Spark Azure Databricks

Databricks Spark Certification and Beyond

Module:
Joins

Lecture:
Dataframe
Joins

Dataframe Joins – Introduction

So far, in this course, we have been working with a single data frame. However, your Spark applications are going to bring together a large number of different data frames. And hence, you may need to join them together.

Joins are all about bringing together two data frames or two tables. I assume you already know SQL, and hence you also know the joins. You should at least know the following:

1. What do we mean by joining two tables?
2. What are the following join types:
    1. Inner Join
    2. Left Outer
    3. Right Outer
    4. Full Outer
    5. Cross Join
3. How to implement the above join types in SQL?
4. What is join key or join condition?

So with the assumption that you understand the above concepts, let us look at joins in Apache Spark.

Spark allows you to perform the following join types. We have five join types in Spark, and I have listed the keywords used for each join type in the screenshot below.

S. No.	Join Type	Dataframe Keywords	SQL Keywords
1.	Inner	inner	INNER
2.	Outer		
2.1	Left	left, leftouter, left_outer	LEFT, LEFT OUTER
2.2	Right	right, rightouter, right_outer	RIGHT, RIGHT OUTER
2.3	Full	outer, full, fullouter, full_outer	FULL , FULL OUTER
3.	Semi - Left	semi, leftsemi, left_semi	SEMI, LEFT SEMI
4.	Anti - Left	anti, leftanti, left_anti	ANTI, LEFT ANTI
5.	Cross	cross	CROSS

We have three types of outer joins: left, right, and full.
Spark Dataframe API is more flexible in terms of keywords. You can use left, leftouter, or the left_outer to define a left outer join. Spark SQL allows you to use the LEFT or the LEFT OUTER keywords. Similarly, I have listed all the keywords here.

S. No.	Join Type	Dataframe Keywords	SQL Keywords
1.	Inner	inner	INNER
2.	Outer		
2.1	Left	left, leftouter, left_outer	LEFT, LEFT OUTER
2.2	Right	right, rightouter, right_outer	RIGHT, RIGHT OUTER
2.3	Full	outer, full, fullouter, full_outer	FULL , FULL OUTER
3.	Semi - Left	semi, leftsemi, left_semi	SEMI, LEFT SEMI
4.	Anti - Left	anti, leftanti, left_anti	ANTI, LEFT ANTI
5.	Cross	cross	CROSS

Let's start our discussion with the inner join.
You need two data frames to perform a join in Spark. So let's assume I have these two data frames given below. The first one is the order Dataframe, and the second one is the product Dataframe. The order Dataframe contains order information.

order

order_id	prod_id	qty	unit_price
01	02	1	350
01	04	1	580
01	07	2	320
02	03	1	450
02	06	1	220
03	01	1	195
04	09	1	270
04	08	2	410
05	02	1	350

product

prod_id	prod_name	list_price	qty
01	Scroll Mouse	250	20
02	Optical Mouse	350	20
03	Wireless Mouse	450	50
04	Wireless Keyboard	580	50
05	Standard Keyboard	360	10
06	16 GB Flash Storage	240	100
07	32 GB Flash Storage	320	50
08	64 GB Flash Storage	430	25

Look at the first three order records. These three records represent one order.
And in this one order, we sold three product codes: 02, 04, and 07.
We sold one quantity of product 02 at the unit price of 350.

order

order_id	prod_id	qty	unit_price
01	02	1	350
01	04	1	580
01	07	2	320
02	03	1	450
02	06	1	220
03	01	1	195
04	09	1	270
04	08	2	410
05	02	1	350

product

prod_id	prod_name	list_price	qty
01	Scroll Mouse	250	20
02	Optical Mouse	350	20
03	Wireless Mouse	450	50
04	Wireless Keyboard	580	50
05	Standard Keyboard	360	10
06	16 GB Flash Storage	240	100
07	32 GB Flash Storage	320	50
08	64 GB Flash Storage	430	25

The second dataframe is the product table.
You have the prod_id, product_name, list_price, and quantity in the stock.

order

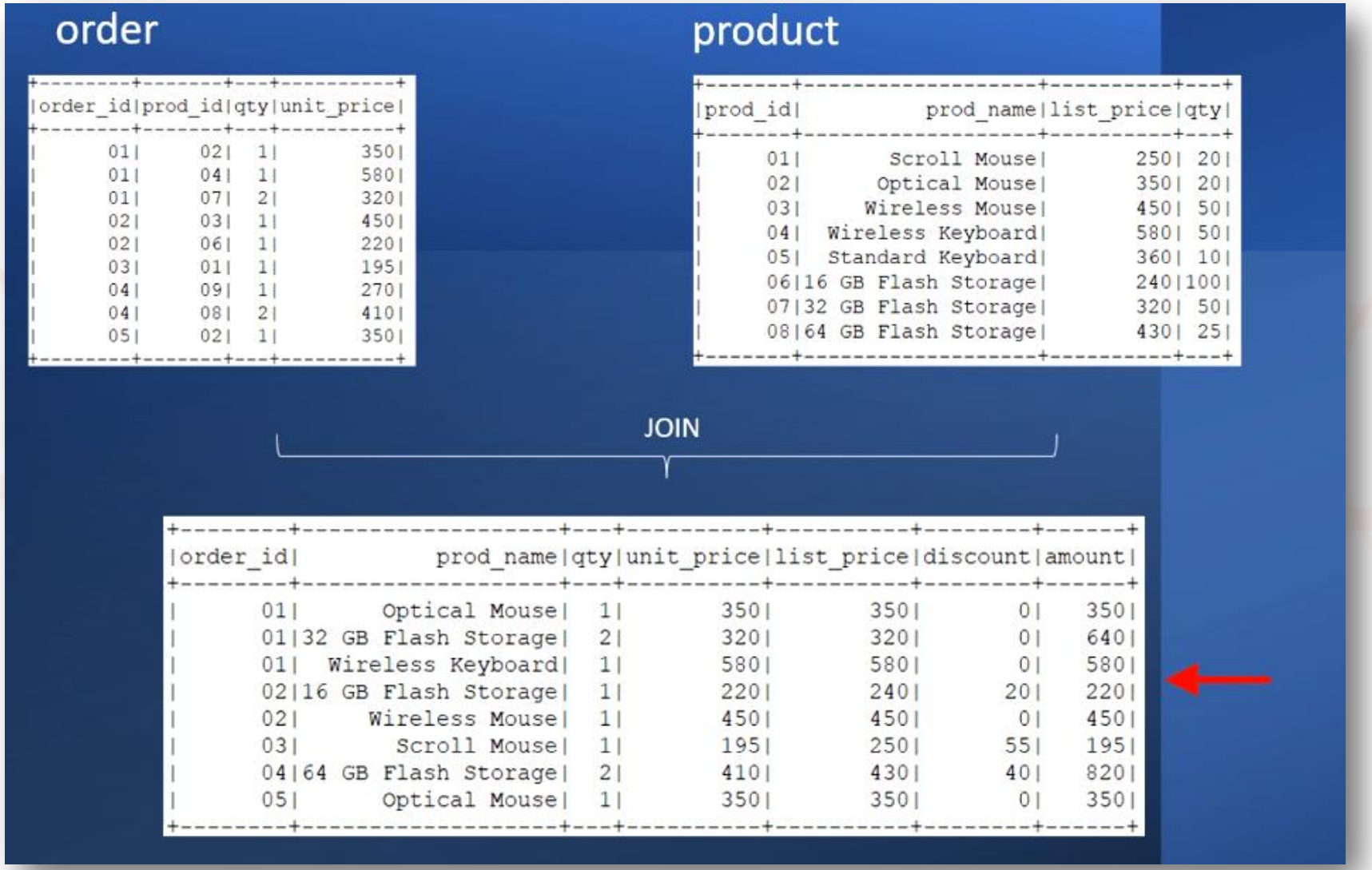
order_id	prod_id	qty	unit_price
01	02	1	350
01	04	1	580
01	07	2	320
02	03	1	450
02	06	1	220
03	01	1	195
04	09	1	270
04	08	2	410
05	02	1	350

product

prod_id	prod_name	list_price	qty
01	Scroll Mouse	250	20
02	Optical Mouse	350	20
03	Wireless Mouse	450	50
04	Wireless Keyboard	580	50
05	Standard Keyboard	360	10
06	16 GB Flash Storage	240	100
07	32 GB Flash Storage	320	50
08	64 GB Flash Storage	430	25

So you have two different datasets.

Now you are given a task to prepare the following output.



order

order_id	prod_id	qty	unit_price
01	02	1	350
01	04	1	580
01	07	2	320
02	03	1	450
02	06	1	220
03	01	1	195
04	09	1	270
04	08	2	410
05	02	1	350

product

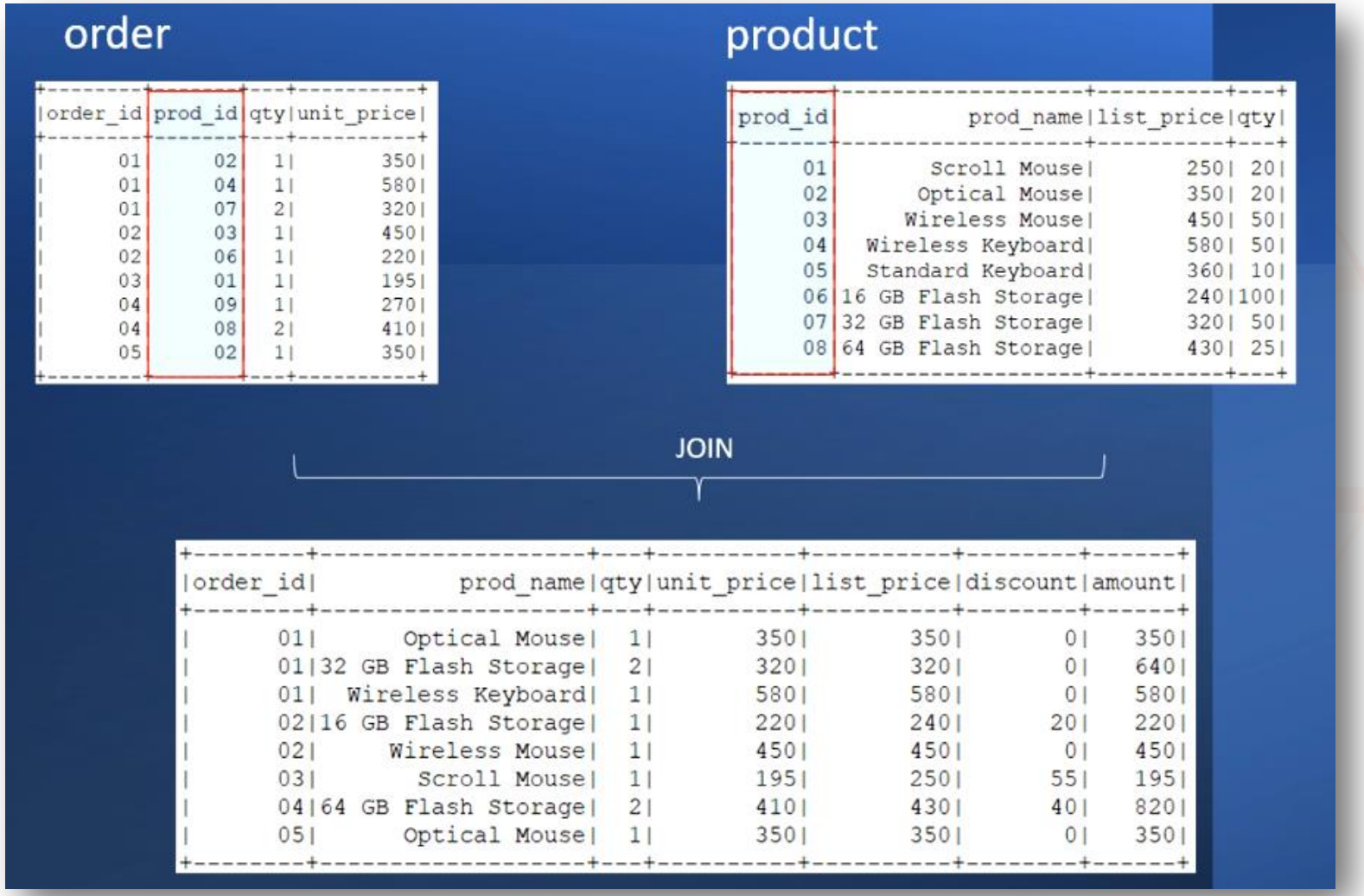
prod_id	prod_name	list_price	qty
01	Scroll Mouse	250	20
02	Optical Mouse	350	20
03	Wireless Mouse	450	50
04	Wireless Keyboard	580	50
05	Standard Keyboard	360	10
06	16 GB Flash Storage	240	100
07	32 GB Flash Storage	320	50
08	64 GB Flash Storage	430	25

order_id	prod_name	qty	unit_price	list_price	discount	amount
01	Optical Mouse	1	350	350	0	350
01	32 GB Flash Storage	2	320	320	0	640
01	Wireless Keyboard	1	580	580	0	580
02	16 GB Flash Storage	1	220	240	20	220
02	Wireless Mouse	1	450	450	0	450
03	Scroll Mouse	1	195	250	55	195
04	64 GB Flash Storage	2	410	430	40	820
05	Optical Mouse	1	350	350	0	350

So you are asked to list the orders with product name, sold quantity, unit price, and list_price.
You are also supposed to calculate the discount and the final amount.

order_id	prod_name	qty	unit_price	list_price	discount	amount
01	Optical Mouse	1	350	350	0	350
01	32 GB Flash Storage	2	320	320	0	640
01	Wireless Keyboard	1	580	580	0	580
02	16 GB Flash Storage	1	220	240	20	220
02	Wireless Mouse	1	450	450	0	450
03	Scroll Mouse	1	195	250	55	195
04	64 GB Flash Storage	2	410	430	40	820
05	Optical Mouse	1	350	350	0	350

We need to list the orders. So I can start with the orders Dataframe. Then I need to bring in the product name from the product Dataframe. So I will join the orders and products using the product id. Once joined, we will have all the information in a single Dataframe, and we can easily calculate the discount and total amount.



order

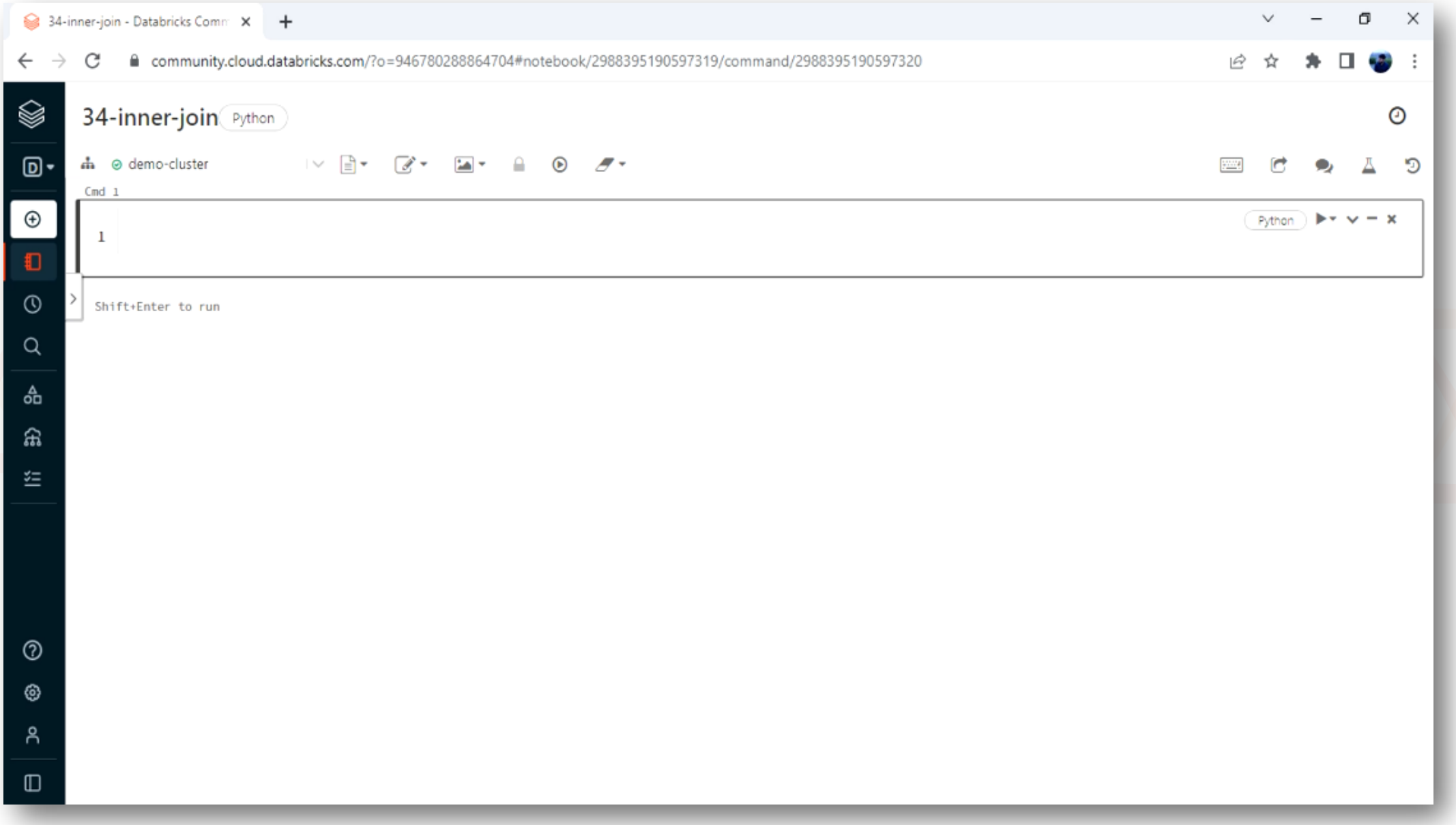
order_id	prod_id	qty	unit_price
01	02	1	350
01	04	1	580
01	07	2	320
02	03	1	450
02	06	1	220
03	01	1	195
04	09	1	270
04	08	2	410
05	02	1	350

product

prod_id	prod_name	list_price	qty
01	Scroll Mouse	250	20
02	Optical Mouse	350	20
03	Wireless Mouse	450	50
04	Wireless Keyboard	580	50
05	Standard Keyboard	360	10
06	16 GB Flash Storage	240	100
07	32 GB Flash Storage	320	50
08	64 GB Flash Storage	430	25

order_id	prod_name	qty	unit_price	list_price	discount	amount
01	Optical Mouse	1	350	350	0	350
01	32 GB Flash Storage	2	320	320	0	640
01	Wireless Keyboard	1	580	580	0	580
02	16 GB Flash Storage	1	220	240	20	220
02	Wireless Mouse	1	450	450	0	450
03	Scroll Mouse	1	195	250	55	195
04	64 GB Flash Storage	2	410	430	40	820
05	Optical Mouse	1	350	350	0	350

Go to your Databricks workspace and create a new notebook. **(Reference: 34-inner-join)**



I will create two sample data frames.
Here is the first one, order_df Dataframe.

