

CSC 735 – Data Analytics

Chapter 8

Joins

Joins

- Allow us to bring together the data from multiple datasets
- This allows us to analyze the combined dataset in ways that we couldn't with just each individual dataset
 - Customers table
 - Transactions table
- Every join must have a join expression (condition) and a join type

Join Expressions

- The join expression is used to decide which rows from the left dataset join with which rows in the right dataset

Join Types

- A join type determines **what** should be in the result of the join
- Join types include: inner joins, outer joins, ...

Inner Joins

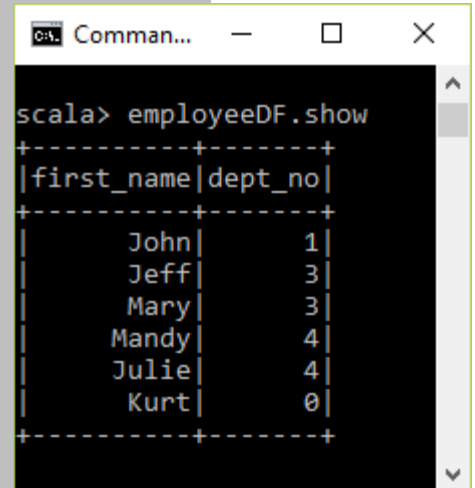
- The most commonly used join type with the join expression containing the **equality comparison**
- The result will contain the rows for which the join expression evaluates to true
- Inner join is the default join type

Creating the Datasets

```
case class Employee(first_name:String, dept_no:Long)
```

```
val employeeDF = Seq( Employee("John", 1),  
                        Employee("Jeff", 3),  
                        Employee("Mary", 3),  
                        Employee("Mandy", 4),  
                        Employee("Julie", 4),  
                        Employee("Kurt", null.asInstanceOf[Int])  
                        ).toDF
```

```
employeeDF.createOrReplaceTempView("employees")
```



```
scala> employeeDF.show  
+-----+-----+  
|first_name|dept_no|  
+-----+-----+  
|John|1|  
|Jeff|3|  
|Mary|3|  
|Mandy|4|  
|Julie|4|  
|Kurt|0|  
+-----+-----+
```

Creating the Datasets

```
case class Dept(id:Long, name:String)
```

```
val deptDF = Seq( Dept(1, "Sales"),  
                  Dept(3, "Engineering"),  
                  Dept(4, "Finance"),  
                  Dept(5, "Marketing")  
                ).toDF
```

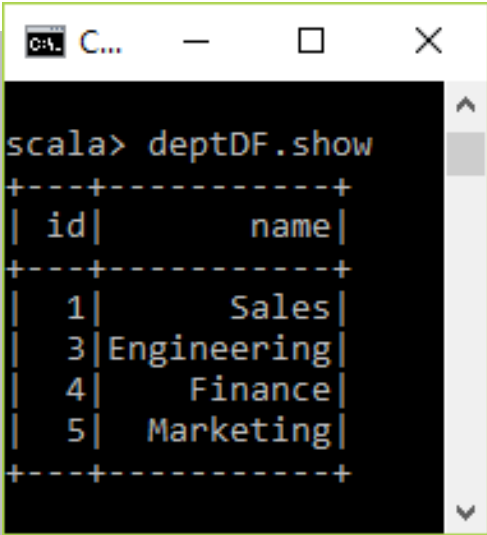
```
deptDF.createOrReplaceTempView("departments")
```

Creating the Datasets

```
case class Dept(id:Long, name:String)
```

```
val deptDF = Seq( Dept(1, "Sales"),  
                  Dept(3, "Engineering"),  
                  Dept(4, "Finance"),  
                  Dept(5, "Marketing")  
                ).toDF
```

```
deptDF.createOrReplaceTempView("departments")
```

A screenshot of a Scala REPL window. The window title is "C:\...". The command "scala> deptDF.show" has been entered. The output is a table with two columns: "id" and "name". The table contains four rows of data: (1, Sales), (3, Engineering), (4, Finance), and (5, Marketing).

id	name
1	Sales
3	Engineering
4	Finance
5	Marketing

Example

```
scala> employeeDF.show
+-----+-----+
|first_name|dept_no|
+-----+-----+
|      John|      1|
|      Jeff|      3|
|      Mary|      3|
|     Mandy|      4|
|     Julie|      4|
|      Kurt|      0|
+-----+-----+
```

```
scala> deptDF.show
+-----+-----+
| id|      name|
+-----+-----+
|  1|     Sales|
|  3|Engineering|
|  4|     Finance|
|  5|Marketing|
+-----+-----+
```

```
scala> employeeDF.join(deptDF, joinExpression).show
+-----+-----+-----+-----+
|first_name|dept_no| id|      name|
+-----+-----+-----+-----+
|      John|      1|  1|     Sales|
|      Jeff|      3|  3|Engineering|
|      Mary|      3|  3|Engineering|
|     Mandy|      4|  4|     Finance|
|     Julie|      4|  4|     Finance|
+-----+-----+-----+-----+
```

Inner Joins

```
val joinExpression = employeeDF.col("dept_no") === deptDF.col("id")
```

Inner Joins

```
val joinExpression = employeeDF.col("dept_no") === deptDF.col("id")  
  
employeeDF.join(deptDF, joinExpression, "inner").show  
  
//employeeDF.join(deptDF, joinExpression).show  
  
//spark.sql("""select * from employees JOIN departments  
on dept_no == id""").show
```

Inner Joins

```
val joinExpression = employeeDF.col("dept_no") === deptDF.col("id")
```

```
employeeDF.join(deptDF, joinExpression, "inner").show
```

```
employeeDF.join(deptDF, joinExpression).show
```

```
spark.sql("""select * from employees JOIN departments  
on dept_no == id""").show
```

```
scala> employeeDF.show
```

first_name	dept_no
John	1
Jeff	3
Mary	3
Mandy	4
Julie	4
Kurt	0

```
scala> deptDF.show
```

id	name
1	Sales
3	Engineering
4	Finance
5	Marketing

```
scala> employeeDF.join(deptDF, joinExpression).show
```

first_name	dept_no	id	name
John	1	1	Sales
Jeff	3	3	Engineering
Mary	3	3	Engineering
Mandy	4	4	Finance
Julie	4	4	Finance

Inner Joins (cont.)

- The join expression can be specified inside the join transformation

```
employeeDF.join(deptDF, 'dept_no === 'id').show
```

Inner Joins (cont.)

- The join expression can be specified inside the join transformation

```
employeeDF.join(deptDF, 'dept_no === 'id').show
```

- The join expression can be specified using the where transformation

```
employeeDF.join(deptDF).where('dept_no === 'id').show
```

Inner Joins (cont.)

- The join expression can be specified inside the join transformation

```
employeeDF.join(deptDF, 'dept_no === 'id').show
```

- The join expression can be specified using the where transformation

```
employeeDF.join(deptDF).where('dept_no === 'id').show
```

- If the column names are not unique, we need to specify which DF a particular column comes from

```
employeeDF.join(deptDF, employeeDF.col("dept_no") === deptDF.col("id")).show
```

Left Outer Joins

- The result includes the rows from the inner join plus the nonmatching rows from the left dataset

Left Outer Joins

- The result includes the rows from the inner join plus the nonmatching rows from the left dataset
- For nonmatching rows, Spark fills in null for the columns of the right dataset

Left Outer Joins

- The result includes the rows from the inner join plus the nonmatching rows from the left dataset
- For nonmatching rows, Spark fills in null for the columns of the right dataset

```
Command Prompt - spark-shell

scala> employeeDF.join(deptDF, 'dept_no === 'id, "left_outer").show
+-----+-----+-----+-----+
|first_name|dept_no|id|name|
+-----+-----+-----+-----+
|John|1|1|Sales|
|Jeff|3|3|Engineering|
|Mary|3|3|Engineering|
|Mandy|4|4|Finance|
|Julie|4|4|Finance|
|Kurt|0|null|null|
+-----+-----+-----+-----+
```

Left Outer Joins

- The result includes the rows from the inner join plus the nonmatching rows from the left dataset
- For nonmatching rows, Spark fills in null for the columns of the right dataset

```
Command Prompt - spark-shell

scala> employeeDF.join(deptDF, 'dept_no === 'id, "left_outer").show
+-----+-----+-----+-----+
|first_name|dept_no|id|name|
+-----+-----+-----+-----+
|John|1|1|Sales|
|Jeff|3|3|Engineering|
|Mary|3|3|Engineering|
|Mandy|4|4|Finance|
|Julie|4|4|Finance|
|Kurt|0|null|null|
+-----+-----+-----+-----+
```

```
Command Prompt - spark-shell

scala> employeeDF.show
+-----+-----+
|first_name|dept_no|
+-----+-----+
|John|1|
|Jeff|3|
|Mary|3|
|Mandy|4|
|Julie|4|
|Kurt|0|
+-----+-----+
```

```
Command Prompt - spark-shell

scala> deptDF.show
+-----+-----+
|id|name|
+-----+-----+
|1|Sales|
|3|Engineering|
|4|Finance|
|5|Marketing|
+-----+-----+
```

