MASTERING SQL JOINS

2023

A QUICK HANDBOOK ON MASTERING SQL JOINS WITH PRACTICAL EXERCISES

DANE WADE

COPYRIGHT

This is an intellectual work by author Dane Wade and dataceps.com to help people learn SQL programming language.

Book Cover And Content Plan Designed By Dane Wade.

For more information, Email the author at : danewade@dataceps.com.

TABLE OF CONTENTS

INTRODUCTION.....1

DOWNLOADING DATASETS AND INSTALLING THE SQL SERVER.....3

WHAT ARE SQL JOINS, AND HOW DOES IT HELP?.....5

CROSS JOIN.....9

Practice questions on cross join.....12

INNER JOIN.....15

Practice questions on cross join.....22

OUTER JOINS.....23

Left Outer Join.....23

Practice questions on Left Outer Join.....28

Right Outer Join.....29

Practice questions on Right Outer Join.....33

Full Outer Join.....34

Practice questions on Full Outer Join.....39

CALCULATED COLUMN IN THE QUERIES WITH JOINS.....40

How to create and utilize calculated columns in SQL queries with joins.....41

HANDS-ON EXERCISES.....50
Set I.....50
Set II.....52

ANSWERS.....58
Set I.....58
Set II.....60

BIBLIOGRAPHY.....74

INTRODUCTION

The data management industry is at its peak at this moment. You and I are so are witnessing one of the biggest revolutions in the history of mankind. Data, which is also considered as oil of the present time, is more valuable than ever.

New high-paying jobs are being created, and the data management industry is at its all-time high!! When I began my career, I never knew that I would end up learning about DBMS or SQL.

During my college years, the one subject that I hated the most was database management; I never liked that subject as it seemed like a subject with numbers all over the place. And I have always hated numbers since my childhood.

I consider myself a creative person, and for me, numbers were never my thing. But my bad relationship with DBMS didn't end in my college.

When I joined my first company, I was onboarded on a project that required SQL, DWH, and DBMS concepts and skills. For me, it was a nightmare. I always believed that the problem was with me. Especially when I saw queries that other developers created. It overwhelmed me.

Those queries were using JOINS with multiple tables, and it never made sense to me. I read thousands of articles on JOINS and saw hundreds of videos. But nothing was helping me to understand this concept.

When I saw other senior developers who were working on queries with multiple JOINS and analyzing data fast using JOINS in their queries, I thought that the problem was with me.

But later, I realized that It was not completely my fault. I never had the correct information. And the people who were teaching this concept didn't lay the correct foundation to make this concept easy to understand.

I never had a step-by-step plan that helped me to practice these concepts and make some good progress on this. As a result, whenever I had to build queries that required JOINS.I found myself doing everything to avoid that task .

But later on, I realized that this would not work for long, and In order for me to grow in my career and in life, I have to understand and master this concept.

I found out that I am not the only person who is facing this problem. There are many people who are also facing the same issue.
In my Facebook community, I asked people to comment on the concept they are struggling the most with. And the most common answer was SQL JOINS!!!

Although I have also mentioned this concept in detail in my book *Simple SQL: Beginner's Guide To Master SQL And Boost Career (Zero To Hero)*

But I wanted to create a resource that is dedicated to this concept only with hands-on exercises. That can also act as a quick study refresher guide.

DOWNLOADING DATASETS AND INSTALLING THE SQL SERVER

To take a practice-based approach, I would recommend you to install the SQL Server version as per your Operating System.

Follow the steps mentioned in the below webpages and carefully follow the steps for a smooth hassle free installation.

For Windows Users : If you have windows operating system . Please follow the steps mentioned in the below link carefully to install SQL Server on your system smoothly.

https://dataceps.com/the-definitive-guide-sql-server-express-edition-installation-on-windows-10-step-by-step/

For Macbook Users : If you have Mac operating system . Please follow the steps mentioned in the below link carefully to install SQL Server on your system smoothly.

https://dataceps.com/the-definitive-guide-sql-server-express-edition-installation-on-mac/

DATASET INSTALLATION:

Once you have installed SQL Server in your machine successfully. Then please download the FREE dataset Installation guide. Visit the below mentioned link to grab this dataset.

https://dataceps.com/dataset-download-sql-joins/

This Mastering SQL book is of no use, if you don't download this dataset and practice the hands-on practice questions mentioned in this book.

WHAT ARE SQL JOINS, AND HOW DOES IT HELP?

In this chapter, you will learn about the basics of SQL joins, the significance of using SQL Joins, and how it helps in data management. In my opinion, understanding the basics before mastering any skill is what one should aim for.

We are in the data management industry, dealing with huge amounts of data. The ultimate purpose of this whole industry is to collect, manage, and extract useful meaning and insights from the raw data.

Raw data in itself is of no use; it can't help anyone.

And in order to extract meaning, relationships, and insights from the data. We require a combination of different datasets.

For example :

An e-commerce company wants to open a new warehouse in 5 different cities. At this moment, they only have a limited amount of funds to open these 5 warehouses.

Wouldn't it be better if they opened the warehouses in the cities from where most of their customers order? Not only will this move decrease the shipping time, but it will also motivate other new customers to shop more from this e-commerce company.

To take this big step of opening 5 high investment warehouses, the company requires meaningful insight. This insight or information is very valuable for this e-commerce company.

And in order to get this information, we must have access to 3 types of information.

1) Customer data
2) Order data
3) customer's location data

Also, the relationship between these three entities can help the business in extracting the correct insight. These datasets independently are of no use to the organization. However, if we compare and combine these datasets, We can extract this valuable insight easily.
JOINS in SQL are basically used to accomplish this purpose only. It helps in *combining and contrasting different datasets* and helps us in extracting high-quality, meaningful insights from the data.

As you might be familiar already, about SQL that is used in Relational Databases. And SQL is completely based on Relational Algebra concepts.
Basically, this relational algebra serves as a theoretical basis for the relational database and SQL.

In simple terms, the concepts of relational algebra were used in order to create SQL concepts that we see and use today.

So if you understand the underlying relational algebra concepts, you will be in a much better position to understand the SQL JOIN concept.

If I talk about JOINS, the concept that a SQL JOIN is based on the basis relational algebra concept– *Cartesian Product*.

CARTESIAN PRODUCT:
In relational algebra, a *cartesian product* or *cross product* is actually a combination of every record present in the first table with every record present in the second table.

Let me explain this with an example :

Suppose we have 2 tables, table 1: Students and table 2: Subjects.
(*See the below tables for reference*)

Students
Dane
Sarah
Patrick

Subjects
Maths
English

Figure 3.1 *Students and Subjects tables*

Now the cartesian product of these two tables/datasets can also be represented as (Students X Subjects), and the output of the cartesian or cross product will be:

Students x Subjects	
Dane	Maths
Sarah	Maths
Patrick	Maths
Dane	English
Sarah	English
Patrick	English

Figure 3.2 *cartesian product of tables students and subjects*

If you look closely in *figure 3.2*, you will observe that each record that is present in Table 1 (Students) is combined with each record present in Table 2 (Subjects).

This is the **core concept** that is working behind the scenes when we are combing datasets. Now if you understand this concept, then understanding other joins will be much easier for you.

There are different types of Joins out there, But I will be discussing the most common ones in this short guide. These JOINS are—CROSS JOIN,INNER JOIN,LEFT JOIN,RIGHT JOIN,FULL JOIN.I will discuss about all these joins in upcoming chapters.

CROSS JOIN

In the last chapter, I explained how the *cross-product* concept of relational algebra serves as the foundation for the JOINS concept in SQL. If you understand the cross product concept, then understanding the concept of cross join will be a cakewalk for you.

But in case, If there is even a single doubt in your mind about the concept of cross product, then I would recommend you revisit the concept once again in the previous chapter and start with the concept of JOINS.

Now, let us get back to the main topic –CROSS JOIN.

CROSS JOIN is a type of SQL JOIN that simply does a CROSS PRODUCT of records from Table 1 and Table 2. (Considering we have CROSS JOIN applied on two tables –Table 1 and Table 2).

