Digital Career Institute

Python Course - Advanced SQL

Digital Career Institute

Goal of the Submodule

The goal of this submodule is to help the student understand how to work with advanced SQL queries and data types. By the end of this submodule, the learners will be able to understand:

- How to query multiple tables using different types of `JOIN` clauses.
- How do `JOIN` operations work on the background.
- How to use advanced SQL data types.
- How to group and use aggregate functions to extract information from the data tables.
- How to combine all this knowledge to define complex analytical SQL queries.

Topics

- How to query multiple tables.
- How to query multiple tables using different types of `JOIN` clauses.
- How to define and use advanced data types.
- How to group and calculate statistics on a set data tables.

Digital Career Institute

Joining Multiple Tables

Introducing JOIN

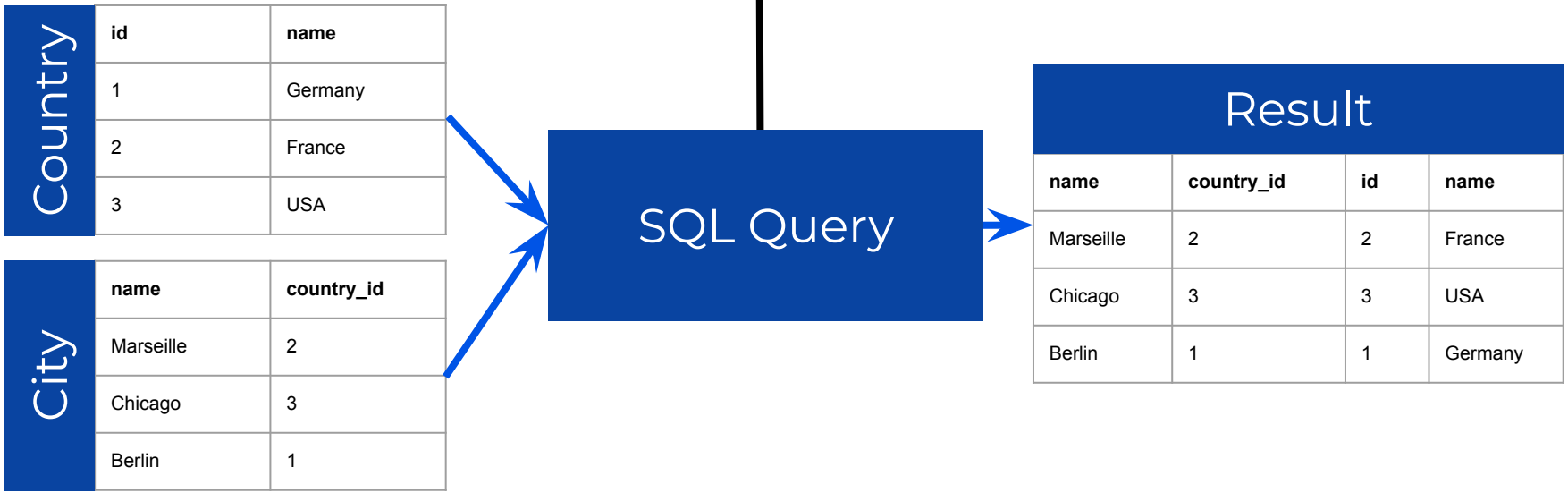
```
SELECT fields
FROM table_a
JOIN table_b on conditions;
```

The **`JOIN`** clause takes all combinations of records between **`table_a`** and **`table_b`** and returns those that match the indicated **`conditions`**.

The **`conditions`** **MUST** include the equality between the foreign key and primary key of those tables.

Introducing JOIN

```
SELECT *
FROM City
JOIN Country ON Country.id = City.country_id;
```



Implicit JOIN

Modern, default joining syntax.

Non implicit

```
SELECT City.name, Country.name
FROM City
JOIN Country ON Country.id = City.country_id;
```

=

Older, but still used sometimes.

