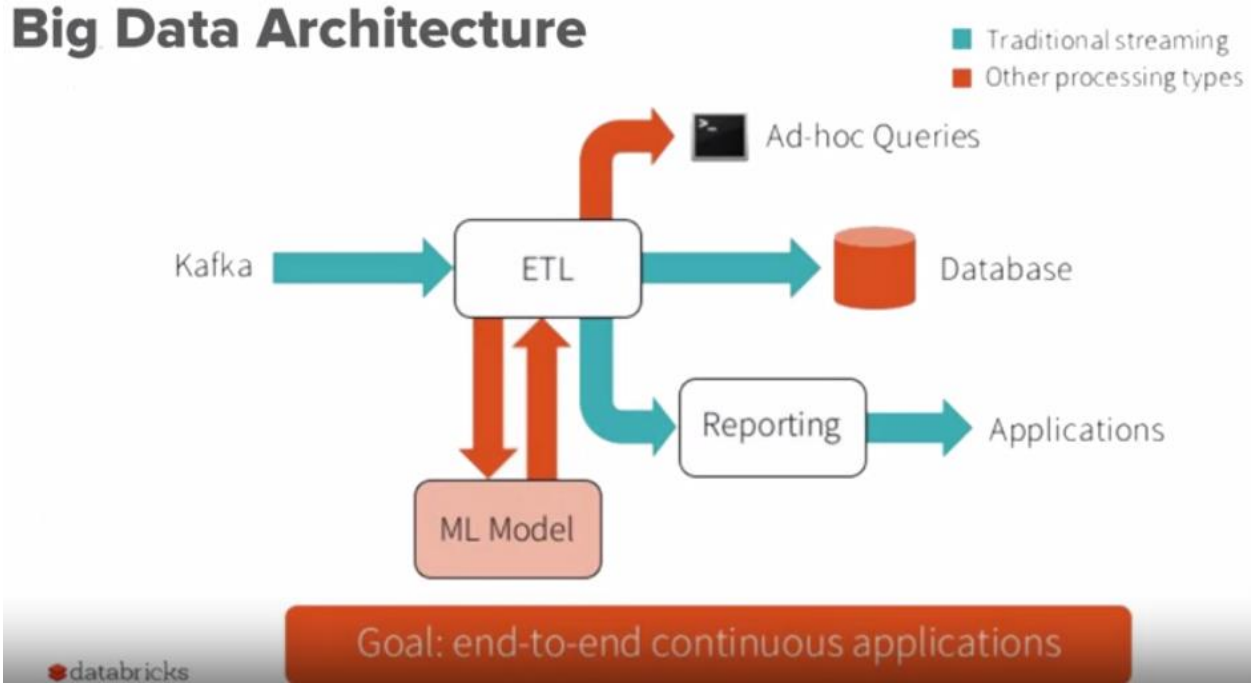
University of California Davis and Databricks collaboration

Distributed Computing with Spark SQL

103 Spark SQL Engineering Data Pipelines

Pipelining: Process of moving data through an application

Data Ecosystems: Legacy Data Warehouses, Databases, Data Lakes



Architecture Needs: Scalability

A scalable way to receive data

- Producers publish to topics
- Consumers consume from topics
- Apache Kafka, Amazon Kinesis, Azure Event Hub

Architecture Needs: ETL Process

Extract – Transform – Load

Raw data → usable data

Other Architecture Needs

Transactional database

Ad hoc queries

Machine learning

Reporting

Scalable data lake

Decoupled Storage and Compute

No data stays on the cluster

Resources are elastic

Updates are easy

IO vs. CPU Bound Problems

IO is network transfer (data transfer)

CPU is computation (processing data)

Data transfer is normally the bottleneck for tasks

Spark solves this by colocating your storage

Data Sources

Traditional databases like Postgres, SQL Server, and MySQL

NoSQL databases/documents stores like MongoDB

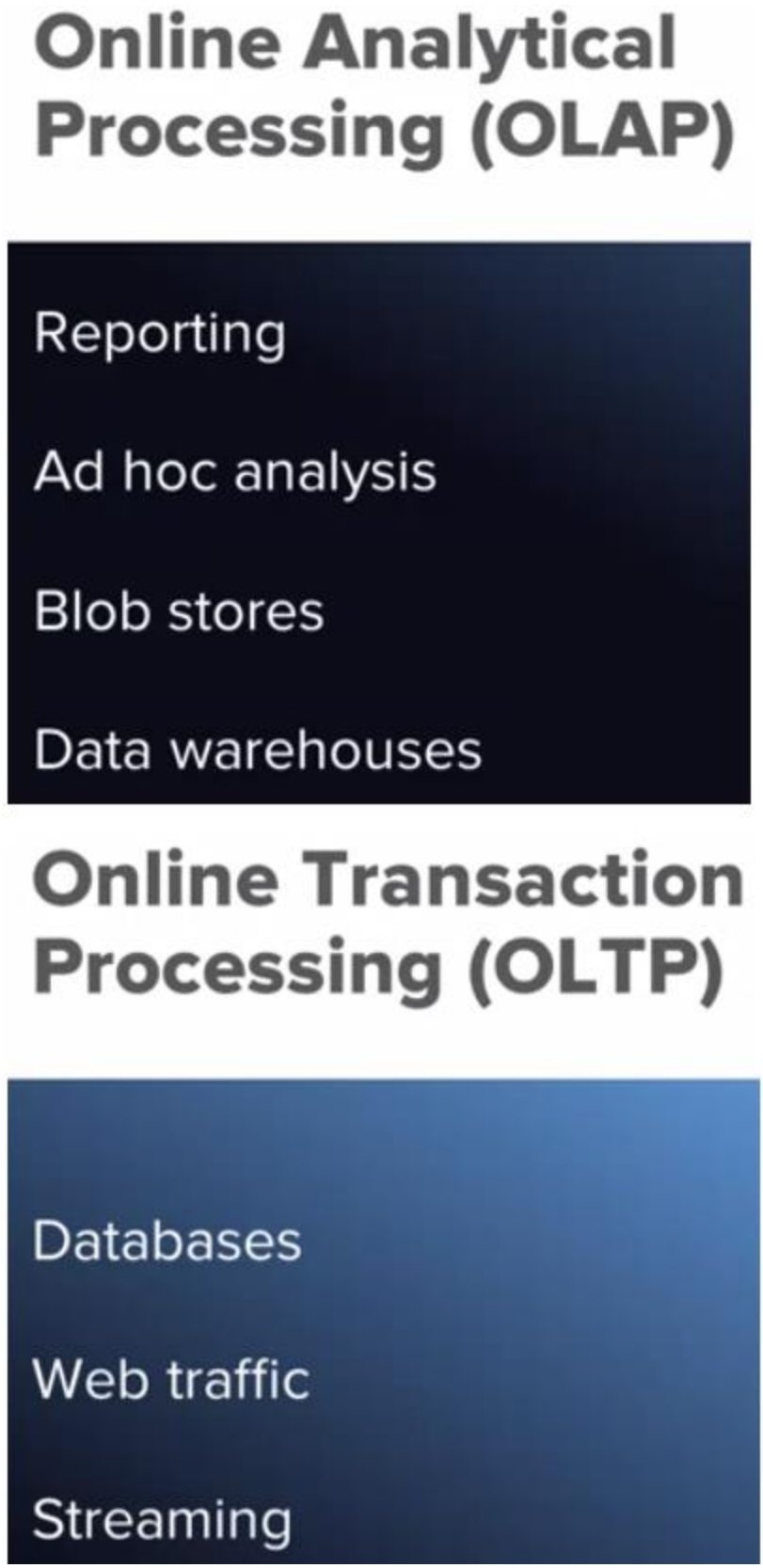
Distributed databases like Cassandra and Redshift

Blob stores like S3 and the Azure Blob

Message brokers like Kafka and Kinesis

Data warehouses like Hive

File types like CSV, Parquet, and Avro



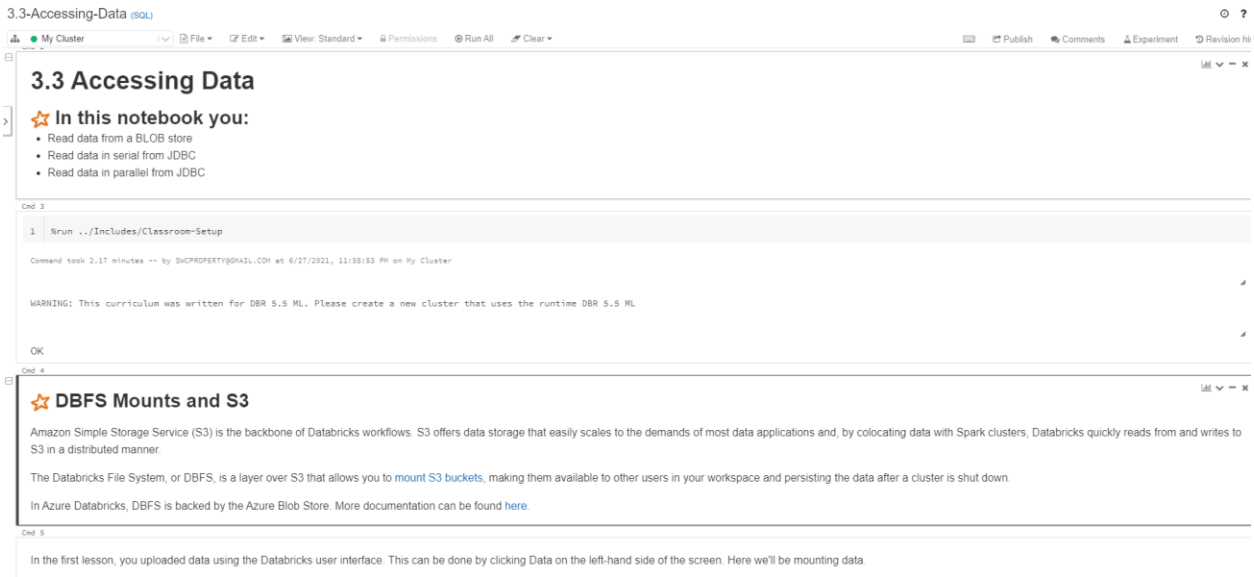
Connecting to two common data sources: BLOB and JDBC

BLOB and JDBC

BLOB is a common way of storing large amounts of data

JDBC is an application programming interface (API) for Java environments

Predicate push down functionality



Define your AWS credentials. Below are defined read-only keys, the name of an AWS bucket, and the mount name we will be referring to in DBFS.

For getting AWS keys, take a look at take a look at the AWS documentation

Cmd 7

```
%python
ACCESS_KEY = "AKIAJBRYNXGHORDHZB4A"
# Encode the Secret Key to remove any "/" characters
SECRET_KEY = "a0BzE1bSegfydr3%2FGE3LSPM6uIV5A4hOUfpH8aFF".replace("/", "%2F")
AWS_BUCKET_NAME = "davis-dsv1071/data/"
MOUNT_NAME = "/mnt/davis-tmp"
```

Command took 0.08 seconds -- by SWCPROPERTY@GMAIL.COM at 6/28/2021, 12:00:37 AM on My Cluster

Cmd 8

Now mount the bucket using the template provided in the docs.

The code below includes error handling logic to handle the case where the mount is already mounted.

Cmd 9

```
%python
try:
  dbutils.fs.mount("s3a://{}:{}@{}".format(ACCESS_KEY, SECRET_KEY, AWS_BUCKET_NAME), MOUNT_NAME)
except:
  print("""{} already mounted. Unmount using `dbutils.fs.unmount("{}")` to unmount first""".format(MOUNT_NAME, MOUNT_NAME))
```

▸ (1) Spark Jobs

Command took 21.79 seconds -- by SWCPROPERTY@GMAIL.COM at 6/28/2021, 12:00:41 AM on My Cluster

Cmd 10

```
%md

Next, explore the mount using `%fs ls` and the name of the mount.
```

Cmd 11

```
%fs ls /mnt/davis-tmp
```

```
%fs ls /mnt/davis-tmp
```

	path	name	size
1	dbfs:/mnt/davis-tmp/dataframes/	dataframes/	0
2	dbfs:/mnt/davis-tmp/fire-calls/	fire-calls/	0
3	dbfs:/mnt/davis-tmp/fire-incidents/	fire-incidents/	0

Showing all 3 rows.

Command took 2.23 seconds -- by SWCPROPERTY@GMAIL.COM at 6/28/2021, 12:05:04 AM on My Cluster

Cmd 12

In practice, always secure your AWS credentials. Do this by either maintaining a single notebook with restricted permissions that holds AWS keys, or delete the cells or notebooks that expose the keys. After a cell used to mount a bucket is run, you can access the data in this mount point in any notebook or any cluster in Databricks, and share the mount between colleagues.