So, If there are 3 records present in Table 1 and there are 4 records present in Table 2. Then the final dataset after implementing the CROSS JOIN will have 3 X 4 records, i.e. 12 total combinations of records. This is because each record present in Table 1 will create a combination with each record present in Table 2.

The Key Idea or purpose of this CROSS JOIN is to present ALL POSSIBLE COMBINATIONS of datasets on which the JOIN is implemented.

Let me explain this concept with a simple example:

Suppose we have 2 tables: Animal and Food.
The 1st table is Animal has 2 records :

A_Id	Animal_Name
1	Dog
2	Cat

Figure 4.1 *Animal table*

And table 2 is Food :

F_Id	Food	Price
1	Fish	5.00
2	Chicken	10.00
3	Milk	5.00

Figure 4.2 *Food table*

The output of the CROSS JOIN will be:

F_Id	Food	Price	A_Id	Animal_Name
1	Fish	5.00	1	Dog
2	Chicken	10.00	1	Dog
3	Milk	5.00	1	Dog
1	Fish	5.00	2	Cat
2	Chicken	10.00	2	Cat
3	Milk	5.00	2	Cat

Figure 4.3 *Cross join of animal and food table*

As you can see above, the result of the CROSS JOIN is “ALL POSSIBLE COMBINATIONS of datasets” of the records present in both tables.

Although on the surface, it seems that the practical implementation of the CROSS JOIN concept might not be much useful. And I agree developers use other JOIN types much more than this one, but still, this CROSS JOIN concept is sometimes used to analyze data.

It's like a tool in your pocket that you can use when required. I think at this point, you might have understood the concept of CROSS JOIN and Cartesian Product.

The reason I am saying this is because the following chapters on the other types of JOINS are dependent on this concept.

But before that, I would recommend you go through all the questions mentioned in this chapter. This will help you to LEVEL UP before you start the next concept.

PRACTICE QUESTIONS ON CROSS JOIN

Question 1:
Implement CROSS JOIN on the below mentioned tables.

Loans_Master Table:

Loan_Id	Customer_Id	Loan_Amount	Interest_Rate	Terms_Of_Repayment
1	104	15000	0.0522	24
2	103	15000	0.0651	12
3	102	7510	0.0445	18
4	101	18000	0.0675	6
5	105	22000	0.095	36

Dataceps_Customers Table:

Customer_Id	First_Name	Last_Name	Phone_Number
101	Dane	Wade	555-1234
102	Jane	Smith	555-5678
103	Michael	Hoffman	555-9998
104	Sarah	Ritter	555-3456
105	David	Gabier	555-6410

Question 2:

Implement CROSS JOIN on the below mentioned tables.

Dataceps_Performance_Review Table:

Performance_Review_ID	Employee_ID	Review_Date	Final_Rating	Final_Comments
1	102	44576	4	Good performance during the year, Can be Improved.
2	101	44620	3	meets expectations.
3	103	44630	5	Superb performance, exceeding expectations.
4	105	44656	2	Needs to be organized and improvement required in communication skills.
5	104	44701	4	Good Overall Work! Demonstrates strong teamwork skills.

Dataceps_Employees Table:

Employee_ID	First_Name	Last_Name	Email_ID	Department_ID	Emp_Salary
101	Dane	Wade	danewade@dataceps.com	1	7666
102	Jane	Smith	jane.smith@dataceps.com	2	6200
103	Michael	Hoffman	michael.hoffman@dataceps.com	1	6500
104	Emily	Ritter	emily.ritter@dataceps.com	3	5100
105	Sarah	Brown	sarah.brown@dataceps.com	2	5821

Question 3:

Implement CROSS JOIN on the below mentioned tables.

Advertisers_Master Table:

Advertiser_ID	Advertiser_Name	Email_Id	Phone
1	DigiXMedia	ads@digixmedia.com	7894123166
2	TRX Ventures	ads@trxventures.com	9864890453
3	HiROI Ads Agency	ads@hiroiads.com	9811537891
4	Dataceps Digital Solutions	ads@dcdsoltion.com	5516542310

Campaigns_Master Table:

Campaign_ID	Campaign_Name	Begin_Date	End_Date	Total_Budget	Advertiser_ID
1	Summer Clearance Sale	01-06-2022	30-06-2022	5430	1
2	Holiday Special Final Offer	01-12-2022	31-12-2022	8020	1
3	Product Launch Sale	01-09-2022	30-09-2022	3300	2
4	Year-End Stock Clearance	15-12-202	15-01-2023	10050	2
5	Spring Collection	01-03-2022	30-04-2022	6450	3
6	Back-to-College Sale	01-08-2022	31-08-2022	4067	4

INNER JOIN

Before we dive into this chapter, I hope you have practiced the questions in the last chapter. Understanding the CROSS JOIN concept is quite essential before we begin this chapter.

I know you're a person with lots of dedication and willingness to succeed. But as an author, It's my duty to take you on this Learning journey step-by-step so that you make the most of it.

Now, let's get back to the INNER JOIN concept. Technically, "INNER JOIN is used to select records that have matching records present in both the tables".

Let me explain it in a simple way, suppose we have two different tables/datasets, and we want to extract records that are present in both datasets.In such cases, *INNER JOIN helps to extract only the matching records*.

INNER JOIN requires a specific condition—Both the tables used in INNER JOIN should have a common column. This common column (present in both tables) should have matching data types and similar values. And the matching of records will happen on the basis of this column only.

So whenever you're JOINING two different tables, make sure to identify the column ON which this INNER JOIN will happen.

The syntax of INNER JOIN looks something like this:

```
SELECT ColumnNames
FROM [Table1]
INNER JOIN [Table2]
ON Table1.ColumnA=Table2.ColumnA;
```

In the above query, you can see that the JOIN is happening ON the ColumnA from table1 and ColumnA from table2.

The illustration for INNER JOIN is usually represented like this:

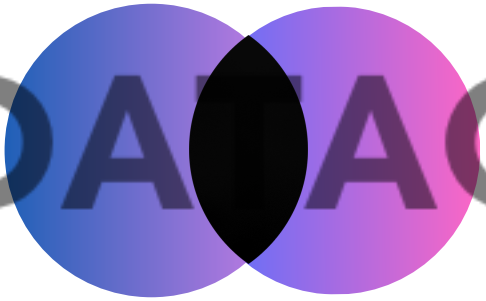

Figure 5.1 *Table 1 INNER JOIN Table 2*

As you can see in this illustration that in, the final OUTCOME is marked in “Black” here. That is the dataset that is COMMON and MATCHING in both of these tables. Hence, the INNER JOIN should return only this dataset.

Let me explain this concept with an example:

DataCeps_Students Table :

Student _Id	Student_Name
1	Liz
2	Dane
3	Sarah

Figure 5.2 *DataCeps_Students table*

Food_Choices Table:

Food_Id	Food_Name	Student_Id
F1	Burger	2
F2	Pizza	1
F3	Noodles	1
F4	Beans & Rice	2

Figure 5.3 *Food_Choices table*

To implement the INNER JOIN between these above two tables or datasets.

The first thing to do here is to identify the column ON which the INNER join condition needs to be implemented.

In this particular example, the common column here is Student_Id From the table “Student” and Student_Id From the table “Food_Choices”.

The query to extract the data from these two tables using INNER JOIN will look something like this :

```
SELECT *
FROM Student S
INNER JOIN Food_Choices FC
ON S.Student_Id =FC.Student_Id
```

And the final outcome after implementing INNER JOIN will be:

Student_Id	Student_Name	Food_Id	Food_Name	Student_Id
2	Dane	F1	Burger	2
1	Liz	F2	Pizza	1
1	Liz	F3	Noodles	1
2	Dane	F4	Beans And Rice	2

Figure 5.4 *INNER JOIN on Student and Food_Choices table*

Now let me explain what happened behind the curtains.

how we reached to this outcome? How all of this happened internally in the system? And how the concept of *cartesian product* is implemented here.

