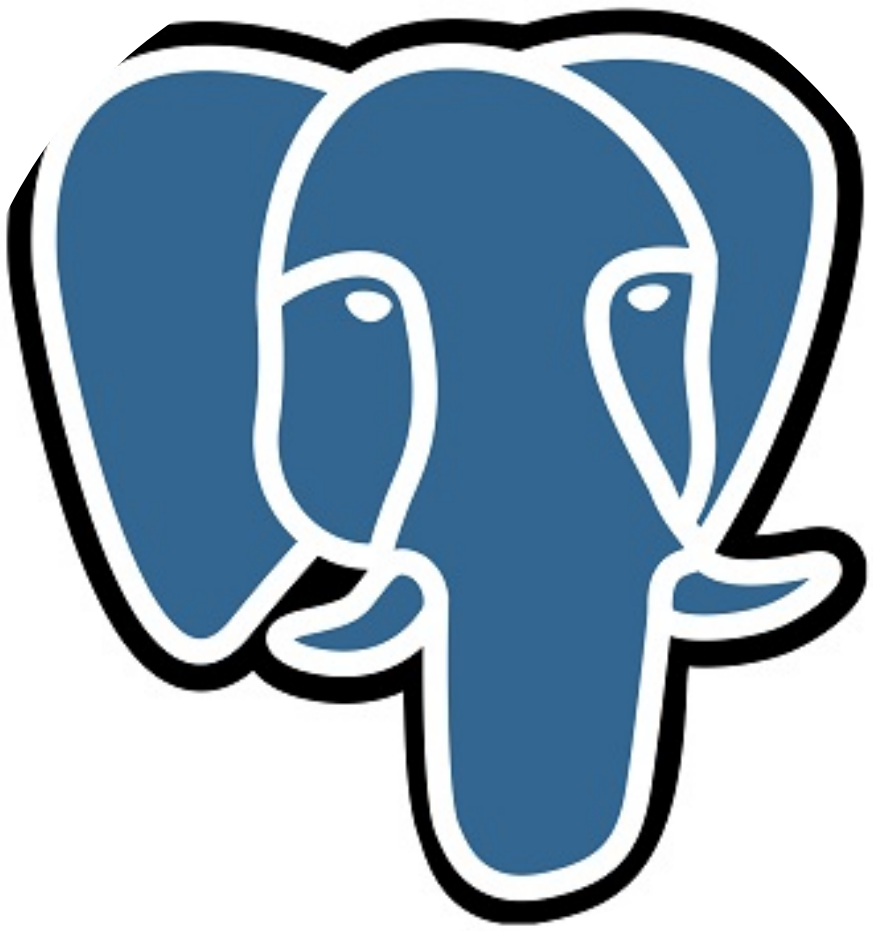


SQL Basics Part III - JOINS

26.10.2019

SQL Basics

AS STATEMENT

★AS allow us to rename columns or table selections with an alias.

Example:

```
SELECT payment_id AS new_column_name
FROM payment_table;
```

★It will return the renamed payment_id column from the payment table.

```
SELECT customer_id, SUM(amount) AS total_spent
FROM payment_table
GROUP BY customer_id;
```

★Now the SUM(amount) column has a better name.

★Note that we can use 'space character' instead of an AS and it will work exactly the same.
★Also besides the column names this statement can be used for table names as well.

JOINS INTRO

★So far we've learned how to select data from a table, choosing which columns and rows we want, and how to sort the result set in a particular order.

★Now we'll move to one of the most important concepts in the database called joining that allows us to relate data in one table to the data in other tables.

★There are several kinds of joins including **INNER JOIN**, **OUTER JOIN** and **self - join**.

WHAT IS A SQL JOIN?

★A SQL join is a Structured Query Language (**SQL**) instruction to combine data from two sets of data (i.e. two tables).

★ SQL is a special-purpose programming language designed for managing information in a relational database management system. The word relational here is key; it specifies that the database management system is organized in such a way that there are clear relations defined between different sets of data.

