Joins

In yesterday class we covered aggregating single datasets, which is helpful,but more often than not, your Spark applications are going to bring together a large number of different datasets. For this reason, joins are an essential part of nearly all Spark workloads. Spark's ability to talk to different data means that you gain the ability to tap into a variety of data sources across your company.

Join Expressions

Join brings together two sets of data, the left and the right, by comparing the value of one or more keys of the left and right and evaluating the result of a join expression that determines whether Spark should bring together the left set of data with the right set of data.

The most common join expression, an equi-join, compares whether the specified keys in your left and right datasets are equal. If they are equal, Spark will combine the left and right datasets. The opposite is true for keys that do not match; Spark discards the rows that do not have matching keys.

Spark also allows for much more sophsticated join policies in addition to equi-joins. We can even use complex types and perform something like checking whether a key exists within an array when you perform a join.

Join Types

We will use the following data to demonstrate Joins:

```
// in Scala
val person = Seq((0, "Bill Chambers", 0, Seq(100)),
                 (1, "Matei Zaharia", 1, Seq(500, 250, 100)),
                 (2, "Michael Armbrust", 1, Seq(250, 100))).toDF("id", "name",
"graduate_program", "spark_status")

val graduateProgram = Seq(
                    (0, "Masters", "School of Information", "UC Berkeley"),  (2,
"Masters", "EECS", "UC Berkeley"),
                    (1, "Ph.D.", "EECS", "UC Berkeley"))
                    .toDF("id", "degree", "department", "school")

val sparkStatus = Seq((500, "Vice President"),    (250, "PMC Member"),
(100, "Contributor"))  .toDF("id", "status")
```

Lets register the tables:

```
person.createOrReplaceTempView("person")
graduateProgram.createOrReplaceTempView("graduateProgram")
sparkStatus.createOrReplaceTempView("sparkStatus")
```

Inner Joins

Inner joins evaluate the keys in both of the DataFrames or tables and include (and join together) only the rows that evaluate to true. In the following example, we join the graduateProgram DataFrame with the person DataFrame to create a new DataFrame:

// in Scala

```
// lets create a join expression
val joinExpression = person.col("graduate_program") === graduateProgram.col("id")
```

Now we will perform the Join, inner join is the default Join:

```
person.join(graduateProgram, joinExpression).show()
```

This is how it looks in sql

```
-- in SQL
SELECT * FROM person JOIN graduateProgram  ON person.graduate_program =
graduateProgram.id
```

Make sure you understand what inner join is doing.If you do not let me know.

Outer Joins

Outer joins evaluate the keys in both of the DataFrames or tables and includes (and joins together) the rows that evaluate to true or false. If there is no equivalent row in either the left or right DataFrame, Spark will insert null:

```
val joinType = "outer"

person.join(graduateProgram, joinExpression, joinType).show()
```

This is how you do it in sql

```
-- in SQL

SELECT * FROM person FULL OUTER JOIN graduateProgram  ON graduate_program =
graduateProgram.id
```

Do you understand what outer join is doing? Why are you getting nulls?

Left Outer Joins

Left outer joins evaluate the keys in both of the DataFrames or tables and includes all rows from the left DataFrame as well as any rows in the right DataFrame that have a match in the left DataFrame. If there is no equivalent row in the right DataFrame, Spark will insert null:

```
val joinType = "left_outer"

graduateProgram.join(person, joinExpression, joinType).show()
```

```
-- in SQL

SELECT * FROM graduateProgram LEFT OUTER JOIN person
  ON person.graduate_program = graduateProgram.id
```

Right Outer Joins

Right outer joins evaluate the keys in both of the DataFrames or tables and includes all rows from the right DataFrame as well as any rows in the left DataFrame that have a match in the right DataFrame. If there is no equivalent row in the left DataFrame, Spark will insert null:

```
val joinType = "right_outer"


person.join(graduateProgram, joinExpression, joinType).show()
```

```
-- in SQL
SELECT * FROM person RIGHT OUTER JOIN graduateProgram  ON person.graduate_program =
graduateProgram.id
```

Cross (Cartesian) Joins(This can crash your spark application!!)

The last of our joins are cross-joins or cartesian products. Cross-joins in simplest terms are inner joins that do not specify a predicate. Cross joins will join every single row in the left DataFrame to ever single row in the right DataFrame. This will cause an absolute explosion in the number of rows contained in the resulting DataFrame. If you have 1,000 rows in each DataFrame, the cross-join of these will result in 1,000,000 (1,000 x 1,000) rows. For this reason, you must very explicitly state that you want a cross-join by using the cross join keyword:

```
val joinType = "cross"
graduateProgram.join(person, joinExpression, joinType).show()
```

```
-- in SQL

SELECT * FROM graduateProgram CROSS JOIN person  ON graduateProgram.id =
person.graduate_program
```