Implicit

```
SELECT City.name, Country.name
FROM City, Country
WHERE Country.id = City.country_id;
```

UPDATE Across Tables

Implicit

```
UPDATE City SET country = Country.name
FROM Country WHERE City.country_id = Country.id;
```

In some RDBMS, like PostgreSQL, an ***UPDATE*** *across tables is still done with an implicit join.*

JOIN & Order

```
SELECT *
FROM City
JOIN Country on Country.id = City.country_id
ORDER BY Country.name;
```

Country

id	name
1	Germany
2	France
3	USA

City

name	country_id
Marseille	2
Chicago	3
Berlin	1

SQL Query

Result

name	country_id	id	name
Marseille	2	2	France
Berlin	1	1	Germany
Chicago	3	3	USA

Types of JOIN

But what if ...

Country

id	name
1	Germany
2	France
3	USA
4	Spain

There are countries not being used in our city table.

City

name	country_id
Marseille	2
Chicago	3
Berlin	1
Barcelona	
Salzburg	7

There are cities with `null` country_id or an id that does not exist.

Types of JOIN

The result is still the same.

The results include only those combinations of records that match the `on` condition.

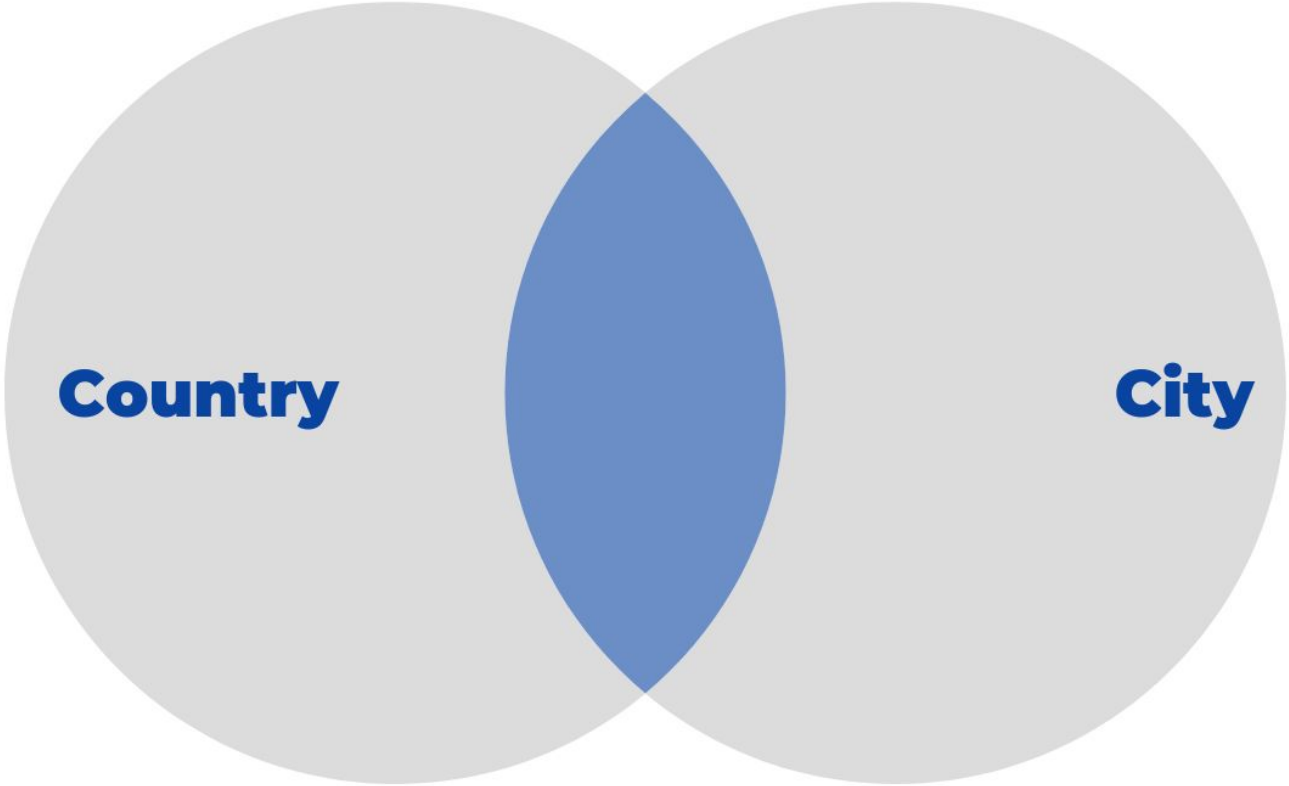
Result

name	country_id	id	name
Marseille	2	2	France
Chicago	3	3	USA
Berlin	1	1	Germany

This type of join is called ***INNER JOIN***.

Types of JOIN

INNER JOIN



INNER JOIN

The `JOIN` clause is a shortcut for the `INNER JOIN` clause.

For clarity purposes, it is usually preferred to use `INNER JOIN`.

```
SELECT {fields}
FROM {table_a}
INNER JOIN {table_b} on {conditions};
```

Other Types of JOIN

Very often, there is a main table and a secondary table.

Losing all the records in the main table whose foreign key is not found on the secondary table may not be desirable.

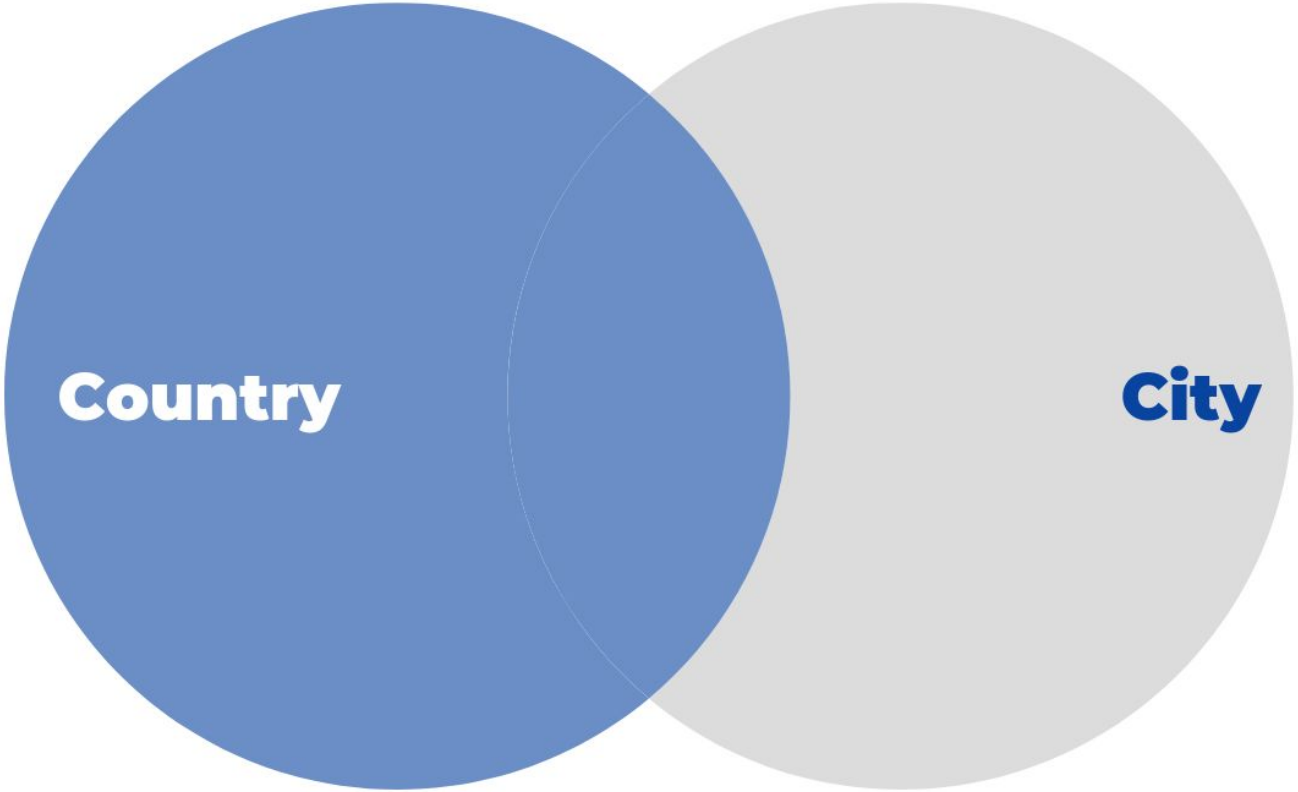
Very often, the project requires to create this kind of result.

