**SPARK SQL JOINS**

1. **In Which kind of Scenarios do we use Inner Join?**

Spark SQL Inner Join is used when you need to combine data from two or more tables based on a common column or columns. Specifically, it is used when you want to return only the rows that have matching values in both tables.

Here are some scenarios where you might use Spark SQL Inner Join:

* When you have a customer table and a sales table, and you want to combine them to see which customers have made purchases. You would join the tables on the customer ID column to get a table that shows all customers and their corresponding sales.
* When you have a product table and a transaction table, and you want to combine them to see which products have been sold. You would join the tables on the product ID column to get a table that shows all products and their corresponding transactions.
* When you have a user table and a friend table, and you want to combine them to see which users have friends. You would join the tables on the user ID column to get a table that shows all users and their corresponding friends.

In general, Inner Join is useful when you need to combine data from multiple tables and want to see only the data that is common to both tables.

1. **In Which kind of Scenarios do we use Left Join?**

Spark SQL Left Join is used when you need to combine data from two or more tables based on a common column or columns, and you want to return all the rows from the left table and the matching rows from the right table. In other words, it returns all the rows from the left table, and only the matching rows from the right table.

* Here are some scenarios where you might use Spark SQL Left Join:
* When you have a customer table and a sales table, and you want to see all customers, including those who have not made any purchases. You would join the tables on the customer ID column, using a Left Join, to get a table that shows all customers and their corresponding sales (if any).
* When you have a product table and a transaction table, and you want to see all products, including those that have not been sold. You would join the tables on the product ID column, using a Left Join, to get a table that shows all products and their corresponding transactions (if any).
* When you have a user table and a friend table, and you want to see all users, including those who have no friends. You would join the tables on the user ID column, using a Left Join, to get a table that shows all users and their corresponding friends (if any).

In general, Left Join is useful when you want to include all the rows from one table, even if there are no matching rows in the other table.

1. **In Which kind of Scenarios do we use Right Join?**

Spark SQL Right Join is used when you need to combine data from two or more tables based on a common column or columns, but you want to include all the rows from the right table, even if there are no matching rows in the left table. Specifically, it is used to return all the rows from the right table and matching rows from the left table.

Here are some scenarios where you might use Spark SQL Right Join:

* When you have a customer table and a sales table, and you want to combine them to see which customers have made purchases. You will use a right join if you want to see all the sales, even if they don't have corresponding customer records.
* When you have a product table and a transaction table, and you want to combine them to see which products have been sold. You will use a right join if you want to see all the transactions, even if they don't have corresponding product records.
* When you have a user table and a friend table, and you want to combine them to see which users have friends. You will use a right join if you want to see all the friends, even if they don't have corresponding user records.

In general, Right Join is useful when you want to include all the rows from the right table, even if there are no matching rows in the left table.

1. **In Which kind of Scenarios do we use Full Outer/Outer Join?**

Spark SQL Full Outer/Outer Join is used when you need to combine data from two or more tables based on a common column or columns, and you want to include all the rows from both tables, even if there are no matching rows in the other table. Specifically, it is used to return all the rows from both tables and matching rows where they exist.

Here are some scenarios where you might use Spark SQL Full Outer/Outer Join:

* When you have a customer table and a sales table, and you want to combine them to see which customers have made purchases. You will use a full outer join if you want to see all the customers and all the sales, even if they don't have corresponding records in the other table.
* When you have a product table and a transaction table, and you want to combine them to see which products have been sold. You will use a full outer join if you want to see all the products and all the transactions, even if they don't have corresponding records in the other table.
* When you have a user table and a friend table, and you want to combine them to see which users have friends. You will use a full outer join if you want to see all the users and all the friends, even if they don't have corresponding records in the other table.

In general, Full Outer/Outer Join is useful when you want to include all the rows from both tables, even if there are no matching rows in the other table.

1. **In Which kind of Scenarios do we use Left-Semi Join?**

Spark SQL Left-Semi Join is used when you want to retrieve only the rows from the left table that have matching rows in the right table. Specifically, it is used to return all the columns from the left table, but only the rows where a match exists in the right table.

Here are some scenarios where you might use Spark SQL Left-Semi Join:

* When you have a customer table and a sales table, and you want to retrieve only the customers who have made purchases. You would use a left-semi join to return only the customer records that have matching sales records.
* When you have a product table and a transaction table, and you want to retrieve only the products that have been sold. You would use a left-semi join to return only the product records that have matching transaction records.
* When you have a user table and a friend table, and you want to retrieve only the users who have friends. You would use a left-semi join to return only the user records that have matching friend records.

In general, Left-Semi Join is useful when you want to retrieve only the rows from the left table that have matching rows in the right table. It can be thought of as a filter on the left table based on the presence of matching rows in the right table.

1. **In Which kind of Scenarios do we use Left-Anti Join?**

Spark SQL Left-Anti Join is used when you want to retrieve only the rows from the left table that do not have matching rows in the right table. Specifically, it is used to return all the columns from the left table, but only the rows where a match does not exist in the right table.

Here are some scenarios where you might use Spark SQL Left-Anti Join:

* When you have a customer table and a sales table, and you want to retrieve only the customers who have not made purchases. You would use a left-anti join to return only the customer records that do not have matching sales records.
* When you have a product table and a transaction table, and you want to retrieve only the products that have not been sold. You would use a left-anti join to return only the product records that do not have matching transaction records.
* When you have a user table and a friend table, and you want to retrieve only the users who do not have friends. You would use a left-anti join to return only the user records that do not have matching friend records.

