PER SCHOLAS

Lesson 304.3.2 - SQL JOIN Tables

PER SCHOLAS

Section 1

SQL JOIN Tables

Learning Objectives:

This presentation explains how to use SQL JOINS with syntax, visual illustrations, and examples.

By the end of this lesson, learners will be able to:

- Describe the JOIN Predicate.
- Demonstrate the inner JOIN, left JOIN, right JOIN, self JOIN, cross JOIN, and Union.

Prerequisite

Note: We will utilize a “**classicmodels”** database for demonstrations and examples in this presentation.

Table of Contents

Overview of Joining Tables

Question:

We have used SELECT statement to query data from a single table. How do we query data from multiple tables that are related by one or more primary key/foreign key relationships?

For example, in the database diagrammed, there are nine tables; all are related, either directly or indirectly.

Some of the connections go through junction tables, defining many-to-many relationships.

Answer: A ***JOIN*** predicate is used to combine rows from two or more tables, based on a related column between them.

Bike Stores database diagram

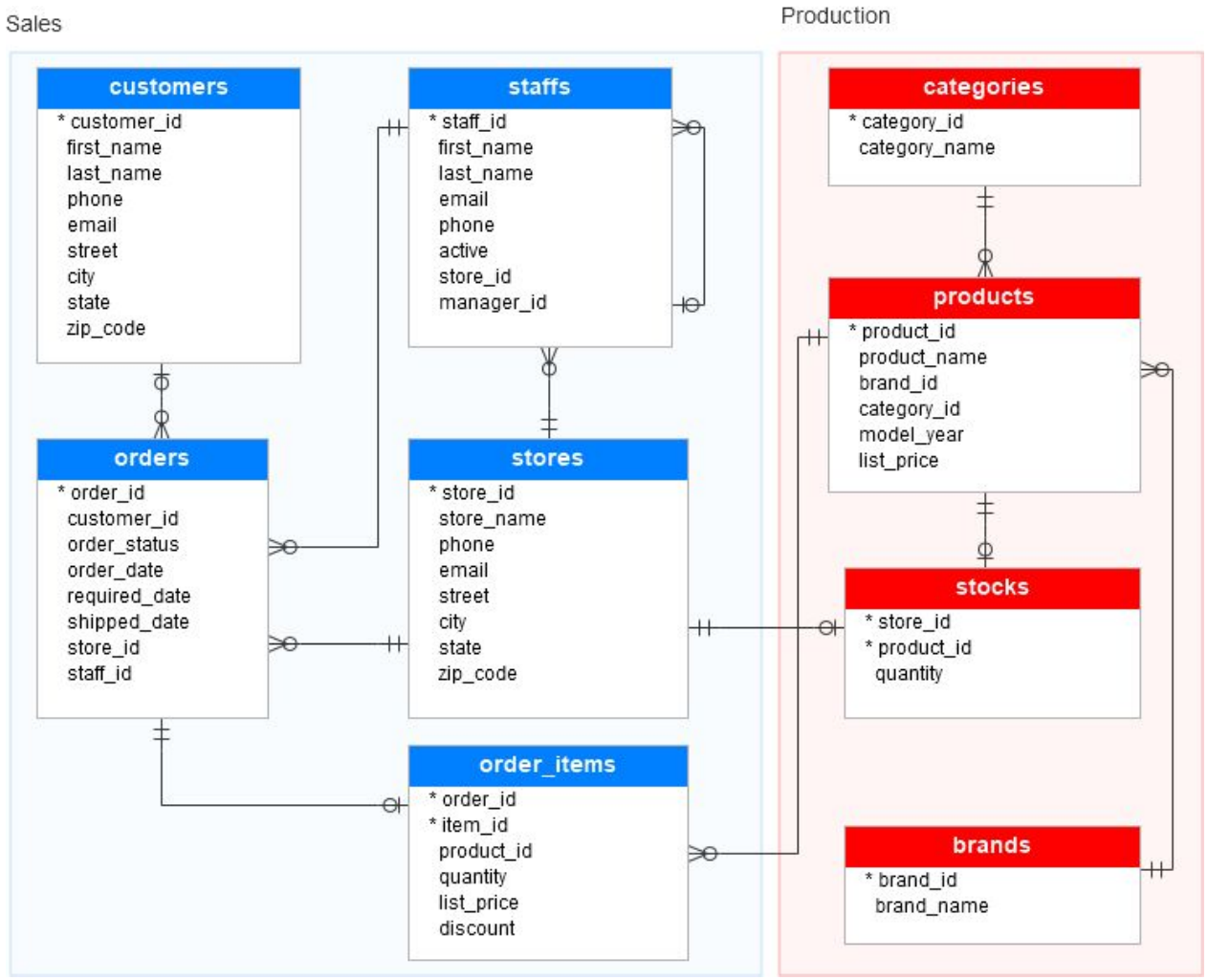


image source: sqlservertutorial.net

Types of JOINS

- JOIN predicate allow us to walk through the relationships between two or more tables in the FROM clause.
- JOINS are queries that combine the data of multiple tables based on their common ***columns (primary key and foreign key)*** and ***constraints*** to produce a combined result set.
- The following JOINS are supported by MySQL:
  1. Inner JOIN
  2. Left JOIN
  3. Right JOIN
  4. Cross JOIN
  5. Self JOIN

Note that MySQL does not support the full Outer JOIN.

General JOIN Syntax

```
Select <column-names>
FROM <left table> [join type] JOIN <right table> ON <join predicate> ....
```

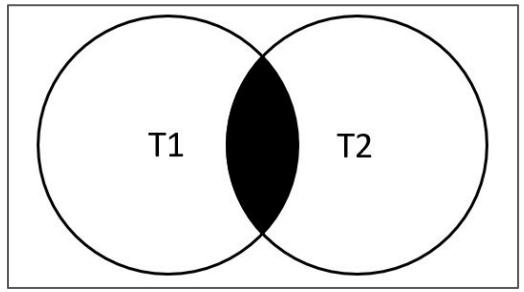
The join predicate is a boolean expression, specifying criteria for matching rows between two or more tables. Most often, the predicate is based on key relationships, but other boolean expressions can be used.

The Join predicate often requires us to use table aliases to distinguish field names.

Overview of INNER JOIN Clause

- The **INNER JOIN** clause joins two or more tables based on a condition, which is known as a join predicate.
- For inner joins, rows from the left and right tables will appear in the output, if and only if they both satisfy the join predicate.
- The **INNER JOIN** includes only matching rows from both tables.

Visual illustration



img src: databasestar.com

```
select a.value, b.value from TableA a join TableB b on a.id=b.id;
```

Table A

id	value
1	A
2	B
3	C
4	D
5	E
6	F

inner join
on a.id=b.id

Table B

id	value
5	J
6	K
7	L
8	M
9	N
10	O

→

Result

a.value	b.value
E	J
F	K

2 rows satisfy the join predicate

Syntax of the INNER JOIN

The following shows the basic syntax of the **"INNER JOIN"** clause that joins tables: **table_1** and **table_2**.

```
SELECT column_list
FROM table_1
INNER JOIN table_2 ON join_condition;
```

The **INNER JOIN** clause compares each row from the first table with every row from the second table.

