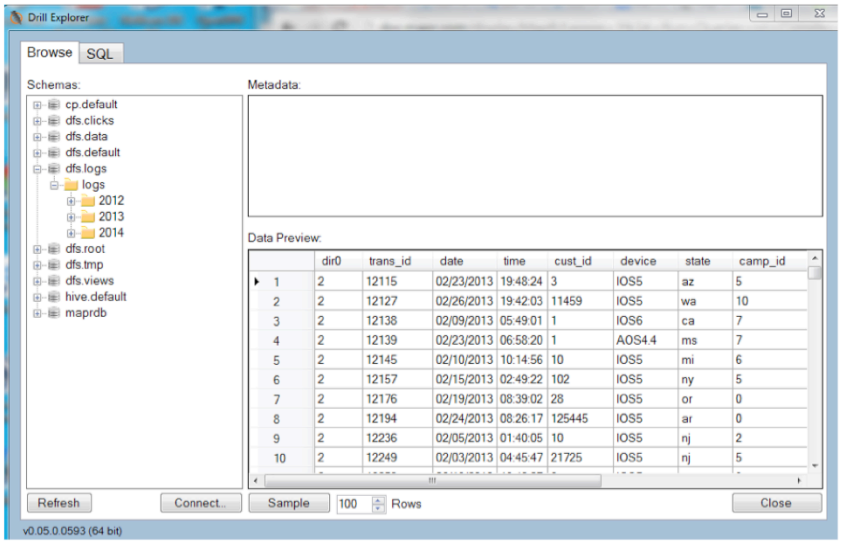
- Our data analysts have done a great job using drill to easily query a variety of data sources to help us target our promotions to get a much higher rate of return for our marketing dollars.
- Yet we have just begun to learn how drill can be used to access our company's data.

Drill & Semi-structured Data



- In addition to writing SQL queries with the command line, drill provides JDBC and ODBC interfaces that can be used to connect to standard BI tools like Tableau, Microstrategy or SAS, and the ODBC driver can be used to access a GUI interface called Drill Explorer

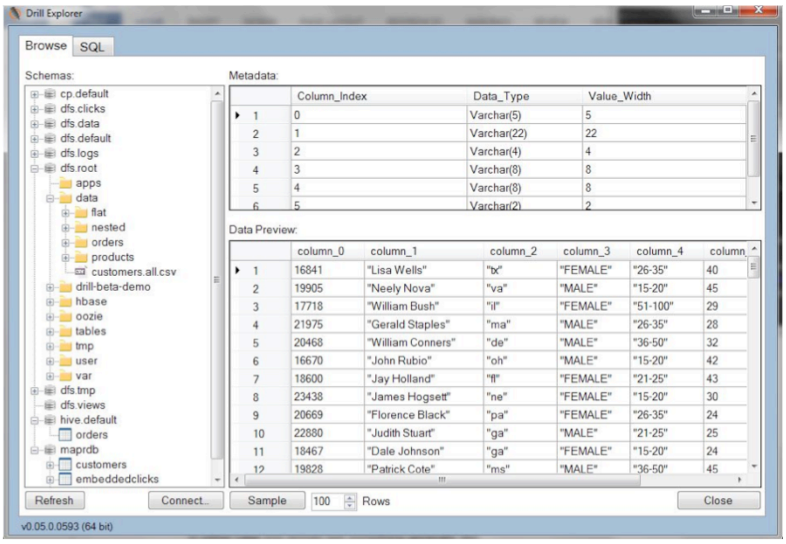
Drill & Semi-structured Data



 28

- Drill explorer can be used by our data analysts to examine and understand the metadata available in the schema-less and semi-structured data before they query it with visualization tools

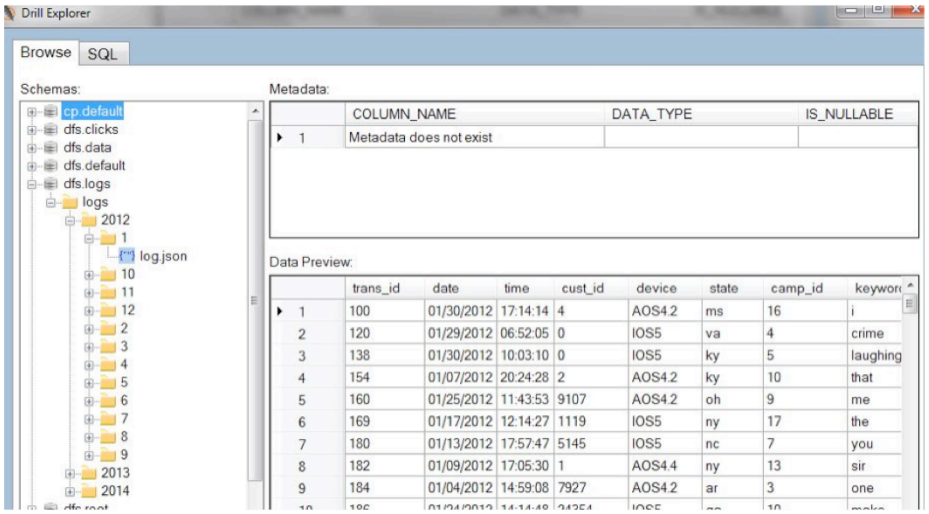
Drill & Semi-structured Data



 MAPR 29

- We can give drill access to multiple data sources, structured, unstructured, known or even unknown

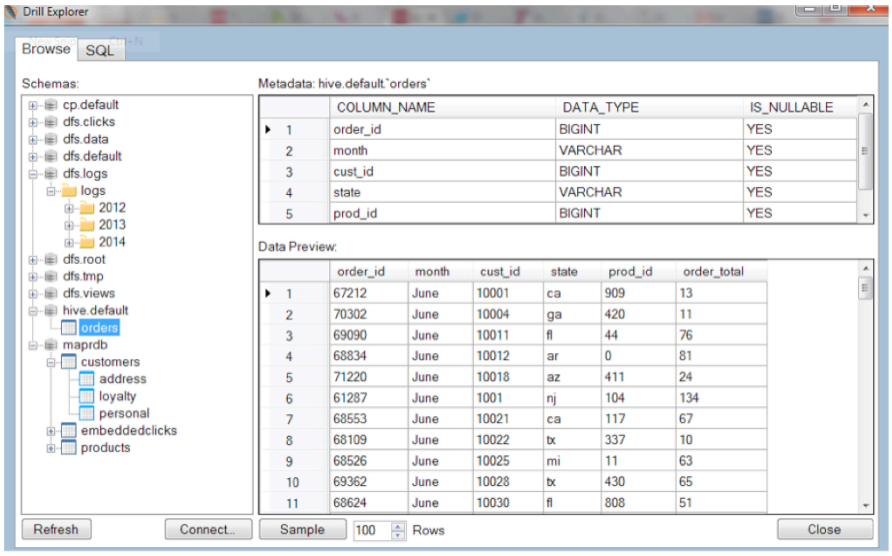
Drill & Semi-structured Data



 30

- Without the need for knowing the schema for the data, the type of data, without a data engineer writing any code and even without typing a single line of SQL, our data analysts can explore all of these different data sources.

Drill & Semi-structured Data



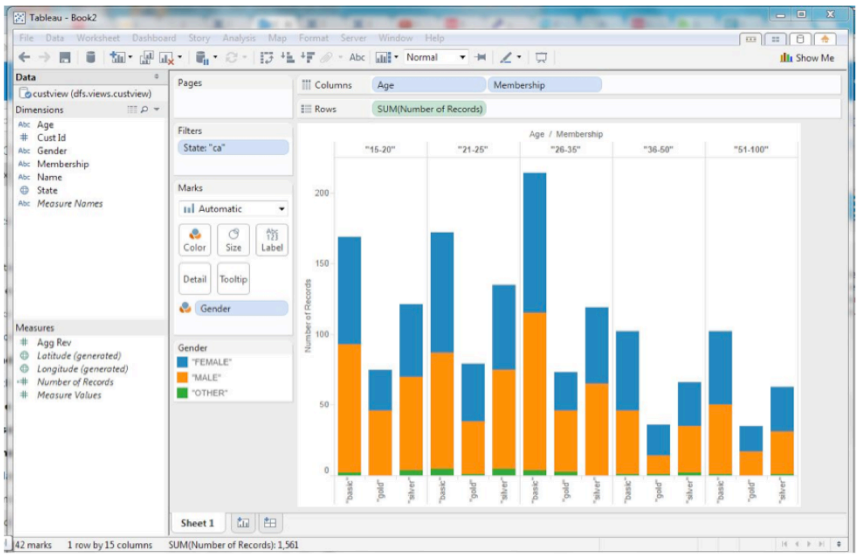
- They can quickly and easily view the structure, size and contents of each of these data sources.

Drill & Semi-structured Data

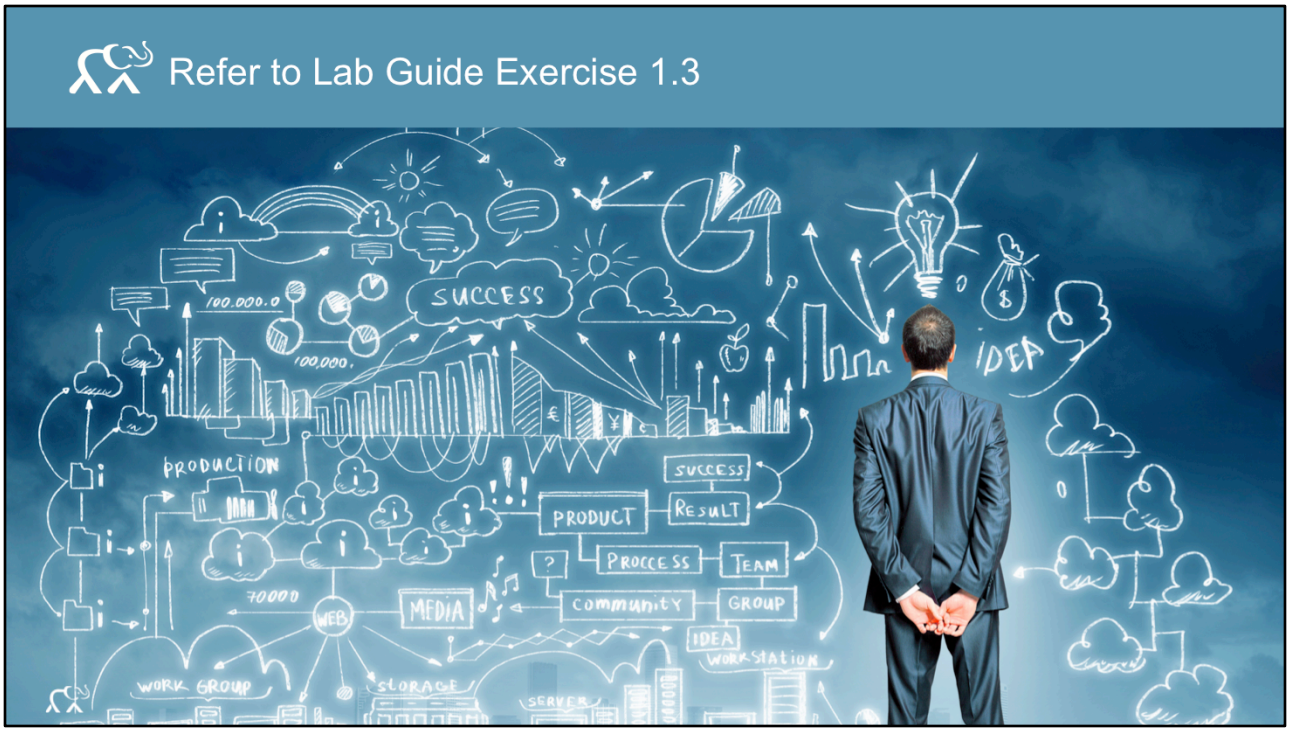
 32

- If they find something of interest, Drill will provide the SQL code for that section of data.
- As we change what we look at in the browser, the SQL will automatically update for us, and we can always edit the SQL as needed

Drill & Semi-structured Data



- With Drill Explorer our data analysts can get a better understanding of our data through quick exploration, and then easily visualize areas of interest with the BI tools that they already use, such as Tableau, Microstrategy or SAS



1. Data Analyst Image

- The Drill explorer gives our Data Analysts a unique tool to learn about data from multiple sources, and even unknown data..

1. Drill Explorer Launch Page

- When we first open the Drill Explorer, we see all of the data sources available to drill. This includes our structured HBase table, as well as our unstructured JSON and parquet data, and any other data that drill can access.

2. Select data source and view the data

- We can select any one of these data sources, and the data will appear in the browser window.
- We don't have to know anything about the data or

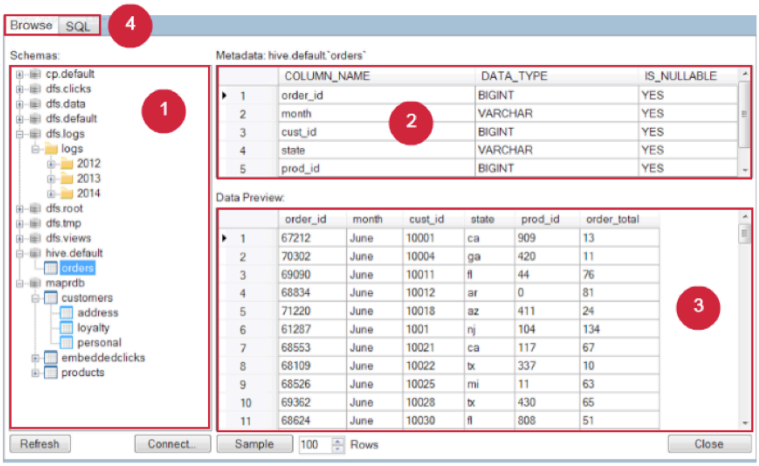
Knowledge Check

____ View all data sources available to Drill

____ Explore data saved in selected sources

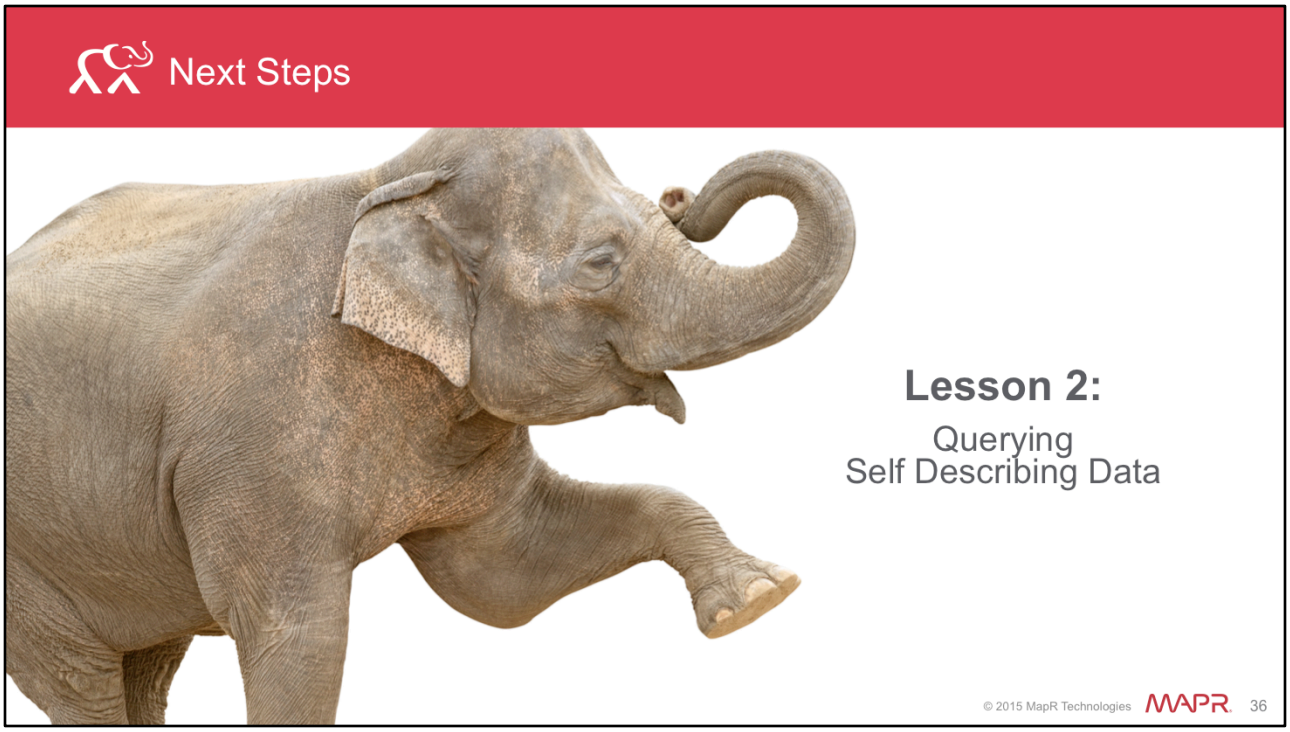
____ See meta data for selected source

____ Choose between exploring data sources or SQL use to query this data



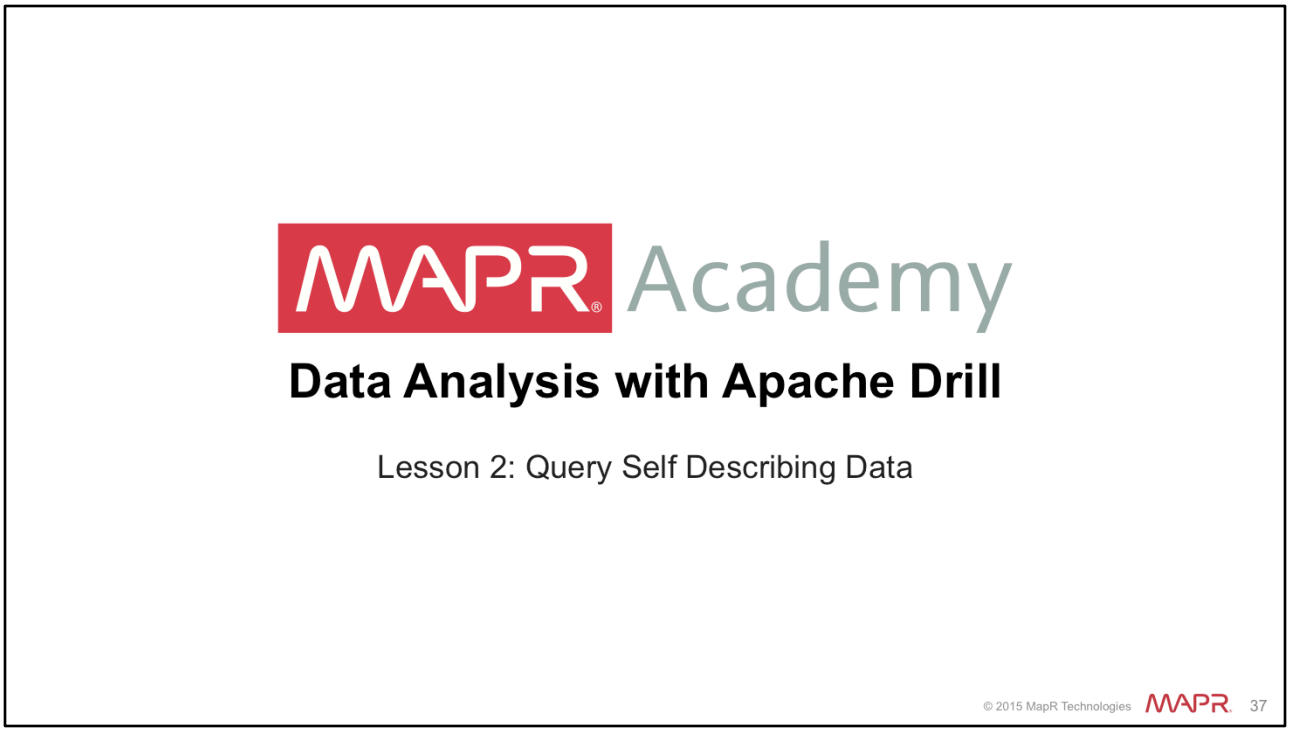
 35

1
3
2
4



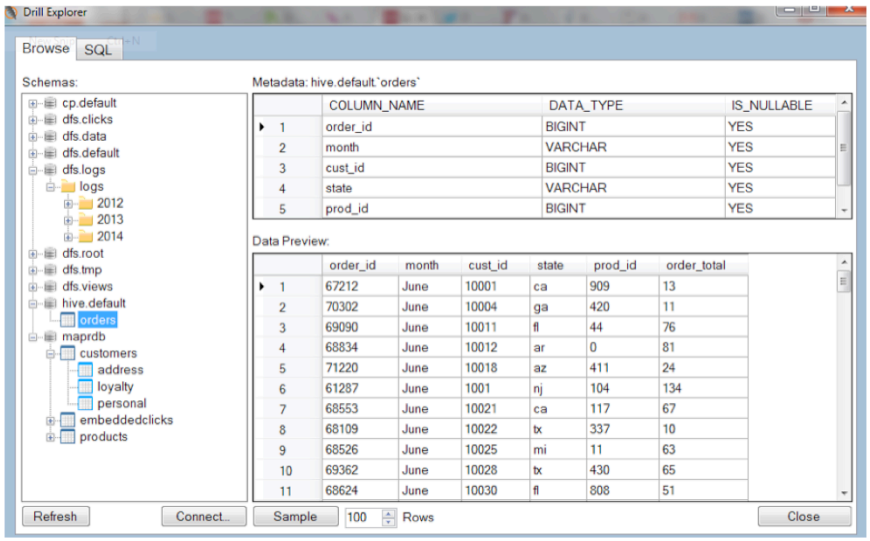
Congratulations, you have completed the first lesson of this course. Continue to lesson 2 to learn about what types of data we can access when using drill, how drill can discover the schema of unknown or complex data, and dig deeper into how drill interacts with data.