In general, Left-Anti Join is useful when you want to retrieve only the rows from the left table that do not have matching rows in the right table. It can be thought of as a filter on the left table based on the absence of matching rows in the right table.

1. **In Which kind of Scenarios do we use Self Join?**

Spark SQL Self Join is used when you want to join a table to itself. Specifically, it is used to return rows from a table where the values in one or more columns match values in another row of the same table.

Here are some scenarios where you might use Spark SQL Self Join:

* When you have a table of employees, and each employee has a manager. You can join the employee table to itself using the manager ID column to return a list of employees and their managers.
* When you have a table of products, and each product has a related product. You can join the product table to itself using the related product ID column to return a list of products and their related products.
* When you have a table of events, and each event has a related event. You can join the event table to itself using the related event ID column to return a list of events and their related events.

In general, Self-Join is useful when you want to join a table to itself based on one or more columns. It can be used to retrieve related information within the same table.

1. **In Which kind of Scenarios do we use Cross Join?**

Spark SQL Cross Join, also known as Cartesian Join, is used when you want to combine all possible rows from two or more tables. Specifically, it returns a result set where each row from the first table is combined with all the rows from the second table, and so on for each additional table.

Here are some scenarios where you might use Spark SQL Cross Join:

* When you have a table of products and a table of customers, and you want to find all possible combinations of products and customers. You would use a cross join to return a result set containing all possible combinations.
* When you have a table of employees and a table of departments, and you want to find all possible combinations of employees and departments. You would use a cross join to return a result set containing all possible combinations.
* When you have a table of locations and a table of dates, and you want to find all possible combinations of locations and dates. You would use a cross join to return a result set containing all possible combinations.

In general, Cross Join is useful when you want to combine all possible rows from two or more tables. It can be used to generate all possible combinations of data, but it can also be very computationally expensive when used with large tables. Therefore, it should be used with caution and only when necessary.

**Source Code in Scala:**

package org.sage.bigdata  
import org.apache.spark.sql.SparkSession  
import org.apache.spark.sql.functions.{array, *col*, *column*, *expr*}  
object SSV {  
 def main(args: Array[String]): Unit = {  
 *println*("hello world")  
  
 val spark = SparkSession.*builder*()  
 .master("local[\*]")  
 .appName("SparkByExamples.com")  
 .getOrCreate()  
 *println*(spark.sparkContext.appName)  
 *println*(spark.sparkContext.applicationId)  
  
 import spark.*sqlContext*.implicits.\_  
  
 val emp = Seq((1, "Smith", -1, "2018", "10", "M", 3000),  
 (2, "Rose", 1, "2010", "20", "M", 4000),  
 (3, "Williams", 1, "2010", "10", "M", 1000),  
 (4, "Jones", 2, "2005", "10", "F", 2000),  
 (5, "Brown", 2, "2010", "40", "", -1),  
 (6, "Brown", 2, "2010", "50", "", -1)  
 )  
 val empColumns = Seq("emp\_id", "name", "superior\_emp\_id", "year\_joined", "emp\_dept\_id", "gender", "salary")  
 val empDF = emp.toDF(empColumns: \_\*)  
 empDF.show(false)  
  
  
 val dept = Seq(("Finance", 10),  
 ("Marketing", 20),  
 ("Sales", 30),  
 ("IT", 40)  
 )  
  
 val deptColumns = Seq("dept\_name", "dept\_id")  
 val deptDF = dept.toDF(deptColumns: \_\*)  
 deptDF.show(false)  
  
 *println*("Inner join")  
 empDF.join(deptDF, empDF("emp\_dept\_id") === deptDF("dept\_id"), "inner").show(false)  
  
 *println*("Outer join")  
 empDF.join(deptDF, empDF("emp\_dept\_id") === deptDF("dept\_id"), "outer").show(false)  
  
 *println*("right join")  
 empDF.join(deptDF, empDF("emp\_dept\_id") === deptDF("dept\_id"), "right").show(false)  
  
 *println*("left join")  
 empDF.join(deptDF, empDF("emp\_dept\_id") === deptDF("dept\_id"), "left").show(false)  
  
 *println*("leftanti join")  
 empDF.join(deptDF, empDF("emp\_dept\_id") === deptDF("dept\_id"), "leftanti").show(false)  
  
 *println*("leftsemi join")  
 empDF.join(deptDF, empDF("emp\_dept\_id") === deptDF("dept\_id"), "leftsemi").show(false)  
  
 *println*("cross join")  
 empDF.join(deptDF, empDF("emp\_dept\_id") === deptDF("dept\_id"), "cross").show(false)  
  
 *println*("self join")  
 empDF.as("emp1").join(empDF.as("emp2"), *col*("emp1.superior\_emp\_id") === *col*("emp2.emp\_id"), "inner")  
 .select(*col*("emp1.emp\_id"), *col*("emp1.name"),  
 *col*("emp2.emp\_id").as("superior\_emp\_id"),  
 *col*("emp2.name").as("superior\_emp\_name"))  
 .show(false)  
 spark.stop()  
 }  
}