Left Outer Joins – SQL Way

```
Command Prompt - spark-shell

scala> spark.sql("select * from employees LEFT OUTER JOIN departments on dept_no == id").show
+-----+-----+-----+-----+
|first_name|dept_no| id|      name|
+-----+-----+-----+-----+
|      John|      1|  1|      Sales|
|      Jeff|      3|  3| Engineering|
|      Mary|      3|  3| Engineering|
|     Mandy|      4|  4|    Finance|
|     Julie|      4|  4|    Finance|
|      Kurt|      0| null|         null|
+-----+-----+-----+-----+
```

Right Outer Joins

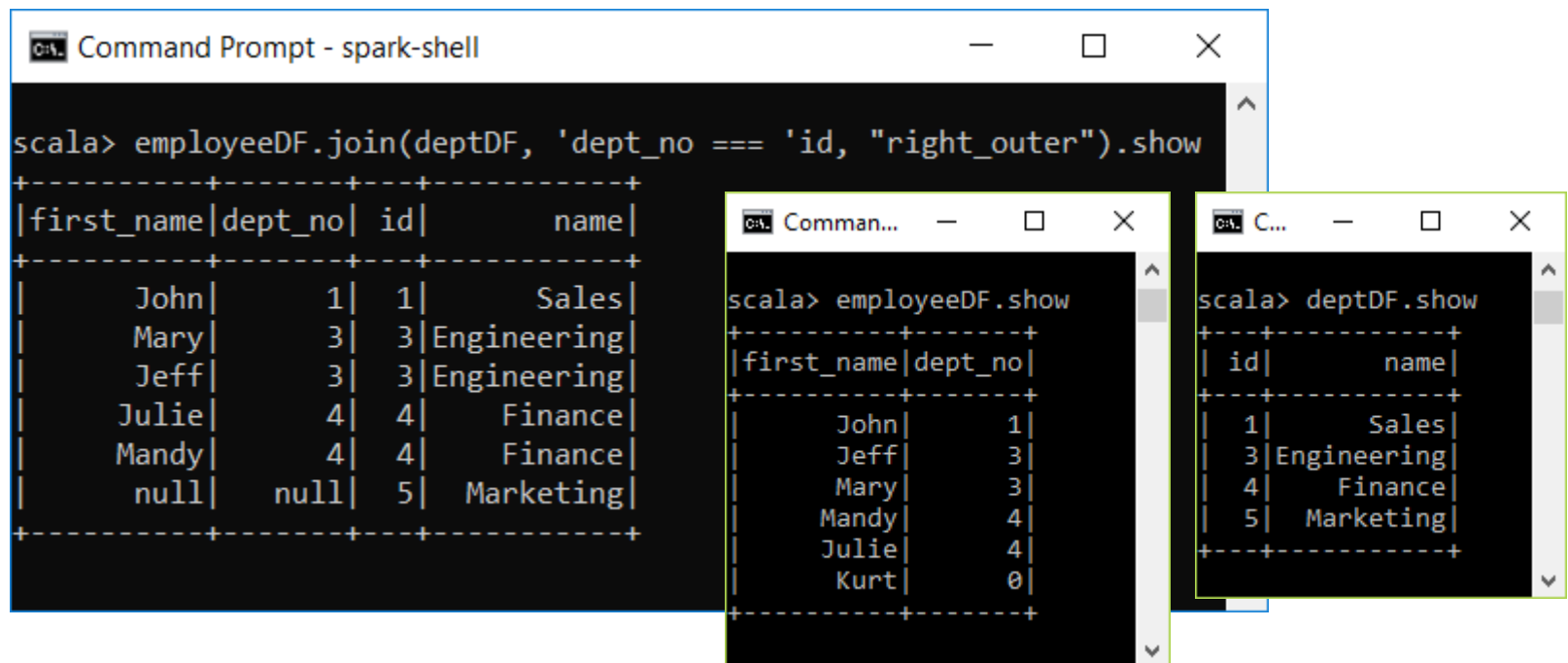
- Analogous to left outer join type
- Result contains matching rows + nonmatching rows from the right dataset

```
Command Prompt - spark-shell

scala> employeeDF.join(deptDF, 'dept_no === 'id, "right_outer").show
+-----+-----+-----+-----+
|first_name|dept_no| id|      name|
+-----+-----+-----+-----+
|      John|      1|  1|      Sales|
|      Mary|      3|  3|Engineering|
|      Jeff|      3|  3|Engineering|
|      Julie|      4|  4|   Finance|
|      Mandy|      4|  4|   Finance|
|      null|    null|  5|Marketing|
+-----+-----+-----+-----+
```

Right Outer Joins

- Analogous to left outer join type
- Result contains matching rows + nonmatching rows from the right dataset



The image shows three overlapping terminal windows from a Spark-shell environment. The largest window on the left displays the command `scala> employeeDF.join(deptDF, 'dept_no === 'id, "right_outer").show` and its output, which is a table with columns `first_name`, `dept_no`, `id`, and `name`. The output shows rows for John, Mary, Jeff, Julie, and Mandy, plus a row with `null` for `dept_no` and `id`, and `Marketing` for `name`. The middle window shows the command `scala> employeeDF.show` and its output, a table with columns `first_name` and `dept_no`, showing rows for John, Jeff, Mary, Mandy, Julie, and Kurt. The rightmost window shows the command `scala> deptDF.show` and its output, a table with columns `id` and `name`, showing rows for 1 (Sales), 3 (Engineering), 4 (Finance), and 5 (Marketing).

```
scala> employeeDF.join(deptDF, 'dept_no === 'id, "right_outer").show
+-----+-----+-----+-----+
|first_name|dept_no| id|      name|
+-----+-----+-----+-----+
|      John|      1|  1|      Sales|
|      Mary|      3|  3|Engineering|
|      Jeff|      3|  3|Engineering|
|      Julie|      4|  4|      Finance|
|      Mandy|      4|  4|      Finance|
|      null|    null|  5|Marketing|
+-----+-----+-----+-----+
```

```
scala> employeeDF.show
+-----+-----+
|first_name|dept_no|
+-----+-----+
|      John|      1|
|      Jeff|      3|
|      Mary|      3|
|      Mandy|      4|
|      Julie|      4|
|      Kurt|      0|
+-----+-----+
```

```
scala> deptDF.show
+-----+-----+
| id|      name|
+-----+-----+
|  1|      Sales|
|  3|Engineering|
|  4|      Finance|
|  5|Marketing|
+-----+-----+
```

Right Outer Joins – SQL Way

```
Command Prompt - spark-shell

scala> spark.sql("select * from employees RIGHT OUTER JOIN departments on dept_no == id").show
+-----+-----+-----+-----+
|first_name|dept_no| id|      name|
+-----+-----+-----+-----+
|      John|      1|  1|      Sales|
|      Mary|      3|  3|Engineering|
|      Jeff|      3|  3|Engineering|
|      Julie|      4|  4|   Finance|
|      Mandy|      4|  4|   Finance|
|      null|   null|  5|Marketing|
+-----+-----+-----+-----+
```

Outer Joins (aka Full Outer Joins)

- Combines both the left outer join and the right outer join

```
employeeDF.join(deptDF, 'dept_no === 'id, "outer").show
spark.sql("""select *
            from employees FULL OUTER JOIN departments
            on dept_no == id""").show
```


Joins - Databricks Commu x

Secure | https://community.cloud.databricks.com/?o=8616461472726830#notebook/253770... ☆

databricks

Home

Workspace

Recents

Data

Clusters

Jobs

Search

Joins (Scala)

J ⓘ ?