Lesson 2:
Querying Self Describing Data with Drill


MAPR Academy
Data Analysis with Apache Drill
Lesson 2: Query Self Describing Data
© 2015 MapR Technologies MAPR 37

In the previous lesson, we learned about how easily Apache Drill can be used to perform standard SQL queries on structured data, and the power of using Drill to simultaneously query structured and semi-structured data.

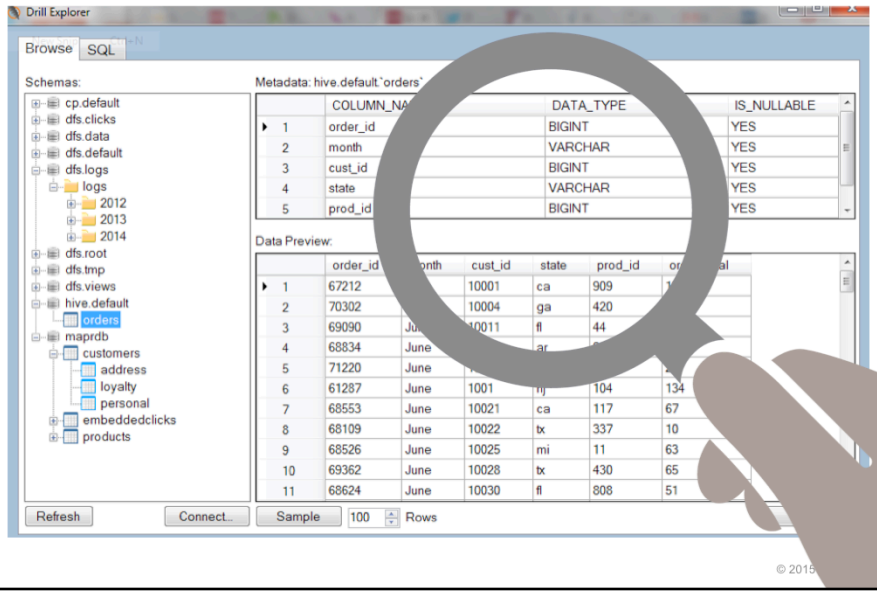
What is Drill?



 39

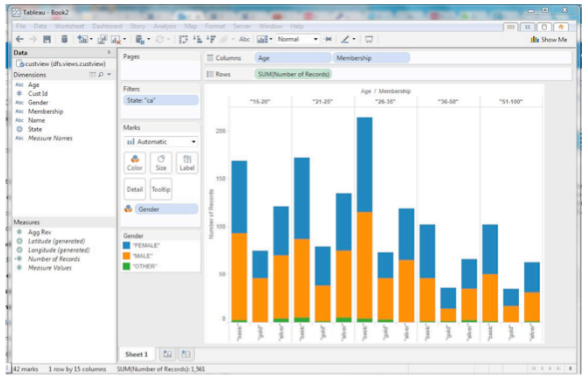
We also introduced the concept of exploring unknown data with Drill Explorer, opening up new understanding of our data without typing a single line of code.

What is Drill?

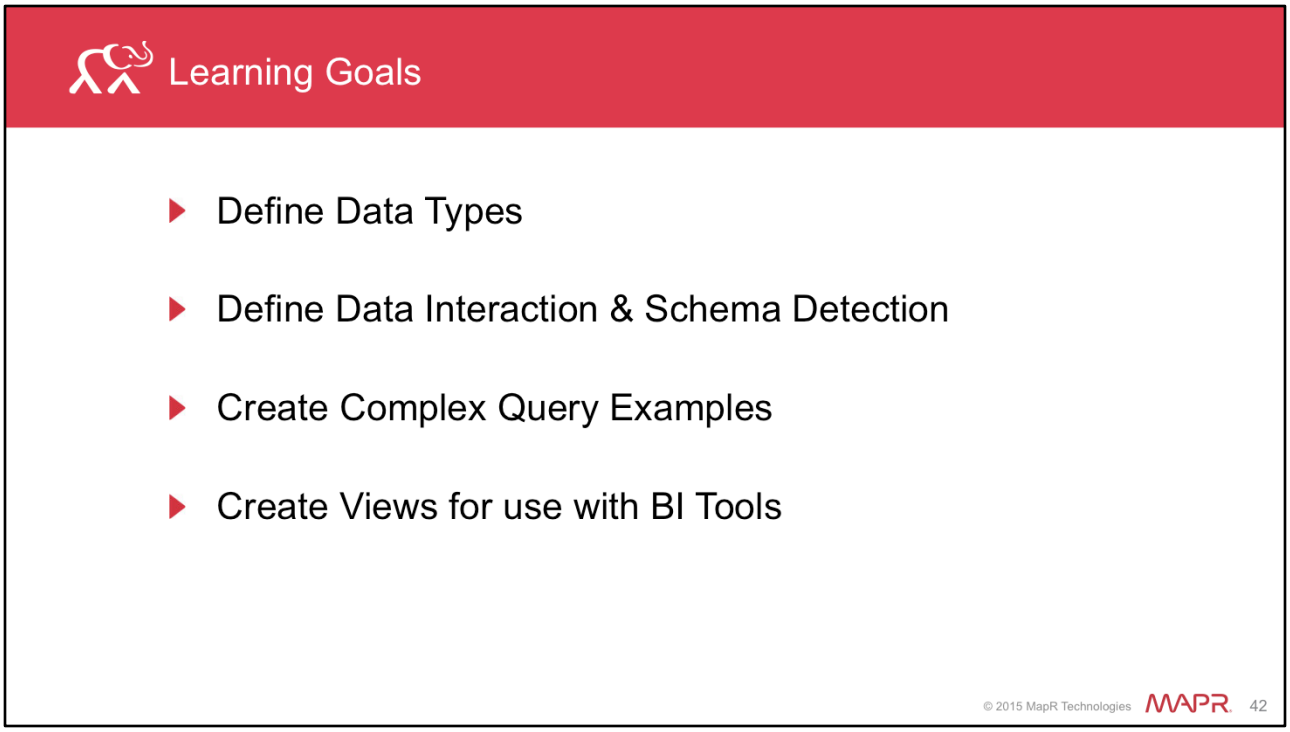


- In this lesson we will look more deeply into the queries we performed in lesson one. We will discuss how Drill is able to determine the schema of data on the fly, and look at how drill interacts with different types of data.

What is Drill?

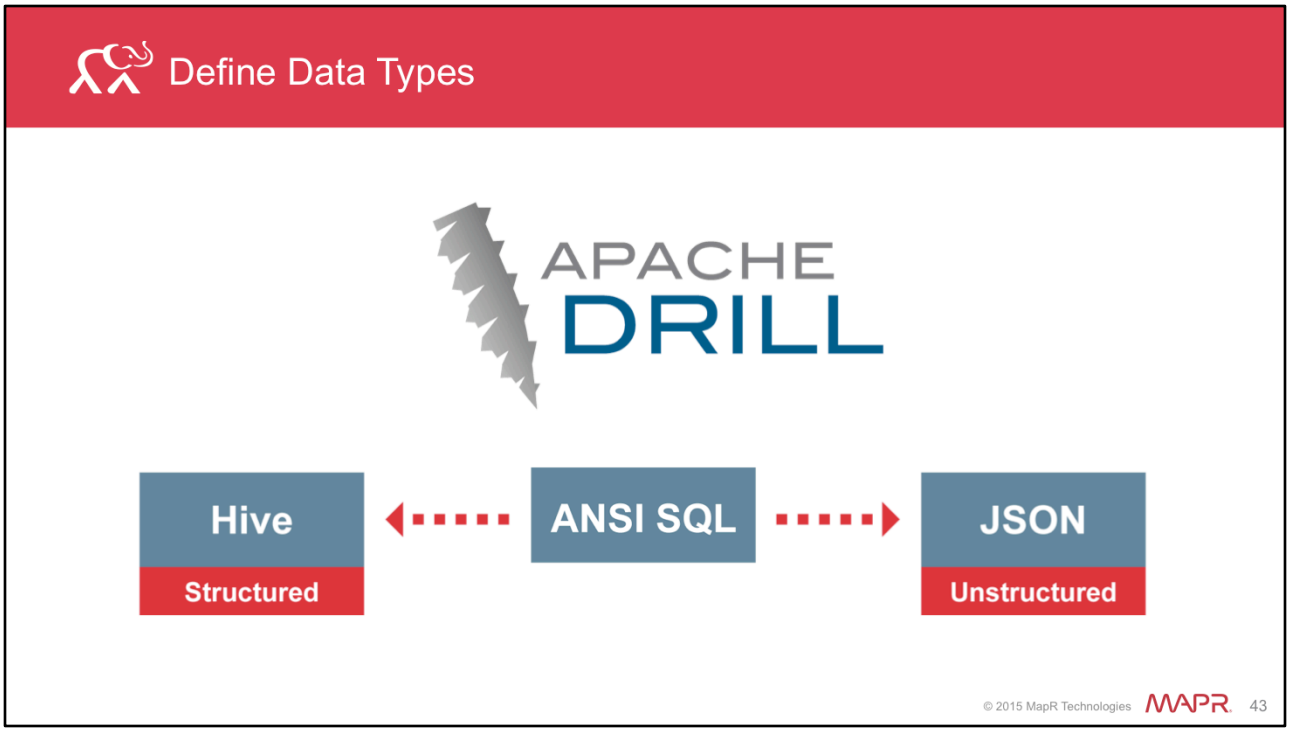


We will also continue to examine the Drill Explorer, and see how we can create new SQL views that we can use to easily visualize our data with standard BI tools via ODBC connectivity.



When you have finished this lesson, you will be able to

Define the different types of data we can explore and query with Drill
Describe how Drill interacts with these data types, and discovers their schema
Explore data to create queries using multiple data sources, including complex data like arrays and maps
Create views to visualize data in our existing BI tools without writing any code

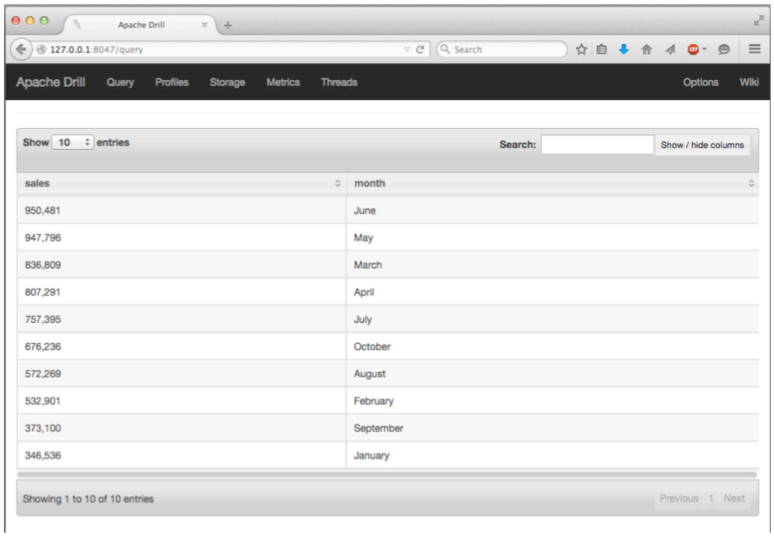


In the previous lesson, we performed SQL queries on several different data types; structured and semi-structured tables, and streaming click data.

These different data types are not just an evolution in our company's data, though. Rather, they each perform a different purpose, and each type of data is put in the format it is, because it best suits that purpose.

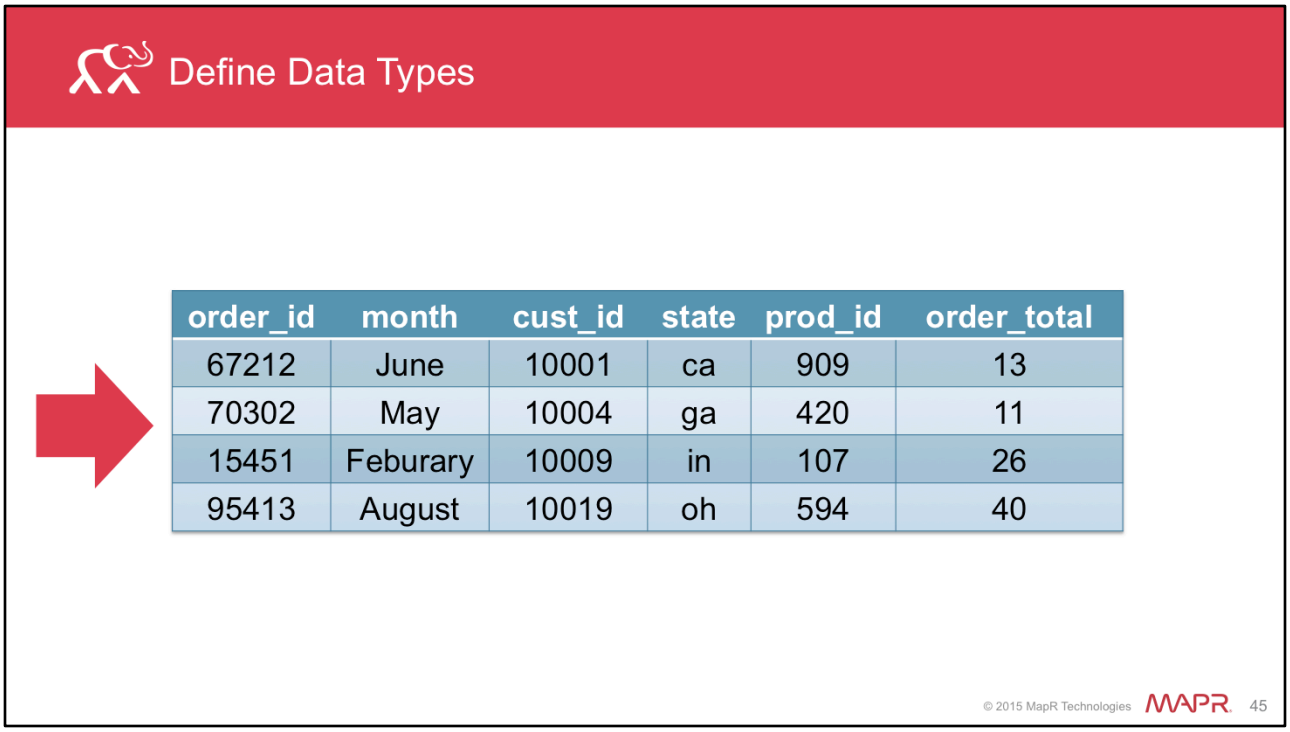
Define Data Types

```
First Query:
SELECT `month`,
SUM(order_total) as
sales FROM hive.orders
GROUP BY `month` ORDER
BY sales desc;
```



 44

Looking back at the queries we made, first we pulled some data from our orders database, stored as a Hive table. Hive is a structured tabular data format, stored on a distributed file system like MapR-FS.

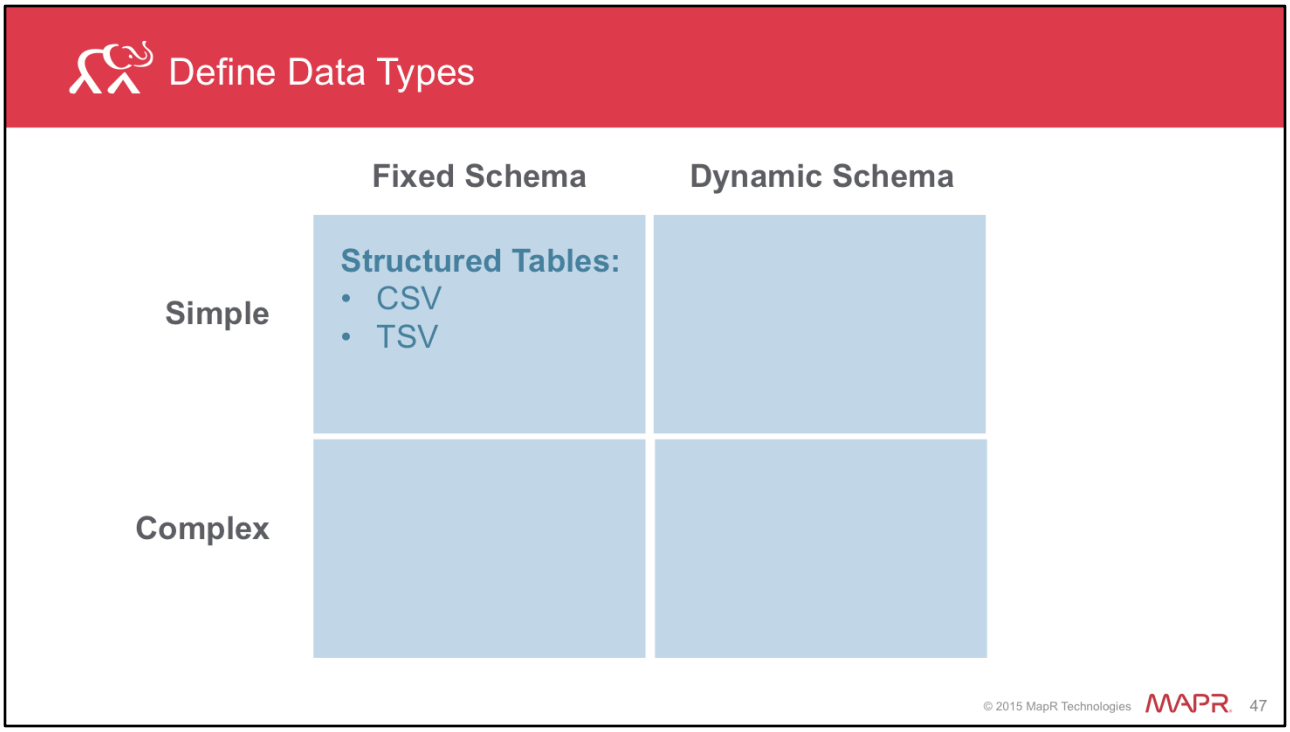


With structured data, the schema of the table is fixed. A standard relational database requires that we know the schema of our database before we load any data.