When the join predicate is based on equality between two columns with the **same name**, SQL gives us a **shortcut.** You can use the ***USING*** *clause* instead as shown below:

```
SELECT column_list
FROM table_1
INNER JOIN table_2 USING (column_name);
```

Both queries will give the identical result.

Ambiguous Column Errors in JOIN

- If you join multiple tables that have the same column name, you have to use a table qualifier to refer to that column in the **SELECT** statement to avoid ambiguous column errors.

- For example, if both `Table1` and `Table2` have the same column named `City` in the `SELECT` statement, you have to refer to the `City` column using the table qualifiers as `Table1.City` or `Table2.City`

- To save time typing the table qualifiers, you can use **table aliases** in the query. For example, you can give the `verylongtablename` table an alias `T` and refer to its columns using `T.column` instead of `verylongtablename.column`.

Example 1: INNER JOIN

If you want to get the *product code* and *product name* from the **products** table, or the *text description* of *product lines* from the **productlines** table, you need to select data from both tables and match rows by comparing the **productline** column from the **products** table with the **productline** column from the **productlines** table, as shown in the following query.

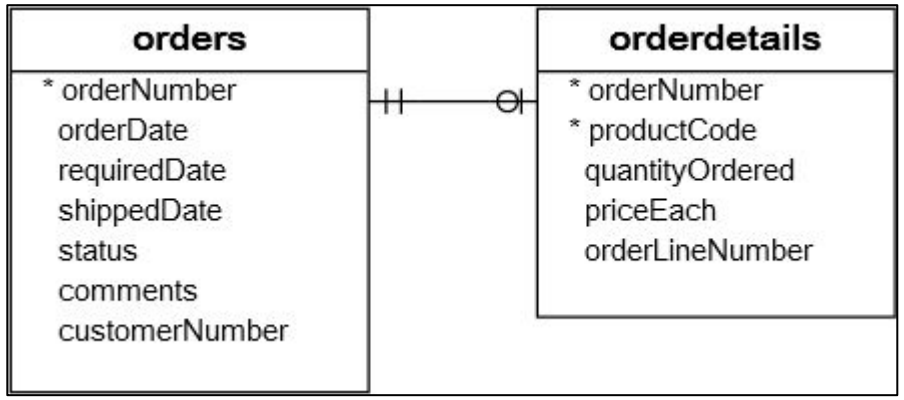
```
SELECT  T1.productCode,  T1.productName,  T2.textDescription
FROM  products T1
INNER JOIN productlines T2 ON T1.productline = T2.productline;
```

Output:

	productCode	productName	textDescription
	S10_1949	1952 Alpine Renault 1300	Attention car enthusiasts: Make your wildest car ownership dreams come true.
	S10_4757	1972 Alfa Romeo GTA	Attention car enthusiasts: Make your wildest car ownership dreams come true.
	S10_4962	1962 LanciaA Delta 16V	Attention car enthusiasts: Make your wildest car ownership dreams come true.
	S12_1099	1968 Ford Mustang	Attention car enthusiasts: Make your wildest car ownership dreams come true.
	S12_1108	2001 Ferrari Enzo	Attention car enthusiasts: Make your wildest car ownership dreams come true.

Example 2: INNER JOIN with GROUP BY Clause

Consider the **orders** and **orderdetails** tables:



We can get the order number, order status, and total sales from the **orders** and **orderdetails** tables using the **INNER JOIN** clause with the GROUP BY clause, as shown in the query below:

```
SELECT  T1.orderNumber,  STATUS, SUM(quantityOrdered * priceEach) total
FRO
M orders AS T1
INNER JOIN orderdetails AS T2 ON T1.orderNumber = T2.orderNumber
GROUP BY T1.orderNumber;
```

Output:

orderNumber	status	total
10100	Shipped	10223.83
10101	Shipped	10549.01
10102	Shipped	5494.78
10103	Shipped	50218.95
10104	Shipped	40206.20

We can write the above query with the **"using"** keyword. See the next slide for a demonstration.

continue…

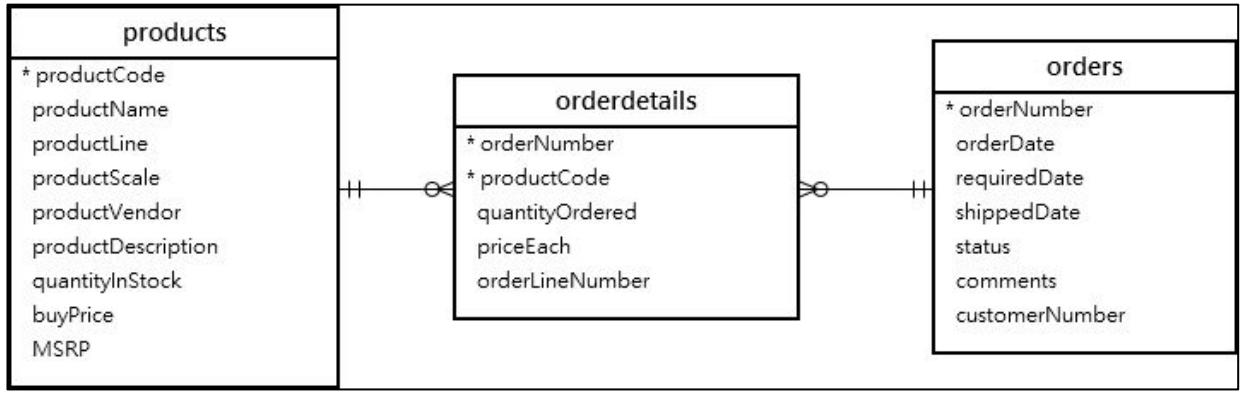
Example 2: INNER JOIN with GROUP BY Clause (continued)

We can write the query on the previous page with the **"using"** keyword:

```
SELECT  T1.orderNumber,  STATUS, SUM(quantityOrdered * priceEach) total
FROM orders AS T1
INNER JOIN orderdetails AS T2 using(orderNumber)
GROUP BY T1.orderNumber;
```

Example 3: INNER JOIN – Join Three Tables

Consider the **products, orders** and **orderdetails** tables. The query below is using two **INNER JOIN** clauses to join three tables: **orders, orderdetails,** and **products.**




```
SELECT orderNumber, orderDate, orderLineNumber, productName, quantityOrdered,  priceEach
FROM    orders
INNER JOIN    orderdetails USING (orderNumber)
INNER JOIN    products USING (productCode)
ORDER BY     orderNumber,     orderLineNumber;
```

