SQL Joins
Class Notes

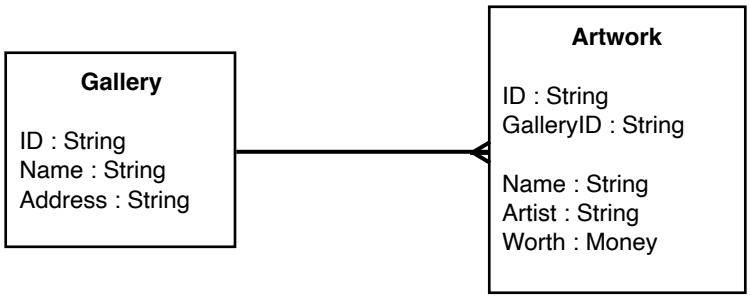
We have designed relational databases where tables have a connection with others via foreign keys but we have not queries such a set up in SQL yet.

We do so by using the normal SELECT statement and adding a join.

It's worth pointing out at this stage that SQL SELECT statements can get very long and complex. So far, we have seen a SELECT statement be able to choose which fields to display, filter records using boolean statements (AND, OR and NOT) and order the results in ascending or descending order. Now we will be joining tables in the same statement and there's more to come. Complex SELECT statements will often need careful and thoughtful assembly to achieve the result you want.

Joining Tables

Let's look again at the gallery and artwork tables from previous sessions…



Okay, so say we want to list all the artwork and the gallery they belong to. We can do this with a select statement but we have four problems.

Problem 1 - Repeated Fields

Both the gallery and the artwork tables have an "ID" and a "Name", so how do we tell SQL which ID and which Name we are talking about?

Well, we could change the table so that the gallery has a GalleryName and the artwork an ArtworkName, but this is a waste of typing and not good practice. There is no need to make the name fields more verbose just to repeat information we already have (that is, which table they're in).

Instead, we use dot notation, which you are probably already familiar with from various object oriented languages like C# and C++. With dot notation, we can start our query like so.

```
SELECT Gallery.Name, Artwork.Name FROM…
```

Note the query is not complete (hence the dot dot dot). There a few things to cover so we're going to construct it as we go.

Problem 2 - From Where?

The select statement ends with “FROM” but which table are we going to say we’re getting the information from? We’re getting fields from *both* tables, after all.

Either will do and - for now - it doesn’t matter which. (It will later, though.)

```
SELECT Gallery.Name, Artwork.Name FROM Artwork…
```

Problem 3 - How Do We Link The Folder Table?

Linking a second table into the query is new syntax. It looks like this…

```
SELECT Gallery.Name, Artwork.Name FROM Artwork INNER JOIN Gallery…
```

Now we are telling SQL to join the first table in the query (Artwork) to the Gallery table. (Don’t worry about what an “inner join” is - we will come back to that.)

Note that if we had used Gallery as the first table, we would need to use Artwork as the second, like so…

```
SELECT Gallery.Name, Artwork.Name FROM Gallery INNER JOIN Artwork…
```

The effect is the same in this example.

Problem 4 - Exactly How Do They Link?

We’ve asked SQL to join the tables for the purposes of this query but we also need to tell it exactly how they are linked. That is, we need to tell SQL what the foreign key in the Artwork table is and what that key referring to in the Gallery table.

Looking at the diagram above, we can see that the foreign key is called “GalleryID” and that it corresponds to the ID field in the Gallery table. This is the las piece of information we need to tell SQL.

```
SELECT Gallery.Name, Artwork.Name FROM Artwork INNER JOIN Gallery ON
Gallery.ID = Artwork.GalleryID;
```

Once again we use dot notation to specify the table for each field we’re talking about. Also note that the equals sign is a single equals - not a double equals as is common in many programming languages.

This now complete query will provide the gallery names and artwork names for all artwork that is actually housed in a gallery. However, we need to revisit that inner join and look at the options there.

