Connecting to JDBC

Apache Spark™ and Databricks® allow you to connect to a number of data stores using JDBC.

In this lesson you:

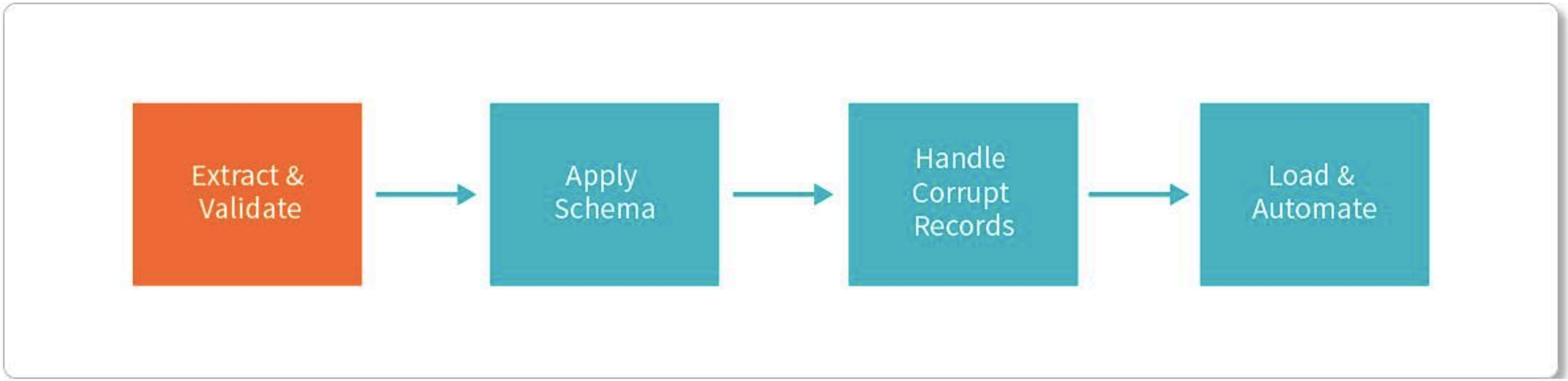
- Read data from a JDBC connection
- Parallelize your read operation to leverage distributed computation

Java Database Connectivity

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 pushdown. **Predicate pushdown 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 pushdown communicated through the JDBC API, making JDBC an ideal data source for Spark workloads.

In the road map for ETL, this is the **Extract and Validate** step:



Recalling the Design Pattern

Recall the design pattern for connecting to data from the previous lesson:

1. Define the connection point.
2. Define connection parameters such as access credentials.
3. Add necessary options.

After adhering to this, read data using `spark.read.options(<option key>, <option value>).<connection_type>(<endpoint>)`. The JDBC connection uses this same formula with added complexity over what was covered in the lesson.

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

Each notebook has a default language that appears in upper corner of the screen next to the notebook name, and you can easily switch between languages in a notebook. To change languages, start your cell with `%python`, `%scala`, `%sql`, or `%r`.

Cmd 10

```
%scala
// run this regardless of language type
Class.forName("org.postgresql.Driver")
```

```
res2: Class[_] = class org.postgresql.Driver
```

Command took 0.28 seconds -- by huseyinyilmaz01@gmail.com at 4/27/2020, 1:42:45 AM on My Cluster

Cmd 11

Define your database connection criteria. In this case, you need the hostname, port, and database name.

Access the database `training` via port `5432` of a Postgres server sitting at the endpoint `server1.databricks.training`.

Combine the connection criteria into a URL.

Cmd 12

```
jdbcHostname = "server1.databricks.training"
jdbcPort = 5432
jdbcDatabase = "training"

jdbcUrl = f"jdbc:postgresql://{jdbcHostname}:{jdbcPort}/{jdbcDatabase}"
```

Command took 0.04 seconds -- by huseyinyilmaz01@gmail.com at 4/27/2020, 1:44:36 AM on My Cluster

Create a connection properties object with the username and password for the database.

Cmd 14

```
connectionProps = {
  "user": "readonly",
  "password": "readonly"
}
```

Command took 0.03 seconds -- by huseyinyilmaz01@gmail.com at 4/27/2020, 1:44:56 AM on My Cluster

Cmd 15

Read from the database by passing the URL, table name, and connection properties into `spark.read.jdbc()`.

