SQL joins

```
SELECT orders.*
FROM orders
JOIN accounts
ON orders.account_id = accounts.id;
```

Above, we are only pulling data from the **orders** table since in the SELECT statement we only reference columns from the **orders** table.

For example, if we want to pull only the **account name** and the dates in which that account placed an order, but none of the other columns, we can do this with the following query:

```
SELECT accounts.name, orders.occurred_at
FROM orders
JOIN accounts
ON orders.account_id = accounts.id;
```

This query only pulls two columns, not all the information in these two tables. Alternatively, the below query pulls all the columns from *both* the **accounts** and **orders** table.

```
SELECT *
FROM orders
JOIN accounts
ON orders.account_id = accounts.id;
```

1. And the first query (SELECT orders.*) you ran pull all the information from *only* the orders table

Important tip and example.

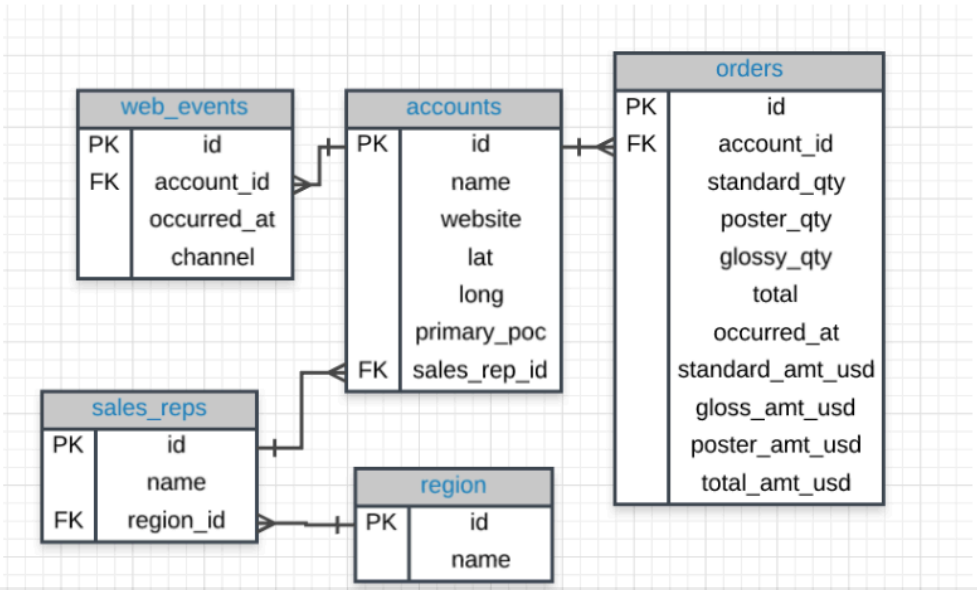
```
SELECT orders.*, accounts.*
FROM accounts
JOIN orders
ON accounts.id = orders.account_id;
```

Notice this result is the same as if you switched the tables in the **FROM** and **JOIN**. Additionally, which side of the `=` a column is listed doesn't matter.

Entity Relationship Diagrams

From the last lesson, you might remember that an **entity relationship diagram** (ERD) is a common way to view data in a database. It is also a key element to understanding how we can pull data from multiple tables.

It will be beneficial to have an idea of what the ERD looks like for Parch & Posey handy, so I have posted it again below. **You might even print a copy to have with you as you work through the exercises in the remaining content.**



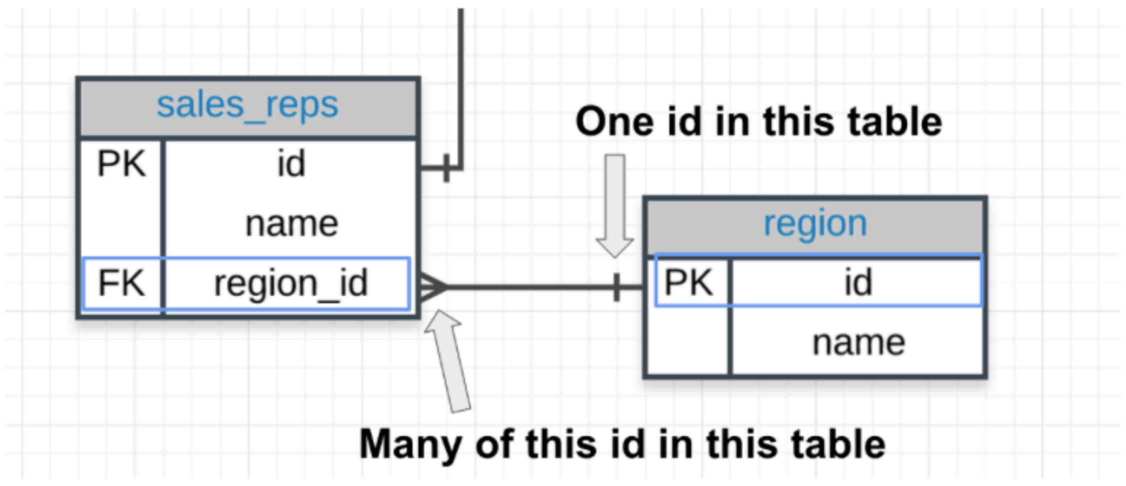
Keys

Primary Key (PK)

