Home (/) About (https://sparkbyexamples.com/about-sparkbyexamples/) | { One stop for all Spark Examples }

Spark by {Examples} (https://sparkbyexamples.com/)

Spark ⌄ (https://sparkbyexamples.com/)

PySpark (https://sparkbyexamples.com/pyspark-tutorial/)

Hive (https://sparkbyexamples.com/apache-hive-tutorial/)

HBase (https://sparkbyexamples.com/apache-hbase-tutorial/)

Kafka (https://sparkbyexamples.com/apache-kafka-tutorials-with-examples/)

FAQ's (https://sparkbyexamples.com/spark-questions/)

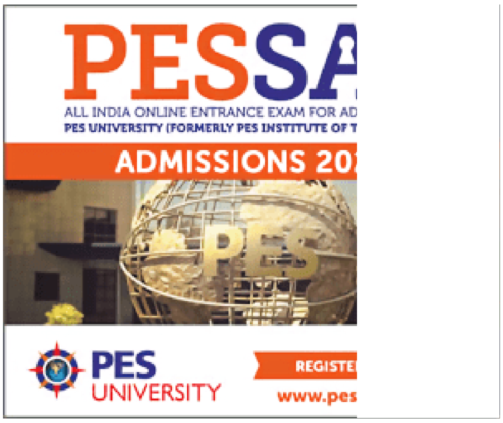
More ⌄ (https://sparkbyexamples.com/)

Spark SQL Join Types with examples

NNK (https://sparkbyexamples.com/author/admin/) -

Apache Spark (https://sparkbyexamples.com/category/spark/)

Spark DataFrame supports all basic SQL Join Types like `INNER`, `LEFT OUTER`, `RIGHT OUTER`, `LEFT ANTI`, `LEFT SEMI`, `CROSS`, `SELF JOIN`. Spark SQL Joins are wider transformations that result in data shuffling over the network hence they have [huge performance issues (https://sparkbyexamples.com/spark/spark-performance-tuning/)](https://sparkbyexamples.com/spark/spark-performance-tuning/) when not designed with care.


PESSA
ALL INDIA ONLINE ENTRANCE EXAM FOR AD
PES UNIVERSITY (FORMERLY PES INSTITUTE OF T
ADMISSIONS 20
PES
PES UNIVERSITY
REGISTE
www.pes

On the other hand Spark SQL Joins comes with more optimization by default (thanks to DataFrames &

Spark RDD – Read CSV (https://sparkbyexamples.com/apache-spark-rdd/spark-load-csv-file-into-rdd/)

Spark RDD – Create RDD (https://sparkbyexamples.com/apache-spark-rdd/different-ways-to-create-spark-rdd/)

Spark RDD – Create Empty RDD (https://sparkbyexamples.com/apache-spark-rdd/spark-how-to-create-an-empty-rdd/)

Spark RDD – Transformations (https://sparkbyexamples.com/apache-spark-rdd/spark-rdd-transformations/)

Spark RDD – Actions (https://sparkbyexamples.com/apache-spark-rdd/spark-rdd-actions/)

Spark RDD – Pair Functions (https://sparkbyexamples.com/apache-spark-rdd/spark-pair-rdd-functions/)

Spark RDD – Repartition and Coalesce (https://sparkbyexamples.com/spark/spark-repartition-vs-coalesce/)

Spark RDD – Shuffle Partitions (https://sparkbyexamples.com/spark/spark-shuffle-partitions/)

Spark RDD – Cache vs Persist (https://sparkbyexamples.com/spark/spark-difference-between-cache-and-persist/)

Spark RDD – Persistance Storage Levels (https://sparkbyexamples.com/spark/spark-persistence-storage-levels/)

Spark RDD – Broadcast Variables (https://sparkbyexamples.com/spark/spark-broadcast-variables/)

Spark RDD – Accumulator Variables (https://sparkbyexamples.com/spark/spark-accumulators/)

Spark RDD – Convert RDD to DataFrame (https://sparkbyexamples.com/

Dataset) however still there would be some performance issues to consider while using.

In this tutorial, you will learn different Join syntaxes and using different Join types on two DataFrames and Datasets using Scala examples. Please access Join on Multiple DataFrames (https://sparkbyexamples.com/spark/spark-join-multiple-dataframes/) in case if you wanted to join more than two DataFrames.

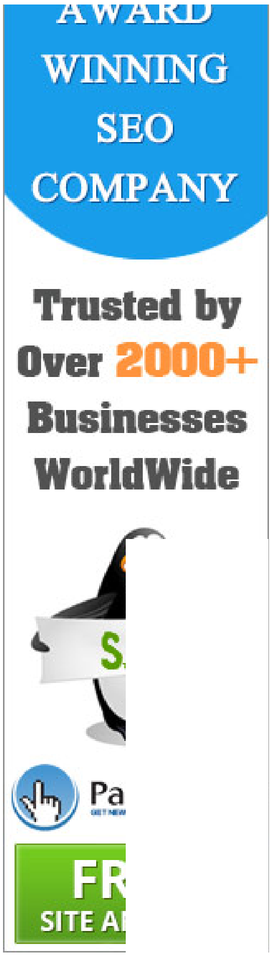
- Join Syntax & Types
- Inner Join
- Full Outer Join
- Left Outer Join
- Right Outer Join
- Left Anti Join
- Left Semi Join
- Self Join
- Using SQL Expression