Here is the second Dataframe, product_df.

Cmd 2

```
1  product_list = [("01", "Scroll Mouse", 250, 20),
2                  ("02", "Optical Mouse", 350, 20),
3                  ("03", "Wireless Mouse", 450, 50),
4                  ("04", "Wireless Keyboard", 580, 50),
5                  ("05", "Standard Keyboard", 360, 10),
6                  ("06", "16 GB Flash Storage", 240, 100),
7                  ("07", "32 GB Flash Storage", 320, 50),
8                  ("08", "64 GB Flash Storage", 430, 25)]
9
10 product_df = spark.createDataFrame(product_list).toDF("prod_id", "prod_name", "list_price", "qty")
```

Command took 0.16 seconds -- by prashant@scholarnest.com at 6/24/2022, 3:50:34 PM on demo-cluster

I want to join these two guys. And we need a join condition for that. We already figured the join condition that we will be joining these two Dataframes using the product id.

```
1  orders_list = [("01", "02", 1, 350),
2                 ("01", "04", 1, 580),
3                 ("01", "07", 2, 320),
4                 ("02", "03", 1, 450),
5                 ("02", "06", 1, 220),
6                 ("03", "01", 1, 195),
7                 ("04", "09", 1, 270),
8                 ("04", "08", 2, 410),
9                 ("05", "02", 1, 350)]
10
11 order_df = spark.createDataFrame(orders_list).toDF("order_id", "prod_id", "qty", "unit_price")
```

Command took 1.32 seconds -- by prashant@scholarnest.com at 6/24/2022, 3:45:45 PM on demo-cluster

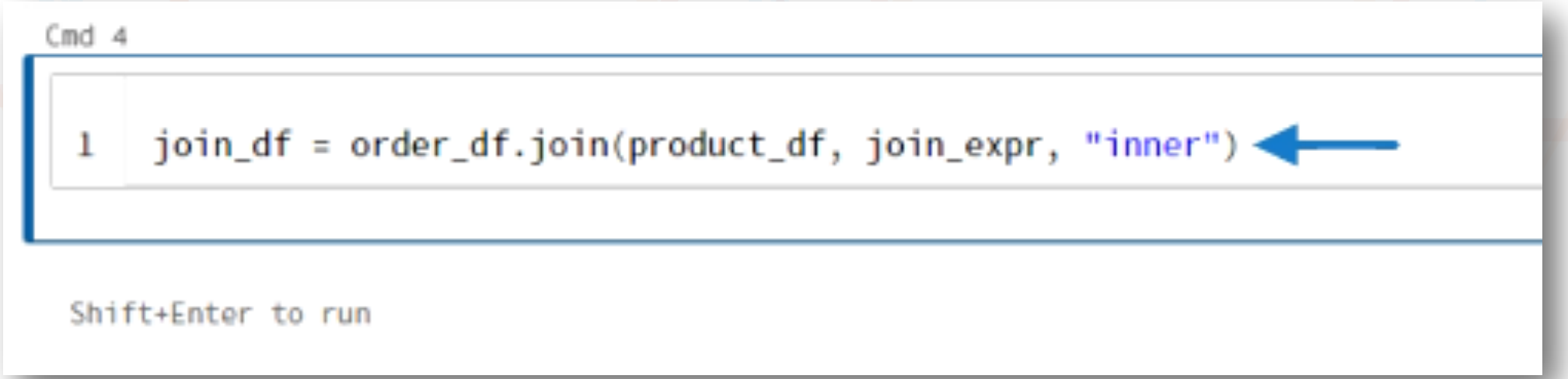
Cmd 2

```
1  product_list = [("01", "Scroll Mouse", 250, 20),
2                  ("02", "Optical Mouse", 350, 20),
3                  ("03", "Wireless Mouse", 450, 50),
4                  ("04", "Wireless Keyboard", 580, 50),
5                  ("05", "Standard Keyboard", 360, 10),
6                  ("06", "16 GB Flash Storage", 240, 100),
7                  ("07", "32 GB Flash Storage", 320, 50),
8                  ("08", "64 GB Flash Storage", 430, 25)]
9
10 product_df = spark.createDataFrame(product_list).toDF("prod_id", "prod_name", "list_price", "qty")
```

Command took 0.16 seconds -- by prashant@scholarnest.com at 6/24/2022, 3:50:34 PM on demo-cluster

Here is the join condition.
I use a dataframe.column name notation to define the join condition, because both the field names are the same. So, I must use the Dataframe name with the field names to avoid column name ambiguity.

This is how we perform Dataframe join as shown in the screenshot below.
Every join operation will have two data frames. One left Dataframe and the other right Dataframe. You can choose which Dataframe you want to make your left Dataframe.
I wanted to consider the order_df as my left data frame. So, I start with the order_df.
We always start with the left data frame and pass in the right data frame as the first argument to the join method.
So in this code example, order_df is my left Dataframe, and the product_df is the right Dataframe.



The join method takes two more arguments: Join condition and Join Type.
You can define a variable for the join condition and pass it here. Or you can type in the expressions directly. It works both ways. I prefer defining a variable for my join condition and using it here.
The inner join is the default join type. So you can skip the third argument if you want to perform an inner join. However, it is best practice to be explicit and specify your join type.

Let us try to understand what is happening in the background when I join two Dataframes. Here is the join code, and here are my two data frames shown below.

```
join_expr = order_df.prod_id == product_df.prod_id
```

```
join_df = order_df.join(product_df, join_expr, "inner")
```

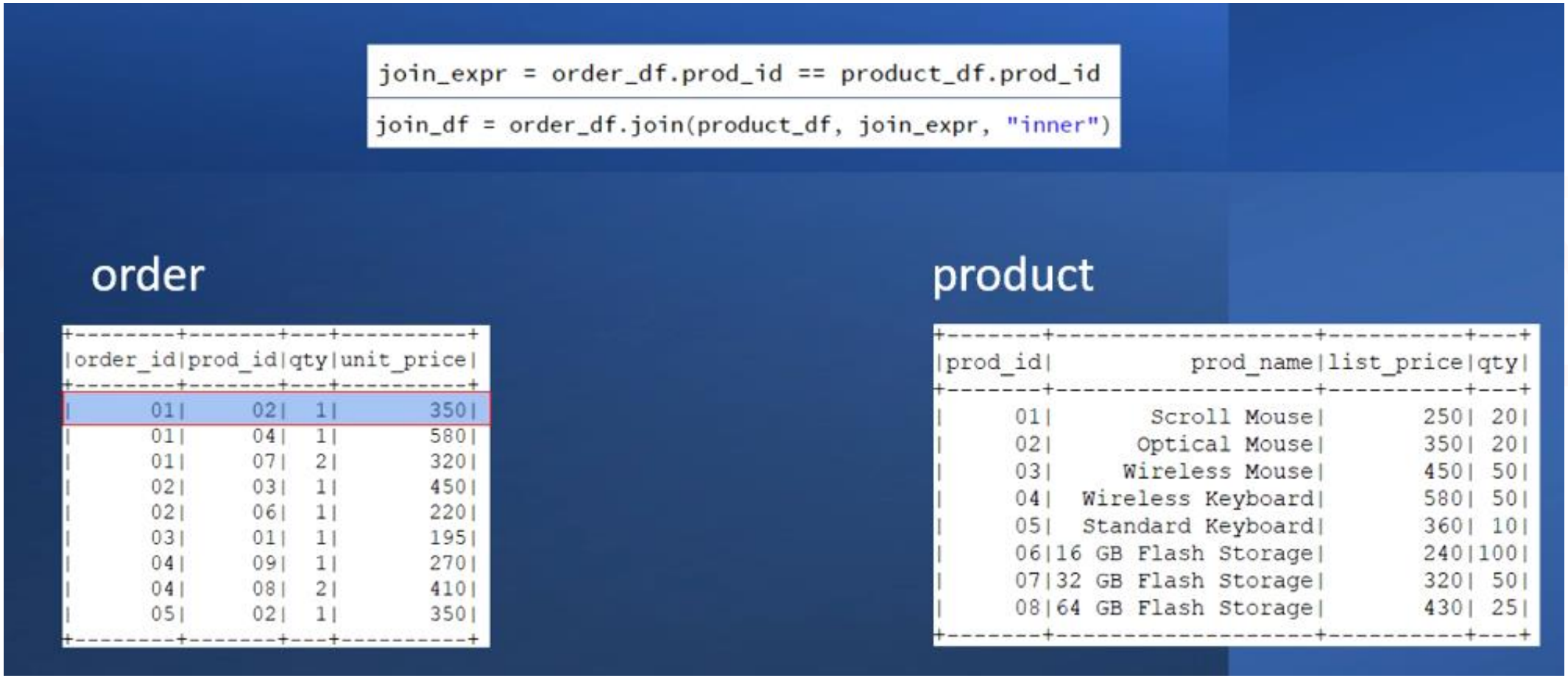
order

order_id	prod_id	qty	unit_price
01	02	1	350
01	04	1	580
01	07	2	320
02	03	1	450
02	06	1	220
03	01	1	195
04	09	1	270
04	08	2	410
05	02	1	350

product

prod_id	prod_name	list_price	qty
01	Scroll Mouse	250	20
02	Optical Mouse	350	20
03	Wireless Mouse	450	50
04	Wireless Keyboard	580	50
05	Standard Keyboard	360	10
06	16 GB Flash Storage	240	100
07	32 GB Flash Storage	320	50
08	64 GB Flash Storage	430	25

The internal implementation of the join is complex. But let's try to understand the logical flow of operations. What logically happens when you perform a join.
Spark is going to take one row from your left data frame. And search for the matching record on the right Dataframe using the join condition.



I found the match for the first record.

```
join_expr = order_df.prod_id == product_df.prod_id
join_df = order_df.join(product_df, join_expr, "inner")
```

order

order_id	prod_id	qty	unit_price
01	02	1	350
01	04	1	580
01	07	2	320
02	03	1	450
02	06	1	220
03	01	1	195
04	09	1	270
04	08	2	410
05	02	1	350

product

prod_id	prod_name	list_price	qty
01	Scroll Mouse	250	20
02	Optical Mouse	350	20
03	Wireless Mouse	450	50
04	Wireless Keyboard	580	50
05	Standard Keyboard	360	10
06	16 GB Flash Storage	240	100
07	32 GB Flash Storage	320	50
08	64 GB Flash Storage	430	25

Then it goes to the second row and repeats the same, trying to find a match.

```
join_expr = order_df.prod_id == product_df.prod_id
join_df = order_df.join(product_df, join_expr, "inner")
```

order

order_id	prod_id	qty	unit_price
01	02	1	350
01	04	1	580
01	07	2	320
02	03	1	450
02	06	1	220
03	01	1	195
04	09	1	270
04	08	2	410
05	02	1	350

product

prod_id	prod_name	list_price	qty
01	Scroll Mouse	250	20
02	Optical Mouse	350	20
03	Wireless Mouse	450	50
04	Wireless Keyboard	580	50
05	Standard Keyboard	360	10
06	16 GB Flash Storage	240	100
07	32 GB Flash Storage	320	50
08	64 GB Flash Storage	430	25

We found the match for the second one also.

```
join_expr = order_df.prod_id == product_df.prod_id
join_df = order_df.join(product_df, join_expr, "inner")
```

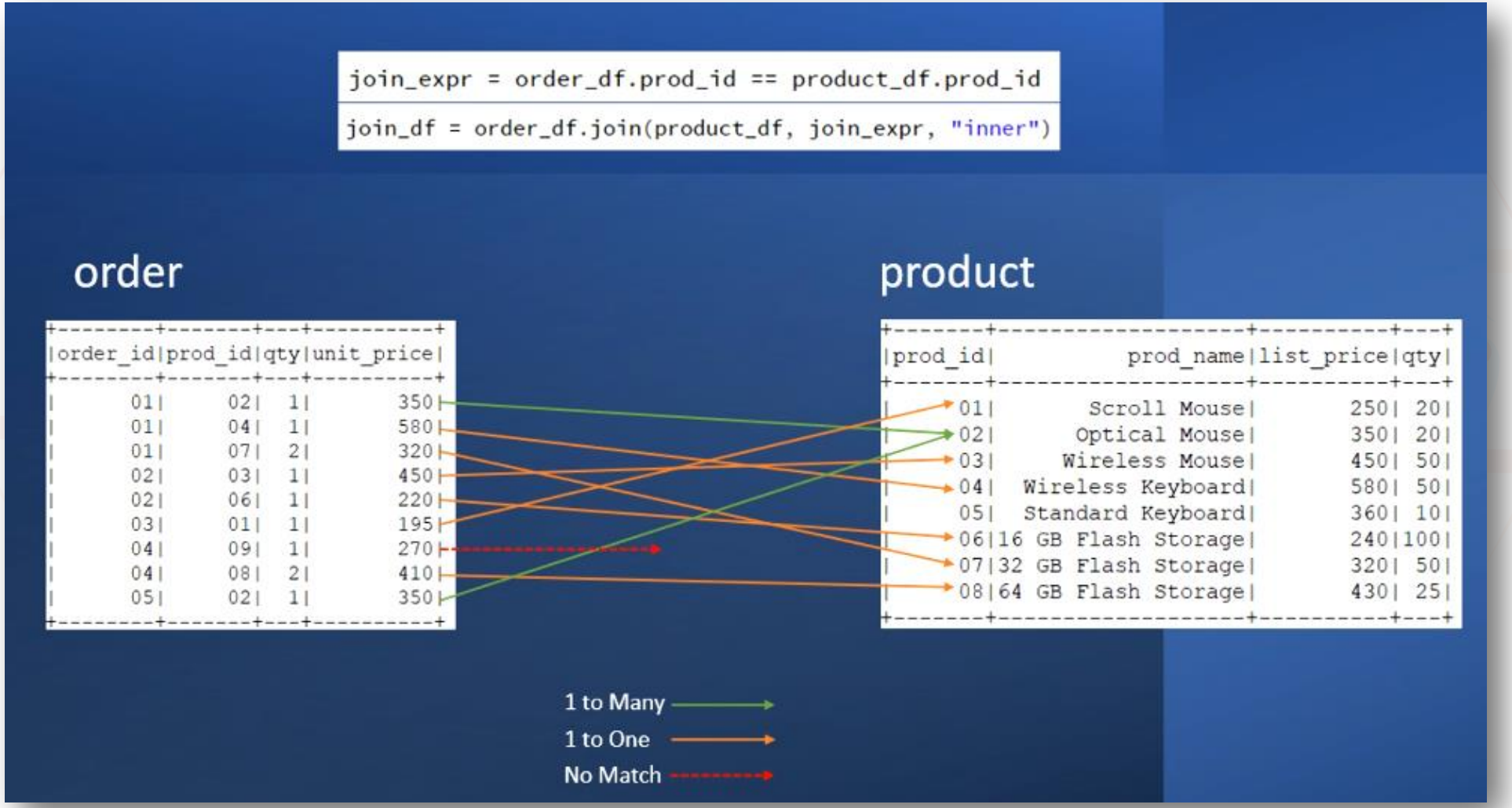
order

order_id	prod_id	qty	unit_price
01	02	1	350
01	04	1	580
01	07	2	320
02	03	1	450
02	06	1	220
03	01	1	195
04	09	1	270
04	08	2	410
05	02	1	350

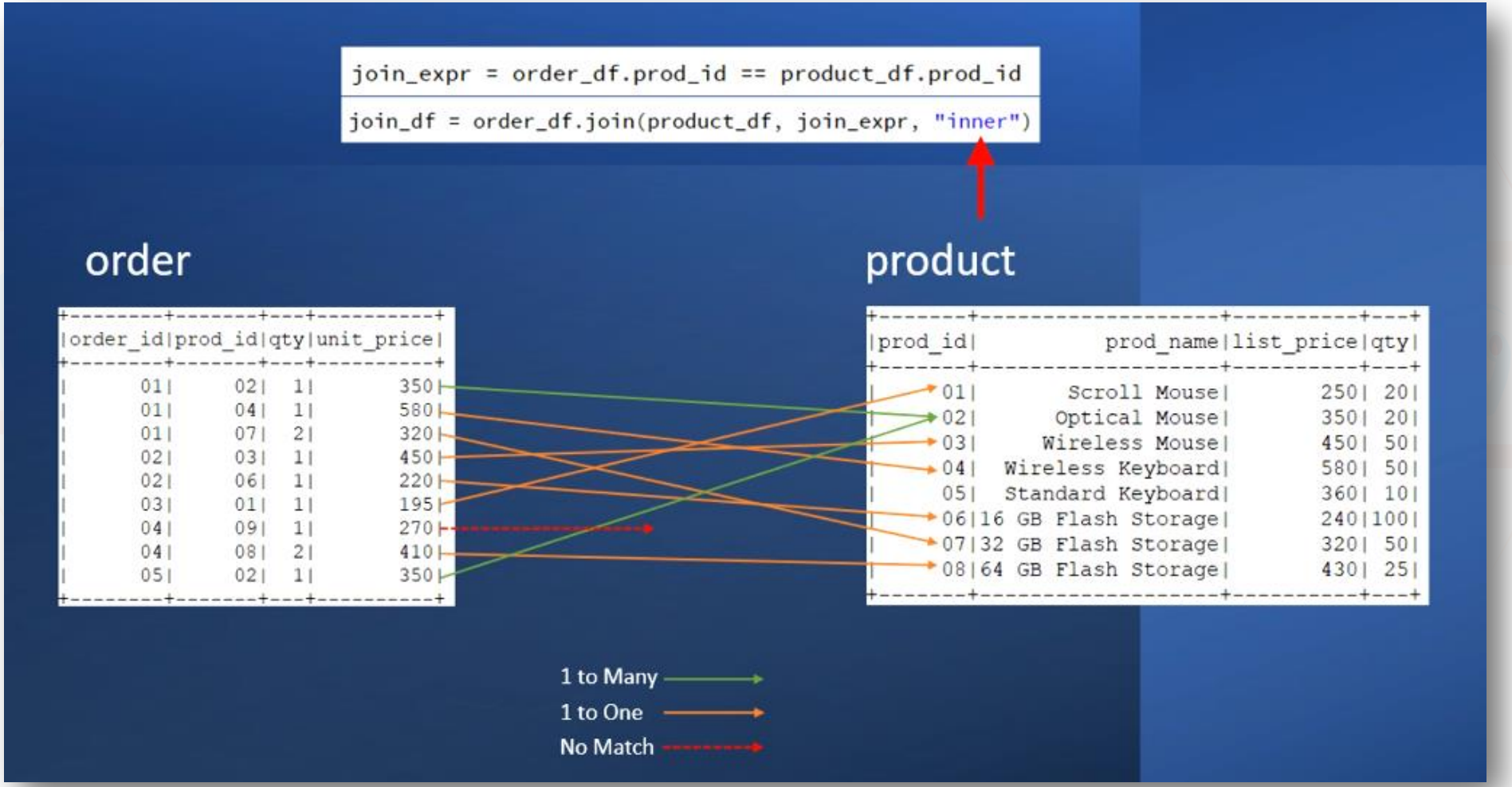
product

prod_id	prod_name	list_price	qty
01	Scroll Mouse	250	20
02	Optical Mouse	350	20
03	Wireless Mouse	450	50
04	Wireless Keyboard	580	50
05	Standard Keyboard	360	10
06	16 GB Flash Storage	240	100
07	32 GB Flash Storage	320	50
08	64 GB Flash Storage	430	25

This goes on for all the records as shown below. Some records will have exactly one match. Some guys might find multiple matches. You may have records that do not find a match. Anything is possible. Spark will do all these matching exercises.



