Querying Data Lakes with SQL

Apache Spark™ and Databricks® make it easy to work with hierarchical data, such as nested JSON records.

Perform exploratory data analysis (EDA) to gain insights from a Data Lake.

In this lesson you:

- Use SQL to query a Data Lake
- Clean messy data sets
- Join two cleaned data sets

Data Lakes

Companies frequently have thousands of large data files gathered from various teams and departments, typically using a diverse variety of formats including CSV, JSON, and XML. Analysts often wish to extract insights from this data.

The classic approach to querying this data is to load it into a central database called a Data Warehouse. This involves the time-consuming operation of designing the schema for the central database, extracting the data from the various data sources, transforming the data to fit the warehouse schema, and loading it into the central database. The analyst can then query this enterprise warehouse directly or query smaller data marts created to optimize specific types of queries.

This classic Data Warehouse approach works well but requires a great deal of upfront effort to design and populate schemas. It also limits historical data, which is restrained to only the data that fits the warehouse's schema.

An alternative to this approach is the Data Lake. A *Data Lake*:

- Is a storage repository that cheaply stores a vast amount of raw data in its native format
- Consists of current and historical data dumps in various formats including XML, JSON, CSV, Parquet, etc.
- Also may contain operational relational databases with live transactional data

Spark is ideal for querying Data Lakes as the Spark SQL query engine is capable of reading directly from the raw files and then executing SQL queries to join and aggregate the Data.

You will see in this lesson that once two tables are created (independent of their underlying file type), we can join them, execute nested queries, and perform other operations across our Data Lake.

Looking at our Data Lake

You can start by reviewing which files are in our Data Lake.

In `dbfs:/mnt/training/crime-data-2016`, there are some Parquet files containing 2016 crime data from several United States cities.

As you can see in the cell below, we have data for Boston, Chicago, New Orleans, and more.

```
%fs ls /mnt/training/crime-data-2016
```

path	name	size
dbfs:/mnt/training/crime-data-2016/Crime-Data-Boston-2016.delta/	Crime-Data-Boston-2016.delta/	0
dbfs:/mnt/training/crime-data-2016/Crime-Data-Boston-2016.parquet/	Crime-Data-Boston-2016.parquet/	0
dbfs:/mnt/training/crime-data-2016/Crime-Data-Chicago-2016.delta/	Crime-Data-Chicago-2016.delta/	0
dbfs:/mnt/training/crime-data-2016/Crime-Data-Chicago-2016.parquet/	Crime-Data-Chicago-2016.parquet/	0
dbfs:/mnt/training/crime-data-2016/Crime-Data-Dallas-2016.delta/	Crime-Data-Dallas-2016.delta/	0
dbfs:/mnt/training/crime-data-2016/Crime-Data-Dallas-2016.parquet/	Crime-Data-Dallas-2016.parquet/	0
dbfs:/mnt/training/crime-data-2016/Crime-Data-Los-Angeles-2016.delta/	Crime-Data-Los-Angeles-2016.delta/	0
dbfs:/mnt/training/crime-data-2016/Crime-Data-Los-Angeles-2016.parquet/	Crime-Data-Los-Angeles-2016.parquet/	0
dbfs:/mnt/training/crime-data-2016/Crime-Data-New-Orleans-2016.parquet/	Crime-Data-New-Orleans-2016.parquet/	0

Command took 1.37 seconds -- by huseyinyilmaz01@gmail.com at 4/3/2020, 2:26:53 PM on test-cluster

The next step in looking at the data is to create a temporary view for each file. Recall that temporary views use a similar syntax to `CREATE TABLE` but using the command `CREATE TEMPORARY VIEW`. Temporary views are removed once your session has ended while tables are persisted beyond a given session.

Start by creating a view of the data from New York and then Boston:

City	Table Name	Path to DBFS file
New York	`CrimeDataNewYork`	`dbfs:/mnt/training/crime-data-2016/Crime-Data-New-York-2016.parquet`
Boston	`CrimeDataBoston`	`dbfs:/mnt/training/crime-data-2016/Crime-Data-Boston-2016.parquet`

Cmd 11

```
%sql

CREATE OR REPLACE TEMPORARY VIEW CrimeDataNewYork
  USING parquet
  OPTIONS (
    path "dbfs:/mnt/training/crime-data-2016/Crime-Data-New-York-2016.parquet"
  )
```

▸ (1) Spark Jobs

OK

Command took 0.77 seconds -- by huseyinyilmaz01@gmail.com at 4/3/2020, 2:31:06 PM on test-cluster

Cmd 12

```
%sql

CREATE OR REPLACE TEMPORARY VIEW CrimeDataBoston
  USING parquet
  OPTIONS (
    path "dbfs:/mnt/training/crime-data-2016/Crime-Data-Boston-2016.parquet"
  )
```

With the view created, it is now possible to review the first couple records of each file.