Result

name	country_id	id	name
Marseille	2	2	France
Chicago	3	3	USA
Berlin	1	1	Germany
Barcelona			
Salzburg	7		

*This type of join is called **LEFT OUTER JOIN** or, simply, **LEFT JOIN**.*

LEFT OUTER JOIN



LEFT OUTER JOIN

```
SELECT *
FROM City
LEFT JOIN Country on Country.id = City.country_id;
```

Records **matching** the on condition

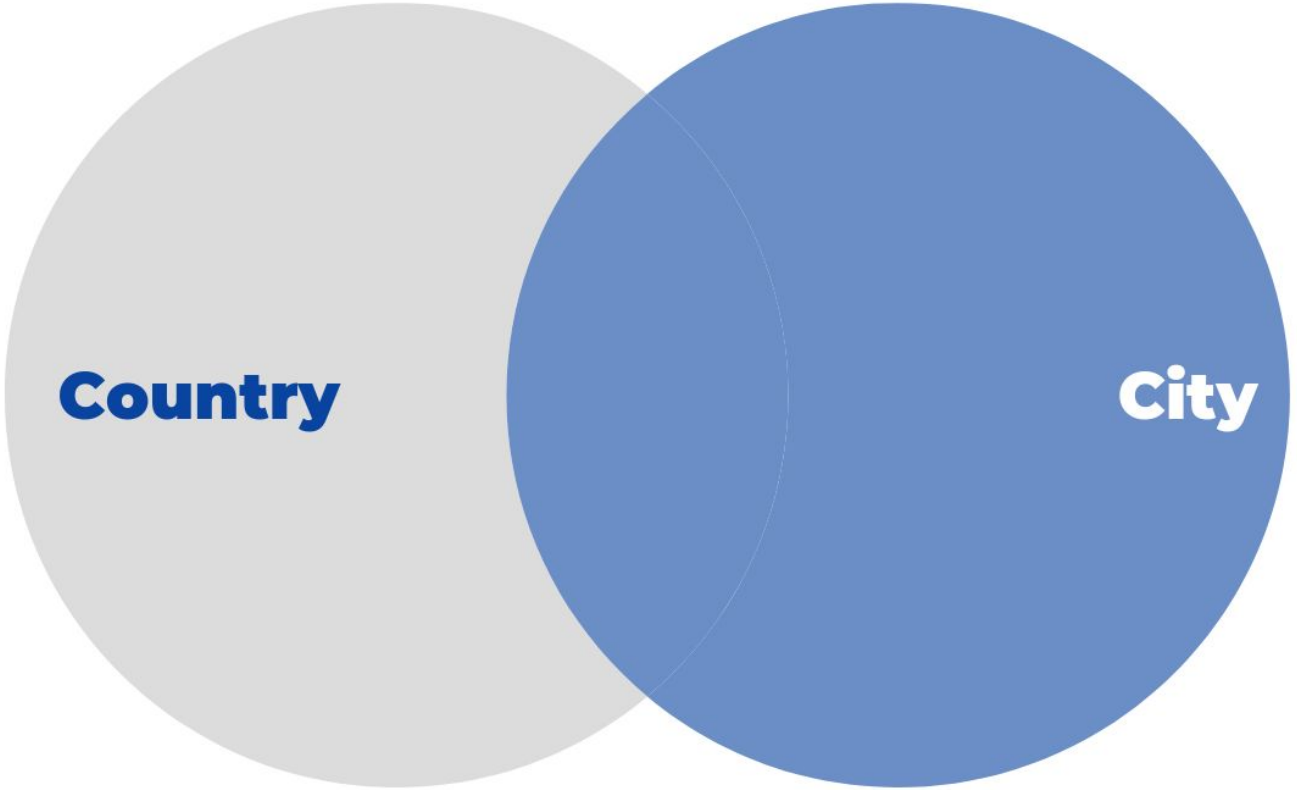
+