1. SQL Join Types & Syntax

Below are the list of all Spark SQL Join Types and Syntaxes.

```
1) join(right: Dataset[_]): Dat
2) join(right: Dataset[_], usin
3) join(right: Dataset[_], usin
4) join(right: Dataset[_], usin
5) join(right: Dataset[_], join
6) join(right: Dataset[_], join
```

The rest of the tutorial explains Join Types using syntax 6 which takes arguments right join DataFrame, join expression and type of join in String.



apache-spark-rdd/convert-spark-rdd-to-dataframe-dataset/)

Spark SQL Tutorial

Spark SQL – Create DataFrame (https://sparkbyexamples.com/spark/different-ways-to-create-a-spark-dataframe/)

Spark SQL – Select Columns (https://sparkbyexamples.com/spark/spark-select-columns-from-dataframe/)

Spark SQL – Add and Update Column (withColumn) (https://sparkbyexamples.com/spark/spark-dataframe-withcolumn/)

Spark SQL – Rename Nested Column (https://sparkbyexamples.com/spark/rename-a-column-on-spark-dataframes/)

Spark SQL – Drop column (https://sparkbyexamples.com/spark/spark-drop-column-from-dataframe-dataset/)

Spark SQL – Where | Filter (https://sparkbyexamples.com/spark/spark-dataframe-where-filter/)

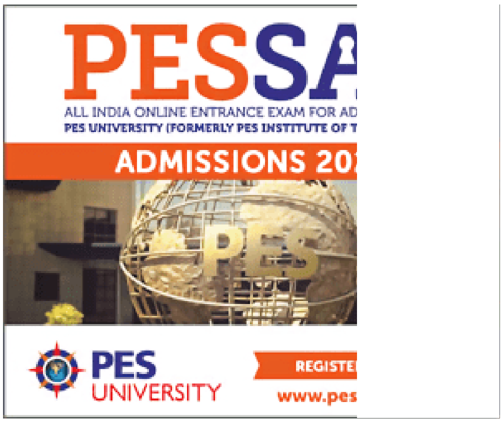
Spark SQL – When Otherwise (https://sparkbyexamples.com/spark/spark-case-when-otherwise-example/)

Spark SQL – Collect data to Driver (https://sparkbyexamples.com/spark/spark-dataframe-collect/)

Spark SQL – Distinct (https://sparkbyexamples.com/spark/spark-remove-duplicate-rows/)

Spark SQL- Pivot Table DataFrame (https://sparkbyexamples.com/spark/how-to-pivot-table-and-unpivot-a-spark-dataframe/)

Spark SQL – Data Types (https://sparkbyexamples.com/spark/spark-sql-dataframe-data-types/)


PESSA
ALL INDIA ONLINE ENTRANCE EXAM FOR AD
PES UNIVERSITY (FORMERLY PES INSTITUTE OF T
ADMISSIONS 20
PES
PES UNIVERSITY
REGISTE
www.pes

For Syntax 4 & 5 you can use either “JoinType” or “Join String” defined on the above table for “joinType” string argument. When you use “JoinType”, you should `import org.apache.spark.sql.catalyst.plans._` as this package defines JoinType objects.

JOINTYPE	JOIN STRING	EQUIVALENT SQL JOIN
Inner.sql	inner	INNER JOIN
FullOuter.sql	outer, full, fullouter, full_outer	FULL OUTER JOIN
LeftOuter.sql	left, leftouter, left_outer	LEFT JOIN
RightOuter.sql	right, rightouter, right_outer	RIGHT JOIN
Cross.sql	cross	
LeftAnti.sql	anti, leftanti, left_anti	
LeftSemi.sql	semi, leftsemi, left_semi	

All Join objects are defined at joinTypes (https://github.com/apache/spark/blob/master/sql/catalyst/src/main/scala/org/apache/spark/sql/catalyst/plans/joinTypes.scala) class, In order to use these you

[Spark SQL – StructType | StructField (https://sparkbyexamples.com/spark/spark-sql-structtype-on-dataframe/)](https://sparkbyexamples.com/spark/spark-sql-structtype-on-dataframe/)

[Spark SQL – Schema (https://sparkbyexamples.com/spark/spark-schema-explained-with-examples/)](https://sparkbyexamples.com/spark/spark-schema-explained-with-examples/)

[Spark SQL – Groupby (https://sparkbyexamples.com/spark/using-groupby-on-dataframe/)](https://sparkbyexamples.com/spark/using-groupby-on-dataframe/)

[Spark SQL – Sort DataFrame (https://sparkbyexamples.com/spark/spark-how-to-sort-dataframe-column-explained/)](https://sparkbyexamples.com/spark/spark-how-to-sort-dataframe-column-explained/)

[Spark SQL – Join Types (https://sparkbyexamples.com/spark/spark-sql-dataframe-join/)](https://sparkbyexamples.com/spark/spark-sql-dataframe-join/)

[Spark SQL – Union and UnionAll (https://sparkbyexamples.com/spark/spark-dataframe-union-and-union-all/)](https://sparkbyexamples.com/spark/spark-dataframe-union-and-union-all/)

[Spark SQL – map() vs mapPartitions() (https://sparkbyexamples.com/spark/spark-map-vs-mappartitions-transformation/)](https://sparkbyexamples.com/spark/spark-map-vs-mappartitions-transformation/)

[Spark SQL – foreach() vs foreachPartition() (https://sparkbyexamples.com/spark/spark-foreachpartition-vs-foreach-explained/)](https://sparkbyexamples.com/spark/spark-foreachpartition-vs-foreach-explained/)