Cmd 13

```
%fs mounts
```

	mountPoint	source	encryptionType
1	/mnt/davis	s3a://davis-dsv1071/data	
2	/databricks-datasets	databricks-datasets	sse-s3
3	/databricks/mlflow-tracking	databricks/mlflow-tracking	sse-s3
4	/databricks-results	databricks-results	sse-s3
5	/mnt/davis-tmp	s3a://davis-dsv1071/data/	
6	/databricks/mlflow-registry	databricks/mlflow-registry	sse-s3
7	/	DatabricksRoot	sse-s3

Showing all 7 rows.

Command took 0.89 seconds -- by SWCPROPERTY@GMAIL.COM at 6/28/2021, 12:05:16 AM on My Cluster

Cmd 14

You now have access to an unlimited store of data. This is a read-only S3 bucket. If you create and mount your own, you can write to it as well.

Now unmount the data.

Cmd 16

```
%python
dbutils.fs.unmount("/mnt/davis-tmp")
```

▸ (1) Spark Jobs

```
/mnt/davis-tmp has been unmounted.
Out[4]: True
```

Command took 21.38 seconds -- by SWCPROPERTY@GMAIL.COM at 6/28/2021, 12:05:20 AM on My Cluster

Cmd 17

☆ Serial JDBC Reads

Java Database Connectivity (JDBC) is an application programming interface (API) that defines database connections in Java environments. Spark is written in Scala, which runs on the Java Virtual Machine (JVM). This makes JDBC the preferred method for connecting to data whenever possible. Hadoop, Hive, and MySQL all run on Java and easily interface with Spark clusters.

Databases are advanced technologies that benefit from decades of research and development. To leverage the inherent efficiencies of database engines, Spark uses an optimization called predicate push down. **Predicate push down uses the database itself to handle certain parts of a query (the predicates).** In mathematics and functional programming, a predicate is anything that returns a Boolean. In SQL terms, this often refers to the `WHERE` clause. Since the database is filtering data before it arrives on the Spark cluster, there's less data transfer across the network and fewer records for Spark to process. Spark's Catalyst Optimizer includes predicate push down communicated through the JDBC API, making JDBC an ideal data source for Spark workloads.

Cmd 18

Run the cell below to confirm you are using the right driver.

Cmd 19

```
%scala
Class.forName("org.postgresql.Driver")
```

```
res4: Class[_] = class org.postgresql.Driver
```

Command took 0.53 seconds -- by SWCPROPERTY@GMAIL.COM at 6/28/2021, 12:06:30 AM on My Cluster

Cmd 20

First we need to create the JDBC String.

```
DROP TABLE IF EXISTS twitterJDBC;

CREATE TABLE IF NOT EXISTS twitterJDBC
USING org.apache.spark.sql.jdbc
OPTIONS (
  driver "org.postgresql.Driver",
  url "jdbc:postgresql://server1.databricks.training:5432/training",
  user "readonly",
  password "readonly",
  dbtable "training.Account"
)
```

OK

Command took 1.15 seconds -- by SWCPROPERTY@GMAIL.COM at 6/28/2021, 12:07:36 AM on My Cluster

Cmd 22

We now have a twitter table.

Cmd 23

```
SELECT * FROM twitterJDBC LIMIT 10
```

▸ (1) Spark Jobs

	userID	screenName	location	friendsCount	followersCount	description	insertID	ETLID
1	937781302685728769	helpcontasedms	Bia, Tutu e Gui.	486	1576	🤑Ative as notificações para não perder nenhuma conta premium 🤑 ⇽ Libero,Faço Games (Ou Sorteio) contas premium e (sorteio) dms...	566935683782	3153
2	161375991	PopTartsUS	United States	4	67859	Tweet your love for Pop-Tarts® toaster pastries. They're crazy good.	566935683783	3153
3	570337310	MarcoYanez52	Venezuela Edo. Bolivar	4515	4510	REVOLUCIONARIO - CHAVISTA "EFICACIA POLITICA Y CALIDAD REVOLUCIONARIA"... ALFREDO MANEIRO. FOTOGRAFO	566935683784	3153
4	952038431122644993	YuuR223	♥10.8	77	85	🏳LG(B)T 女の欠けらも無い奴ですこんにちは。気になったらポチっと。繁がりましょ(3_ヽ)_	566935683785	3153
5	951142300641619968	Heaven_2097	null	37	34	голосую и только	566935683786	3153
6	204382100	stlt_may	ぎり東京	603	343	舞台とテーマパークが生きがい。夏と秋は大阪に住みたい東京民→次in2/4、3/18。ONE PIECE→剣士最愛船長轟雷。 USJ・ディズニー。 特撮・俳優・2.5次元。 Psycho le Cemu、SOPHIA、人狼TLPT。 魔女子レイヤーa:206368 写真垢→ @hiyohy	566935683787	3153
7	2560383828	imesod	Miami, FL	650	720	the brokest expensive bitch (@matsboomchafsky 's queen)	566935683788	3153

Showing all 10 rows.

Command took 5.90 seconds -- by SWCPROPERTY@GMAIL.COM at 6/28/2021, 12:07:41 AM on My Cluster

Add a subquery to `dbtable`, which pushes the predicate to JDBC to process before transferring the data to your Spark cluster.

Cmd 25

```
DROP TABLE IF EXISTS twitterPhilippinesJDBC;

CREATE TABLE twitterPhilippinesJDBC
USING org.apache.spark.sql.jdbc
OPTIONS (
  driver "org.postgresql.Driver",
  url "jdbc:postgresql://server1.databricks.training:5432/training",
  user "readonly",
  password "readonly",
  dbtable "(SELECT * FROM training.Account WHERE location = 'Philippines') as subq"
)
```

OK

Command took 0.74 seconds -- by SWCPROPERTY@GMAIL.COM at 6/28/2021, 12:11:04 AM on My Cluster

Cmd 26

```
SELECT * FROM twitterPhilippinesJDBC LIMIT 10
```

▸ (1) Spark Jobs

	userID	screenName	location	friendsCount	followersCount	description	insertID	ETLID
1	877436554766987264	m0wskito	Philippines	224	52	schatz	558345749937	3153
2	2807579618	destinedtoV	Philippines	158	72	But I fell for you so bad. Though I knew that you and I weren't a good fit (ctto: pic)	549755819681	3153
3	422488190	erickmacasaet	Philippines	349	456	Broke but still Grinding at my 20's	566935684829	3153
4	54405827	asdfghkjirah	Philippines	1837	954	Jeremiah 29:11 x A wandering soul that just wants to go to Seoul	575525617821	3153
5	945951715689250818	seokpiggy	Philippines	512	171	My handsomeness is staining my life.	558345751013	3153
6	384854316	lovatic_McDLove	Philippines	665	306	Obsessing over Demi Lovato, Girls' Generation, APink, IU, BlackPink 💔	575525618268	3153
7	123891658	itsgirlie1818	Philippines	524	1094	i love music and i'm a fangirl :)	549755821532	3153

Showing all 10 rows.

Command took 1.16 seconds -- by SWCPROPERTY@GMAIL.COM at 6/28/2021, 12:11:19 AM on My Cluster

Parallel JDBC Reads

Cmd 28

```
1  SELECT min(userID) as minID, max(userID) as maxID from twitterJDBC
```

▸ (2) Spark Jobs

	minID	maxID
1	509	959519629566672896

Showing all 1 rows.

Command took 2.76 seconds -- by SWCPROPERTY@GMAIL.COM at 6/28/2021, 12:12:01 AM on My Cluster

Cmd 29

```
1  DROP TABLE IF EXISTS twitterParallelJDBC;
2
3  CREATE TABLE IF NOT EXISTS twitterParallelJDBC
4  USING org.apache.spark.sql.jdbc
5  OPTIONS (
6    driver "org.postgresql.Driver",
7    url "jdbc:postgresql://server1.databricks.training:5432/training",
8    user "readonly",
9    password "readonly",
10   dbtable "training.Account",
11   partitionColumn '"userID"',
12   lowerBound 2591,
13   upperBound 951253910555168768,
14   numPartitions 25
15 )
```

OK

Command took 0.59 seconds -- by SWCPROPERTY@GMAIL.COM at 6/28/2021, 12:12:07 AM on My Cluster

```
SELECT * from twitterParallelJDBC
```

▸ (1) Spark Jobs

	userID	screenName	location	friendsCount	followersCount	description
1	161375991	PopTartsUS	United States	4	67859	Tweet your love for Pop-Tarts® toaster pastries. They're crazy good.
2	570337310	MarcoYanez52	Venezuela Edo. Bolivar	4515	4510	REVOLUCIONARIO - CHAVISTA "EFICACIA POLITICA Y CALIDAD REVOLUCI FOTOGRAFO
3	204382100	stlt_may	ぎり東京	603	343	舞台とチーマパークが生きがい。夏と秋は大阪に住みたい東京民→次in2/4、3/1 ィズニー。 特撮・俳優・2.5次元。 Psycho le Cemu、SOPHIA、 人狼TLPT、
4	2560383828	imesod	Miami, FL	650	720	the brokest expensive bitch (@matsboomchafsky 's queen)
5	625643590	thunderlouisx	LF.	128	4994	todo el día mirándome a mí, tooooooodo el día
6	228659757	ploryy_exo12	null	1579	1249	ชื่อเกียจแล้วจะเจริญมั้ย? / ไรท์เตอร์ที่มือหวยๆแต่อยากแต่ง จบปะ? / #EXO #kaillu ❤ 18+
7	2796412314	fffo42	@fffo42	26066	28802	استغرام 0559855359/ابي fffo42/ ارحب بلقاء ازواج للتواصل/DB5B3CAD سناب / fffo423

Truncated results, showing first 1000 rows.

Command took 3.44 seconds -- by SWCPROPERTY@GMAIL.COM at 6/28/2021, 12:12:43 AM on My Cluster

Cmd 31

```
%python
%timeit sql("SELECT * from twitterJDBC").describe()
```

▸ (16) Spark Jobs

33.7 s ± 1.84 s per loop (mean ± std. dev. of 7 runs, 1 loop each)

Command took 4.47 minutes -- by SWCPROPERTY@GMAIL.COM at 6/28/2021, 12:12:51 AM on My Cluster

Cmd 32

```
%python
%timeit sql("SELECT * from twitterParallelJDBC").describe()
```

▸ (16) Spark Jobs

36.4 s ± 1.12 s per loop (mean ± std. dev. of 7 runs, 1 loop each)

Command took 4.90 minutes -- by SWCPROPERTY@GMAIL.COM at 6/28/2021, 12:15:23 AM on My Cluster

3.4-File-Formats (SQL)

My Cluster | File | Edit | View: Standard | Permissions | Run All | Clear

Cmd 2

3.4 File Formats

In this notebook you:

- Compare file formats and compression types
- Examine Parquet

Cmd 3

```
1  %run ../Includes/Classroom-Setup
```

```
Command took 2.04 minutes -- by SWCPROPERTY@GMAIL.COM at 6/29/2021, 7:35:36 PM on My Cluster


WARNING: This curriculum was written for DBR 5.5 ML. Please create a new cluster that uses the runtime DBR 5.5 ML


OK
```

Cmd 4

Comparing File Formats

Cmd 5

Let's take a look at a colon delimited file sitting on S3.

Cmd 6

```
1  %fs ls /mnt/davis/fire-calls/fire-calls-colon.txt
```

	path	name	size
1	dbfs:/mnt/davis/fire-calls/fire-calls-colon.txt	fire-calls-colon.txt	1829688314

Showing all 1 rows.

```
Command took 2.03 seconds -- by SWCPROPERTY@GMAIL.COM at 6/29/2021, 7:38:18 PM on My Cluster
```

Take a look at the first few lines of the file.

Cmd 8

```
%fs head --maxBytes=1000 /mnt/davis/fire-calls/fire-calls-colon.txt
```

```
[Truncated to first 1000 bytes]
Call Number:Unit ID:Incident Number:Call Type:Call Date:Watch Date:Received DtTm:Entry DtTm:Dispatch DtTm:Response DtTm:On Scene DtTm:Transport DtTm:Hospital DtTm:Call Final Disposition:Available DtTm:Address:City:Zipcode of Incident:Battalion:Station Area:Box:Original Priority:Priority:Final Priority:ALS Unit:Call Type Group:Number of Alarms:Unit Type:Unit sequence in call dispatch:Fire Prevention District:Supervisor District:Neighborhooods - Analysis Boundaries:Location:RowID
001030101:E18:00306091:Medical Incident:04/12/2000:04/12/2000:"04/12/2000 09:00:29 PM":"04/12/2000 09:01:40 PM":"04/12/2000 09:02:00 PM":"":"":"":Other:"":2000 Block of 37TH AVE:SF:94116:B08:18:0757:3:3:3:false:"":1:ENGINE:1:8:4:Sunset/Parkside:(37.7487247711275, -122.495504020186):001030101-E18
001030104:M14:00030612:Medical Incident:04/12/2000:04/12/2000:"04/12/2000 09:09:02 PM":"04/12/2000 09:10:17 PM":"04/12/2000 09:10:29 PM":"04/12/2000 09:12:11 PM":"04/12/2000 09:19:36 PM":"04/12/2000 09:43:57 PM":"04/1
```

Command took 0.75 seconds -- by SWCPROPERTY@GMAIL.COM at 6/29/2021, 7:38:31 PM on My Cluster

Cmd 9

Create a temporary view of the file using `:` as the separator.