The result of cartesian product of these two tables is :

Student_Id	Student_ Name	Food_Id	Food_Name	Student_Id
1	Liz	F1	Burger	2
1	Liz	F2	Pizza	1
1	Liz	F3	Noodles	1
1	Liz	F4	Beans And Rice	2
2	Dane	F1	Burger	2
2	Dane	F2	Pizza	1
2	Dane	F3	Noodles	1
2	Dane	F4	Beans And Rice	2
3	Sarah	F1	Burger	2
3	Sarah	F2	Pizza	1
3	Sarah	F3	Noodles	1
3	Sarah	F4	Beans And Rice	2

Figure 5.5 *Cartesian Product of Student and Food_Choices table*

The total count of the records after doing the cartesian product is 3X 4, i.e.12 records.

But as you know that in the INNER JOIN, we have a special condition ON the column that is c*ommon in both tables.*

In this case, that condition is on the column Student_Id from the table Student and column Student_Id from the table Food_Choices.

Now, this condition used in INNER JOIN, will allow only records that have MATCHING Student_ID present in both the tables involved here.

Let me explain this with an illustration :

Student_Id	Student_Name	Food_Id	Food_Name	Student_Id
1	Liz	F1	Burger	2
1	Liz	F2	Pizza	1
1	Liz	F3	Noodles	1
1	Liz	F4	Beans And Rice	2
2	Dane	F1	Burger	2
2	Dane	F2	Pizza	1
2	Dane	F3	Noodles	1
2	Dane	F4	Beans And Rice	2
3	Sarah	F1	Burger	2
3	Sarah	F2	Pizza	1
3	Sarah	F3	Noodles	1
3	Sarah	F4	Beans And Rice	2

Figure 5.6 *Cartesian Product results highlighted based on INNER JOIN concept*

As you can see in the above illustration, the records with matching Student_ID are highlighted in this final outcome of the Cartesian product.

This is why, eventually, in the final outcome after implementing the INNER JOIN is this:

Student_Id	Student_ Name	Food_Id	Food_Name	Student_Id
2	Dane	F1	Burger	2
1	Liz	F2	Pizza	1
1	Liz	F3	Noodles	1
2	Dane	F4	Beans And Rice	2

***Figure 5.7** INNER JOIN Join Student and Food_Choices table*

I hope, at this point, you understand the concept of INNER JOIN.

INNER JOIN is the most used JOIN. As you just observed, it helps in finding the MATCHING records present in both the table on which the INNER JOIN is implemented.

Whether you want to combine data from multiple datasets or you're doing some data analysis, you can take advantage of this JOIN type.

Before you move on to the next chapter, Please go through all the questions mentioned after this chapter. Reading these concepts will not help you out until you practice them on your own system.

PRACTICE QUESTIONS ON INNER JOIN

Question 1:
Implement INNER JOIN on the below mentioned tables.

1. Loans_Master Table
2. Dataceps_Customers Table

Question 2:

Implement INNER JOIN on the below mentioned tables.

1. Dataceps_Employees Table.
2. Dataceps_Performance_Review Table.

Question 3:

Implement INNER JOIN on the below mentioned tables.

1. Advertisers_Master Table.
2. Campaigns_Master Table

OUTER JOINS

JOINS are not only used to combine datasets to extract only the matching records. As I explained in the last chapter, INNER JOIN in SQL is used to display the records that match / present in both the tables on which the INNER JOIN is implemented.

In this chapter, I will introduce another type of JOIN where the final output can have UNMATCHED records. These types of JOINS are called OUTER JOINS.
There are 3 different types of OUTER JOINS in SQL, but the main concept of all 3 of them is the same – The result of OUTER JOIN will have unmatched records.

These 3 different types of OUTER JOINS are:
1) LEFT OUTER JOIN or LEFT JOIN
2) RIGHT OUTER JOIN or RIGHT JOIN
3) FULL OUTER JOIN or FULL JOIN

Let me explain each JOIN type...

LEFT OUTER JOIN

LEFT OUTER JOIN or popularly known as LEFT JOIN, displays ALL the records present in the LEFT table and ONLY the MATCHING records present in the RIGHT Table.
Please read the above paragraph again!
This JOIN from the "OUTER JOIN family" also requires a common column to establish the JOIN condition. Else it will be difficult to find what's matching or what's not matching. Right?

The criterion for selecting this common column is the same as we discussed in the previous chapter. This common column (present in both tables) should have matching data types and similar values.
The syntax of LEFT JOIN looks something like this:

```
SELECT ColumnNames
FROM [Table1]
LEFT JOIN [Table2]
ON Table1.ColumnA=Table2.ColumnA;
```

In the above syntax, you can see that the JOIN is happening ON the ColumnA from Table1 and ColumnA from Table2.

Below is a figure that clearly illustrates the concept of LEFT JOIN:

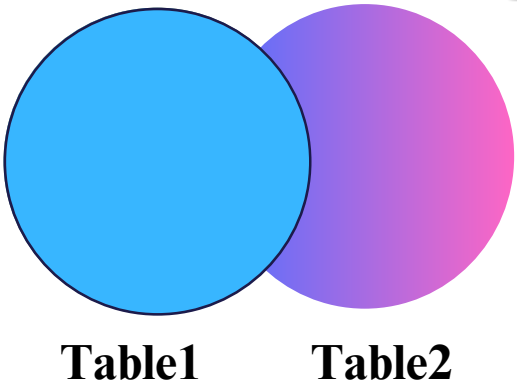

Figure 6.1 *LEFT JOIN on Table 1 and Table 2*

Now let me explain this concept with an example.

I am taking the same tables that I used in INNER JOIN chapter, However, I have added a couple more records to make this concept easier to understand. And the final dataset is :

DataCeps_Students table:

Student _Id	Student_Name
1	Liz
2	Dane
3	Sarah

Figure 6.2 *Dataceps_Student table*

Food_Choices table:

Food_Id	Food_Name	Student_Id
F1	Burger	2
F2	Pizza	1
F3	Noodles	1
F4	Beans & Rice	2
F5	Pasta	NULL

Figure 6.3 *Food_Choices table*

Let's first see what will be Cartesian Product output of the above two tables will be :

Student_Id	Student_Name	Food_Id	Food_Name	Student_Id
1	Liz	F1	Burger	2
1	Liz	F2	Pizza	1
1	Liz	F3	Noodles	1
1	Liz	F4	Beans & Rice	2
1	Liz	F5	Pasta	NULL
2	Dane	F1	Burger	2
2	Dane	F2	Pizza	1
2	Dane	F3	Noodles	1
2	Dane	F4	Beans & Rice	2
2	Dane	F5	Pasta	NULL
3	Sarah	F1	Burger	2
3	Sarah	F2	Pizza	1
3	Sarah	F3	Noodles	1
3	Sarah	F4	Beans & Rice	2
3	Sarah	F5	Pasta	NULL

Figure 6.4 *Cartesian Product Of Dataceps_Student and Food_Choices table with highlighted records based on LEFT JOIN concept*

Now if we go by the rules of LEFT JOIN, We should *SELECT all records that are present in the LEFT table and ONLY the matching records present in the RIGHT table*.As you can see in the cartesian product output, All the highlighted records that have a MATCHING record present in the RIGHT TABLE will be in the final Output of the LEFT JOIN.

In addition to this, the highlighted record with Student_ID =3 is not in the Right table (Food_Choices).

Therefore, To satisfy the condition of LEFT JOIN – include ALL records present in LEFT TABLE to be present in the FINAL Output. We need to represent the values in the RIGHT column with NULL values. Basically, this tells us that the particular record doesn't have a matching value in the RIGHT table.

Hence the FINAL Output of the LEFT JOIN query will be :

Student_Id	Student_Name	Food_Id	Food_Name	Student_Id
1	Liz	F2	Pizza	1
1	Liz	F3	Noodles	1
2	Dane	F1	Burger	2
2	Dane	F4	Beans & Rice	2
3	Sarah	NULL	NULL	NULL

Figure 6.5 *Results of LEFT JOIN on Dataceps_Student and Food_Choices table*

I hope you understand this concept of LEFT OUTER JOIN or LEFT JOIN clearly now.