Define Data Types

order_id	month	cust_id	state	prod_id	order_total
67212	June	10001	ca	909	13
70302	May	10004	ga	420	11
15451	Feburary	10009	in	107	26
95413	August	10019	oh	594	40

With most SQL tools, we also have to know the schema of the database when we pull information back out again. Later in this lesson, we will discuss how Drill is able to discover the schema of a Hive table for you.



Structured data is flat, and very fast and easy to query, which is why relational databases have been so successful for so long. Structured data is also rigid, however, and does not allow for the schema of the data to change without overhauling the entire database, and all of the records it holds. Structured data is well suited for data sets that we know will be consistent, like that from our order form. Every order will contain the same set of fields, yielding consistent data.

Define Data Types

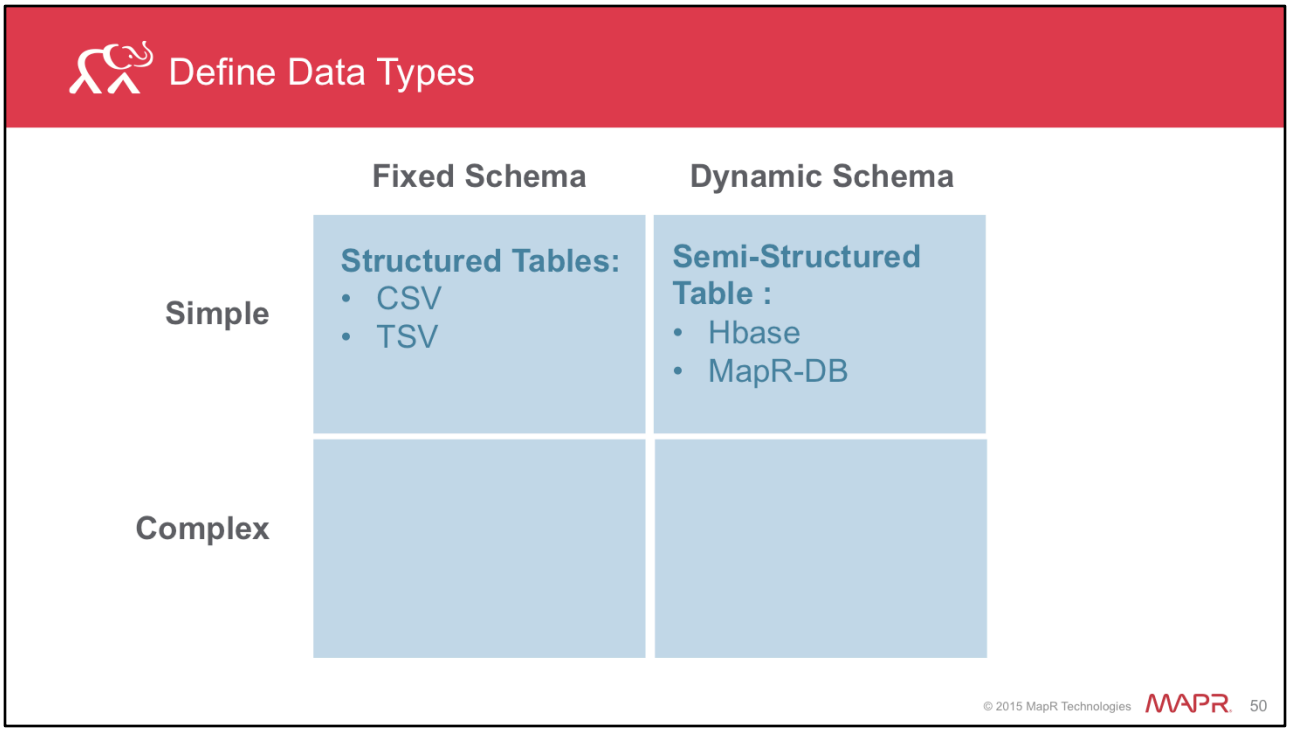
cust_id	name	state	gender	age	agg_rev	membership
16841	Lisa Wells	tx	FEMALE	26-35	40	basic
19905	Neely Nova	va	MALE	15-20	45	basic
17718	William Bush	il	FEMALE	51-100	29	basic
21975	Gerald Staples	ma	MALE	26-35	28	basic

The second set of queries included our customer and product data stored in HBase tables. Like our structured data, HBase also stores flat data in a table. HBase is an evolution of Google's Big Table, and was originally designed to be used with a distributed file system like MapR-FS.

Define Data Types

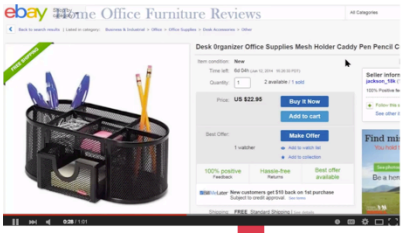
cust_id	name	state	gender	age	agg_rev	membership	reviews
16841	Lisa Wells	tx	FEMALE	26-35	40	basic	NULL
19905	Neely Nova	va	MALE	15-20	45	basic	NULL
17718	William Bush	il	FEMALE	51-100	29	basic	NULL
21975	Gerald Staples	ma	MALE	26-35	28	basic	TiqA9ybgewk

Different from the structured data, HBase uses key-value pairs to locate the requested data, and this allows HBase the advantage of having a dynamic schema. We can change the structure of the table on the fly without affecting any previously stored data. With HBase, we can add new columns and rows to our table as we come across new data needs, and fill only the fields for a row that have data. HBase tables will often have a large amount of NULL value cells, as most rows will not share all of the columns that are filled other rows.



HBase is well suited for a data set that needs to be flexible. Since HBase stores data in binary form, we can store whatever data we need to in our table, access it through a key value pair, and not have to worry about the structure of the data.

Define Data Types



cust_id	name	state	gender	age	agg_rev	membership	reviews
16841	Lisa Wells	tx	FEMALE	26-35	40	basic	NULL
19905	Neely Nova	va	MALE	15-20	45	basic	NULL
17718	William Bush	il	FEMALE	51-100	29	basic	NULL
21975	Gerald Staples	ma	MALE	26-35	28	basic	TiqA9ybgewk

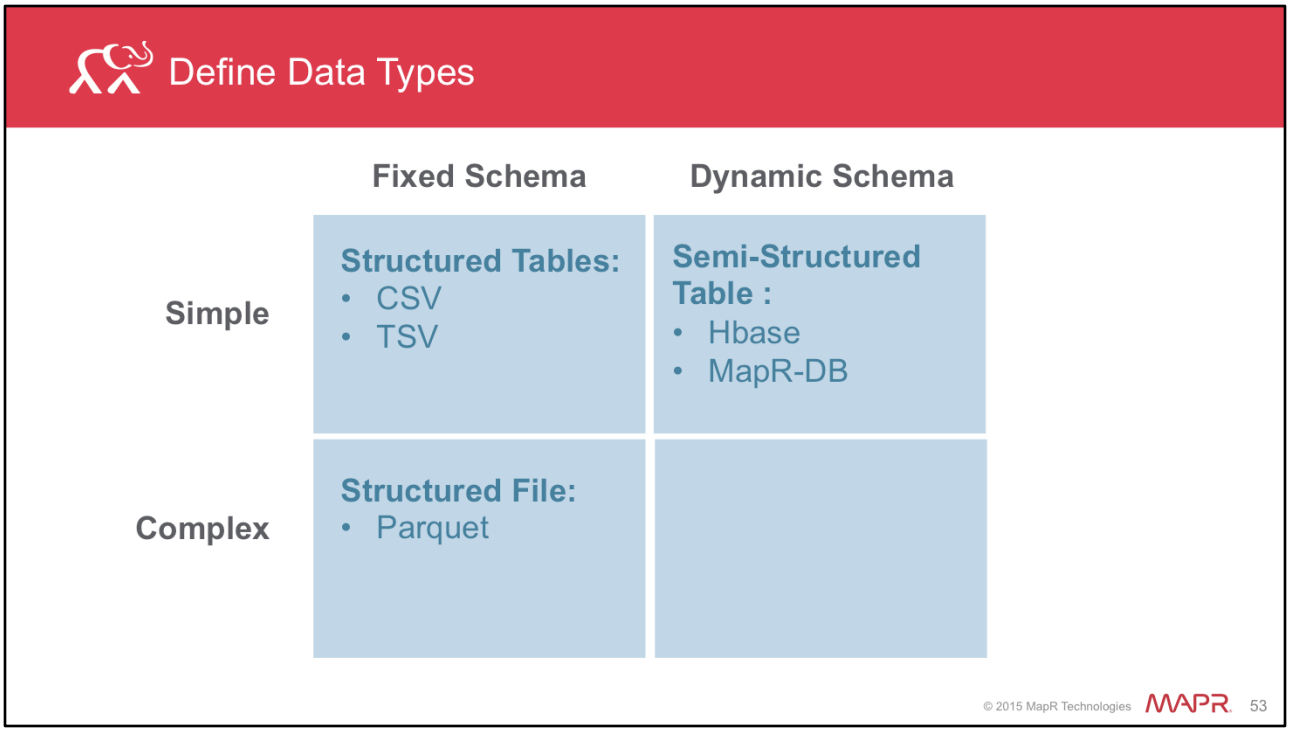
We are using HBase for our customer and product databases. We can store text data, metadata, customer or product images and anything else about our customer and our products all in the same database. We also have the flexibility to add new types of data, such as advanced customer security or product videos, as they come available.

Define Data Types

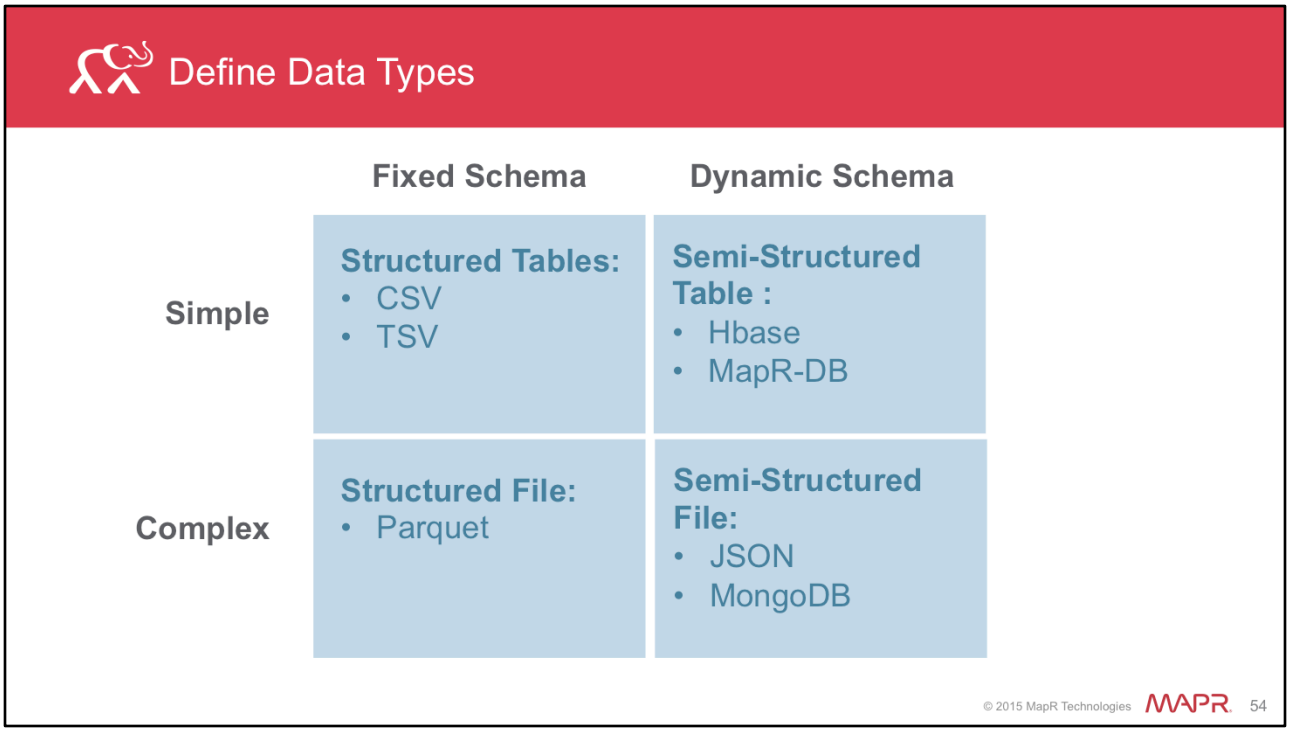
Data Preview:

	trans_id	date	time	user_info	trans_info
1	31920	2014-04-26	12:17:12	{ "cust_id" : 22526, "device" : "IOS5", "state" : "il"}	{ "prod_id" : [174, 2], "purch_flag" : "false"}
2	31026	2014-04-20	13:50:29	{ "cust_id" : 16368, "device" : "AOS4.2", "state" : "nc"}	{ "prod_id" : [ ], "purch_flag" : "false"}
3	33848	2014-04-10	04:44:42	{ "cust_id" : 21449, "device" : "IOS6", "state" : "oh"}	{ "prod_id" : [582], "purch_flag" : "false"}
4	32383	2014-04-18	06:27:47	{ "cust_id" : 20323, "device" : "IOS5", "state" : "oh"}	{ "prod_id" : [710, 47], "purch_flag" : "false"}
5	32359	2014-04-19	23:13:25	{ "cust_id" : 15360, "device" : "IOS5", "state" : "ca"}	{ "prod_id" : [0, 8, 170, 173, 1, 124, 46, 764, 30, 711, 0, 3, 25], "purch_
6	30422	2014-04-23	01:46:05	{ "cust_id" : 15957, "device" : "IOS7", "state" : "sc"}	{ "prod_id" : [628], "purch_flag" : "false"}
7	35898	2014-04-18	13:28:56	{ "cust_id" : 20677, "device" : "IOS7", "state" : "ny"}	{ "prod_id" : [ ], "purch_flag" : "false"}
8	32421	2014-04-15	11:00:53	{ "cust_id" : 23599, "device" : "IOS5", "state" : "ri"}	{ "prod_id" : [1, 0, 87, 170, 445, 6, 0, 2, 7, 7, 110], "purch_flag" : "false
9	39447	2014-04-21	01:34:06	{ "cust_id" : 16122, "device" : "IOS6", "state" : "fl"}	{ "prod_id" : [26, 504, 1, 47, 31, 116, 167, 71, 20, 305, 554, 0, 189, 384,
10	35373	2014-04-02	06:53:28	{ "cust_id" : 15342, "device" : "IOS5", "state" : "ms"}	{ "prod_id" : [15, 0], "purch_flag" : "false"}
11	30778	2014-04-13	01:20:05	{ "cust_id" : 16996, "device" : "AOS4.2", "state" : "oh"}	{ "prod_id" : [40, 499, 15], "purch_flag" : "true"}
12	32120	2014-04-28	06:58:19	{ "cust_id" : 21402, "device" : "IOS5", "state" : "nm"}	{ "prod_id" : [ ], "purch_flag" : "false"}
13	30706	2014-04-15	21:08:15	{ "cust_id" : 15344, "device" : "IOS7", "state" : "ny"}	{ "prod_id" : [27], "purch_flag" : "false"}
14	38207	2014-04-28	18:57:55	{ "cust_id" : 16841, "device" : "AOS4.2", "state" : "mo"}	{ "prod_id" : [181, 0, 33, 708, 92], "purch_flag" : "false"}
15	32955	2014-04-15	20:41:58	{ "cust_id" : 15492, "device" : "AOS4.4", "state" : "wi"}	{ "prod_id" : [242, 0, 2, 3, 166, 26, 0, 0], "purch_flag" : "false"}

In our second set of exercises, we also queried web click data that is stored as parquet and JSON files. Parquet stores data in a columnar format, and was designed to handle complex, nested data in a Hadoop framework.



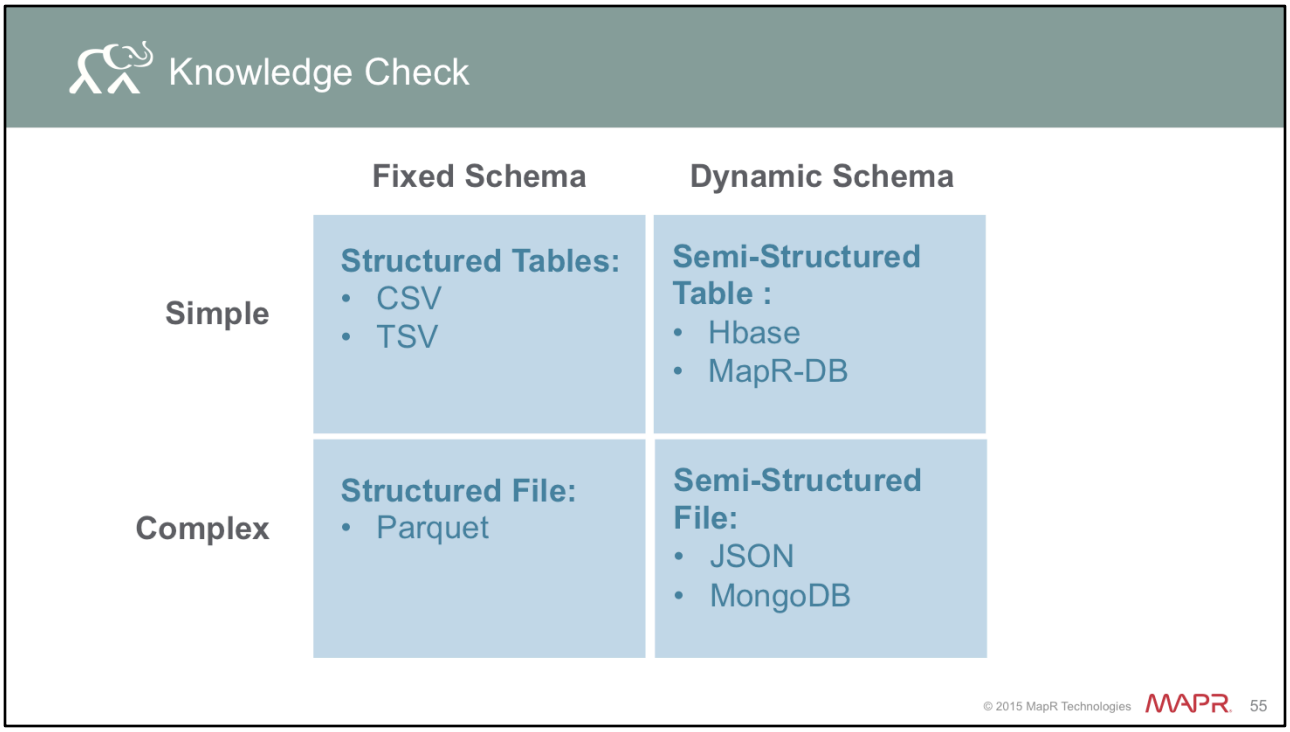
Parquet uses a pre-defined, fixed schema, similar to our structured tables. Parquet files are very fast to query. Since we frequently query our user product search logs, we periodically convert these files into parquet format for faster query times.