A **primary key** is a unique column in a particular table. This is the first column in each of our tables. Here, those columns are all called **id**, but that doesn't necessarily have to be the name. **It is common that the primary key is the first column in our tables in most databases.**

Foreign Key (FK)

A **foreign key** is a column in one table that is a primary key in a different table. We can see in the Parch



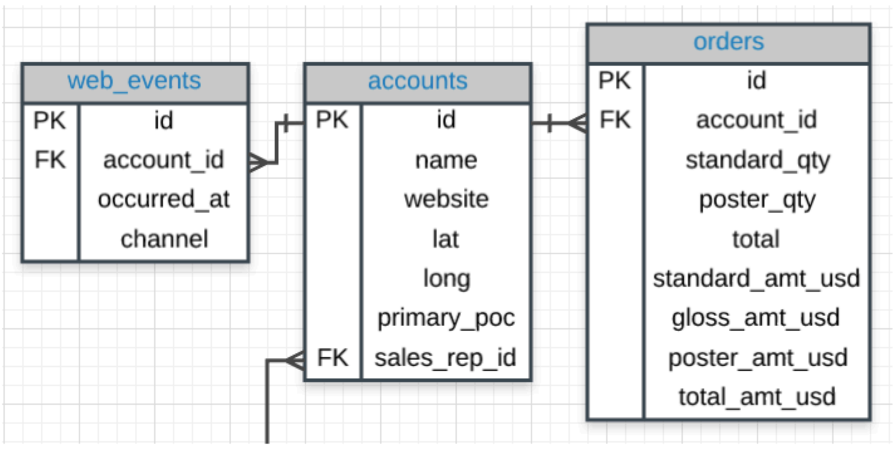
Primary - Foreign Key Link

In the above image you can see that:

1. The **region_id** is the foreign key.
2. The region_id is **linked** to id - this is the primary-foreign key link that connects these two tables.
3. The crow's foot shows that the **FK** can actually appear in many rows in the **sales_reps** table.
4. While the single line is telling us that the **PK** shows that id appears only once per row in this table.

JOIN More than Two Tables

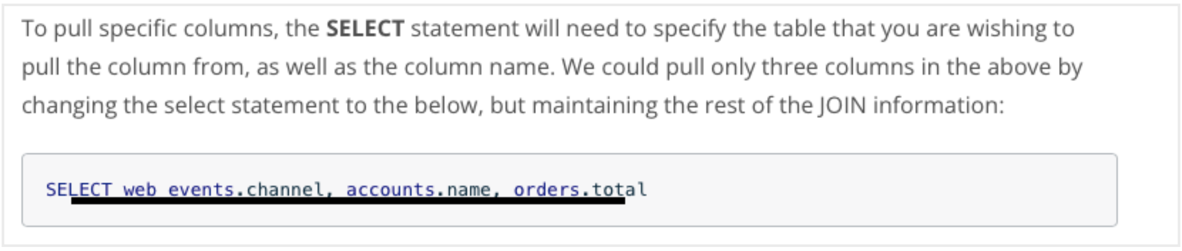
This same logic can actually assist in joining more than two tables together. Look at the three tables below.



The Code

If we wanted to join all three of these tables, we could use the same logic. The code below pulls all of the data from all of the joined tables.