RELATIONAL DATABASE EXAMPLE

★ Imagine we're running a store and would like to record information about our customers and their orders. By using a relational database, you can save this information as two tables that represent two distinct entities: customers and orders.

Customers

customer_id	first_name	last_name	email	address	city	state	zip
1	George	Washington	gwashington@usa.gov	3200 Mt Vernon Hwy	Mount Vernon	VA	22121
2	John	Adams	jadams@usa.gov	1250 Hancock St	Quincy	MA	02169
3	Thomas	Jefferson	tjefferson@usa.gov	931 Thomas Jefferson Pkwy	Charlottesville	VA	22902
4	James	Madison	jmadison@usa.gov	11350 Constitution Hwy	Orange	VA	22960
5	James	Monroe	jmonroe@usa.gov	2050 James Monroe Pkwy	Charlottesville	VA	22902

★ Note that, we associate a unique customer number, or primary key, with each customer record.

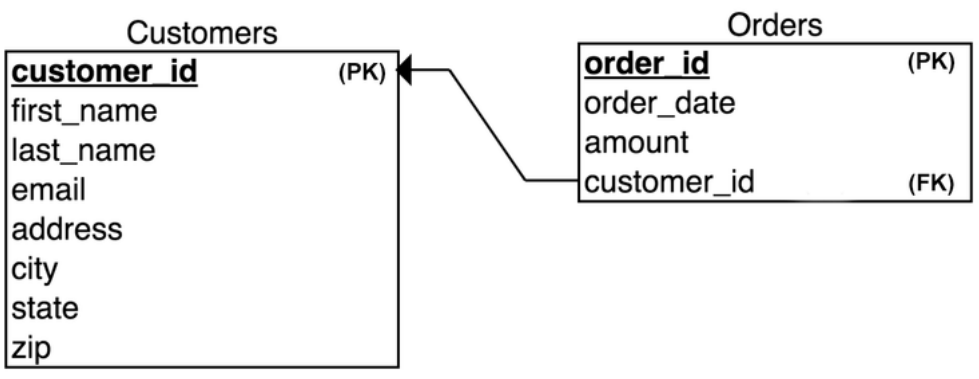
Orders

order_id	order_date	amount	customer_id
1	07/04/1776	$234.56	1
2	03/14/1760	$78.50	3
3	05/23/1784	$124.00	2
4	09/03/1790	$65.50	3
5	07/21/1795	$25.50	10
6	11/27/1787	$14.40	9

★ Again each order has its own unique identification key – order_id for this table – assigned to it as well.

Relational Model

★ These two tables share similar information. You can see these simple relations diagrammed below:



★ Note that the orders table contains two keys: one for the order and one for the customer who placed that order. In scenarios when there are multiple keys in a table, the key that refers to the entity being described in that table is called the **primary key** (PK) and other key is called a **foreign key** (FK).

★ In our example, `order_id` is a primary key in the orders table, while `customer_id` is both a primary key in the customers table and a foreign key in the orders table. Primary and foreign keys are essential to describing relations between the tables, and in performing SQL joins.

SQL JOIN EXAMPLE

★ Let’s say we want to find all orders placed by a particular customer. We can do this by joining the customers and orders tables together using the relationship established by the `customer_id` key:

Syntax:

```
★First, we specify the column in both tables from which you want to select data in the select clause.
★Second, we specify the main table.
★Third, we specify the table that the main table joins to. In addition we put a join condition after the ON keyword.
```

Example:

```
SELECT order_date, order_amount
FROM customers
JOIN orders ON customers.customer_id = orders.customer_id
WHERE customer_id = 3;
```

★ Here, we're joining the two tables using the `join` keyword, and specifying what key to use when joining the tables.
★ Eğer belirtilen column sadece tek bir tabloda var ise direkt column adı belirtilebiliyor ama ikisinde de varsa **table.column** şeklinde belirtmemiz gerekiyor.