[Spark SQL – map() vs flatMap() (https://sparkbyexamples.com/spark/spark-map-vs-flatmap-with-examples/)](https://sparkbyexamples.com/spark/spark-map-vs-flatmap-with-examples/)

[Spark SQL – Persist and Cache (https://sparkbyexamples.com/spark/spark-dataframe-cache-and-persist-explained/)](https://sparkbyexamples.com/spark/spark-dataframe-cache-and-persist-explained/)

[Spark SQL – UDF (User Defined Functions) (https://sparkbyexamples.com/spark/spark-sql-udf/)](https://sparkbyexamples.com/spark/spark-sql-udf/)

[Spark SQL – Array (ArrayType) Column (https://sparkbyexamples.com/spark/spark-array-arraytype-dataframe-column/)](https://sparkbyexamples.com/spark/spark-array-arraytype-dataframe-column/)

need to import `org.apache.spark.sql.catalyst.plans.{LeftOuter,Inner,....}`.

Before we jump into Spark SQL Join examples, first, let's create an `emp` and `dept` [DataFrame's (https://sparkbyexamples.com/spark/different-ways-to-create-a-spark-dataframe/)](https://sparkbyexamples.com/spark/different-ways-to-create-a-spark-dataframe/). here, column `emp_id` is unique on emp and `dept_id` is unique on the dept dataset's and emp_dept_id from emp has a reference to dept_id on dept dataset.

```
val emp = Seq((1,"Smith",-1,"
  (2,"Rose",1,"2010","20","M"
  (3,"Williams",1,"2010","10"
  (4,"Jones",2,"2005","10","F
  (5,"Brown",2,"2010","40",""
    (6,"Brown",2,"2010","50",
)
val empColumns = Seq("emp_id"
     "emp_dept_id","gender","
import spark.sqlContext.impli
val empDF = emp.toDF(empColum
empDF.show(false)

val dept = Seq(("Finance",10)
  ("Marketing",20),
  ("Sales",30),
  ("IT",40)
)

val deptColumns = Seq("dept_n
val deptDF = dept.toDF(deptCo
deptDF.show(false)
```

This print "emp" and "dept" DataFrame to console.

[Spark SQL – Map (MapType) column (https://sparkbyexamples.com/spark/spark-dataframe-map-maptype-column/)](https://sparkbyexamples.com/spark/spark-dataframe-map-maptype-column/)

[Spark SQL – Flatten Nested Struct Column (https://sparkbyexamples.com/spark/spark-flatten-nested-struct-column/)](https://sparkbyexamples.com/spark/spark-flatten-nested-struct-column/)

[Spark SQL – Flatten Nested Array Column (https://sparkbyexamples.com/spark/spark-flatten-nested-array-column-to-single-column/)](https://sparkbyexamples.com/spark/spark-flatten-nested-array-column-to-single-column/)

[Spark SQL – Explode Array & Map Columns (https://sparkbyexamples.com/spark/explode-spark-array-and-map-dataframe-column/)](https://sparkbyexamples.com/spark/explode-spark-array-and-map-dataframe-column/)

[Spark SQL – Sampling (https://sparkbyexamples.com/spark/spark-sampling-with-examples/)](https://sparkbyexamples.com/spark/spark-sampling-with-examples/)

[Spark SQL – Partitioning (https://sparkbyexamples.com/spark/spark-partitioning-understanding/)](https://sparkbyexamples.com/spark/spark-partitioning-understanding/)

Spark SQL Functions

[Spark SQL String Functions (https://sparkbyexamples.com/spark/usage-of-spark-sql-string-functions/)](https://sparkbyexamples.com/spark/usage-of-spark-sql-string-functions/)

[Spark SQL Date and Timestamp Functions (https://sparkbyexamples.com/spark/spark-sql-date-and-time-functions/)](https://sparkbyexamples.com/spark/spark-sql-date-and-time-functions/)

[Spark SQL Array Functions (https://sparkbyexamples.com/spark/spark-sql-array-functions/)](https://sparkbyexamples.com/spark/spark-sql-array-functions/)

[Spark SQL Map Functions (https://sparkbyexamples.com/spark/spark-sql-map-functions/)](https://sparkbyexamples.com/spark/spark-sql-map-functions/)

[Spark SQL Sort Functions (https://sparkbyexamples.com/spark/spark-sql-sort-functions/)](https://sparkbyexamples.com/spark/spark-sql-sort-functions/)

[Spark SQL Aggregate Functions (https://sparkbyexamples.com/](https://sparkbyexamples.com/)

```
Emp Dataset
+------+--------+--------------
|emp_id|name    |superior_emp_i
+------+--------+--------------
|1     |Smith   |-1
|2     |Rose    |1
|3     |Williams|1
|4     |Jones   |2
|5     |Brown   |2
|6     |Brown   |2
+------+--------+--------------

Dept Dataset
+---------+-------+
|dept_name|dept_id|
+---------+-------+
|Finance  |10     |
|Marketing|20     |
|Sales    |30     |
|IT       |40     |
+---------+-------+
```

2. Inner Join

Spark `Inner` join is the default join and it's mostly used, It is used to join two DataFrames/Datasets on key columns, and where keys don't match the rows get dropped from both datasets (`emp` & `dept`).

```
  empDF.join(deptDF,empDF("emp_
    .show(false)
```