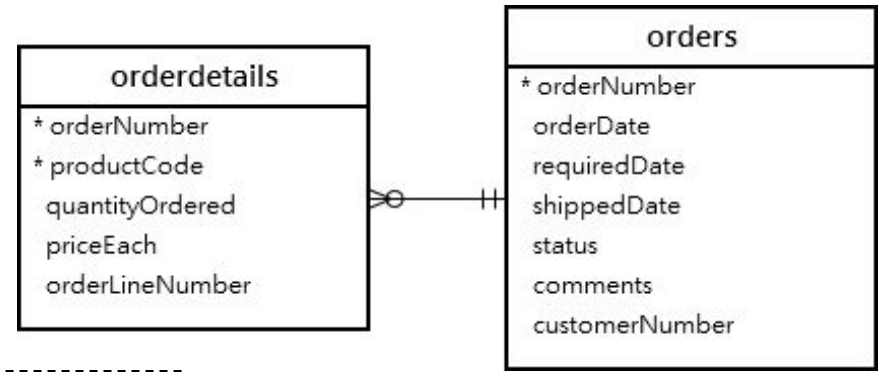
orderNumber	orderDate	orderLineNumber	productName	quantityOrdered	priceEach
10100	2003-01-06	1	1936 Mercedes Benz 500k Roadster	49	35.29
10100	2003-01-06	2	1911 Ford Town Car	50	55.09
10100	2003-01-06	3	1917 Grand Touring Sedan	30	136.00
10100	2003-01-06	4	1932 Alfa Romeo 8C2300 Spider Sport	22	75.46
10101	2003-01-09	1	1928 Mercedes-Benz SSK	26	167.06
10101	2003-01-09	2	1938 Cadillac V-16 Presidential Limousine	46	44.35

This picture shows the partial output

Example 4: INNER JOIN With GROUP BY and Having Clause

Let's take a look at the **orders** and **orderdetails** tables from the ***classicmodels*** database.

The following query finds sales orders whose value total is greater than $60K.




```
SELECT orderNumber, SUM(priceEach * quantityOrdered) as total
FROM orderdetails
INNER JOIN orders USING (orderNumber)
GROUP BY orderNumber HAVING SUM(priceEach * quantityOrdered) > 60000;
```

	orderNumber	total
	10165	67392.85
	10287	61402.00
	10310	61234.67

It returns three rows, which means that there are three sales orders whose total values total greater than $60K.

Natural JOIN

The **INNER JOIN** is also called a **Natural JOIN**. We could also write the previous query without INNER or using a keyword. See below:

```
select a.value, b.value from A a natural join B b;
```

For a Natural JOIN, the JOIN predicate is formed “naturally” based on fields from both tables with matching names. Tables A and B both have an “id” field; therefore, the Natural JOIN forms the predicate based on the equality of **A.id** and **B.id**.

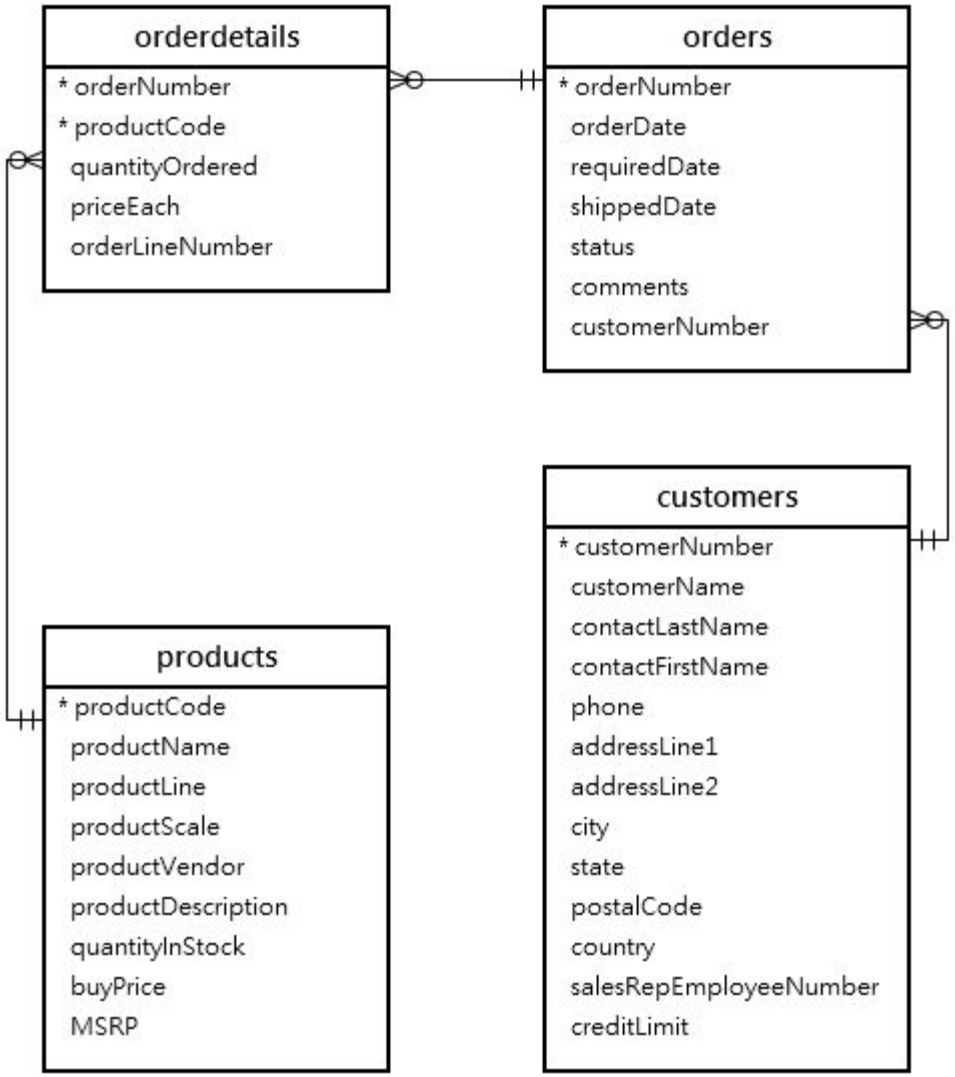
You should be aware of Natural JOIN syntax in case you encounter it, but its use is strongly discouraged. As databases evolve, it is not uncommon to rename fields, and this process will cause Natural JOINS, based on those fields, to return incorrect results without generating an error message.

Challenge!

Take a look at the image on the right; it shows us how the tables for ***orders, orderdetails, customers,*** and ***products*** are related in the "**classicmodels**" database.

Problem statement: Write a query to display the customer number, customer name, order number, order date, product code, product name, and price.

You can use the **JOIN, GROUP By,** and **HAVING** clauses.

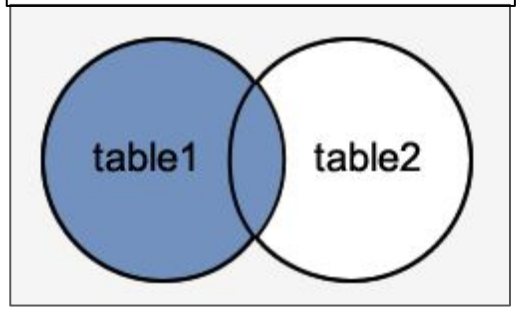


Overview of LEFT JOIN Clause

In a LEFT JOIN, all rows from the left table are guaranteed to be represented in the *result set* whether or not a row from the right table is found to satisfy the predicate.