Cmd 10

```
CREATE OR REPLACE TEMPORARY VIEW fireCallsCSV
USING CSV
OPTIONS (
    path "/mnt/davis/fire-calls/fire-calls-colon.txt",
    header "true",
    sep ":"
  )
```

▸ (1) Spark Jobs

OK

Command took 1.16 seconds -- by SWCPROPERTY@GMAIL.COM at 6/29/2021, 7:38:35 PM on My Cluster

Cmd 11

Take a look at the types of data in the table.

Cmd 12

```
DESCRIBE fireCallsCSV
```

	col_name	data_type	comment
1	Call Number	string	null
2	Unit ID	string	null
3	Incident Number	string	null
4	Call Type	string	null
5	Call Date	string	null
6	Watch Date	string	null
7	Received DtTm	string	null

Showing all 34 rows.

Command took 0.23 seconds -- by SWCPROPERTY@GMAIL.COM at 6/29/2021, 7:40:10 PM on My Cluster

Cmd 13

Are these data types correct? All of them are string types.

We need to tell Spark to infer the schema.

Cmd 14

```
CREATE OR REPLACE TEMPORARY VIEW fireCallsCSV
USING CSV
OPTIONS (
    path "/mnt/davis/fire-calls/fire-calls-colon.txt",
    header "true",
    sep ":",
    inferSchema "true"
  )
```

▸ (2) Spark Jobs

OK

Command took 1.04 minutes -- by SWCPROPERTY@GMAIL.COM at 6/29/2021, 7:41:19 PM on My Cluster

Cmd 15

Now take a look at how Spark inferred the data types.

Cmd 16

```
DESCRIBE fireCallsCSV
```

	col_name	data_type	comment
1	Call Number	int	null
2	Unit ID	string	null
3	Incident Number	int	null
4	Call Type	string	null
5	Call Date	string	null
6	Watch Date	string	null
7	Received DtTm	string	null

Showing all 34 rows.

Command took 0.21 seconds -- by SWCPROPERTY@GMAIL.COM at 6/29/2021, 7:44:41 PM on My Cluster

Cmd 17

Wow, that took a long time just to figure out the schema for this file!

Now let's try the same thing with compressed files (Gzip and Bzip formats).

Notice that the bzip file is the most compact - we will see if it is the fastest to operate on.

Cmd 18

```
1  %fs ls /mnt/davis/fire-calls/fire-calls-colon.txt
```

	path	name	size
1	dbfs:/mnt/davis/fire-calls/fire-calls-colon.txt	fire-calls-colon.txt	1829688314

Showing all 1 rows.

Command took 1.03 seconds -- by SWCPROPERTY@GMAIL.COM at 6/29/2021, 7:44:47 PM on My Cluster

Cmd 19

```
1  %fs ls /mnt/davis/fire-calls/fire-calls-1p.txt.gz
```

	path	name	size
1	dbfs:/mnt/davis/fire-calls/fire-calls-1p.txt.gz/_SUCCESS	_SUCCESS	0
2	dbfs:/mnt/davis/fire-calls/fire-calls-1p.txt.gz/_committed_1074000646993103229	_committed_1074000646993103229	116
3	dbfs:/mnt/davis/fire-calls/fire-calls-1p.txt.gz/_started_1074000646993103229	_started_1074000646993103229	0
4	dbfs:/mnt/davis/fire-calls/fire-calls-1p.txt.gz/part-00000-tid-4222569572634380978-7f3cb922-b07c-4d71-b14f-f65d838c499a-15121-c000.csv.gz	part-00000-tid-4222569572634380978-7f3cb922-b07c-4d71-b14f-f65d838c499a-15121-c000.csv.gz	259566857

Showing all 4 rows.

Command took 0.77 seconds -- by SWCPROPERTY@GMAIL.COM at 6/29/2021, 7:44:50 PM on My Cluster

Cmd 20

```
%fs ls /mnt/davis/fire-calls/fire-calls-1p.txt.bzip
```

	path	name	size
1	dbfs:/mnt/davis/fire-calls/fire-calls-1p.txt.bzip/_SUCCESS	_SUCCESS	0
2	dbfs:/mnt/davis/fire-calls/fire-calls-1p.txt.bzip/_committed_7012599954241226118	_committed_7012599954241226118	117
3	dbfs:/mnt/davis/fire-calls/fire-calls-1p.txt.bzip/_started_7012599954241226118	_started_7012599954241226118	0
4	dbfs:/mnt/davis/fire-calls/fire-calls-1p.txt.bzip/part-00000-tid-7012599954241226118-f407e362-c5a4-49ec-973b-3420b6ae94fd-15136-c000.csv.bz2	part-00000-tid-7012599954241226118-f407e362-c5a4-49ec-973b-3420b6ae94fd-15136-c000.csv.bz2	192991035

Showing all 4 rows.

Command took 0.74 seconds -- by SWCPROPERTY@GMAIL.COM at 6/29/2021, 7:44:53 PM on My Cluster

Cmd 21

Let's start by reading in the gzipped file.

```
CREATE OR REPLACE TEMPORARY VIEW fireCallsCSVgzip
USING CSV
OPTIONS (
    path "/mnt/davis/fire-calls/fire-calls-1p.txt.gz",
    header "true",
    sep ":",
    inferSchema "true"
  )
```

▸ (2) Spark Jobs

OK

Command took 39.74 seconds -- by SWCPROPERTY@GMAIL.COM at 6/29/2021, 7:44:59 PM on My Cluster

Cmd 23

Wow! That took way longer than inferring the schema on the uncompressed data. Even though it took up less storage space, we had to pay for that in computation.

You'll notice that the resulting view is comprised of only 1 partition, which makes this data very slow to query later on.

Cmd 24

```
%python
sql("SELECT * FROM fireCallsCSVgzip").rdd.getNumPartitions()
```

Out[2]: 1

Command took 0.24 seconds -- by SWCPROPERTY@GMAIL.COM at 6/29/2021, 7:48:16 PM on My Cluster

Cmd 25

Let's compare the speed of reading in the gzip file to the bzip file!

Cmd 26

```
CREATE OR REPLACE TEMPORARY VIEW fireCallsCSVbzip
USING CSV
OPTIONS (
    path "/mnt/davis/fire-calls/fire-calls-1p.txt.bzip",
    header "true",
    sep ":",
    inferSchema "true"
  )
```

```
%python
sql("SELECT * FROM fireCallsCSVbzip").rdd.getNumPartitions()
```

```
Out[3]: 8
```

```
Command took 0.18 seconds -- by SWCPROPERTY@GMAIL.COM at 6/29/2021, 7:51:09 PM on My Cluster
```

Cmd 28

Bzip is a "splittable" file format, so it is much better to use than gzip when working with row-based formats for querying later on.

Now let's go ahead and compare that to reading in from a columnar format: Parquet.

Cmd 29

```
CREATE OR REPLACE TEMPORARY VIEW fireCallsParquet
USING Parquet
OPTIONS (
    path "/mnt/davis/fire-calls/fire-calls-1p.parquet"
  )
```

▸ (1) Spark Jobs

OK

```
Command took 1.35 seconds -- by SWCPROPERTY@GMAIL.COM at 6/29/2021, 7:51:12 PM on My Cluster
```

Cmd 30

```
DESCRIBE fireCallsParquet
```

	col_name	data_type	comment
1	Call_Number	int	null
2	Unit_ID	string	null
3	Incident_Number	int	null
4	Call_Type	string	null
5	Call_Date	string	null
6	Watch_Date	string	null
7	Received_DtTm	string	null

Showing all 34 rows.

Look at how fast it is to get the schema from a Parquet file! That is because the Parquet file stores the data and the associated metadata.

Cmd 32

Compare the performance between the three file types. We are going to use a Python helper function called timeit to calculate how long the query takes to execute.

Cmd 33

```
%python
parquetDF = sql("SELECT * FROM fireCallsParquet")
%timeit -n1 -r1 parquetDF.select("City").where("City == 'San Francisco'").count()
```

▸ (2) Spark Jobs

▸ parquetDF: pyspark.sql.dataframe.DataFrame = [Call_Number: integer, Unit_ID: string ... 32 more fields]

```
7.87 s ± 0 ns per loop (mean ± std. dev. of 1 run, 1 loop each)
```

Command took 8.05 seconds -- by SWCPROPERTY@GMAIL.COM at 6/29/2021, 7:52:05 PM on My Cluster

Cmd 34

```
%python
csvDF = sql("SELECT * FROM fireCallsCSV")
%timeit -n1 -r1 csvDF.select("City").where("City == 'San Francisco'").count()
```

▸ (2) Spark Jobs

▸ csvDF: pyspark.sql.dataframe.DataFrame = [Call Number: integer, Unit ID: string ... 32 more fields]

```
40.9 s ± 0 ns per loop (mean ± std. dev. of 1 run, 1 loop each)
```

Command took 41.10 seconds -- by SWCPROPERTY@GMAIL.COM at 6/29/2021, 7:52:07 PM on My Cluster

Cmd 35

```
%python
gzipDF = sql("SELECT * FROM fireCallsCSVgzip")
%timeit -n1 -r1 gzipDF.select("City").where("City == 'San Francisco'").count()
```

▸ (2) Spark Jobs

▸ gzipDF: pyspark.sql.dataframe.DataFrame = [Call Number: integer, Unit ID: string ... 32 more fields]

```
24.7 s ± 0 ns per loop (mean ± std. dev. of 1 run, 1 loop each)
```

Command took 24.95 seconds -- by SWCPROPERTY@GMAIL.COM at 6/29/2021, 7:52:07 PM on My Cluster

```
%python
bzipDF = sql("SELECT * FROM fireCallsCSVbzip")
%timeit -n1 -r1 bzipDF.select("City").where("City == 'San Francisco'").count()
```

▸ (2) Spark Jobs

▸ bzipDF: pyspark.sql.dataframe.DataFrame = [Call Number: integer, Unit ID: string ... 32 more fields]

```
1min 19s ± 0 ns per loop (mean ± std. dev. of 1 run, 1 loop each)
```

Command took 1.33 minutes -- by SWCPROPERTY@GMAIL.COM at 6/29/2021, 7:52:08 PM on My Cluster

Cmd 37

Reading from Parquet Files

"Apache Parquet is a columnar storage format available to any project in the Hadoop ecosystem, regardless of the choice of data processing framework, data model or programming language."

Parquet

https://parquet.apache.org

Cmd 38

About Parquet Files

- Free & Open Source.
- Increased query performance over row-based data stores.
- Provides efficient data compression.
- Designed for performance on large data sets.
- Supports limited schema evolution.
- Is a splittable "file format".
- A Column-Oriented data store

Row Format

ID	Name	Score
1	john	4.1
2	mike	3.5
3	sally	6.4

Column Format

ID:	1	2	3
Name:	john	mike	sally
Score:	4.1	3.5	6.4

See also

- https://parquet.apache.org
- https://en.wikipedia.org/wiki/Apache_Parquet

Cmd 39

```
1  SELECT count(`Incident Number`)
2  FROM firecallsCSV
3  WHERE Priority > 1
```

▸ (2) Spark Jobs

	count(Incident Number)
1	4401920

Showing all 1 rows.

Command took 37.62 seconds -- by SWCPROPERTY@GMAIL.COM at 6/29/2021, 8:30:43 PM on My Cluster

Cmd 40

```
1  SELECT count(`Incident_Number`)
2  FROM firecallsParquet
3  WHERE Priority > 1
```

▸ (2) Spark Jobs

	count(Incident_Number)
1	4401920

Showing all 1 rows.