The next step is to combine these matching records from the left and the right data frames to create a new data frame. That's where the join type comes into play. The join type decides what to do with the unmatched records. This example is applying inner join.



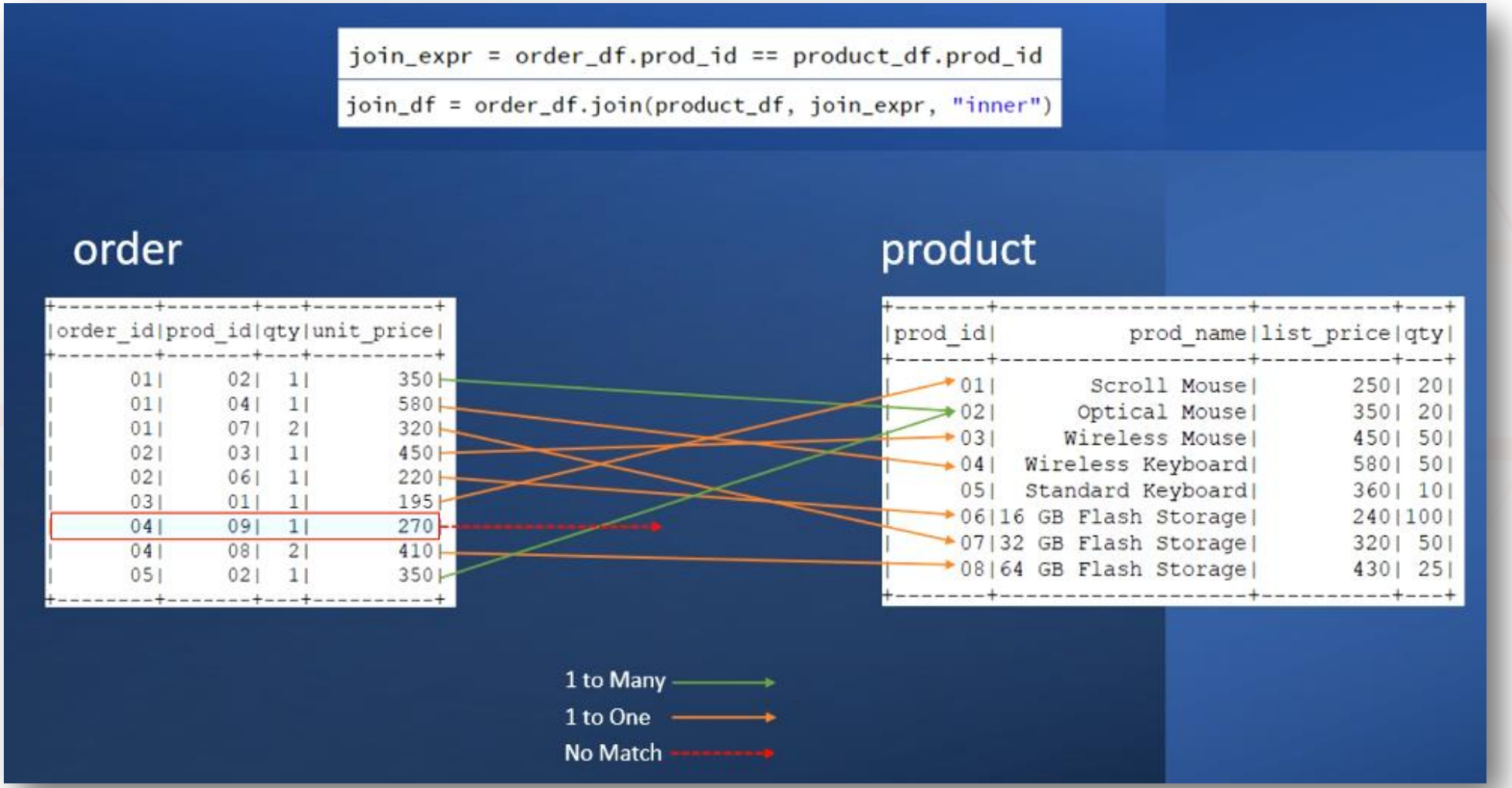
order

order_id	prod_id	qty	unit_price
01	02	1	350
01	04	1	580
01	07	2	320
02	03	1	450
02	06	1	220
03	01	1	195
04	09	1	270
04	08	2	410
05	02	1	350

product

prod_id	prod_name	list_price	qty
01	Scroll Mouse	250	20
02	Optical Mouse	350	20
03	Wireless Mouse	450	50
04	Wireless Keyboard	580	50
05	Standard Keyboard	360	10
06	16 GB Flash Storage	240	100
07	32 GB Flash Storage	320	50
08	64 GB Flash Storage	430	25

The inner join will take only those records that match with at least one record on the other side. So, in our example, everyone has got a match except the product code 09. This guy doesn't have a matching product. So the inner join will exclude this record from the join.



order_id	prod_id	qty	unit_price
01	02	1	350
01	04	1	580
01	07	2	320
02	03	1	450
02	06	1	220
03	01	1	195
04	09	1	270
04	08	2	410
05	02	1	350

prod_id	prod_name	list_price	qty
01	Scroll Mouse	250	20
02	Optical Mouse	350	20
03	Wireless Mouse	450	50
04	Wireless Keyboard	580	50
05	Standard Keyboard	360	10
06	16 GB Flash Storage	240	100
07	32 GB Flash Storage	320	50
08	64 GB Flash Storage	430	25

Here is the code to join the Dataframe and display it.

Here is the joined Dataframe. So we see only eight records. The record for the prod_id 09 is missing. We are performing an inner join, and that's how inner join works. It will exclude unmatched records. Product code 09 remained unmatched, so we do not see that order line item here.

Cmd 5

```
1  join_df.orderBy("order_id").show()
```

▸ (3) Spark Jobs

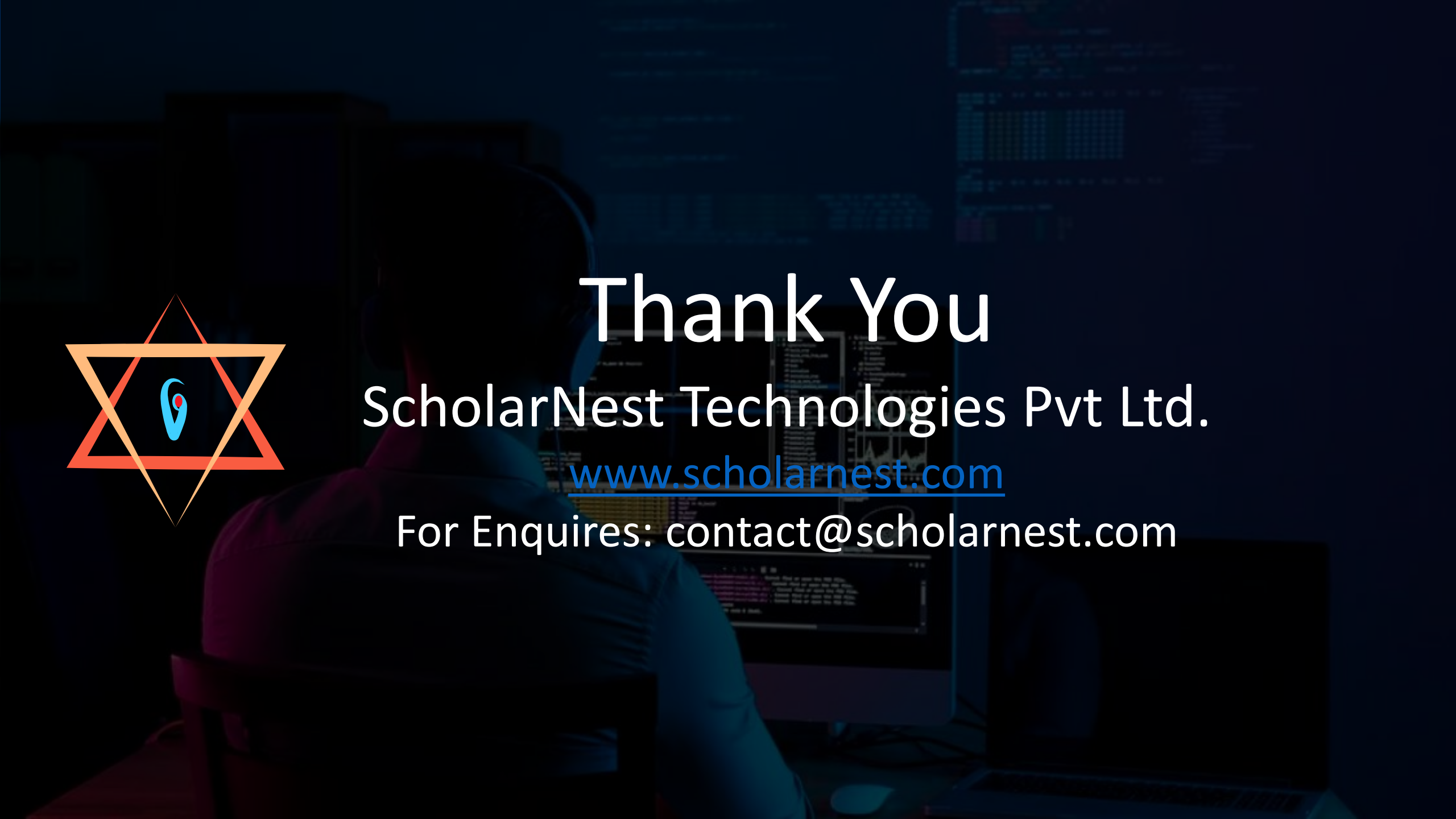
order_id	prod_id	qty	unit_price	prod_id	prod_name	list_price	qty
01	02	1	350	02	Optical Mouse	350	20
01	07	2	320	07	32 GB Flash Storage	320	50
01	04	1	580	04	Wireless Keyboard	580	50
02	06	1	220	06	16 GB Flash Storage	240	100
02	03	1	450	03	Wireless Mouse	450	50
03	01	1	195	01	Scroll Mouse	250	20
04	08	2	410	08	64 GB Flash Storage	430	25
05	02	1	350	02	Optical Mouse	350	20

Command took 7.57 seconds -- by prashant@scholarnest.com at 6/24/2022, 4:00:38 PM on demo-cluster

Here is a quick summary of what we learned in this chapter.
We have 5 join types listed below. And we also have the template code for joining two Dataframes given below. You can take a left Dataframe and call the join() method. The first argument for the join method is the right Dataframe. Then you give a join condition and set the join type. We have already learned to implement the inner join.

S. No.	Join Type	Dataframe Keywords	SQL Keywords
1.	Inner	inner	INNER
2.	Outer		
2.1	Left	left, leftouter, left_outer	LEFT, LEFT OUTER
2.2	Right	right, rightouter, right_outer	RIGHT, RIGHT OUTER
2.3	Full	outer, full, fullouter, full_outer	FULL , FULL OUTER
3.	Semi - Left	semi, leftsemi, left_semi	SEMI, LEFT SEMI
4.	Anti - Left	anti, leftanti, left_anti	ANTI, LEFT ANTI
5.	Cross	cross	CROSS

```
left_df.join(right_df, join_expr, join_type)
```

Thank You

ScholarNest Technologies Pvt Ltd.

www.scholarnest.com

For Enquires: contact@scholarnest.com

Spark Azure Databricks

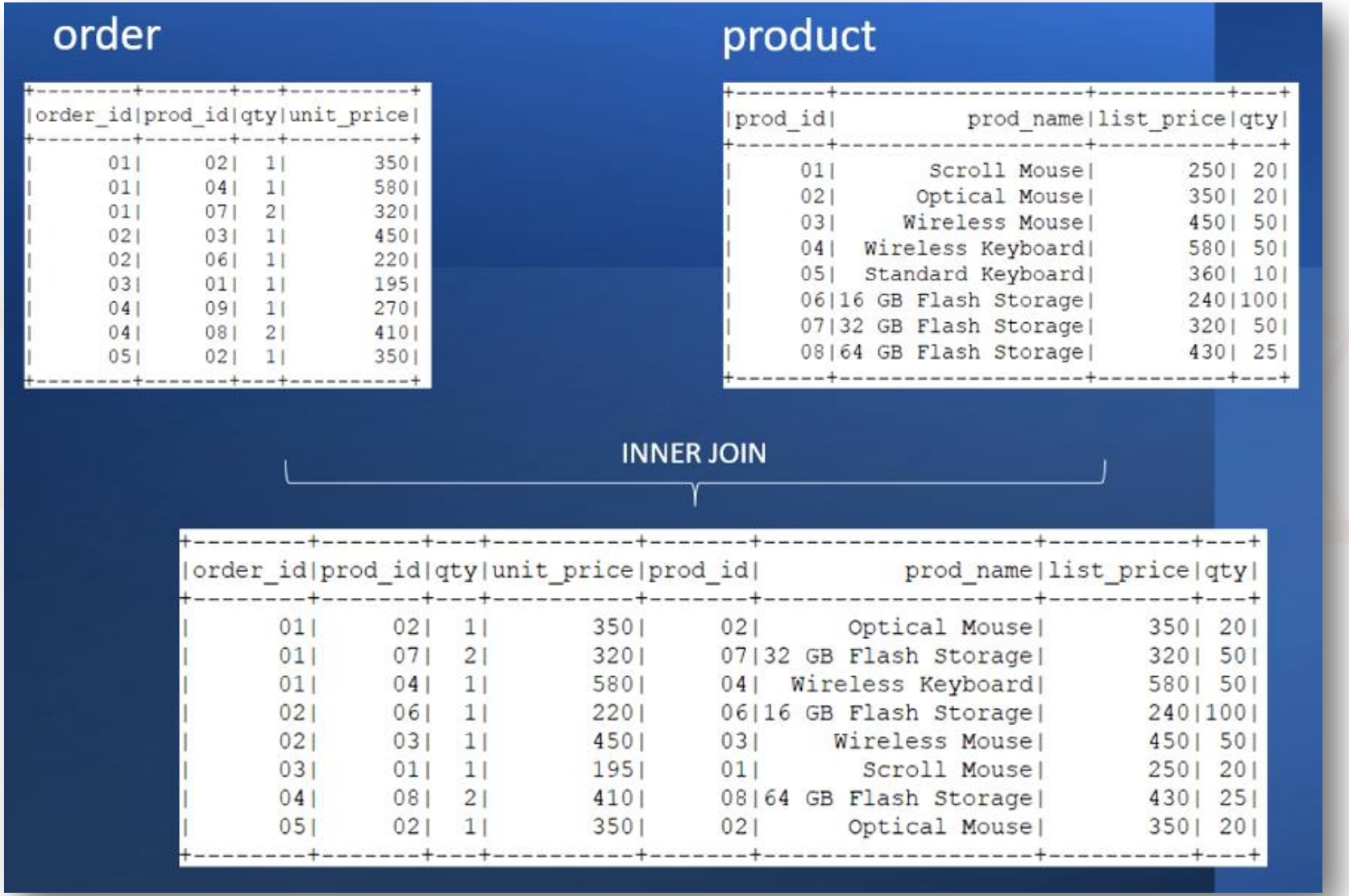
Databricks Spark Certification and Beyond

Module:
Joins

Lecture:
Outer
Joins

Outer Joins

We created the following data frames (order_df and product_df) in the earlier lecture, and performed an inner join on these two Dataframes to get the Dataframe shown below.



order

order_id	prod_id	qty	unit_price
01	02	1	350
01	04	1	580
01	07	2	320
02	03	1	450
02	06	1	220
03	01	1	195
04	09	1	270
04	08	2	410
05	02	1	350

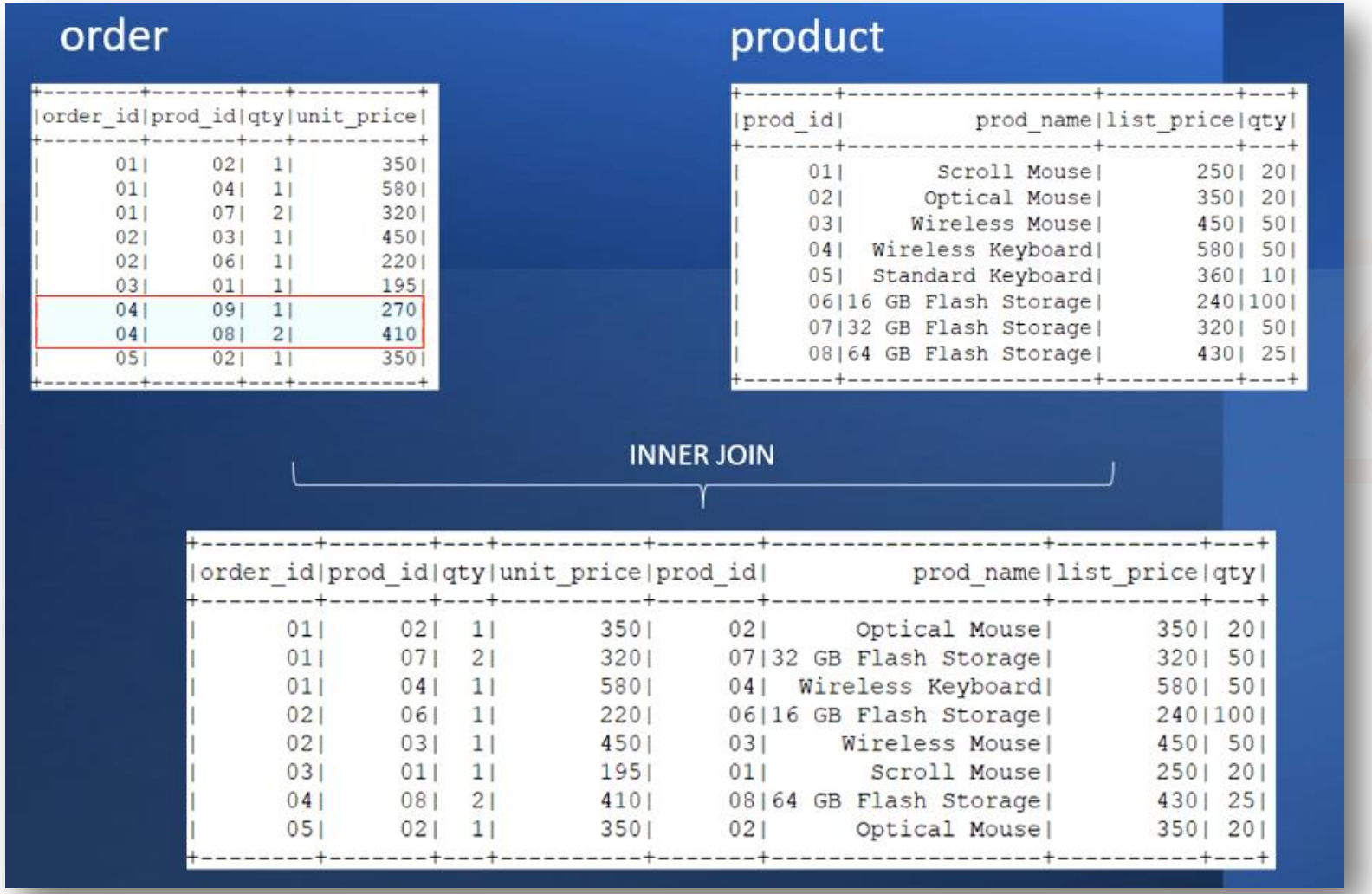
product

prod_id	prod_name	list_price	qty
01	Scroll Mouse	250	20
02	Optical Mouse	350	20
03	Wireless Mouse	450	50
04	Wireless Keyboard	580	50
05	Standard Keyboard	360	10
06	16 GB Flash Storage	240	100
07	32 GB Flash Storage	320	50
08	64 GB Flash Storage	430	25

