SQL JOINS ARE EASY

1. Introduction

Many beginners to SQL find joins complicated and counter-intuitive. When I first learned SQL at my IT education, the word "join" was almost immediately followed by "difficult". The easy alternative was to use subselects. But at some point, I suddenly perceived them for what they are, and they became very simple.

Because subselects are very slow and unsupported by several databases, they should be avoided whenever possible (which is almost always). With this article I intend to show that SQL joins are indeed simple and that subselects can be easily avoided. I think that the only reason people find joins difficult, is because of the myth that surrounds them.

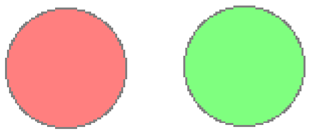
The examples in this article are of course very simple, and more advanced use of joins can be a lot more complicated, but because this article is not meant to be an exhaustive resource on joins, it doesn't deal with the more complex issues.

---

2. Sets and SQL

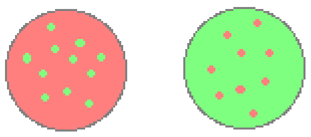
When thinking about the contents of your database tables, you have to think of them in terms of mathematicalsets; it's all about intersections, unions, etc. When thinking in terms of sets, you can use Venn diagrams to illustrate abstract concepts. Although it must be said that strictly speaking, Venn diagrams don't apply to SQL, because the items in the collections (the tables) are not identical, but because they refer to eachother, Venn diagrams can still be used to clarify the concept.

Take these two collections:



Two collections

Consider the red collection customers and the green collection orders. Some of these orders belong to some of these customers, i.e. the two collections have some overlap. This can be visualized thusly:



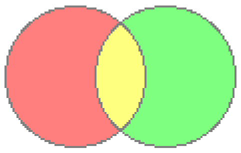
Two collections with overlap

Let's say you want to select all the customers who have placed orders. The subselect way of doing this is:

```
SELECT * FROM customers
WHERE id IS IN (SELECT DISTINCT customer_id FROM orders);
```

This is a very procedural way of looking at the problem: going by each record step by step and compare it to a list; it's very much like going through a for-loop. But because SQL is a functional language, we are going to look at our problem in a functional way.

To begin to understand joins, we must first have a different perspective on what we really want. In set terminology: we want the *intersection* of customers and orders. In graphical terms, this is expressed like:



Two collections intersected

We're interested in the yellow part. This yellow part, or the *inner* part (hint), are all the customers with their orders. All we have to do now is express this in SQL:

```sql
SELECT *
FROM customers AS customer_record
INNER JOIN orders AS order_record
ON order_record.customer_id = customer_record.id;
```

Remember that your result will now also include all the rows from the orders table, because after all, the intersection was all the customers *with their orders*. If you only want the customer data, you have to do the following, obviously:

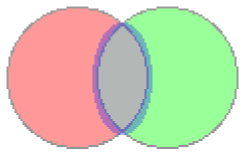
```sql
SELECT customer_record.*
FROM customers AS customer_record
INNER JOIN orders AS order_record
ON order_record.customer_id = customer_record.id;
```

See what the *inner* join does? Couldn't be simpler.

---

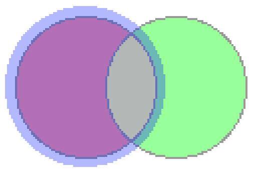
3. The big picture

Now we'll have to extend this to the big picture. The most common types of joins can be made very clear by some very simple illustrations. We already discussed the *inner join*, which is this area of two collections (the area marked blue):



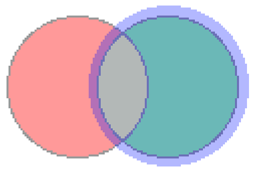
Inner join

The *left outer join* results in all the rows of the left collection, and where present, the rows of the right collection (in other words, it doesn't leave out rows from the left collection). That is this area of the two collections:



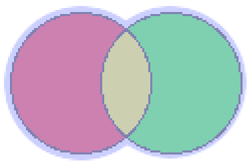
Left outer join

The *right outer join* does exactly the same as the left outer join, but in reverse, so that would be this area of the collection:



Right outer join

The *full outer join*, which is basically a left outer join and right outer join added together, simply returns everything, like this:



Full outer join

Piece of pie, right?

Source : http://www.halfgaar.net/sql-joins-are-easy