Command took 2.65 seconds -- by SWCPROPERTY@GMAIL.COM at 6/29/2021, 8:30:49 PM on My Cluster

Cmd 41

How much faster can this query get if we read from a partitioned Parquet file?

```
%fs ls /mnt/davis/fire-calls/fire-calls-8p.parquet
```

	path	name	size
1	dbfs:/mnt/davis/fire-calls/fire-calls-8p.parquet/_SUCCESS	_SUCCESS	0
2	dbfs:/mnt/davis/fire-calls/fire-calls-8p.parquet/_committed_1695916237803481236	_committed_1695916237803481236	824
3	dbfs:/mnt/davis/fire-calls/fire-calls-8p.parquet/_committed_3872868080952213493	_committed_3872868080952213493	1634
4	dbfs:/mnt/davis/fire-calls/fire-calls-8p.parquet/_committed_vacuum5173795997807967562	_committed_vacuum5173795997807967562	96
5	dbfs:/mnt/davis/fire-calls/fire-calls-8p.parquet/_started_3872868080952213493	_started_3872868080952213493	0
6	dbfs:/mnt/davis/fire-calls/fire-calls-8p.parquet/part-00000-tid-3872868080952213493-d32b4468-0db9-431b-8ec8-1e97c3e7d738-22403-c000.snappy.parquet	part-00000-tid-3872868080952213493-d32b4468-0db9-431b-8ec8-1e97c3e7d738-22403-c000.snappy.parquet	70689109
	dbfs:/mnt/davis/fire-calls/fire-calls-8p.parquet/part-00001-tid-3872868080952213493-d32b4468-0db9-431b-8ec8-	part-00001-tid-3872868080952213493-d32b4468-0db9-431b-8ec8-1e97c3e7d738-22404-c000.snappy.parquet	70249066

Showing all 13 rows.

Command took 1.69 seconds -- by SWCPROPERTY@GMAIL.COM at 6/29/2021, 8:33:56 PM on My Cluster

Cmd 43

```
CREATE OR REPLACE TEMPORARY VIEW fireCallsParquet8
USING PARQUET
OPTIONS (
    path "/mnt/davis/fire-calls/fire-calls-8p.parquet"
  )
```

▸ (1) Spark Jobs

OK

Command took 0.95 seconds -- by SWCPROPERTY@GMAIL.COM at 6/29/2021, 8:33:59 PM on My Cluster

Cmd 44

```
SELECT count(`Incident_Number`)
FROM firecallsParquet8
WHERE Priority > 1
```

▸ (2) Spark Jobs

	count(Incident_Number)
1	4401920

```
Command took 3.37 seconds -- by SWCPROPERTY@GMAIL.COM at 6/29/2021, 8:34:02 PM on My Cluster
```

Cmd 45

> This file has 8 partitions rather than 1. Look at the speed improvement!

Cmd 46

Reading CSV

- `spark.read.csv(..)`
- There are a large number of options when reading CSV files including headers, column separator, escaping, etc.
- We can allow Spark to infer the schema at the cost of first reading in the entire file
- Large CSV files should always have a schema pre-defined

Cmd 47

Reading Parquet

- `spark.read.parquet(..)`
- Parquet files are the preferred file format for big-data
- It is a columnar file format
- It is a splittable file format
- It offers a lot of performance benefits over other formats including predicate push down
- Unlike CSV, the schema is read in, not inferred
- Reading the schema from Parquet's metadata can be extremely efficient

Cmd 48

Comparison

Type	Inference Type	Inference Speed	Reason	Should Supply Schema?
CSV	Full-Data-Read	Slow	File size	Yes
Parquet	Metadata-Read	Fast/Medium	Number of Partitions	No (most cases)
Tables	n/a	n/a	Predefined	n/a

JSON	Full-Read-Data	Slow	File size	Yes
Text	Dictated	Zero	Only 1 Column	Never
JDBC	DB-Read	Fast	DB Schema	No

Cmd 49

Reading Tables

- `spark.read.table(..)`
- The Databricks platform allows us to register a huge variety of data sources as tables via the Databricks UI
- Any `DataFrame` (from CSV, Parquet, whatever) can be registered as a temporary view
- Tables/Views can be loaded via the `DataFrameReader` to produce a `DataFrame`
- Tables/Views can be used directly in SQL statements

Cmd 50

Reading JSON

- `spark.read.json(..)`
- JSON represents complex data types unlike CSV's flat format
- Has many of the same limitations as CSV (needing to read the entire file to infer the schema)
- Like CSV has a lot of options allowing control on date formats, escaping, single vs. multiline JSON, etc.

Cmd 51

Reading Text

- `spark.read.text(..)`
- Reads one line of text as a single column named `value`
- Is the basis for more complex file formats such as fixed-width text files

Reading JDBC

- `spark.read.jdbc(..)`
- Requires one database connection per partition
- Has the potential to overwhelm the database
- Requires specification of a stride to properly balance partitions

Cmd 53

3.5 Schemas and Types

☆ In this notebook you:

- Motivate the use of schemas and types
- Read from JSON without a schema
- Read from JSON with a schema

Cmd 3

```
%run ../Includes/Classroom-Setup
```

```
Command took 9.00 seconds -- by SWCPROPERTY@GMAIL.COM at 6/29/2021, 9:04:43 PM on My Cluster
```

```
WARNING: This curriculum was written for DBR 5.5 ML. Please create a new cluster that uses the runtime DBR 5.5 ML
```

```
OK
```

Cmd 4

☆ Why Schemas Matter

Cmd 5

Schemas are at the heart of data structures in Spark. **A schema describes the structure of your data by naming columns and declaring the type of data in that column.** Rigorously enforcing schemas leads to significant performance optimizations and reliability of code.

Why is open source Spark so fast, and why is Databricks Runtime even faster? While there are many reasons for these performance improvements, two key reasons are:

- First and foremost, Spark runs first in memory rather than reading and writing to disk.
- Second, using DataFrames allows Spark to optimize the execution of your queries because it knows what your data looks like.

Two pillars of computer science education are data structures, the organization and storage of data and algorithms, and the computational procedures on that data. A rigorous understanding of computer science involves both of these domains. When you apply the most relevant data structures, the algorithms that carry out the computation become significantly more eloquent.

Schemas with Semi-Structured JSON Data

Tabular data, such as that found in CSV files or relational databases, has a formal structure where each observation, or row, of the data has a value (even if it's a NULL value) for each feature, or column, in the data set.

Semi-structured data does not need to conform to a formal data model. Instead, a given feature may appear zero, once, or many times for a given observation.

Semi-structured data storage works well with hierarchical data and with schemas that may evolve over time. One of the most common forms of semi-structured data is JSON data, which consists of attribute-value pairs.

Cmd 7

☆ Reading from JSON w/ InferSchema

Reading in JSON isn't that much different than reading in CSV files.

Let's start with taking a look at all the different options that go along with reading in JSON files.

Cmd 8

JSON Lines

Much like the CSV reader, the JSON reader also assumes:

- That there is one JSON object per line and
- That it's delineated by a new-line.

This format is referred to as **JSON Lines** or **newline-delimited JSON**

More information about this format can be found at http://jsonlines.org.

Note: *Spark 2.2 was released on July 11th 2016. With that comes File IO improvements for CSV & JSON, but more importantly,* ***support for parsing multi-line JSON and CSV files****. You can read more about that (and other features in Spark 2.2) in the Databricks Blog.*

Cmd 9

Take a look at the sample of our JSON data.

Cmd 10

```
%fs ls /mnt/davis/fire-calls/fire-calls-truncated.json
```

	path	name	size
1	dbfs:/mnt/davis/fire-calls/fire-calls-truncated.json	fire-calls-truncated.json	221798942

Showing all 1 rows.

```
Command took 0.83 seconds -- by SWCPROPERTY@GMAIL.COM at 6/29/2021, 9:04:57 PM on My Cluster
```

Cmd 11

Like we did with the CSV file, we can use **%fs head ...** to take a look at the first few lines of the file.

Cmd 12

```
%fs head /mnt/davis/fire-calls/fire-calls-truncated.json
```

```
[Truncated to first 65536 bytes]
{"Call Number":1030118,"Unit ID":"E08","Incident Number":30625,"Call Type":"Medical Incident","Call Date":"04/12/2000","Watch Date":"04/12/2000","Received DtTm":"04/12/2000 09:27:45 PM","Entry DtTm":"04/12/2000 09:28:58 PM","Dispatch DtTm":"04/12/2000 09:29:21 PM","Response DtTm":"04/12/2000 09:31:26 PM","On Scene DtTm":"04/12/2000 09:32:34 PM","Call Final Disposition":"Other","Available DtTm":"04/12/2000 09:45:28 PM","Address":"4TH ST/CHANNEL ST","City":"SF","Battalion":"B03","Station Area":"08","Box":"2226","Original Priority":"3","Priority":"3","Final Priority":3,"ALS Unit":false,"Number of Alarms":1,"Unit Type":"ENGINE","Unit sequence in call dispatch":1,"Fire Prevention District":"3","Supervisor District":"6","Neighborhoooods - Analysis Boundaries":"None","Location":"(37.7750268633971, -122.392346204303)","RowID":"001030118-E08"}
{"Call Number":1030122,"Unit ID":"M18","Incident Number":30630,"Call Type":"Medical Incident","Call Date":"04/12/2000","Watch Date":"04/12/2000","Received DtTm":"04/12/2000 09:31:55 PM","Entry DtTm":"04/12/2000 09:33:48 PM","Dispatch DtTm":"04/12/2000 09:34:10 PM","Response DtTm":"04/12/2000 09:35:59 PM","On Scene DtTm":"04/12/2000 09:45:22 PM","Call Final Disposition":"Other","Available DtTm":"04/12/2000 09:49:52 PM","Address":"1800 Block of IRVING ST","City":"SF","Zipcode of Incident":94122,"Battalion":"B08","Station Area":"22","Box":"7424","Original Priority":"1","Priority":"1","Final Priority":2,"ALS Unit":false,"Number of Alarms":1,"Unit Type":"MEDIC","Unit sequence in call dispatch":1,"Fire Prevention District":"8","Supervisor District":"4","Neighborhoooods - Analysis Boundaries":"Sunset/Parkside","Location":"(37.763482287794, -122.477678638767)","RowID":"001030122-M18"}
{"Call Number":1030154,"Unit ID":"M36","Incident Number":30662,"Call Type":"Medical Incident","Call Date":"04/12/2000","Watch Date":"04/12/2000","Received DtTm":"04/12/2000 10:43:54 PM","Entry DtTm":"04/12/2000 10:45:53 PM","Dispatch DtTm":"04/12/2000 10:49:59 PM","Response DtTm":"04/12/2000 10:50:35 PM","On Scene DtTm":"04/12/2000 10:53:18 PM","Transport DtTm":"04/12/2000 11:11:36 PM","Hospital DtTm":"04/12/2000 11:22:17 PM","Call Final Disposition":"Other","Available DtTm":"04/12/2000 11:42:43 PM","Address":"0 Block of SOUTH VAN NESS AVE","City":"SF","Zipcode of Incident":94103,"Battalion":"B02","Station Area":"36","Box":"5177","Original Priority":"1","Priority":"1","Final Priority":2,"ALS Unit":false,"Number of Alarms":1,"Unit Type":"MEDIC","Unit sequence in call dispatch":1,"Fire Prevention District":"2","Supervisor District":"6","Neighborhoooods - Analysis Boundaries":"Mission","Location":"(37.7741251002903, -122.41881021183)","RowID":"001030154-M36"}
{"Call Number":1040007,"Unit ID":"E12","Incident Number":30697,"Call Type":"Structure Fire","Call Date":"04/13/2000","Watch Date":"04/12/2000","Received DtTm":"04/13/2000 12:19:54 AM","Entry DtTm":"04/13/2000 12:29:24 AM","Dispatch DtTm":"04/13/2000 12:29:35 AM","Response DtTm":"04/13/2000 12:31:25 AM","On Scene DtTm":"04/13/2000 12:32:36 AM","Call Final Disposition":"Other","Available DtTm":"04/13/2000 12:33:18 AM","Address":"CLAYTON ST/PARNASSUS AV","City":"SF","Zipcode of Incident":94117,"Battalion":"B05","Station Area":"12","Box":"5151","Original Priority":"3","Priority":"3","Final Priority":3,"ALS Unit":true,"Number of Alarms":1,"Unit Type":"ENGINE","Unit sequence in call dispatch":1,"Fire Prevention District":"5","Supervisor District":"5","Neighborhoooods - Analysis Boundaries":"Haight Ashbury","Location":"(37.7651387353822, -122.44763462758)","RowID":"001040007-E12"}
{"Call Number":1040021,"Unit ID":"M14","Incident Number":30711,"Call Type":"Medical Incident","Call Date":"04/13/2000","Watch Date":"04/12/2000","Received DtTm":"04/13/2000 01:17:25 AM","Entry DtTm":"04/13/2000 01:18:44 AM","Dispatch DtTm":"04/13/2000 01:20:02 AM","Response DtTm":"04/13/2000 01:21:40 AM","On Scene DtTm":"04/13/2000 01:24:05 AM","Transport DtTm":"04/13/2000 01:56:02 AM","Hospital DtTm":"04/13/2000 02:33:33 AM","Call Final Disposition":"Other","Available DtTm":"04/13/2000 02:40:25 AM","Address":"500 Block of 38TH AVE","City":"SF","Zipcode of Incident":94121,"Battalion":"B07","Station Area":"34","Box":"7255","Original Priority":"3","Priority":"3","Final Pri
Command took 0.72 seconds -- by SWCPROPERTY@GMAIL.COM at 6/29/2021, 9:05:02 PM on My Cluster
```

Read the JSON File

The command to read in JSON looks very similar to that of CSV.