Attached: cluster 3

⌨️ ↻ 🗨️ 🔍

```
employeeDF.join(deptDF, 'dept_no == 'id, "outer").show
```

▶ (5) Spark Jobs

first_name	dept_no	id	name
Kurt	0	null	null
null	null	5	Marketing
John	1	1	Sales
Jeff	3	3	Engineering
Mary	3	3	Engineering
Mandy	4	4	Finance
Julie	4	4	Finance

Command took 2.16 seconds -- by jamilsaquer@missouristate.edu at 10/22/2018, 10:35:26 PM on cluster 3

Cmd 18

```
spark.sql("select * from employees FULL OUTER JOIN departments on dept_no == id").show
```

▶ (5) Spark Jobs

first_name	dept_no	id	name
Kurt	0	null	null
null	null	5	Marketing
John	1	1	Sales
Jeff	3	3	Engineering
Mary	3	3	Engineering
Mandy	4	4	Finance
Julie	4	4	Finance

Send Feedback

Left Semi Joins

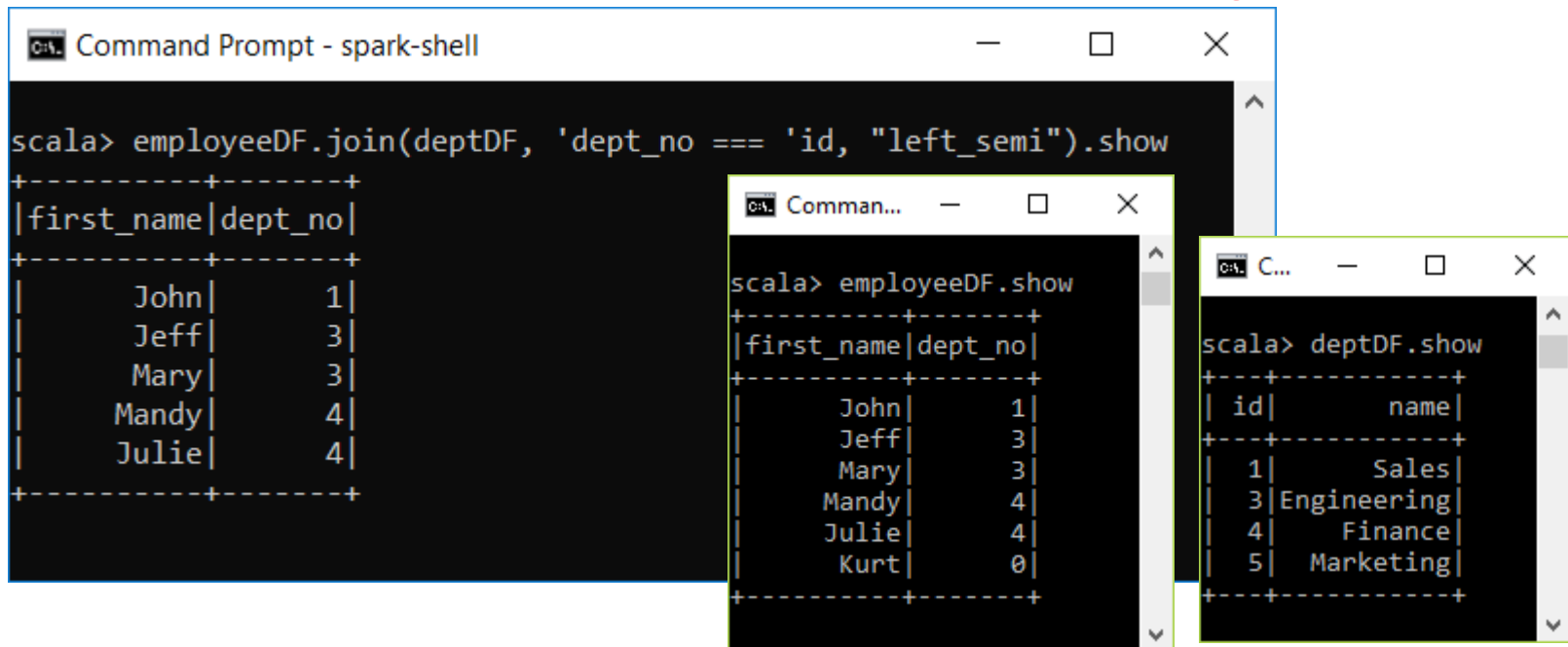
- It is similar to the inner join but contains rows only from the left dataset
- I.e., the result dataset contains only the rows in the left dataset that have matching rows in the other dataset

Left Semi Joins - Example

```
Command Prompt - spark-shell

scala> employeeDF.join(deptDF, 'dept_no === 'id, "left_semi").show
+-----+-----+
|first_name|dept_no|
+-----+-----+
|      John|      1|
|      Jeff|      3|
|      Mary|      3|
|     Mandy|      4|
|     Julie|      4|
+-----+-----+
```

Left Semi Joins - Example



The image shows three overlapping terminal windows from a 'spark-shell' environment. The largest window on the left shows the execution of a left semi join between 'employeeDF' and 'deptDF'. The two smaller windows on the right show the individual data for 'employeeDF' and 'deptDF' before the join operation.

```
scala> employeeDF.join(deptDF, 'dept_no === 'id, "left_semi").show
```

first_name	dept_no
John	1
Jeff	3
Mary	3
Mandy	4
Julie	4

```
scala> employeeDF.show
```

first_name	dept_no
John	1
Jeff	3
Mary	3
Mandy	4
Julie	4
Kurt	0

```
scala> deptDF.show
```

id	name
1	Sales
3	Engineering
4	Finance
5	Marketing

Left Semi Joins - Example

The image displays three overlapping screenshots of a Spark shell (Command Prompt - spark-shell) showing SQL queries and their results.

Top Left Screenshot: Shows the execution of a left semi join query.

```
scala> employeeDF.join(deptDF, 'dept_no === 'id, "left_semi").show
```

first_name	dept_no
John	1
Jeff	3
Mary	3
Mandy	4
Julie	4

Top Middle Screenshot: Shows the execution of a query to display the employee DataFrame.

```
scala> employeeDF.show
```

first_name	dept_no
John	1
Jeff	3
Mary	3
Mandy	4
Julie	4
Kurt	0

Top Right Screenshot: Shows the execution of a query to display the department DataFrame.

```
scala> deptDF.show
```

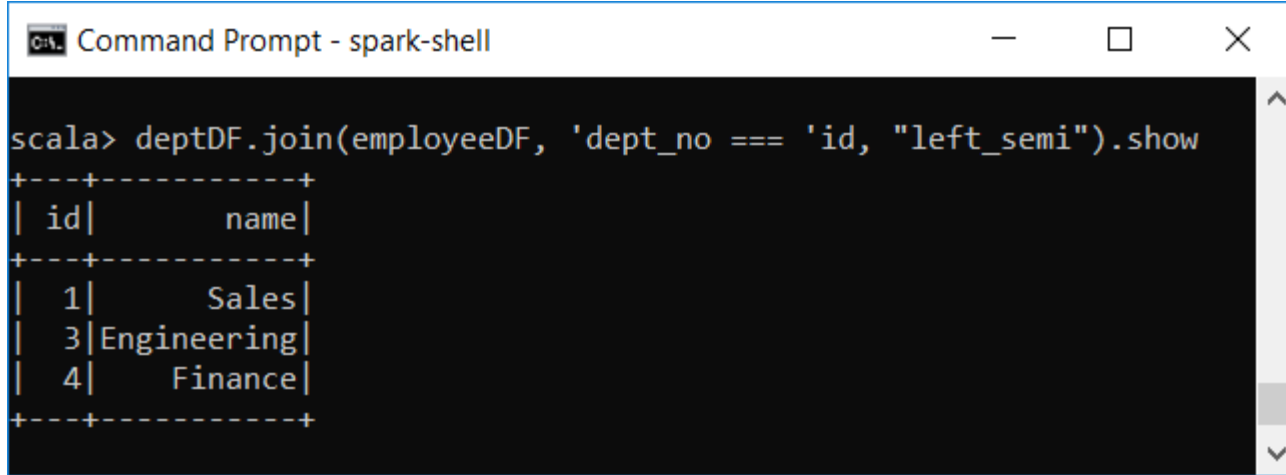
id	name
1	Sales
3	Engineering
4	Finance
5	Marketing

Bottom Screenshot: Shows the execution of an inner join query.

```
scala> employeeDF.join(deptDF, 'dept_no === 'id, "inner").show
```

first_name	dept_no	id	name
John	1	1	Sales
Jeff	3	3	Engineering
Mary	3	3	Engineering
Mandy	4	4	Finance
Julie	4	4	Finance