When no matching row from the right table is found, those fields will be **NULL**:

Visual illustration



img src: databasestar.com

```
select a.value, b.value from TableA a LEFT JOIN TableB b on a.id=b.id;
```

Table A

id	value
1	A
2	B
3	C
4	D
5	E
6	F

```
left join
on
a.id=b.id
```

Table B

id	value
5	J
6	K
7	L
8	M
9	N
10	O

Result

a.value	b.value
A	NULL
B	NULL
C	NULL
D	NULL
E	J
F	K

Using LEFT JOIN

One common usage for **LEFT JOIN** is to count records for which there is no match. Consider our previous example:

```
select count(*) from A a left join B b using(id) where b.value is NULL;
```

- This query returns 4 rows – there are 4 unmatched rows between the tables.

- **Remember** that we also use JOINS in succession. When the results of a **LEFT JOIN** flows into another JOIN, we are guaranteed that all rows from the left-most table will be represented. If we use **LEFT JOINS** in succession, we are guaranteed that the final ***result set*** will have all rows of the left-most table represented.
- When we use **COUNT(*)** with GROUP BY, we often want to include groups with a count of zero. LEFT JOIN lets us do this.

Syntax - LEFT JOIN

The syntax for the LEFT JOIN in SQL is:

```
SELECT columns
FROM table1
LEFT JOIN table2
ON table1.column_name = table2.column_name;
```

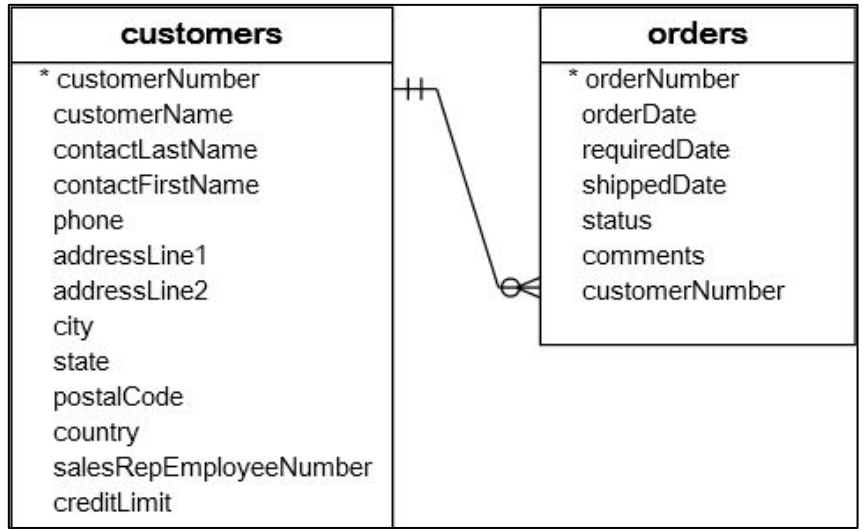
You can use the **USING** keyword syntax like this:

```
SELECT columns
FROM table1
LEFT JOIN table2
USING(column_name);
```

Example 1: LEFT JOIN Clause - Join Two Tables

Let's take a look at the image on the right, which shows the **customers** and **orders** tables.

- Each order in the **orders** table must belong to a customer in the **customers** table.
- Each customer in the **customers** table can have zero or more orders in the orders table.



To find all customers regardless of their order status, you can use the **LEFT JOIN** clause:

```
SELECT  c.customerNumber,  c.customerName, o.orderNumber, o.status
FROM customers c
LEFT JOIN orders o ON c.customerNumber = o.customerNumber;
```

customerNumber	customerName	orderNumber	status
166	Handji Gifts& Co	10288	Shipped
166	Handji Gifts& Co	10409	Shipped
167	Herkku Gifts	10181	Shipped
167	Herkku Gifts	10188	Shipped
167	Herkku Gifts	10289	Shipped
168	American Souvenirs Inc	NULL	NULL

This image shows the partial output:

Example 2: LEFT JOIN Clause - Find Unmatched Rows

The **LEFT JOIN** clause is very useful when you want to find the rows in the left table that do not match with the rows in the right table. To find the ***unmatched*** rows between two tables, you add a ***WHERE clause*** to the **SELECT** statement to select only rows whose column values in the right table contain the **NULL** values.

For example, to find all customers who have not ordered any products, you can use the following query:

```
SELECT c.customerNumber, c.customerName, orderNumber, o.STATUS
FROM customers c
LEFT JOIN orders o ON c.customerNumber = o.customerNumber
WHERE orderNumber IS NULL;
```

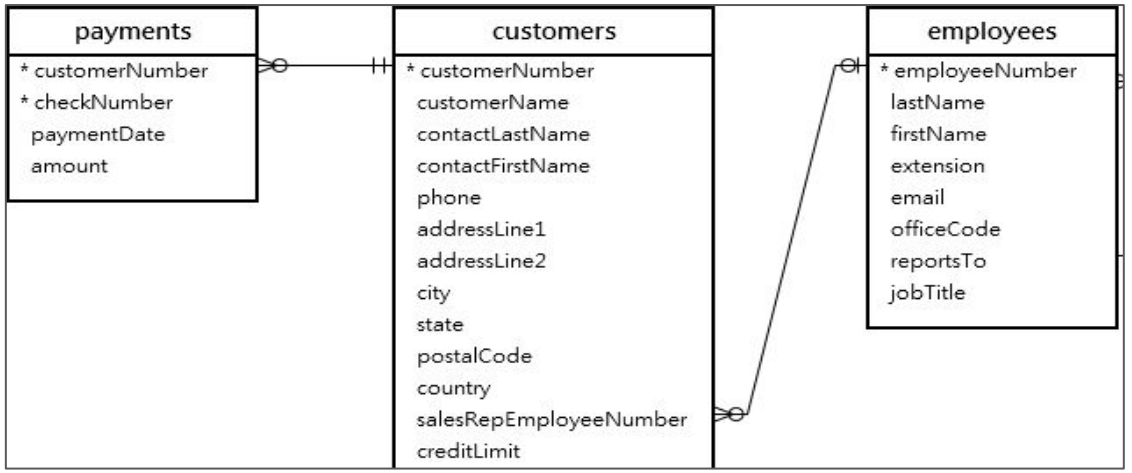
This image shows the partial output.

customerNumber	customerName	orderNumber	status
125	Havel & Zbyszek Co	NULL	NULL
168	American Souvenirs Inc	NULL	NULL
169	Porto Imports Co.	NULL	NULL
206	Asian Shopping Network, Co	NULL	NULL
223	Natürlich Autos	NULL	NULL
237	ANG Resellers	NULL	NULL
247	Messner Shopping Network	NULL	NULL
273	Franken Gifts, Co	NULL	NULL
293	BG&E Collectables	NULL	NULL
303	Schuyler Imports	NULL	NULL