In addition to reading the JSON file, we will also print the resulting schema.

Cmd 14

```
1  CREATE OR REPLACE TEMPORARY VIEW fireCallsJSON
2  USING JSON
3  OPTIONS (
4      path "/mnt/davis/fire-calls/fire-calls-truncated.json"
5    )
```

▸ (1) Spark Jobs

OK

Command took 9.92 seconds -- by SWCPROPERTY@GMAIL.COM at 6/29/2021, 9:07:39 PM on My Cluster

Cmd 15

```
1  DESCRIBE fireCallsJSON
```

	col_name	data_type	comment
1	ALS Unit	boolean	null
2	Address	string	null
3	Available DtTm	string	null
4	Battalion	string	null
5	Box	string	null
6	Call Date	string	null
7	Call Final Disposition	string	null

Showing all 34 rows.

Command took 0.09 seconds -- by SWCPROPERTY@GMAIL.COM at 6/29/2021, 9:07:53 PM on My Cluster

Cmd 16

Take a look at the table.

3.5-Schemas-and-Types (SQL)

My Cluster | File | Edit | View: Standard | Permissions | Run All | Clear | Publish | Comments | Experiment | Revision hist

```
1  SELECT * FROM fireCallsJSON LIMIT 10
```

(1) Spark Jobs

	ALS Unit	Address	Available DtTm	Battalion	Box	Call Date	Call Final Disposition	Call Number	Call Type	Call Type Group	City	Dispatch DtTm	Entry DtTm
1	false	4TH ST/CHANNEL ST	04/12/2000 09:45:28 PM	B03	2226	04/12/2000	Other	1030118	Medical Incident	null	SF	04/12/2000 09:29:21 PM	04/12/2000 0
2	false	1800 Block of IRVING ST	04/12/2000 09:49:52 PM	B08	7424	04/12/2000	Other	1030122	Medical Incident	null	SF	04/12/2000 09:34:10 PM	04/12/2000 0
3	false	0 Block of SOUTH VAN NESS AVE	04/12/2000 11:42:43 PM	B02	5117	04/12/2000	Other	1030154	Medical Incident	null	SF	04/12/2000 10:49:59 PM	04/12/2000 1
4	true	CLAYTON ST/PARNASSUS AV	04/13/2000 12:33:18 AM	B05	5151	04/13/2000	Other	1040007	Structure Fire	null	SF	04/13/2000 12:29:35 AM	04/13/2000 1
5	true	500 Block of 38TH AVE	04/13/2000 02:40:25 AM	B07	7255	04/13/2000	Other	1040021	Medical Incident	null	SF	04/13/2000 01:20:02 AM	04/13/2000 0
6	true	200 Block of MADRID ST	04/13/2000 09:26:54 AM	B09	0613	04/13/2000	Other	1040061	Medical Incident	null	SF	04/13/2000 07:55:54 AM	04/13/2000 0
7	false	2800 Block of BROADWAY	04/13/2000 09:39:36 AM	B04	4226	04/13/2000	Other	1040079	Alarms	null	SF	04/13/2000 09:34:10 AM	04/13/2000 0

Showing all 10 rows.

Command took 0.68 seconds -- by SWCPROPERTY@GMAIL.COM at 6/29/2021, 9:07:56 PM on My Cluster

Cmd 18

Review: Reading from JSON w/ InferSchema

While there are similarities between reading in CSV & JSON there are some key differences:

- We only need one job even when inferring the schema.
- There is no header which is why there isn't a second job in this case - the column names are extracted from the JSON object's attributes.
- Unlike CSV which reads in 100% of the data, the JSON reader only samples the data.

  Note: In Spark 2.2 the behavior was changed to read in the entire JSON file.

Cmd 19

☆ Reading from JSON w/ a User Defined Schema

Cmd 20

User-Defined Schemas

Spark infers schemas from the data, as detailed in the example above. Challenges with inferred schemas include:

- Schema inference means Spark scans all of your data, creating an extra job, which can affect performance
- Consider providing alternative data types (for example, change a `Long` to a `Integer`)
- Consider throwing out certain fields in the data, to read only the data of interest

```
CREATE OR REPLACE TEMPORARY VIEW fireCallsJSON (
  `Call Number` INT,
  `Unit ID` STRING,
  `Incident Number` INT,
  `Call Type` STRING,
  `Call Date` STRING,
  `Watch Date` STRING,
  `Received DtTm` STRING,
  `Entry DtTm` STRING,
  `Dispatch DtTm` STRING,
  `Response DtTm` STRING,
  `On Scene DtTm` STRING,
  `Transport DtTm` STRING,
  `Hospital DtTm` STRING,
  `Call Final Disposition` STRING,
  `Available DtTm` STRING,
  `Address` STRING,
  `City` STRING,
  `Zipcode of Incident` INT,
  `Battalion` STRING,
  `Station Area` STRING,
  `Box` STRING,
  `Original Priority` STRING,
  `Priority` STRING,
  `Final Priority` INT,
  `ALS Unit` BOOLEAN,
  `Call Type Group` STRING,
  `Number of Alarms` INT,
  `Unit Type` STRING,
  `Unit sequence in call dispatch` INT,
  `Fire Prevention District` STRING,
  `Supervisor District` STRING,
  `Neighborhooods - Analysis Boundaries` STRING,
  `Location` STRING,
  `RowID` STRING
)
USING JSON
OPTIONS (
    path "/mnt/davis/fire-calls/fire-calls-truncated.json"
)
```

OK

Command took 0.27 seconds -- by SWCPROPERTY@GMAIL.COM at 6/29/2021, 9:08:52 PM on My Cluster

Cmd 22

Take a look at how much faster that process was!

Cmd 23

Primitive and Non-primitive Types

The Spark `types` package provides the building blocks for constructing schemas.

A primitive type contains the data itself. The most common primitive types include:

Numeric	General	Time
FloatType	StringType	TimestampType
IntegerType	BooleanType	DateType
DoubleType	NullType	
LongType		
ShortType		

Non-primitive types are sometimes called reference variables or composite types. Technically, non-primitive types contain references to memory locations and not the data itself. Non-primitive types are the composite of a number of primitive types such as an Array of the primitive type `Integer`.

The two most common composite types are `ArrayType` and `MapType`. These types allow for a given field to contain an arbitrary number of elements in either an Array/List or Map/Dictionary form.

See the Spark documentation for a complete picture of types in Spark.

Cmd 24

Review: Reading from JSON w/ User-Defined Schema

- Just like CSV, providing the schema avoids the extra jobs.
- The schema allows us to rename columns and specify alternate data types.
- Can get arbitrarily complex in its structure.

3.6 Writing Data

In this notebook you:

- Employ the writing design pattern
- Control partitions for database writes using REPARTITION and COALESCE hints

Cmd 3

```
%run ../Includes/Classroom-Setup
```

Command took 7.10 seconds -- by SWCPROPERTY@GMAIL.COM at 6/29/2021, 9:24:03 PM on My Cluster

WARNING: This curriculum was written for DBR 5.5 ML. Please create a new cluster that uses the runtime DBR 5.5 ML

OK

Cmd 4

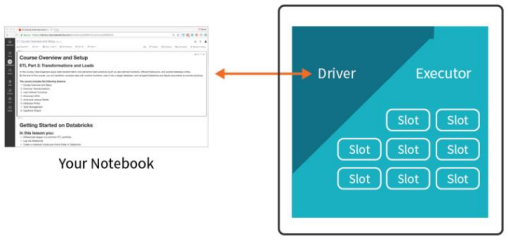
Writing From Spark

Cmd 5

Writing to a database in Spark differs from other tools largely due to its distributed nature. There are a number of variables that can be tweaked to optimize performance, largely relating to how data is organized on the cluster. Partitions are the first step in understanding performant database connections.

A partition is a portion of your total data set, which is divided into many of these portions so Spark can distribute your work across a cluster.

The other concept needed to understand Spark's computation is a slot (also known as a core). **A slot/core is a resource available for the execution of computation in parallel.** In brief, a partition refers to the distribution of data while a slot refers to the distribution of computation.



As a general rule of thumb, the number of partitions should be a multiple of the number of cores. For instance, with 5 partitions and 8 slots, 3 of the slots will be underutilized. With 9 partitions and 8 slots, a job will take twice as long as it waits for the extra partition to finish.

Cmd 6

Import the dataset.

Cmd 7

```
CREATE OR REPLACE TEMPORARY VIEW fireCallsCSV
USING CSV
OPTIONS (
    path "/mnt/davis/fire-calls/fire-calls-truncated.csv",
    header "true",
    sep ":"
  )
```

▸ (1) Spark Jobs

OK

Command took 0.84 seconds -- by SWCPROPERTY@GMAIL.COM at 6/29/2021, 9:25:22 PM on My Cluster

Command took 0.84 seconds -- by SWCPROPERTY@GMAIL.COM at 6/29/2021, 9:25:22 PM on My Cluster

Cmd 8

Switch to the Python API to run write commands. You can do this `%python` and then run `sql(" <My Normal SQL Query> ")`

Cmd 9

```
%python
df = sql("SELECT * FROM fireCallsCSV")

display(df)
```

▸ (1) Spark Jobs

▸ df: pyspark.sql.dataframe.DataFrame = [Call Number: string, Unit ID: string ... 32 more fields]

	Call Number	Unit ID	Incident Number	Call Type	Call Date	Watch Date	Received DtTm	Entry DtTm	Dispatch DtTm	Response DtTm	On Scene DtTm
1	001030118	E08	00030625	Medical Incident	04/12/2000	04/12/2000	04/12/2000 09:27:45 PM	04/12/2000 09:28:58 PM	04/12/2000 09:29:21 PM	04/12/2000 09:31:26 PM	04/12/2000 09:32:34 PM
2	001030122	M18	00030630	Medical Incident	04/12/2000	04/12/2000	04/12/2000 09:31:55 PM	04/12/2000 09:33:48 PM	04/12/2000 09:34:10 PM	04/12/2000 09:35:59 PM	04/12/2000 09:45:22 PM
3	001030154	M36	00030662	Medical Incident	04/12/2000	04/12/2000	04/12/2000 10:43:54 PM	04/12/2000 10:45:53 PM	04/12/2000 10:49:59 PM	04/12/2000 10:50:35 PM	04/12/2000 10:53:18 PM
4	001040007	E12	00030697	Structure Fire	04/13/2000	04/12/2000	04/13/2000 12:19:54 AM	04/13/2000 12:29:24 AM	04/13/2000 12:29:35 AM	04/13/2000 12:31:25 AM	04/13/2000 12:32:36 AM
5	001040021	M14	00030711	Medical Incident	04/13/2000	04/12/2000	04/13/2000 01:17:25 AM	04/13/2000 01:18:44 AM	04/13/2000 01:20:02 AM	04/13/2000 01:21:40 AM	04/13/2000 01:24:05 AM
6	001040061	M43	00030749	Medical Incident	04/13/2000	04/12/2000	04/13/2000 07:51:29 AM	04/13/2000 07:55:35 AM	04/13/2000 07:55:54 AM	04/13/2000 07:59:58 AM	null
7	001040079	E10	00030766	Alarms	04/13/2000	04/13/2000	04/13/2000 09:31:19 AM	04/13/2000 09:33:04 AM	04/13/2000 09:34:10 AM	04/13/2000 09:35:52 AM	04/13/2000 09:37:59 AM

Truncated results, showing first 1000 rows.

Command took 0.99 seconds -- by SWCPROPERTY@GMAIL.COM at 6/29/2021, 9:27:02 PM on My Cluster

Cmd 10

Write using the `.write(" < path> ")` method on the DataFrame.

Cmd 11

```
%python
df.write.mode("OVERWRITE").csv(username + "/fire-calls.csv")
```

▸ (1) Spark Jobs

Command took 16.32 seconds -- by SWCPROPERTY@GMAIL.COM at 6/29/2021, 9:27:05 PM on My Cluster

Take a look at the file we just wrote.