Join Types

Inner join

We just used an inner join. This means if a piece of artwork exists in the table but is not linked to a gallery, it will not be displayed. Also, if a gallery has no artwork, it will also not be displayed. The query only displays records if the tables are linked on that record.

As previously mentioned, if you are using an inner join, it doesn’t matter what order you have the tables in.

Left And Right Joins

Note that, with our query, one table is to the left of the “JOIN” keyword and one is to the right.

```
SELECT Gallery.Name, Artwork.Name FROM Artwork INNER JOIN Gallery ON
Gallery.ID = Artwork.GalleryID;
```

For an inner join, this order doesn't matter. For a left and right join, it does. A left join looks like so…

```
SELECT Gallery.Name, Artwork.Name FROM Artwork LEFT JOIN Gallery ON Gallery.ID
= Artwork.GalleryID;
```

And a right join looks like so…

```
SELECT Gallery.Name, Artwork.Name FROM Artwork RIGHT JOIN Gallery ON
Gallery.ID = Artwork.GalleryID;
```

Okay, so how do they work?

With an inner join, records would only be displayed *if* they are linked. If a piece of art is in a gallery, it will be listed. If a gallery has artworks, it will be listed. Any artwork not in a gallery and any gallery without any art will not be included.

With a left join, everything in the left most table will be listed whether it's linked or not. In a right join, everything in the right most table will be listed whether it's linked or not.

Note that the order that you put the tables is not the same as the order in which you display the fields. You choose what order to display the fields in the first part of the query.

SELECT **Gallery.Name, Artwork.Name**…

Examples

The best way to see the difference is with examples. Here is the SQL to create a simple gallery and artwork database…

```
create table Galleries(id integer, name varchar(100));

insert into Galleries values (1, "The Hogarth Gallery"), (2, "Rumpole's Museum
and Gallery"), (3, "Unfinished, unnamed gallery");

create table Artwork(id integer, galleryID integer, name varchar(100));

insert into Artwork values (1, 1, "Still life with balloon animals"), (2, ,
"Engine buried in mud"), (3, null, "Ambivalence in Blue”);
```

Note that “Ambivalence in blue” is not at any gallery and that the “Unfinished, unnamed gallery” has no artwork.

Inner Join Example

```
select Galleries.name, Artwork.name from Artwork inner join Galleries on
Artwork.galleryID=galleries.id;

The Hogarth Gallery                Still life with balloon animals
Rumpole's Museum and Gallery       Engine buried in mud
```

Left Join Examples

Take note of which table is left and which is right in the query.

```
select Galleries.name, Artwork.name from Artwork left join Galleries on
Artwork.galleryID=galleries.id;

The Hogarth Gallery           Still life with balloon animals
Rumpole's Museum and Gallery  Engine buried in mud
NULL                          Ambivalence in Blue

select Galleries.name, Artwork.name from Galleries left join Artwork on
Artwork.galleryID=Galleries.id;

The Hogarth Gallery           Still life with balloon animals
Rumpole's Museum and Gallery  Engine buried in mud
Unfinished, unnamed gallery   NULL
```

Right Join Examples

Again, take note of which table is left and which is right in the query.

```
select Galleries.name, Artwork.name from Artwork right join Galleries on
Artwork.galleryID=galleries.id;

The Hogarth Gallery           Still life with balloon animals
Rumpole's Museum and Gallery  Engine buried in mud
Unfinished, unnamed gallery   NULL

select Galleries.name, Artwork.name from Galleries right join Artwork on
Artwork.galleryID=galleries.id;

The Hogarth Gallery           Still life with balloon animals
Rumpole's Museum and Gallery  Engine buried in mud
NULL                          Ambivalence in Blue
```

A Final Note

Joins can be a bit fiddly not just to understand but to predict what you'll get. Experience and practice will eventually make them second nature but, in the meantime, you should experiment and tweak until you get the result you want.