Please go through the practice questions mentioned just after this section, to understand this concept on a much deeper level.

PRACTICE QUESTIONS ON LEFT JOIN

Question 1:
Implement LEFT JOIN on the below mentioned tables.

1. Loans_Master Table.
2. Dataceps_Customers Table.

Question 2:

Implement LEFT JOIN on the below mentioned tables.

1. Dataceps_Performance_Review Table.
2. Dataceps_Employees Table.

Question 3:

Implement LEFT JOIN on the below mentioned tables.

1. Advertisers_Master Table.
2. Campaigns_Master Table.

RIGHT OUTER JOIN

Alright, now you have progressed a lot. I am sure you understood the last section on the LEFT OUTER JOIN and practiced the questions diligently. This section is all about the RIGHT OUTER JOIN or popularly known as RIGHT JOIN and how it works.

Technically, *The RIGHT OUTER JOIN selects ALL the records that are in the RIGHT table and then ONLY the matching records present in the LEFT table.*

This JOIN From the "OUTER JOIN family" also requires a common column to establish the JOIN condition. Else it will be difficult to find what's matching or what's not matching.

Right?

The syntax of RIGHT JOIN looks something like this:

```
SELECT ColumnNames
FROM [Table1]
RIGHT JOIN [Table2]
ON Table1.ColumnA=Table2.ColumnA;
```

In the above syntax, you can see that the JOIN is happening ON the ColumnA from Table1 and ColumnA from Table2.

Below is a figure that clearly illustrates the concept of RIGHT JOIN:

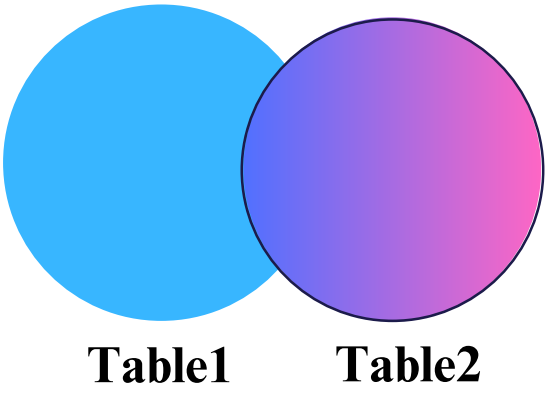

Figure 6.6 *Illustration for RIGHT JOIN*

This JOIN is also part of the OUTER JOIN family. Therefore, it also requires a common column on which the condition is implemented.

The criterion for selecting this common column is the same as we discussed in the previous section on Left Join. This common column (present in both tables) should have matching data types and similar values. Now let me explain this concept with an example.

Let's consider the same tables DataCeps_Students and Food_Choices table again.

Student _Id	Student_Name
1	Liz
2	Dane
3	Sarah

Figure 6.7 *DataCeps_Students table*

Food_Id	Food_Name	Student_Id
F1	Burger	2
F2	Pizza	1
F3	Noodles	1
F4	Beans & Rice	2
F5	Pasta	NULL

Figure 6.8 *Food_Choices table*

The cartesian product of the above 2 tables will be:

Student_Id	Student_Name	Food_Id	Food_Name	Student_Id
1	Liz	F1	Burger	2
1	Liz	F2	Pizza	1
1	Liz	F3	Noodles	1
1	Liz	F4	Beans & Rice	2
1	Liz	F5	Pasta	NULL
2	Dane	F1	Burger	2
2	Dane	F2	Pizza	1
2	Dane	F3	Noodles	1
2	Dane	F4	Beans & Rice	2
2	Dane	F5	Pasta	NULL
3	Sarah	F1	Burger	2
3	Sarah	F2	Pizza	1
3	Sarah	F3	Noodles	1
3	Sarah	F4	Beans & Rice	2
3	Sarah	F5	Pasta	NULL

Figure 6.9 *Cartesian Product Of Dataceps_Student and Food_Choices table with highlighted records based on RIGHT JOIN concept*

Now if we go by the rules of RIGHT JOIN, We should *SELECT all records that are present in the RIGHT table and ONLY the matching records present in the RIGHT table.*

As you can see in the *figure 6.9*, I have highlighted the records that are present in the RIGHT table and have MATCHING values present in the LEFT table.

Also, you can see that the values that are only present in the RIGHT table and are not in the LEFT table will be populating NULL values in the final dataset against such records.

In addition to this, You can see that the record with NULL student_Id value is not matching with any value in the LEFT table. But as per the RIGHT JOIN concept, that record will appear in the final result even if it doesn't have a matching value in the LEFT table.

Hence, the final output after implementing the rules of RIGHT JOIN will be:

Student_Id	Student_Name	Food_Id	Food_Name	Student_Id
2	Dane	F1	Burger	2
1	Liz	F2	Pizza	1
1	Liz	F3	Noodles	1
2	Dane	F4	Beans & Rice	2
NULL	NULL	F5	Pasta	NULL

Figure 6.10 *Dataceps_Students RIGHT JOIN Food_Choices table*

I hope you understand this concept of RIGHT OUTER JOIN or RIGHT JOIN clearly now.
Please go through the practice questions mentioned just after this section to understand this concept on a much deeper level.

PRACTICE QUESTIONS ON RIGHT JOIN

Question 1:
Implement RIGHT JOIN on the below mentioned tables.

1. Loans_Master Table.
2. Dataceps_Customers Table.

Question 2:
Implement RIGHT JOIN on the below mentioned tables.

1. Dataceps_Performance_Review Table.
2. Dataceps_Employees Table.

Question 3:
Implement RIGHT JOIN on the below mentioned tables.

1. Advertisers_Master Table.
2. Campaigns_Master Table.

FULL OUTER JOIN

Let's now talk about the final member of the OUTER JOIN family – FULL OUTER JOIN.

FULL OUTER JOIN or FULL JOIN is a type of JOIN that *displays ALL the records present in both tables irrespective of the fact whether the records are MATCHING or NOT-MATCHING based on the common column present in both tables.*

In other words, FULL OUTER JOIN reflects the data that is present in both LEFT and RIGHT join.

The syntax of FULL OUTER JOIN looks something like this:


```
SELECT ColumnNames
FROM [Table1]
FULL OUTER JOIN [Table2]
ON Table1.ColumnA=Table2.ColumnA;
```

In the above syntax, you can see that the JOIN is happening ON ColumnA from Table1 and ColumnA from Table2.

Below is a figure that clearly illustrates the concept of FULL OUTER JOIN:

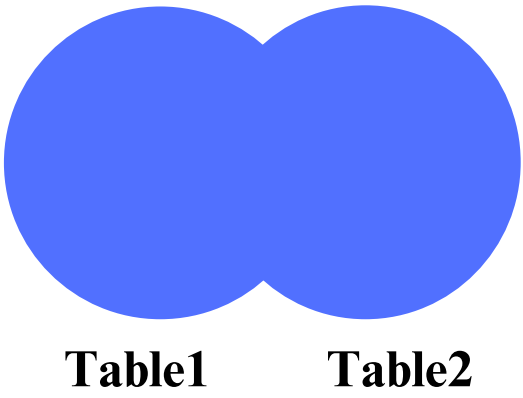

Figure 6.11 *Table1 FULL OUTER JOIN Table2*

This JOIN is also part of the OUTER JOIN family. Therefore, it also requires a common column on which the condition is implemented.

The criterion for selecting this common column is the same as we discussed in the previous section on Left Join. This common column (present in both tables) should have matching data types and similar values.
Now let me explain this concept with an example.
Let's consider the same tables DataCeps_Students and Food_Choices table again.