Here is the result of the above SQL query, which includes two orders placed by Thomas Jefferson (customer_id = 3):

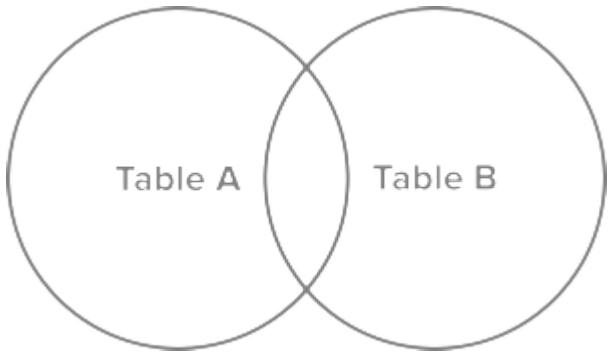
order_id	order_date	order_amount
2	3/14/1760	$78.50
4	9/03/1790	$65.50

★ This particular join is an example of an "inner" join. Depending on the kind of analysis you'd like to perform, you may want to use a different method. There are actually a number of different ways to join the two tables together, depending on your application.

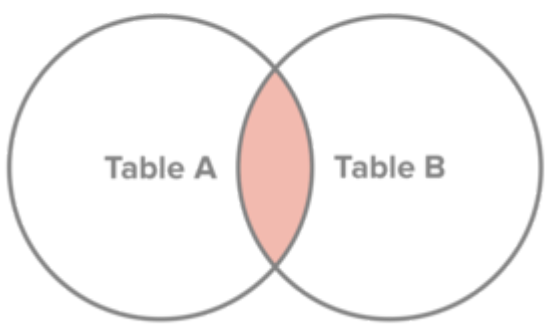
SQL JOIN TYPES

★ There are four basic types of SQL joins: **inner**, **left**, **right**, and **full**. The easiest and most intuitive way to explain the difference between these four types is by using a Venn diagram, which shows all possible logical relations between data sets.

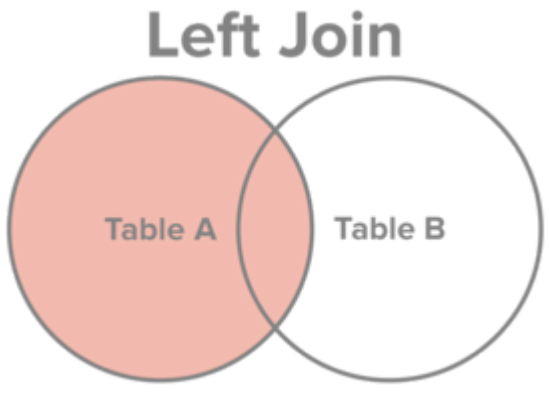
★ Let's say we have two sets of data in our relational database: table A and table B, with some sort of relation specified by primary and foreign keys. The result of joining these tables together can be visually represented by the following diagram:



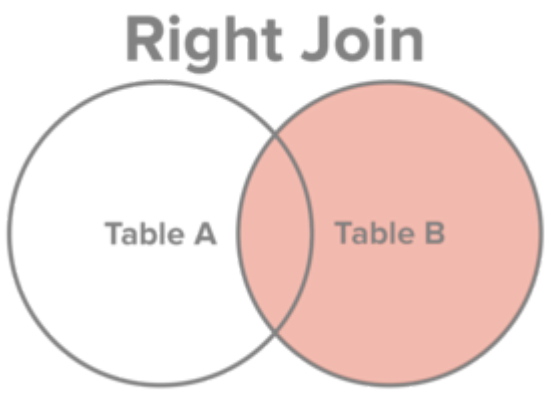
★ The extent of the overlap, if any, is determined by how many records in Table A match the records in Table B. Depending on what subset of data we would like to select from the two tables, the four join types can be visualized by highlighting the corresponding sections of the Venn diagram:



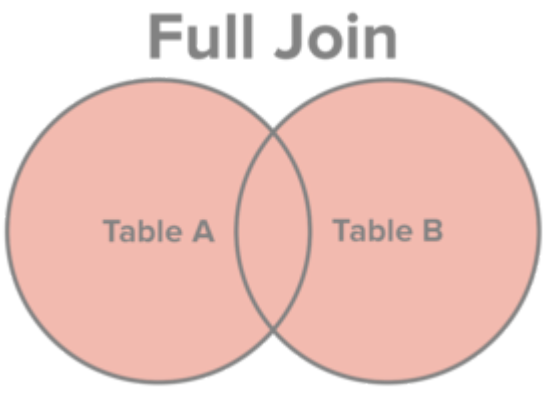
Select all records from Table A and Table B, where the join condition is met.



