Pyspark Joins by Example

This entry was posted in Python Spark *on January 27, 2018 by Will*

Summary: Pyspark DataFrames have a join method which takes three parameters: DataFrame on the right side of the join, Which fields are being joined on, and what type of join (inner, outer, left_outer, right_outer, leftsemi). You call the join method from the left side DataFrame object such as `df1.join(df2, df1.col1 == df2.col1, 'inner')`.

One of the challenges of working with Pyspark (the python shell of Apache Spark) is that it's Python and Pandas but with some subtle differences. For example, you can't just `dataframe.column.lower()` to create a lowercase version of a string column, instead you use a function `lower(dataframe.column)` and that just doesn't feel right. So I often have to reference the documentation just to get my head straight.

This pyspark tutorial is my attempt at cementing how joins work in Pyspark once and for all. I'll be using the example data from Coding Horror's explanation of SQL joins. For the official documentation, see here. Let's get started!

Setting up the Data in Pyspark

```
valuesA = [('Pirate',1),('Monkey',2),('Ninja',3),('Spaghetti',4)]
TableA = spark.createDataFrame(valuesA,['name','id'])

valuesB = [('Rutabaga',1),('Pirate',2),('Ninja',3),('Darth Vader',4)]
TableB = spark.createDataFrame(valuesB,['name','id'])

TableA.show()
TableB.show()
```

Table A

```
+---------+---+
|     name| id|
+---------+---+
|   Pirate|  1|
|   Monkey|  2|
|    Ninja|  3|
|Spaghetti|  4|
+---------+---+
```

Table B

```
+-----------+---+
|       name| id|
+-----------+---+
|   Rutabaga|  1|
|     Pirate|  2|
|      Ninja|  3|
|Darth Vader|  4|
+-----------+---+
```

In order to create a DataFrame in Pyspark, you can use a list of structured tuples. In this case, we create TableA with a 'name' and 'id' column. The `spark.createDataFrame` takes two parameters: a list of tuples and a list of column names.

The `DataFrameObject.show()` command displays the contents of the DataFrame. The image above has been altered to put the two tables side by side and display a title above the tables.

The last piece we need to perform is to create an alias for these tables. The alias, like in SQL, allows you to distinguish where each column is coming from. The columns are named the same so how can you know if 'name' is referencing TableA or TableB? The alias provides a short name for referencing fields and for referencing the fields after creation of the joined table.

```
ta = TableA.alias('ta')
tb = TableB.alias('tb')
```

Now we can use refer to the DataFrames as `ta.name` or `tb.name`. Let's move on to the actual joins!

Pyspark Inner Join Example

```
inner_join = ta.join(tb, ta.name == tb.name)
inner_join.show()
```

```
+------+---+------+---+
|  name| id|  name| id|
+------+---+------+---+
| Ninja|  3| Ninja|  3|
|Pirate|  1|Pirate|  2|
+------+---+------+---+
```

An inner join is the default join type used. The fully qualified code might look like `ta.join(tb, ta.name == tb.name, 'inner')`. Ultimately, this translates to the following SQL statement:

```
SELECT ta.*, tb.*
FROM ta
INNER JOIN tb
ON ta.name = tb.name
```

Now if you want to reference those columns in a later step, you'll have to use the `col` function and include the alias. For example `inner_join.filter(col('ta.id' > 2))` to filter the TableA ID column to any row that is greater than two.

Pyspark Left Join Example

```
left_join = ta.join(tb, ta.name == tb.name,how='left') # Could also use 'left_outer'
left_join.show()
```

```
+---------+---+------+----+
|     name| id|  name|  id|
+---------+---+------+----+
|Spaghetti|  4|  null|null|
|    Ninja|  3| Ninja|   3|
|   Pirate|  1|Pirate|   2|
|   Monkey|  2|  null|null|
+---------+---+------+----+
```

Notice that Table A is the left hand-side of the query. You are calling join on the ta DataFrame. So it's just like in SQL where the FROM table is the left-hand side in the join. You can also think of it as you're reading from left to right so TableA is the left-most table being referenced.

You can use 'left' or 'left_outer' and the results are exactly the same. It seems like this is a convenience for people coming from different SQL flavor backgrounds.

Notice how the results now include 'null' values. In the example below, you can use those nulls to filter for these values.

Pyspark Left Join and Filter Example

```
left_join = ta.join(tb, ta.name == tb.name,how='left') # Could also use 'left_outer'
left_join.filter(col('tb.name').isNull()).show()
```

```
+---------+---+----+----+
|     name| id|name|  id|
+---------+---+----+----+
|Spaghetti|  4|null|null|
|   Monkey|  2|null|null|
+---------+---+----+----+
```

Using the `isNull` or `isNotNull` methods, you can filter a column with respect to the null values inside of it. As in SQL, this is very handy if you want to get the records found in the left side but not found in the right side of a join.

Pyspark Right Join Example

```
right_join = ta.join(tb, ta.name == tb.name,how='right') # Could also use 'right_outer'
right_join.show()
```

```
+------+----+-----------+---+
|  name|  id|       name| id|
+------+----+-----------+---+
|  null|null|   Rutabaga|  1|
| Ninja|   3|      Ninja|  3|
|Pirate|   1|     Pirate|  2|
|  null|null|Darth Vader|  4|
+------+----+-----------+---+
```

Again, the code is read from left to right so table A is the left side and table B is the right side. If you want to select all records from table B and return data from table A when it matches, you choose 'right' or 'right_outer' in the last parameter. As in the example above, you could combine this with the isNull to identify records found in the right table but not found in the left table.

Pyspark Full Outer Join Example

```
full_outer_join = ta.join(tb, ta.name == tb.name,how='full') # Could also use
'full_outer'
full_outer_join.show()
```

```
+---------+----+-----------+----+
|     name|  id|       name|  id|
+---------+----+-----------+----+
|     null|null|   Rutabaga|   1|
|Spaghetti|   4|       null|null|
|    Ninja|   3|      Ninja|   3|
|   Pirate|   1|     Pirate|   2|
|   Monkey|   2|       null|null|
|     null|null|Darth Vader|   4|
+---------+----+-----------+----+
```

Finally, we get to the full outer join. This shows all records from the left table and all the records from the right table and nulls where the two do not match.

PysPark SQL Joins Gotchas and Misc

If you're paying attention, you'll notice a couple issues that makes using Pyspark SQL joins a little annoying when coming from a SQL background.

1. Alias References: If you do not apply an alias to the dataframe, you'll receive an error *after* you create your joined dataframe. With two columns named the same thing, referencing one of the duplicate named columns returns an error that essentially says it doesn't know which one you selected. In SQL Server and other languages, the SQL engine wouldn't let that query go through or it would automatically append a prefix or suffix to that field name.

2. Cross Join: You can also perform a cartesian product using the crossjoin method. This is useful if you're looking to repeat every row in table A for every row in table B.

3. The 'leftsemi' option: I didn't cover this option above (since Jeff Atwood didn't either). but if you care only for the left columns and just want to pull in the records that match in both table A and table B, you can choose 'leftsemi'. This StackOverflow post gives a good explanation.

4. Joining on Multiple Columns: In the second parameter, you use the `&` (ampersand) symbol for and and the `|` (pipe) symbol for or between columns.

And that's it! I hope you learned something about Pyspark joins! If you feel like going old school, check out my post on Pyspark RDD Examples. But DataFrames are the wave of the future in the Spark world so keep pushing your Pyspark SQL knowledge!

Post navigation

← Why Saying a 'One Unit Increase' Doesn't Work in Logistic Regression