PER SCHOLAS

Example 3: LEFT JOIN Clause - Find Unmatched Rows

Take a look at the right-hand figure, which shows how the tables for **employees, customers,** and **payments** are related in the "**classicmodels**" database.



lastName	firstName	customerName	checkNumber	amount
Bondur	Gerard	NULL	NULL	NULL
Bow	Anthony	NULL	NULL	NULL
Kato	Yoshimi	NULL	NULL	NULL
King	Tom	NULL	NULL	NULL
Murphy	Diane	NULL	NULL	NULL
Patterson	Mary	NULL	NULL	NULL
Firrelli	Jeff	NULL	NULL	NULL
Patterson	William	NULL	NULL	NULL
Hernandez	Gerard	Alpha Cognac	AF40894	33818.34
Hernandez	Gerard	Alpha Cognac	HR224331	12432.32
Hernandez	Gerard	Alpha Cognac	KI744716	14232.70
Tseng	Foon Yue	American Souvenirs Inc	NULL	NULL
Castillo	Pamela	Amica Models & Co.	IJ399820	33924.24
Castillo	Pamela	Amica Models & Co.	NE404084	48298.99
Fixter	Andy	Anna's Decorations, Ltd	EM979878	27083.78
Fixter	Andy	Anna's Decorations, Ltd	KM841847	38547.19
Fixter	Andy	Anna's Decorations, Ltd	LE432182	41554.73
Fixter	Andy	Anna's Decorations, Ltd	OJ819725	29848.52

The below query uses two **LEFT JOIN** clauses to join the three tables: **employees, customers,** and **payments.**

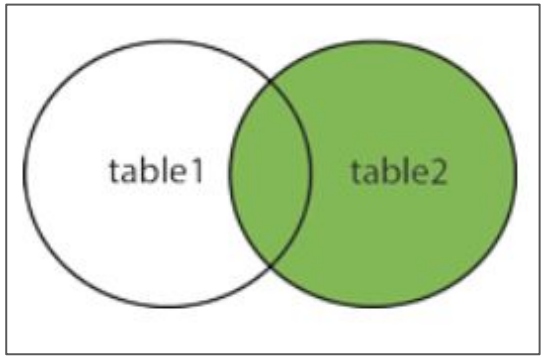
Output:

```
SELECT lastName, firstName, customerName,
checkNumber, amount
FROM employees
LEFT JOIN customers ON
    employeeNumber = salesRepEmployeeNumber
LEFT JOIN payments ON
    payments.customerNumber =
customers.customerNumber
ORDER BY customerName,  checkNumber;
```

Overview of RIGHT JOIN Clause

- **RIGHT JOIN** is similar to LEFT JOIN, except that the treatment of the joined tables is reversed.
- In a **RIGHT JOIN**, all rows from the right table are guaranteed to be represented in the *result set*, whether or not a row from the left table is found to satisfy the predicate.
- When no matching row from the left table is found, those fields will be **NULL**:

Visual illustration



img src: databasestar.com

```
select a.value, b.value from TableA a RIGHT JOIN TableB b ON a.id=b.id;
```

Table A

id	value
1	A
2	B
3	C
4	D
5	E
6	F

right join
on
a.id=b.id

Table B

id	value
5	J
6	K
7	L
8	M
9	N
10	O

Result

a.value	b.value
E	J
F	K
NULL	L
NULL	M
NULL	N
NULL	O

Syntax - RIGHT JOIN

The syntax for the RIGHT JOIN in SQL is:

```
SELECT columns
FROM table1
RIGHT JOIN table2
ON table1.column_name = table2.column_name;
```

You can use the **USING** syntax like this:

```
SELECT columns
FROM table1
RIGHT JOIN table2
USING(column_name);
```

Example 1: Right JOIN Clause - Join Two Tables

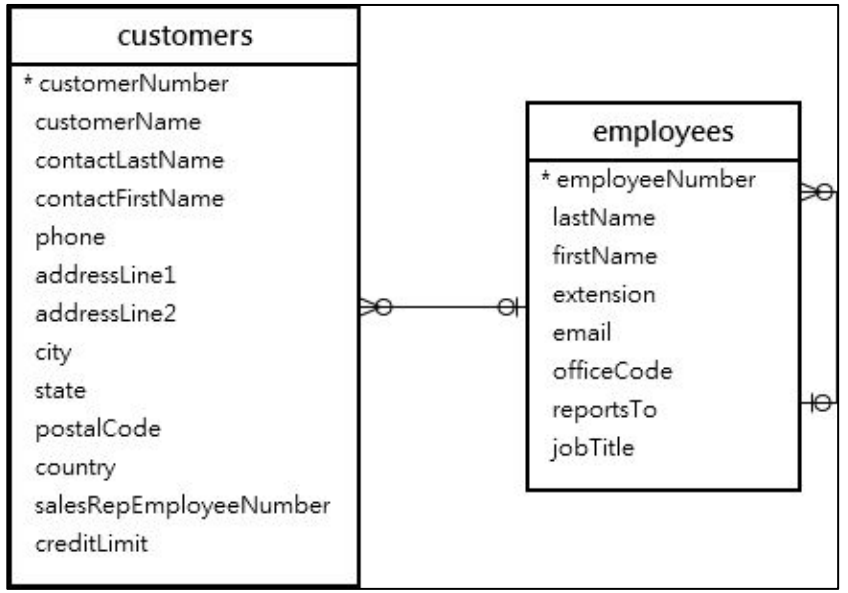
Take a look at the image on the right; it shows how the tables for **employees** and **customers** are related in the "**classicmodels**" database.

The column ***salesRepEmployeeNumber*** foreign key in the **customers** table links to the column ***employeeNumber*** in the **employees** table.

A sales representative, or an employee, may be in charge of zero or more customers, and each customer is taken care of by zero or one sales representative.

If the value in the column ***salesRepEmployeeNumber*** is **NULL**, this means that the customer does not have a sales representative.