When we apply Inner join on our datasets, It drops "`emp_dept_id`" 50 from "`emp`" and "`dept_id`" 30 from "`dept`" datasets. Below is the result of the above Join expression.

spark/spark-sql-aggregate-functions/)

Spark SQL Window Functions (https://sparkbyexamples.com/spark/spark-sql-window-functions/)

Spark SQL JSON Functions (https://sparkbyexamples.com/spark/spark-most-used-json-functions-with-examples/)

Spark Data Source API

Spark – Read & Write CSV file (https://sparkbyexamples.com/spark/spark-read-csv-file-into-dataframe/)

Spark – Read and Write JSON file (https://sparkbyexamples.com/spark/spark-read-and-write-json-file/)

Spark – Read & Write Parquet file (https://sparkbyexamples.com/spark/spark-read-write-dataframe-parquet-example/)

Spark – Read & Write XML file (https://sparkbyexamples.com/spark/spark-read-write-xml/)

Spark – Read & Write Avro files (https://sparkbyexamples.com/spark/read-write-avro-file-spark-dataframe/)

Spark – Read & Write Avro files (Spark version 2.3.x or earlier) (https://sparkbyexamples.com/spark/using-avro-data-files-from-spark-sql-2-3-x/)

Spark – Read & Write HBase using “hbase-spark” Connector (https://sparkbyexamples.com/spark/spark-read-write-using-hbase-spark-connector/)

Spark – Read & Write from HBase using Hortonworks (https://sparkbyexamples.com/spark/create-spark-dataframe-from-hbase-using-hortonworks/)

Spark – Read & Write ORC file (https://sparkbyexamples.com/spark/spark-read-orc-file-into-dataframe/)

```
+------+--------+--------------
|emp_id|name    |superior_emp_i
+------+--------+--------------
|1     |Smith   |-1
|2     |Rose    |1
|3     |Williams|1
|4     |Jones   |2
|5     |Brown   |2
+------+--------+--------------
```

3. Full Outer Join

`Outer` a.k.a `full`, `fullouter` join returns all rows from both Spark DataFrame/Datasets, where join expression doesn’t match it returns null on respective record columns.

```
empDF.join(deptDF,empDF("emp_
   .show(false)
empDF.join(deptDF,empDF("emp_
   .show(false)
empDF.join(deptDF,empDF("emp_
   .show(false)
```

From our “`emp`” dataset’s “`emp_dept_id`” with value 50 doesn’t have a record on “`dept`” hence dept columns have null and “`dept_id`” 30 doesn’t have a record in “`emp`” hence you see null’s on emp columns. Below is the result of the above Join expression.

```
+------+--------+--------------
|emp_id|name    |superior_emp_i
+------+--------+--------------
|2     |Rose    |1
|5     |Brown   |2
|1     |Smith   |-1
|3     |Williams|1
|4     |Jones   |2
|6     |Brown   |2
|null  |null    |null
+------+--------+--------------
```

Spark – Read Binary File (https://sparkbyexamples.com/spark/spark-read-binary-file-into-dataframe/)

Spark Streaming & Kafka

Spark Streaming – OutputModes (https://sparkbyexamples.com/spark/spark-streaming-outputmode/)

Spark Streaming – Reading Files From Directory (https://sparkbyexamples.com/spark/spark-streaming-read-json-files-from-directory/)

Spark Streaming – Reading Data From TCP Socket (https://sparkbyexamples.com/spark/spark-streaming-from-tcp-socket/)

Spark Streaming – Processing Kafka Messages in JSON Format (https://sparkbyexamples.com/spark/spark-streaming-with-kafka/)

Spark Streaming – Processing Kafka messages in AVRO Format (https://sparkbyexamples.com/spark/spark-streaming-consume-and-produce-kafka-messages-in-avro-format/)

Spark SQL Batch – Consume & Produce Kafka Message (https://sparkbyexamples.com/spark/spark-batch-processing-produce-consume-kafka-topic/)

4. Left Outer Join

Spark `Left` a.k.a `Left Outer` join returns all rows from the left DataFrame/Dataset regardless of match found on the right dataset when join expression doesn’t match, it assigns null for that record and drops records from right where match not found.

```
  empDF.join(deptDF,empDF("emp_
    .show(false)
  empDF.join(deptDF,empDF("emp_
    .show(false)
```

From our dataset, “`emp_dept_id`” 5o doesn’t have a record on “`dept`” dataset hence, this record contains null on “`dept`” columns (dept_name & dept_id). and “`dept_id`” 30 from “`dept`” dataset dropped from the results. Below is the result of the above Join expression.

```
+------+--------+--------------
|emp_id|name    |superior_emp_i
+------+--------+--------------
|1     |Smith   |-1
|2     |Rose    |1
|3     |Williams|1
|4     |Jones   |2
|5     |Brown   |2
|6     |Brown   |2
+------+--------+--------------
```

5. Right Outer Join

Spark `Right` a.k.a `Right Outer` join is opposite of `left` join, here it returns all rows from the right DataFrame/Dataset regardless of math found on the left dataset, when join expression doesn’t match, it assigns null for that record and drops records from left where match not found.

```
empDF.join(deptDF,empDF("emp_
  .show(false)
empDF.join(deptDF,empDF("emp_
  .show(false)
```

n our example, the right dataset pt_id" 30 doesn't have it on the left aset "emp" hence, this record tains null on "emp" columns. and p_dept_id" 50 dropped as a match found on left. Below is the result of above Join expression.

```
------+--------+--------------
emp_id|name    |superior_emp_i
------+--------+--------------
4     |Jones   |2
3     |Williams|1
1     |Smith   |-1
2     |Rose    |1
null  |null    |null
|5     |Brown   |2
+------+--------+--------------
```

6. Left Semi Join

Spark `Left Semi` join is similar to `inner` join difference being `leftsemi` join returns all columns from the left DataFrame/Dataset and ignores all columns from the right dataset. In other words, this join returns columns from the only left dataset for the records match in the right dataset on join expression, records not matched on join expression are ignored from both left and right datasets.

