Apache Spark SQL join types interview Q&As

Posted on March 12, 2022

Q1. What are the different Spark SQL join types?
A1. There are different SQL join types like inner join, left/right outer joins, full outer join, left semi-join, left anti-join and self-join.

Q2. Given the below tables, can give examples of the above join types?

```
1
2  package com.myapp
3
4  import org.apache.spark.sql.SparkSession
5
6  object MySparkApp4 {
7
8    case class Customer(code: String, name: String)
9    case class Order(id: String, product: String, custCode: String)
10
11   def main(args: Array[String]): Unit = {
12     val spark = SparkSession.builder().master("local[*]").appName("S
13
14     // Create the Customers
15     val cust1 = new Customer("10", "Sam")
16     val cust2 = new Customer("20", "Peter")
17     val cust3 = new Customer("30", "Mary")
18
19     val customers = Seq(cust1, cust2, cust3)
20     val dfCustomers = spark.sqlContext.createDataFrame(customers)
21
22     // Create the orders
23     val order1 = new Order("123", "TV", "10")
24     val order2 = new Order("456", "Phone", "20")
25     val order3 = new Order("789", "Radio", "40")
26
27     val orders = Seq(order1, order2, order3)
28     val dfOrders = spark.sqlContext.createDataFrame(orders)
29
30     dfCustomers.show()
31     dfOrders.show()
32
33   }
34
35 }
36
```

300+ Java Interview FAQs

300+ Java FAQs 🔥
16+ Java Key Areas Q&As
150+ Java Architect FAQs
80+ Java Code Quality Q&As
150+ Java Coding Q&As

300+ Big Data Interview FAQs

300+ Big Data FAQs 🔥

FAQs Data - 01: SQL
FAQs Data - 02: Data Modelling
FAQs Data - 02: Data Warehouse
FAQs Data - 03: Big Data
FAQs Data - 04: Hadoop (HDFS)
FAQs Data - 05: MapReduce
FAQs Data - 06: Hive
FAQs Data - 07: Impala
FAQs Data - 08: Spark
FAQs Data - 09: Spark SQL
FAQs Data - 10: Apache Kafka
FAQs Data - 11: Data Governance
FAQs Data - 11: NoSQL
FAQs Data - 12: Data security
FAQs Data - 13: Analytics & Science
FAQs Data - 14: AWS
FAQs Data - 15: Sqoop & Nifi
FAQs Data - 16: Yarn, Zookeeper
FAQs Data - 40: Scala 140+ FAQs

Output:

Customers table:

```
+----+-----+
|code| name|
+----+-----+
|  10|  Sam|
|  20|Peter|
|  30| Mary|
+----+-----+
```

Orders table:

```
+---+-------+--------+
| id|product|custCode|
+---+-------+--------+
|123|     TV|      10|
|456|  Phone|      20|
|789|  Radio|      40|
+---+-------+--------+
```

A2. Here are the join examples. The tables are joined on code & custCode.

inner join

The **inner** is the default join in Spark and most commonly used. It drops columns that are not matched by the keys. As you can see below only codes 10 & 20 are displayed as they are in both tables.

```
dfCustomers.join(dfOrders,dfCustomers("code") ===  dfOrders("custCod
      .show(false)
```

Output:

```
+----+-----+---+-------+--------+
|code|name |id |product|custCode|
+----+-----+---+-------+--------+
|10  |Sam  |123|TV     |10      |
|20  |Peter|456|Phone  |20      |
+----+-----+---+-------+--------+
```

full outer join

The **full** or **fullouter** join returns all rows from both the tables, and where join expression doesn't match it returns null on respective record columns.

```
dfCustomers.join(dfOrders,dfCustomers("code") ===  dfOrders("custCo
    .show(false)
```

Output:

```
+----+-----+----+-------+--------+
|code|name |id  |product|custCode|
+----+-----+----+-------+--------+
|10  |Sam  |123 |TV     |10      |
|20  |Peter|456 |Phone  |20      |
|30  |Mary |null|null   |null    |
|null|null |789 |Radio  |40      |
+----+-----+----+-------+--------+
```

left outer join

The **left** or **leftouter** join returns all rows from the left table regardless of match found on the right table or not, and it assigns null for those records where no match in the right table. It drops records where no match is found in both tables. The **right** or **rightouter** will do the reverse.

```
dfCustomers.join(dfOrders,dfCustomers("code") ===  dfOrders("custCod
    .show(false)
```

Output:

```
+----+-----+----+-------+--------+
|code|name |id  |product|custCode|
+----+-----+----+-------+--------+
|10  |Sam  |123 |TV     |10      |
|20  |Peter|456 |Phone  |20      |
|30  |Mary |null|null   |null    |
+----+-----+----+-------+--------+
```

leftsemi join

The **leftsemi** join is similar to inner join with the difference being leftsemi join returns all columns from the left table and ignores all columns from the right table. The rightsemi does the opposite.

```
dfCustomers.join(dfOrders,dfCustomers("code") ===  dfOrders("custCode
    .show(false)
```