INNER JOIN

order_id	prod_id	qty	unit_price	prod_id	prod_name	list_price	qty
01	02	1	350	02	Optical Mouse	350	20
01	07	2	320	07	32 GB Flash Storage	320	50
01	04	1	580	04	Wireless Keyboard	580	50
02	06	1	220	06	16 GB Flash Storage	240	100
02	03	1	450	03	Wireless Mouse	450	50
03	01	1	195	01	Scroll Mouse	250	20
04	08	2	410	08	64 GB Flash Storage	430	25
05	02	1	350	02	Optical Mouse	350	20

We also noticed the missing record in the final result. For example, order_id 4 had two products. However, we got only one in the join outcome. The other record is missing, because we performed an inner join, and product id 9 is not present in the product master. This record is skipped because the inner join could not find a matching record on the right side.



order

order_id	prod_id	qty	unit_price
01	02	1	350
01	04	1	580
01	07	2	320
02	03	1	450
02	06	1	220
03	01	1	195
04	09	1	270
04	08	2	410
05	02	1	350

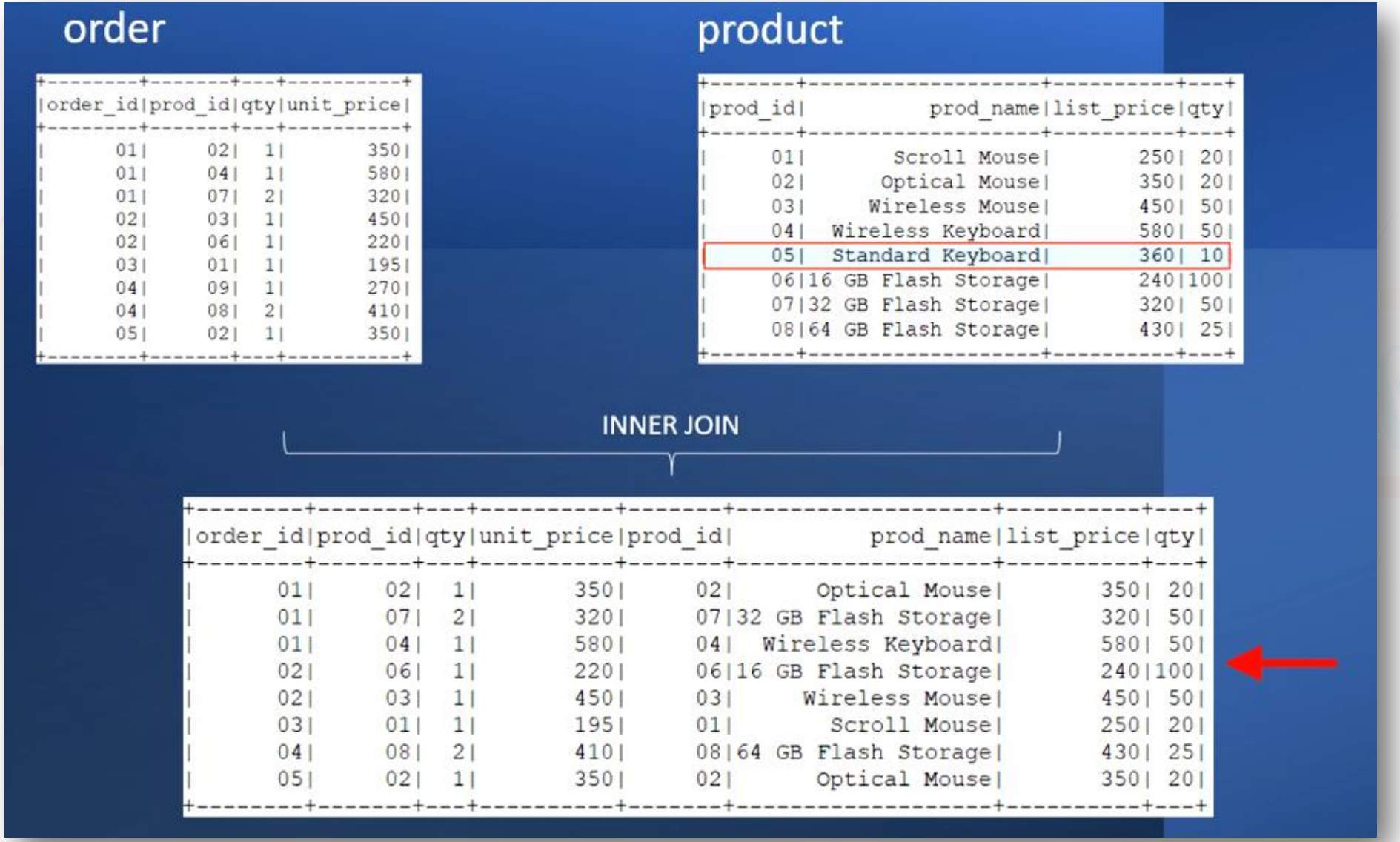
product

prod_id	prod_name	list_price	qty
01	Scroll Mouse	250	20
02	Optical Mouse	350	20
03	Wireless Mouse	450	50
04	Wireless Keyboard	580	50
05	Standard Keyboard	360	10
06	16 GB Flash Storage	240	100
07	32 GB Flash Storage	320	50
08	64 GB Flash Storage	430	25

INNER JOIN

order_id	prod_id	qty	unit_price	prod_id	prod_name	list_price	qty
01	02	1	350	02	Optical Mouse	350	20
01	07	2	320	07	32 GB Flash Storage	320	50
01	04	1	580	04	Wireless Keyboard	580	50
02	06	1	220	06	16 GB Flash Storage	240	100
02	03	1	450	03	Wireless Mouse	450	50
03	01	1	195	01	Scroll Mouse	250	20
04	08	2	410	08	64 GB Flash Storage	430	25
05	02	1	350	02	Optical Mouse	350	20

Similarly, the Standard Keyboard doesn't show up in the outcome because none of the orders had a reference to the standard keyboard.



order

order_id	prod_id	qty	unit_price
01	02	1	350
01	04	1	580
01	07	2	320
02	03	1	450
02	06	1	220
03	01	1	195
04	09	1	270
04	08	2	410
05	02	1	350

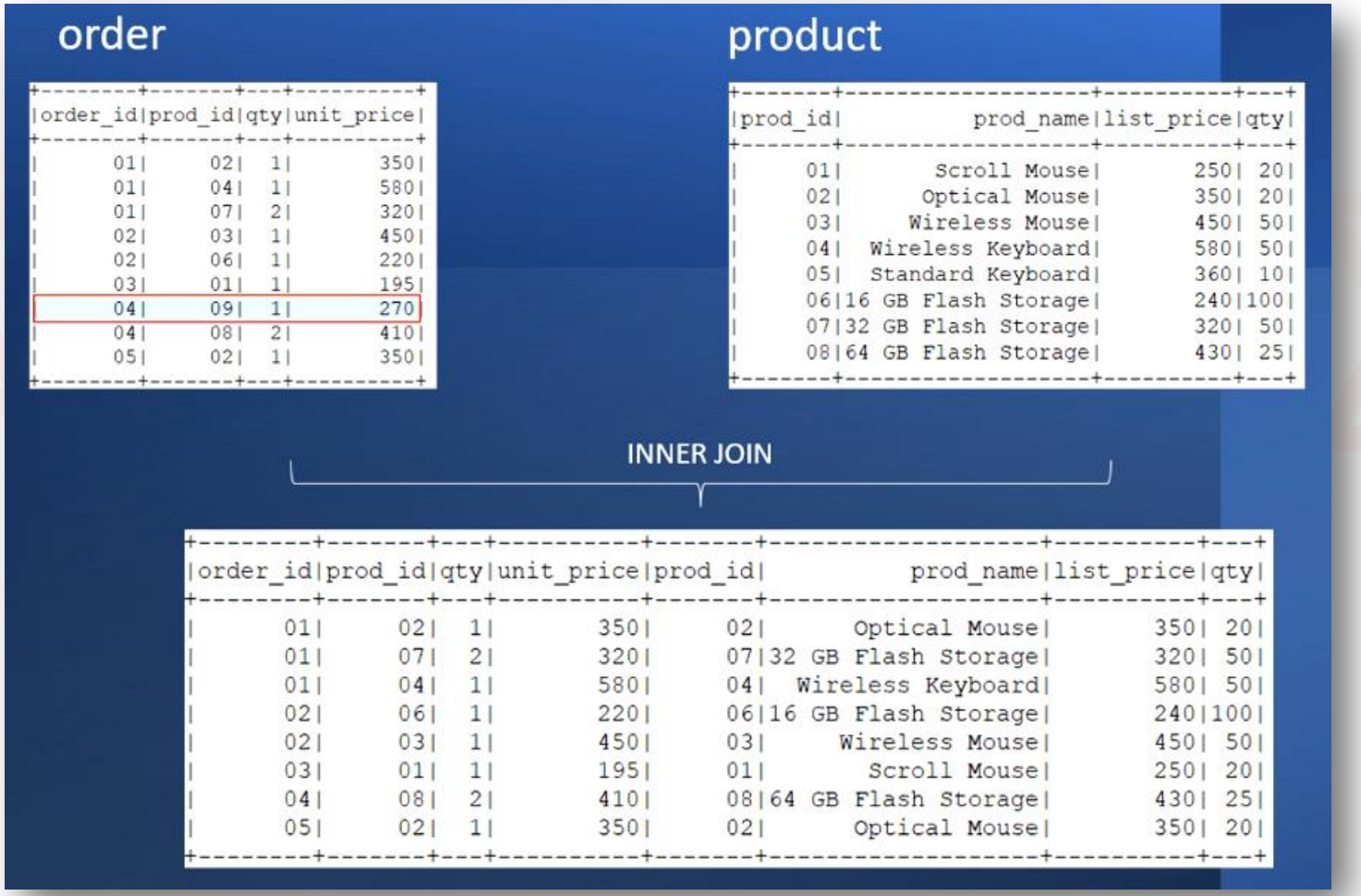
product

prod_id	prod_name	list_price	qty
01	Scroll Mouse	250	20
02	Optical Mouse	350	20
03	Wireless Mouse	450	50
04	Wireless Keyboard	580	50
05	Standard Keyboard	360	10
06	16 GB Flash Storage	240	100
07	32 GB Flash Storage	320	50
08	64 GB Flash Storage	430	25

INNER JOIN

order_id	prod_id	qty	unit_price	prod_id	prod_name	list_price	qty
01	02	1	350	02	Optical Mouse	350	20
01	07	2	320	07	32 GB Flash Storage	320	50
01	04	1	580	04	Wireless Keyboard	580	50
02	06	1	220	06	16 GB Flash Storage	240	100
02	03	1	450	03	Wireless Mouse	450	50
03	01	1	195	01	Scroll Mouse	250	20
04	08	2	410	08	64 GB Flash Storage	430	25
05	02	1	350	02	Optical Mouse	350	20

The inner join is going to take only those records that are matched on both sides. So, the inner join makes sense in only those requirements when you know that your data will match on both sides, and you purposefully want to skip the missing records. However, a missing record doesn't make much sense in this example. We sold one unit of something called product id 09. We don't know the item's name, but we made a transaction. So, we cannot lose it from the final result.



order

order_id	prod_id	qty	unit_price
01	02	1	350
01	04	1	580
01	07	2	320
02	03	1	450
02	06	1	220
03	01	1	195
04	09	1	270
04	08	2	410
05	02	1	350

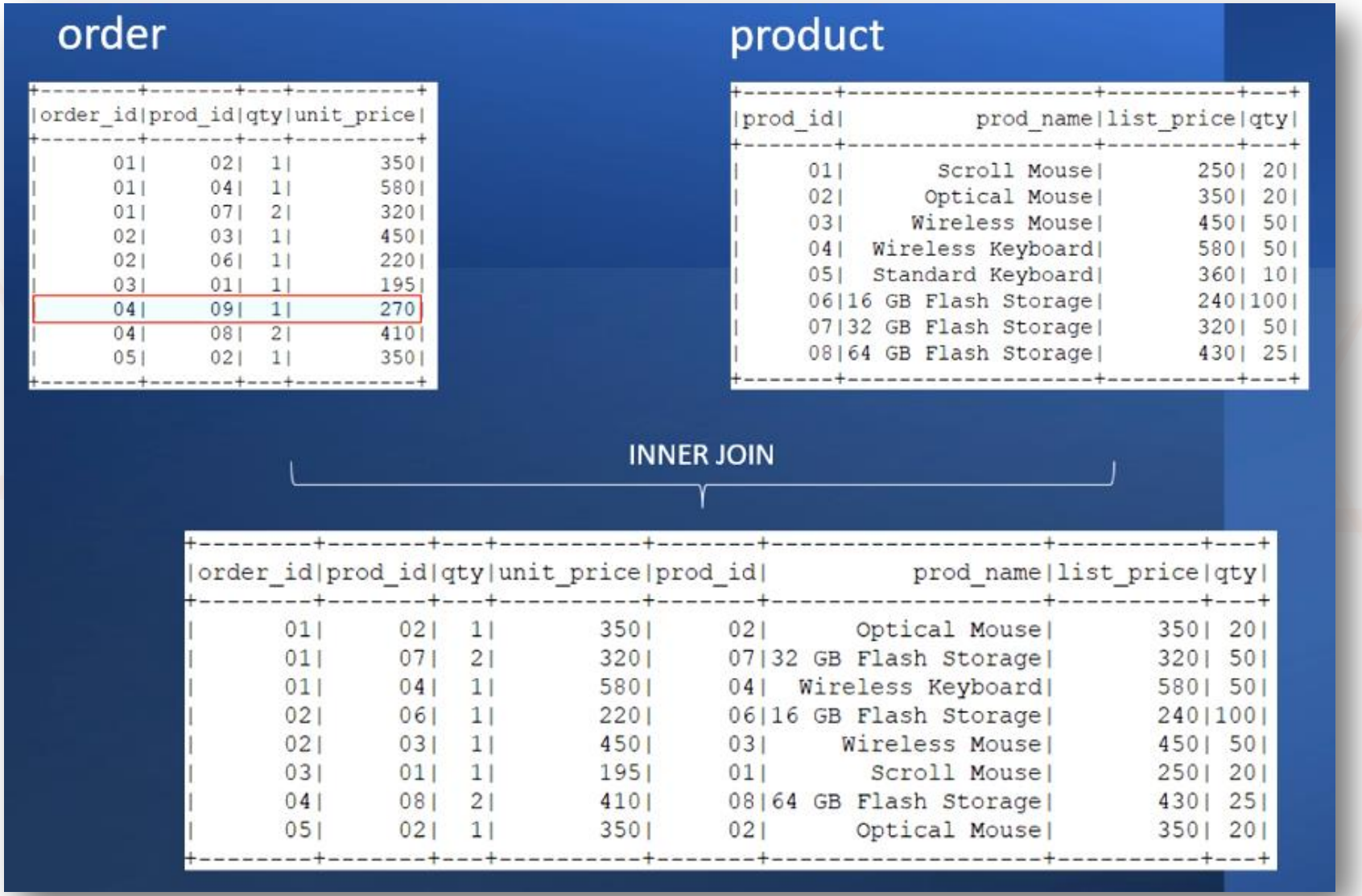
product

prod_id	prod_name	list_price	qty
01	Scroll Mouse	250	20
02	Optical Mouse	350	20
03	Wireless Mouse	450	50
04	Wireless Keyboard	580	50
05	Standard Keyboard	360	10
06	16 GB Flash Storage	240	100
07	32 GB Flash Storage	320	50
08	64 GB Flash Storage	430	25

INNER JOIN

order_id	prod_id	qty	unit_price	prod_id	prod_name	list_price	qty
01	02	1	350	02	Optical Mouse	350	20
01	07	2	320	07	32 GB Flash Storage	320	50
01	04	1	580	04	Wireless Keyboard	580	50
02	06	1	220	06	16 GB Flash Storage	240	100
02	03	1	450	03	Wireless Mouse	450	50
03	01	1	195	01	Scroll Mouse	250	20
04	08	2	410	08	64 GB Flash Storage	430	25
05	02	1	350	02	Optical Mouse	350	20

In order to fix this problem, we need to use Outer Join.
The outer join will take up all the records even if they do not have a matching pair.