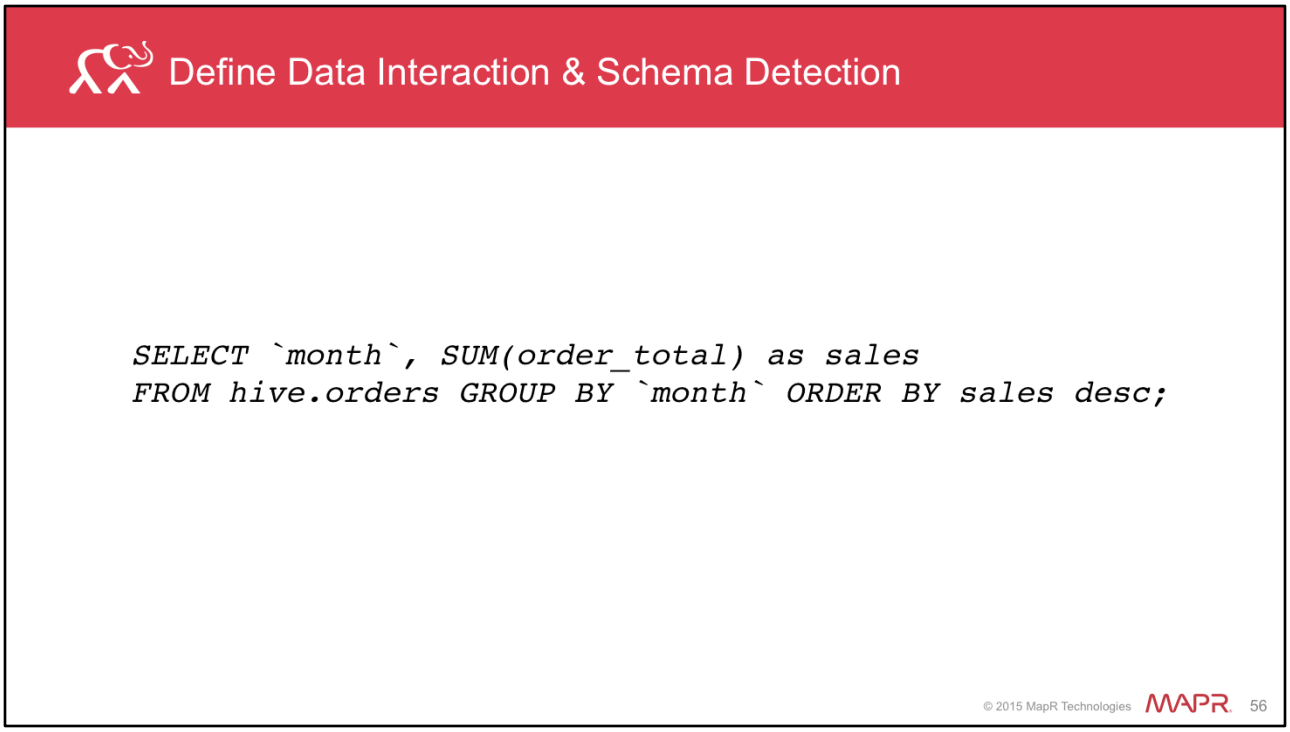
We also keep our click data in JSON format. JSON is the most flexible of our file formats. JSON files can be complex and nested, like our parquet files, and also being schema-less, like our HBase tables.
With Drill we can, and have used familiar ANSI SQL to query data that fits into any of these categories. Drill gives us unparalleled ease and flexibility to query our companies data.

(highlight the entire table)
(rollover quadrants of the table to show:
upper left: Flat, structured data is fast to query, but very inflexible to changes. Best suited for known data sets that will not change.
upper right: Semi-structured or NoSQL tables are very flexible, and still provide structure to our data. Best suited for very large

Knowledge Check

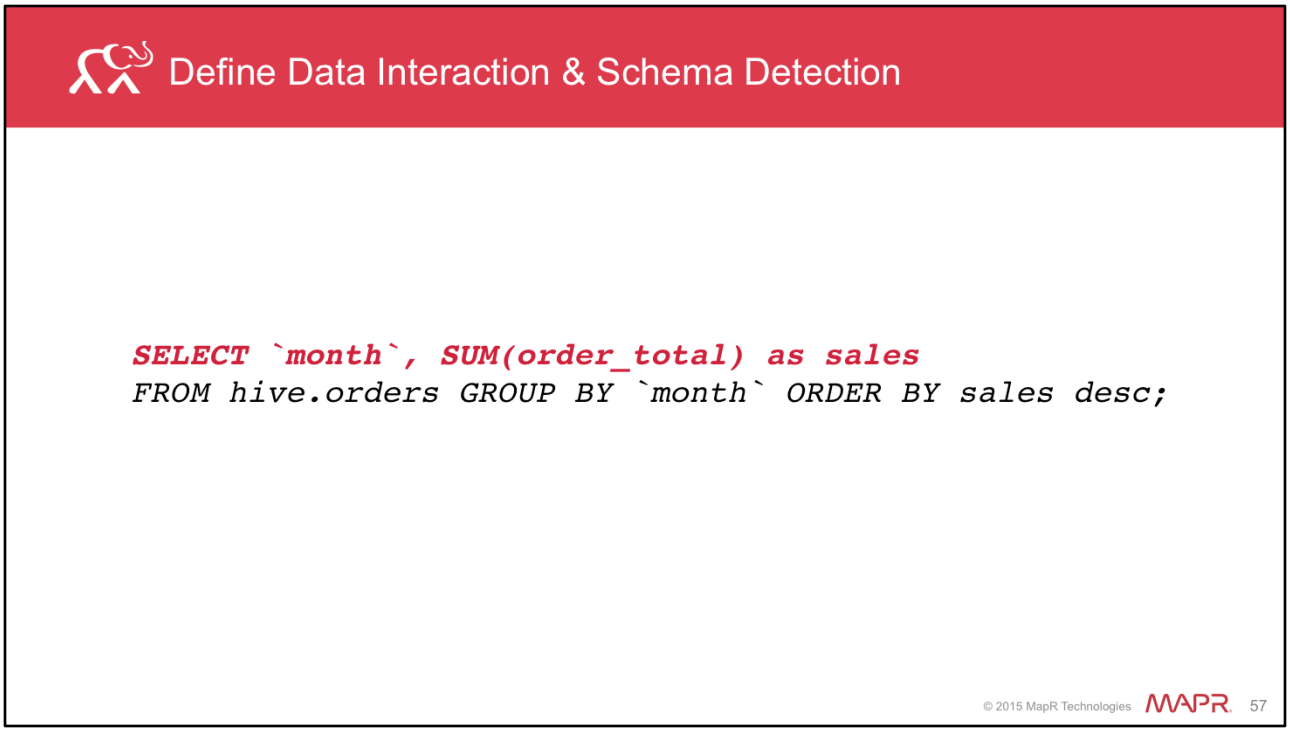


Which quadrants contain data types that Drill can use to run SQL queries?
Which quadrants contain data types that other SQL tools can use SQL to run queries?

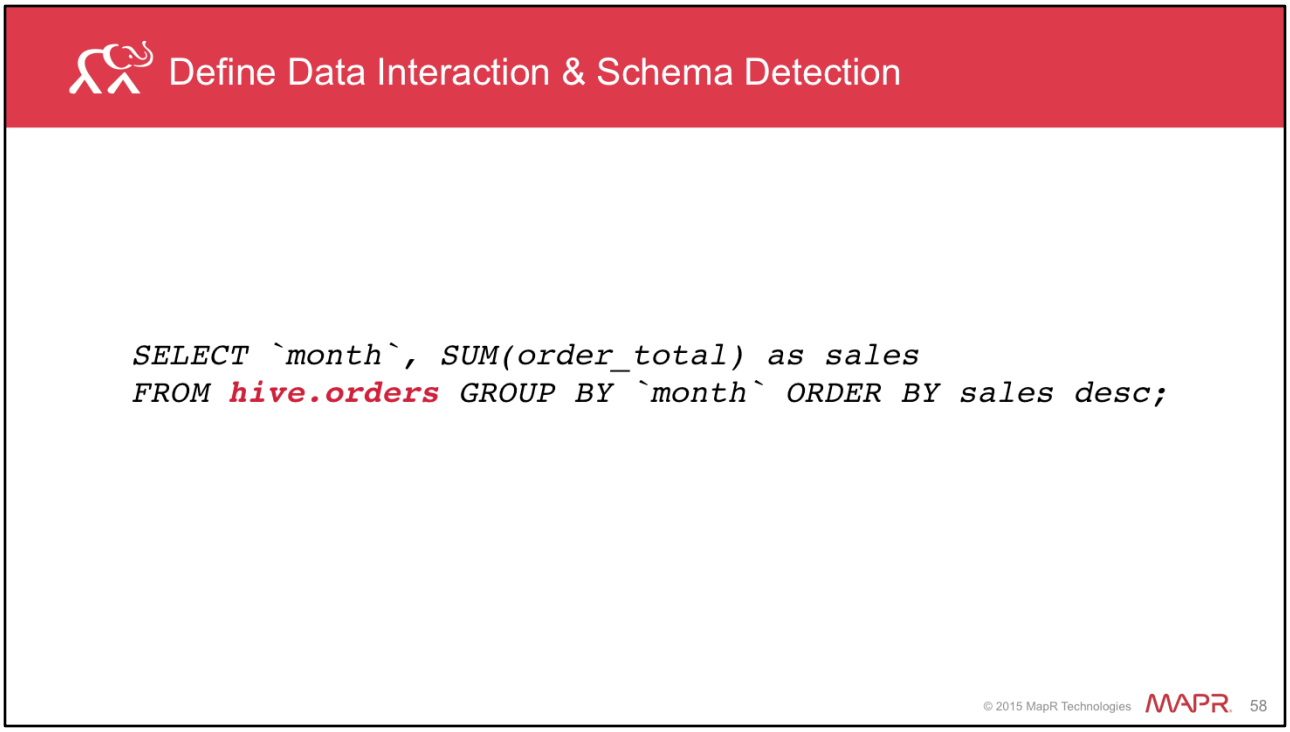


When we look more closely at our queries, we can see how drill interacts with each of these different types of data.

For example, going back to our first query…

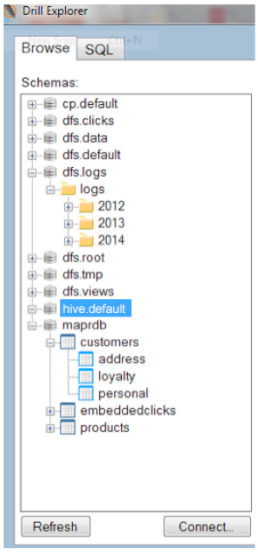


the SELECT statement defines what data we are pulling out of the table, and what action we are going to perform on it, just like we are used to from any standard SQL query.



In the FROM statement, however, we start to see how Drill interacts with our data sources. hive.orders tells Drill where to find the data we are requesting in the select statement.

Define Data Interaction & Schema Detection

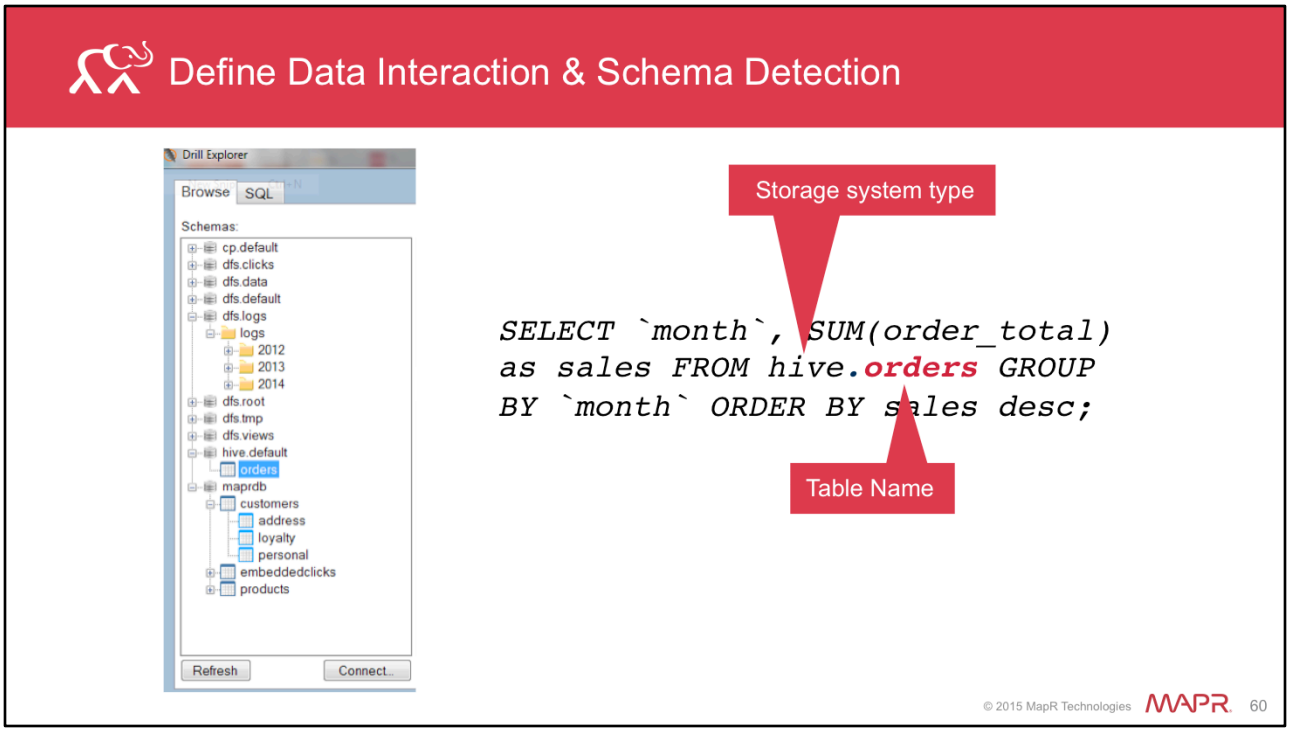


Storage system type

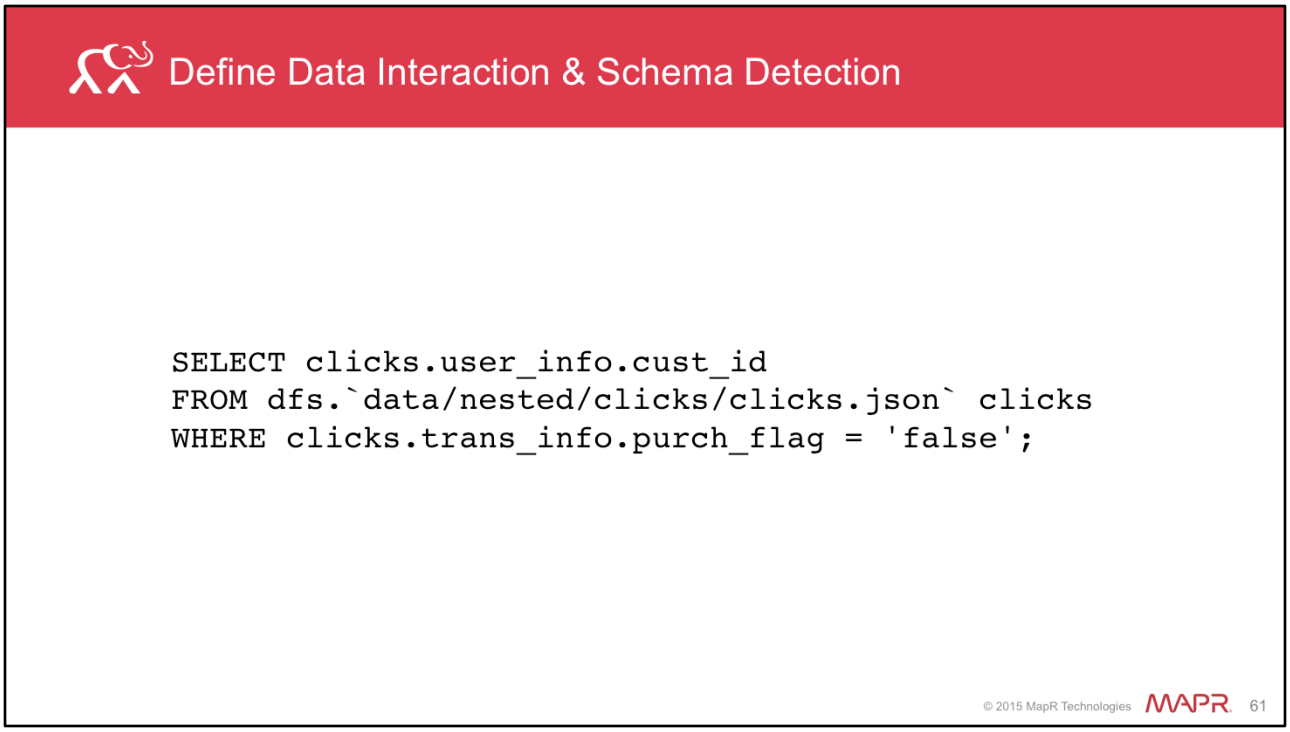
```
SELECT `month`, SUM(order_total)
as sales FROM hive.orders GROUP
BY `month` ORDER BY sales desc;
```

 59

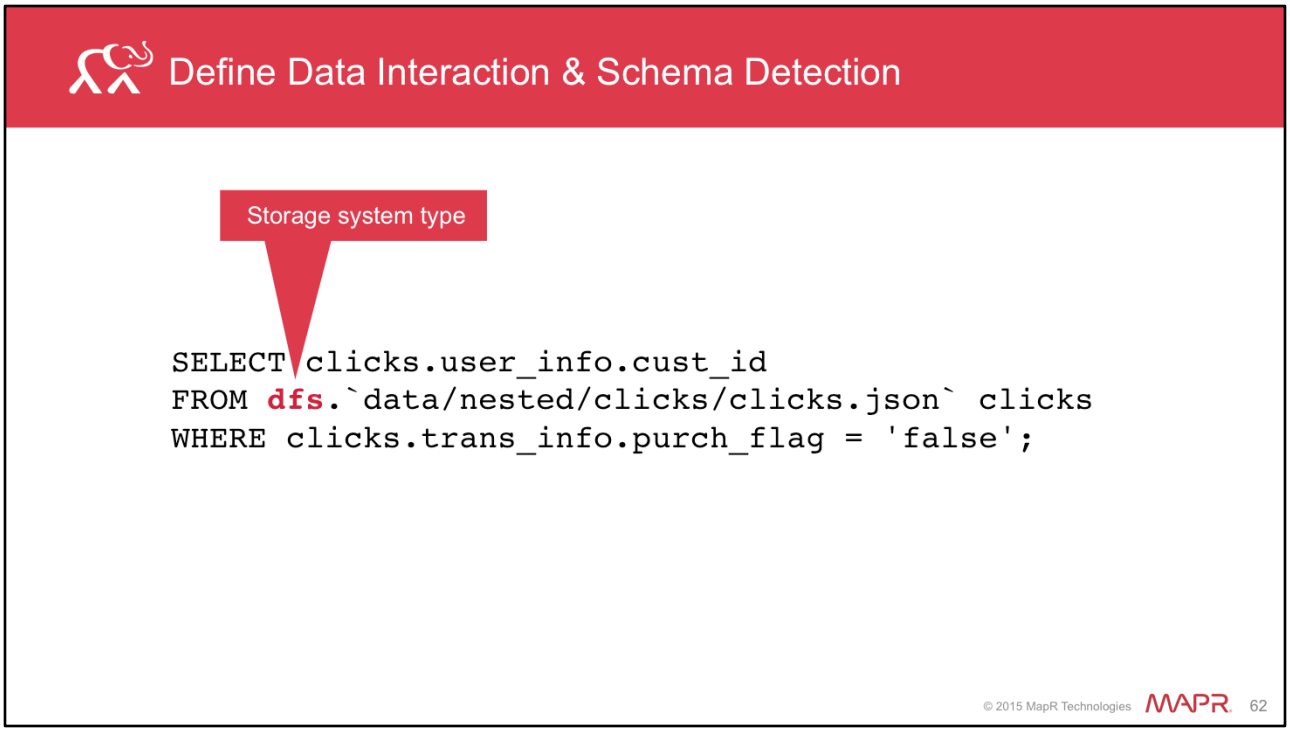
In this case, hive points Drill to the Hive storage on our cluster.