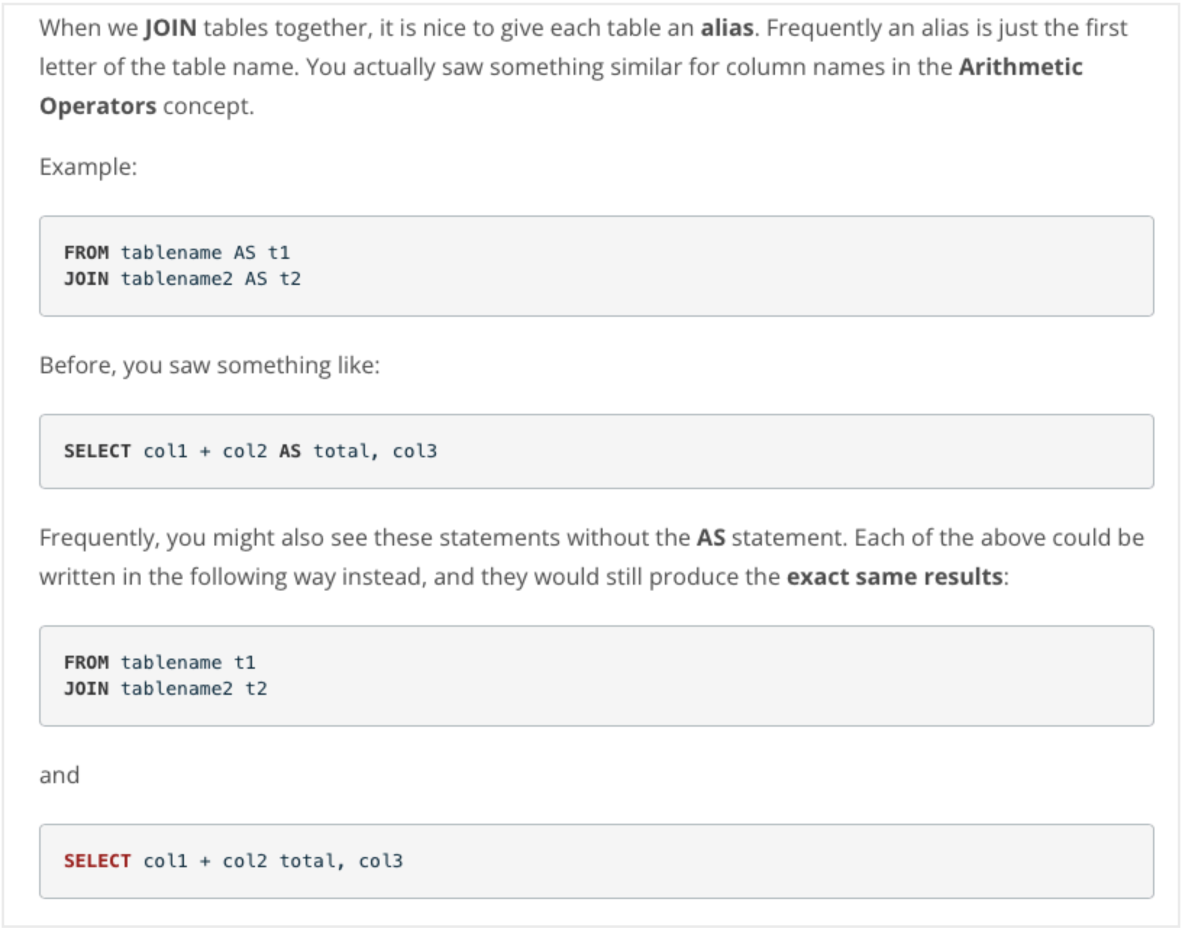
```
SELECT *
FROM web_events
JOIN accounts
ON web_events.account_id = accounts.id
JOIN orders
ON accounts.id = orders.account_id
```



To pull specific columns, the **SELECT** statement will need to specify the table that you are wishing to pull the column from, as well as the column name. We could pull only three columns in the above by changing the select statement to the below, but maintaining the rest of the JOIN information:

```
SELECT web_events.channel, accounts.name, orders.total
```

Joining Table by giving Alias.



When we **JOIN** tables together, it is nice to give each table an **alias**. Frequently an alias is just the first letter of the table name. You actually saw something similar for column names in the **Arithmetic Operators** concept.

Example:

```
FROM tablename AS t1
JOIN tablename2 AS t2
```

Before, you saw something like:

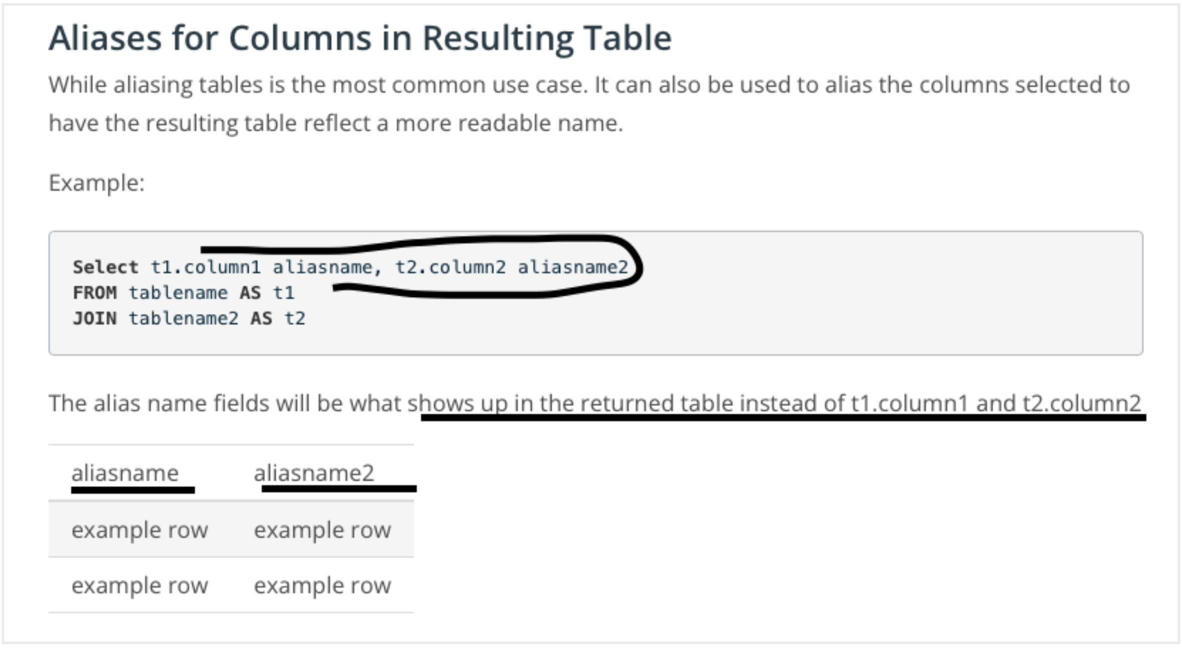
```
SELECT col1 + col2 AS total, col3
```