Student _Id	Student_Name
1	Liz
2	Dane
3	Sarah

Figure 6.12 *Dataceps_Students Table*

Food_Id	Food_Name	Student_Id
F1	Burger	2
F2	Pizza	1
F3	Noodles	1
F4	Beans & Rice	2
F5	Pasta	NULL

Figure 6.13 *Food_Choices Table*

The cartesian product of the table will be:

Student_Id	Student_Name	Food_Id	Food_Name	Student_Id
1	Liz	F1	Burger	2
1	Liz	F2	Pizza	1
1	Liz	F3	Noodles	1
1	Liz	F4	Beans & Rice	2
1	Liz	F5	Pasta	NULL
2	Dane	F1	Burger	2
2	Dane	F2	Pizza	1
2	Dane	F3	Noodles	1
2	Dane	F4	Beans & Rice	2
2	Dane	F5	Pasta	NULL
3	Sarah	F1	Burger	2
3	Sarah	F2	Pizza	1
3	Sarah	F3	Noodles	1
3	Sarah	F4	Beans & Rice	2
3	Sarah	F5	Pasta	NULL

Figure 6.14 *Cartesian Product Of Dataceps_Student and Food_Choices table with highlighted records based on FULL JOIN concept*

As per the FULL JOIN rules, The final output must have *MATCHED and UNMATCHED records based on the common column Student_Id* (ON which the JOIN is implemented).

As you can see in the final output of cartesian product in *figure 6.14*, I have highlighted the matching Student_Id with a different color.

The idea is to find the matching student_Id in the other table, and then that record will be eligible as an output for the FULL OUTER JOIN.

Let me now explain the process to find out the NOT MATCHING records.

To get the non-matching records, we need to examine both tables. As you can see, in this example. The record with student_Id doesn't have any MATCHING record present in the other table (Food_Choices) table.
And the record with a NULL value also doesn't have any MATCHING record present in the Dataceps_Students table.Hence, The core idea for the records where we don't have any MATCHING values present in either of the tables.

The final output will have NULL values assigned for the values that originate from the other table.

The final output of the FULL OUTER join will be:

Student_Id	Student_Name	Food_Id	Food_Name	Student_Id
1	Liz	F2	Pizza	1
1	Liz	F3	Noodles	1
2	Dane	F1	Burger	2
2	Dane	F4	Beans & Rice	2
3	Sarah	NULL	NULL	NULL
NULL	NULL	F5	Pasta	NULL

Figure 6.15 *FULL JOIN on Dataceps_Student and Food_Choices table*

As you can see in the above *figure 6.15*, For the record with Student_Id =3 Of Dataceps_Student table. The column values Food_Id, Food_Name, and Student_Id from the other table have NULL values.

And the 6th record with NULL student_Id from the table Food_Choices. The column values Student_Id and Student_Name are coming as NULL.

I hope you understand this concept clearly now. Please go through the practice questions after this section to make this concept more concrete now.

PRACTICE QUESTIONS ON FULL JOIN

Question 1:

Implement RIGHT JOIN on the below mentioned tables.

1. Loans_Master Table.
2. Dataceps_Customers Table.

Question 2:

Implement FULL JOIN on the below mentioned tables.

1. Dataceps_Performance_Review Table.
2. Dataceps_Employees Table.

Question 3:

Implement FULL JOIN on the below mentioned tables.

1. Advertisers_Master Table.
2. Campaigns_Master Table.

CALCULATED COLUMN IN THE QUERIES WITH JOINS

Welcome to the magical world of Calculated Columns! In this chapter, you'll explore how to transform, clean, enrich, manipulate, and refine the data in the columns of SQL queries where SQL JOIN is implemented. These calculations on the columns can be done with the help of aggregate functions, conditional statements, string functions, etc.

The raw data is of no use if it's just sitting on tables. Right?

Unless we are not able to JOIN tables and pull out the correct information in a meaningful way, then what's the purpose of storing data in a database?
When we transform or clean the data, it starts to make sense.

Suppose I can see the marks of each student after combining them with another table using SQL Joins. Then calculating the overall percentage of those students will bring meaningful insight.
However, Calculating percentages is totally a mathematical operation that uses a mathematical formula:

Percentage= Marks Obtained/Total Marks * 100

We can use this formula in a simple SQL query that has JOINS or even without JOINS.

Similarly, we can combine the First_Name, Middle_Name, and Last_Name to give us a Full_Name of a student.

We can compute these values dynamically based on our custom requirements. These columns are actually created virtually with the help of other columns in the SELECT statements. That means we are not storing them physically anywhere.These dynamically created columns help in data enrichment, data cleaning, and data transformation.

This concept of calculated columns actually assists in data analysis, data transformations for data warehouse, creating reports and dashboards for business, and the list is endless.

HOW TO CREATE AND UTILIZE CALCULATED COLUMNS IN SQL QUERIES WITH JOINS

In this section of this chapter, You will explore how to actually implement these calculated columns on queries where SQL Joins are implemented.
To create a calculated column or derived column, the first thing we need is to have is expression.

An expression is completely based on the requirement or desired transformation.

This expression is calculated for each record present in the dataset, creating the derived column for each record dynamically. Then in order to give a name to the newly created derived column in the dataset, we can apply an alias to the column. Let me explain this concept with an example.
Let's take 2 tables to understand this concept – Dataceps_Performance_Review and Dataceps_Employee.

Performance_ Review_ID	Employee_ID	Review_ Date	Final_Rating	Final_Comments
1	102	44576	4	Good performance during the year, Can be Improved.
2	101	44620	3	meets expectations.
3	103	44630	5	Superb performance, exceeding expectations.
4	105	44656	2	Needs to be organized and improvement reqiured in communication skills.
5	104	44701	4	Good Overall Work ! Demonstrates strong teamwork skills.

Figure 7.1 *Dataceps_Performance_Review Table*

Employee_ID	First_Name	Last_Name	Email_ID	Departm ent_ID	Emp_Salary
101	Dane	Wade	danewade@dataceps.com	1	7666
102	Jane	Smith	jane.smith@dataceps.com	2	6200
103	Michael	Hoffman	michael.hoffman@dataceps.com	1	6500
104	Emily	Ritter	emily.ritter@dataceps.com	3	5100
105	Sarah	Brown	sarah.brown@dataceps.com	2	5821

Figure 7.2 *Dataceps_Employees Table*

Now, let's suppose the requirement is to display the employee rating with the final comments along with their name. However, the name should be a combination of First Name and Last Name Separated by a blank and Should appear as Employee_Name in the final dataset.

```
SELECT
CONCAT(DE.First_Name,' ',DE.Last_Name) AS
Employee_Full_Name,
DPR.Final_Rating,
DPR.Final_Comments
FROM [Mastering_SQL].[dbo].[Dataceps_Employees] DE
INNER JOIN [Mastering_SQL].[dbo].
[Dataceps_Performance_Review] DPR
ON DE.[Employee_ID]=DPR.[Employee_ID]
```

In the above query, you can see that the table Dataceps_Employee is aliased as DE, and [Dataceps_Performance_Review] is aliased as DPR.

And the transformation is the concatenation of First_Name and Last_Name separated by a blank. To create this concatenation, I used a string function CONCAT here.

Creating derived columns or calculated columns in a query with JOINS is quite similar to a query where there is NO JOIN.

The only thing that needs to be considered while creating these expressions using different operators, functions, or conditional statements is to mention the correct column name with the relevant table aliases.
As you can see in the above example, I am pulling the First_Name and Last_Name from DE that is table Dataceps_Employee. And then the rest of the fields from the DPR table alias.

There are many ways this expression is created to create calculated columns. We can utilize the different SQL System defined functions in order to create these expressions that eventually help us in transforming data.

I will be explaining only a few relevant examples for each function in this section due to the limited scope of the book if you want to learn in-depth about these different types of functions in detail. Then I would recommend learning it in my book *Simple SQL: Beginner's Guide To Master SQL And Boost Career (Zero To Hero)*.
I have a dedicated chapter on different types of SQL Functions in detail, with examples and practice questions as well.