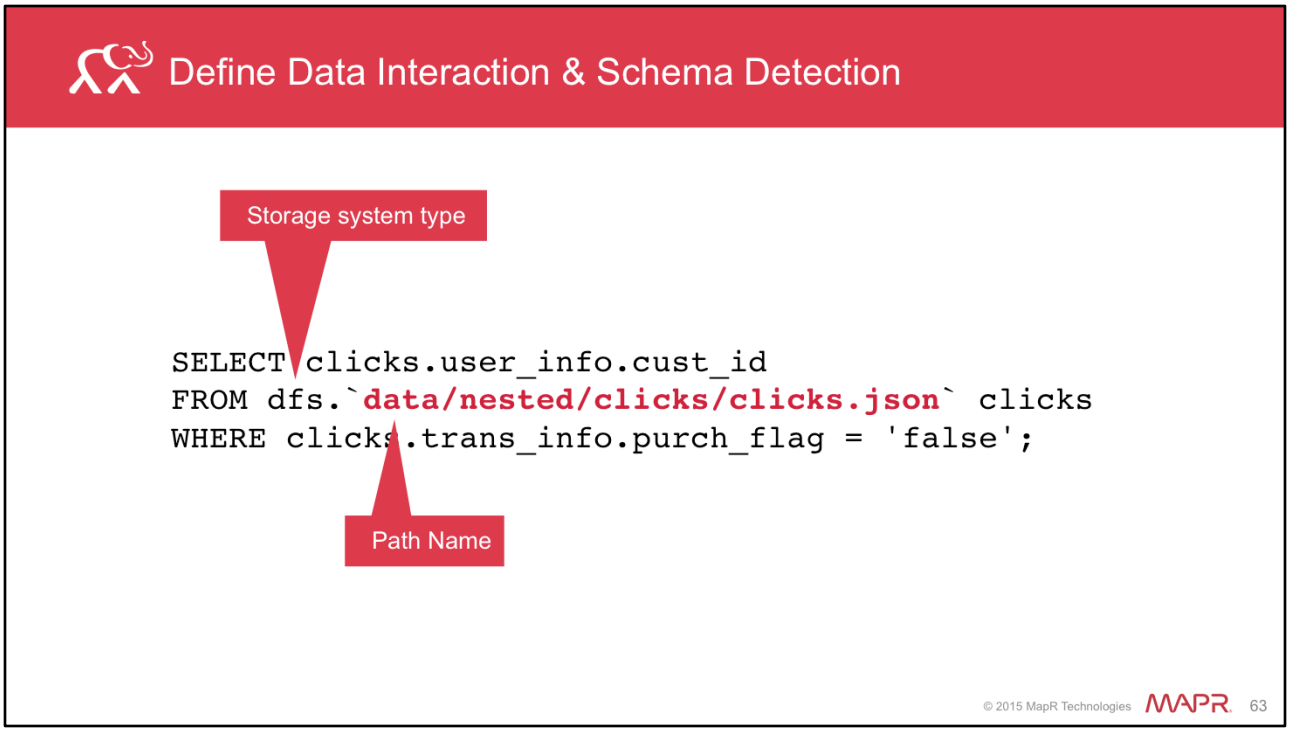
and then 'orders' is the name of the table from which we are requesting data.



Then, examining the fifth query we made,



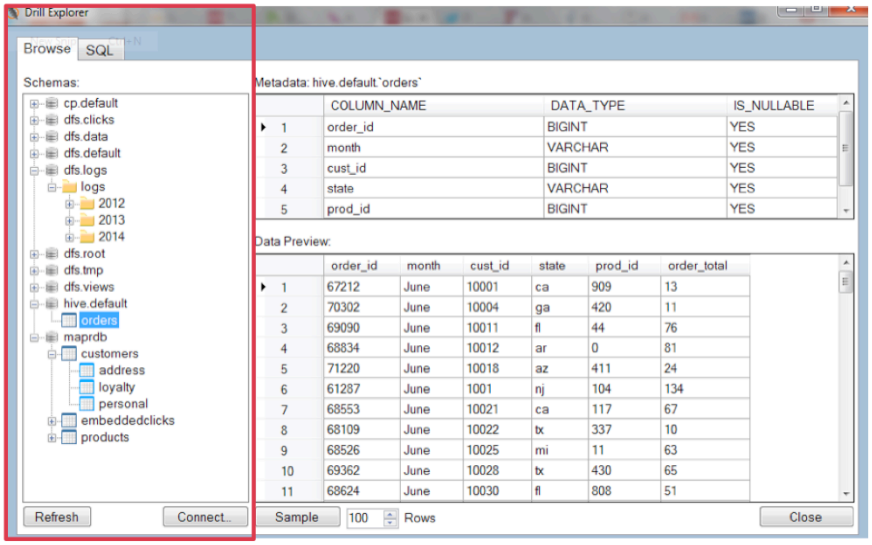
the FROM statement points Drill to the DFS filesystem



and then `data/nested/clicks/clicks.json` provides the pathname to the file where Drill can find the data we are requesting.

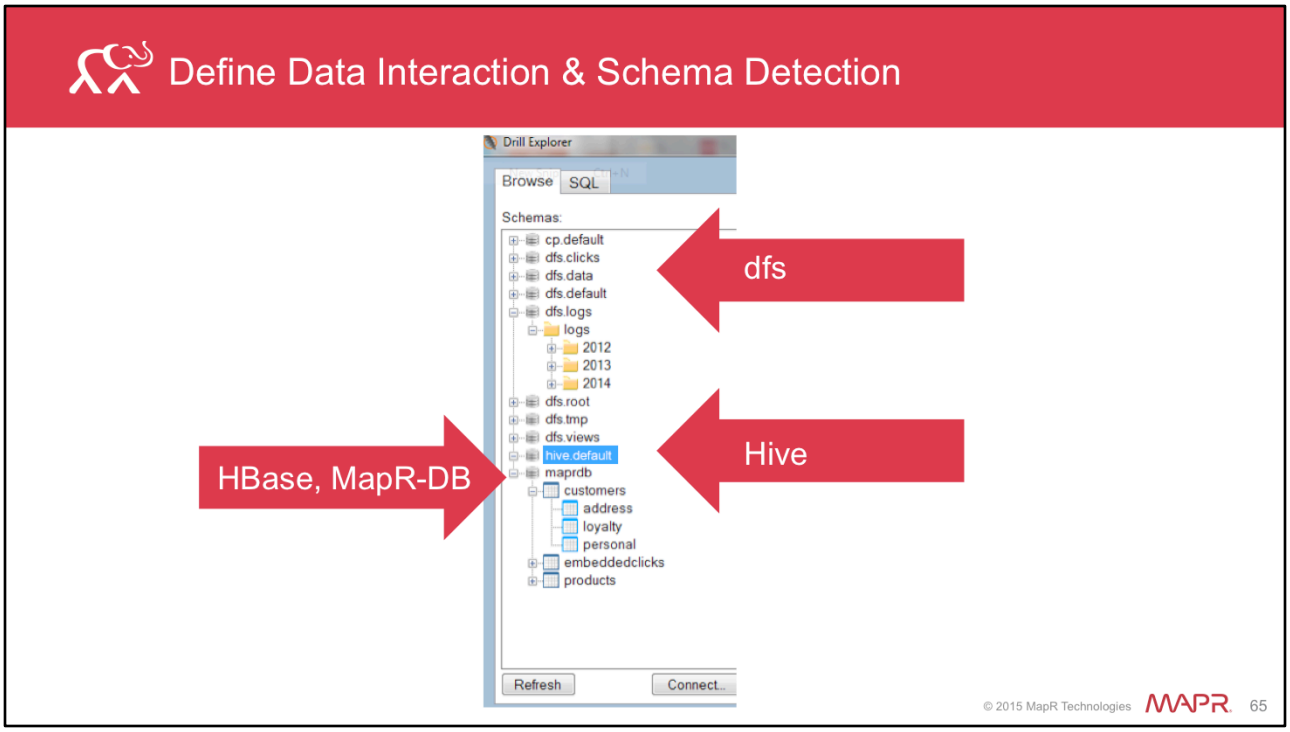
With Drill, we don't have to connect to the database prior to submitting a query. All we have to do is include the data source location in the FROM statement of our query, and Drill finds the data we are requesting.

Define Data Interaction & Schema Detection



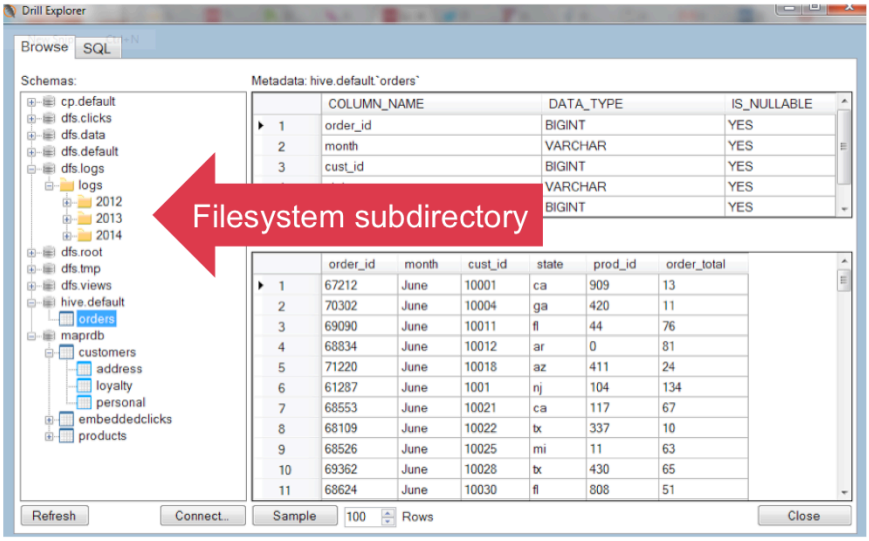
 MAPR 64

Looking at the Drill explorer, we can see the different data sources that Drill can query in this way.



The storage system type can be a data base like Hive, HBase or MapR-DB, or a distributed filesystem like HDFS or MapR-FS.

Define Data Interaction & Schema Detection



 66

We can include a workspace, which can be a hive database, an HBase namespace, or filesystem subdirectory.

Define Data Interaction & Schema Detection

```
SELECT `month`, SUM(order_total) as sales
FROM hive.orders GROUP BY `month` ORDER BY sales desc;
```

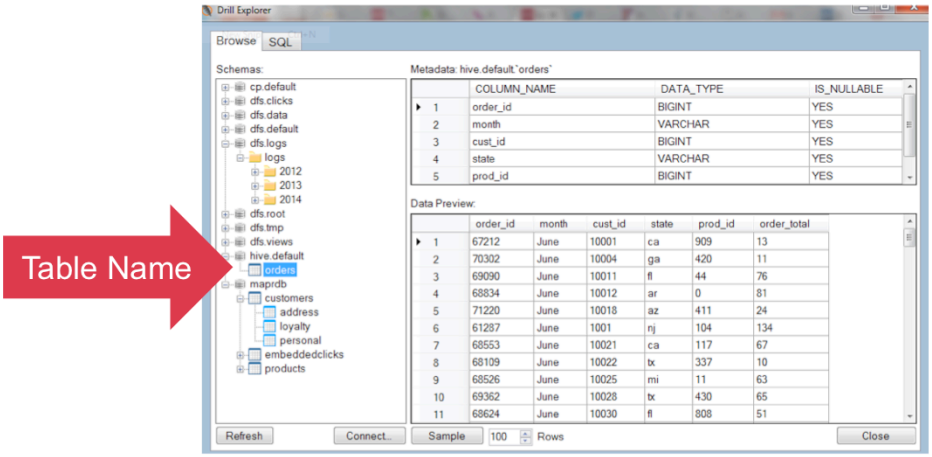
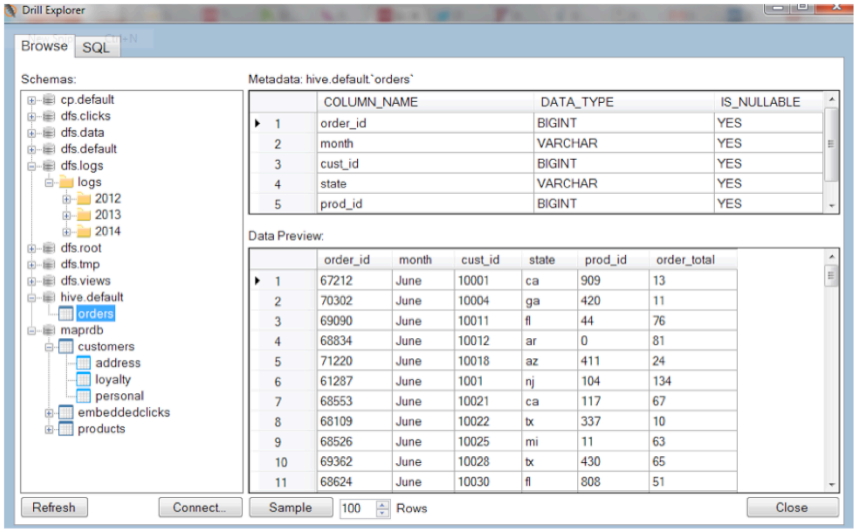


Table Name

and finally we include the path to the table or file that we want to query.

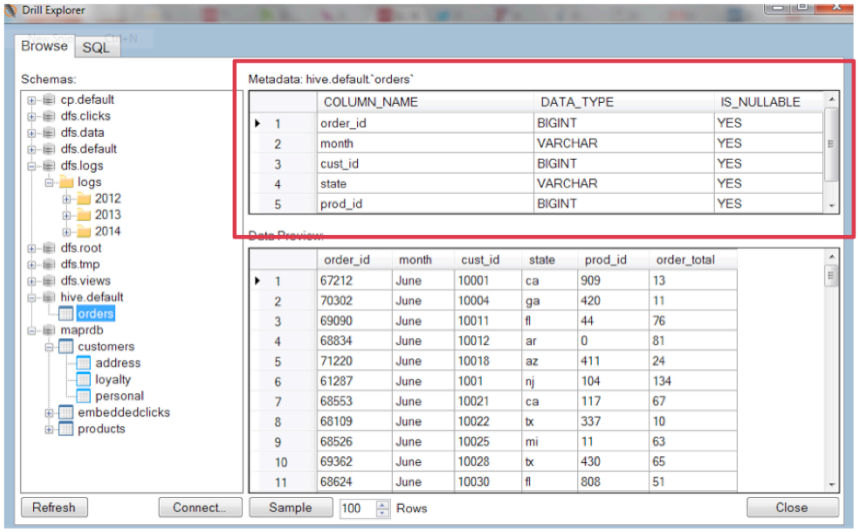
Define Data Interaction & Schema Detection



 MAPR 68

The Drill Explorer gives us some further insight into how Drill discovers the schema of the data available to us. In the previous lesson, we introduced the idea of exploring data. Exploring data is a very powerful tool for our data analysts when looking at unknown tables, or data streams with dynamic schemas.

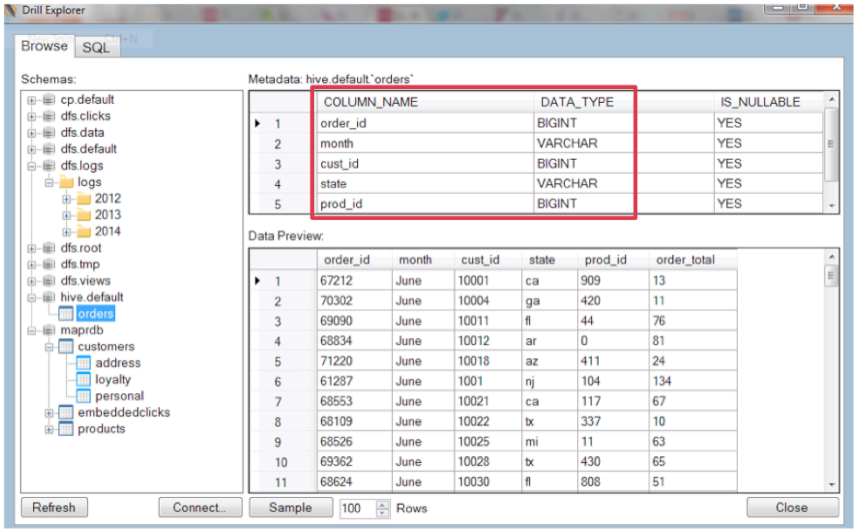
Define Data Interaction & Schema Detection



 69

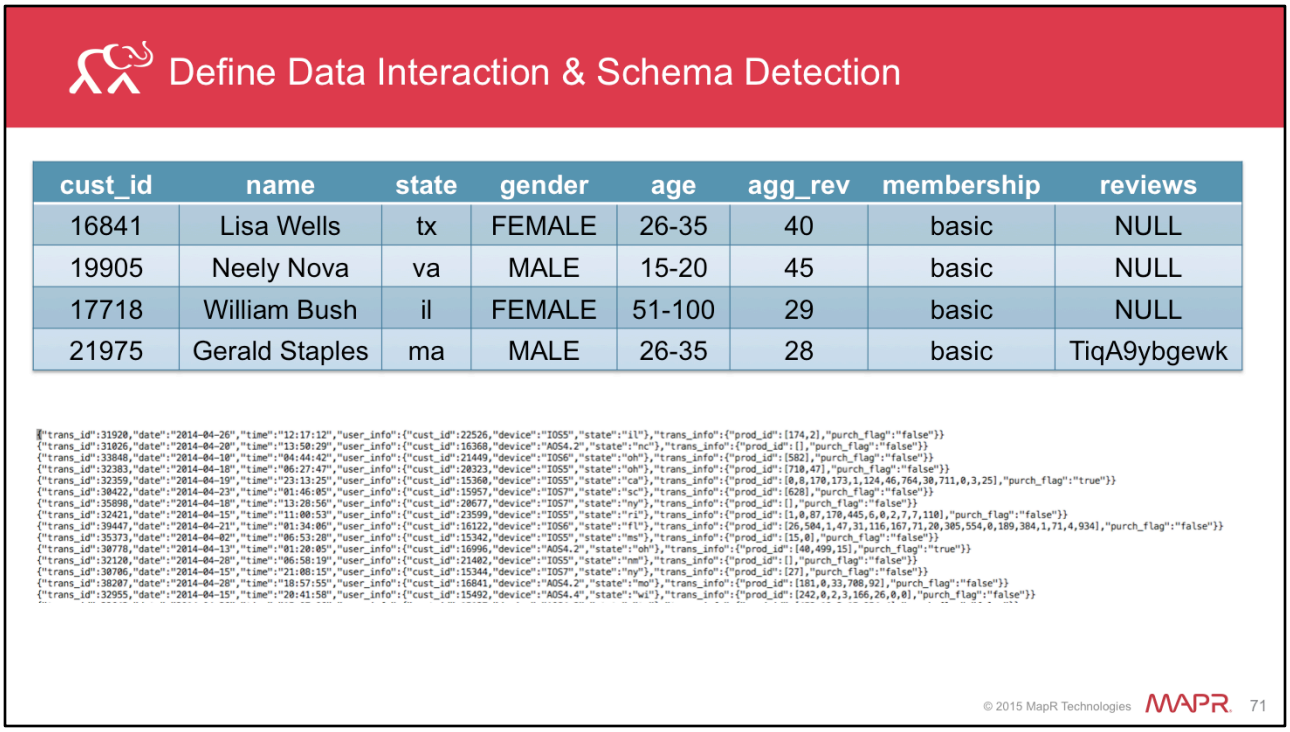
For example, if we have unknown structured data, our data analysts can first use the Drill explorer to learn about the data before writing queries. When we look at our orders table stored in Hive, we can see that Drill is able to learn the column names, the data types and whether the data is nullable. We did not tell Drill the table schema, and normally we would have to execute HiveQL statements to learn the schema of a table.