Frequently, you might also see these statements without the **AS** statement. Each of the above could be written in the following way instead, and they would still produce the **exact same results**:

```
FROM tablename t1
JOIN tablename2 t2
```

and

```
SELECT col1 + col2 total, col3
```



Aliases for Columns in Resulting Table

While aliasing tables is the most common use case. It can also be used to alias the columns selected to have the resulting table reflect a more readable name.

Example:

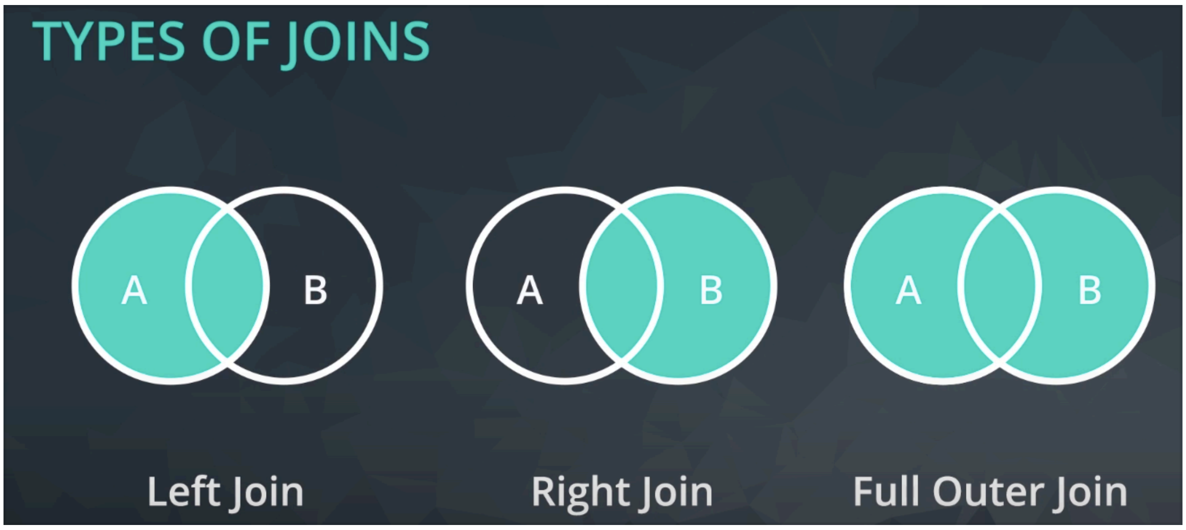
```
Select t1.column1 aliasname, t2.column2 aliasname2
FROM tablename AS t1
JOIN tablename2 AS t2
```

The alias name fields will be what shows up in the returned table instead of t1.column1 and t2.column2

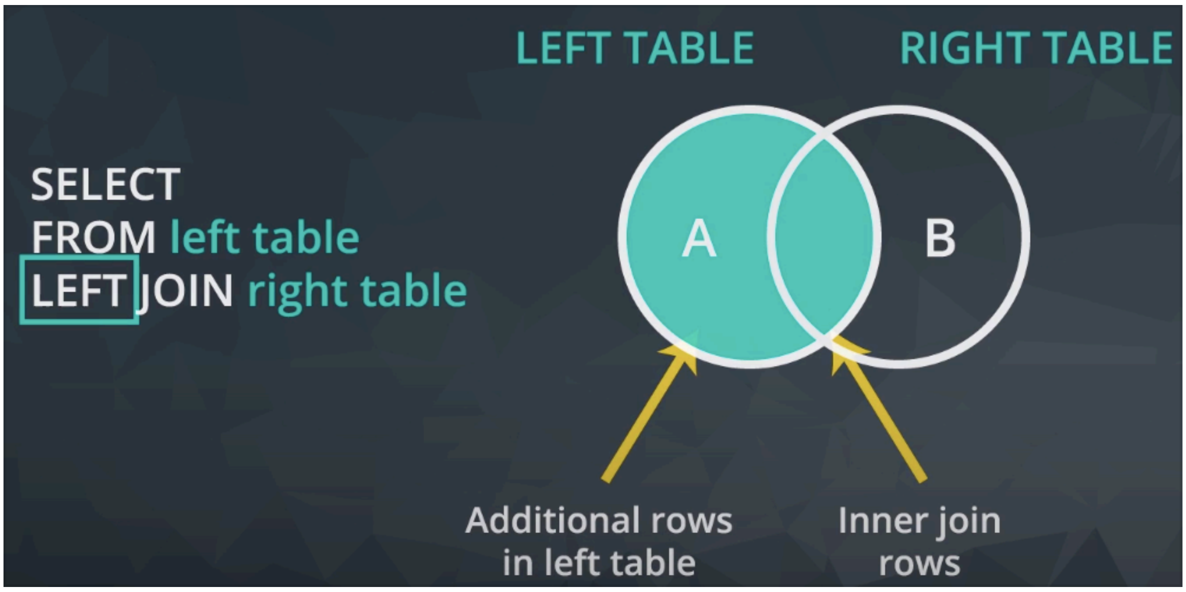
aliasname	aliasname2
example row	example row
example row	example row