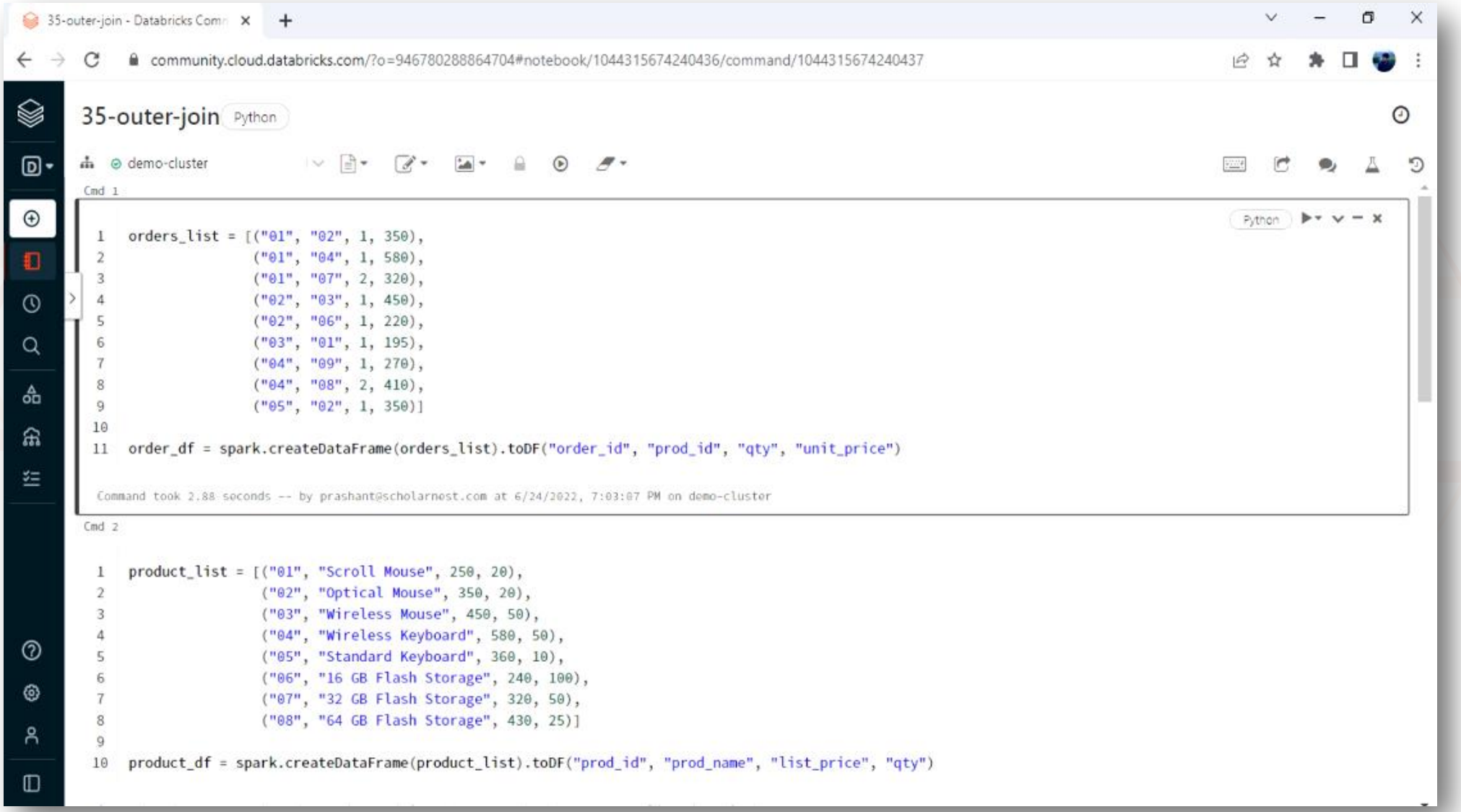


order_id	prod_id	qty	unit_price
01	02	1	350
01	04	1	580
01	07	2	320
02	03	1	450
02	06	1	220
03	01	1	195
04	09	1	270
04	08	2	410
05	02	1	350

prod_id	prod_name	list_price	qty
01	Scroll Mouse	250	20
02	Optical Mouse	350	20
03	Wireless Mouse	450	50
04	Wireless Keyboard	580	50
05	Standard Keyboard	360	10
06	16 GB Flash Storage	240	100
07	32 GB Flash Storage	320	50
08	64 GB Flash Storage	430	25

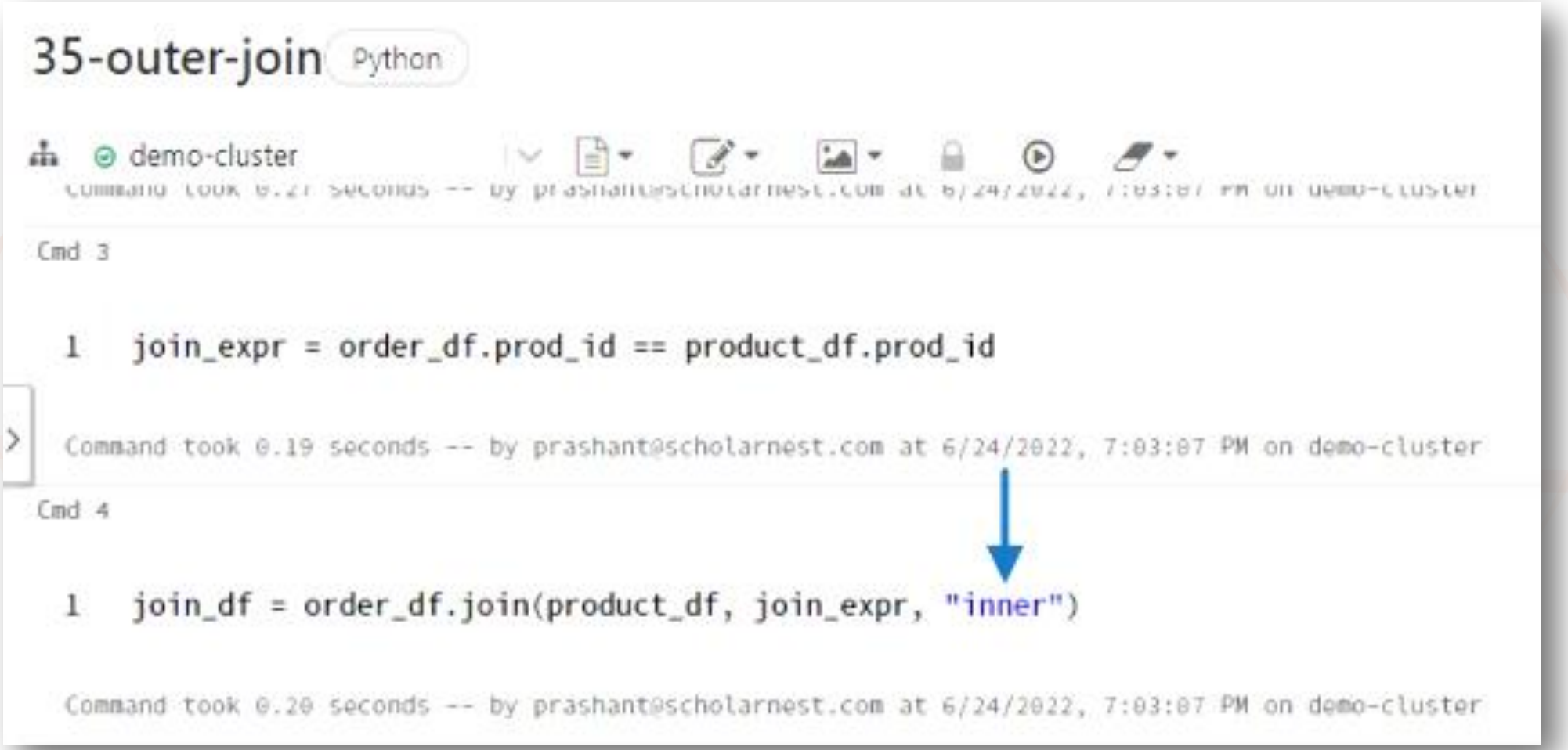
order_id	prod_id	qty	unit_price	prod_id	prod_name	list_price	qty
01	02	1	350	02	Optical Mouse	350	20
01	07	2	320	07	32 GB Flash Storage	320	50
01	04	1	580	04	Wireless Keyboard	580	50
02	06	1	220	06	16 GB Flash Storage	240	100
02	03	1	450	03	Wireless Mouse	450	50
03	01	1	195	01	Scroll Mouse	250	20
04	08	2	410	08	64 GB Flash Storage	430	25
05	02	1	350	02	Optical Mouse	350	20

I cloned the earlier notebook and created a new one. **(Reference : 35-outer-join.ipynb)**
The code is the same as earlier. I have the order_df. Then I also have the product_df.



Then I have the join condition here which we used for inner join, this condition remains the same for outer join as well.

The next cell is applying the join.
This code is doing an inner join, and I will change the join type to outer.



Now we have changed the join type to outer. Now, we can display the Dataframe and visualize the difference between inner and outer joins.

```
Cmd 4

1  join_df = order_df.join(product_df, join_expr, "outer")

Command took 0.08 seconds -- by prashant@scholarnest.com at 6/24/2022, 7:04:21 PM on demo-cluster
```


Here is the Dataframe for outer join.
Now, let us try to understand this in detail.

Cmd 5

```
1  join_df.orderBy("order_id").show()
```

▸ (3) Spark Jobs

```
+--------+-------+----+----------+-------+--------------------+----------+----+
|order_id|prod_id| qty|unit_price|prod_id|           prod_name|list_price| qty|
+--------+-------+----+----------+-------+--------------------+----------+----+
|    null|   null|null|      null|     05|  Standard Keyboard|       360|  10|
|      01|     04|   1|       580|     04|  Wireless Keyboard|       580|  50|
|      01|     02|   1|       350|     02|       Optical Mouse|       350|  20|
|      01|     07|   2|       320|     07|32 GB Flash Storage|       320|  50|
|      02|     03|   1|       450|     03|     Wireless Mouse|       450|  50|
|      02|     06|   1|       220|     06|16 GB Flash Storage|       240| 100|
|      03|     01|   1|       195|     01|        Scroll Mouse|       250|  20|
|      04|     08|   2|       410|     08|64 GB Flash Storage|       430|  25|
|      04|     09|   1|       270|   null|                null|      null|null|
|      05|     02|   1|       350|     02|       Optical Mouse|       350|  20|
+--------+-------+----+----------+-------+--------------------+----------+----+
```

Command took 1.87 seconds -- by prashant@scholarnest.com at 6/24/2022, 7:04:25 PM on demo-cluster

We got three records of order id 1.
They were expected because we have a matching record on the right side.

order

order_id	prod_id	qty	unit_price
01	02	1	350
01	04	1	580
01	07	2	320
02	03	1	450
02	06	1	220
03	01	1	195
04	09	1	270
04	08	2	410
05	02	1	350

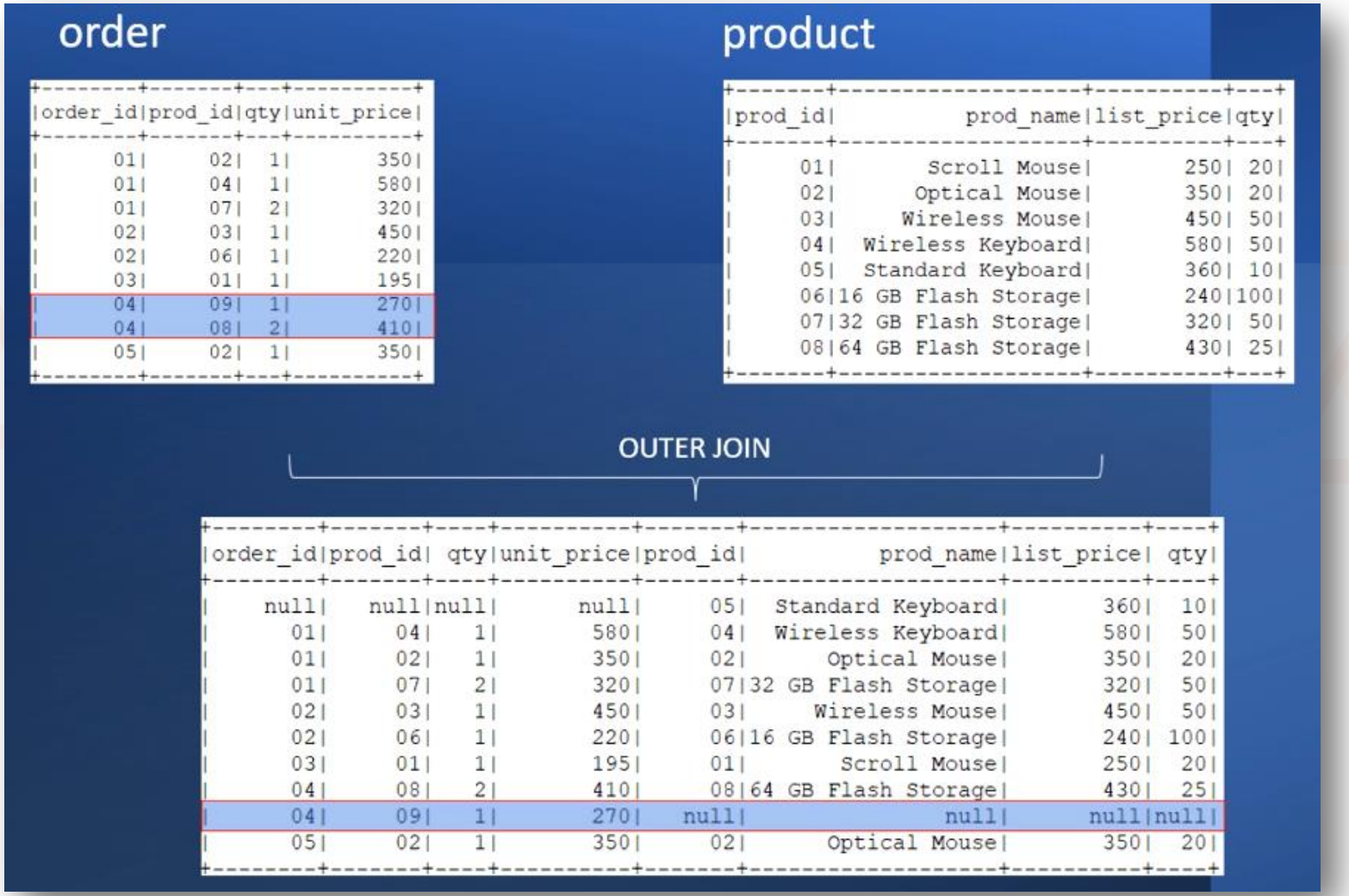
product

prod_id	prod_name	list_price	qty
01	Scroll Mouse	250	20
02	Optical Mouse	350	20
03	Wireless Mouse	450	50
04	Wireless Keyboard	580	50
05	Standard Keyboard	360	10
06	16 GB Flash Storage	240	100
07	32 GB Flash Storage	320	50
08	64 GB Flash Storage	430	25

OUTER JOIN

order_id	prod_id	qty	unit_price	prod_id	prod_name	list_price	qty
null	null	null	null	05	Standard Keyboard	360	10
01	04	1	580	04	Wireless Keyboard	580	50
01	02	1	350	02	Optical Mouse	350	20
01	07	2	320	07	32 GB Flash Storage	320	50
02	03	1	450	03	Wireless Mouse	450	50
02	06	1	220	06	16 GB Flash Storage	240	100
03	01	1	195	01	Scroll Mouse	250	20
04	08	2	410	08	64 GB Flash Storage	430	25
04	09	1	270	null	null	null	null
05	02	1	350	02	Optical Mouse	350	20

Now, look at the order id 04. We are now also getting the product id 9. This item doesn't have a matching record on the right side. But you still get it here because we used outer join. And since we do not have a match on the right side, we do not know anything about these column values. Hence, these values are null. So, we got the record from the right side even if the left side is missing.



order

order_id	prod_id	qty	unit_price
01	02	1	350
01	04	1	580
01	07	2	320
02	03	1	450
02	06	1	220
03	01	1	195
04	09	1	270
04	08	2	410
05	02	1	350

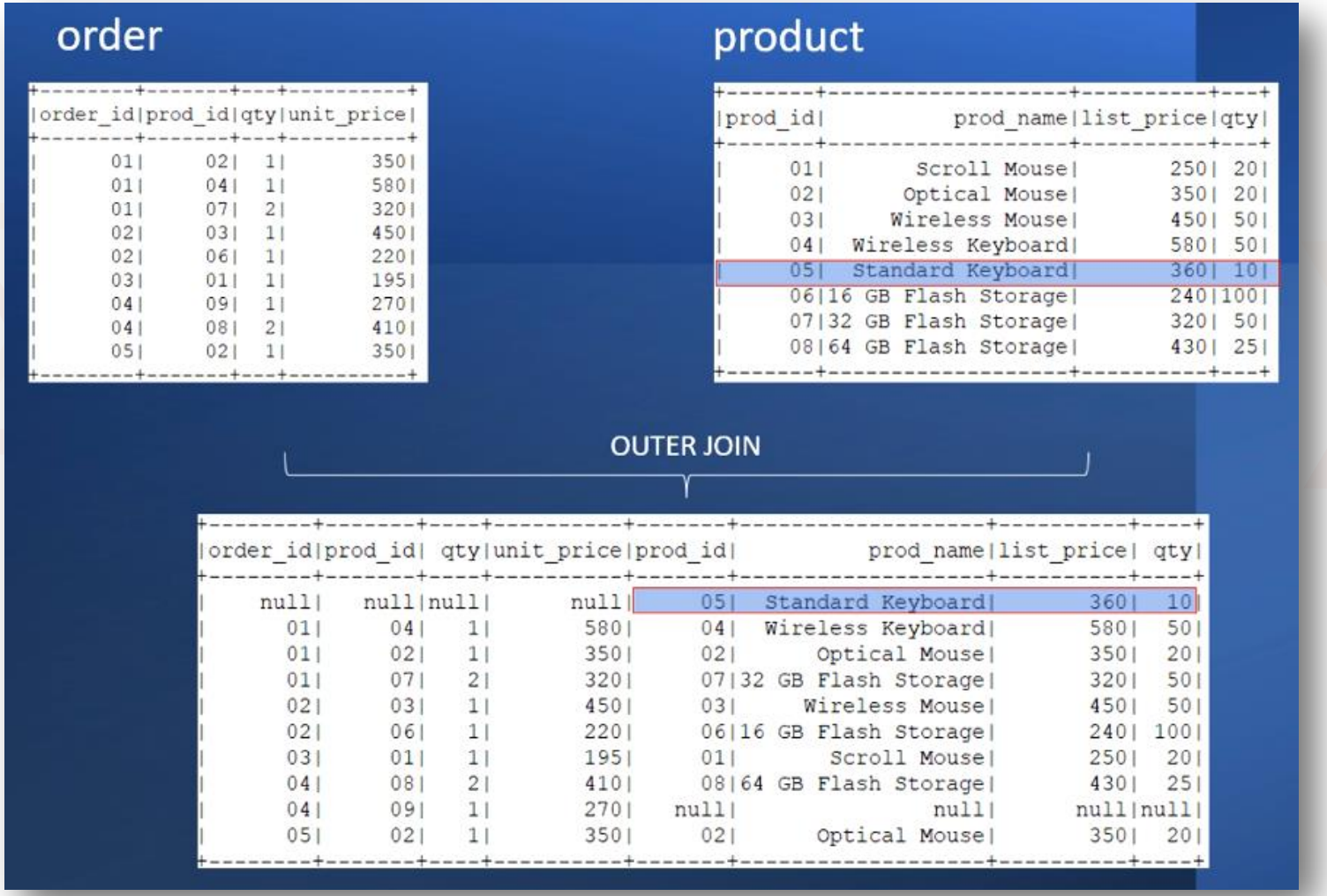
product

prod_id	prod_name	list_price	qty
01	Scroll Mouse	250	20
02	Optical Mouse	350	20
03	Wireless Mouse	450	50
04	Wireless Keyboard	580	50
05	Standard Keyboard	360	10
06	16 GB Flash Storage	240	100
07	32 GB Flash Storage	320	50
08	64 GB Flash Storage	430	25

OUTER JOIN

order_id	prod_id	qty	unit_price	prod_id	prod_name	list_price	qty
null	null	null	null	05	Standard Keyboard	360	10
01	04	1	580	04	Wireless Keyboard	580	50
01	02	1	350	02	Optical Mouse	350	20
01	07	2	320	07	32 GB Flash Storage	320	50
02	03	1	450	03	Wireless Mouse	450	50
02	06	1	220	06	16 GB Flash Storage	240	100
03	01	1	195	01	Scroll Mouse	250	20
04	08	2	410	08	64 GB Flash Storage	430	25
04	09	1	270	null	null	null	null
05	02	1	350	02	Optical Mouse	350	20

Similarly, we also got one record from the right side even though none of the left side orders referred to the standard keyboard. It means, the outer join will ensure that we get each record from both sides at least once, even if there is no match on the other side. And this type of join is known as full outer join. Why full? Because we take every record from the left and right both.



order

order_id	prod_id	qty	unit_price
01	02	1	350
01	04	1	580
01	07	2	320
02	03	1	450
02	06	1	220
03	01	1	195
04	09	1	270
04	08	2	410
05	02	1	350

product

prod_id	prod_name	list_price	qty
01	Scroll Mouse	250	20
02	Optical Mouse	350	20
03	Wireless Mouse	450	50
04	Wireless Keyboard	580	50
05	Standard Keyboard	360	10
06	16 GB Flash Storage	240	100
07	32 GB Flash Storage	320	50
08	64 GB Flash Storage	430	25

OUTER JOIN

order_id	prod_id	qty	unit_price	prod_id	prod_name	list_price	qty
null	null	null	null	05	Standard Keyboard	360	10
01	04	1	580	04	Wireless Keyboard	580	50
01	02	1	350	02	Optical Mouse	350	20
01	07	2	320	07	32 GB Flash Storage	320	50
02	03	1	450	03	Wireless Mouse	450	50
02	06	1	220	06	16 GB Flash Storage	240	100
03	01	1	195	01	Scroll Mouse	250	20
04	08	2	410	08	64 GB Flash Storage	430	25
04	09	1	270	null	null	null	null
05	02	1	350	02	Optical Mouse	350	20

However, for our example, the standard keyboard doesn't make any sense because I intend to generate a sales report. I have not sold a single unit of a standard keyboard. So, this record doesn't make any sense in my report. I want to remove the standard keyboard from the result. And in order to do that, we have to look at the variation of outer join.