Records from the **main table without a match**.

Result

name	country_id	id	name
Marseille	2	2	France
Chicago	3	3	USA
Berlin	1	1	Germany
Barcelona			
Salzburg	7		

RIGHT OUTER JOIN



RIGHT OUTER JOIN

```
SELECT *
FROM City
RIGHT JOIN Country on Country.id = City.country_id;
```

Records **matching** the on condition

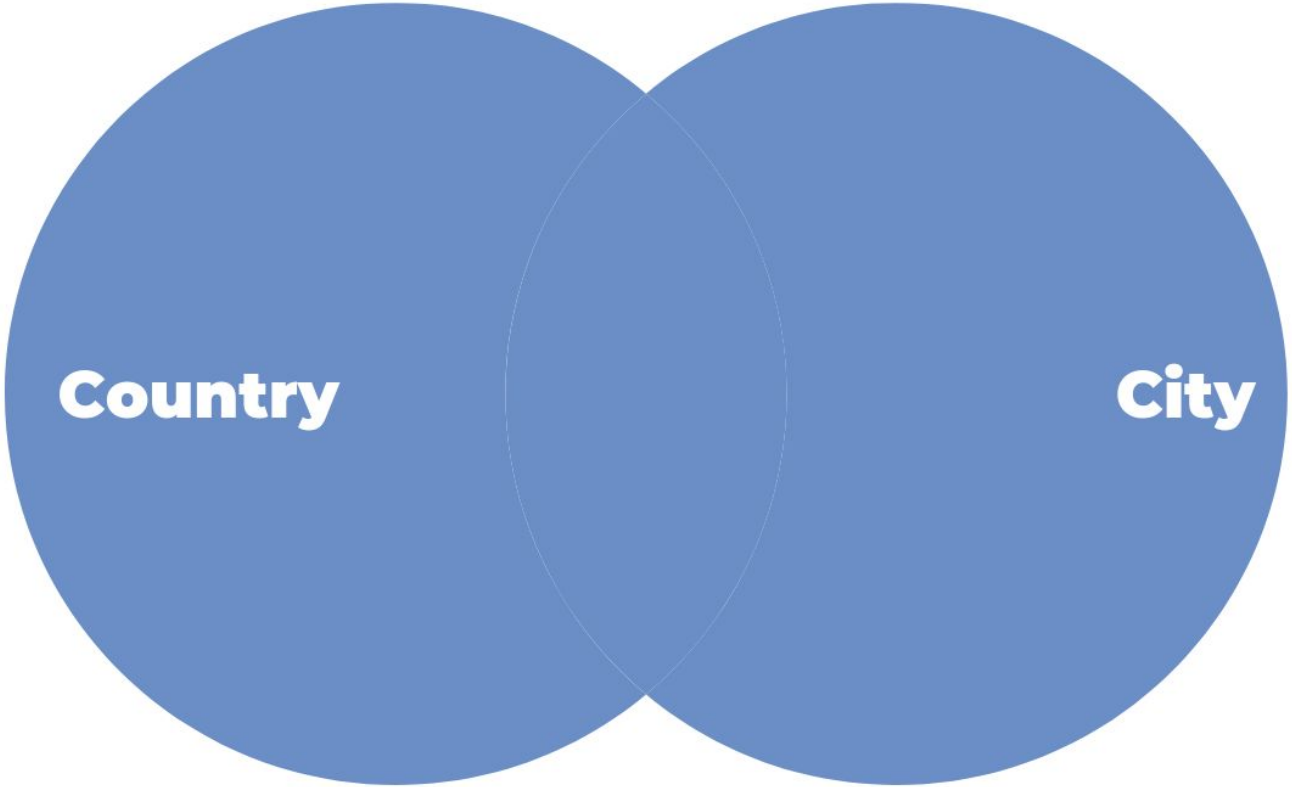
+

Records from the **secondary table without a match**.