Output:

```
+----+-----+
```

```
|code|name |
+----+-----+
|10  |Sam  |
|20  |Peter|
+----+-----+
```

leftanti join

The **leftanti** join does the exact opposite of the Spark leftsemi join, where the join returns only the columns from the left table for which no match found in the right table. The rightsemi does the opposite.

```
dfCustomers.join(dfOrders,dfCustomers("code") ===  dfOrders("custCode
      .show(false)
```

Output:

```
+----+----+
|code|name|
+----+----+
|30  |Mary|
+----+----+
```

So, the **union** of "**leftsemi**" & "**leftanti**" will give the left table columns of the leftouter join.

```
+----+-----+
|code|name |
+----+-----+
|10  |Sam  |
|20  |Peter|
|30  |Mary |
+----+-----+
```

Join types are not complete without discussing self join, cartesian (cross) join & theta join. These joins don't have a specific join types. Let's create different sample data to demonstrate these joins.

self-join

The self-join can use any of the above mentioned join types to join a table to itself. The self-joins are useful in querying hierarchical data. The example below joins to itself to find the employee names & their managers.

```
package com.myapp

import org.apache.spark.sql.SparkSession
import org.apache.spark.sql.functions.{col}

object MySparkApp5 {
```

```
8
9   case class Employee(id: String, name: String,  managerId:String)
10
11  def main(args: Array[String]): Unit = {
12    val spark = SparkSession.builder().master("local[*]").appName("S
13
14    // Create the Employees
15    val employee1 = new Employee("10", "Peter", "20")
16    val employee2 = new Employee("20", "Mary", null)
17    val employee3 = new Employee("30", "Sam", "20")
18    val employee4 = new Employee("40", "Jessica", "30")
19    val employee5 = new Employee("50", "Anne", "10")
20
21    val employees = Seq(employee1, employee2, employee3, employee4,
22
23    val df1 = spark.sqlContext.createDataFrame(employees)
24    df1.show(false)
25
26    val dfSelfJoined = df1.as("emp").join(df1.as("mgr"),
27                         col("emp.managerId") === col("mgr.id"))
28                        .select(col("emp.name").as("employee"),
29                          col("mgr.name") as ("reports_to"))
30
31    dfSelfJoined.show(false)
32
33  }
34 }
35
36
```

Output:

```
1  +---+-------+---------+
2  |id |name   |managerId|
3  +---+-------+---------+
4  |10 |Peter  |20       |
5  |20 |Mary   |null     |
6  |30 |Sam    |20       |
7  |40 |Jessica|30       |
8  |50 |Anne   |10       |
9  +---+-------+---------+
10
11 +--------+----------+
12 |employee|reports_to|
13 +--------+----------+
14 |Anne    |Peter     |
15 |Sam     |Mary      |
16 |Peter   |Mary      |
17 |Jessica |Sam       |
18 +--------+----------+
19
20
```

cartesian or cross join

The cartesian joins generate a “cartesian product”, which is defined as the product of two tables. If all employees above can perform all of the tasks, then you get product of both tables. Cross joins of very large tables can lead to performance issues, hence refer to performance tuning post for strategies to handle cross joins of large tables.

The example below self joins on the length of names.

```
1
2
3  package com.myapp
```

```
4
5  import org.apache.spark.sql.SparkSession
6  import org.apache.spark.sql.functions.{col}
7
8  object MySparkApp5 {
9
10   case class Employee(id: String, name: String,  managerId:String)
11   case class Task(task_id: String, task_name: String)
12
13   def main(args: Array[String]): Unit = {
14     val spark = SparkSession.builder().master("local[*]").appName("S
15
16     // Create the Employees
17     val employee1 = new Employee("10", "Peter", "20")
18     val employee2 = new Employee("20", "Mary", null)
19     val employee3 = new Employee("30", "Sam", "20")
20     val employee4 = new Employee("40", "Jessica", "30")
21     val employee5 = new Employee("50", "Anne", "10")
22
23     val employees = Seq(employee1, employee2, employee3, employee4,
24
25     val dfEmployees = spark.sqlContext.createDataFrame(employees)
26     dfEmployees.show(false)
27
28     val task1 = new Task("TA-01", "Accounting")
29     val task2 = new Task("TA-02", "Marketing")
30     val task3 = new Task("TA-03", "Administrative")
31
32     val tasks = Seq(task1, task2, task3)
33     val dfTasks = spark.sqlContext.createDataFrame(tasks)
34     dfTasks.show(false)
35
36     val dfCrossJoin = dfEmployees.join(dfTasks)
37     dfCrossJoin.show(false)
38
39   }
40 }
41
42
```