Spark offers you three types of outer joins:

1. **Full Outer Join** – All the records from both sides, even if we do not have a match.
2. **Left Outer Join** – All the records from the left side even if we do not have a matching pair on the right side.
3. **Right Outer Join** – All the records from the right side even if we do not have a matching record on the left side.

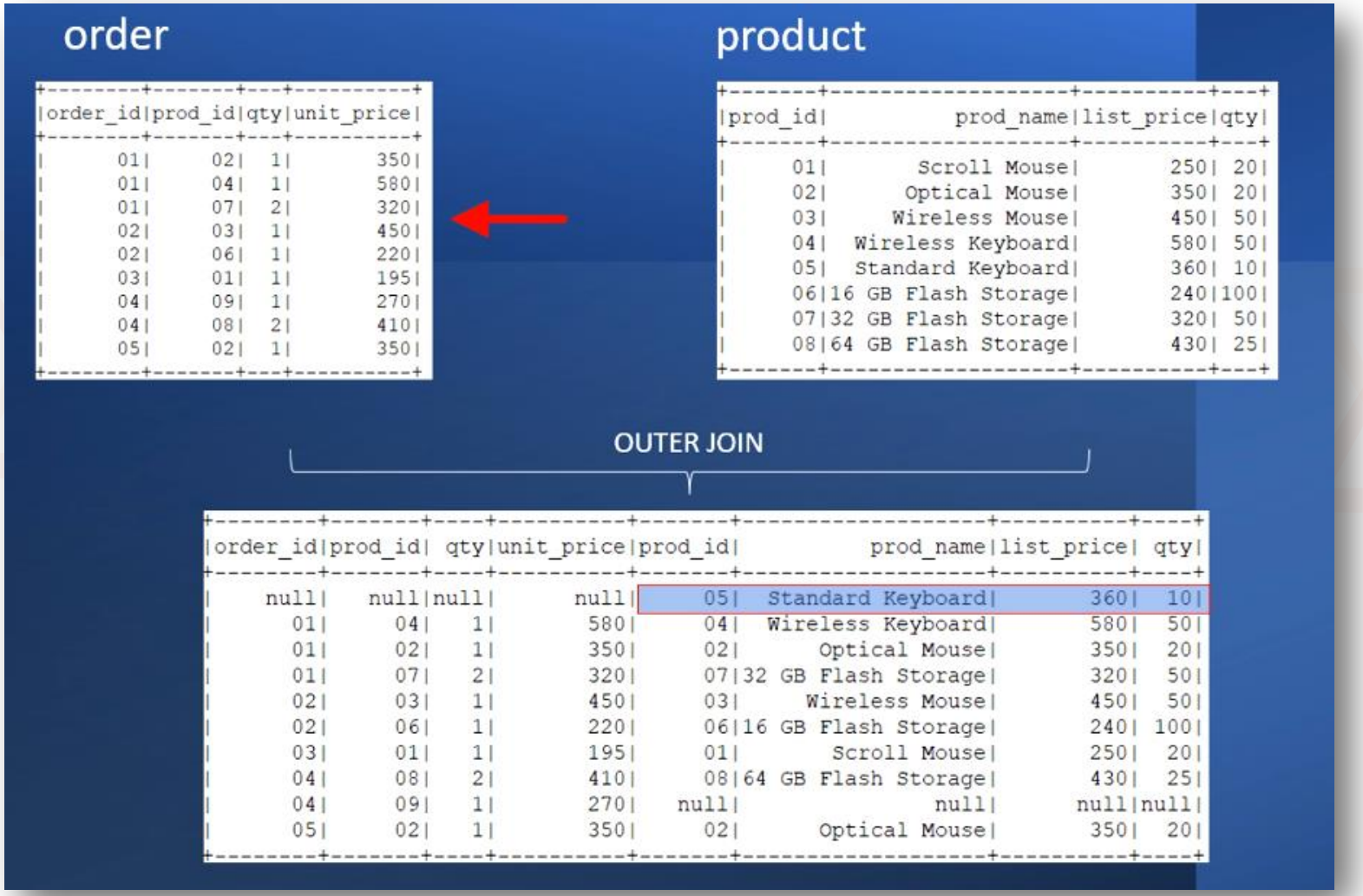
But remember, the matching records will always appear.

No matter if you use inner, full outer, left outer, or the right outer.

Join is all about matching the records on two sides. So the matching records always appear.

All four types of join that we discussed are there to play with the records that are not matching.

How do we lose the standard keyboard result? This record is present on the right side but not on the left side. We want all the records from the left side because we want to see all the sold items. But we do not care about the right side. So, the left join makes perfect sense here.



order

order_id	prod_id	qty	unit_price
01	02	1	350
01	04	1	580
01	07	2	320
02	03	1	450
02	06	1	220
03	01	1	195
04	09	1	270
04	08	2	410
05	02	1	350

product

prod_id	prod_name	list_price	qty
01	Scroll Mouse	250	20
02	Optical Mouse	350	20
03	Wireless Mouse	450	50
04	Wireless Keyboard	580	50
05	Standard Keyboard	360	10
06	16 GB Flash Storage	240	100
07	32 GB Flash Storage	320	50
08	64 GB Flash Storage	430	25

OUTER JOIN

order_id	prod_id	qty	unit_price	prod_id	prod_name	list_price	qty
null	null	null	null	05	Standard Keyboard	360	10
01	04	1	580	04	Wireless Keyboard	580	50
01	02	1	350	02	Optical Mouse	350	20
01	07	2	320	07	32 GB Flash Storage	320	50
02	03	1	450	03	Wireless Mouse	450	50
02	06	1	220	06	16 GB Flash Storage	240	100
03	01	1	195	01	Scroll Mouse	250	20
04	08	2	410	08	64 GB Flash Storage	430	25
04	09	1	270	null	null	null	null
05	02	1	350	02	Optical Mouse	350	20

Now I have changed the code to left join and displayed it below. You can see that we got the desired result.

Cmd 4

```
1  join_df = order_df.join(product_df, join_expr, "left")
```

Command took 0.05 seconds -- by prashant@scholarnest.com at 6/24/2022, 7:10:49 PM on demo-cluster

Cmd 5

```
1  join_df.orderBy("order_id").show()
```

▸ (3) Spark Jobs

order_id	prod_id	qty	unit_price	prod_id	prod_name	list_price	qty
01	02	1	350	02	Optical Mouse	350	20
01	07	2	320	07	32 GB Flash Storage	320	50
01	04	1	580	04	Wireless Keyboard	580	50
02	03	1	450	03	Wireless Mouse	450	50
02	06	1	220	06	16 GB Flash Storage	240	100
03	01	1	195	01	Scroll Mouse	250	20
04	08	2	410	08	64 GB Flash Storage	430	25
04	09	1	270	null	null	null	null
05	02	1	350	02	Optical Mouse	350	20

Command took 1.63 seconds -- by prashant@scholarnest.com at 6/24/2022, 7:10:50 PM on demo-cluster

However, I still see a small problem. Do you see these two nulls highlighted below? I don't like these nulls in my report. I know these values are missing on the right side, so we are getting these nulls. But still, can I get some meaningful value here. One option could be to use the product id. I mean, if we do not know the name, we can show the product id here. Similarly, we should consider showing the unit price if we do not know the list price.

Cmd 5

```
1  join_df.orderBy("order_id").show()
```

▸ (3) Spark Jobs

order_id	prod_id	qty	unit_price	prod_id	prod_name	list_price	qty
01	02	1	350	02	Optical Mouse	350	20
01	07	2	320	07	32 GB Flash Storage	320	50
01	04	1	580	04	Wireless Keyboard	580	50
02	03	1	450	03	Wireless Mouse	450	50
02	06	1	220	06	16 GB Flash Storage	240	100
03	01	1	195	01	Scroll Mouse	250	20
04	08	2	410	08	64 GB Flash Storage	430	25
04	09	1	270	null	null	null	null
05	02	1	350	02	Optical Mouse	350	20

Command took 1.63 seconds -- by prashant@scholarnest.com at 6/24/2022, 7:10:50 PM on demo-cluster

I added two with column transformations and used the coalesce() function.
The coalesce() function will take the first non-null value from the given list of columns.
So I will take the product name, but if that one is null, we take the product id.
Similarly, we fix the list price also.

Cmd 4

```
1  import pyspark.sql.functions as sf
2
3  join_df = order_df.join(product_df, join_expr, "left") \
4                    .withColumn("prod_name", sf.coalesce(product_df.prod_name, order_df.prod_id)) \
5                    .withColumn("list_price", sf.coalesce(product_df.list_price, order_df.unit_price))
```

Command took 0.05 seconds -- by prashant@scholarnest.com at 6/24/2022, 7:10:49 PM on demo-cluster

Cmd 5

Here is the output for the previous code. You can see that we have eliminated the desired two nulls in the joined Dataframe.

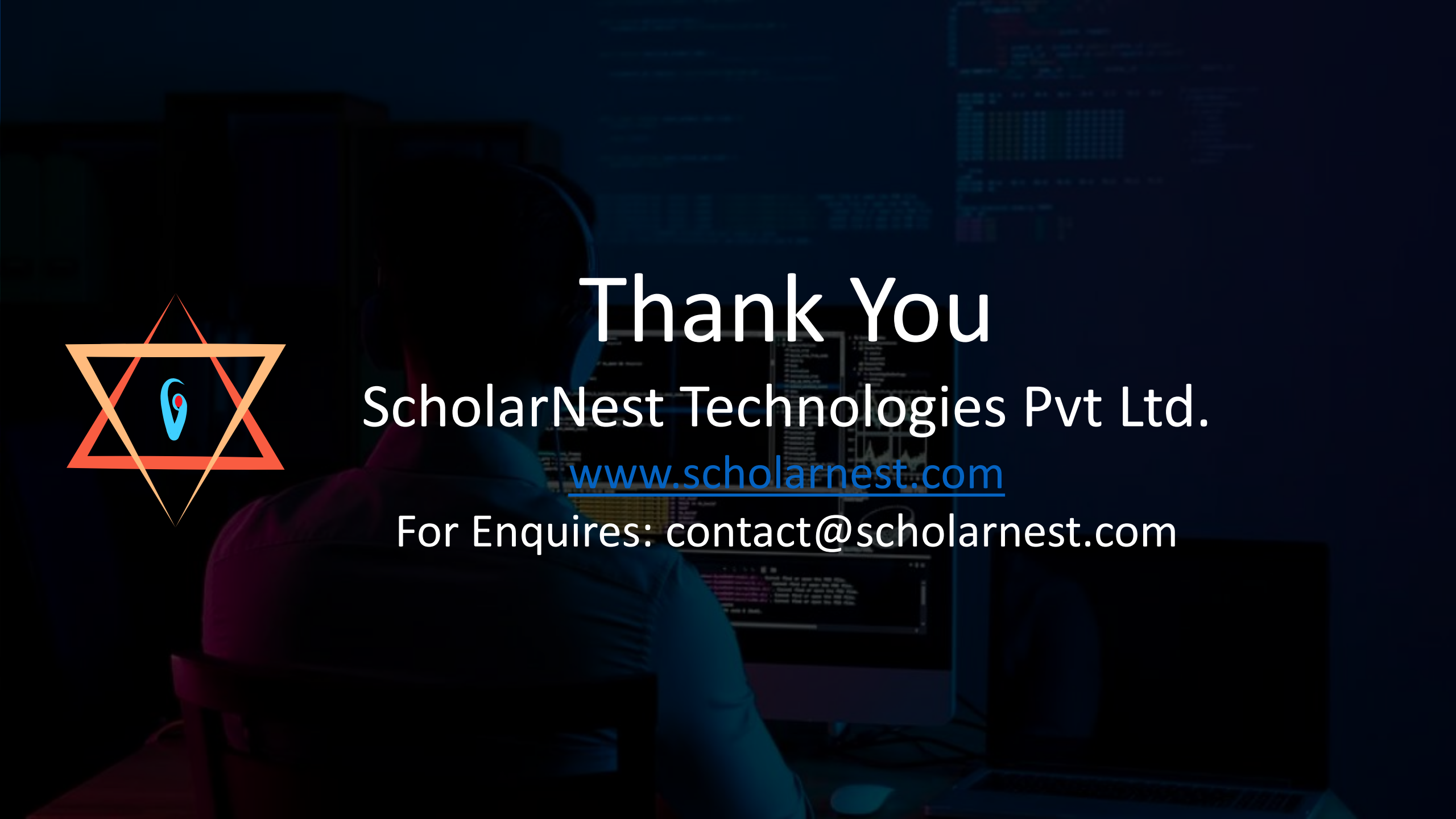
Cmd 5

```
1  join_df.orderBy("order_id").show()
```

▸ (3) Spark Jobs

```
+--------+-------+---+----------+-------+--------------------+----------+-----+
|order_id|prod_id|qty|unit_price|prod_id|           prod_name|list_price|  qty|
+--------+-------+---+----------+-------+--------------------+----------+-----+
|      01|     02|  1|       350|     02|       Optical Mouse|       350|   20|
|      01|     07|  2|       320|     07|32 GB Flash Storage|       320|   50|
|      01|     04|  1|       580|     04|  Wireless Keyboard|       580|   50|
|      02|     03|  1|       450|     03|      Wireless Mouse|       450|   50|
|      02|     06|  1|       220|     06|16 GB Flash Storage|       240|  100|
|      03|     01|  1|       195|     01|        Scroll Mouse|       250|   20|
|      04|     08|  2|       410|     08|64 GB Flash Storage|       430|   25|
|      04|     09|  1|       270|   null|                  09|       270| null|
|      05|     02|  1|       350|     02|       Optical Mouse|       350|   20|
+--------+-------+---+----------+-------+--------------------+----------+-----+
```

Command took 2.28 seconds -- by prashant@scholarnest.com at 6/24/2022, 7:52:53 PM on demo-cluster



Thank You

ScholarNest Technologies Pvt Ltd.

www.scholarnest.com

For Enquires: contact@scholarnest.com

Spark Azure Databricks

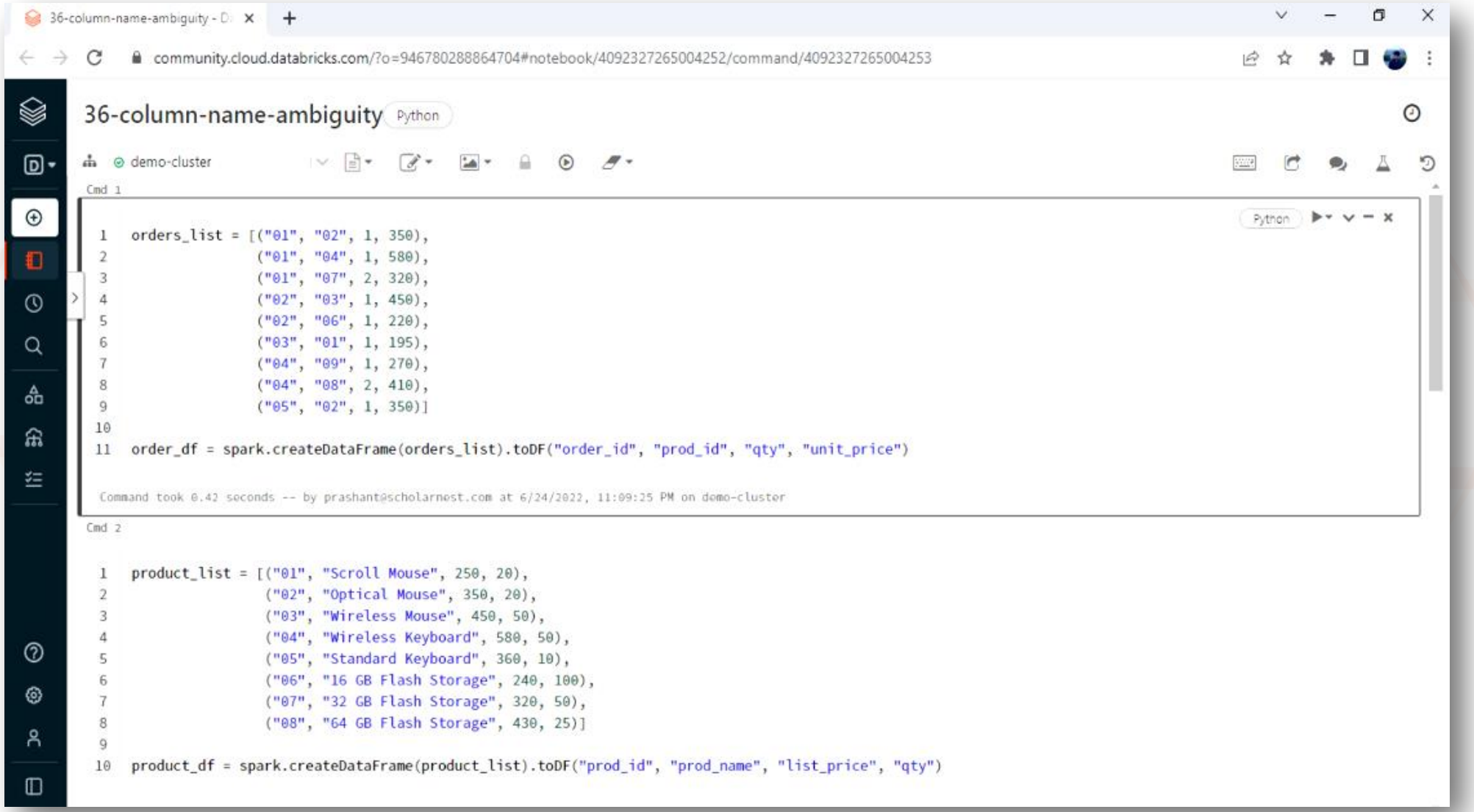
Databricks Spark Certification and Beyond

Module:
Joins

Lecture:
Column
Name
Ambiguity

Dataframe Joins and Column Name Ambiguity

Go to your Databricks workspace and create a clone of your previous notebook and renamed it **(Reference : 36-column-name-ambiguity.ipynb)**



I already have the previous code. I have my order_df and product_df. Then, I defined the join condition here and implemented left join. You might have noticed that I am using Dataframe dot column notation to refer to the column names. It started by defining the join condition. I have the join condition written as *order_df.prod_id == product_df.prod_id.* And this is necessary because we have the same column names in both the data frames. Also, look at these two with column expressions. I am using the Dataframe name with every column.