LEFT and Right Join

The Left and the right join contain all the data of an inner join / Join and the extra rows in the left or right table.

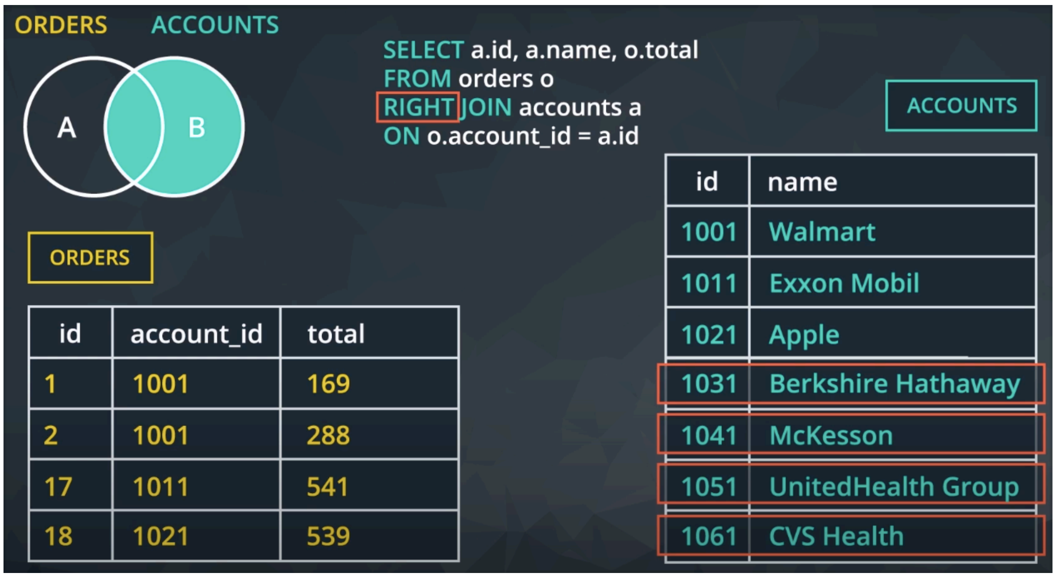


Syntax of LEFT join. All the rows of inner join and additional rows of left table are included.



```
SELECT
FROM left table
LEFT JOIN right table
```

And if the table name Is written after right/left join statement then that is considered as right table and hence the result be as follows :



```
SELECT a.id, a.name, o.total
FROM orders o
RIGHT JOIN accounts a
ON o.account_id = a.id
```