Left Semi Joins - Example

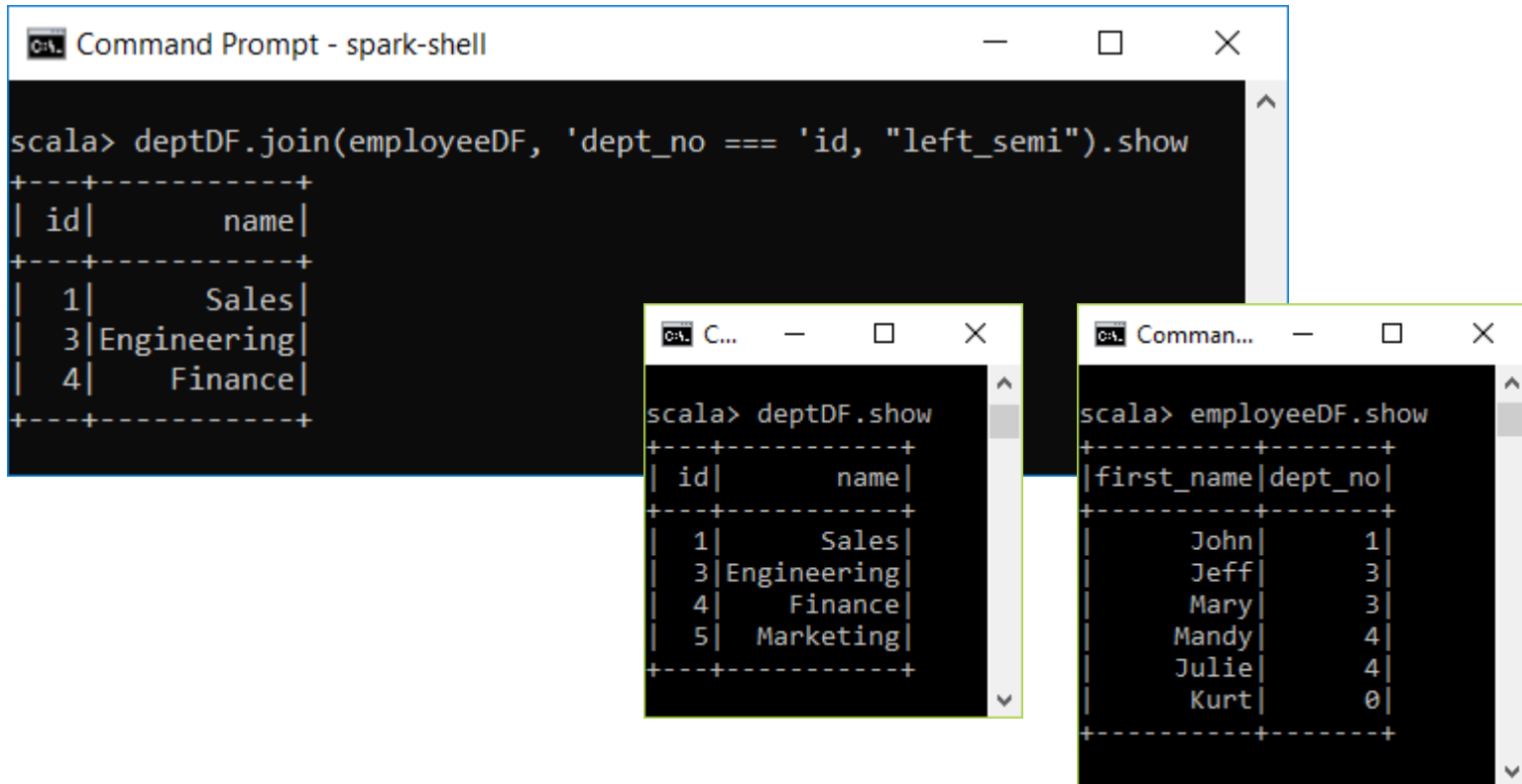


```
scala> deptDF.join(employeeDF, 'dept_no === 'id, "left_semi").show
```

id	name
1	Sales
3	Engineering
4	Finance

The screenshot shows a terminal window titled "Command Prompt - spark-shell". It contains the command `scala> deptDF.join(employeeDF, 'dept_no === 'id, "left_semi").show`. Below the command, the output is displayed as a table with two columns: `id` and `name`. The table contains three rows of data: `(1, Sales)`, `(3, Engineering)`, and `(4, Finance)`. The table is formatted with dashed lines for the header and body, and a vertical line separating the columns.

Left Semi Joins - Example



The image displays three terminal windows from a Spark-shell environment, illustrating a left semi join operation. The top window shows the execution of a join command and its result. The bottom-left window shows the contents of the 'deptDF' table. The bottom-right window shows the contents of the 'employeeDF' table.

```
scala> deptDF.join(employeeDF, 'dept_no === 'id, "left_semi").show
```

id	name
1	Sales
3	Engineering
4	Finance

```
scala> deptDF.show
```

id	name
1	Sales
3	Engineering
4	Finance
5	Marketing

```
scala> employeeDF.show
```

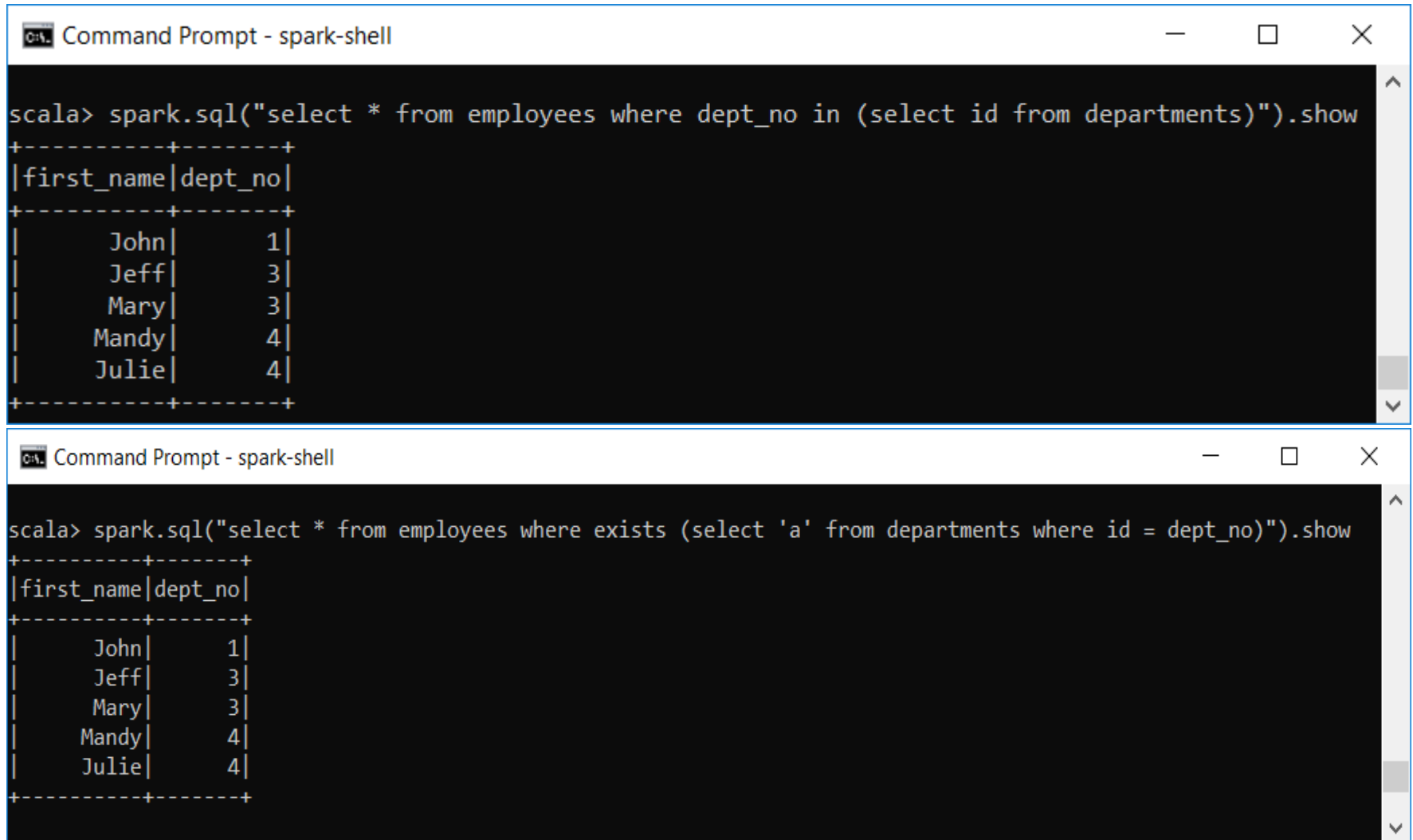
first_name	dept_no
John	1
Jeff	3
Mary	3
Mandy	4
Julie	4
Kurt	0

Left Semi Joins – SQL Way

```
Command Prompt - spark-shell
scala> spark.sql("select * from employees LEFT SEMI JOIN departments on dept_no == id").show
+-----+-----+
|first_name|dept_no|
+-----+-----+
|      John|      1|
|      Jeff|      3|
|      Mary|      3|
|     Mandy|      4|
|     Julie|      4|
+-----+-----+
```


Left Semi Joins – Equivalent SQL Commands

- You can achieve left semi join with IN or EXISTS in SQL



The image displays two screenshots of a 'Command Prompt - spark-shell' window, demonstrating SQL queries for left semi joins.