Look at the join_df dataframe. We have two prod_id columns. The first one comes from the order_df and the second one is from the product_df. Similarly, we have two quantities.
The first one is the order quantity, and the second one is the stock quantity. And this is a common problem with joins. You might get duplicate column names in the join result when you join two or more data frames.

Cmd 5

```
1  join_df.orderBy("order_id").show()
```

▸ (3) Spark Jobs

order_id	prod_id	qty	unit_price	prod_id	prod_name	list_price	qty
01	02	1	350	02	Optical Mouse	350	20
01	07	2	320	07	32 GB Flash Storage	320	50
01	04	1	580	04	Wireless Keyboard	580	50
02	03	1	450	03	Wireless Mouse	450	50
02	06	1	220	06	16 GB Flash Storage	240	100
03	01	1	195	01	Scroll Mouse	250	20
04	08	2	410	08	64 GB Flash Storage	430	25
04	09	1	270	null	09	270	null
05	02	1	350	02	Optical Mouse	350	20

You cannot refer to these duplicate names without the Dataframe name, spark will throw an error. You can see it below.
The error says : Reference 'prod_id' is ambiguous.
The join_df has two prod_id, and Spark doesn't know which prod_id you want to select.
And this is a common problem.

We have three approaches to fix the column ambiguity:

1. Prefix the column name with the Dataframe name
2. Rename columns before the join operation
3. Drop or rename the duplicate columns after the join

I am already using the first approach. I am prefixing the Dataframe name with the column names everywhere.
The second approach is renaming the columns before joining two data frames.
I mean, you already know you want to join order_df and product_df.
And you also know both the data frames have duplicate column names. So you might want to use the withColumnRenamed() on the product_df and rename all the duplicate columns and create a newly renamed Dataframe. Then you can use the renamed Dataframe for the join operation.
The third approach is to let join happen and fix the problem afterward. Let us see this third approach in action.

Look at the join_df.

You have two prod_id, and both have the same value. I can drop one of them. It is useless.

Cmd 5

```
1  join_df.orderBy("order_id").show()
```

▸ (3) Spark Jobs

order_id	prod_id	qty	unit_price	prod_id	prod_name	list_price	qty
01	02	1	350	02	Optical Mouse	350	20
01	07	2	320	07	32 GB Flash Storage	320	50
01	04	1	580	04	Wireless Keyboard	580	50
02	03	1	450	03	Wireless Mouse	450	50
02	06	1	220	06	16 GB Flash Storage	240	100
03	01	1	195	01	Scroll Mouse	250	20
04	08	2	410	08	64 GB Flash Storage	430	25
04	09	1	270	null	09	270	null
05	02	1	350	02	Optical Mouse	350	20

Command took 2.96 seconds -- by prashant@scholarnest.com at 6/24/2022, 11:09:25 PM on demo-cluster

I added a drop() method right after the join operation.
Now if we run the display() command, and it will show you only one prod_id.

You can see the updated Dataframe, one of the prod_id is gone. But we still have a duplicate qty column. But these two quantities are different. They are not duplicates. Only the column name is duplicated. But I can rename one of them. However, renaming after joining is tricky because the withColumnRenamed() will not work. So I have to use the withColumn() to copy the existing column with a new name and then drop() the older one.

Cmd 5

```
1  join_df.orderBy("order_id").show()
```

▸ (3) Spark Jobs

order_id	prod_id	qty	unit_price	prod_name	list_price	qty
01	02	1	350	Optical Mouse	350	20
01	07	2	320	32 GB Flash Storage	320	50
01	04	1	580	Wireless Keyboard	580	50
02	03	1	450	Wireless Mouse	450	50
02	06	1	220	16 GB Flash Storage	240	100
03	01	1	195	Scroll Mouse	250	20
04	08	2	410	64 GB Flash Storage	430	25
04	09	1	270	09	270	null
05	02	1	350	Optical Mouse	350	20

Command took 1.32 seconds -- by prashant@scholarnest.com at 6/24/2022, 11:20:42 PM on demo-cluster

Here is the code to remove the duplicity of qty column.

Cmd 4

```
1  import pyspark.sql.functions as sf
2
3  join_df = order_df.join(product_df, join_expr, "left") \
4                    .drop(product_df.prod_id) \
5                    .withColumn("stock_qty", product_df.qty) \
6                    .drop(product_df.qty) \
7                    .withColumn("prod_name", sf.coalesce(product_df.prod_name, order_df.prod_id)) \
8                    .withColumn("list_price", sf.coalesce(product_df.list_price, order_df.unit_price))
```

```
Command took 0.09 seconds -- by prashant@scholarnest.com at 6/24/2022, 11:35:52 PM on demo-cluster
```


We got the desired results. Now we don't have any duplicity in this Dataframe.

Cmd 5

```
1  join_df.orderBy("order_id").show()
```

▸ (3) Spark Jobs

order_id	prod_id	qty	unit_price	prod_name	list_price	stock_qty
01	02	1	350	Optical Mouse	350	20
01	07	2	320	32 GB Flash Storage	320	50
01	04	1	580	Wireless Keyboard	580	50
02	03	1	450	Wireless Mouse	450	50
02	06	1	220	16 GB Flash Storage	240	100
03	01	1	195	Scroll Mouse	250	20
04	08	2	410	64 GB Flash Storage	430	25
04	09	1	270	09	270	null
05	02	1	350	Optical Mouse	350	20

Command took 1.25 seconds -- by prashant@scholarnest.com at 6/24/2022, 11:37:50 PM on demo-cluster

Join often causes duplicate column names in the join result. You must be aware of this fact and fix this problem.
The approach to fixing this problem is straightforward.

1. Drop the duplicate columns because you do not need duplicate data in the same Dataframe.
2. If the columns are not duplicated, and only the name is conflicting, then you should rename them.

Now that we have finished joining two data frames and prepared a new Dataframe, can you achieve the following result?

order_id	prod_name	qty	unit_price	list_price	discount	amount
01	Optical Mouse	1	350	350	0	350
01	32 GB Flash Storage	2	320	320	0	640
01	Wireless Keyboard	1	580	580	0	580
02	16 GB Flash Storage	1	220	240	20	220
02	Wireless Mouse	1	450	450	0	450
03	Scroll Mouse	1	195	250	55	195
04	64 GB Flash Storage	2	410	430	40	820
05	Optical Mouse	1	350	350	0	350

Here is the code for the same and you can see that we achieved the desired output.

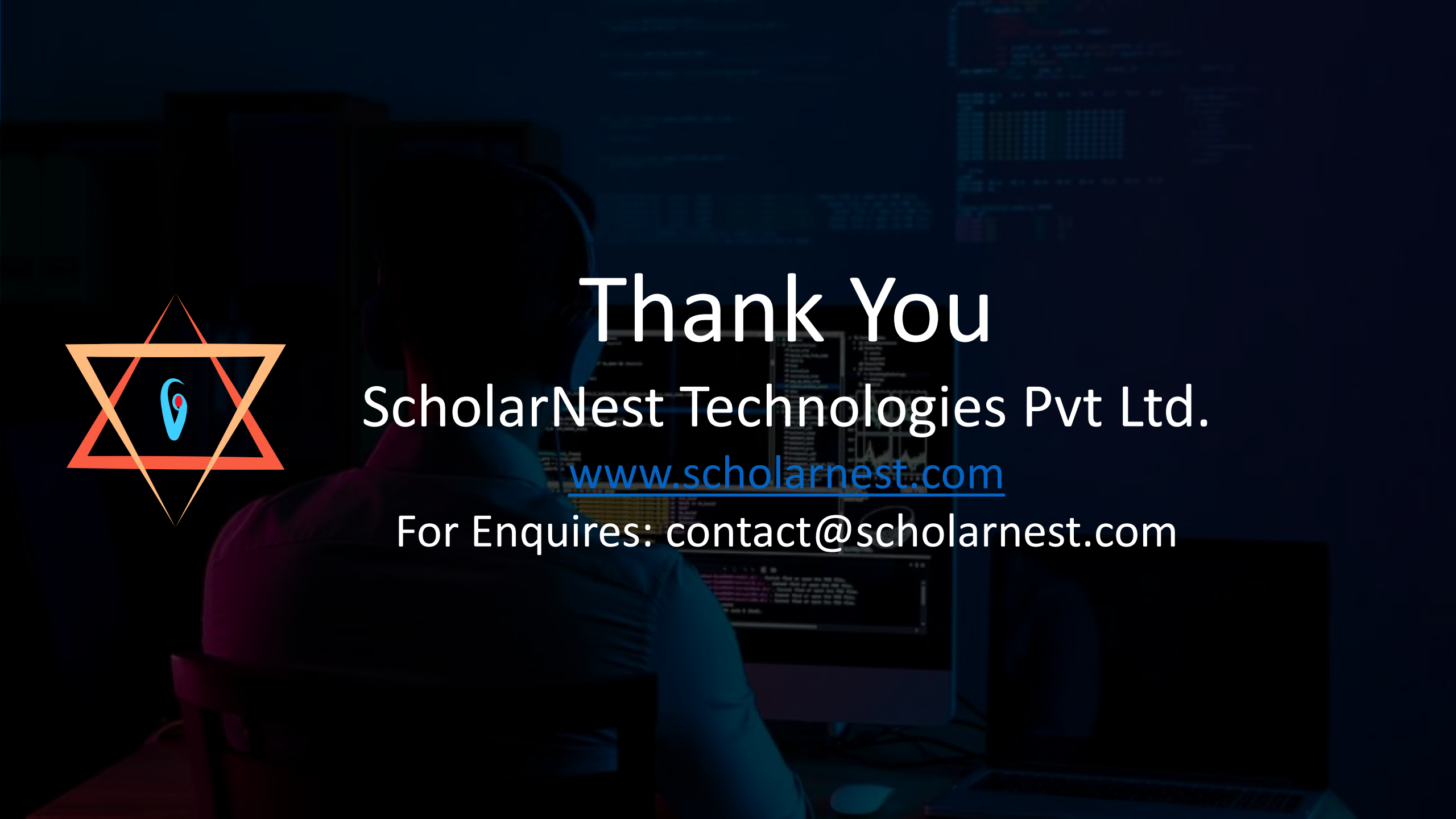
Cmd 6

```
1  join_df.selectExpr("order_id", "prod_name", "qty", "unit_price", "list_price",
2                     "(list_price - unit_price) * qty as discount", "qty * unit_price as amount") \
3          .orderBy("order_id") \
4          .show()
```

▸ (3) Spark Jobs

```
+--------+------------------+---+----------+----------+--------+------+
|order_id|         prod_name|qty|unit_price|list_price|discount|amount|
+--------+------------------+---+----------+----------+--------+------+
|      01|     Optical Mouse|  1|       350|       350|       0|   350|
|      01|32 GB Flash Storage|  2|       320|       320|       0|   640|
|      01| Wireless Keyboard|  1|       580|       580|       0|   580|
|      02|    Wireless Mouse|  1|       450|       450|       0|   450|
|      02|16 GB Flash Storage|  1|       220|       240|      20|   220|
|      03|      Scroll Mouse|  1|       195|       250|      55|   195|
|      04|64 GB Flash Storage|  2|       410|       430|      40|   820|
|      04|                09|  1|       270|       270|       0|   270|
|      05|     Optical Mouse|  1|       350|       350|       0|   350|
+--------+------------------+---+----------+----------+--------+------+
```

Command took 1.43 seconds -- by prashant@scholarnest.com at 6/24/2022, 11:44:31 PM on demo-cluster



Thank You

ScholarNest Technologies Pvt Ltd.

www.scholarnest.com

For Enquires: contact@scholarnest.com

Spark Azure Databricks

Databricks Spark Certification and Beyond

Module:
Joins

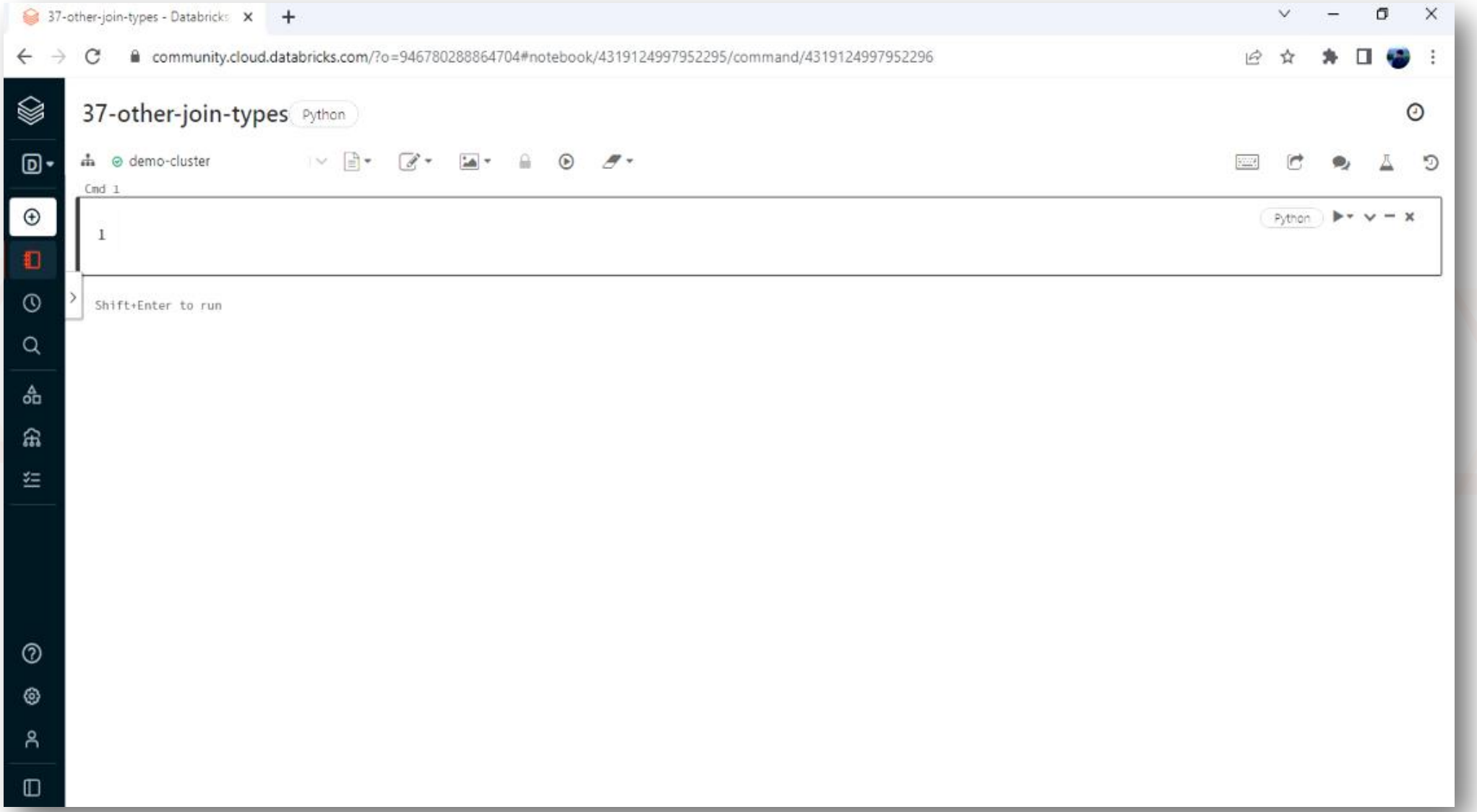
Lecture:
Other Join Types

More on Dataframe Joins

In this chapter, I will discuss the following types of joins in Apache Spark:

1. Cross Join
2. Semi Join
3. Anti Join

Go to your Databricks workspace and create a new notebook. **(Reference : 37-other-join-types.ipynb)**



Let's assume you are an electronics retail organization running multiple stores across the country. And you deal with a bunch of products. You have a product master of all the products shown below. So you have a product_df Dataframe. I am keeping it simple with only two fields: product id and product name.

Cmd 1

```
1   product_list = [("01", "Scroll Mouse"),
2                   ("02", "Optical Mouse"),
3                   ("03", "Wireless Mouse"),
4                   ("04", "Wireless Keyboard"),
5                   ("05", "Standard Keyboard"),
6                   ("06", "16 GB Flash Storage"),
7                   ("07", "32 GB Flash Storage"),
8                   ("08", "64 GB Flash Storage")]
9
10  product_df = spark.createDataFrame(product_list).toDF("prod_id", "prod_name")
```

Command took 0.84 seconds -- by prashant@scholarnest.com at 7/1/2022, 10:34:49 AM on demo-cluster

You also have stores across the country.
I have simulated that information and created another store Dataframe. And this one is also simple. We have two fields: store_id and store_name.