Notice in the example below:

- The `CrimeDataNewYork` and `CrimeDataBoston` datasets use different names for the columns
- The data itself is formatted differently and different names are used for similar concepts

This is common in a Data Lake. Often files are added to a Data Lake by different groups at different times. While each file itself usually has clean data, there is little consistency across files. The advantage of this strategy is that anyone can contribute information to the Data Lake and that Data Lakes scale to store arbitrarily large and diverse data. The tradeoff for this ease in storing data is that it doesn't have the rigid structure of a more traditional relational data model so the person querying the Data Lake will need to clean the data before extracting useful insights.

The alternative to a Data Lake is a Data Warehouse. In a Data Warehouse, a committee often regulates the schema and ensures data is cleaned before being made available. This makes querying much easier but also makes gathering the data much more expensive and time-consuming. Many companies choose to start with a Data Lake to accumulate data. Then, as the need arises, they clean the data and produce higher quality tables for querying. This reduces the upfront costs while still making data easier to query over time. These cleaned tables can even be later loaded into a formal data warehouse through nightly batch jobs. In this way, Apache Spark can be used to manage and query both Data Lakes and Data Warehouses.

```
%sql

SELECT * FROM CrimeDataNewYork
```

(1) Spark Jobs

complaintNumber	keyCode	offenseDescription	policeDeptCode	policeDeptDescription	lawCategoryCode	jurisdictionDesc	borough	precinct	locationOfOccurrenc
227505849	344	ASSAULT 3 & RELATED OFFENSES	101	ASSAULT 3	MISDEMEANOR	N.Y. POLICE DEPT	BRONX	52	INSIDE
586068632	235	DANGEROUS DRUGS	511	CONTROLLED SUBSTANCE, POSSESSI	MISDEMEANOR	PORT AUTHORITY	MANHATTAN	14	INSIDE
155423129	105	ROBBERY	389	ROBBERY,DWELLING	FELONY	N.Y. POLICE DEPT	BRONX	43	INSIDE
653964645	344	ASSAULT 3 & RELATED OFFENSES	101	ASSAULT 3	MISDEMEANOR	N.Y. POLICE DEPT	MANHATTAN	25	FRONT OF
988275798	235	DANGEROUS DRUGS	567	MARIJUANA, POSSESSION 4 & 5	MISDEMEANOR	N.Y. POLICE DEPT	MANHATTAN	7	OPPOSITE OF

Showing the first 1000 rows.

```
%sql

SELECT * FROM CrimeDataBoston
```

(1) Spark Jobs

INCIDENT_NUMBER	OFFENSE_CODE	OFFENSE_CODE_GROUP	OFFENSE_DESCRIPTION	DISTRICT	REPORTING_AREA	SHOOTING	OCCURRED_ON_DATE	YEAR	MONTI
I172071639	1402	Vandalism	VANDALISM	D4	905	null	2017-08-29T19:38:00.000+0000	2017	8
I172071637	3114	Investigate Property	INVESTIGATE PROPERTY	E13	574	null	2017-08-29T16:00:00.000+0000	2017	8
I172071635	3201	Property Lost	PROPERTY - LOST	A1	102	null	2017-08-29T21:00:00.000+0000	2017	8
I172071633	3115	Investigate Person	INVESTIGATE PERSON	C11	240	null	2017-08-26T18:00:00.000+0000	2017	8
I172071632	562	Other Burglary	BURGLARY - OTHER - NO FORCE	C6	232	null	2017-08-29T21:04:00.000+0000	2017	8

Cmd 16

Same type of data, different structure

In this section, we examine crime data to figure out how to extract homicide statistics.

Because our data sets are pooled together in a Data Lake, each city may use different field names and values to indicate homicides, dates, etc.