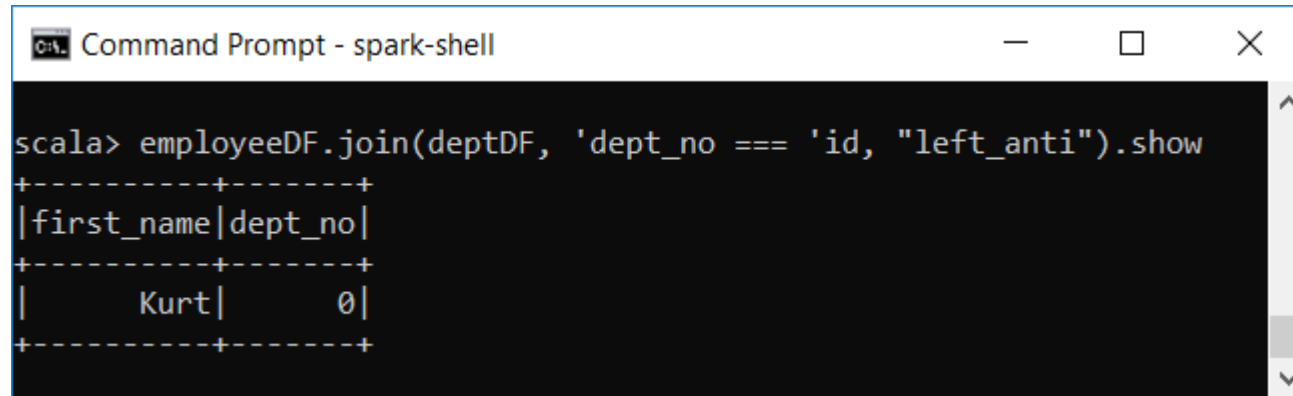
Top Screenshot: The query uses the `IN` clause: `scala> spark.sql("select * from employees where dept_no in (select id from departments)").show`. The output shows a table with columns `first_name` and `dept_no`, containing rows for John (dept 1), Jeff (dept 3), Mary (dept 3), Mandy (dept 4), and Julie (dept 4).

Bottom Screenshot: The query uses the `EXISTS` clause: `scala> spark.sql("select * from employees where exists (select 'a' from departments where id = dept_no)").show`. The output is identical to the top screenshot, showing the same five employees and their department numbers.

first_name	dept_no
John	1
Jeff	3
Mary	3
Mandy	4
Julie	4

Left Anti Joins

- Left anti join results in rows from only the left dataset if, and only if, there is **NO matching** row in right dataset



```
Command Prompt - spark-shell

scala> employeeDF.join(deptDF, 'dept_no === 'id, "left_anti").show
+-----+-----+
|first_name|dept_no|
+-----+-----+
|      Kurt|      0|
+-----+-----+
```

Left Anti Joins

- Left anti join results in rows from only the left dataset if, and only if, there is NO matching row in right dataset

```
Command Prompt - spark-shell

scala> employeeDF.join(deptDF, 'dept_no === 'id, "left_anti").show
+-----+-----+
|first_name|dept_no|
+-----+-----+
|      Kurt|      0|
+-----+-----+
```

```
Command Prompt - spark-shell

scala> employeeDF.show
+-----+-----+
|first_name|dept_no|
+-----+-----+
|      John|      1|
|      Jeff|      3|
|      Mary|      3|
|     Mandy|      4|
|     Julie|      4|
|      Kurt|      0|
+-----+-----+
```

```
Command Prompt - spark-shell

scala> deptDF.show
+-----+-----+
| id|      name|
+-----+-----+
|  1|     Sales|
|  3|Engineering|
|  4|     Finance|
|  5|   Marketing|
+-----+-----+
```

Left Anti Joins

- Left anti join results in rows from only the left dataset if, and only if, there is NO matching row in right dataset

```
Command Prompt - spark-shell

scala> employeeDF.join(deptDF, 'dept_no === 'id, "left_anti").show
+-----+-----+
|first_name|dept_no|
+-----+-----+
|      Kurt|      0|
+-----+-----+
```

```
Command Prompt - spark-shell

scala> employeeDF.show
+-----+-----+
|first_name|dept_no|
+-----+-----+
|      John|      1|
|      Jeff|      3|
|      Mary|      3|
|     Mandy|      4|
|     Julie|      4|
|      Kurt|      0|
+-----+-----+
```

```
Command Prompt - spark-shell

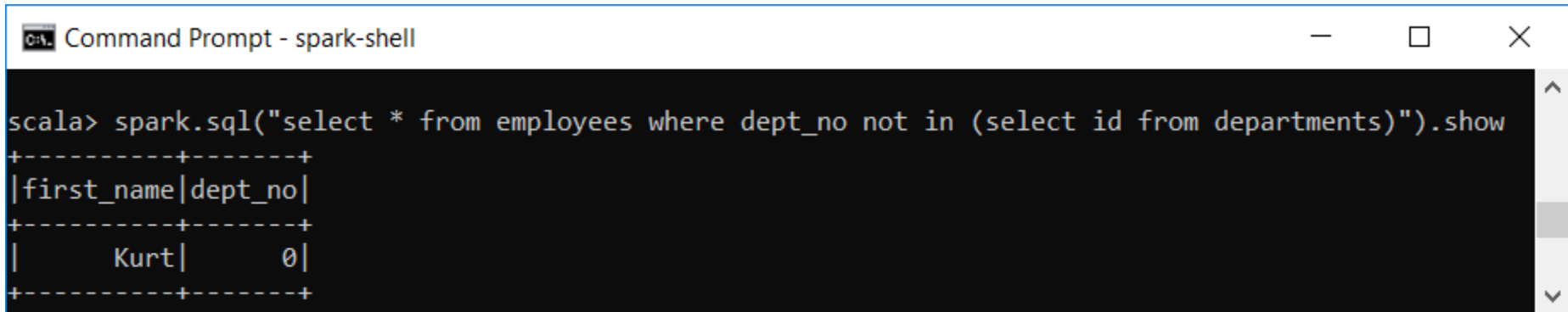
scala> deptDF.join(employeeDF, 'dept_no === 'id, "left_anti").show
+-----+-----+
| id|      name|
+-----+-----+
|  5|Marketing|
+-----+-----+
```

```
Command Prompt - spark-shell

scala> deptDF.show
+-----+-----+
| id|      name|
+-----+-----+
|  1|    Sales|
|  3|Engineering|
|  4|    Finance|
|  5|   Marketing|
+-----+-----+
```

Left Anti Joins – Equivalent SQL Commands

- You can think of anti joins as a NOT IN or NOT EXISTS SQL style filter



```
Command Prompt - spark-shell

scala> spark.sql("select * from employees where dept_no not in (select id from departments)").show
+-----+-----+
|first_name|dept_no|
+-----+-----+
|      Kurt|      0|
+-----+-----+
```

Left Anti Joins – Equivalent SQL Commands

- You can think of anti joins as a NOT IN or NOT EXISTS SQL style filter

```
Command Prompt - spark-shell

scala> spark.sql("select * from employees where dept_no not in (select id from departments)").show
+-----+-----+
|first_name|dept_no|
+-----+-----+
|      Kurt|      0|
+-----+-----+
```

```
Command Prompt - spark-shell

scala> spark.sql("select * from employees where not exists (select 'a' from departments where id = dept_no)").show
+-----+-----+
|first_name|dept_no|
+-----+-----+
|      Kurt|      0|
+-----+-----+
```

Cross (aka Cartesian) Joins

- Simple to use because no join expression is needed
- Every single row in the left dataset will join with every single row in the right dataset
- Size of the result dataset can be huge
- Better to specify as a transformation

```
employeeDF.crossJoin(deptDF).show
```

```
//SQL Way
```

```
spark.sql("select * from employees CROSS JOIN departments").show()
```

Cross Joins

```
scala> employeeDF.show
```

first_name	dept_no
John	1
Jeff	3
Mary	3
Mandy	4
Julie	4
Kurt	0

```
scala> deptDF.show
```

id	name
1	Sales
3	Engineering
4	Finance
5	Marketing

Cross Joins

```
Command Prompt - spark-shell

scala> employeeDF.crossJoin(deptDF).show(25)
+-----+-----+-----+
|first_name|dept_no| id|      name|
+-----+-----+-----+
|      John|      1|  1|      Sales|
|      John|      1|  3|Engineering|
|      John|      1|  4|   Finance|
|      John|      1|  5|Marketing|
|      Jeff|      3|  1|      Sales|
|      Jeff|      3|  3|Engineering|
|      Jeff|      3|  4|   Finance|
|      Jeff|      3|  5|Marketing|
|      Mary|      3|  1|      Sales|
|      Mary|      3|  3|Engineering|
|      Mary|      3|  4|   Finance|
|      Mary|      3|  5|Marketing|
|     Mandy|      4|  1|      Sales|
|     Mandy|      4|  3|Engineering|
|     Mandy|      4|  4|   Finance|
|     Mandy|      4|  5|Marketing|
|     Julie|      4|  1|      Sales|
|     Julie|      4|  3|Engineering|
|     Julie|      4|  4|   Finance|
|     Julie|      4|  5|Marketing|
|      Kurt|      0|  1|      Sales|
|      Kurt|      0|  3|Engineering|
|      Kurt|      0|  4|   Finance|
|      Kurt|      0|  5|Marketing|
+-----+-----+-----+
```

```
Command Prompt - spark-shell

scala> employeeDF.show
+-----+-----+
|first_name|dept_no|
+-----+-----+
|      John|      1|
|      Jeff|      3|
|      Mary|      3|
|     Mandy|      4|
|     Julie|      4|
|      Kurt|      0|
+-----+-----+
```

```
Command Prompt - spark-shell

scala> deptDF.show
+-----+-----+
| id|      name|
+-----+-----+
|  1|      Sales|
|  3|Engineering|
|  4|   Finance|
|  5|Marketing|
+-----+-----+
```