Cmd 13

```
%python
dbutils.fs.ls(username + "/fire-calls.csv")
```

```
Out[4]: [FileInfo(path='dbfs:/SWCPROPERTY@GMAIL.COM/fire-calls.csv/_SUCCESS', name='_SUCCESS', size=0),
 FileInfo(path='dbfs:/SWCPROPERTY@GMAIL.COM/fire-calls.csv/_committed_4696862914843696193', name='_committed_4696862914843696193', size=736),
 FileInfo(path='dbfs:/SWCPROPERTY@GMAIL.COM/fire-calls.csv/_started_4696862914843696193', name='_started_4696862914843696193', size=0),
 FileInfo(path='dbfs:/SWCPROPERTY@GMAIL.COM/fire-calls.csv/part-00000-tid-4696862914843696193-c27ede8b-f5a1-4feb-87bb-fcb046eb7ac8-107-1-c000.csv', name='part-00000-tid-4696862914843696193-c27ede8b-f5a1-4feb-87bb-fcb046eb7ac8-107-1-c000.csv', size=11656129),
 FileInfo(path='dbfs:/SWCPROPERTY@GMAIL.COM/fire-calls.csv/part-00001-tid-4696862914843696193-c27ede8b-f5a1-4feb-87bb-fcb046eb7ac8-108-1-c000.csv', name='part-00001-tid-4696862914843696193-c27ede8b-f5a1-4feb-87bb-fcb046eb7ac8-108-1-c000.csv', size=11655265),
 FileInfo(path='dbfs:/SWCPROPERTY@GMAIL.COM/fire-calls.csv/part-00002-tid-4696862914843696193-c27ede8b-f5a1-4feb-87bb-fcb046eb7ac8-109-1-c000.csv', name='part-00002-tid-4696862914843696193-c27ede8b-f5a1-4feb-87bb-fcb046eb7ac8-109-1-c000.csv', size=11655614),
 FileInfo(path='dbfs:/SWCPROPERTY@GMAIL.COM/fire-calls.csv/part-00003-tid-4696862914843696193-c27ede8b-f5a1-4feb-87bb-fcb046eb7ac8-110-1-c000.csv', name='part-00003-tid-4696862914843696193-c27ede8b-f5a1-4feb-87bb-fcb046eb7ac8-110-1-c000.csv', size=11654117),
 FileInfo(path='dbfs:/SWCPROPERTY@GMAIL.COM/fire-calls.csv/part-00004-tid-4696862914843696193-c27ede8b-f5a1-4feb-87bb-fcb046eb7ac8-111-1-c000.csv', name='part-00004-tid-4696862914843696193-c27ede8b-f5a1-4feb-87bb-fcb046eb7ac8-111-1-c000.csv', size=11662937),
 FileInfo(path='dbfs:/SWCPROPERTY@GMAIL.COM/fire-calls.csv/part-00005-tid-4696862914843696193-c27ede8b-f5a1-4feb-87bb-fcb046eb7ac8-112-1-c000.csv', name='part-00005-tid-4696862914843696193-c27ede8b-f5a1-4feb-87bb-fcb046eb7ac8-112-1-c000.csv', size=11673999),
 FileInfo(path='dbfs:/SWCPROPERTY@GMAIL.COM/fire-calls.csv/part-00006-tid-4696862914843696193-c27ede8b-f5a1-4feb-87bb-fcb046eb7ac8-113-1-c000.csv', name='part-00006-tid-4696862914843696193-c27ede8b-f5a1-4feb-87bb-fcb046eb7ac8-113-1-c000.csv', size=11674564),
 FileInfo(path='dbfs:/SWCPROPERTY@GMAIL.COM/fire-calls.csv/part-00007-tid-4696862914843696193-c27ede8b-f5a1-4feb-87bb-fcb046eb7ac8-114-1-c000.csv', name='part-00007-tid-4696862914843696193-c27ede8b-f5a1-4feb-87bb-fcb046eb7ac8-114-1-c000.csv', size=7589695)]
```

Command took 0.26 seconds -- by SWCPROPERTY@GMAIL.COM at 6/29/2021, 9:28:24 PM on My Cluster

Cmd 14

Notice that it wrote a number of different parts, one for each partition. Now let's get a sense for how to check how many partitions our data has, which once again is in Python.

Cmd 15

```
%python
df.rdd.getNumPartitions()
```

```
Out[5]: 8
```

Command took 0.10 seconds -- by SWCPROPERTY@GMAIL.COM at 6/29/2021, 9:28:29 PM on My Cluster

Controlling Concurrency

Cmd 17

In the context of JDBC database writes, **the number of partitions determine the number of connections used to push data through the JDBC API.** There are two ways to control this parallelism:

Hint	Transformation Type	Use	Evenly distributes data across partitions?
`SELECT /*+ COALESCE(n) */`	narrow (does not shuffle data)	reduce the number of partitions	no
`SELECT /*+ REPARTITION(n) */`	wide (includes a shuffle operation)	increase the number of partitions	yes

Cmd 18

Create a new temporary view that coalesces all of our data to a single partition. This can be done using a hint.

Cmd 19

```
CREATE OR REPLACE TEMPORARY VIEW fireCallsCSV1p
  AS
SELECT /*+ COALESCE(1) */ *
FROM fireCallsCSV
```

OK

Command took 0.26 seconds -- by SWCPROPERTY@GMAIL.COM at 6/29/2021, 9:29:30 PM on My Cluster

Cmd 20

Now check the number of partitions.

Cmd 21

```
%python
sql("SELECT * FROM fireCallsCSV1p").rdd.getNumPartitions()
```

Out[6]: 1

Command took 0.18 seconds -- by SWCPROPERTY@GMAIL.COM at 6/29/2021, 9:29:34 PM on My Cluster

Cmd 22

We now have a single partition. What if we want more partitions? Let's use a `REPARTITION` hint to evenly distribute our data across 8 partitions.

```
CREATE OR REPLACE TEMPORARY VIEW fireCallsCSV8p
  AS
SELECT /*+ REPARTITION(8) */ *
FROM fireCallsCSV
```

OK

Command took 0.18 seconds -- by SWCPROPERTY@GMAIL.COM at 6/29/2021, 9:42:57 PM on My Cluster

Cmd 24

```
%python
sql("SELECT * FROM fireCallsCSV8p").rdd.getNumPartitions()
```

▸ (1) Spark Jobs

Out[7]: 8

Command took 6.82 seconds -- by SWCPROPERTY@GMAIL.COM at 6/29/2021, 9:42:58 PM on My Cluster

Cmd 25

Now you can save the results.

Cmd 26

```
%python
sql("SELECT * FROM fireCallsCSV8p").write.mode("OVERWRITE").csv(username + "/fire-calls-repartitioned.csv")
```

▸ (2) Spark Jobs

Command took 14.90 seconds -- by SWCPROPERTY@GMAIL.COM at 6/29/2021, 9:43:35 PM on My Cluster

Cmd 27

Click the arrow next to `Spark Jobs` under the following code cell in order to see a breakdown of the job you triggered. Click the next arrow to see a breakdown of the stages.

When you repartitioned the DataFrame to 8 partitions, 8 stages were needed, one to write each partition of the data.

Cmd 28

View the results.

```
%python
dbutils.fs.ls(username + "/fire-calls-repartitioned.csv")
```

```
Out[10]: [FileInfo(path='dbfs:/SWCPROPERTY@GMAIL.COM/fire-calls-repartitioned.csv/_SUCCESS', name='_SUCCESS', size=0),
 FileInfo(path='dbfs:/SWCPROPERTY@GMAIL.COM/fire-calls-repartitioned.csv/_committed_4850019547792766540', name='_committed_4850019547792766540', size=736),
 FileInfo(path='dbfs:/SWCPROPERTY@GMAIL.COM/fire-calls-repartitioned.csv/_committed_8947687671900415182', name='_committed_8947687671900415182', size=1458),
 FileInfo(path='dbfs:/SWCPROPERTY@GMAIL.COM/fire-calls-repartitioned.csv/_started_4850019547792766540', name='_started_4850019547792766540', size=0),
 FileInfo(path='dbfs:/SWCPROPERTY@GMAIL.COM/fire-calls-repartitioned.csv/_started_8947687671900415182', name='_started_8947687671900415182', size=0),
 FileInfo(path='dbfs:/SWCPROPERTY@GMAIL.COM/fire-calls-repartitioned.csv/part-00000-tid-8947687671900415182-00bc201e-cca0-45f0-a1d7-611f21cb9cd4-147-1-c000.csv', name='part-00000-tid-8947687671900415182-00bc201e-cca0-45f0-a1d7-
611f21cb9cd4-147-1-c000.csv', size=1116157?),
 FileInfo(path='dbfs:/SWCPROPERTY@GMAIL.COM/fire-calls-repartitioned.csv/part-00001-tid-8947687671900415182-00bc201e-cca0-45f0-a1d7-611f21cb9cd4-148-1-c000.csv', name='part-00001-tid-8947687671900415182-00bc201e-cca0-45f0-a1d7-
611f21cb9cd4-148-1-c000.csv', size=11150515),
 FileInfo(path='dbfs:/SWCPROPERTY@GMAIL.COM/fire-calls-repartitioned.csv/part-00002-tid-8947687671900415182-00bc201e-cca0-45f0-a1d7-611f21cb9cd4-149-1-c000.csv', name='part-00002-tid-8947687671900415182-00bc201e-cca0-45f0-a1d7-
611f21cb9cd4-149-1-c000.csv', size=11151977),
 FileInfo(path='dbfs:/SWCPROPERTY@GMAIL.COM/fire-calls-repartitioned.csv/part-00003-tid-8947687671900415182-00bc201e-cca0-45f0-a1d7-611f21cb9cd4-150-1-c000.csv', name='part-00003-tid-8947687671900415182-00bc201e-cca0-45f0-a1d7-
611f21cb9cd4-150-1-c000.csv', size=11146159),
 FileInfo(path='dbfs:/SWCPROPERTY@GMAIL.COM/fire-calls-repartitioned.csv/part-00004-tid-8947687671900415182-00bc201e-cca0-45f0-a1d7-611f21cb9cd4-151-1-c000.csv', name='part-00004-tid-8947687671900415182-00bc201e-cca0-45f0-a1d7-
611f21cb9cd4-151-1-c000.csv', size=11148942),
 FileInfo(path='dbfs:/SWCPROPERTY@GMAIL.COM/fire-calls-repartitioned.csv/part-00005-tid-8947687671900415182-00bc201e-cca0-45f0-a1d7-611f21cb9cd4-152-1-c000.csv', name='part-00005-tid-8947687671900415182-00bc201e-cca0-45f0-a1d7-
611f21cb9cd4-152-1-c000.csv', size=11155668),
 FileInfo(path='dbfs:/SWCPROPERTY@GMAIL.COM/fire-calls-repartitioned.csv/part-00006-tid-8947687671900415182-00bc201e-cca0-45f0-a1d7-611f21cb9cd4-153-1-c000.csv', name='part-00006-tid-8947687671900415182-00bc201e-cca0-45f0-a1d7-
611f21cb9cd4-153-1-c000.csv', size=11150977),
 FileInfo(path='dbfs:/SWCPROPERTY@GMAIL.COM/fire-calls-repartitioned.csv/part-00007-tid-8947687671900415182-00bc201e-cca0-45f0-a1d7-611f21cb9cd4-154-1-c000.csv', name='part-00007-tid-8947687671900415182-00bc201e-cca0-45f0-a1d7-
611f21cb9cd4-154-1-c000.csv', size=11156505)]
Command took 0.25 seconds -- by SWCPROPERTY@GMAIL.COM at 6/29/2021, 9:44:47 PM on My Cluster
```

Cmd 30

Summary

- Alter the number of concurrent writes using `SELECT /*+ COALESCE(n) */` or `SELECT /*+ REPARTITION(n) */`
- Note that `coalesce` means something different in ANSI SQL (that is, to return the first non-null value)
- Files like CSV and Parquet might look like single files, but Spark saves them as a folder of different parts

Cmd 31

3.7 Managed and Unmanaged Tables

☆ In this notebook you:

- Write to managed and unmanaged tables
- Explore the effect of dropping tables on the metadata and underlying data

Cmd 3

```
%run ../Includes/Classroom-Setup
```

```
Command took 6.85 seconds -- by SWCPROPERTY@GMAIL.COM at 6/29/2021, 9:47:55 PM on My Cluster

WARNING: This curriculum was written for DBR 5.5 ML. Please create a new cluster that uses the runtime DBR 5.5 ML

OK
```

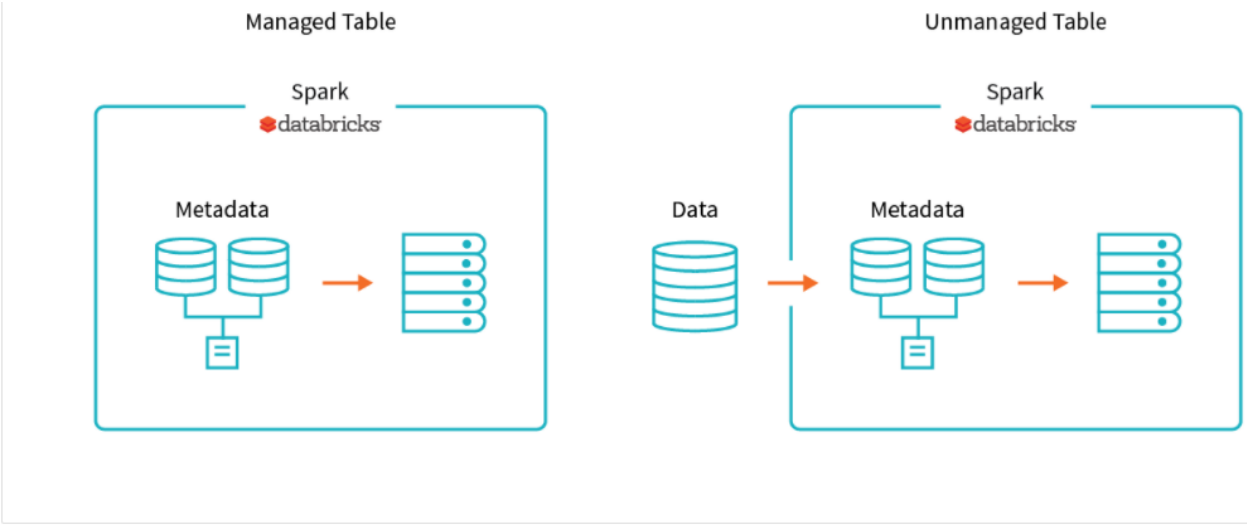
Cmd 4

☆ Managed and Unmanaged Tables

Cmd 5

A **managed table** is a table that manages both the data itself as well as the metadata. In this case, a `DROP TABLE` command removes both the metadata for the table as well as the data itself.