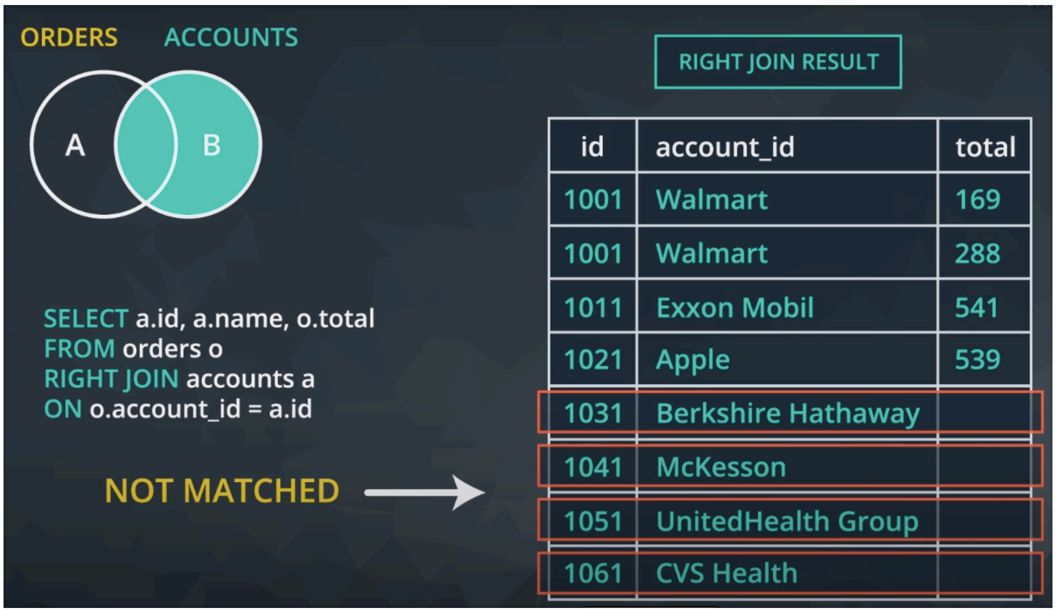
ORDERS

id	account_id	total
1	1001	169
2	1001	288
17	1011	541
18	1021	539

ACCOUNTS

id	name
1001	Walmart
1011	Exxon Mobil
1021	Apple
1031	Berkshire Hathaway
1041	McKesson
1051	UnitedHealth Group
1061	CVS Health

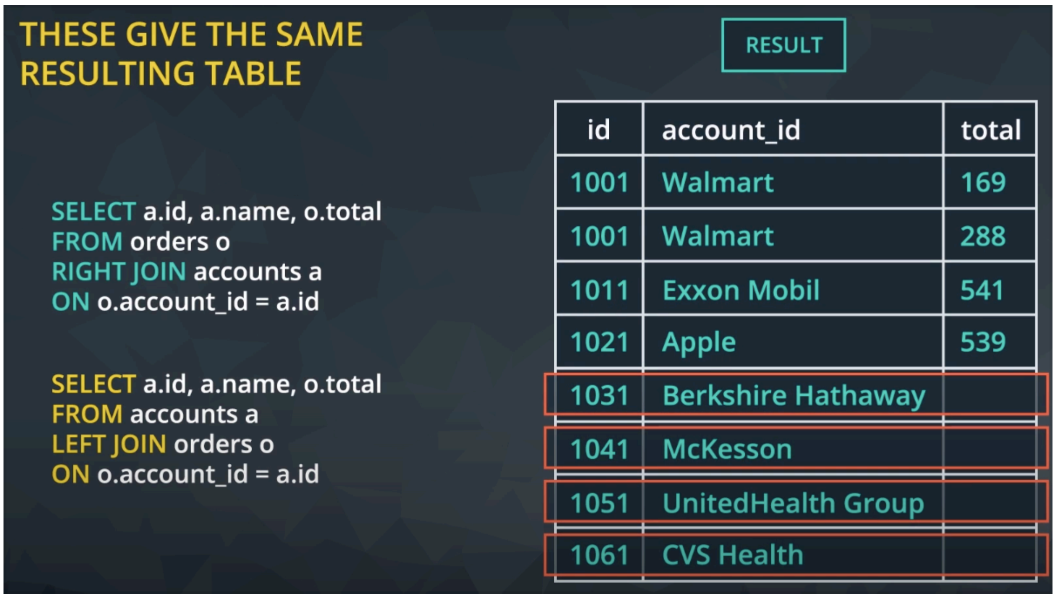
Result of the Right join table written above:



```
SELECT a.id, a.name, o.total
FROM orders o
RIGHT JOIN accounts a
ON o.account_id = a.id
```

id	account_id	total
1001	Walmart	169
1001	Walmart	288
1011	Exxon Mobil	541
1021	Apple	539
1031	Berkshire Hathaway	
1041	McKesson	
1051	UnitedHealth Group	
1061	CVS Health	

Important : If we use left/right join after interchanging table name , then the results will be exactly the same.