Define Data Interaction & Schema Detection

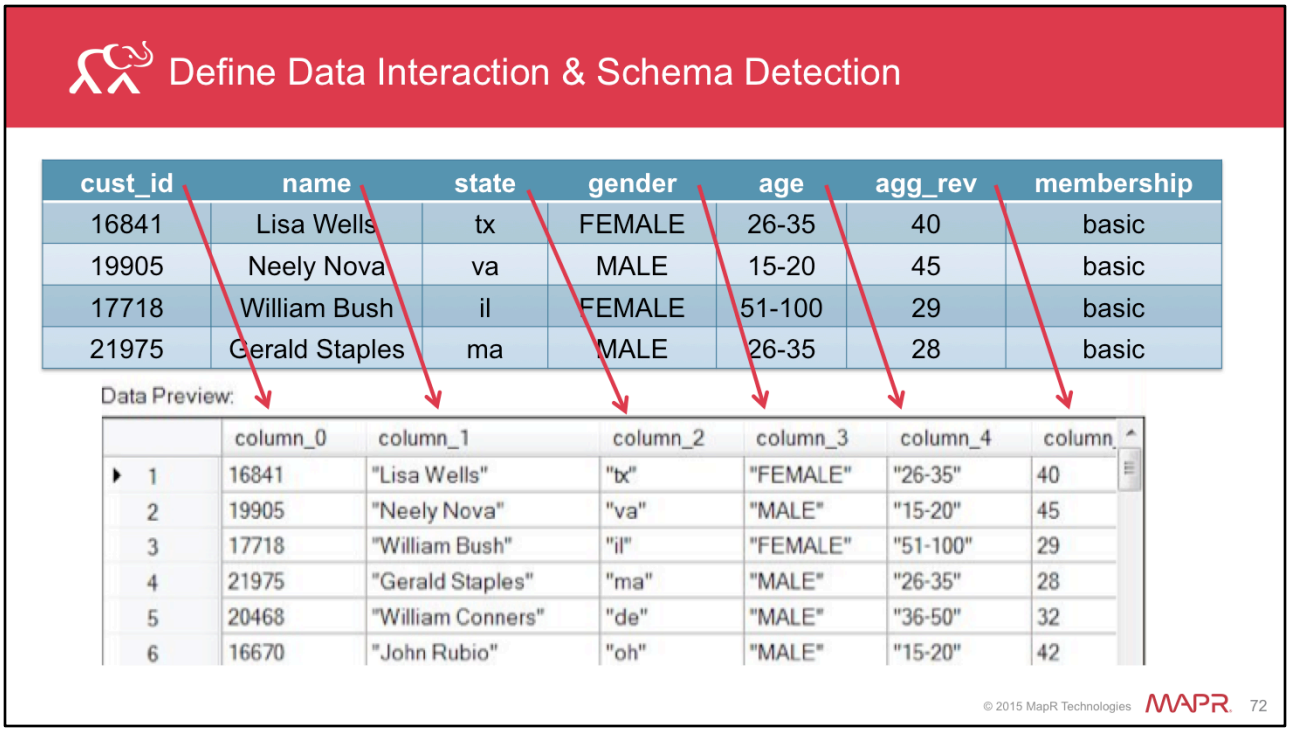


 70

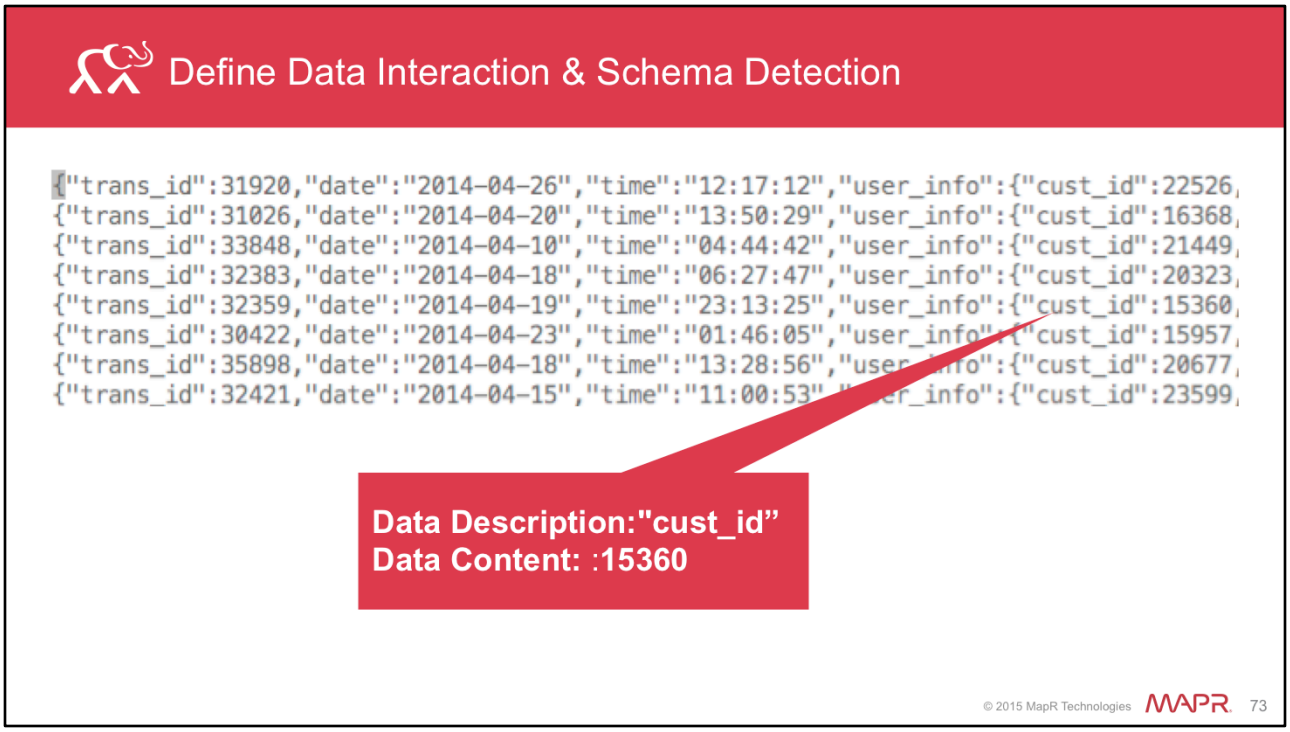
Drill uses the HCatalog Connector to access the HCATALOG_COLUMNS metadata. Drill discovers the schema and shows us the table names and data types. Drill does the work for us, showing us the data structure and data types, so that we can then write the queries that we need.



At the Big Office Supply Company, our data sources also include data streams that have a dynamic schema. Our HBase tables and JSON click data are both examples of data that is dynamic, as it changes its schema over time.

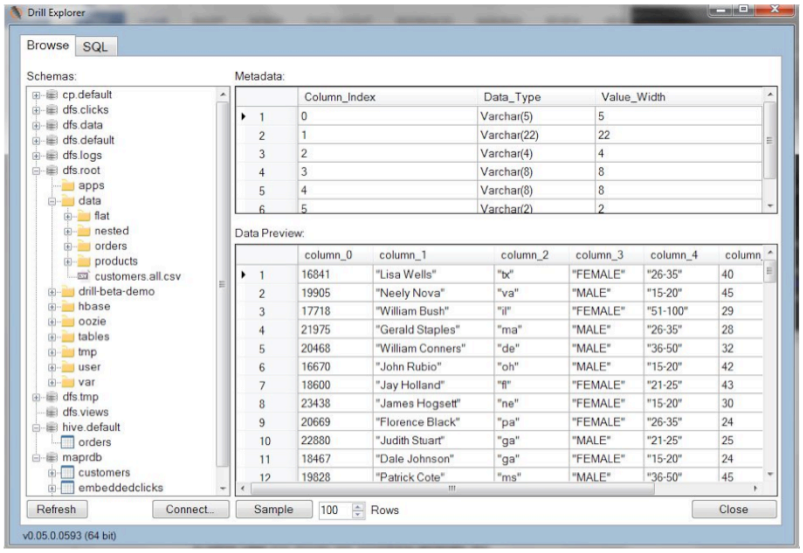


When exploring HBase or MapR-DB, Drill reads the table column headers, which are saved as text. We can see all of the column headers currently in the table when using the Drill Explorer. Since these columns are read at the time we select the data source, they are always current. As with the Hive table, Drill is doing the work for us, and we do not need to execute any code to view the dynamic schema.



With text based docs like our JSON click logs, Drill reads the schema from the document itself. JSON is self describing data, meaning that each data element contains the data description preceding the data content. Drill will read these data description elements, and present them to us as schema metadata.

Define Data Interaction & Schema Detection



Exploring the data with Drill Explorer gives us the data schema information that we need to make effective queries of data that is unknown or dynamic in structure.

Knowledge Check

Match the following definitions

A. `hive.orders`

B. `Dfs.'data/nested/clicks/clicks.json'`

C. `HCATALOG_COLUMNS`

D. `"user_info":{"cust_id":5324,"device":"AOS4.3","state":"ca"}`

__ Tells Drill what table to query

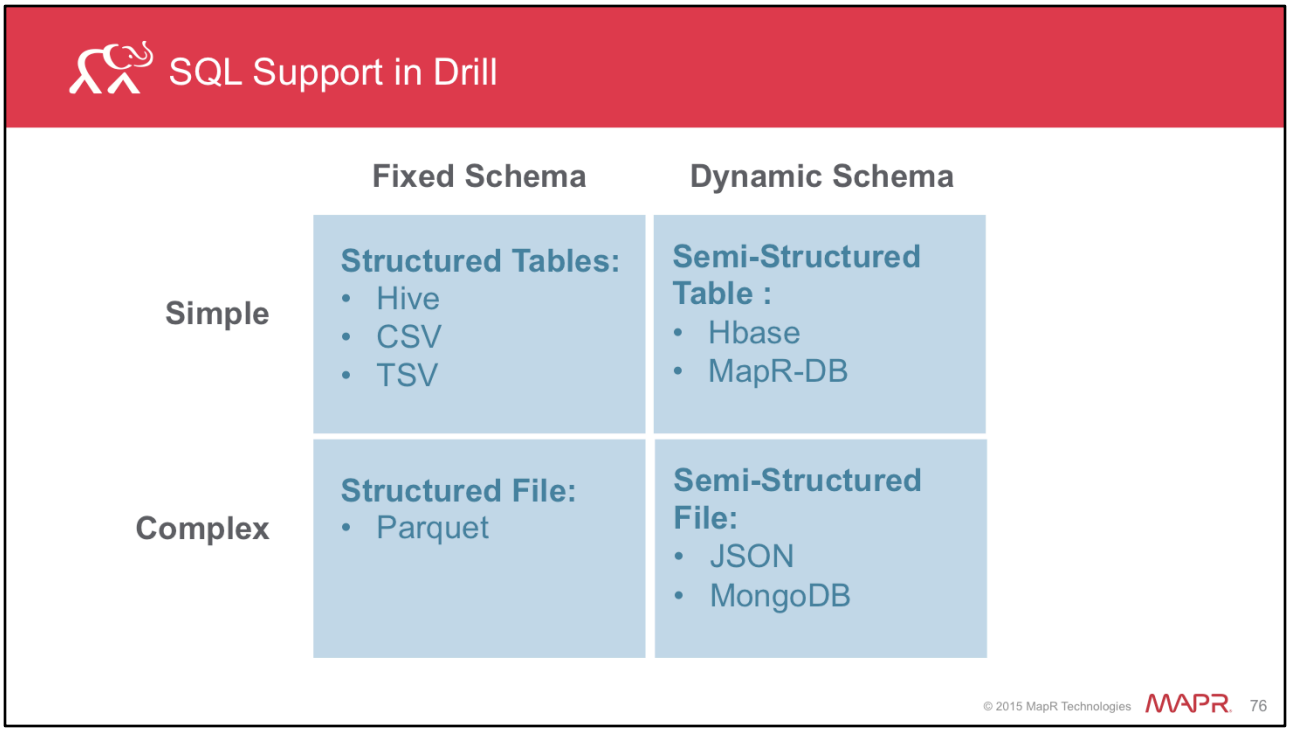
__ Metadata used to discover the schema of a Hive table

__ Tells Drill what file to query

__ Self describing JSON data

 75

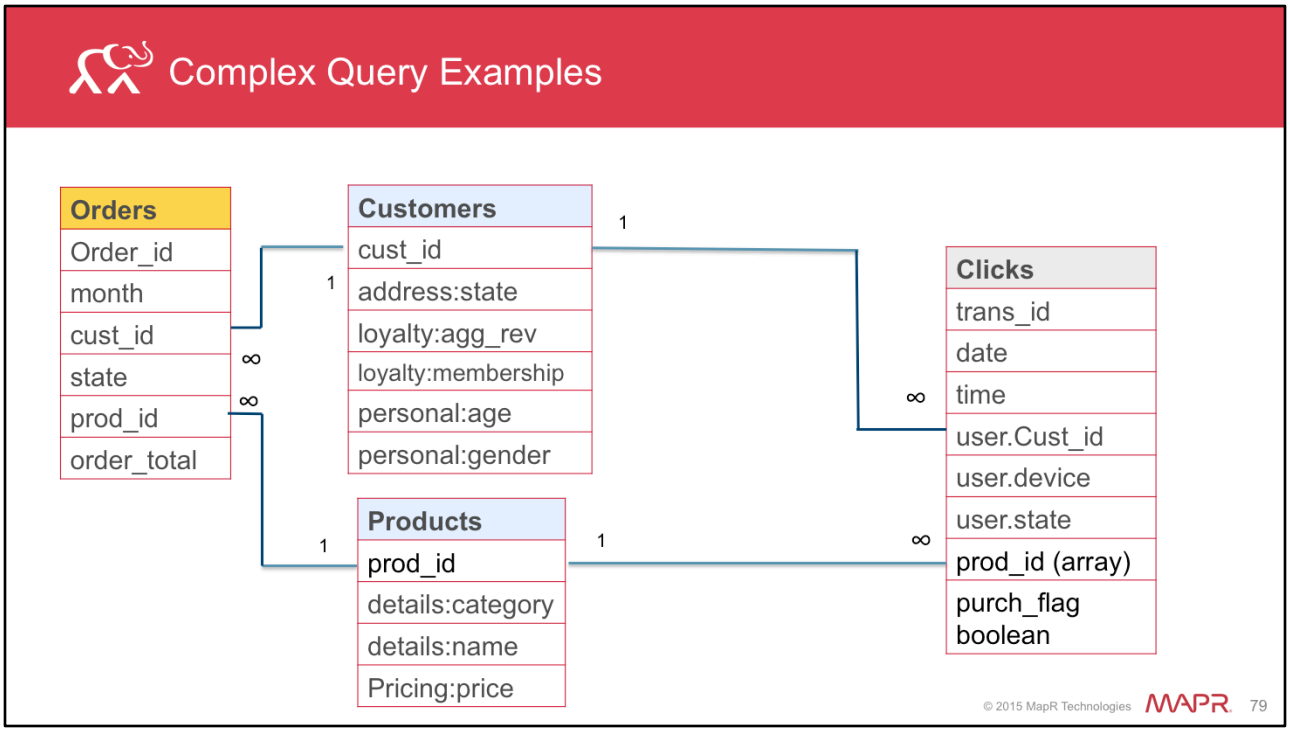
A
C
B
D



Drill makes it easy for us to query a variety of simple, complex, unknown and dynamic data sources. We can use Drill to explore our data, keeping our queries current and efficient as our data changes.