The same result can be achieved using select on the result of the inner join however, using this join would be efficient.

```
empDF.join(deptDF,empDF("emp_
  .show(false)
```

Below is the result of the above join expression.

```
leftsemi join
+------+--------+--------------
|emp_id|name    |superior_emp_i
+------+--------+--------------
|1     |Smith   |-1
|2     |Rose    |1
|3     |Williams|1
|4     |Jones   |2
|5     |Brown   |2
+------+--------+--------------
```

7. Left Anti Join

`Left Anti` join does the exact opposite of the Spark `leftsemi` join, `leftanti` join returns only columns from the left DataFrame/Dataset for non-matched records.

```
  empDF.join(deptDF,empDF("emp_
    .show(false)
```

Yields below output

```
+------+-----+-----------------+-
|emp_id|name |superior_emp_id|y
+------+-----+-----------------+-
|6     |Brown|2                |2
+------+-----+-----------------+-
```

8. Self Join

Spark Joins are not complete without a self join, Though there is no self-join type available, we can use any of the above-explained join types to join DataFrame to itself. below example use `inner` self join

```
  empDF.as("emp1").join(empDF.a
    col("emp1.superior_emp_id")
    .select(col("emp1.emp_id"),
      col("emp2.emp_id").as("su
      col("emp2.name").as("supe
      .show(false)
```

Here, we are joining `emp` dataset with itself to find out superior `emp_id` and `name` for all employees.

```
+------+--------+--------------
|emp_id|name    |superior_emp_i
+------+--------+--------------
|2     |Rose    |1
|3     |Williams|1
|4     |Jones   |2
|5     |Brown   |2
|6     |Brown   |2
+------+--------+--------------
```

9. Using SQL Expression

Since Spark SQL support native SQL syntax, we can also write join operations after creating temporary tables on DataFrame’s and using `spark.sql()`

```
  empDF.createOrReplaceTempView
  deptDF.createOrReplaceTempVie
//SQL JOIN
  val joinDF = spark.sql("selec
  joinDF.show(false)

  val joinDF2 = spark.sql("sele
  joinDF2.show(false)
```

10. Source Code | Scala Example

```
package com.sparkbyexamples.spa

import org.apache.spark.sql.Spa
import org.apache.spark.sql.fun
object JoinExample extends App

  val spark: SparkSession = Spa
    .master("local[1]")
    .appName("SparkByExamples.c
    .getOrCreate()

  spark.sparkContext.setLogLeve

  val emp = Seq((1,"Smith",-1,"
    (2,"Rose",1,"2010","20","M"
    (3,"Williams",1,"2010","10"
    (4,"Jones",2,"2005","10","F
    (5,"Brown",2,"2010","40",""
      (6,"Brown",2,"2010","50",
  )
  val empColumns = Seq("emp_id"
  import spark.sqlContext.impli
  val empDF = emp.toDF(empColum
  empDF.show(false)

  val dept = Seq(("Finance",10)
    ("Marketing",20),
    ("Sales",30),
    ("IT",40)
  )

  val deptColumns = Seq("dept_n
  val deptDF = dept.toDF(deptCo
  deptDF.show(false)


  println("Inner join")
  empDF.join(deptDF,empDF("emp_
    .show(false)

  println("Outer join")
  empDF.join(deptDF,empDF("emp_
    .show(false)
  println("full join")
  empDF.join(deptDF,empDF("emp_
    .show(false)
  println("fullouter join")
  empDF.join(deptDF,empDF("emp_
    .show(false)

  println("right join")
  empDF.join(deptDF,empDF("emp_
    .show(false)
  println("rightouter join")
  empDF.join(deptDF,empDF("emp_
    .show(false)

  println("left join")
  empDF.join(deptDF,empDF("emp_
    .show(false)
  println("leftouter join")
  empDF.join(deptDF,empDF("emp_
    .show(false)

  println("leftanti join")
  empDF.join(deptDF,empDF("emp_
    .show(false)

  println("leftsemi join")
  empDF.join(deptDF,empDF("emp_
    .show(false)

  println("cross join")
  empDF.join(deptDF,empDF("emp_
    .show(false)

  println("Using crossJoin()")
  empDF.crossJoin(deptDF).show(

  println("self join")
  empDF.as("emp1").join(empDF.a
    col("emp1.superior_emp_id")
    .select(col("emp1.emp_id"),
      col("emp2.emp_id").as("su
      col("emp2.name").as("supe
      .show(false)

  empDF.createOrReplaceTempView
  deptDF.createOrReplaceTempVie

  //SQL JOIN
  val joinDF = spark.sql("selec
  joinDF.show(false)

  val joinDF2 = spark.sql("sele
  joinDF2.show(false)

}
```

Examples explained here are available at the GitHub (https://github.com/spark-examples/spark-scala-examples/blob/master/src/main/scala/com/sparkbyexamples/spark/dataframe/join/JoinExample.scala) project for reference.

