Master PySpark: From Zero to Big Data Hero!!

Joins in Dataframe – Part 1

Joins in PySpark

Joins are used to combine two DataFrames based on a common column or condition. PySpark supports several types of joins, similar to SQL. Below are explanations and examples for each type of join.

---

1. Inner Join

Code:

```
inner_join = df1.join(df2, on="common_column", how="inner")
```

Explanation:

- **Purpose**: Returns rows where there is a match in both DataFrames (df1 and df2) based on the common_column.
- **Behavior**: Rows with no matching value in either DataFrame are excluded.
- **Use Case**: When you only need records that exist in both DataFrames.

---

2. Left Join (Left Outer Join)

Code:

```
left_join = df1.join(df2, on="common_column", how="left")
```

Explanation:

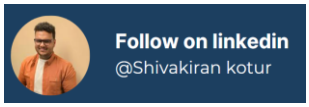
- **Purpose**: Returns all rows from df1 and the matching rows from df2. If no match exists in df2, the result will contain NULL for columns from df2.
- **Behavior**: All rows from the left DataFrame (df1) are preserved, even if there's no match in the right DataFrame (df2).
- **Use Case**: When you want to retain all rows from df1, even if there's no match in df2.

---

3. Right Join (Right Outer Join)

Code:

```
right_join = df1.join(df2, on="common_column", how="right")
```

Explanation:

- **Purpose**: Returns all rows from df2 and the matching rows from df1. If no match exists in df1, the result will contain NULL for columns from df1.
- **Behavior**: All rows from the right DataFrame (df2) are preserved, even if there's no match in the left DataFrame (df1).
- **Use Case**: When you want to retain all rows from df2, even if there's no match in df1.

---

4. Full Join (Outer Join)

Code:

```
full_join = df1.join(df2, on="common_column", how="outer")
```

Explanation:

- **Purpose**: Returns all rows when there is a match in either df1 or df2. Non-matching rows will have NULL values in the columns from the other DataFrame.
- **Behavior**: Retains all rows from both DataFrames, filling in NULL where there is no match.
- **Use Case**: When you want to retain all rows from both DataFrames, regardless of whether there's a match.

---

5. Left Semi Join

Code:

```
left_semi_join = df1.join(df2, on="common_column", how="left_semi")
```

Explanation:

- **Purpose**: Returns only the rows from df1 where there is a match in df2. It behaves like an inner join but only keeps columns from df1.
- **Behavior**: Filters df1 to only keep rows that have a match in df2.
- **Use Case**: When you want to filter df1 to keep rows with matching keys in df2, but you don't need columns from df2.

---

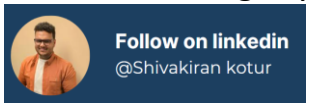
6. Left Anti Join

Code:

```
left_anti_join = df1.join(df2, on="common_column", how="left_anti")
```

Explanation:

- **Purpose**: Returns only the rows from df1 that do **not** have a match in df2.
- **Behavior**: Filters out rows from df1 that have a match in df2.
- **Use Case**: When you want to filter df1 to keep rows with no matching keys in df2.



Follow me on LinkedIn – Shivakiran kotur

7. Cross Join

Code:

```
cross_join = df1.crossJoin(df2)
```

Explanation:

- **Purpose**: Returns the Cartesian product of df1 and df2, meaning every row of df1 is paired with every row of df2.
- **Behavior**: The number of rows in the result will be the product of the row count of df1 and df2.
- **Use Case**: Typically used in edge cases or for generating combinations of rows, but be cautious as it can result in a very large DataFrame.

---

8. Join with Explicit Conditions

Code:

```
inner_join = df1.join(df2, (df1["columnA"] == df2["columnB"]), "inner")
```

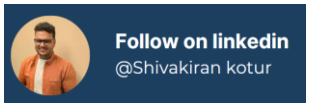
Explanation:

- **Purpose**: This is an example of an inner join where the common columns have different names in df1 and df2.
- **Behavior**: Joins df1 and df2 based on a condition where columnA from df1 matches columnB from df2.
- **Use Case**: When the join condition involves columns with different names or more complex conditions.

Conclusion:

- **Inner Join**: Matches rows from both DataFrames.
- **Left/Right Join**: Keeps all rows from the left or right DataFrame and matches where possible.
- **Full Join**: Keeps all rows from both DataFrames.
- **Left Semi**: Filters df1 to rows that match df2 without including columns from df2.
- **Left Anti**: Filters df1 to rows that do not match df2.
- **Cross Join**: Returns the Cartesian product, combining all rows of both DataFrames.
- **Explicit Condition Join**: Allows complex join conditions, including columns with different names.

These joins are highly useful for various types of data integration and analysis tasks in PySpark.



Follow me on LinkedIn – Shivakiran kotur