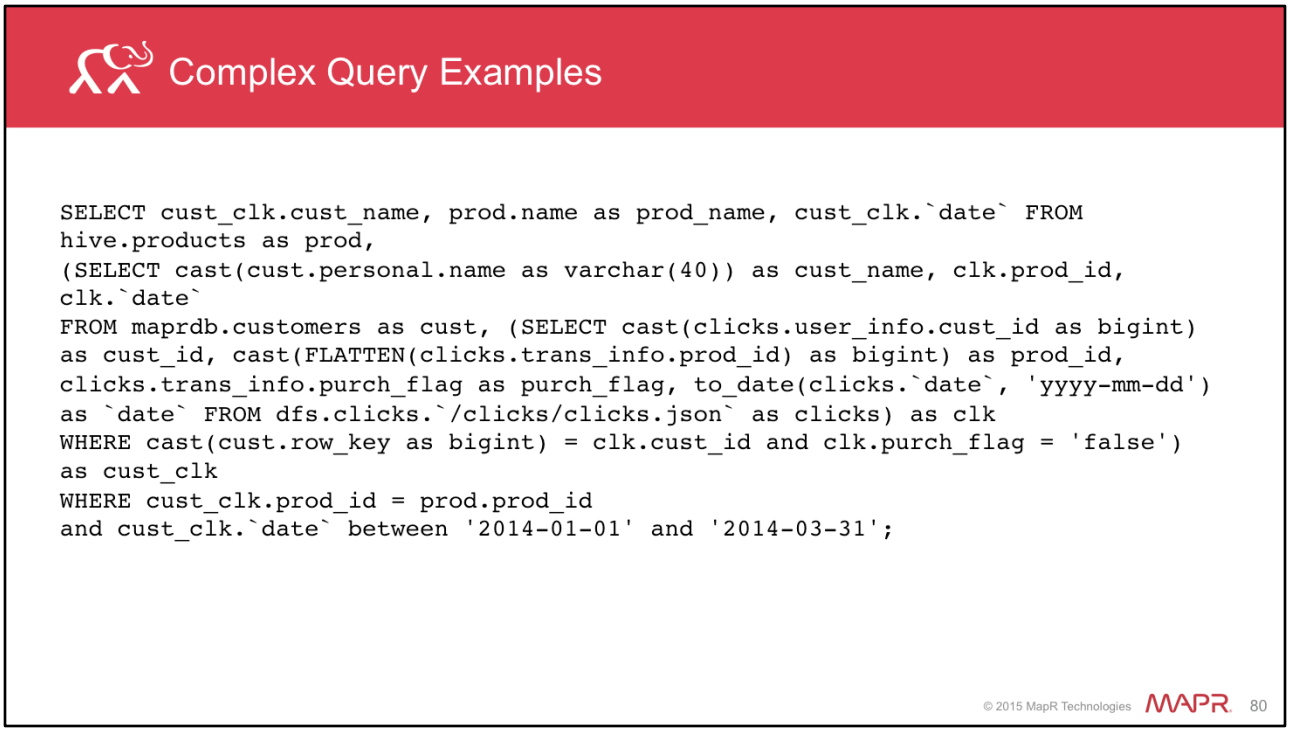
- Drill supports ANSI SQL – theory on the query we are about to show
- List key data types
- Key operators – joins, aggregators, functions
- Show a couple of sample queries, doing joins, but without complex data

- Extending ANSI SQL to work with complex types such as arrays and maps
- Do an example with flatten and KV Gen
- Show convert to/from – steer away from cast



Some of the most useful insights to our company, however, are gained when we combine data across multiple sources, such as combining our click log data with customer locations or purchasing habits.

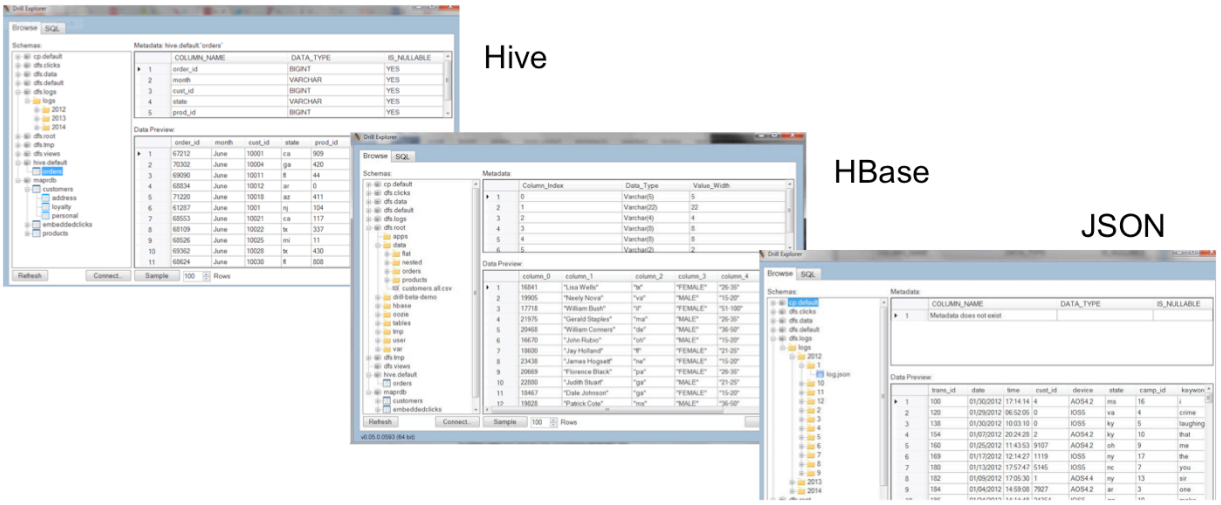
We store all of these different data sources in different formats, to best suit the type of data in them. They also all have different schemas, some of which are known and fixed, while others are dynamic and constantly changing.



In our final query in Lesson 1, we joined our structured Hive orders table, with our semi structured HBase customers table and JSON click logs.

When we examine this query, we can see how Drill provides us with the information we need to first build the query, and then the power to execute a single query across all of these different types of data.

Complex Query Examples



We knew that we wanted to perform a query that combined information from three different sources of data. Prior to writing the query, we used the Drill Explorer to view these three data sources, learning their current schema and how the data is stored.

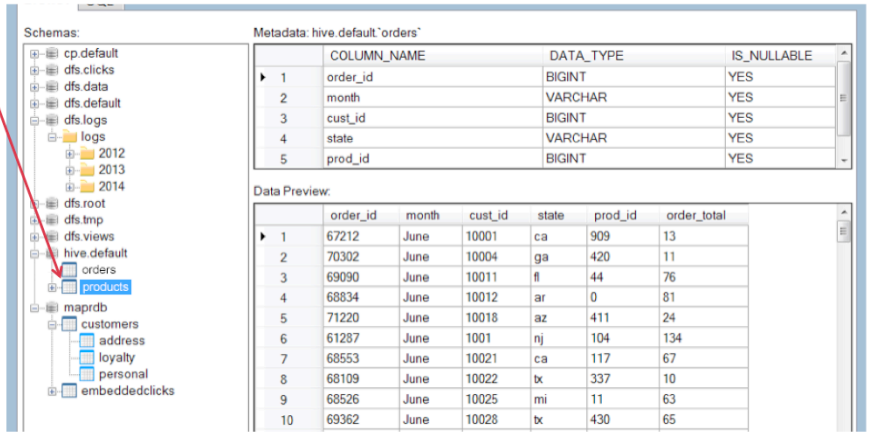
Complex Query Examples

```
SELECT cust_clk.cust_name, prod.name as prod_name, cust_clk.`date` FROM
hive.products as prod,
(SELECT cast(cust.personal.name as varchar(40)) as cust_name, clk.prod_id,
clk.`date`
FROM maprdb.customers as cust, (SELECT cast(clicks.user_info.cust_id as bigint)
as cust_id, cast(FLATTEN(clicks.trans_info.prod_id) as bigint) as prod_id,
clicks.trans_info.purch_flag as purch_flag, to_date(clicks.`date`, 'yyyy-mm-dd')
as `date` FROM dfs.clicks.`/clicks/clicks.json` as clicks) as clk
WHERE cast(cust.row_key as bigint) = clk.cust_id and clk.purch_flag = 'false')
as cust_clk
WHERE cust_clk.prod_id = prod.prod_id
and cust_clk.`date` between '2014-01-01' and '2014-03-31';
```


Once we learned how to access the data, and what types of data we were querying, we can then build a single query that joins all of these sources; simple, complex, structured and dynamic, into this single query.

Complex Query Examples

```
SELECT cust_clk.cust_name, prod.name as prod_name, cust_clk.`date`
FROM hive.products as prod
```

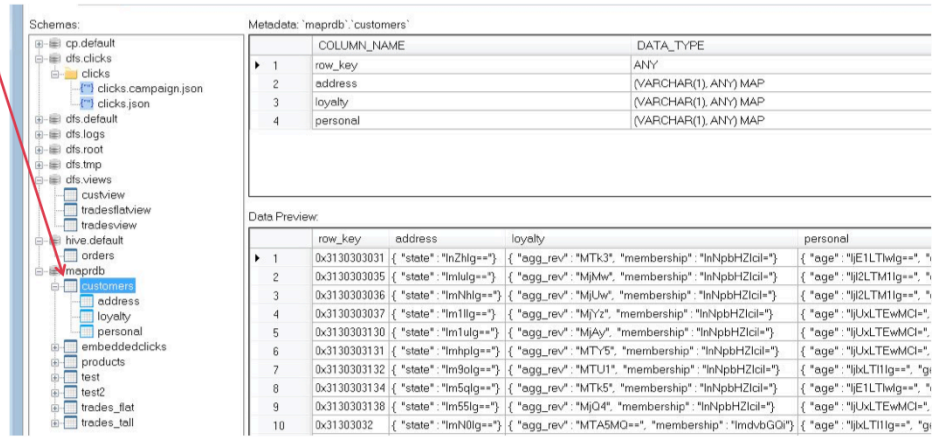


 83

Highlight 3 Select statements – We nested the SELECT statements, each targeting one of our data sources. Once we learned where each piece of data is stored, we can then map how we will join our data sources.

Complex Query Examples

```
SELECT cast(cust.personal.name as varchar(40)) as cust_name,
clk.prod_id, clk.`date`
FROM maprdb.customers as cust
```

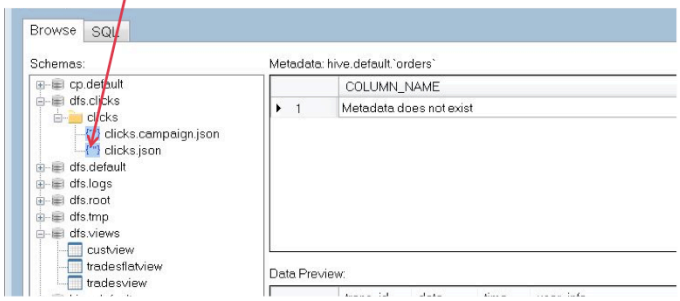


 84

Highlight 3 Select statements – We nested the SELECT statements, each targeting one of our data sources. Once we learned where each piece of data is stored, we can then map how we will join our data sources.

Complex Query Examples

```
SELECT cast(clicks.user_info.cust_id as bigint) as
cust_id, cast(FLATTEN(clicks.trans_info.prod_id) as
bigint) as prod_id, clicks.trans_info.purch_flag as
purch_flag, to_date(clicks.`date`, 'yyyy-mm-dd') as
`date` FROM dfs.clicks.`/clicks/clicks.json` as clicks
```



Highlight 3 Select statements – We nested the SELECT statements, each targeting one of our data sources. Once we learned where each piece of data is stored, we can then map how we will join our data sources.

Complex Query Examples

```
cast(cust.personal.name as varchar(40))
```

Metadata: `mapr.db`.`customers`

	COLUMN_NAME	DATA_TYPE	IS_NULLABLE
1	row_key	ANY	NO
2	address	(VARCHAR(1), ANY) MAP	NO
3	loyalty	(VARCHAR(1), ANY) MAP	NO
4	personal	(VARCHAR(1), ANY) MAP	NO

 86

Highlight ‘cast’ functions – When we explored the data before writing this query, we were able to determine the type of data in each part of the query. Based on this information, we are able to cast each part of our query into the appropriate type.

Complex Query Examples

```
cast(clicks.user_info.cust_id as bigint)
```

Metadata: hive.default.`orders`

	COLUMN_NAME	DATA_TYPE	IS_NULLABLE
1	order_id	BIGINT	YES
2	month	VARCHAR	YES
3	cust_id	BIGINT	YES
4	state	VARCHAR	YES
5	prod_id	BIGINT	YES

Data Preview:

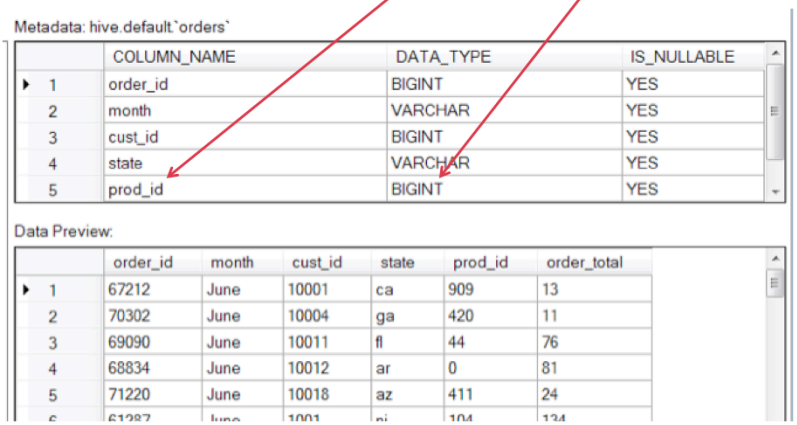
	order_id	month	cust_id	state	prod_id	order_total
1	67212	June	10001	ca	909	13
2	70302	June	10004	ga	420	11
3	69090	June	10011	fl	44	76
4	68834	June	10012	ar	0	81
5	71220	June	10018	az	411	24

 87

Highlight 'cast' functions – When we explored the data before writing this query, we were able to determine the type of data in each part of the query. Based on this information, we are able to cast each part of our query into the appropriate type.

Complex Query Examples

```
cast(FLATTEN(clicks.trans_info.prod_id) as bigint)
```



Metadata: hive.default.`orders`

	COLUMN_NAME	DATA_TYPE	IS_NULLABLE
1	order_id	BIGINT	YES
2	month	VARCHAR	YES
3	cust_id	BIGINT	YES
4	state	VARCHAR	YES
5	prod_id	BIGINT	YES

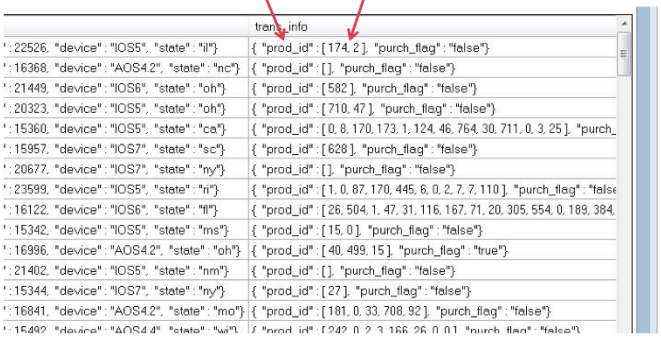
Data Preview:

	order_id	month	cust_id	state	prod_id	order_total
1	67212	June	10001	ca	909	13
2	70302	June	10004	ga	420	11
3	69090	June	10011	fl	44	76
4	68834	June	10012	ar	0	81
5	71220	June	10018	az	411	24

Highlight ‘cast’ functions – When we explored the data before writing this query, we were able to determine the type of data in each part of the query. Based on this information, we are able to cast each part of our query into the appropriate type.

Complex Query Examples

```
(FLATTEN(clicks.trans_info.prod_id) as bigint) as prod_id,
clicks.trans_info.purch_flag as purch_flag, to_date(clicks.`date`, 'yyyy-
mm-dd') as `date` FROM dfs.clicks.`/clicks/clicks.json` as clicks) as clk
```



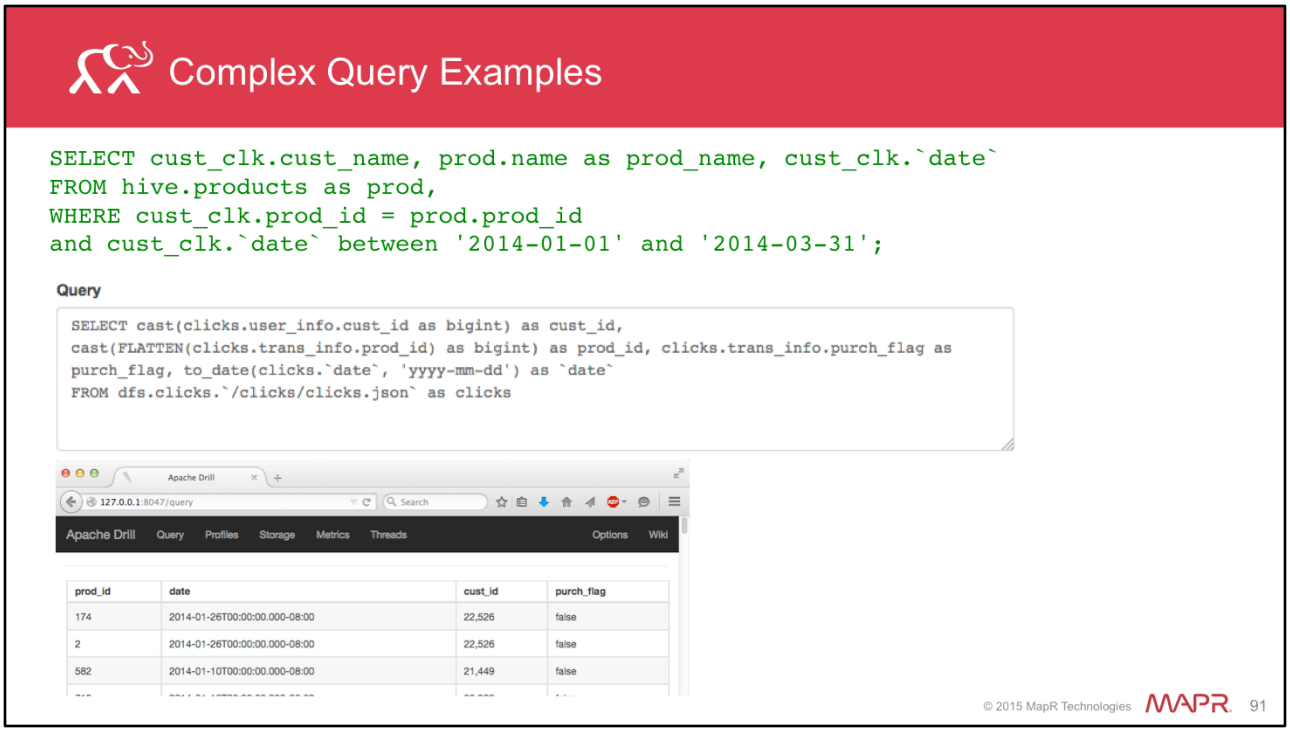
	trans_info
': 22526, "device" : "IOS5", "state" : "il"}	{ "prod_id" : [174, 2], "purch_flag" : "false"}
': 16368, "device" : "AOS4.2", "state" : "nc"}	{ "prod_id" : [ ], "purch_flag" : "false"}
': 21449, "device" : "IOS6", "state" : "oh"}	{ "prod_id" : [582], "purch_flag" : "false"}
': 20323, "device" : "IOS5", "state" : "oh"}	{ "prod_id" : [710, 47], "purch_flag" : "false"}
': 15360, "device" : "IOS5", "state" : "ca"}	{ "prod_id" : [0, 8, 170, 173, 1, 124, 46, 764, 30, 711, 0, 3, 25], "purch_
': 15957, "device" : "IOS7", "state" : "sc"}	{ "prod_id" : [628], "purch_flag" : "false"}
': 20677, "device" : "IOS7", "state" : "ny"}	{ "prod_id" : [ ], "purch_flag" : "false"}
': 23599, "device" : "IOS5", "state" : "ri"}	{ "prod_id" : [1, 0, 87, 170, 445, 6, 0, 2, 7, 7, 110], "purch_flag" : "false
': 16122, "device" : "IOS6", "state" : "fl"}	{ "prod_id" : [26, 504, 1, 47, 31, 116, 167, 71, 20, 305, 554, 0, 189, 384,
': 15342, "device" : "IOS5", "state" : "ms"}	{ "prod_id" : [15, 0], "purch_flag" : "false"}
': 16996, "device" : "AOS4.2", "state" : "oh"}	{ "prod_id" : [40, 499, 15], "purch_flag" : "true"}
': 21402, "device" : "IOS5", "state" : "nm"}	{ "prod_id" : [ ], "purch_flag" : "false"}
': 15344, "device" : "IOS7", "state" : "ny"}	{ "prod_id" : [27], "purch_flag" : "false"}
': 16841, "device" : "AOS4.2", "state" : "mo"}	{ "prod_id" : [181, 0, 33, 708, 92], "purch_flag" : "false"}
': 15492, "device" : "AOS4.4", "state" : "wi"}	{ "prod_id" : [242, 0, 2, 3, 166, 26, 0, 0], "purch_flag" : "false"}