Output:

```
1
2  +---+-------+---------+
3  |id |name   |managerId|
4  +---+-------+---------+
5  |10 |Peter  |20       |
6  |20 |Mary   |null     |
7  |30 |Sam    |20       |
8  |40 |Jessica|30       |
9  |50 |Anne   |10       |
10 +---+-------+---------+
11
12 +-------+--------------+
13 |task_id|task_name     |
14 +-------+--------------+
15 |TA-01  |Accounting    |
16 |TA-02  |Marketing     |
17 |TA-03  |Administrative|
18 +-------+--------------+
19
20 +---+-------+---------+-------+--------------+
21 |id |name   |managerId|task_id|task_name     |
22 +---+-------+---------+-------+--------------+
23 |10 |Peter  |20       |TA-01  |Accounting    |
24 |10 |Peter  |20       |TA-02  |Marketing     |
25 |10 |Peter  |20       |TA-03  |Administrative|
26 |20 |Mary   |null     |TA-01  |Accounting    |
27 |20 |Mary   |null     |TA-02  |Marketing     |
28 |20 |Mary   |null     |TA-03  |Administrative|
29 |30 |Sam    |20       |TA-01  |Accounting    |
30 |30 |Sam    |20       |TA-02  |Marketing     |
```

```
|30 |Sam    |20       |TA-03  |Administrative|
|40 |Jessica|30       |TA-01  |Accounting    |
|40 |Jessica|30       |TA-02  |Marketing     |
|40 |Jessica|30       |TA-03  |Administrative|
|50 |Anne   |10       |TA-01  |Accounting    |
|50 |Anne   |10       |TA-02  |Marketing     |
|50 |Anne   |10       |TA-03  |Administrative|
+---+-------+---------+-------+--------------+
```

theta join

A theta is a join that links tables based on a relationship other than the equality between two columns.

```
package com.myapp

import org.apache.spark.sql.SparkSession
import org.apache.spark.sql.functions.{col, length}

object MySparkApp5 {

  case class Employee(id: String, name: String,  managerId:String)
  case class Task(task_id: String, task_name: String)

  def main(args: Array[String]): Unit = {
    val spark = SparkSession.builder().master("local[*]").appName("S

    // Create the Employees
    val employee1 = new Employee("10", "Peter", "20")
    val employee2 = new Employee("20", "Mary", null)
    val employee3 = new Employee("30", "Sam", "20")
    val employee4 = new Employee("40", "Jessica", "30")
    val employee5 = new Employee("50", "Anne", "10")

    val employees = Seq(employee1, employee2, employee3, employee4,

    val dfEmployees = spark.sqlContext.createDataFrame(employees)
    dfEmployees.show(false)


    val dfThetaJoin = dfEmployees.as("emp1").join(dfEmployees.as("em
                           length(col("emp1.name")) > length(col("e
    dfThetaJoin.show(false)

  }
}
```

Output:

```
+---+-------+---------+
|id |name   |managerId|
+---+-------+---------+
|10 |Peter  |20       |
|20 |Mary   |null     |
|30 |Sam    |20       |
|40 |Jessica|30       |
|50 |Anne   |10       |
+---+-------+---------+

+---+-------+---------+---+-----+---------+
|id |name   |managerId|id |name |managerId|
+---+-------+---------+---+-----+---------+
|10 |Peter  |20       |20 |Mary |null     |
|10 |Peter  |20       |30 |Sam  |20       |
```

```
|10 |Peter  |20       |50 |Anne |10       |
|20 |Mary   |null     |30 |Sam  |20       |
|40 |Jessica|30       |10 |Peter|20       |
|40 |Jessica|30       |20 |Mary |null     |
|40 |Jessica|30       |30 |Sam  |20       |
|40 |Jessica|30       |50 |Anne |10       |
|50 |Anne   |10       |30 |Sam  |20       |
+---+-------+---------+---+-----+---------+

```

‹ Spark interview Q&As with coding examples in Scala – part 11: add column values conditionally, lit & typedlit

23: Scala Traits, Mixins, self-type annotation & DI with Cake pattern ›

Arulkumaran

Mechanical Engineer to self-taught Java engineer within 2 years & a **freelancer** within 3 years. Freelancing since 2003. Preparation empowered me to **attend 190+ job interviews** & choose from **150+ job offers** with sought-after contract rates. Author of the book "**Java/J2EE job interview companion**", which sold **35K+ copies** & superseded by this site with 2,050+ registered users. Amazon.com profile | **Reviews** | LinkedIn | LinkedIn Group | YouTube

Contact us: java-interview@hotmail.com