Cmd 2

```
1  stores_list = [("S1", "Down Town"),
2                 ("S2", "Cape Town"),
3                 ("S3", "China Town")]
4
5  stores_df = spark.createDataFrame(stores_list).toDF("store_id", "store_name")
```

Command took 0.12 seconds -- by prashant@scholarnest.com at 7/1/2022, 10:37:11 AM on demo-cluster

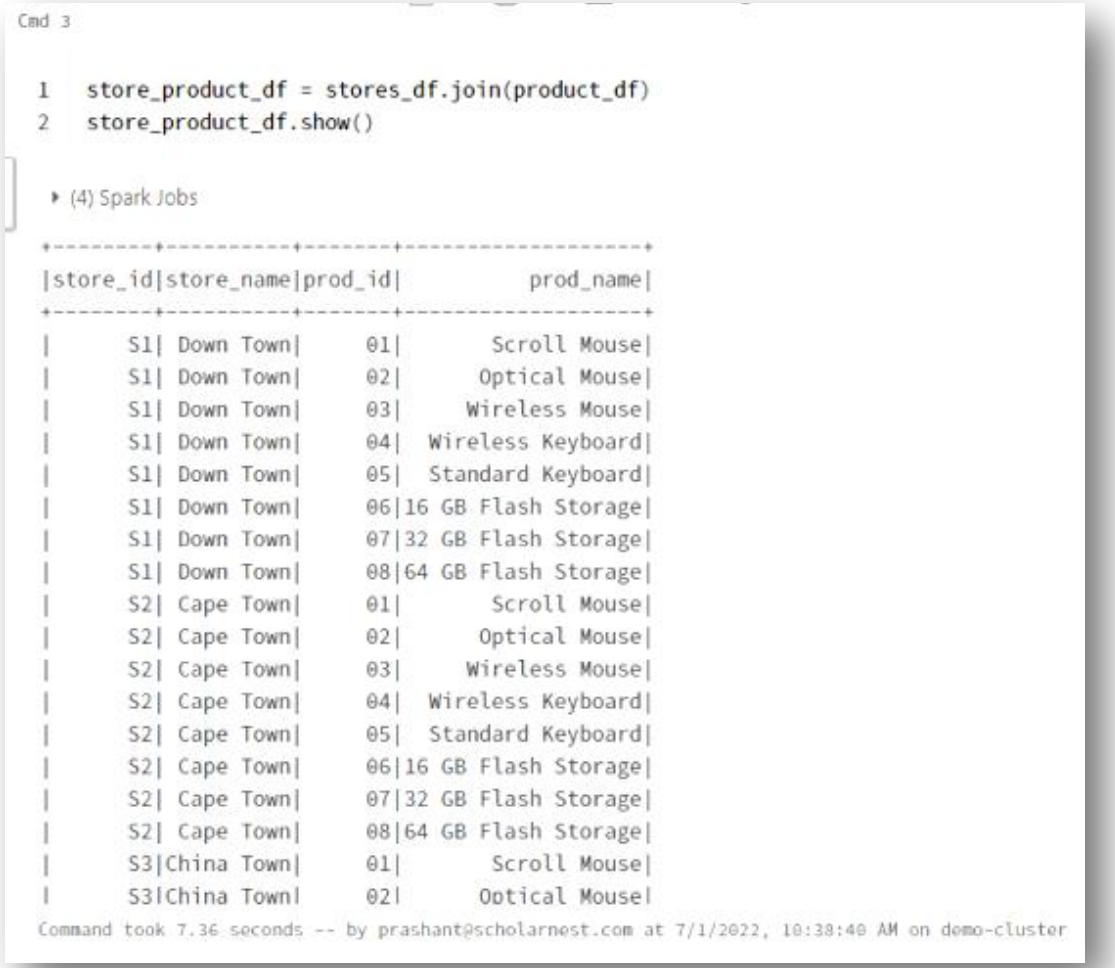
Now I give you a requirement to create a new Dataframe with all combinations of stores and products as shown below. So we have a list of all the products for the Down Town store. Then we have a list of all the products for the Cape Town store. And this goes on for all the stores and lists all the products for each store. This one is the most exhaustive list of store product combinations. All stores might not be selling all the products, but we want to make an exhaustive list of all possibilities.

store_id	store_name	prod_id	prod_name
S1	Down Town	01	Scroll Mouse
S1	Down Town	02	Optical Mouse
S1	Down Town	03	Wireless Mouse
S1	Down Town	04	Wireless Keyboard
S1	Down Town	05	Standard Keyboard
S1	Down Town	06	16 GB Flash Storage
S1	Down Town	07	32 GB Flash Storage
S1	Down Town	08	64 GB Flash Storage
S2	Cape Town	01	Scroll Mouse
S2	Cape Town	02	Optical Mouse
S2	Cape Town	03	Wireless Mouse

In order to achieve the given requirement, we need to use cross join.
I can cross join stores with the products, and Spark will match each store will all products to create this list.

store_id	store_name	prod_id	prod_name
S1	Down Town	01	Scroll Mouse
S1	Down Town	02	Optical Mouse
S1	Down Town	03	Wireless Mouse
S1	Down Town	04	Wireless Keyboard
S1	Down Town	05	Standard Keyboard
S1	Down Town	06	16 GB Flash Storage
S1	Down Town	07	32 GB Flash Storage
S1	Down Town	08	64 GB Flash Storage
S2	Cape Town	01	Scroll Mouse
S2	Cape Town	02	Optical Mouse
S2	Cape Town	03	Wireless Mouse

Here is the code for cross join. The stores_df is my left Dataframe, and I join it with the product_df. The product_df becomes my right Dataframe. I am not setting any join conditions because the cross join doesn't need any conditions. It will match all rows from the left Dataframe with all rows of the right Dataframe. And this matching is unconditional. So we do not need any condition. We do not need to specify the join type also. Because unconditional join is always a cross join. And you can also see the output Dataframe.



Cross join is exhaustive, and it is costly for large data frames.
So you should avoid performing cross joins.
You might never need to implement cross join in real life.
However, you can do it on small data frames when you have a genuine requirement.

Here is another requirement.
You are given an order data. You have order_id, store_id, prod_id, quantity, and unit price.

Cmd 4

```
1  orders_list = [("01", "S1", "02", 1, 350),
2                 ("01", "S1", "04", 1, 580),
3                 ("01", "S1", "07", 2, 320),
4                 ("02", "S2", "03", 1, 450),
5                 ("02", "S2", "06", 1, 220),
6                 ("03", "S2", "01", 1, 195),
7                 ("04", "S1", "09", 1, 270),
8                 ("04", "S1", "08", 2, 410),
9                 ("05", "S3", "02", 1, 350)]
10
11 order_df = spark.createDataFrame(orders_list).toDF("order_id", "store_id", "prod_id", "qty", "unit_price")
```

```
Command took 0.14 seconds -- by prashant@scholarnest.com at 7/1/2022, 10:45:37 AM on demo-cluster
```


Now I asked you the following requirement.
You have an exhaustive list of all store and item possibilities as shown below. This data is the combination of all stores with all products. In an ideal case, all our stores must be selling all items. But I want to filter this list and see only those not sold by the stores.

```
Cmd 5

1  store_product_df.show()

▸ (4) Spark Jobs

+--------+----------+-------+--------------------+
|store_id|store_name|prod_id|           prod_name|
+--------+----------+-------+--------------------+
|      S1| Down Town|     01|        Scroll Mouse|
|      S1| Down Town|     02|      Optical Mouse|
|      S1| Down Town|     03|     Wireless Mouse|
|      S1| Down Town|     04|  Wireless Keyboard|
|      S1| Down Town|     05|  Standard Keyboard|
|      S1| Down Town|     06|16 GB Flash Storage|
|      S1| Down Town|     07|32 GB Flash Storage|
|      S1| Down Town|     08|64 GB Flash Storage|
|      S2| Cape Town|     01|        Scroll Mouse|
|      S2| Cape Town|     02|      Optical Mouse|
|      S2| Cape Town|     03|     Wireless Mouse|
|      S2| Cape Town|     04|  Wireless Keyboard|
|      S2| Cape Town|     05|  Standard Keyboard|
|      S2| Cape Town|     06|16 GB Flash Storage|
|      S2| Cape Town|     07|32 GB Flash Storage|
|      S2| Cape Town|     08|64 GB Flash Storage|
|      S3|China Town|     01|        Scroll Mouse|
|      S3|China Town|     02|      Optical Mouse|
Command took 3.56 seconds -- by prashant@scholarnest.com at 7/1/2022, 10:46:06 AM on demo-cluster
```


So, we want to filter a Dataframe based on the availability of records in another Dataframe.
It means for the given example, I want to filter the store_product_df if the stores do not sell these products.
I already have order data. So I can filter out records from store_product_df if a record for the store/product combination does not exist in the orders_df.
If you know SQL, you already know how to do it.
SQL gives you the EXISTS and NOT EXISTS clause to do these things. Spark does not have EXISTS clause. But we have semi and anti joins for the same.

Semi-Join → EXISTS
Anti-Join → NOT EXISTS

This requirement wants us to see only those records that do not exist in the orders.
Why? Because we want to see what is not sold by the store.
So this one is an anti join requirement.

Here is the code for the given requirement. I defined the join condition first. It matches the records between left and right data frames. And I want to match the records where store_id and prod_id column values are the same.
Then I am starting with the store_product_df. That one is my left Dataframe. And it is essential to choose your left data frame correctly. I want to see records from the store_product_df so that one is my left Dataframe. The order_df is the right data frame because I do not want to see data from the orders. I will simply use the orders to implement the filter. Anti join is always left_anti join. So whatever data you want to see must be the left data frame.

Cmd 6

```
1  join_expr = (store_product_df.store_id == order_df.store_id) & \
2              (store_product_df.prod_id == order_df.prod_id)
3
4  store_product_df.join(order_df, join_expr, "left_anti") \
5                  .orderBy("store_id", "prod_id") \
6                  .show()
```


You can see the output of our code, we got the desired list of records that is not sold by the store.

Cmd 6

```
1  join_expr = (store_product_df.store_id == order_df.store_id) & \
2              (store_product_df.prod_id == order_df.prod_id)
3
4  store_product_df.join(order_df, join_expr, "left_anti") \
5                  .orderBy("store_id", "prod_id") \
6                  .show()
```

▸ (3) Spark Jobs

```
+--------+----------+-------+--------------------+
|store_id|store_name|prod_id|           prod_name|
+--------+----------+-------+--------------------+
|      S1| Down Town|     01|        Scroll Mouse|
|      S1| Down Town|     03|      Wireless Mouse|
|      S1| Down Town|     05|   Standard Keyboard|
|      S1| Down Town|     06|16 GB Flash Storage|
|      S2| Cape Town|     02|       Optical Mouse|
|      S2| Cape Town|     04|   Wireless Keyboard|
|      S2| Cape Town|     05|   Standard Keyboard|
|      S2| Cape Town|     07|32 GB Flash Storage|
|      S2| Cape Town|     08|64 GB Flash Storage|
|      S3|China Town|     01|        Scroll Mouse|
|      S3|China Town|     03|      Wireless Mouse|
|      S3|China Town|     04|   Wireless Keyboard|
|      S3|China Town|     05|   Standard Keyboard|
|      S3|China Town|     06|16 GB Flash Storage|
|      S3|China Town|     07|32 GB Flash Storage|
|      S3|China Town|     08|64 GB Flash Storage|
```


The semi-join or the left_semi join is opposite to the anti join.
So if you want to see records from your left data frame that exists in the right Dataframe, you can use the semi join, as shown below.

Cmd 7

```
1  join_expr = ["store_id", "prod_id"]
2
3  store_product_df.join(order_df, join_expr, "left_semi") \
4                  .orderBy("store_id", "prod_id") \
5                  .show()
```

▸ (3) Spark Jobs

```
+--------+-------+----------+------------------+
|store_id|prod_id|store_name|         prod_name|
+--------+-------+----------+------------------+
|      S1|     02| Down Town|      Optical Mouse|
|      S1|     04| Down Town|  Wireless Keyboard|
|      S1|     07| Down Town|32 GB Flash Storage|
|      S1|     08| Down Town|64 GB Flash Storage|
|      S2|     01| Cape Town|       Scroll Mouse|
|      S2|     03| Cape Town|     Wireless Mouse|
|      S2|     06| Cape Town|16 GB Flash Storage|
|      S3|     02|China Town|      Optical Mouse|
+--------+-------+----------+------------------+
```

Command took 4.07 seconds -- by prashant@scholarnest.com at 7/1/2022, 10:53:16 AM on demo-cluster

Everything is the same as the anti-join code except for two things:

1. The join condition looks different.
2. We are using left_semi join, so the result shows only those records that also exist in the orders.

Cmd 7

```
1  join_expr = ["store_id", "prod_id"]
2
3  store_product_df.join(order_df, join_expr, "left_semi") \
4                  .orderBy("store_id", "prod_id") \
5                  .show()
```

▸ (3) Spark Jobs

```
+--------+-------+----------+-------------------+
|store_id|prod_id|store_name|          prod_name|
+--------+-------+----------+-------------------+
|      S1|     02| Down Town|      Optical Mouse|
|      S1|     04| Down Town|  Wireless Keyboard|
|      S1|     07| Down Town|32 GB Flash Storage|
|      S1|     08| Down Town|64 GB Flash Storage|
|      S2|     01| Cape Town|       Scroll Mouse|
|      S2|     03| Cape Town|     Wireless Mouse|
|      S2|     06| Cape Town|16 GB Flash Storage|
|      S3|     02|China Town|      Optical Mouse|
+--------+-------+----------+-------------------+
```

Command took 4.07 seconds -- by prashant@scholarnest.com at 7/1/2022, 10:53:16 AM on demo-cluster

Let me explain the join condition. The standard approach for defining the join condition is like this:

```
join_expr = (store_product_df.store_id == order_df.store_id) & \
            (store_product_df.prod_id == order_df.prod_id)
```

This one is the proper approach to defining the join condition. It is highly readable, and it makes good sense. However, we have a lot of typing in this approach. So Spark gave you a shortcut. Just give me a list of column names. Like this:

```
join_expr = ["store_id", "prod_id"]
```

And that's what I used in the semi-join example shown in the previous slide. So I am giving a column list. And spark will convert it to the proper join condition like this.

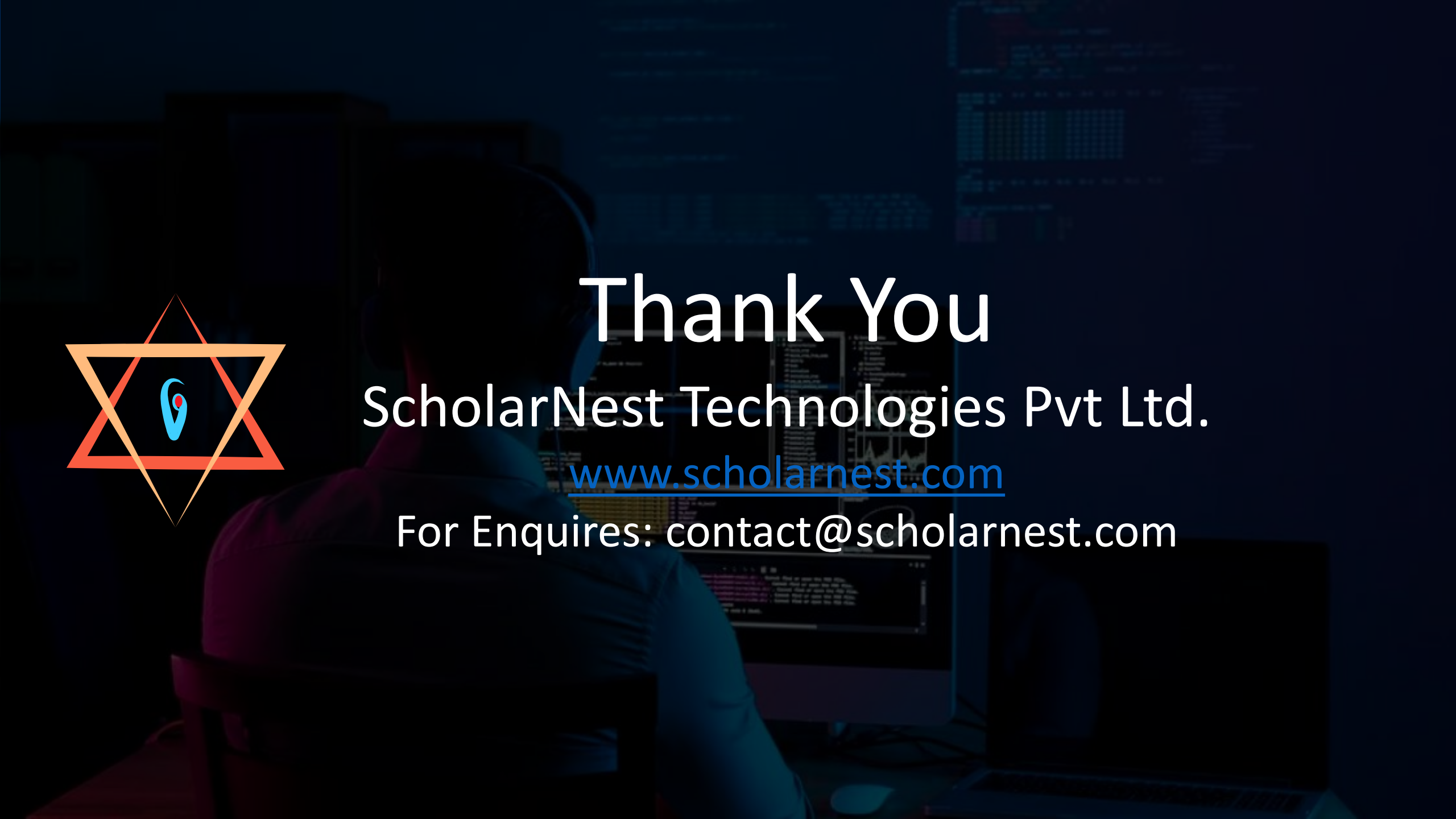
```
join_expr = (store_product_df.store_id == order_df.store_id) & \
            (store_product_df.prod_id == order_df.prod_id)
```

But remember. This shortcut is not allowed in all cases.

The shortcut approach works when you meet the following conditions:

1. The key column names are the same in both the data frames. In our case, we have store_id and prod_id, and these column names are the same in both data frames. So we meet the first criteria.
2. You want to implement the equality operator. The translated condition will be store_id in the left data frame is equal to the store_id in the right data frame. And that's what we want to meet the second criteria.
3. All the key column conditions will be combined using the AND operator. And that's what we want.

So If you have the same key columns in two data frames, you want to check for equality and combine them with AND condition. You can simply pass the list of key columns, and Spark will automatically create a proper condition for you.



Thank You

ScholarNest Technologies Pvt Ltd.

www.scholarnest.com

For Enquires: contact@scholarnest.com