Highlight FLATTEN statement – with Drill, we can perform a join that includes complex data types like arrays, maps and JSON blobs. When we explore the data with Drill explorer, we can see that we have a complex data type, and flatten it in our final query

Complex Query Examples

```
SELECT cust_clk.cust_name, prod.name as prod_name, cust_clk.`date` FROM
hive.products as prod,
(SELECT cast(cust.personal.name as varchar(40)) as cust_name, clk.prod_id, clk.`date`
FROM maprdb.customers as cust, (SELECT cast(clicks.user_info.cust_id as bigint) as
cust_id,
cast(FLATTEN(clicks.trans_info.prod_id) as bigint) as prod_id,
clicks.trans_info.purch_flag as purch_flag, to_date(clicks.`date`, 'yyyy-mm-dd') as
`date`
FROM dfs.clicks.`/clicks/clicks.json` as clicks) as clk
WHERE cast(cust.row_key as bigint) = clk.cust_id and clk.purch_flag = 'false') as
cust_clk
WHERE cust_clk.prod_id = prod.prod_id
and cust_clk.`date` between '2014-01-01' and '2014-03-31';
```


Color code each sub query – Drill does most of our work for us. When we explore the data, we can view the SQL code needed to access each piece of data individually. Once we can access each piece on its own, we just need to join them together into the final query. Drill is then able to perform this single query on all of these data sources, and give us the results we need



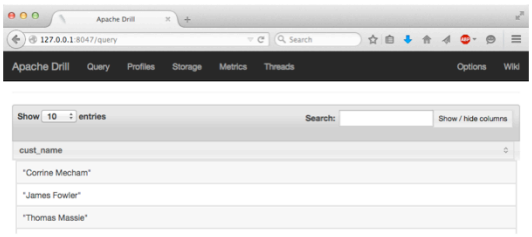
Color code each sub query – Drill does most of our work for us. When we explore the data, we can view the SQL code needed to access each piece of data individually. Once we can access each piece on its own, we just need to join them together into the final query. Drill is then able to perform this single query on all of these data sources, and give us the results we need

Complex Query Examples

```
(SELECT cast(cust.personal.name as varchar(40)) as cust_name, clk.prod_id, clk.`date`
FROM maprdb.customers as cust,
WHERE cast(cust.row_key as bigint) = clk.cust_id and clk.purch_flag = 'false') as
cust_clk
```

Query

```
select cast(cust.personal.name as varchar(40)) as cust_name
from maprdb.customers as cust, dfs.clicks.`/clicks/clicks.json` as clk
where cast(cust.row_key as bigint) = cast(clk.user_info.cust_id as bigint)
and cast(clk.trans_info.purch_flag as varchar(20)) = 'false' and
clk.trans_info.prod_id[0] is not null limit 10
```



 MAPR 92

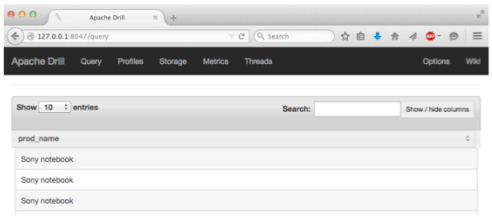
Color code each sub query – Drill does most of our work for us. When we explore the data, we can view the SQL code needed to access each piece of data individually. Once we can access each piece on its own, we just need to join them together into the final query. Drill is then able to perform this single query on all of these data sources, and give us the results we need

Complex Query Examples

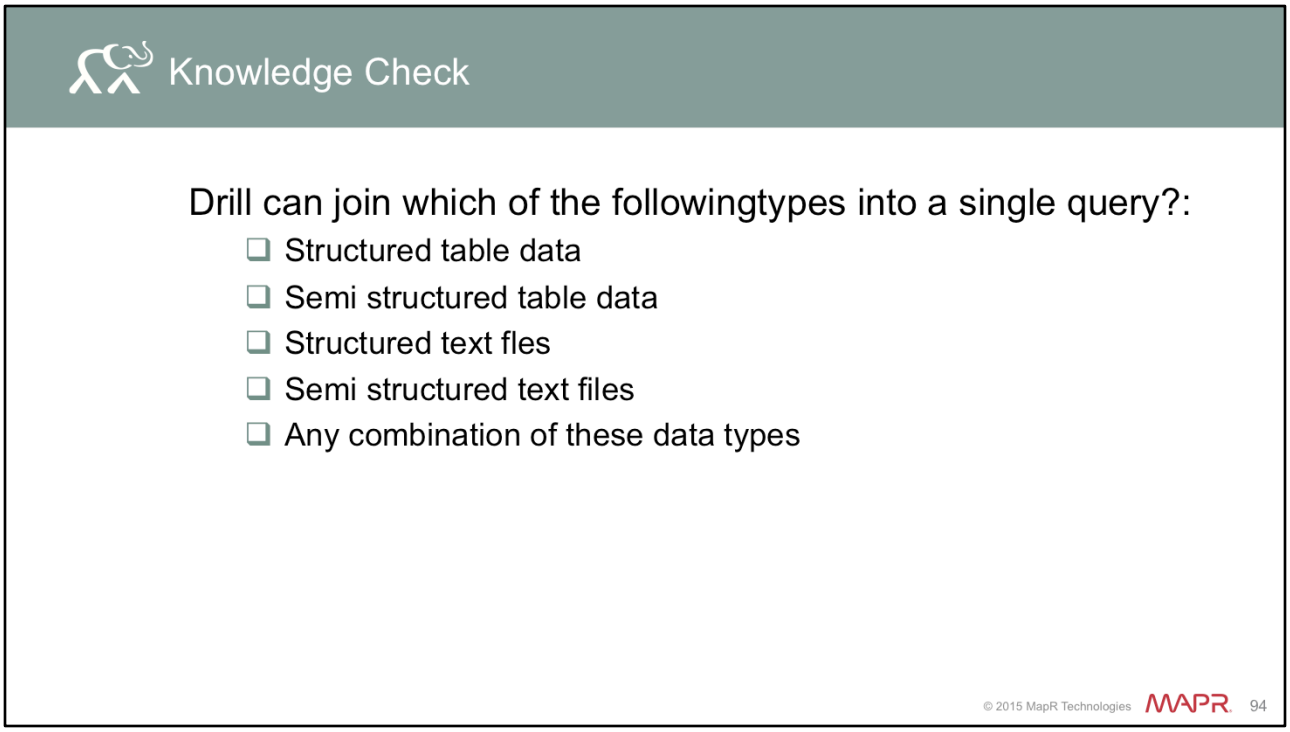
```
SELECT cast(clicks.user_info.cust_id as bigint) as cust_id,
cast(FLATTEN(clicks.trans_info.prod_id) as bigint) as prod_id,
clicks.trans_info.purch_flag as purch_flag, to_date(clicks.`date`, 'yyyy-mm-dd') as
`date`
FROM dfs.clicks.`/clicks/clicks.json` as clicks as clk
```

Query

```
select prod.name as prod_name
from hive.products as prod, dfs.clicks.`/clicks/clicks.json` as clk
where cast(clk.trans_info.prod_id[0] as bigint) = prod.prod_id and cast(clk.trans_info.purch_flag
as varchar(20)) = 'false' and clk.trans_info.prod_id[0] is not null limit 10
```



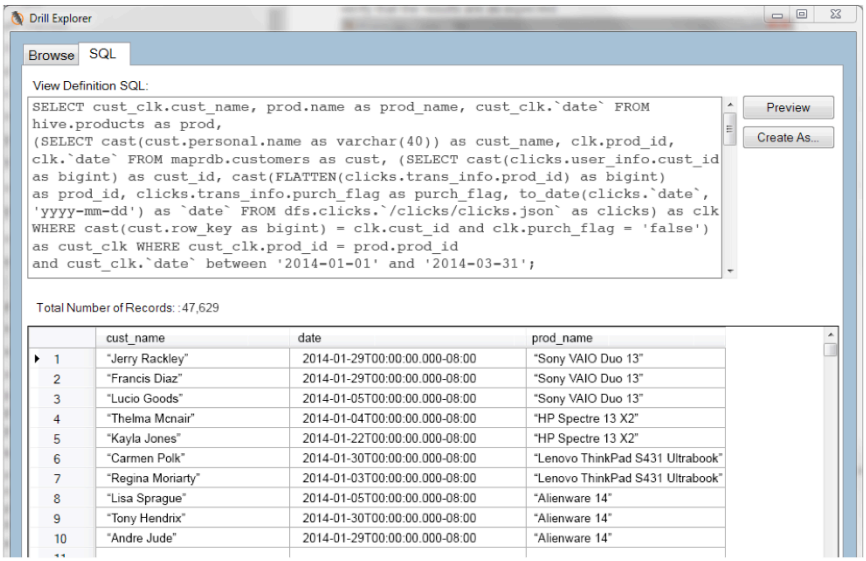
Color code each sub query – Drill does most of our work for us. When we explore the data, we can view the SQL code needed to access each piece of data individually. Once we can access each piece on its own, we just need to join them together into the final query. Drill is then able to perform this single query on all of these data sources, and give us the results we need



Any combination of these data types

- Add tables into views section
- Look at sales PPT for views information

Views

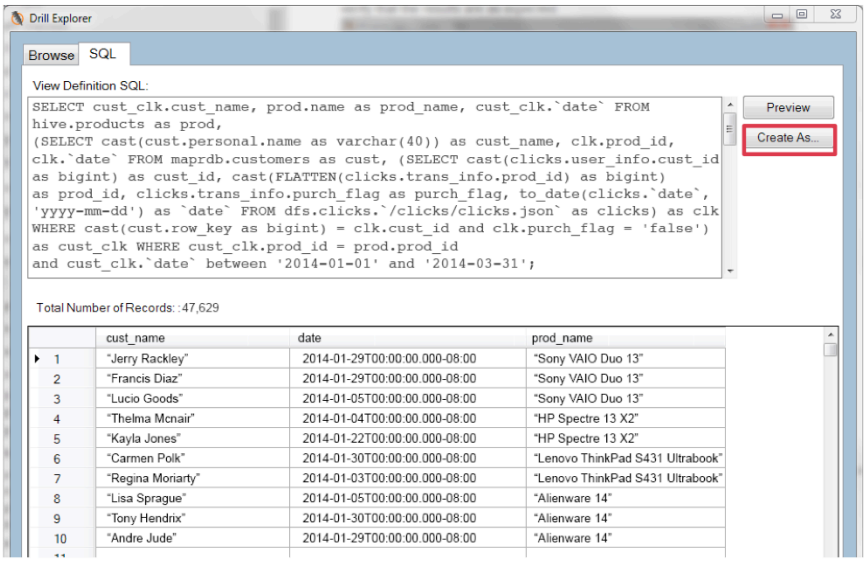


As we explore data using Drill, we will often find some interesting connection we did not know before, or we may have a relationship in our data that we want to return to and review frequently.

For example, when exploring the data for our recent promotion, we examined our sales and click data over time and geographic region, and were able to target our promotion based on customer views. Now, we want to watch the click data to see how our promotion affects spending in the regions that we targeted.

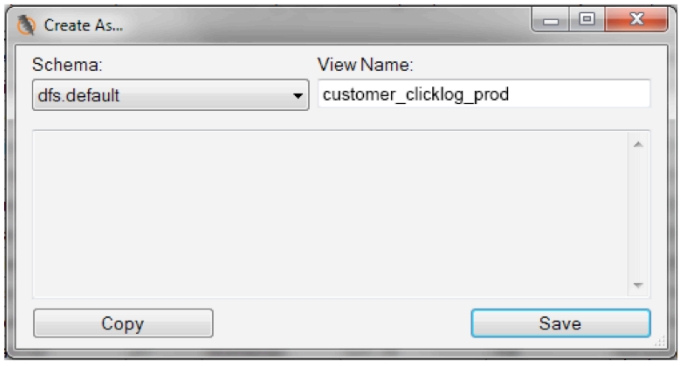
In Drill Explorer, we select the data that we want to view in the browser, or copy our SQL directly into the View Definition SQL field, and then select preview to look at the data that is returned from that query. If we find this query useful to us, we can save it as a view. A view saves the SQL and data source information for us, so that we can instantly reuse the query in the Drill Explorer and even share it with other BI tools through the ODBC interface.

Views



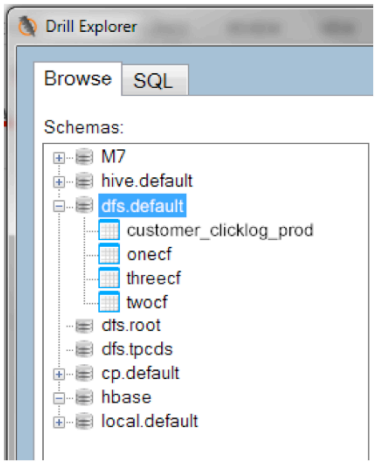
We create our new view by selecting the “Create As…” button.

Views



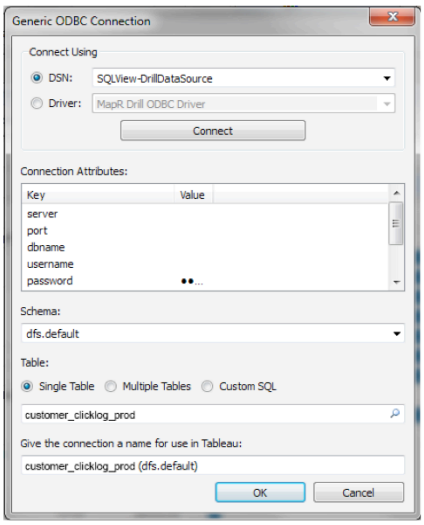
we give our view a useful name describing the data, select where to store it and hit save.

Views



We can see our new view in the data sources. Selecting this view will fill the Data Preview window with the results of our saved query.

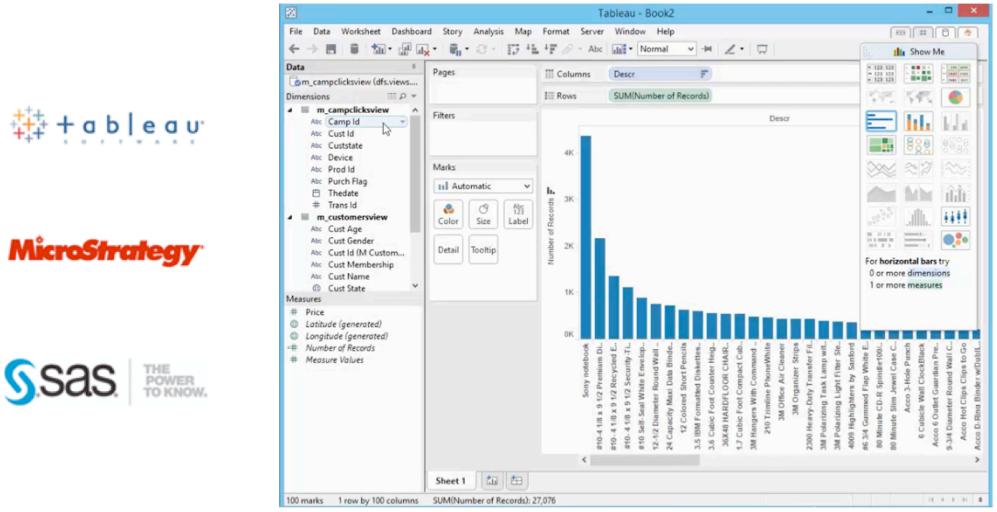
Views



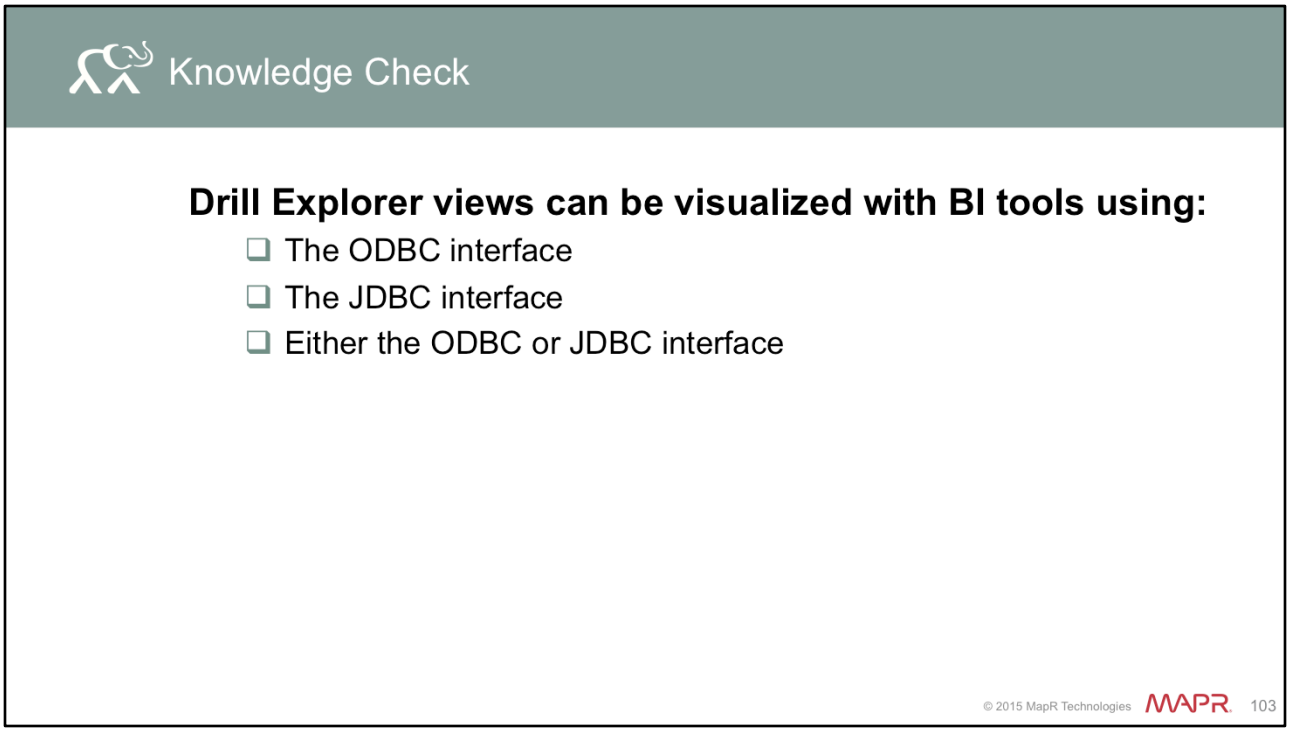
 101

With our view saved, we can share this information with any BI tool we have that uses an ODBC interface. We simply open our BI tool, point to the view, and then visualize the query in the interface.

Views



With our view saved, we can share this information with any BI tool we have that uses an ODBC interface. We simply open our BI tool, point to the view, and then visualize the query in the interface.



the ODBC interface

Congratulations, you have completed the Drill Essentials on demand course! You now have a good background on Apache Drill, and how it is a powerful new tool for exploring and visualizing structured and semi structured data.

Explore additional courses at the MapR training academy and the other resources at doc.mapr.com and answers.mapr.com to learn more about Hadoop and the ecosystem of components *(Elephants playing poker)*