For example:

- Some cities use the value "HOMICIDE", "CRIMINAL HOMICIDE" or even "MURDER"
- In New York, the column is named `offenseDescription` but, in Boston, the column is named `OFFENSE_CODE_GROUP`
- In New York, the date of the event is in the `reportDate` column but, in Boston, there is a single column named `MONTH`

Cmd 17

To get started, create a temporary view containing only the homicide-related rows.

At the same time, normalize the data structure of each table so that all the columns (and their values) line up with each other.

In the case of New York and Boston, here are the unique characteristics of each data set:

	Offense-Column	Offense-Value	Reported-Column	Reported-Data Type
New York	`offenseDescription`	starts with "murder" or "homicide"	`reportDate`	`timestamp`
Boston	`OFFENSE_CODE_GROUP`	"Homicide"	`MONTH`	`integer`

For the upcoming aggregation, you will need to alter the New York data set to include a `month` column which can be computed from the `reportDate` column using the `month()` function. Boston already has this column.

One helpful tool for finding the offences we're looking for is using regular expressions supported by SQL

We can also normalize the values with the `CASE` , `WHEN` , `THEN` & `ELSE` expressions but that is not required for the task at hand.

Cmd 19

```
%sql

CREATE OR REPLACE TEMPORARY VIEW HomicidesNewYork AS
  SELECT month(reportDate) AS month, offenseDescription AS offense
  FROM CrimeDataNewYork
  WHERE lower(offenseDescription) LIKE 'murder%' OR lower(offenseDescription) LIKE 'homicide%'
```

OK

Command took 0.14 seconds -- by huseyinyilmaz01@gmail.com at 4/3/2020, 4:08:07 PM on test-cluster

Cmd 20

```
%sql

CREATE OR REPLACE TEMPORARY VIEW HomicidesBoston AS
  SELECT month, OFFENSE_CODE_GROUP AS offense
  FROM CrimeDataBoston
  WHERE lower(OFFENSE_CODE_GROUP) = 'homicide'
```

OK

Command took 0.03 seconds -- by huseyinyilmaz01@gmail.com at 4/3/2020, 4:08:30 PM on test-cluster

```sql
SELECT * FROM HomicidesNewYork LIMIT 5
```

(1) Spark Jobs

month	offense
12	MURDER & NON-NEGL. MANSLAUGHTER
12	MURDER & NON-NEGL. MANSLAUGHTER
12	MURDER & NON-NEGL. MANSLAUGHTER
12	MURDER & NON-NEGL. MANSLAUGHTER
12	MURDER & NON-NEGL. MANSLAUGHTER

1

Command took 0.68 seconds -- by huseyinyilmaz01@gmail.com at 4/3/2020, 4:08:59 PM on test-cluster

Cmd 23

```sql
%sql

SELECT * FROM HomicidesBoston LIMIT 5
```

(1) Spark Jobs

month	offense
8	Homicide
8	Homicide
8	Homicide
8	Homicide
8	Homicide

Cmd 24

Analyzing our data

Cmd 25

Now that we have normalized the homicide data for each city we can combine the two by taking their union.

When we are done, we can then aggregate that data to compute the number of homicides per month.

Start by creating a new view called `HomicidesBostonAndNewYork` which simply unions the result of two `SELECT` statements together.

NOTE See this Stack Overflow post for the difference between `UNION` and `UNION ALL`

```
%sql

CREATE OR REPLACE TEMPORARY VIEW HomicidesBostonAndNewYork AS
  SELECT * FROM HomicidesNewYork
    UNION ALL
  SELECT * FROM HomicidesBoston
```

OK

Command took 0.16 seconds -- by huseyinyilmaz01@gmail.com at 4/3/2020, 9:18:09 PM on test-cluster

Cmd 28

You can now see below all the data in one table:

Cmd 29

```
%sql

SELECT *
FROM HomicidesBostonAndNewYork
ORDER BY month
```

▸ (1) Spark Jobs

month	offense
1	MURDER & NON-NEGL. MANSLAUGHTER
1	MURDER & NON-NEGL. MANSLAUGHTER
1	MURDER & NON-NEGL. MANSLAUGHTER
1	MURDER & NON-NEGL. MANSLAUGHTER
1	MURDER & NON-NEGL. MANSLAUGHTER
1	MURDER & NON-NEGL. MANSLAUGHTER

And finally we can perform a simple aggregation to see the number of homicides per month:

Cmd 31

```
%sql

SELECT month, count(*) AS homicides
FROM HomicidesBostonAndNewYork
GROUP BY month
ORDER BY month
```

▸ (1) Spark Jobs

month	homicides
1	29
2	21
3	29
4	36
5	38
6	42
7	45
8	50

Exercise 1

Merge the crime data for Chicago with the data for New York and Boston and then update our final aggregation of counts-by-month.