Result

name	country_id	id	name
Marseille	2	2	France
Chicago	3	3	USA
Berlin	1	1	Germany
		4	Spain

FULL OUTER JOIN



FULL OUTER JOIN

```
SELECT *
FROM Country
FULL JOIN City on Country.city_id = city.id;
```

Records **matching** the on condition

+

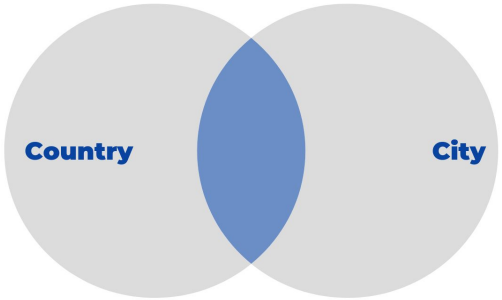
Records from **any table without a match**.

Result

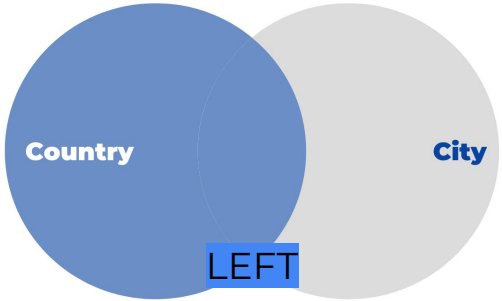
name	country_id	id	name
Marseille	2	2	France
Chicago	3	3	USA
Berlin	1	1	Germany
Barcelona			
Salzburg	7		
		4	Spain

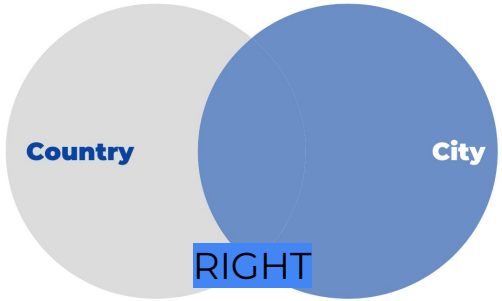
Summary

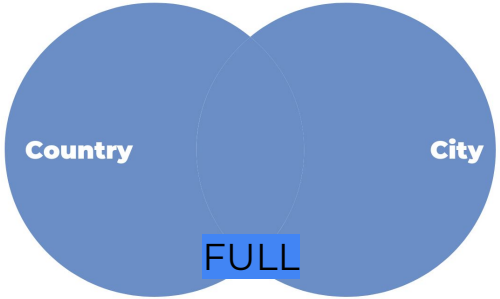
INNER



OUTER







Using JOINs

Filtering JOINs

```
SELECT Location.name, City.name FROM Location
INNER JOIN City on City.id = Location.city_id
WHERE City.country_id = 3;
```

Location

name	city_id
Headquarters	2
Location 2	1
Location 3	2
Location 4	3
Location 5	4
Location 6	

Result

name	name
Headquarters	Chicago
Location 3	Chicago

Filtering JOINs

OK

```
SELECT Location.name, City.name FROM Location
INNER JOIN City on City.id = Location.city_id
WHERE City.country_id = 3;
```

These two statements are equivalent.