SQL System defined functions that help in creating calculated functions are mentioned below:

1) Mathematical Functions: Mathematical functions allow to create expressions to carry out mathematical operations. Some of these operations are –addition, subtraction, division, multiplication, etc.

For Example: If I want to display the First_Name, Last_Name, and the salary of an employee incremented by 20%, who has a final rating of 4.
To create this new derived column, I need to create a mathematical expression.This expression calculates 20% of the salary of those employees and add them up to the current salary, and then displays it in the final results in a new column.

Hence, the formula is :

New Salary = (0.2 X Current_Salary)+ Current_Salary

Where, ***(0.2 X Current_Salary)*** calculates the 20% of the current salary.

Let me now, translate this into SQL code:

```
SELECT DE.First_Name,
DE.Last_Name,
((0.2)*DE.Emp_Salary+DE.Emp_Salary) AS Incremented_Salary
FROM [Mastering_SQL].[dbo].[Dataceps_Employees] DE
INNER JOIN
[Mastering_SQL].[dbo].[Dataceps_Performance_Review] DPR
ON DE.[Employee_ID]=DPR.[Employee_ID]
WHERE DPR.Final_Rating=4
```

As you can see in the above query, I used mathematical functions present in SQL to translate this expression into SQL code.

2) String Functions: String functions helps in carrying out the transformation and manipulation of text data. Some of these operations are changing cases of string values, trimming extra characters, matching patterns, and concatenation of string values.

For Example: Display the names of employees in the upper case along with their Final Performance review comments.

The SQL code for this will be:

```
SELECT
UPPER(DE.First_Name)AS Emp_First_Name,
UPPER(DE.Last_Name) AS Emp_Last_Name,
DPR.Final_Comments
FROM [Mastering_SQL].[dbo].[Dataceps_Employees] DE
INNER JOIN
[Mastering_SQL].[dbo].[Dataceps_Performance_Review] DPR
ON DE.[Employee_ID]=DPR.[Employee_ID]
```

As you can see in this query, the First_Name and Last_Name in uppercase are derived from the First_Name and Last_Name columns from the table Dataceps_Employee (alias DE). And the Final_Comments column is being derived from the table Dataceps_Performance_Review (alias DPR).

3) Date And Time Functions: In SQL, Date And Time functions allow to perform operations and manipulations on the date and time column values. Some of these date and time functions are – DATEPART, DATEDIFF, DATEDIFF, etc.

For example: Write a SQL query to retrieve the first name of the employees along with the extracted month from the review date. The extracted month should be displayed as a month name instead of a numeric value. That means 1 represents January, 2 represents February, and so on.

SQL code for this will be:

```
SELECT Emp.First_Name  AS Employee_Name,
DPR.Final_Rating,
DATENAME(MONTH, DPR.Review_Date)  AS Review_Month
FROM Dataceps_Performance_Review DPR
JOIN Dataceps_Employees Emp
ON DPR.Employee_ID = Emp.Employee_ID
```


4) Conditional Statements: Conditional Statements in SQL helps in performing different operations and actions based on the different custom conditions. These custom conditions are created on the basis of desired output or requirements.

The most common conditional function that is used in SELECT is CASE Statements.

CASE statements: CASE statements in SQL helps in performing conditional logic within the SELECT part of the query. CASE statements allow to have multiple conditions, and when a value satisfies any particular condition, then the result related to it is displayed.

For Example:
Display the first name, last name, final ratings, and rating status of all the employees. To calculate the rating status based on the final rating value, use below logic:

1. When the final rating is 4 or above, the rating status should be "Top Rating".
2. When the final rating is between 2 and 4 , the rating status is "Average Rating".
3. When the final rating is 2 or below, the rating status is "Low Rating".

When there is no rating provided, the rating status is "No Rating Provided".
Use CASE STATEMENT to create the Final rating calculated column.

SQL code for this will be:

```
SELECT DE.First_Name,DE.Last_Name,DPR.Final_Rating,
CASE
WHEN DPR.Final_Rating>=4 THEN 'Top Rating'
WHEN DPR.Final_Rating>2 AND DPR.Final_Rating<4
THEN 'Average Rating'
WHEN DPR.Final_Rating<=2 THEN 'Low Rating'
ELSE 'No Rating Provided'
END AS Rating_Status
FROM [Mastering_SQL].[dbo].[Dataceps_Employees] DE
INNER JOIN
[Mastering_SQL].[dbo].Dataceps_Performance_Review] DPR
ON DE.[Employee_ID]=DPR.[Employee_ID]
```

5) SQL Conversion Functions: In SQL, Conversion functions help in converting one datatype to another datatype. This helps in data cleaning and transformation requirements.
Some common conversion functions are – CAST, CONVERT, TRY_CAST, TRY_CONVERT, etc.

For Example: Display the first_name, Final Rating, and Review date in the output. The review date should be in datetime format instead of date format.

SQL code for this will be:

```
SELECT
DE.first_name,
DPR.Final_Rating,
CONVERT(datetime, DPR.Review_Date) AS Review_Date
FROM
Dataceps_Employees DE
INNER JOIN
Dataceps_Performance_Review DPR
ON DE.Employee_ID = DPR.Employee_ID;
```


6) Aggregate Functions: Aggregate functions in SQL helps in performing an operation on multiple rows or group of rows of a particular column and returns a single value. Aggregate functions are usually used with a GROUP BY clause to perform the desired operation.
The different types of aggregate functions are COUNT, SUM, AVG, MIN, and MAX.

For Example: Display the average rating of employees for each department along with department_Id.

SQL code for this will be:

```sql
SELECT e.Department_ID,
AVG(pr.Final_Rating) AS Average_Rating
FROM Dataceps_Performance_Review pr
INNER JOIN Dataceps_Employees e
ON pr.Employee_ID = e.Employee_ID
GROUP BY e.Department_ID
```

Calculated columns in SQL queries play an essential role in data transformation and cleaning. This plays an important role in data analysis, extracting meaningful insights, data management, and much more.

The next chapter is completely dedicated to hands-on practice questions that will allow you to implement the concepts that you learned in this chapter as well as the previous ones.

HANDS ON EXERCISES

This chapter is totally dedicated for the SQL practice questions, In the intial set of practice questions there are some problems where you have to simply follow the instructions and practice SQL joins. Basically the level of difficulty is EASY to MEDIUM.

In the next set of questions, in this chapter, the questions will be evolved to a level where you have to understand a question written in simple English that requires a certain outcome/result. And in order to get that result, you need to implement the correct logic.

You can find all the tables mentioned in the below questions in the dataset that you configured earlier.

SET I

Practice Question #1 :

Select the product name and ProductNumber from the product table and theunit price from the SalesOrderDetail table after implementing the CROSS JOIN.

Practice Question #2 :

Select the product name from the product table and the unit price from the SalesOrderDetail table after implementing the INNER JOIN on the column Product_Id.

Practice Question #3 :

Select the product name from the product table, and the unit price from the SalesOrderDetail table after implementing the LEFT JOIN on the column ProductId.

Practice Question #4 :

Select the product name from the product table and the unit price from the SalesOrderDetail table after implementing the RIGHT JOIN from salesorderDetail table to product table on the column ProductId.

SET II

When we work on a project, Usually the requirements are not provided in technical languages . The requirements are provided in plain English, and then we have to convert those requirements into technical logic and then eventually convert it into SQL code.

Below are some practice problems that will help you in brushing up on that skill:

Practice Question #5 :

Retrieve the first name and last name of a customer and the order date for all those customers from the table "Customer" and "SalesOrderHeader" tables.

Practice Question #6 :

Find the SalesOrderId, ProductId, ProductName, ProductName, and ProductNumber for all the products present in the [SalesOrderDetail] and [Product] tables.

Practice Question #7 :

Find all the Product_Id, ProductName, ProductNumber, SalesOrderId, And OrderQuantity from the [Product] and [SalesOrderDetail] tables, even if the [SalesOrderDetail] doesn't have any data related to the corresponding product.