Conclusion

In this tutorial, you have learned Spark SQL Join types `INNER`, `LEFT OUTER`, `RIGHT OUTER`, `LEFT ANTI`, `LEFT SEMI`, `CROSS`, `SELF` joins usage, and examples with Scala.

References:

- W3schools (https://www.w3schools.com/sql/sql_join.asp)

Happy Learning !!

Share this:

(https://sparkbyexamples.com/spark/spark-sql-dataframe-join/?share=facebook&nb=1)

(https://sparkbyexamples.com/spark/spark-sql-dataframe-join/?share=reddit&nb=1)

(https://sparkbyexamples.com/spark/spark-sql-dataframe-join/?share=pinterest&nb=1)

(https://sparkbyexamples.com/spark/spark-sql-dataframe-join/?share=tumblr&nb=1)

(https://sparkbyexamples.com/spark/spark-sql-dataframe-join/?share=pocket&nb=1)

(https://sparkbyexamples.com/spark/spark-sql-dataframe-join/?share=linkedin&nb=1)

(https://sparkbyexamples.com/spark/spark-sql-dataframe-join/?share=twitter&nb=1)

TAGS: CROSS JOIN (HTTPS://SPARKBYEXAMPLES.COM/TAG/CROSS-JOIN/), DATAFRAME JOIN (HTTPS://SPARKBYEXAMPLES.COM/TAG/DATAFRAME-JOIN/), INNER JOIN (HTTPS://SPARKBYEXAMPLES.COM/TAG/INNER-JOIN/), LEFT ANTI SEMI JOIN (HTTPS://SPARKBYEXAMPLES.COM/TAG/LEFT-ANTI-SEMI-JOIN/), LEFT JOIN (HTTPS://SPARKBYEXAMPLES.COM/TAG/LEFT-JOIN/), LEFT SEMI JOIN (HTTPS://SPARKBYEXAMPLES.COM/TAG/LEFT-SEMI-JOIN/), OUTER JOIN (HTTPS://SPARKBYEXAMPLES.COM/TAG/OUTER-JOIN/), RIGHT JOIN (HTTPS://SPARKBYEXAMPLES.COM/TAG/RIGHT-JOIN/), SQL JOIN (HTTPS://SPARKBYEXAMPLES.COM/TAG/SQL-JOIN/)

(https://sparkbyexa

NNK (Https://Sparkbyexamples.Com/Author/Admin/)

SparkByExamples.com is a Big Data and Spark examples community page, all examples are simple and mples.com/author/admin/)

easy to understand and well tested in our development environment Read more .. (https://sparkbyexamples.com/about-sparkbyexamples/)

› THIS POST HAS 7 COMMENTS

Nikhil 20 MAR 2021 REPLY

Please help me to resolve error:
I have 2 df
CONTRACT_PBP_LISTDF

```
+———–+
|CONTRACT_NBR|
+———–+
|H0755 |
|H2961 |
|H0151 |
|H0303 |
|H0315 |
```

Trnf_PDE_ContractDF

```
+————-+
|CONTRACT_NBR1|
+————-+
|H2531 |
+————-+
```

applying left join:

```
Trnf_PDE_ContractDF.join(CONTRACT_PBP_LISTDF, Trnf_PDE_ContractDF("CONTRACT_NBR1") === CONTRACT_PBP_LISTDF("CONTRACT_NBR"), "left").show(false)
```