?

```
SELECT Location.name, City.name FROM Location
INNER JOIN City
on City.id = Location.city_id AND City.country_id = 3;
```

Filtering JOINs

```
SELECT {fields} FROM {table_a}
INNER JOIN {table_b} on {join_conditions}
WHERE {filter_conditions};
```

For the sake of semantics and readability the `on` keyword should define only the **joining conditions** and the `WHERE` clause should define only the **filtering conditions**.

Multiple JOINs

```
SELECT Location.name, City.name, Country.name FROM Location
LEFT JOIN City ON City.id = Location.city_id
LEFT JOIN Country ON Country.id = City.country_id;
```

Location

name	city_id
Headquarters	2
Location 2	1
Location 3	2
Location 4	3
Location 5	4
Location 6	

Result

name	name	name
Headquarters	Chicago	USA
Location 2	Marseille	France
Location 3	Chicago	USA
Location 4	Berlin	Germany
Location 5	Barcelona	
Location 6		

Multiple JOINs

*The order of the **`JOIN`** clauses matters.*

OK

```
SELECT Location.name, City.name, Country.name FROM Location
LEFT JOIN City on City.id = Location.city_id
LEFT JOIN Country on Country.id = City.country_id;
```

```
SELECT Location.name, City.name, Country.name FROM Location
LEFT JOIN Country on Country.id = City.country_id
LEFT JOIN City on City.id = Location.city_id;
```

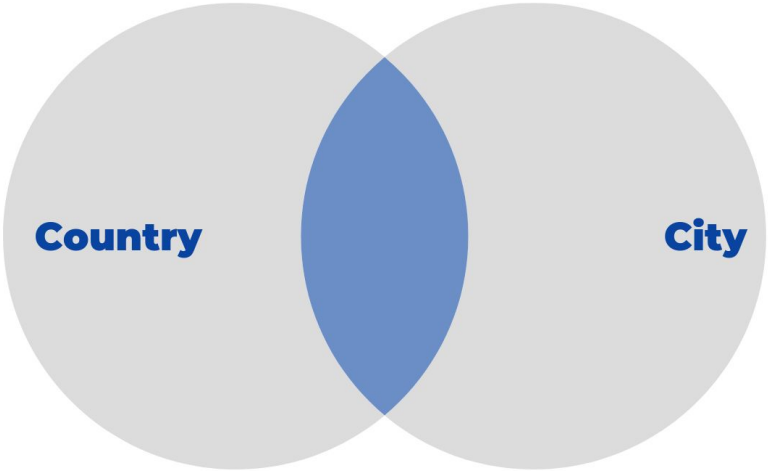
At this point **`City`** is still undefined.

How JOINs work

How JOINs Work

The basic `JOIN` operations can be visualized as Venn diagrams from set theory.

But the `JOIN` operation does not actually work like this and not all `JOIN` types can be explained this way.



Venn diagram of an INNER JOIN.

How JOINs Work

Remember the first slide on JOINs:

The **`JOIN`** clause **takes all combinations of records** between **`table_a`** and **`table_b`** and returns those that match the indicated **`conditions`**.

CROSS JOIN

```
SELECT *
FROM City
CROSS JOIN Country;
```

The results include **every possible combination** between records on both tables.

All **JOIN** operations derive from a **CROSS JOIN**. They are filters of this main cross table.

Result

	name character va	country_id integer	id integer	name character va
1	Marseille	2	1	Germany
2	Marseille	2	2	France
3	Marseille	2	3	USA
4	Marseille	2	4	Spain
5	Chicago	3	1	Germany
6	Chicago	3	2	France
7	Chicago	3	3	USA
8	Chicago	3	4	Spain
9	Berlin	1	1	Germany
10	Berlin	1	2	France
11	Berlin	1	3	USA
12	Berlin	1	4	Spain
13	Barcelona		1	Germany
14	Barcelona		2	France
15	Barcelona		3	USA
16	Barcelona		4	Spain
17	Salzburg	7	1	Germany
18	Salzburg	7	2	France
19	Salzburg	7	3	USA
20	Salzburg	7	4	Spain