Select all records from Table A, along with records from Table B for which the join condition is met (if at all).



Select all records from Table B, along with records from Table A for which the join condition is met (if at all).



Select all records from Table A and Table B, regardless of whether the join condition is met or not.

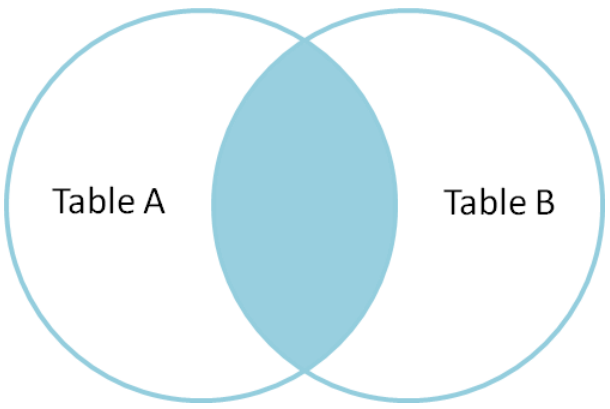
★ Let's use the tables we introduced in the “What is a SQL join?” section to show examples of these joins in action. The relationship between the two tables is specified by the `customer_id` key, which is the "primary key" in customers table and a "foreign key" in the orders table:

customer_id	first_name	last_name	email	address	city	state	zipcode
1	George	Washington	gwashington@usa.gov	3200 Mt Vernon Hwy	Mount Vernon	VA	22121
2	John	Adams	jadams@usa.gov	1250 Hancock St	Quincy	MA	02169
3	Thomas	Jefferson	tjefferson@usa.gov	931 Thomas Jefferson Pkwy	Charlottesville	VA	22902
4	James	Madison	jmadison@usa.gov	11350 Constitution Hwy	Orange	VA	22960
5	James	Monroe	jmonroe@usa.gov	2050 James Monroe Parkway	Charlottesville	VA	22902

order_id	order_date	amount	customer_id
1	07/04/1776	$234.56	1
2	03/14/1760	$78.50	3
3	05/23/1784	$124.00	2
4	09/03/1790	$65.50	3
5	07/21/1795	$25.50	10
6	11/27/1787	$14.40	9

INNER JOIN

★ Let’s say we wanted to get a list of those customers who placed an order and the details of the order they placed.
★ This would be a perfect fit for an inner join since an inner join returns records at the intersection of the two tables.



Example:

```
SELECT first_name,last_name,order_date,order_amount
FROM customers
INNER JOIN orders ON customers.customer_id = orders.customer_id;
```

first_name	last_name	order_date	order_amount
George	Washington	07/4/1776	$234.56
John	Adams	05/23/1784	$124.00
Thomas	Jefferson	03/14/1760	$78.50
Thomas	Jefferson	09/03/1790	$65.50

★ Note that only George Washington, John Adams and Thomas Jefferson placed orders, with Thomas Jefferson placing two separate orders on 3/14/1760 and 9/03/1790.
★ Inner join ile hangi müşterilerimizin sipariş verdiğini ve ilgili siparişlerin detayını görmüş oluyoruz.

How It Works:

★ For each row in the main table customers, PostgreSQL scans the other table orders to check if any row that matches the condition.
★If it finds a match, it combines columns of both rows into one row and add the combined row to the returned result set.
★If it finds another match, it repeats the process and add another row to the returned result set.