Unmanaged tables manage the metadata from a table such as the schema and data location, but the data itself sits in a different location, often backed by a blob store like the Azure Blob or S3. Dropping an unmanaged table drops only the metadata associated with the table while the data itself remains in place.



Cmd 6

Start with a managed table.

Cmd 7

```
USE default;

DROP TABLE IF EXISTS tableManaged;

CREATE TABLE tableManaged (
  var1 INT,
  var2 INT
);

INSERT INTO tableManaged
  VALUES (1, 1), (2, 2)
```

▸ (7) Spark Jobs

	num_affected_rows	num_inserted_rows
1	2	2

Use `DESCRIBE EXTENDED` to describe the contents of the table. Scroll down to see the table `Type`.

Notice the location is also `dbfs:/user/hive/warehouse/< your database >/tablemanaged`.

Cmd 9

```
1  DESCRIBE EXTENDED tableManaged
```

	col_name	data_type	comment
1	var1	int	
2	var2	int	
3			
4	# Partitioning		
5	Not partitioned		
6			
7	# Detailed Table Information		

Showing all 12 rows.

Command took 0.32 seconds -- by SWCPROPERTY@GMAIL.COM at 6/29/2021, 9:53:39 PM on My Cluster

Cmd 10

Now use an external, or unmanaged, table

Cmd 11

```
1  DROP TABLE IF EXISTS tableUnmanaged;
2
3  CREATE EXTERNAL TABLE tableUnmanaged (
4    var1 INT,
5    var2 INT
6  )
7  STORED AS parquet
8  LOCATION '/tmp/unmanagedTable'
```

OK

Command took 0.37 seconds -- by SWCPROPERTY@GMAIL.COM at 6/29/2021, 9:53:44 PM on My Cluster

Describe the table and look for the `Type`

Cmd 13

```
1  DESCRIBE EXTENDED tableUnmanaged
```

	col_name	data_type	comment
1	var1	int	null
2	var2	int	null
3			
4	# Detailed Table Information		
5	Database	default	
6	Table	tableunmanaged	
7	Owner	root	

Showing all 19 rows.

Command took 0.11 seconds -- by SWCPROPERTY@GMAIL.COM at 6/29/2021, 9:53:48 PM on My Cluster

Cmd 14

This is an external, or managed table. If we were to shut down our cluster, this data will persist. Now insert values into the table.

Cmd 15

```
1  INSERT INTO tableUnmanaged
2    VALUES (1, 1), (2, 2)
```

▸ (1) Spark Jobs

OK

Command took 2.53 seconds -- by SWCPROPERTY@GMAIL.COM at 6/29/2021, 9:54:59 PM on My Cluster

Cmd 16

Take a look at the result.

Cmd 17

```
1  SELECT * FROM tableUnmanaged
```

▸ (2) Spark Jobs

	var1	var2
1	1	1
2	2	2

Showing all 2 rows.

Command took 1.21 seconds -- by SWCPROPERTY@GMAIL.COM at 6/29/2021, 9:55:20 PM on My Cluster

Cmd 18

Now view the underlying files in where the data was persisted.

Cmd 19

```
1  %fs ls /tmp/unmanagedTable
```

	path	name	size
1	dbfs:/tmp/unmanagedTable/_SUCCESS	_SUCCESS	0
2	dbfs:/tmp/unmanagedTable/_committed_9163936857881258695	_committed_9163936857881258695	224
3	dbfs:/tmp/unmanagedTable/_started_9163936857881258695	_started_9163936857881258695	0
4	dbfs:/tmp/unmanagedTable/part-00000-tid-9163936857881258695-6bc7ec1b-d19e-4bed-a7d6-edcd8ced1141-164-1-c000.snappy.parquet	part-00000-tid-9163936857881258695-6bc7ec1b-d19e-4bed-a7d6-edcd8ced1141-164-1-c000.snappy.parquet	680
5	dbfs:/tmp/unmanagedTable/part-00001-tid-9163936857881258695-6bc7ec1b-d19e-4bed-a7d6-edcd8ced1141-165-1-c000.snappy.parquet	part-00001-tid-9163936857881258695-6bc7ec1b-d19e-4bed-a7d6-edcd8ced1141-165-1-c000.snappy.parquet	680

Showing all 5 rows.

Command took 1.37 seconds -- by SWCPROPERTY@GMAIL.COM at 6/29/2021, 9:56:12 PM on My Cluster

☆ Dropping Managed and Unmanaged Tables

Cmd 21

Confirm that the underlying files exist for the managed table.

Cmd 22

```
%fs ls dbfs:/user/hive/warehouse/tablemanaged
```

	path	name	size
1	dbfs:/user/hive/warehouse/tablemanaged/_delta_log/	_delta_log/	0
2	dbfs:/user/hive/warehouse/tablemanaged/part-00000-10bff7e2-1f19-442e-8223-9e3b13ef8234-c000.snappy.parquet	part-00000-10bff7e2-1f19-442e-8223-9e3b13ef8234-c000.snappy.parquet	694
3	dbfs:/user/hive/warehouse/tablemanaged/part-00001-198e8c77-7c39-460d-8270-e62897c1ff92-c000.snappy.parquet	part-00001-198e8c77-7c39-460d-8270-e62897c1ff92-c000.snappy.parquet	694

Showing all 3 rows.

Command took 1.13 seconds -- by SWCPROPERTY@GMAIL.COM at 6/29/2021, 9:56:19 PM on My Cluster

Cmd 23

Now drop the managed table.

Cmd 24

```
DROP TABLE tableManaged
```

OK

Command took 2.44 seconds -- by SWCPROPERTY@GMAIL.COM at 6/29/2021, 9:57:22 PM on My Cluster

Cmd 25

Take a look--the files are gone!

Cmd 26

```
%fs ls dbfs:/user/hive/warehouse/tablemanaged
```

SparkSQL 103 Engineering Data Pipelines.docx - Word

⊞FileNotFoundException: /user/hive/warehouse/tablemanaged

Command took 0.59 seconds -- by SWCPROPERTY@GMAIL.COM at 6/29/2021, 9:58:11 PM on My Cluster

Cmd 27

Now drop the unmanaged, or external, table.

Cmd 28

```
DROP TABLE tableUnmanaged
```

OK

Command took 0.50 seconds -- by SWCPROPERTY@GMAIL.COM at 6/29/2021, 9:58:15 PM on My Cluster

Cmd 29

Now take a look at the underlying files.

Cmd 30

```
%fs ls /tmp/unmanagedTable
```

	path	name	size
1	dbfs:/tmp/unmanagedTable/_SUCCESS	_SUCCESS	0
2	dbfs:/tmp/unmanagedTable/_committed_9163936857881258695	_committed_9163936857881258695	224
3	dbfs:/tmp/unmanagedTable/_started_9163936857881258695	_started_9163936857881258695	0
4	dbfs:/tmp/unmanagedTable/part-00000-tid-9163936857881258695-6bc7ec1b-d19e-4bed-a7d6-edcd8ced1141-164-1-c000.snappy.parquet	part-00000-tid-9163936857881258695-6bc7ec1b-d19e-4bed-a7d6-edcd8ced1141-164-1-c000.snappy.parquet	680
5	dbfs:/tmp/unmanagedTable/part-00001-tid-9163936857881258695-6bc7ec1b-d19e-4bed-a7d6-edcd8ced1141-165-1-c000.snappy.parquet	part-00001-tid-9163936857881258695-6bc7ec1b-d19e-4bed-a7d6-edcd8ced1141-165-1-c000.snappy.parquet	680

Showing all 5 rows.

Command took 0.41 seconds -- by SWCPROPERTY@GMAIL.COM at 6/29/2021, 9:58:34 PM on My Cluster

Cmd 31

They're still there!

Summary

- Use external/unmanaged tables when you want to persist your data once the cluster has shut down
- Use managed tables when you only want ephemeral data

Cmd 33

Engineering Data Pipelines

Module 3 Assignment

In this assignment you:

- Create a table with persistent data and a specified schema
- Populate table with specific entries
- Change partition number to compare query speeds

For each **bold** question, input its answer in Coursera.

Cmd 3

```
1 %run ../Includes/Classroom-Setup
```

```
Command took 6.26 seconds -- by SWCPROPERTY@GMAIL.COM at 6/29/2021, 10:05:00 PM on My Cluster


WARNING: This curriculum was written for DBR 5.5 ML. Please create a new cluster that uses the runtime DBR 5.5 ML


OK
```

Cmd 4

Create a table whose data will remain after you drop the table and after the cluster shuts down. Name this table `newTable` and specify the location to be at `/tmp/newTableLoc`

Set up the table to have the following schema:

```
`Address` STRING,
`City` STRING,
`Battalion` STRING,
`Box` STRING,
```

Run the following cell first to remove any files stored at `/tmp/newTableLoc` before creating our table. Be sure to first re-run that cell each time you create `newTable` .

This course was designed to work with Databricks Runtime 5.5 LTS ML, which uses Spark 2.4. If you are running a later version of the Databricks Runtime, you might have to add an additional `STORED AS parquet` to your query due to a bug.

```python
%python
# removes files stored at '/tmp/newTableLoc'
dbutils.fs.rm("/tmp/newTableLoc", True)
```

Out[2]: False

Command took 0.20 seconds -- by SWCPROPERTY@GMAIL.COM at 6/29/2021, 10:07:31 PM on My Cluster

Cmd 6

```
Drop table if exists newTable;
Create external table newTable (
`Address` STRING,
`City` STRING,
`Battalion` STRING,
`Box` STRING
)
Stored as parquet
Location '/tmp/newTableLoc'
```

OK

Command took 0.48 seconds -- by SWCPROPERTY@GMAIL.COM at 6/29/2021, 10:19:28 PM on My Cluster

Cmd 7

Check that the data type of each column is what we want.

Question 1

What type of table is `newTable` ? "EXTERNAL" or "MANAGED"?

Cmd 8

```
DESCRIBE EXTENDED newTable
```

	col_name	data_type	comment
12	Created By	Spark 3.1.0	
13	Type	EXTERNAL	
14	Provider	hive	
15	Table Properties	[transient_lastDdlTime=1625030369]	
16	Location	dbfs:/tmp/newTableLoc	
17	Serde Library	org.apache.hadoop.hive.ql.io.parquet.serde.ParquetHiveSerDe	

Run the following cell to read in the data stored at `/mnt/davis/fire-calls/fire-calls-truncated.json`. Check that the columns of the data are of the correct types (not all strings).

Cmd 10

```
CREATE OR REPLACE TEMPORARY VIEW  fireCallsJSON (
  `ALS Unit` boolean,
  `Address` string,
  `Available DtTm` string,
  `Battalion` string,
  `Box` string,
  `Call Date` string,
  `Call Final Disposition` string,
  `Call Number` long,
  `Call Type` string,
  `Call Type Group` string,
  `City` string,
  `Dispatch DtTm` string,
  `Entry DtTm` string,
  `Final Priority` long,
  `Fire Prevention District` string,
  `Hospital DtTm` string,
  `Incident Number` long,
  `Location` string,
  `Neighborhooods - Analysis Boundaries` string,
  `Number of Alarms` long,
  `On Scene DtTm` string,
  `Original Priority` string,
  `Priority` string,
  `Received DtTm` string,
  `Response DtTm` string,
  `RowID` string,
  `Station Area` string,
  `Supervisor District` string,
  `Transport DtTm` string,
  `Unit ID` string,
  `Unit Type` string,
  `Unit sequence in call dispatch` long,
  `Watch Date` string,
  `Zipcode of Incident` long
)
USING JSON
OPTIONS (
    path "/mnt/davis/fire-calls/fire-calls-truncated.json"
);
```

```
DESCRIBE fireCallsJSON
```

	col_name	data_type	comment
1	ALS Unit	boolean	null
2	Address	string	null
3	Available DtTm	string	null
4	Battalion	string	null
5	Box	string	null
6	Call Date	string	null
7	Call Final Disposition	string	null

Showing all 34 rows.