Cmd 33

Step 1

Create the initial view of the Chicago data.

1. The source file is `dbfs:/mnt/training/crime-data-2016/Crime-Data-Chicago-2016.parquet`
2. Name the view `CrimeDataChicago`
3. View the data with a simple `SELECT` statement

Cmd 34

```
%sql
-- TODO

create or replace temporary view CrimeDataChicago
using parquet
options (path "dbfs:/mnt/training/crime-data-2016/Crime-Data-Chicago-2016.parquet")
```

▸ (1) Spark Jobs

OK

Command took 0.89 seconds -- by huseyinyilmaz01@gmail.com at 4/3/2020, 9:44:54 PM on test-cluster

Cmd 36

Step 2

Create a new view that normalizes the data structure.

1. Name the view `HomicidesChicago`
2. The table should have at least two columns: `month` and `offense`
3. Filter the data to only include homicides
4. View the data with a simple `SELECT` statement

Hint: You will need to use the `month()` function to extract the month-of-the-year.

Hint: To find out which values for each offense constitutes a homicide, produce a distinct list of values from the table `CrimeDataChicago`.

Cmd 37

```
%sql
-- TODO

create or replace temporary view HomicidesChicago as
select month(date) as month, lower(primaryType) as offense from CrimeDataChicago
where lower(primaryType) like "homicide%"
```

OK

Command took 0.10 seconds -- by huseyinyilmaz01@gmail.com at 4/3/2020, 10:23:31 PM on test-cluster

Cmd 39

Step 3

Create a new view that merges all three data sets (New York, Boston, Chicago):

1. Name the view `AllHomicides`
2. Use the `UNION ALL` expression introduced earlier to merge all three tables
   - `HomicidesNewYork`
   - `HomicidesBoston`
   - `HomicidesChicago`
3. View the data with a simple `SELECT` statement

Hint: To union three tables together, copy the previous example and just add as second `UNION` statement followed by the appropriate `SELECT` statement.

Cmd 40

```
%sql
-- TODO

create or replace temporary view AllHomicides as
select * from HomicidesNewYork
union all
select * from HomicidesBoston
union all
select * from HomicidesChicago
```

OK

Command took 0.10 seconds -- by huseyinyilmaz01@gmail.com at 4/3/2020, 10:28:48 PM on test-cluster

Step 4

Create a new view that counts the number of homicides per month.

1. Name the view `HomicidesByMonth`
2. Rename the column `count(1)` to `homicides`
3. Group the data by `month`
4. Sort the data by `month`
5. Count the number of records for each aggregate
6. View the data with a simple `SELECT` statement

Cmd 43

```
%sql
-- TODO

create or replace temporary view HomicidesByMonth as
select month, count(1) as homicides from AllHomicides
group by month
order by month
```

OK

Command took 0.04 seconds -- by huseyinyilmaz01@gmail.com at 4/3/2020, 10:38:57 PM on test-cluster

Summary

- Spark SQL allows you to easily manipulate data in a Data Lake
- Temporary views help to save your cleaned data for downstream analysis

Cmd 46

Review Questions

Q: What is a Data Lake?
A: Data Lakes are a loose collection of data files gathered from various sources. Spark loads each file as a table and then executes queries joining and aggregating these files.

Q: What are some advantages of Data Lakes over more classic Data Warehouses?
A: Data Lakes allow for large amounts of data to be aggregated from many sources with minimal ceremony or overhead. Data Lakes also allow for very very large files. Powerful query engines such as Spark can read the diverse collection of files and execute complex queries efficiently.

Q: What are some advantages of Data Warehouses?
A: Data warehouses are neatly curated to ensure data from all sources fit a common schema. This makes them very easy to query.

Q: What's the best way to combine the advantages of Data Lakes and Data Warehouses?
A: Start with a Data Lake. As you query, you will discover cases where the data needs to be cleaned, combined, and made more accessible. Create periodic Spark jobs to read these raw sources and write new "golden" tables that are cleaned and more easily queried.