THESE GIVE THE SAME RESULTING TABLE

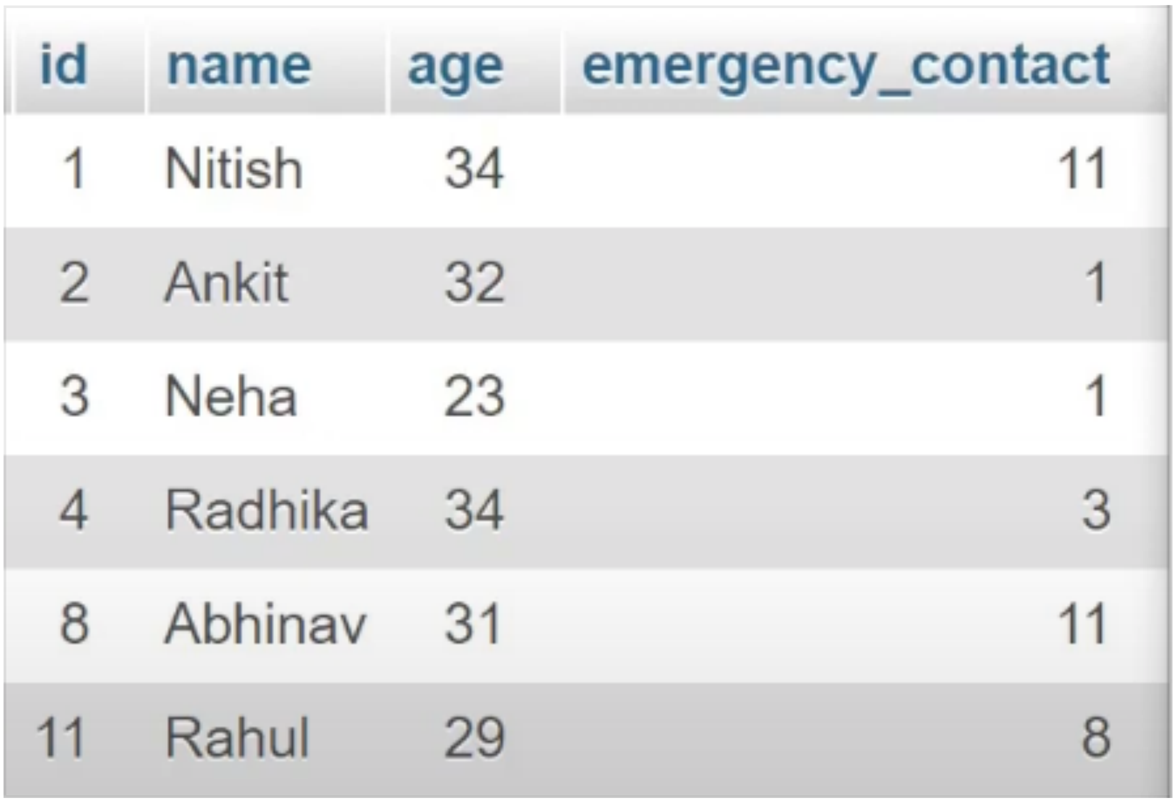
```
SELECT a.id, a.name, o.total
FROM orders o
RIGHT JOIN accounts a
ON o.account_id = a.id

SELECT a.id, a.name, o.total
FROM accounts a
LEFT JOIN orders o
ON o.account_id = a.id
```

id	account_id	total
1001	Walmart	169
1001	Walmart	288
1011	Exxon Mobil	541
1021	Apple	539
1031	Berkshire Hathaway	
1041	McKesson	
1051	UnitedHealth Group	
1061	CVS Health	

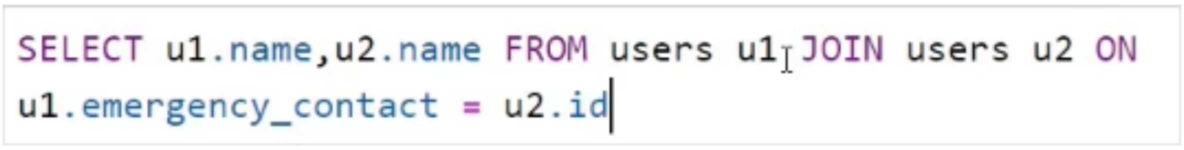
SELF-JOIN

If we need to show emergency contact name of all the users , we have to do using self join.



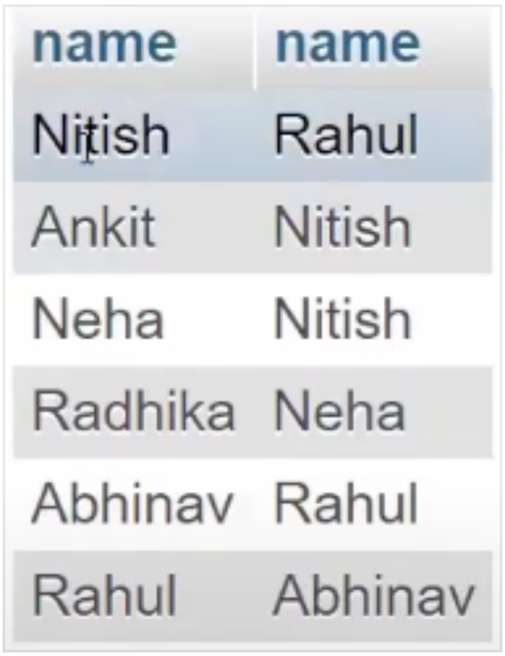
id	name	age	emergency_contact
1	Nitish	34	11
2	Ankit	32	1
3	Neha	23	1
4	Radhika	34	3
8	Abhinav	31	11
11	Rahul	29	8

The query will be as follows:

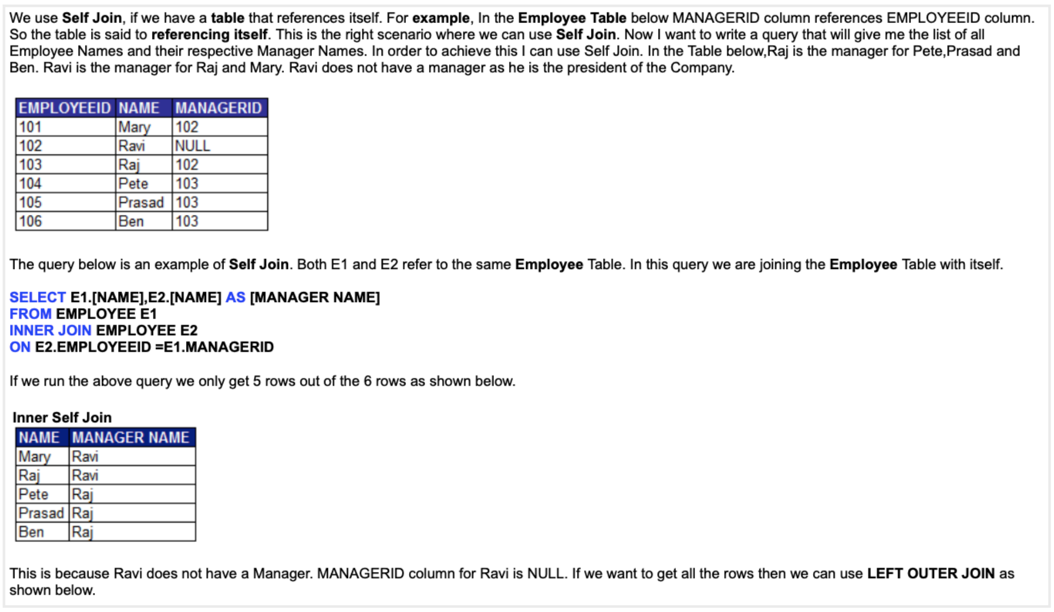


```
SELECT u1.name,u2.name FROM users u1 JOIN users u2 ON
u1.emergency_contact = u2.id
```

Result will be :



name	name
Nitish	Rahul
Ankit	Nitish
Neha	Nitish
Radhika	Neha
Abhinav	Rahul
Rahul	Abhinav



We use **Self Join**, if we have a **table** that references itself. For **example**, In the **Employee Table** below MANAGERID column references EMPLOYEEID column. So the table is said to **referencing itself**. This is the right scenario where we can use **Self Join**. Now I want to write a query that will give me the list of all Employee Names and their respective Manager Names. In order to achieve this I can use Self Join. In the Table below,Raj is the manager for Pete,Prasad and Ben. Ravi is the manager for Raj and Mary. Ravi does not have a manager as he is the president of the Company.

EMPLOYEEID	NAME	MANAGERID
101	Mary	102
102	Ravi	NULL
103	Raj	102
104	Pete	103
105	Prasad	103
106	Ben	103

The query below is an example of **Self Join**. Both E1 and E2 refer to the same **Employee** Table. In this query we are joining the **Employee** Table with itself.

```sql
SELECT E1.[NAME],E2.[NAME] AS [MANAGER NAME]
FROM EMPLOYEE E1
INNER JOIN EMPLOYEE E2
ON E2.EMPLOYEEID =E1.MANAGERID
```

If we run the above query we only get 5 rows out of the 6 rows as shown below.

Inner Self Join

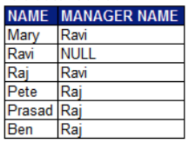
NAME	MANAGER NAME
Mary	Ravi
Raj	Ravi
Pete	Raj
Prasad	Raj
Ben	Raj

This is because Ravi does not have a Manager. MANAGERID column for Ravi is NULL. If we want to get all the rows then we can use **LEFT OUTER JOIN** as shown below.

```
SELECT E1.[NAME],E2.[NAME] AS [MANAGER NAME]
FROM EMPLOYEE E1
LEFT OUTER JOIN EMPLOYEE E2
ON E2.EMPLOYEEID =E1.MANAGERID
```

If we execute the above query we get all the rows, including the row that has a null value in the MANAGERID column. The results are shown below. The MANAGERNAME for 2nd record is NULL as Ravi does not have a Manager.

Left Outer Self Join



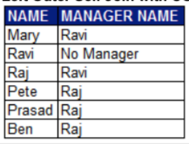
NAME	MANAGER NAME
Mary	Ravi
Ravi	NULL
Raj	Ravi
Pete	Raj
Prasad	Raj
Ben	Raj

Let us now slightly modify the above query using **COALESCE** as shown below. Read COALESCE function in SQL Server to understand **COALESCE** in a greater detail.

```
SELECT E1.[NAME],COALESCE(E2.[NAME],'No Manager') AS [MANAGER NAME]
FROM EMPLOYEE E1
LEFT JOIN EMPLOYEE E2
ON E2.EMPLOYEEID =E1.MANAGERID
```

If we execute the above query the output will be as shown in the image below. This is how **COALESCE** can be used.

Left Outer Self Join with COALESCE



NAME	MANAGER NAME
Mary	Ravi
Ravi	No Manager
Raj	Ravi
Pete	Raj
Prasad	Raj
Ben	Raj