Command took 0.21 seconds -- by SWCPROPERTY@GMAIL.COM at 6/29/2021, 10:23:09 PM on My Cluster

Cmd 11

Take a look at the table to make sure it looks correct.

Cmd 12

```
SELECT * FROM fireCallsJSON LIMIT 10
```

▸ (1) Spark Jobs

	ALS Unit	Address	Available DtTm	Battalion	Box	Call Date	Call Final Disposition	Call Number	Call Type	Call Type Group	City
1	false	4TH ST/CHANNEL ST	04/12/2000 09:45:28 PM	B03	2226	04/12/2000	Other	1030118	Medical Incident	null	SF
2	false	1800 Block of IRVING ST	04/12/2000 09:49:52 PM	B08	7424	04/12/2000	Other	1030122	Medical Incident	null	SF
3	false	0 Block of SOUTH VAN NESS AVE	04/12/2000 11:42:43 PM	B02	5117	04/12/2000	Other	1030154	Medical Incident	null	SF
4	true	CLAYTON ST/PARNASSUS AV	04/13/2000 12:33:18 AM	B05	5151	04/13/2000	Other	1040007	Structure Fire	null	SF
5	true	500 Block of 38TH AVE	04/13/2000 02:40:25 AM	B07	7255	04/13/2000	Other	1040021	Medical Incident	null	SF
6	true	200 Block of MADRID ST	04/13/2000 09:26:54 AM	B09	0613	04/13/2000	Other	1040061	Medical Incident	null	SF
7	false	2800 Block of BROADWAY	04/13/2000 09:39:36 AM	B04	4226	04/13/2000	Other	1040079	Alarms	null	SF

Showing all 10 rows.

Command took 0.50 seconds -- by SWCPROPERTY@GMAIL.COM at 6/29/2021, 10:23:16 PM on My Cluster

Now let's populate `newTable` with some of the rows from the `fireCallsJSON` table you just loaded. We only want to include fire calls whose `Final Priority` is `3`.

Cmd 14

```
Insert into newTable
Select `Address`, `City`, `Battalion`, `Box`
From fireCallsJSON
Where `Final Priority` = 3;

SELECT * FROM newTable;
```

▸ (2) Spark Jobs

	Address	City	Battalion	Box
1	2500 Block of BROADWAY	SF	B04	4122
2	100 Block of BERRY ST	SF	B03	2171
3	100 Block of APPLETON AVE	SF	B06	5646
4	400 Block of JONES ST	SF	B03	1461
5	300 Block of GRANT AVE	SF	B01	1315
6	6TH ST/MARKET ST	SF	B03	2248
7	800 Block of 3RD ST	SF	B03	2150

Truncated results, showing first 1000 rows.

Command took 14.28 seconds -- by SWCPROPERTY@GMAIL.COM at 6/29/2021, 10:31:57 PM on My Cluster

Cmd 15

Question 2

How many rows are in `newTable`?

Cmd 16

```
select count(*) from newTable
```

▸ (2) Spark Jobs

	count(1)
1	191039

Sort the rows of `newTable` by ascending `Battalion`.

Question 3

What is the "Battalion" of the first entry in the sorted table?

Cmd 18

```
1  Select *
2  From newTable
3  Order by Battalion
4  limit 5
```

▸ (1) Spark Jobs

	Address	City	Battalion	Box
1	300 Block of GEARY ST	SF	B01	1411
2	500 Block of GEARY ST	SF	B01	1462
3	900 Block of NORTH POINT ST	SF	B01	1623
4	300 Block of COLUMBUS AVE	SF	B01	1311
5	1300 Block of SACRAMENTO ST	SF	B01	1465

Showing all 5 rows.

Command took 1.66 seconds -- by SWCPROPERTY@GMAIL.COM at 6/29/2021, 10:34:45 PM on My Cluster

Cmd 19

Let's see how this table is stored in our file system.

Note: You should have specified the location of the table to be `/tmp/newTableLoc` when you created it.

Cmd 20

```
1  %fs ls dbfs:/tmp/newTableLoc
```

	path	name	size
1	dbfs:/tmp/newTableLoc/_SUCCESS	_SUCCESS	0
2	dbfs:/tmp/newTableLoc/_committed_1444447007168180236	_committed_1444447007168180236	824
3	dbfs:/tmp/newTableLoc/_started_1444447007168180236	_started_1444447007168180236	0
4	dbfs:/tmp/newTableLoc/part-00000-tid-1444447007168180236-a31e9805-4505-4a4e-ab2e-4e5d618ff9b0-479-1-c000.snappy.parquet	part-00000-tid-1444447007168180236-a31e9805-4505-4a4e-ab2e-4e5d618ff9b0-479-1-c000.snappy.parquet	186551
5	dbfs:/tmp/newTableLoc/part-00001-tid-1444447007168180236-a31e9805-4505-4a4e-ab2e-4e5d618ff9b0-480-1-c000.snappy.parquet	part-00001-tid-1444447007168180236-a31e9805-4505-4a4e-ab2e-4e5d618ff9b0-480-1-c000.snappy.parquet	189917
6	dbfs:/tmp/newTableLoc/part-00002-tid-1444447007168180236-a31e9805-4505-4a4e-ab2e-4e5d618ff9b0-481-1-c000.snappy.parquet	part-00002-tid-1444447007168180236-a31e9805-4505-4a4e-ab2e-4e5d618ff9b0-481-1-c000.snappy.parquet	188393
	dbfs:/tmp/newTableLoc/part-00003-tid-1444447007168180236-a31e9805-4505-4a4e-ab2e-4e5d618ff9b0-482-1-	part-00003-tid-1444447007168180236-a31e9805-4505-4a4e-ab2e-4e5d618ff9b0-482-1-c000.snappy.parquet	181212

Showing all 11 rows.

Command took 1.03 seconds -- by SWCPROPERTY@GMAIL.COM at 6/29/2021, 10:49:12 PM on My Cluster

Cmd 21

First run the following cell to check how many partitions are in this table. Did the number of partitions match the number of files our data was stored as?

Cmd 22

Let's try increasing the number of partitions to 256. Create this as a new table and call it `newTablePartitioned`

Cmd 23

```
Create or Replace Temporary View newTablePartitioned
 AS
 SELECT /*+ REPARTITION(256) */ *
 FROM newTable
```

OK

Command took 0.09 seconds -- by SWCPROPERTY@GMAIL.COM at 6/29/2021, 10:49:20 PM on My Cluster

Cmd 24

Now let's take a look at how this new table is stored.

Cmd 25

```
DESCRIBE EXTENDED newTablePartitioned
```

	col_name	data_type	comment
1	Address	string	null
2	City	string	null
3	Battalion	string	null
4	Box	string	null

Showing all 4 rows.

Command took 0.05 seconds -- by SWCPROPERTY@GMAIL.COM at 6/29/2021, 10:49:25 PM on My Cluster

Cmd 26

Copy the location of the `newTablePartitioned` from the table above and take a look at the files stored at that location. Now how many parts is our data stored?

Cmd 27

```
%fs ls dbfs:/user/hive/warehouse/databricks.db/newtablepartitioned
```

⊞FileNotFoundException: /user/hive/warehouse/databricks.db/newtablepartitioned

Command took 0.46 seconds -- by SWCPROPERTY@GMAIL.COM at 6/29/2021, 10:49:36 PM on My Cluster

Cmd 28

Now sort the rows of `newTablePartitioned` by ascending `Battalion` and compare how long this query takes.

Question 4

Was this query faster or slower on the table with increased partitions?

Cmd 29

```
SELECT * FROM newTablePartitioned ORDER BY `Battalion`
```

▸ (2) Spark Jobs

	Address	City	Battalion	Box
1	1100 Block of CALIFORNIA ST	SF	B01	1445
2	700 Block of GREEN ST	San Francisco	B01	1352
3	400 Block of GREENWICH ST	SF	B01	1335
4	300 Block of CLAY ST	SF	B01	1161

5	KEARNY ST/SACRAMENTO ST	SF	B01	1244
6	POST ST/TAYLOR ST	SF	B01	1451
7	2200 Block of STOCKTON ST	SF	B01	1341

Truncated results, showing first 1000 rows.

Command took 4.43 seconds -- by SWCPROPERTY@GMAIL.COM at 6/29/2021, 10:49:49 PM on My Cluster

Cmd 30

Run the following cell to see where the data of the original `newTable` is stored.

Cmd 31

```
%fs ls dbfs:/tmp/newTableLoc
```

	path	name	size
1	dbfs:/tmp/newTableLoc/_SUCCESS	_SUCCESS	0
2	dbfs:/tmp/newTableLoc/_committed_1444447007168180236	_committed_1444447007168180236	824
3	dbfs:/tmp/newTableLoc/_started_1444447007168180236	_started_1444447007168180236	0
4	dbfs:/tmp/newTableLoc/part-00000-tid-1444447007168180236-a31e9805-4505-4a4e-ab2e-4e5d618ff9b0-479-1-c000.snappy.parquet	part-00000-tid-1444447007168180236-a31e9805-4505-4a4e-ab2e-4e5d618ff9b0-479-1-c000.snappy.parquet	186551
5	dbfs:/tmp/newTableLoc/part-00001-tid-1444447007168180236-a31e9805-4505-4a4e-ab2e-4e5d618ff9b0-480-1-c000.snappy.parquet	part-00001-tid-1444447007168180236-a31e9805-4505-4a4e-ab2e-4e5d618ff9b0-480-1-c000.snappy.parquet	189917
6	dbfs:/tmp/newTableLoc/part-00002-tid-1444447007168180236-a31e9805-4505-4a4e-ab2e-4e5d618ff9b0-481-1-c000.snappy.parquet	part-00002-tid-1444447007168180236-a31e9805-4505-4a4e-ab2e-4e5d618ff9b0-481-1-c000.snappy.parquet	188393
	dbfs:/tmp/newTableLoc/part-00003-tid-1444447007168180236-a31e9805-4505-4a4e-ab2e-4e5d618ff9b0-482-1-	part-00003-tid-1444447007168180236-a31e9805-4505-4a4e-ab2e-4e5d618ff9b0-482-1-c000.snappy.parquet	181212

Showing all 11 rows.

Command took 1.10 seconds -- by SWCPROPERTY@GMAIL.COM at 6/29/2021, 10:50:03 PM on My Cluster

Cmd 32

Now drop the table `newTable`.

```
DROP TABLE newTable;

-- The following line should error!
-- SELECT * FROM newTable;
```

OK

Command took 0.32 seconds -- by SWCPROPERTY@GMAIL.COM at 6/29/2021, 10:50:20 PM on My Cluster

Cmd 34

Question 5

Does the data stored within the table still exist at the original location (`dbfs:/tmp/newTableLoc`) after you dropped the table? (Answer "yes" or "no")

Cmd 35

```
%fs ls dbfs:/tmp/newTableLoc

```

	path	name	size
1	dbfs:/tmp/newTableLoc/_SUCCESS	_SUCCESS	0
2	dbfs:/tmp/newTableLoc/_committed_1444447007168180236	_committed_1444447007168180236	824
3	dbfs:/tmp/newTableLoc/_started_1444447007168180236	_started_1444447007168180236	0
4	dbfs:/tmp/newTableLoc/part-00000-tid-1444447007168180236-a31e9805-4505-4a4e-ab2e-4e5d618ff9b0-479-1-c000.snappy.parquet	part-00000-tid-1444447007168180236-a31e9805-4505-4a4e-ab2e-4e5d618ff9b0-479-1-c000.snappy.parquet	186551
5	dbfs:/tmp/newTableLoc/part-00001-tid-1444447007168180236-a31e9805-4505-4a4e-ab2e-4e5d618ff9b0-480-1-c000.snappy.parquet	part-00001-tid-1444447007168180236-a31e9805-4505-4a4e-ab2e-4e5d618ff9b0-480-1-c000.snappy.parquet	189917
6	dbfs:/tmp/newTableLoc/part-00002-tid-1444447007168180236-a31e9805-4505-4a4e-ab2e-4e5d618ff9b0-481-1-c000.snappy.parquet	part-00002-tid-1444447007168180236-a31e9805-4505-4a4e-ab2e-4e5d618ff9b0-481-1-c000.snappy.parquet	188393
	dbfs:/tmp/newTableLoc/part-00003-tid-1444447007168180236-a31e9805-4505-4a4e-ab2e-4e5d618ff9b0-482-1-	part-00003-tid-1444447007168180236-a31e9805-4505-4a4e-ab2e-4e5d618ff9b0-482-1-c000.snappy.parquet	181212

Showing all 11 rows.

Command took 0.40 seconds -- by SWCPROPERTY@GMAIL.COM at 6/29/2021, 10:50:27 PM on My Cluster

Cmd 36