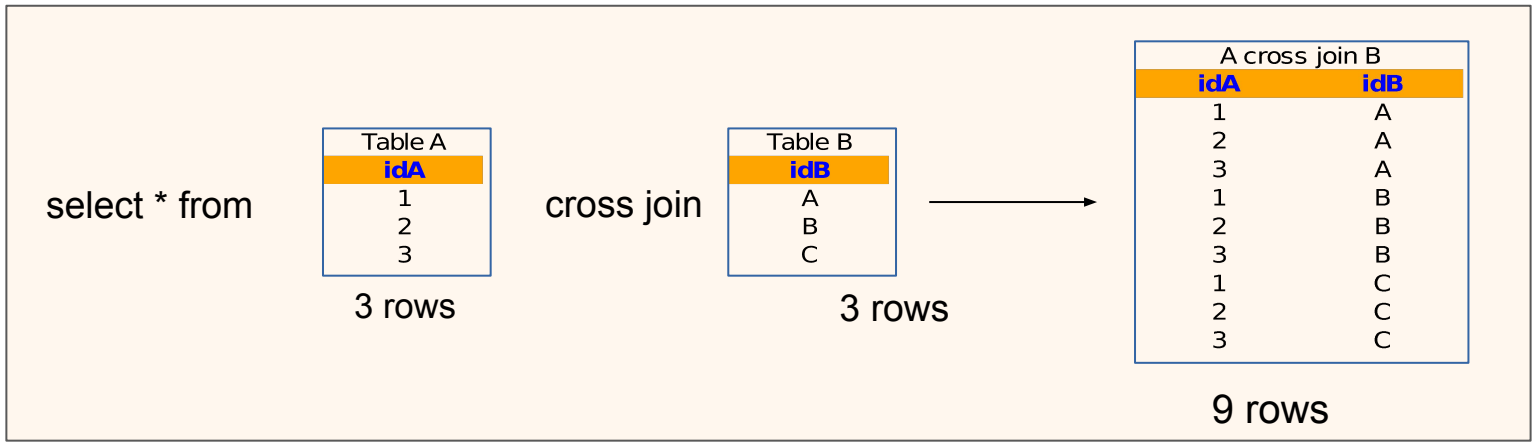
```
SELECT customerNumber, salesRepEmployeeNumber
FROM employees
RIGHT JOIN customers
ON employeeNumber = salesRepEmployeeNumber
ORDER BY customerName;
```

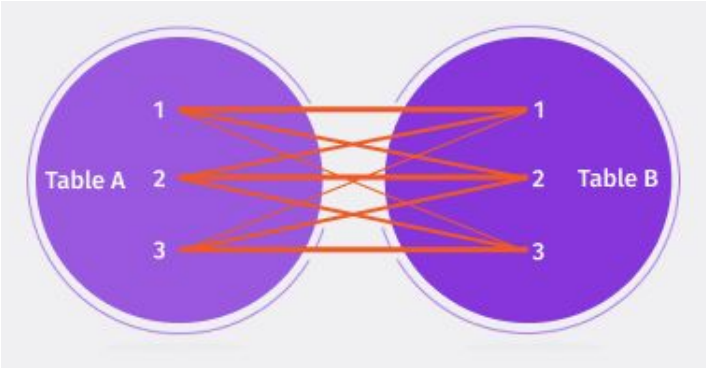


customerNumber	salesRepEmployeeNumber
242	1370
168	1286
249	1401
237	NULL
276	1611
465	NULL
206	NULL
348	NULL
103	1370
471	1611
114	1611
333	1611
256	1370
406	1337

Overview of CROSS JOIN Clause

The **CROSS JOIN** is an unfiltered join – it uses a no join predicate. The output of a CROSS JOIN is a ***cartesian product – every possible combination of rows are in the left and right tables.***





img src: databasestar.com

- **CROSS JOIN** produces a *result set*, which is the number of rows in the first table multiplied by the number of rows in the second table.
- **CROSS JOIN** on very large tables can generate a huge result set.

continue…

Overview of CROSS JOIN Clause (continued)

The following illustrates the syntax of the **CROSS JOIN** clause that joins two tables: t1 and t2:

```
SELECT * FROM t1
CROSS JOIN t2;
```

The **CROSS JOIN** clause does not have a join predicate. In other words, it does not have the **ON** or **USING** keyword.

If you add a **WHERE** clause in case table t1 and t2 have a relationship, the **CROSS JOIN** works like the **INNER JOIN** clause as shown in the following query:

```
SELECT * FROM t1
CROSS JOIN t2
WHERE t1.id = t2.id;
```

Example: CROSS JOIN Clause

The example below uses the CROSS JOIN Clause to join **customers** with **payments**:

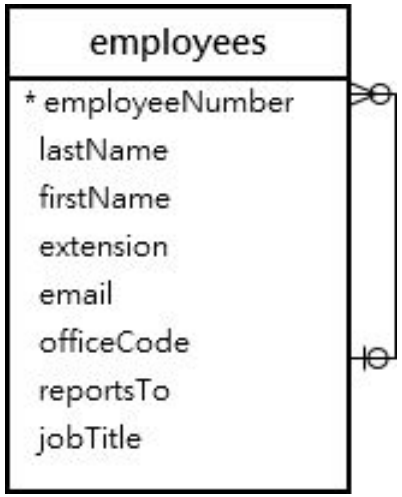
```
SELECT * FROM customers e
CROSS JOIN payments p;
```

Overview of SELF JOIN

There is no SELF JOIN keyword, We just write an ordinary join where both tables involved in the join are the same table

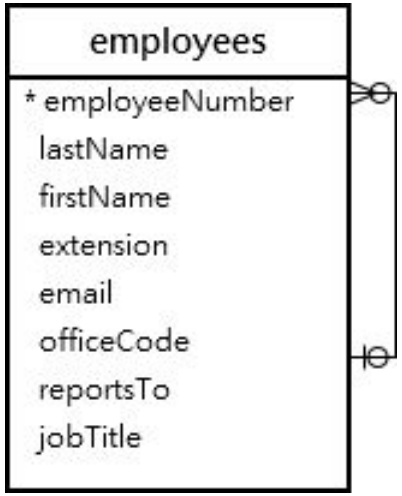
- A **SELF JOIN** allows you to join a table to itself. It helps to query hierarchical data or compare rows within the same table.
- Some tables define a foreign key to their own primary key. A common scenario is an employee table, where each row includes a foreign key to the employee's manager, who is also an employee.

Note: SELF JOINS require the use of a table alias to distinguish the field names.



Overview of SELF JOIN (continued)

- Take a look at the image on the right; this is the **employees** table from the classicmodels database.
- The **employees** table stores not only employee data but also the organizational structure data. The **reportsTo** column is used to determine the manager id of an employee.
- For example, **Patterson** reports to **Murphy** because the value in the **reportsTo** column of **Patterson** is 1002 (**Murphy).**
- **Murphy** has no manager, so the **reportsTo** column has a **NULL** value, as shown below:



employeeNumber	lastName	firstName	extension	email	officeCode	reportsTo	jobTitle
1002	Murphy	Diane	x5800	dmurphy@classicmodelcars.com	1	NULL	President
1056	Patterson	Mary	x4611	mpatterso@classicmodelcars.com	1	1002	VP Sales
1076	Firrelli	Jeff	x9273	jfirrelli@classicmodelcars.com	1	1002	VP Marketing
1088	Patterson	William	x4871	wpatterson@classicmodelcars.com	6	1056	Sales Manager (APAC)
1102	Bondur	Gerard	x5408	gbondur@classicmodelcars.com	4	1056	Sale Manager (EMEA)
1143	Bow	Anthony	x5428	abow@classicmodelcars.com	1	1056	Sales Manager (NA)
1165	Jennings	Leslie	x3291	ljennings@classicmodelcars.com	1	1143	Sales Rep
1166	Thompson	Leslie	x4065	lthompson@classicmodelcars.com	1	1143	Sales Rep