getting error:

```
Exception in thread "main" org.apache.spark.sql.AnalysisException: Detected implicit cartesian product for LEFT OUTER join between logical plans
Aggregate [H2531]
+- Project
+- Filter (isnotnull(_c0#16) && (substring(_c0#16, 1, 3) = BTR))
+- Relation[_c0#16] csv
and
Filter (isnotnull(CONTRACT_NBR#0) && (H2531 = CONTRACT_NBR#0))
+- Relation[CONTRACT_NBR#0] JDBCRelation((select NVL(contract_nbr, ") as CONTRACT_NBR from ofsc.contract_pbp_list where actv_ind = 'Y')) [numPartitions=1]
Join condition is missing or trivial.
Either: use the CROSS JOIN syntax to
```

```
allow cartesian products between these
relations, or: enable implicit cartesian
products by setting the configuration
variable
spark.sql.crossJoin.enabled=true;
at
org.apache.spark.sql.catalyst.optimizer.
CheckCartesianProducts$$anonfun$app
ly$22.applyOrElse(Optimizer.scala:1295
)
at
org.apache.spark.sql.catalyst.optimizer.
CheckCartesianProducts$$anonfun$app
ly$22.applyOrElse(Optimizer.scala:1292
)
at
org.apache.spark.sql.catalyst.trees.Tre
eNode$$anonfun$2.apply(TreeNode.sca
la:256)
at
org.apache.spark.sql.catalyst.trees.Tre
eNode$$anonfun$2.apply(TreeNode.sca
la:256)
at
org.apache.spark.sql.catalyst.trees.Curr
entOrigin$.withOrigin(TreeNode.scala:7
0)
at
org.apache.spark.sql.catalyst.trees.Tre
eNode.transformDown(TreeNode.scala:
255)
at
org.apache.spark.sql.catalyst.plans.logi
cal.LogicalPlan.org$apache$spark$sql$
catalyst$plans$logical$AnalysisHelper$
$super$transformDown(LogicalPlan.sca
la:29)
at
org.apache.spark.sql.catalyst.plans.logi
cal.AnalysisHelper$class.transformDow
n(AnalysisHelper.scala:149)
at
org.apache.spark.sql.catalyst.plans.logi
cal.LogicalPlan.transformDown(Logical
Plan.scala:29)
at
org.apache.spark.sql.catalyst.plans.logi
cal.LogicalPlan.transformDown(Logical
Plan.scala:29)
at
org.apache.spark.sql.catalyst.trees.Tre
eNode$$anonfun$transformDown$1.app
ly(TreeNode.scala:261)
at
```

```
org.apache.spark.sql.catalyst.trees.TreeNode$$anonfun$transformDown$1.apply(TreeNode.scala:261)
at org.apache.spark.sql.catalyst.trees.TreeNode$$anonfun$4.apply(TreeNode.scala:326)
at org.apache.spark.sql.catalyst.trees.TreeNode.mapProductIterator(TreeNode.scala:187)
at org.apache.spark.sql.catalyst.trees.TreeNode.mapChildren(TreeNode.scala:324)
at org.apache.spark.sql.catalyst.trees.TreeNode.transformDown(TreeNode.scala:261)
at org.apache.spark.sql.catalyst.plans.logical.LogicalPlan.org$apache$spark$sql$catalyst$plans$logical$AnalysisHelper$$super$transformDown(LogicalPlan.scala:29)
at org.apache.spark.sql.catalyst.plans.logical.AnalysisHelper$class.transformDown(AnalysisHelper.scala:149)
at org.apache.spark.sql.catalyst.plans.logical.LogicalPlan.transformDown(LogicalPlan.scala:29)
at org.apache.spark.sql.catalyst.plans.logical.LogicalPlan.transformDown(LogicalPlan.scala:29)
at org.apache.spark.sql.catalyst.trees.TreeNode$$anonfun$transformDown$1.apply(TreeNode.scala:261)
at org.apache.spark.sql.catalyst.trees.TreeNode$$anonfun$transformDown$1.apply(TreeNode.scala:261)
at org.apache.spark.sql.catalyst.trees.TreeNode$$anonfun$4.apply(TreeNode.scala:326)
at org.apache.spark.sql.catalyst.trees.TreeNode.mapProductIterator(TreeNode.scala:187)
at
```

```
org.apache.spark.sql.catalyst.trees.TreeNode.mapChildren(TreeNode.scala:324)
at
org.apache.spark.sql.catalyst.trees.TreeNode.transformDown(TreeNode.scala:261)
at
org.apache.spark.sql.catalyst.plans.logical.LogicalPlan.org$apache$spark$sql$catalyst$plans$logical$AnalysisHelper$$super$transformDown(LogicalPlan.scala:29)
at
org.apache.spark.sql.catalyst.plans.logical.AnalysisHelper$class.transformDown(AnalysisHelper.scala:149)
at
org.apache.spark.sql.catalyst.plans.logical.LogicalPlan.transformDown(LogicalPlan.scala:29)
at
org.apache.spark.sql.catalyst.plans.logical.LogicalPlan.transformDown(LogicalPlan.scala:29)
at
org.apache.spark.sql.catalyst.trees.TreeNode$$anonfun$transformDown$1.apply(TreeNode.scala:261)
at
org.apache.spark.sql.catalyst.trees.TreeNode$$anonfun$transformDown$1.apply(TreeNode.scala:261)
at
org.apache.spark.sql.catalyst.trees.TreeNode$$anonfun$4.apply(TreeNode.scala:326)
at
org.apache.spark.sql.catalyst.trees.TreeNode.mapProductIterator(TreeNode.scala:187)
at
org.apache.spark.sql.catalyst.trees.TreeNode.mapChildren(TreeNode.scala:324)
at
org.apache.spark.sql.catalyst.trees.TreeNode.transformDown(TreeNode.scala:261)
at
org.apache.spark.sql.catalyst.plans.logical.LogicalPlan.org$apache$spark$sql$catalyst$plans$logical$AnalysisHelper$$super$transformDown(LogicalPlan.sca
la:29)
at
org.apache.spark.sql.catalyst.plans.logical.AnalysisHelper$class.transformDown(AnalysisHelper.scala:149)
at
org.apache.spark.sql.catalyst.plans.logical.LogicalPlan.transformDown(LogicalPlan.scala:29)
at
org.apache.spark.sql.catalyst.plans.logical.LogicalPlan.transformDown(LogicalPlan.scala:29)
at
org.apache.spark.sql.catalyst.trees.TreeNode.transform(TreeNode.scala:245)
at
org.apache.spark.sql.catalyst.optimizer.CheckCartesianProducts$.apply(Optimizer.scala:1292)
at
org.apache.spark.sql.catalyst.optimizer.CheckCartesianProducts$.apply(Optimizer.scala:1274)
at
org.apache.spark.sql.catalyst.rules.RuleExecutor$$anonfun$execute$1$$anonfun$apply$1.apply(RuleExecutor.scala:87)
at
org.apache.spark.sql.catalyst.rules.RuleExecutor$$anonfun$execute$1$$anonfun$apply$1.apply(RuleExecutor.scala:84)
at
scala.collection.IndexedSeqOptimized$class.foldl(IndexedSeqOptimized.scala:57)
at
scala.collection.IndexedSeqOptimized$class.foldLeft(IndexedSeqOptimized.scala:66)
at
scala.collection.mutable.WrappedArray.foldLeft(WrappedArray.scala:35)
at
org.apache.spark.sql.catalyst.rules.RuleExecutor$$anonfun$execute$1.apply(RuleExecutor.scala:84)
at
org.apache.spark.sql.catalyst.rules.RuleExecutor$$anonfun$execute$1.apply(RuleExecutor.scala:76)
at
```

```
scala.collection.immutable.List.foreach(List.scala:392)
at org.apache.spark.sql.catalyst.rules.RuleExecutor.execute(RuleExecutor.scala:76)
at org.apache.spark.sql.execution.QueryExecution.optimizedPlan$lzycompute(QueryExecution.scala:66)
at org.apache.spark.sql.execution.QueryExecution.optimizedPlan(QueryExecution.scala:66)
at org.apache.spark.sql.execution.QueryExecution.sparkPlan$lzycompute(QueryExecution.scala:72)
at org.apache.spark.sql.execution.QueryExecution.sparkPlan(QueryExecution.scala:68)
at org.apache.spark.sql.execution.QueryExecution.executedPlan$lzycompute(QueryExecution.scala:77)
at org.apache.spark.sql.execution.QueryExecution.executedPlan(QueryExecution.scala:77)
at org.apache.spark.sql.Dataset.withAction(Dataset.scala:3359)
at org.apache.spark.sql.Dataset.head(Dataset.scala:2544)
at org.apache.spark.sql.Dataset.take(Dataset.scala:2758)
at org.apache.spark.sql.Dataset.getRows(Dataset.scala:254)
at org.apache.spark.sql.Dataset.showString(Dataset.scala:291)
at org.apache.spark.sql.Dataset.show(Dataset.scala:747)
at org.apache.spark.sql.Dataset.show(Dataset.scala:724)
at com.optum.etlmodernization.ofsc.PdeSubmitContractVal2C$.main(PDE_Submit
_contract_val_2c.scala:99)
at
com.optum.etlmodernization.ofsc.PdeSubmitContractVal2C.main(PDE_Submit_contract_val_2c.scala)
Process finished with exit code 1
```