Handling Duplicate Column Names

- A problem with joins is that a resulting DF can have multiple columns with the same name
- If a DF has two columns with the same name, it would be difficult to refer to these columns

Handling Duplicate Column Names

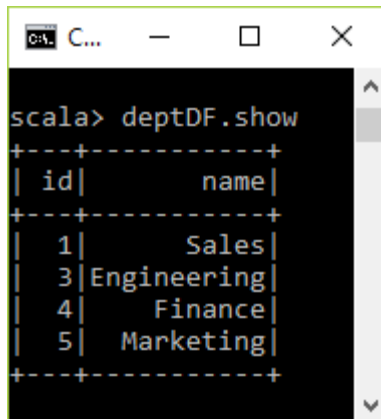
- A problem with joins is that a resulting DF can have multiple columns with the same name
- If a DF has two columns with the same name, it would be difficult to refer to these columns

```
val deptDF2 = deptDF.withColumn("dept_no", 'id)  
deptDF2.show
```

Handling Duplicate Column Names

- A problem with joins is that a resulting DF can have multiple columns with the same name
- If a DF has two columns with the same name, it would be difficult to refer to these columns

```
val deptDF2 = deptDF.withColumn("dept_no", 'id)  
deptDF2.show
```

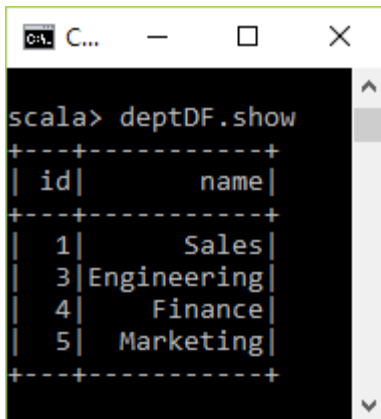


```
scala> deptDF.show  
+---+  
| id | name |  
+---+  
| 1 | Sales |  
| 3 | Engineering |  
| 4 | Finance |  
| 5 | Marketing |  
+---+
```

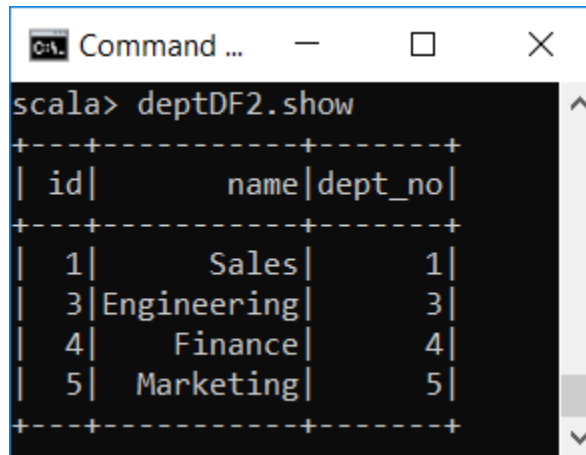
Handling Duplicate Column Names

- A problem with joins is that a resulting DF can have multiple columns with the same name
- If a DF has two columns with the same name, it would be difficult to refer to these columns

```
val deptDF2 = deptDF.withColumn("dept_no", 'id)  
deptDF2.show
```



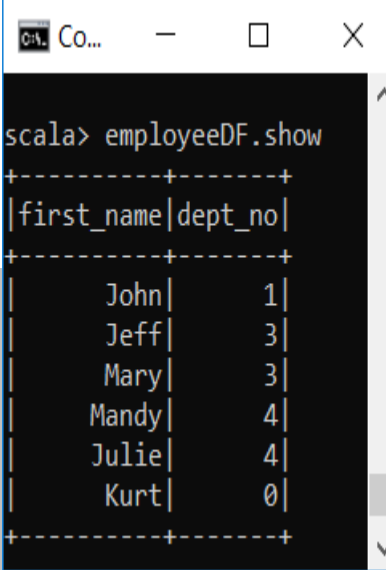
```
scala> deptDF.show  
+-----+  
| id |    name |  
+-----+  
|  1 |   Sales |  
|  3 | Engineering |  
|  4 |   Finance |  
|  5 | Marketing |  
+-----+
```



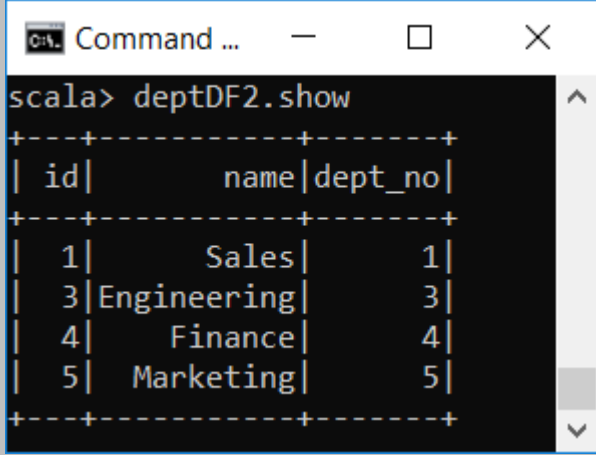
```
scala> deptDF2.show  
+-----+  
| id |    name | dept_no |  
+-----+  
|  1 |   Sales |       1 |  
|  3 | Engineering |       3 |  
|  4 |   Finance |       4 |  
|  5 | Marketing |       5 |  
+-----+
```

Handling Duplicate Column Names

```
val dupNameDF = employeeDF.join(deptDF2,  
    employeeDF.col("dept_no") === deptDF2.col("dept_no"))
```



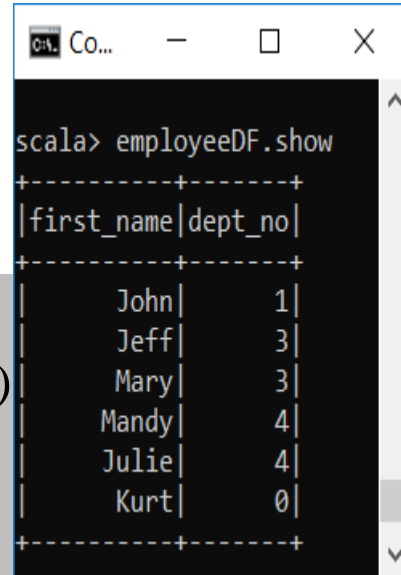
```
scala> employeeDF.show  
+-----+-----+  
|first_name|dept_no|  
+-----+-----+  
|      John|      1|  
|      Jeff|      3|  
|      Mary|      3|  
|      Mandy|     4|  
|      Julie|     4|  
|       Kurt|      0|  
+-----+-----+
```



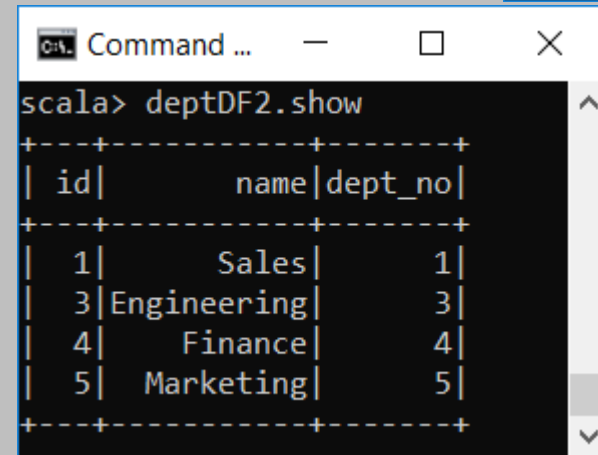
```
scala> deptDF2.show  
+---+-----+-----+  
| id|      name|dept_no|  
+---+-----+-----+  
|  1|     Sales|      1|  
|  3|Engineering|      3|  
|  4|   Finance|      4|  
|  5|Marketing|      5|  
+---+-----+-----+
```

Handling Duplicate Column Names

```
val dupNameDF = employeeDF.join(deptDF2,  
    employeeDF.col("dept_no") === deptDF2.col("dept_no"))  
dupNameDF.printSchema  
root  
|-- first_name: string (nullable = true)  
|-- dept_no: long (nullable = false)  
|-- id: long (nullable = false)  
|-- name: string (nullable = true)  
|-- dept_no: long (nullable = false)
```