Example 1: SELF JOIN

You can use the SELF JOIN to determine who reports to whom; to do so, we can use the INNER JOIN.

```
SELECT m.employeeNumber AS ManagerID, m.lastName AS Manager,  e.lastName AS 'employee',
e.employeeNumber AS EmployeeID
FROM  employees e
INNER JOIN employees m ON m.employeeNumber = e.reportsTo
ORDER BY m.employeeNumber;
```

ManagerID	Manager	employee	EmployeeID
1002	Murphy	Patterson	1056
1002	Murphy	Firrelli	1076
1056	Patterson	Patterson	1088
1056	Patterson	Bondur	1102
1056	Patterson	Bow	1143
1056	Patterson	Nishi	1621
1088	Patterson	Fixter	1611
1088	Patterson	Marsh	1612
1088	Patterson	King	1619
1102	Bondur	Bondur	1337
1102	Bondur	Hernandez	1370
1102	Bondur	Castillo	1401
1102	Bondur	Bott	1501
1102	Bondur	Jones	1504

sult 67 ×

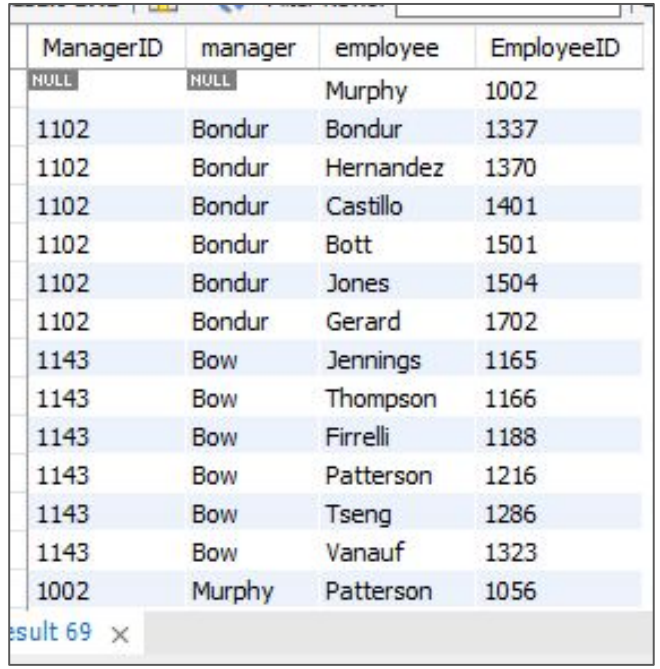
In this example, we referenced the employees table twice; once as **e** for the ***employees*** and the other as **m** for the ***managers***. The JOIN predicate matches the employee and manager relationship using the values in the `m.employeeNumber = e.reportsTo.`

continue…

Example 1: SELF JOIN (continued)

Note: in the previous image that the ***employees*** column does not include **Murphy**; this is because of the INNER JOIN effect. If you replace the ***INNER JOIN clause by the LEFT JOIN*** clause as shown in the following query, you will get the result set that includes **Murphy** in the employee column:

```
SELECT   m.employeeNumber as ManagerID,
m.lastName manager,  e.lastName employee,
e.employeeNumber as EmployeeID
FROM employees e
LEFT JOIN employees m
ON m.employeeNumber = e.reportsTo
ORDER BY manager;
```



ManagerID	manager	employee	EmployeeID
NULL	NULL	Murphy	1002
1102	Bondur	Bondur	1337
1102	Bondur	Hernandez	1370
1102	Bondur	Castillo	1401
1102	Bondur	Bott	1501
1102	Bondur	Jones	1504
1102	Bondur	Gerard	1702
1143	Bow	Jennings	1165
1143	Bow	Thompson	1166
1143	Bow	Firrelli	1188
1143	Bow	Patterson	1216
1143	Bow	Tseng	1286
1143	Bow	Vanauf	1323
1002	Murphy	Patterson	1056

Example 2: SELF JOIN - Compare Successive Rows

By using the **SELF JOIN**, you can display a list of customers who are located in the same city by joining the customers table to itself.

```
SELECT c1.city, c1.customerName, c2.customerName
FROM customers c1
INNER JOIN customers c2
  ON c1.city = c2.city
  AND c1.customername > c2.customerName
ORDER BY c1.city;
```

	city	customerName	customerName
	Auckland	Kelly's Gift Shop	Down Under Souveniers, Inc
	Auckland	GiftsForHim.com	Down Under Souveniers, Inc
	Auckland	Kelly's Gift Shop	GiftsForHim.com
	Boston	Gifts4AllAges.com	Diecast Collectables
	Brickhaven	Online Mini Collectables	Auto-Moto Classics Inc.
	Brickhaven	Collectables For Less Inc.	Auto-Moto Classics Inc.
	Brickhaven	Online Mini Collectables	Collectables For Less Inc.
	Cambridge	Marta's Replicas Co.	Cambridge Collectables Co.
	Frankfurt	Messner Shopping Network	Blauer See Auto, Co.
	Glendale	Gift Ideas Corp.	Boards & Toys Co.
	Lisboa	Porto Imports Co.	Lisboa Souveniers, Inc
	London	Stylish Desk Decors, Co.	Double Decker Gift Stores, Ltd

In this example, the table customer is joined to itself using the following JOIN conditions:

- `c1.city = c2.city` makes sure that both customers have the same city.
- `c.customerName > c2.customerName` ensures that no same customer is included.

Overview of FULL JOIN Clause

The **FULL JOIN** Clause combines the effects of left and right JOINS – all rows of both left and right tables are guaranteed to appear in the result set, regardless of whether the JOIN predicate is satisfied or not.

When any row from either table fails to satisfy the JOIN condition, NULL values will appear in the output:

```
Notice: MySQL does not support FULL JOIN, but the
same result is achieved using UNION operator.
```

Overview of Union Operator

The **UNION** operator allows you to combine two or more result sets of queries into a single result set. The following illustrates the syntax of the UNION operator:

```
SELECT column_list
 UNION [DISTINCT | ALL]
SELECT column_list
 UNION [DISTINCT | ALL]
SELECT column_list
```