NNK 22 MAR 2021 REPLY

May I know what version of Spark are you using?

sunilbhola

14 NOV 2020 REPLY

Inner join section – When we apply Inner join on our datasets, It drops "emp_dept_id" 60 from — it should be 50 not 60
|6 |Brown |2 |2010 |50 |
|-1 |

NNK 15 NOV 2020 REPLY

Thanks, Sunilbhola for correcting it. It's a typo and has fixed now.

Vaggelis 8 NOV 2020 REPLY

Very nice tutorials and thank you very much for the content but this is not applicable to multiple dataframes JOIN. It works only for two dataframes.

NNK 8 NOV 2020 REPLY

Hi Vaggelis, Thanks for your comments. Agree with you. I have another article [Spark SQL Join Multiple DataFrames]()

(https://sparkbyexamples.com/spark/spark-join-multiple-dataframes/), please check.

Anonymous

22 JAN 2020 REPLY

very informative

Leave a Reply

Categories

Apache Hadoop (https://sparkbyexamples.com/category/hadoop/)

Apache Spark (https://sparkbyexamples.com/category/spark/)

Recent Posts

Spark regexp_replace() – Replace String Value (https://sparkbyexamples.com/spark/spark-regexp_replace-replace-string-value/)

How to Run a PySpark Script from Python?

About SparkByExamples.Com

SparkByExamples.com is a Big Data and Spark examples community page, all examples are simple and easy to understand, and well tested in our development environment Read more .. (https://sparkbyexamples.com/about-sparkbyexamples/)

Apache Spark Streaming (https://sparkbyexamples.com/category/spark/apache-spark-streaming/)

Apache Kafka (https://sparkbyexamples.com/category/kafka/)

Apache HBase (https://sparkbyexamples.com/category/hbase/)

Apache Cassandra (https://sparkbyexamples.com/category/cassandra/)

Snowflake Database (https://sparkbyexamples.com/category/snowflake/)

H2O Sparkling Water (https://sparkbyexamples.com/category/h2o-sparkling-water/)

PySpark (https://sparkbyexamples.com/category/pyspark/)

(https://sparkbyexamples.com/pyspark/run-pyspark-script-from-python-subprocess/)

Spark SQL like() Using Wildcard Example (https://sparkbyexamples.com/spark/spark-sql-like-using-wildcard-example/)

Spark isin() & IS NOT IN Operator Example (https://sparkbyexamples.com/spark/spark-isin-is-not-in-operator-example/)

Spark – Get Size/Length of Array & Map Column (https://sparkbyexamples.com/spark/spark-get-size-length-of-array-map-column/)

Spark Using Length/Size Of a DataFrame Column (https://sparkbyexamples.com/spark/spark-using-length-size-of-a-dataframe-column/)

Spark rlike() Working with Regex Matching Examples (https://sparkbyexamples.com/spark/spark-rlike-regex-matching-examples/)

Spark Check String Column Has Numeric Values (https://sparkbyexamples.com/spark/spark-check-string-column-has-numeric-values/)

Spark Check Column Data Type is Integer or String (https://sparkbyexamples.com/spark/spark-check-column-data-type-is-integer-or-string/)

Follow Us

(https://twitter.com/sparkbyexamples)

(https://www.facebook.com/sparkbyexamples/)

(https://www.linkedin.com/in/n-nk-b860a8193/)

(https://github.com/spark-examples/)