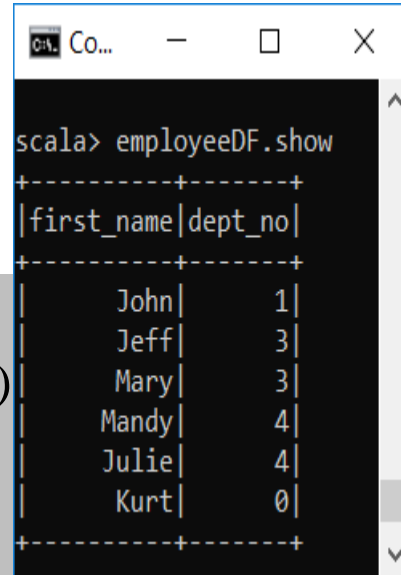
```
scala> employeeDF.show  
+-----+-----+  
|first_name|dept_no|  
+-----+-----+  
|      John|      1|  
|      Jeff|      3|  
|      Mary|      3|  
|      Mandy|      4|  
|      Julie|      4|  
|      Kurt|      0|  
+-----+-----+
```



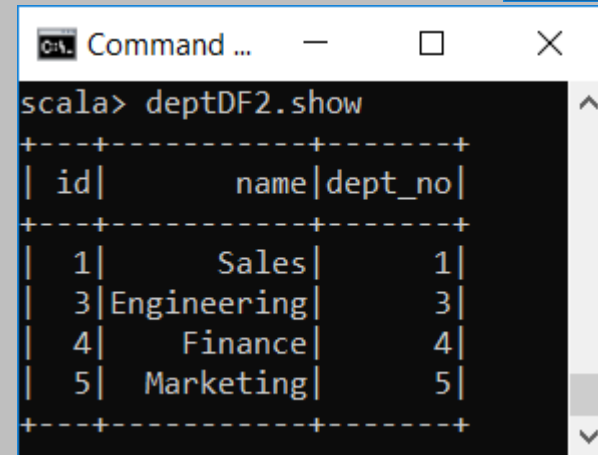
```
scala> deptDF2.show  
+---+-----+-----+  
| id|      name|dept_no|  
+---+-----+-----+  
|  1|     Sales|      1|  
|  3|Engineering|      3|  
|  4|   Finance|      4|  
|  5|  Marketing|      5|  
+---+-----+-----+
```

Handling Duplicate Column Names

```
val dupNameDF = employeeDF.join(deptDF2,  
    employeeDF.col("dept_no") === deptDF2.col("dept_no"))  
dupNameDF.printSchema  
root  
|-- first_name: string (nullable = true)  
|-- dept_no: long (nullable = false)  
|-- id: long (nullable = false)  
|-- name: string (nullable = true)  
|-- dept_no: long (nullable = false)  
  
dupNameDF.select("dept_no")
```



```
scala> employeeDF.show  
+-----+-----+  
|first_name|dept_no|  
+-----+-----+  
|      John|      1|  
|      Jeff|      3|  
|      Mary|      3|  
|      Mandy|      4|  
|      Julie|      4|  
|      Kurt|      0|  
+-----+-----+
```



```
scala> deptDF2.show  
+---+-----+-----+  
| id|      name|dept_no|  
+---+-----+-----+  
|  1|     Sales|      1|  
|  3|Engineering|      3|  
|  4|   Finance|      4|  
|  5|  Marketing|      5|  
+---+-----+-----+
```


Handling Duplicate Column Names

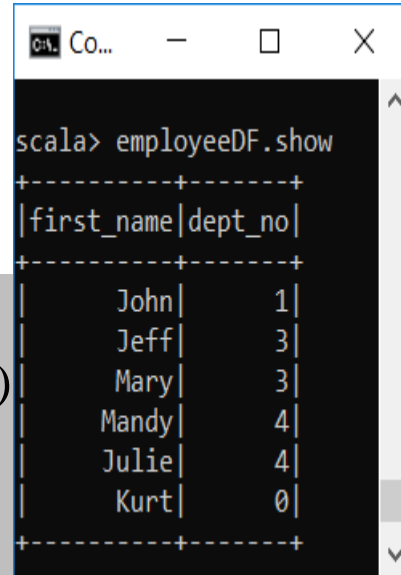
```
val dupNameDF = employeeDF.join(deptDF2,  
    employeeDF.col("dept_no") === deptDF2.col("dept_no"))  
dupNameDF.printSchema
```

root

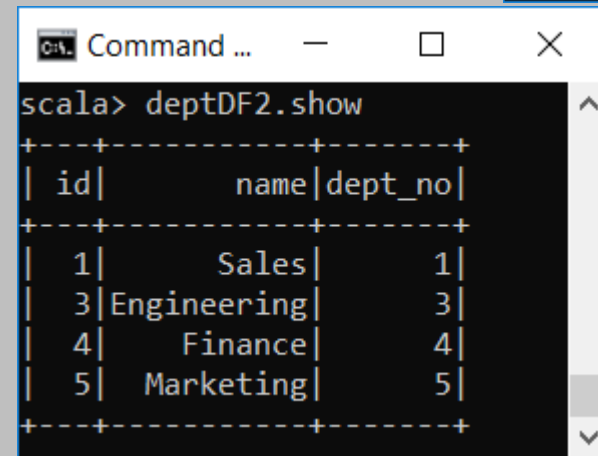
```
-- first_name: string (nullable = true)  
-- dept_no: long (nullable = false)  
-- id: long (nullable = false)  
-- name: string (nullable = true)  
-- dept_no: long (nullable = false)
```

```
dupNameDF.select("dept_no")
```

org.apache.spark.sql.AnalysisException: Reference 'dept_no' is **ambiguous**, could be: dept_no, dept_no.;



```
scala> employeeDF.show  
+-----+-----+  
|first_name|dept_no|  
+-----+-----+  
|John|1|  
|Jeff|3|  
|Mary|3|  
|Mandy|4|  
|Julie|4|  
|Kurt|0|  
+-----+-----+
```



```
scala> deptDF2.show  
+---+-----+-----+  
|id|name|dept_no|  
+---+-----+-----+  
|1|Sales|1|  
|3|Engineering|3|  
|4|Finance|4|  
|5|Marketing|5|  
+---+-----+-----+
```

Handling Duplicate Column Names

- There are several ways to deal with duplicate columns

Handling Duplicate Column Names

- There are several ways to deal with duplicate columns
 - Using a different join API
 - Specifying a column name with its original DF
 - Renaming a column before the join
 - Dropping a column after the join

1. Using a Different Join API

- These versions of join automatically remove duplicate column names

- **join(right: DF, usingColumn: String): DF**
Inner join with a single column
- **join(right: DF, usingColumns: Seq[String]): DF**
Inner join with multiple columns
- **join(right: DF, usingColumns: Seq[String], joinType: String): DF**
Join with multiple columns and a join type

Using a Different Join API – Example

```
Command Prompt - spark-shell

scala> employeeDF.join(deptDF2, "dept_no").show
+-----+-----+-----+-----+
|dept_no|first_name| id|      name|
+-----+-----+-----+-----+
|      1|      John|  1|     Sales|
|      3|      Jeff|  3|Engineering|
|      3|      Mary|  3|Engineering|
|      4|      Mandy|  4|   Finance|
|      4|      Julie|  4|   Finance|
+-----+-----+-----+-----+
```

```
Command ...

scala> deptDF2.show
+---+-----+-----+
| id|      name|dept_no|
+---+-----+-----+
|  1|     Sales|      1|
|  3|Engineering|      3|
|  4|   Finance|      4|
|  5|Marketing|      5|
+---+-----+-----+
```

```
Co...

scala> employeeDF.show
+-----+-----+
|first_name|dept_no|
+-----+-----+
|      John|      1|
|      Jeff|      3|
|      Mary|      3|
|      Mandy|     4|
|      Julie|     4|
|      Kurt|      0|
+-----+-----+
```

Using a Different Join API – Example

```
Command Prompt - spark-shell

scala> employeeDF.join(deptDF2, Seq("dept_no")).show
+-----+-----+-----+-----+
|dept_no|first_name| id|      name|
+-----+-----+-----+-----+
|      1|      John|  1|     Sales|
|      3|      Jeff|  3|Engineering|
|      3|      Mary|  3|Engineering|
|      4|     Mandy|  4|   Finance|
|      4|     Julie|  4|   Finance|
+-----+-----+-----+-----+
```

Using a Different Join API – Example

```
Command Prompt - spark-shell

scala> employeeDF.join(deptDF2, Seq("dept_no")).show
+-----+-----+-----+-----+
|dept_no|first_name| id|      name|
+-----+-----+-----+-----+
|      1|      John|  1|     Sales|
|      3|      Jeff|  3|Engineering|
|      3|      Mary|  3|Engineering|
|      4|     Mandy|  4|   Finance|
|      4|     Julie|  4|   Finance|
+-----+-----+-----+-----+
```