CROSS JOIN

```
SELECT *
FROM City
CROSS JOIN Country;
```

=

```
SELECT *
FROM City
LEFT JOIN Country on True;
```

=

```
SELECT *
FROM City
INNER JOIN Country on True;
```

Result

	name character va	country_id integer	id integer	name character va
1	Marseille	2	1	Germany
2	Marseille	2	2	France
3	Marseille	2	3	USA
4	Marseille	2	4	Spain
5	Chicago	3	1	Germany
6	Chicago	3	2	France
7	Chicago	3	3	USA
8	Chicago	3	4	Spain
9	Berlin	1	1	Germany
10	Berlin	1	2	France
11	Berlin	1	3	USA
12	Berlin	1	4	Spain
13	Barcelona		1	Germany
14	Barcelona		2	France
15	Barcelona		3	USA
16	Barcelona		4	Spain
17	Salzburg	7	1	Germany
18	Salzburg	7	2	France
19	Salzburg	7	3	USA
20	Salzburg	7	4	Spain

INNER JOIN Example

```
SELECT Location.name, City.name FROM Location
INNER JOIN City (1)
on City.id = Location.city_id (2)
WHERE City.country_id = 3; (3)
```

1

CROSS JOIN

name	name
Location 2	Marseille
Location 3	Marseille
Location 4	Marseille
Location 5	Marseille
...	...

+

2

ON CONDITION

name	name
Headquarters	Chicago
Location 2	Marseille
Location 3	Chicago
Location 4	Berlin
Location 5	Barcelona

3

WHERE COND.

name	name
Headquarters	Chicago
Location 3	Chicago

INNER JOIN Example

```
SELECT *
FROM City
JOIN Country on Country.id = City.country_id;
```

CROSS JOIN

	name character va	country_id integer	id integer	name character va
1	Marseille	2	1	Germany
2	Marseille	2	2	France
3	Marseille	2	3	USA
4	Marseille	2	4	Spain
5	Chicago	3	1	Germany
6	Chicago	3	2	France
7	Chicago	3	3	USA
8	Chicago	3	4	Spain
9	Berlin	1	1	Germany
10	Berlin	1	2	France
11	Berlin	1	3	USA
12	Berlin	1	4	Spain
13	Barcelona		1	Germany
14	Barcelona		2	France
15	Barcelona		3	USA
16	Barcelona		4	Spain
17	Salzburg	7	1	Germany
18	Salzburg	7	2	France
19	Salzburg	7	3	USA
20	Salzburg	7	4	Spain

Result

name	country_id	id	name
Marseille	2	2	France
Chicago	3	3	USA
Berlin	1	1	Germany

Edge Case Examples

The **on** condition can be anything that evaluates to **True** or **False**.

```
SELECT *
FROM City
JOIN Country on False;
```

or

```
SELECT *
FROM City
JOIN Country on
Country.id = City.country_id OR city_id + id = 5;
```

CROSS JOIN

	name character va	country_id integer	id integer	name character va
1	Marseille	2	1	Germany
2	Marseille	2	2	France
3	Marseille	2	3	USA
4	Marseille	2	4	Spain
5	Chicago	3	1	Germany
6	Chicago	3	2	France
7	Chicago	3	3	USA
8	Chicago	3	4	Spain
9	Berlin	1	1	Germany
10	Berlin	1	2	France
11	Berlin	1	3	USA
12	Berlin	1	4	Spain
13	Barcelona		1	Germany
14	Barcelona		2	France
15	Barcelona		3	USA
16	Barcelona		4	Spain
17	Salzburg	7	1	Germany
18	Salzburg	7	2	France
19	Salzburg	7	3	USA
20	Salzburg	7	4	Spain

We learned ...

- How to cross reference data from multiple tables using JOIN clauses.
- That there are different types of JOIN clauses we can use.
- How to filter and use multiple JOINs in one statement.
- That the joining operation actually filters a complete cross reference of records to produce the result.

Contact Details
DCI Digital Career Institute gGmbH

Digital Career Institute