Example:

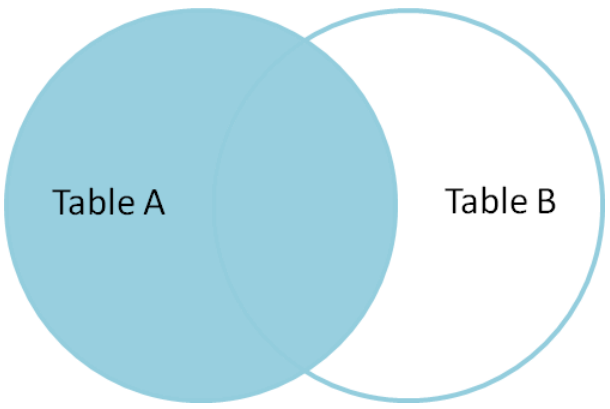
```
SELECT title, COUNT(store_id) AS copies_at_store1
FROM film
INNER JOIN inventory  ON inventory.film_id = film.film_id
```

```
WHERE store_id=1
GROUP BY title
ORDER BY title;
```

★First, the joining happens, then where statement make an elimination, later the resultant set is grouped by title and count function is executed, finally the set is ordered.

LEFT OUTER JOIN

★ If we wanted to simply append information about orders to our customers table, regardless of whether a customer placed an order or not, we would use a left join. A left join returns all records from table A and any matching records from table B.



Example:

```
SELECT first_name,last_name,order_date,order_amount
FROM customers
LEFT JOIN orders ON customers.customer_id = orders.customer_id;
```

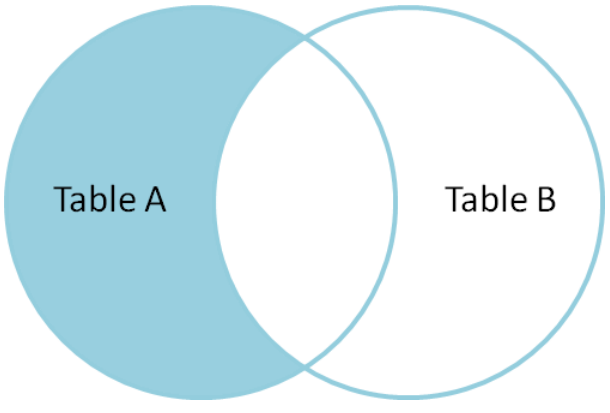
first_name	last_name	order_date	order_amount
George	Washington	07/04/1776	$234.56
John	Adams	05/23/1784	$124.00
Thomas	Jefferson	03/14/1760	$78.50
Thomas	Jefferson	09/03/1790	$65.50
James	Madison	NULL	NULL
James	Monroe	NULL	NULL

★Temelde inner join üzerine main table'ın diğer elemanları da ekleniyor.
★Yeni tablo ile artık tüm müşteriler için sipariş bilgisini görebiliyoruz

---

★ Note that since there were no matching records for James Madison and James Monroe in our orders table, the `order_date` and `order_amount` are `NULL`, which simply means there is no data for these fields.

★ So why would this be useful? By simply adding a "`WHERE order_date IS NULL`" line to our SQL query, it returns a list of all customers who have not placed an order:



Example:

```
SELECT first_name,last_name,order_date,order_amount
FROM customers
LEFT JOIN orders ON customers.customer_id = orders.customer_id;
WHERE order_date IS NULL
```

James	Madison	NULL	NULL
James	Monroe	NULL	NULL

RIGHT OUTER JOIN

★ Right join is a mirror version of the left join and allows to get a list of all orders, appended with customer information.

Example:

```
SELECT first_name,last_name,order_date,order_amount
FROM customers
RIGHT JOIN orders ON customers.customer_id = orders.customer_id;
```

first_name	last_name	order_date	order_amount
George	Washington	07/04/1776	$234.56
Thomas	Jefferson	03/14/1760	$78.50
John	Adams	05/23/1784	$124.00
Thomas	Jefferson	09/03/1790	$65.50
NULL	NULL	07/21/1795	$25.50
NULL	NULL	11/27/1787	$14.40

★ Note that since there were no matching customer records for orders placed in 1795 and 1787, the first_name and last_name fields are NULL in the resulting set.
★ Why is this useful? Simply adding a "**WHERE** first_name **IS NULL**" line to our SQL query returns a list of all orders for which we failed to record information about the customers who placed them:

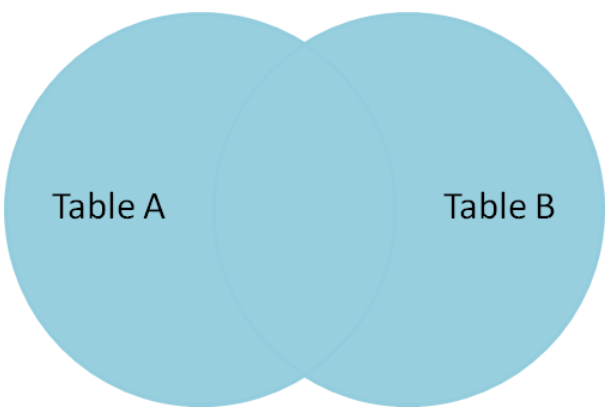
```
SELECT first_name,last_name,order_date,order_amount
FROM customers
RIGHT JOIN orders ON customers.customer_id = orders.customer_id
WHERE first_name IS NULL;
```

NULL	NULL	07/21/1795	$25.50
NULL	NULL	11/27/1787	$14.40

★ Aslında RIGHT JOIN yerine LEFT JOIN'i bırakıp yalnızca main table ile diğer table'ın yerlerini değiştirsek yine aynı şey oluyor.

FULL OUTER JOIN

★ Finally, for a list of all records from both tables, we can use a full join.



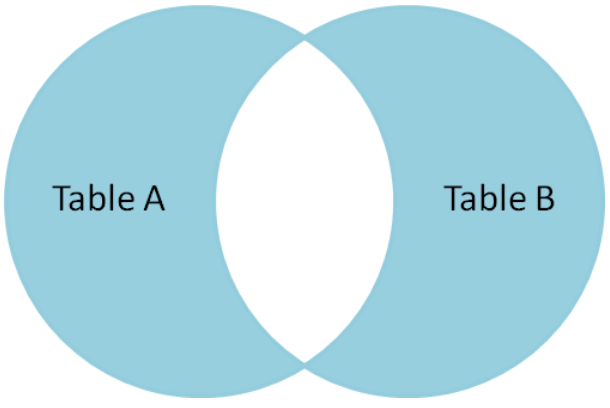
Example:

```
SELECT first_name,last_name,order_date,order_amount
FROM customers
FULL JOIN orders ON customers.customer_id = orders.customer_id;
```

first_name	last_name	order_date	order_amount
George	Washington	07/04/1776	$234.56
Thomas	Jefferson	03/14/1760	$78.50
John	Adams	05/23/1784	$124.00
Thomas	Jefferson	09/03/1790	$65.50

NULL	NULL	07/21/1795	$25.50
NULL	NULL	11/27/1787	$14.40
James	Madison	NULL	NULL
James	Monroe	NULL	NULL

★ As before we can use full outer join with a where statement and find both the customer entries without orders and order entries without customers.




```
SELECT first_name,last_name,order_date,order_amount
FROM customers
FULL JOIN orders ON customers.customer_id = orders.customer_id;
WHERE first_name IS NULL OR order_date IS NULL;
```

NULL	NULL	07/21/1795	$25.50
NULL	NULL	11/27/1787	$14.40
James	Madison	NULL	NULL
James	Monroe	NULL	NULL

UNION STATEMENT

★The **UNION** operator combines result sets of two or more SELECT statements into a single result set.
★We often use the **UNION** operator to combine data from similar tables that are not perfectly normalized

---

Syntax:

```
SELECT column1,column2
FROM table1
UNION
SELECT columnA,columnB
FROM table2
```

★ There are 2 major rules for the UNION statement:

- Both queries must return the same number of columns.
- The corresponding columns in the queries must have compatible data types.

★ Note that The **UNION** operator removes all duplicate rows unless the **UNION ALL** is used.

★The **UNION** operator may place the rows in the first query before,after or between the rows in the result set of the second query. (Yani doğrudan uç uca yapıştırmıyor da datalar iç içe giriyor.)

More blogs