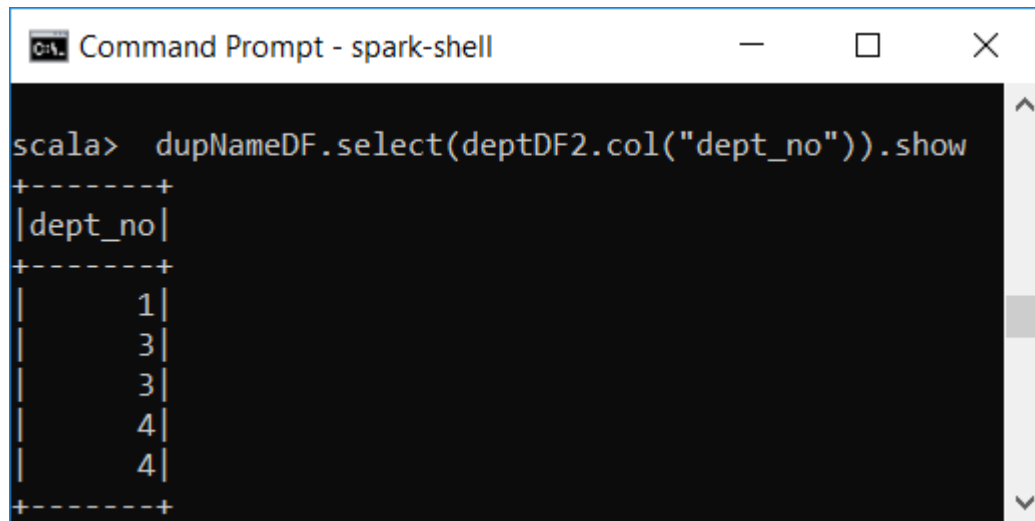
```
Command Prompt - spark-shell

scala> employeeDF.join(deptDF2, Seq("dept_no"), "left_semi").show
+-----+-----+
|dept_no|first_name|
+-----+-----+
|      1|      John|
|      3|      Jeff|
|      3|      Mary|
|      4|     Mandy|
|      4|     Julie|
+-----+-----+
```

2. Specifying a Column Name with its Original DF

- The joined DF remembers which columns come from which original DF during the joining process

```
dupNameDF.select(deptDF2.col("dept_no"))
```



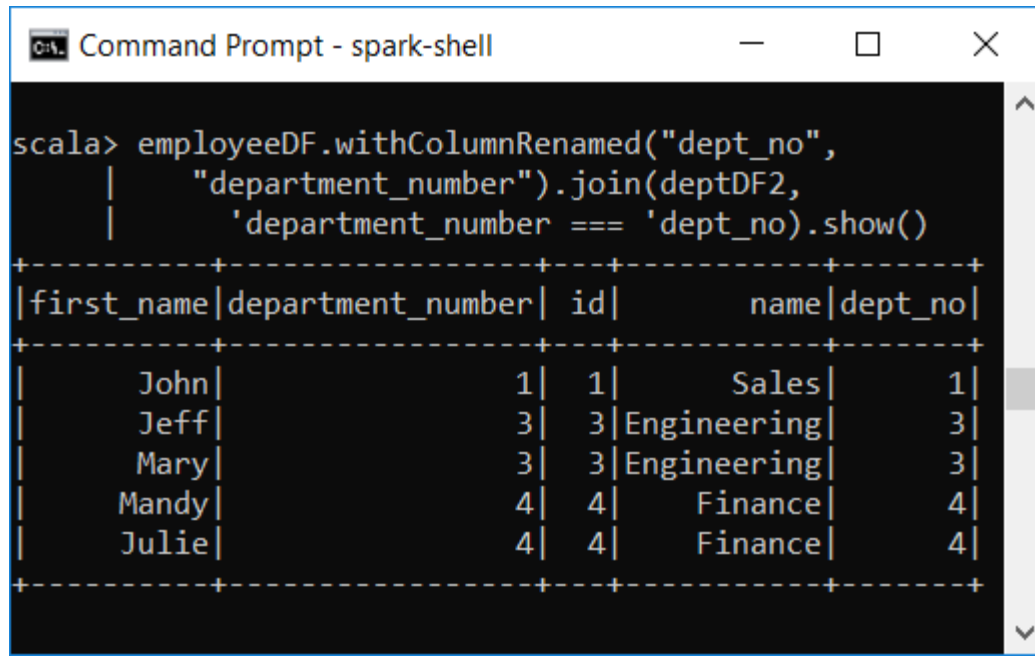
```
Command Prompt - spark-shell

scala> dupNameDF.select(deptDF2.col("dept_no")).show
+-----+
|dept_no|
+-----+
|      1|
|      3|
|      3|
|      4|
|      4|
+-----+
```


3. Renaming a Column before the Join

- Use the **withColumnRenamed** transformation before doing the join

```
employeeDF.withColumnRenamed("dept_no",  
    "department_number")  
    .join(deptDF2, 'department_number === 'dept_no).show()
```



```
Command Prompt - spark-shell

scala> employeeDF.withColumnRenamed("dept_no",
    |         "department_number").join(deptDF2,
    |         'department_number === 'dept_no).show()
+-----+-----+-----+-----+-----+
|first_name|department_number|id|name|dept_no|
+-----+-----+-----+-----+-----+
|John|1|1|Sales|1|
|Jeff|3|3|Engineering|3|
|Mary|3|3|Engineering|3|
|Mandy|4|4|Finance|4|
|Julie|4|4|Finance|4|
+-----+-----+-----+-----+-----+
```

4. Dropping a Column after the Join

- We need to refer to the column via the original source DF

```
val dupNameDF = employeeDF.join(deptDF2,  
    employeeDF.col("dept_no") === deptDF2.col("dept_no"))  
  
dupNameDF.drop(deptDF2.col("dept_no")).select("dept_no").show  
W
```

```
scala> dupNameDF.drop(  
    | deptDF2.col("dept_no")  
    | ).select("dept_no").show  
+-----+  
|dept_no|  
+-----+  
|      1|  
|      3|  
|      3|  
|      4|  
|      4|  
+-----+
```

```
scala> dupNameDF.show  
+-----+-----+-----+-----+-----+  
|first_name|dept_no| id|      name|dept_no|  
+-----+-----+-----+-----+-----+  
|      John|      1|  1|      Sales|      1|  
|      Jeff|      3|  3|Engineering|      3|  
|      Mary|      3|  3|Engineering|      3|  
|      Mandy|      4|  4|  Finance|      4|  
|      Julie|      4|  4|  Finance|      4|  
+-----+-----+-----+-----+-----+
```

How Spark Performs Joins

- Joins can be expensive

How Spark Performs Joins

- Joins can be expensive
- There are two join strategies: **shuffle "hash" join** and **broadcast join**

How Spark Performs Joins

- Joins can be expensive
- There are two join strategies: **shuffle "hash" join** and **broadcast join**
- Strategy used is based on the size of the two datasets

How Spark Performs Joins

- Joins can be expensive
- There are two join strategies: **shuffle "hash" join** and **broadcast join**
- Strategy used is based on the size of the two datasets
- If both datasets are large (**big tables**), shuffle **hash** join is used

How Spark Performs Joins

- Joins can be expensive
- There are two join strategies: **shuffle "hash" join** and **broadcast join**
- Strategy used is based on the size of the two datasets
- If both datasets are large (**big tables**), shuffle hash join is used
- When at least one of the datasets is a **small table**, **broadcast join** is used

Shuffle "Hash" Join

- A join combines the matching rows of two datasets
- Matching rows need to be brought to the same partition

Shuffle "Hash" Join

- A join combines the matching rows of two datasets
- Matching rows need to be brought to the same partition
- First, compute the hash value of the columns in the join expression of each row in each dataset

Shuffle "Hash" Join

- A join combines the matching rows of two datasets
- Matching rows need to be brought to the same partition
- First, compute the hash value of the columns in the join expression of each row in each dataset
- Shuffle the rows with the same hash value to the same partition

Shuffle "Hash" Join

- A join combines the matching rows of two datasets
- Matching rows need to be brought to the same partition
- First, compute the hash value of the columns in the join expression of each row in each dataset
- Shuffle the rows with the same hash value to the same partition
- Join these rows

Broadcast "Hash" Join

- Applicable when one of the datasets is a small table

Broadcast "Hash" Join

- Applicable when one of the datasets is a small table
- Broadcast a copy of the entire small table to each of the partitions of the larger dataset

Broadcast "Hash" Join

- Applicable when one of the datasets is a small table
- Broadcast a copy of the entire small table to each of the partitions of the larger dataset
- Iterate through each row of the larger dataset to find the matching rows to join

Broadcast Hash Join

- Broadcast join is preferred when possible
- Spark can automatically figure out whether to use a broadcast hash join or shuffle hash join
- We can provide a hint to Spark to use a broadcast join

```
import org.apache.spark.sql.functions.broadcast
//print the explain plan to verify broadcast hash join strategy is used
employeeDF.join(broadcast(deptDF),
    employeeDF.col("dept_no") === deptDF.col("id")).explain()
== Physical Plan ==
*(1) BroadcastHashJoin [dept_no#634L], [id#640L], Inner, BuildRight, false
:- LocalTableScan [first_name#633, dept_no#634L]
+- BroadcastExchange HashedRelationBroadcastMode(List(input[0, bigint, false]))
+- LocalTableScan [id#640L, name#641]
```