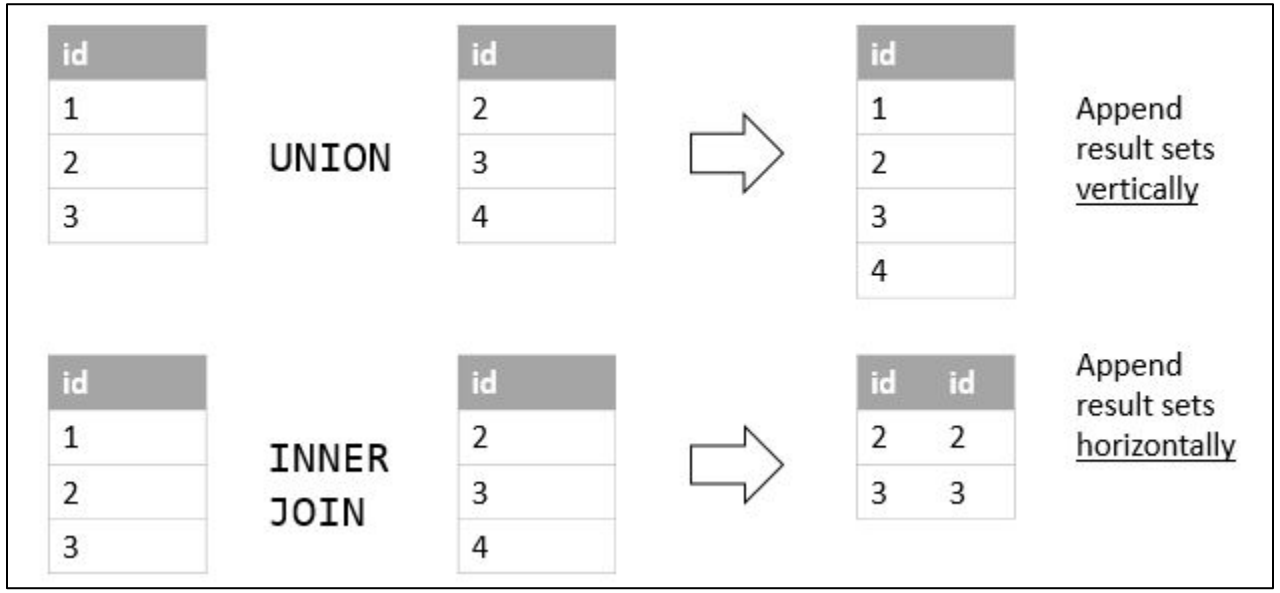
```
select a.value, b.value from a left join b using(id)
union
select a.value, b.value from a right join b using(id);
```

Result

a.value	b.value
E	J
F	K
A	NULL
B	NULL
C	NULL
D	NULL
NULL	L
NULL	M
NULL	N
NULL	O

UNION Operator vs. JOIN Clause

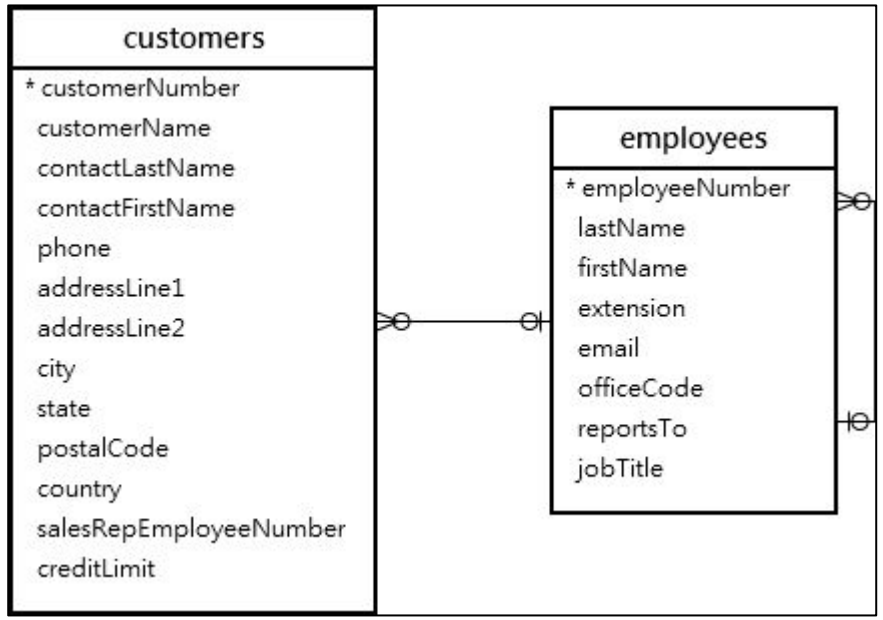
A **JOIN** combines a result set ***horizontally;*** whereas a **UNION** appends a result set ***vertically***. The following illustration shows the difference between UNION and JOIN:



Example: Union Operator

We will use the customers and employees tables for this demonstration.

- Suppose you want to combine the *first name* and *last name* of employees and customers into a single result set. To do this, you can use the **UNION** operator:




```
SELECT firstName, lastName
FROM employees
UNION
SELECT contactFirstName, contactLastName
FROM customers;
```

firstName	lastName
Diane	Murphy
Mary	Patterson
Jeff	Firrelli
William	Patterson
Gerard	Bondur
Anthony	Bow
Leslie	Jennings
Leslie	Thompson
Julie	Firrelli
Steve	Patterson
Foon Yue	Tseng
George	Vanauf
Loui	Bondur
Gerard	Hernandez

Result 72

Joining Without JOIN

Joins can be achieved **without** the JOIN keyword by using the WHERE clause to define the join predicate.

```
select a.value, b.value from A a, B b
where a.id=b.id;
```

This style of joining is called "**Theta Style.**" A Theta Style is a JOIN that links tables based on a relationship other than equality between two columns. A Theta Style could use any operator other than the "equal" operator.

The **JOIN** keyword offers distinct advantages over Theta Style join

- The left and right tables are clear
- Left, right, and full joins are possible.
- **"Using"** keyword syntax is possible.

Hands-On Lab

Complete the following Labs. You can find them on Canvas under the **Assignment** section.

- GLAB - 304.6.1 - Joins and Clauses - Classicmodels Database
- GLAB - 304.6.2 - Joins and Clauses - Banking Database

If you have any technical questions while performing the lab activity, ask your instructors for assistance.

Practice Assignments

1. **Complete the following practice assignments:**
   A. The JOIN operation
   B. JOIN and clauses
   C. Self join
   D. SQL JOINS - Exercises, Practice, Solution - JOINS

2. **Complete the following practice assignments (administered through HackerRank).**
   A. Population Census
   B. African Cities
   C. The PADs

Note: Use your office hours to complete this assignment. If you have any technical questions while performing the assignment activity, ask your instructors for assistance.

Check Your Knowledge

- What Are SQL JOINs?
- What are the different types of JOINS in SQL?
- How can you join a table to itself?

Summary

In this presentation, we discussed JOIN queries, which allow us to walk through the relationships between two or more tables in the FROM clause. JOINS are queries that combine the data of multiple tables based on their common columns and constraints to produce a combined result set:

- A JOIN implemented without any other modifiers is by default an INNER JOIN.
- A Right JOIN returns all records from the right table, and the matched records from the left table; while a Left JOIN returns all records from the left table and the matched records from the right table
- An INNER JOIN returns records that have matching values in both tables.
- A CROSS JOIN produces a result set, which is the number of rows in the first table, multiplied by the number of rows in the second table.
- A SELF JOIN returns a table that is joined with itself.
- A JOIN implemented without the JOIN keyword is called Theta Style.

References

https://en.wikipedia.org/wiki/Join_(SQL)

https://www.techonthenet.com/mysql/group_by.php

https://www.techonthenet.com/mysql/between.php

Questions ?