Cmd 16

```
tableName = "training.people_1m"

peopleDF = spark.read.jdbc(url=jdbcUrl, table=tableName, properties=connectionProps)

display(peopleDF)
```

(1) Spark Jobs

peopleDF: pyspark.sql.dataframe.DataFrame = [id: integer, firstName: string ... 6 more fields]

id	firstName	middleName	lastName	gender	birthDate	ssn	salary
766094	Margarito	Vincent	Scawen	M	1997-03-28T00:00:00.000+0000	930-33-6642	58134
766095	Sherman	Mitch	Darnell	M	1962-02-11T00:00:00.000+0000	916-54-3972	78157
766096	Ernest	Jorge	Tetlow	M	1971-08-17T00:00:00.000+0000	930-24-3622	73869
766097	Sonny	Harold	Moakson	M	1992-06-04T00:00:00.000+0000	974-45-5360	41297
766098	Bradly	Hai	Heustice	M	1996-09-23T00:00:00.000+0000	901-30-4829	73274
766099	Genaro	Alfonso	Acock	M	1967-12-08T00:00:00.000+0000	902-73-5943	54489

Exercise 1: Parallelizing JDBC Connections

The command above was executed as a serial read through a single connection to the database. This works well for small data sets; at scale, parallel reads are necessary for optimal performance.

See the Managing Parallelism section of the Databricks documentation.

Cmd 18

Step 1: Find the Range of Values in the Data

Parallel JDBC reads entail assigning a range of values for a given partition to read from. The first step of this divide-and-conquer approach is to find bounds of the data.

Calculate the range of values in the `id` column of `peopleDF`. Save the minimum to `dfMin` and the maximum to `dfMax`. **This should be the number itself rather than a DataFrame that contains the number.** Use `.first()` to get a Scala or Python object.

Hint: See the `min()` and `max()` functions in Python `pyspark.sql.functions` or Scala `org.apache.spark.sql.functions`.

Cmd 19

```python
# TODO
from pyspark.sql.functions import min, max

dfMin = peopleDF.select(min("id")).first()[0]
dfMax = peopleDF.select(max("id")).first()[0]

print("DataFrame minimum: {}\nDataFrame maximum: {}".format(dfMin, dfMax))
```

(2) Spark Jobs

```
DataFrame minimum: 1
DataFrame maximum: 1000000
```

Command took 1.42 seconds -- by buscwjwilson010gmail.com at 4/27/2020, 2:07:55 AM on My Cluster

Step 2: Define the Connection Parameters.

Referencing the documentation, define the connection parameters for this read.

Use 8 partitions.

Assign the results to `peopleDFParallel`.

Setting the column for your parallel read introduces unexpected behavior due to a bug in Spark. To make sure Spark uses the capitalization of your column, use `'"id"'` for your column. Monitor the issue here.

Cmd 22

```
# TODO
peopleDFParallel = spark.read.jdbc(
  url=jdbcUrl,                    # the JDBC URL
  table="training.people_1m",     # the name of the table
  column="id",                    # the name of a column of an integral type that will be used for partitioning.
  lowerBound=dfMin,               # the minimum value of columnName used to decide partition stride.
  upperBound=dfMax,               # the maximum value of columnName used to decide partition stride
  numPartitions=8,               # the number of partitions/connections
  properties=connectionProps      # the connection properties
)

display(peopleDFParallel)
```

(1) Spark Jobs

peopleDFParallel: pyspark.sql.dataframe.DataFrame = [id: integer, firstName: string ... 6 more fields]

id	firstName	middleName	lastName	gender	birthDate	ssn	salary
1	Lydia	Ula	Rubinowicz	F	1997-02-02T00:00:00.000+0000	927-54-8759	70110
2	Diamond	Carletta	Melesk	F	1984-10-21T00:00:00.000+0000	939-18-5247	74024
3	Yen	Julienne	Recher	F	1988-11-24T00:00:00.000+0000	929-26-8667	83619

Step 3: Compare the Serial and Parallel Reads

Compare the two reads with the `%timeit` function.

Cmd 25

Display the number of partitions in each DataFrame by running the following:

Cmd 26

```
print("Partitions:", peopleDF.rdd.getNumPartitions())
print("Partitions:", peopleDFParallel.rdd.getNumPartitions())
```

```
Partitions: 1
Partitions: 8
```

Command took 0.06 seconds -- by huseyinyilmaz01@gmail.com at 4/27/2020, 2:11:29 AM on My Cluster

Cmd 27

Invoke `%timeit` followed by calling a `.describe()`, which computes summary statistics, on both `peopleDF` and `peopleDFParallel`.

Cmd 28

```
%timeit peopleDF.describe()
%timeit peopleDFParallel.describe()
```

Cancel Running command...

▸ (11) Spark Jobs

```
19.9 s ± 591 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
```

Cmd 29

What is the difference between serial and parallel reads? Note that your results vary drastically depending on the cluster and number of partitions you use

Review

Question: What is JDBC?

Answer: JDBC stands for Java Database Connectivity, and is a Java API for connecting to databases such as MySQL, Hive, and other data stores.

Question: How does Spark read from a JDBC connection by default?

Answer: With a serial read. With additional specifications, Spark conducts a faster, parallel read. Parallel reads take full advantage of Spark's distributed architecture.

Question: What is the general design pattern for connecting to your data?

Answer: The general design patter is as follows:

1. Define the connection point
2. Define connection parameters such as access credentials
3. Add necessary options such as for headers or parallelization