Practice Question #8 :

Find all the AddressID, AddressLine1, CountryRegion, and CustomerID data from the tables CustomerAddress and [Address] tables, even if it doesn't have any related customer_id in the customer_address table.

Practice Question #9 :

Display the Customer's First_Name, Last_Name, Company_Name, Address and Country_Region of all the customers present in the customer table. The other related information can be pulled from [CustomerAddress] and [Address] table.

Practice Question #10 :

Display the Customer's First_Name, Last_Name, Company_Name, Address and Country_Region of all the customers present in the customer table even if they don't have any address related information in [CustomerAddress] and [Address] table.

Practice Question #11 :

Display the name, color, and list_price of products where the product category name is Accessories by joining the tables [Product] and [ProductCategory]

Practice Question #12 :

Find all the names, productnumber, color, and description from the tables product and ProductAndDescription where the description contains the word “off-road.”

Practice Question #13:

Find the ProductModels and their total count where the description contains the word “aluminum” and the color of the product is ‘Black’

Practice Question #14:

Find Country with most shipped order quantities where Status=5 from sales order header table means shipped

Practice Question #15:

Find and display the customerId, FirstName, LastName, EmailAddress and Company Name of customers who never made an order.

Practice Question #16:

Find the TOP 10 Company Names With the Highest Order values (Line_Total Values) DESC, Consolidate the sum of all orders for each company

Practice Question #17 :

Find Company Names With the Lowest Order Values Order By Order Values ASC, consolidate the orders value for each company.

Practice Question #18 :

Find the top 5 customers with the highest total order values across all countries, along with their respective country and total order values.

Practice Question #19 :

SELECT the product models along with their total count where the color is ‘Red’ and the description of the product has the word aluminium in it and the total count is greater than 250.

Practice Question #20 :

SELECT the product names along with their total count where the color is neither red ‘Red’ nor ‘Black’ and the description of the product category is not ‘Bikes’.

Practice Question #21 :

Find the name, ListPrice, OrderQuantity, SalesPerson, and the customer name that is concatenated using the customer’s first name and last name where the Salesperson field contains the string ‘jae0’ from the tables SalesOrderDetail, Customer and Product and order by DESC.

Practice Question #22 :

Display the Customer_Id,Customer_Name(*after concatenating First_Name and Last_Name*) their shipping address as Customer_Shipping_Address(*After concatenating Address line 1 and address line 2*) and replace null with blank space And Country region-that have a shipping address in the United states region and TaxAmount greater than 500.

Practice Question #23 :

Find the Names of the 5 highest selling products based on their total order quantities sold till date.

Practice Question #24:

Display the Product name, Total Order quantities for every product sold, along with Sales Status.When the order quantity <=10, then the Sales status should be 'Low Sales'; when the order quantity <= 20, the Sales status should be 'Medium Sales'. When the order quantity <= 30, the Sales status should be 'High Sales', For other cases display 'Not Sufficient Data'. When the order quantity > 30, the Sales status should be 'Super High Sales', And the order should be based on TotalOrderQuantity in Descending order.

HINT : Do a GROUP BY based on the Column 'Name' based on the table Product Tables Involved : SalesOrderDetail,Product

Practice Question #25:

Display the names of customers who have the highest order value within each country (based on shipping address), along with their order quantity. Add the total sum of orders and find the highest order value customer for each country

Practice Question #26:

Find the 2nd highest selling product in each country.

Practice Question #27:

Find the products that have never been sold.

ANSWERS

SET I

Practice Question #1:

```
SELECT
P.[Name],
P.ProductNumber,
SO.UnitPrice
FROM [dbo].[SalesOrderDetail] SO
CROSS JOIN [dbo].[Product] P
```


Practice Question #2:

```
SELECT P.[Name],
SO.UnitPrice
FROM [dbo].[SalesOrderDetail] SO
INNER JOIN [dbo].[Product] P
ON SO.ProductID=P.ProductID
```

Practice Question #3:

```
SELECT P.[Name],
SO.UnitPrice
FROM [dbo].[SalesOrderDetail] SO
LEFT JOIN [dbo].[Product] P
ON SO.ProductID=P.ProductID
```

Practice Question #4:


```
SELECT P.[Name],
SO.UnitPrice
FROM [dbo].[SalesOrderDetail] SO
RIGHT JOIN [dbo].[Product] P
ON SO.ProductID=P.ProductID
```

SET II

Practice Question #5:

```
SELECT
C.FirstName,
C.LastName,
SOH.OrderDate
FROM [dbo].[SalesOrderHeader] SOH
INNER JOIN [dbo].[Customer] C
ON C.CustomerID=SOH.CustomerID
```


Practice Question #6:

```
SELECT SOD.SalesOrderID,
P.ProductID,
P.ProductNumber,
P.[Name]
FROM [dbo].[SalesOrderDetail] SOD
INNER JOIN [dbo].[Product] P
ON SOD.ProductID=P.ProductID
```

Practice Question #7:

```
SELECT P.ProductID,
P.ProductNumber,
P.[Name],
SOD.SalesOrderID,
SOD.OrderQty
FROM [dbo].[Product] P
LEFT JOIN [dbo].[SalesOrderDetail] SOD
ON SOD.ProductID=P.ProductID
```


Practice Question #8:

```
SELECT
A.AddressID,A.AddressLine1,A.Coun
tryRegion,CA.CustomerID
FROM CustomerAddress CA
LEFT JOIN
[Address] A
ON CA.AddressID=A.AddressID
```

Practice Question #9:

```
SELECT C.FirstName,
C.LastName,
C.CompanyName
,A.AddressLine1,A.CountryRegion
FROM [dbo].[Customer] C
INNER JOIN CustomerAddress CA
ON C.CustomerID=CA.CustomerID
INNER JOIN [Address] A
ON CA.AddressID=A.AddressID
```

Practice Question #10:


```
SELECT C.FirstName,
C.LastName,
C.CompanyName,
A.AddressLine1,
A.CountryRegion
FROM [dbo].[Customer] C
LEFT JOIN CustomerAddress CA
ON C.CustomerID=CA.CustomerID
LEFT JOIN [Address] A
ON CA.AddressID=A.AddressID
```

Practice Question #11:

```
SELECT P.[Name],
P.[Color],
P.[ListPrice]
FROM [dbo].[Product] P
INNER JOIN [dbo].[ProductCategory] PC
ON P.ProductID=P.ProductID
WHERE PC.[Name] LIKE '%Accessories%'
```

Practice Question #12:


```
SELECT P.[Name],
P.ProductNumber,
P.Color,
PAD.[Description]
FROM [dbo].[Product] P
INNER JOIN
[dbo].[ProductAndDescription] PAD
ON P.ProductID=PAD.ProductID
WHERE PAD.[Description] LIKE '%off-road%'
```

Practice Question #13:

```
SELECT ProductModel,
COUNT(1) AS TotalCount
FROM [dbo].[ProductAndDescription] PAD
INNER JOIN [dbo].[Product] P
ON P.ProductID=PAD.ProductID
WHERE  [Description]  LIKE  '%aluminum%'
AND P.Color='Black'
GROUP BY ProductModel
```

Practice Question #14:


```
SELECT TOP 1 CountryRegion,
SUM(SOD.OrderQty) AS Shipped_Order_Qty
FROM
[dbo].[SalesOrderHeader] SOH
INNER JOIN [dbo].[Address] A
ON SOH.ShipToAddressID=A.AddressID
INNER JOIN dbo.SalesOrderDetail SOD
ON SOH.SalesOrderID=SOD.SalesOrderID
GROUP BY CountryRegion
ORDER BY Shipped_Order_Qty DESC
```

Practice Question #15:

```
SELECT C.CustomerID,
C.FirstName,
C.LastName,
C.EmailAddress,
C.CompanyName
FROM [dbo].[Customer] C
LEFT JOIN [dbo].[SalesOrderHeader] SOH
ON C.CustomerID=SOH.CustomerID
WHERE SOH.CustomerID IS NULL
```

Practice Question #16:


```
SELECT TOP 10
C.CompanyName,SUM(SOD.LineTotal) AS
Total_Order_Value
FROM [dbo].[SalesOrderHeader] SOH
INNER JOIN [dbo].[Customer] C
ON SOH.CustomerID=C.CustomerID
INNER JOIN [dbo].[SalesOrderDetail] SOD
ON SOH.SalesOrderID=SOD.SalesOrderID
GROUP BY C.CompanyName
ORDER BY 2 DESC
```

Practice Question #17:

```
SELECT TOP 5
C.CompanyName,SUM(SOD.LineTotal) AS
Total_Order_Value
FROM [dbo].[SalesOrderHeader] SOH
INNER JOIN [dbo].[Customer] C
ON SOH.CustomerID=C.CustomerID
INNER JOIN [dbo].[SalesOrderDetail] SOD
ON SOH.SalesOrderID=SOD.SalesOrderID
GROUP BY C.CompanyName
ORDER BY 2 ASC
```

Practice Question #18:


```
SELECT TOP 5
C.FirstName,C.LastName,A.CountryRegion,
SUM(SOD.LineTotal) AS Order_Value
FROM [dbo].[SalesOrderDetail] SOD
INNER JOIN [dbo].[SalesOrderHeader] SOH
ON SOD.SalesOrderID=SOH.SalesOrderID
INNER JOIN dbo.[Address] A
ON SOH.ShipToAddressID=A.AddressID
INNER JOIN dbo.Customer C
ON SOH.CustomerID=C.CustomerID
GROUP BY
A.CountryRegion,C.FirstName,C.LastName
ORDER BY Order_Value DESC
```

Practice Question #19:

```
SELECT ProductModel,COUNT(1) AS
Total_Count
FROM [dbo].[Product] P
INNER JOIN
[dbo].[ProductAndDescription] PAD
ON P.ProductID=P.ProductID
WHERE P.Color='Red'
AND PAD.[Description] LIKE '%aluminium%'
GROUP BY ProductModel
HAVING COUNT(1)>250
```

Practice Question #20:


```
SELECT P.[Name],COUNT(1) AS Total_Count
FROM [dbo].[Product] P
INNER JOIN
[dbo].[ProductCategory] PC
ON
P.ProductCategoryID= P.ProductCategoryID
WHERE
P.Color NOT IN ('%Black%' ,'%Red%')
AND PC.[Name] NOT LIKE '%Bikes%'
GROUP BY P.[Name]
ORDER BY P.[Name] DESC
```

Practice Question #21:

```
SELECT P.Name],P.ListPrice,SOD.OrderQty,
CONCAT (C.FirstName,' ',C.LastName) AS
Customer_Name,
C.SalesPerson
FROM [dbo].[SalesOrderDetail] SOD
INNER JOIN [dbo].[SalesOrderHeader] SOH
ON SOD.SalesOrderID=SOH.SalesOrderID
INNER JOIN Customer C
ON C.CustomerID=SOH.CustomerID
AND C.SalesPerson LIKE '%jae0%'
INNER JOIN Product P
ON SOD.ProductID=P.ProductID
ORDER BY OrderQty DESC
```

Practice Question #22:

```
SELECT SOH.CustomerID,
CONCAT (C.FirstName,' ',
C.LastName) AS Customer_Name ,
CONCAT(A.AddressLine1,'',
NULLIF(A.AddressLine2,''))
AS Customer_Shipping_Address,
A.CountryRegion
FROM
[dbo].[SalesOrderHeader] SOH
INNER JOIN dbo.Customer C
ON SOH.CustomerID=C.CustomerID
INNER JOIN [dbo].[Address] A
ON A.AddressID=SOH.ShipToAddressID
WHERE SOH.TaxAmt>500
AND A.CountryRegion='United States'
```


Practice Question #23:

```
SELECT TOP 5   P.Name AS ProductName,
SUM(SOD.OrderQty) AS TotalOrderQuantity
FROM  Product P
INNER JOIN  SalesOrderDetail SOD
ON P.ProductID = SOD.ProductID
GROUP BY  P.Name
ORDER BY SUM(SOD.OrderQty) DESC;
```

Practice Question #24:

```
SELECT P.[Name] AS
ProductName,SUM(SOD.OrderQty)
AS TotalOrderQuantity,
CASE
WHEN SUM(SOD.OrderQty) <= 10
THEN 'Low Sales'
WHEN SUM(SOD.OrderQty) <= 20
THEN 'Medium Sales'
WHEN SUM(SOD.OrderQty) <= 30
THEN 'High Sales'
WHEN SUM(SOD.OrderQty) > 30
THEN 'Super High Sales'
ELSE 'Not Sufficient Data'
END AS SalesStatus

FROM Product P
INNER JOIN SalesOrderDetail SOD
ON P.ProductID = SOD.ProductID
GROUP BY P.Name
ORDER BY TotalOrderQuantity DESC;
```

Practice Question #25:

```
WITH CustomerTotalOrderValue
AS(
SELECT C.CustomerID,ShipToAddressID,
SUM(SOD.LineTotal) AS Total_Order_Value
FROM [dbo].[SalesOrderHeader] SOH
INNER JOIN dbo.Customer C
ON SOH.CustomerID=C.CustomerID
INNER JOIN dbo.SalesOrderDetail SOD
ON SOH.SalesOrderID=SOD.SalesOrderID
GROUP BY
C.CustomerID,SOH.ShipToAddressID
),
FindRank AS
(
SELECT C.FirstName,C.LastName,
A.CountryRegion,
RANK() OVER(PARTITION BY A.CountryRegion
ORDER BY CTOV.Total_Order_Value DESC )
AS RNK,
CTOV.Total_Order_Value
FROM CustomerTotalOrderValue CTOV
INNER JOIN dbo.Customer C
ON CTOV.CustomerID=C.CustomerID
INNER JOIN dbo.[Address] A
ON CTOV.ShipToAddressID=A.AddressID
)

SELECT * FROM FindRank WHERE RNK=1
```

Practice Question #26:

```
WITH Product_Rank AS
(
SELECT SOD.ProductID,SOD.OrderQty,
A.CountryRegion,
DENSE_RANK() OVER(PARTITION BY
A.CountryRegion ORDER BY SOD.OrderQty
DESC
)AS RNK,
SOH.ShipToAddressID
FROM [dbo].[SalesOrderHeader] SOH
INNER JOIN [dbo].[SalesOrderDetail] SOD
ON SOD.SalesOrderID=SOH.SalesOrderID
INNER JOIN dbo.[Address] A
ON A.AddressID=SOH.ShipToAddressID

)
SELECT DISTINCT P.
[Name],PR.CountryRegion
FROM Product P
INNER JOIN Product_Rank PR
ON P.ProductID=PR.ProductID
WHERE PR.RNK=2
```

Practice Question #27:

```
SELECT P.ProductID
FROM [dbo].[Product] P
LEFT JOIN [dbo].[SalesOrderDetail] SOD
ON P.ProductID=SOD.ProductID
WHERE SOD.ProductID IS NULL
```

BIBLIOGRAPHY

Simple SQL: Beginner's Guide To Master SQL And Boost Career (Zero To Hero): Wade, Dane: 9798833376164: Amazon.com: Books. (2022, June 3). Simple SQL: Beginner's Guide to Master SQL and Boost Career (Zero to Hero): Wade, Dane: 9798833376164: Amazon.com: Books. https://www.amazon.com/Simple-SQL-Beginners-Master-Career/dp/B0B2V3W7DG

Drkusic, E., & posts by Emil Drkusic →, V. A. (2020, February 21). Learn SQL: Set Theory. SQL Shack - Articles About Database Auditing, Server Performance, Data Recovery, and More. https://www.sqlshack.com/learn-sql-set-theory/

How to Restore a .bak File using Azure Data Studio. (n.d.). How to Restore a .Bak File Using Azure Data Studio. https://www.quackit.com/sql_server/mac/how_to_restore_a_bak_file_using_azure_data_studio.cfm