Let's do some coding practice to continue learning about INNER JOINs in this video. Joining our normalized tables back together to get the results set information we need.

Let's make the v_art our current database so we can query it.

```
USE v_art;
```

Let's get a list of all the titles of each artwork.

```
SELECT title
FROM artwork;
```

What if we wanted to find out the artist who painted each one and add their name to the results here alongside their artwork.

If we look at everything in artwork:

```
SELECT * FROM artwork;
```

There is no first and last name fields. That's because when we designed and normalized this database we had to separate out those tables. So the only thing we have in this artwork table that shows who created the artwork is an artist_id as a foreign key. We can add this column to our result set by adding it to the SELECT statement.

```
SELECT artist_id, title
FROM artwork;
```

But that isn't really what we wanted. We wanted their names. Also, it's good to keep in mind that users won't know what the artist_id number is for paintings. Primary and foreign keys usually are not seen by the end users so in order to make a database application user friendly we want to allow users to search and see data that they understand. The result set here is not really useful to our users because they don't know what artists are represented by those artist_id numbers.

What we need is information from 2 different tables. The first and last names of the artists from the artist table and the title of the artwork from the artwork table. Up until now we have only queried our database from one table at a time, but in the real world of database querying, you would use joins all the time to get result sets that combine the useful information from multiple tables.

Let's get a result set that gets columns from two different tables (the artist and artwork). We do this with a new keyword that goes with the FROM clause. The keyword join is a sub-clause of FROM. We can take the same query as before, and add the first name and last name to the select clause but because artwork doesn't contain those 2 columns we will add a JOIN sub-clause after the first from and table name with the second table that does contain those columns. When we type the keyword JOIN it is implying an INNER JOIN. You can type the keyword INNER in front of JOIN and it means the same thing. If you leave off the keyword INNER and just have JOIN it is still an inner join. We'll talk about outer joins in another video.

If we try and run it at this point we unintentionally get what's called a cross join which returns every combination of a row from one table and a row from the other table.

```
SELECT fname, lname, title
FROM artwork JOIN artist;
```

That is not what we want. We only want the results of where the tables are related. Or in other words where the primary key of the artist table is related to the foreign key of the artwork table. So we have to include a join condition with an ON clause. The ON clause states how the two tables are related, or which keys we are using to relate those two tables. We know that artist_id is the primary key of artist and artist_id is the foreign key of artwork. Those are the two keys or columns that relate those two tables together. So we use those with the ON clause.

```
SELECT fname, lname, title
FROM artwork JOIN artist
        ON artist_id = artist_id;
```

But if we try to run this now, it will say that artist_id in the ON clause is ambiguous. That's because we have the same exact name for our primary and foreign key names "artist_id". It wants to know which artist_id we are talking about. (Note: if the primary and foreign key names had been different, we wouldn't have gotten this error). But because ours are the same name we have to say which table those columns or key are coming from. We will use what's called a table qualifier. All that means is we proceed the column name with the name of the table it comes from and a period or dot.

```
SELECT fname, lname, title
FROM artwork JOIN artist
        ON artwork.artist_id = artist.artist_id;
```

Now we get the results we expected. Vincent van Gogh's first and last name is repeated for each artwork he created because he had 3 related paintings in the artwork table.

He has the artist_id of 1 in the artist table.

```
SELECT * FROM artist;
```

He has the artist_id of 1 as the foreign key 3 times in the artwork table for Irises, Starry Night and Sunflowers.

```
SELECT * FROM artwork;
```

```
SELECT fname, lname, title
FROM artwork JOIN artist
        ON artwork.artist_id = artist.artist_id;
```

And now his first and last name show up for those same 3 paintings in our join.

There is a USING clause that can be used in place of the ON clause and I'll show it here, but it wasn't in your preparation materials for a reason.

```
SELECT fname, lname, title
FROM artwork JOIN artist
        USING(artist_id);
```

You will get the same results but your primary and foreign key would have to have the same exact name (which is true of our tables). I want to make you aware of it if you see it in code but remember the USING clause isn't universal for all RDBMS and there are some limitations with USING. So, this course won't be teaching this clause.

So let's switch back to an ON clause. And let's add a filter. We want to only see Leonardo diVinci's artwork.

```
SELECT fname, lname, title
FROM artist JOIN artwork
  ON artist.artist_id = artwork.artist_id
WHERE lname = 'da Vinci';
```

Even if only want to see the title and we take off fname and lname. We'd still need a join because our WHERE clause is using a column from a different column. So it's not just the SELECT statement that you need to look at to see how many tables you will use, but other clauses as well. Because now our SELECT only has a column from one table but we still need the join because our WHERE uses a different table than the SELECT does.

```
SELECT title
FROM artist JOIN artwork
  ON artist.artist_id = artwork.artist_id
WHERE lname = 'da Vinci';
```

Let's look at the bike database.

```
USE bike;
```

Let's look at the relationship between the product table and the category table.

```
  SELECT * FROM category;
```

There are 7 different categories, with category_id used as the primary key. This is a type of look up table.

```
SELECT * FROM product;
```

In the product table category_id is used as a foreign key to show the category each bike belongs to.

Let's get a result set with the product name, the category name and the list price.

```
SELECT product_name, category_name, list_price
FROM product
        JOIN category
        ON product.category_id = category.category_id;
```

If we wanted to add the brand name to this result set, this would involve 3 tables. We'd have to add the brand table as well. Again this is a lookup type of table with 9 different brands with brand_id as the primary key.

```
SELECT * FROM brand;
```

The product table also has the brand_id as a foreign key. This is how the two tables are related.

```
SELECT product_name, category_name, brand_name, list_price
FROM product
  JOIN category
  ON product.category_id = category.category_id
  JOIN brand
  ON product.brand_id = brand.brand_id;
```

Let's add a filter. I only want to see the bikes in the "Children Bicycles" category.

```
SELECT product_name, category_name, brand_name, list_price
FROM product
  JOIN category
  ON product.category_id = category.category_id
  JOIN brand
  ON product.brand_id = brand.brand_id
WHERE category_name = "Children Bicycles";
```

You can use aliases for your tables to cut down on how much typing you need to do with Joins.

```
SELECT product_name, category_name, brand_name, list_price
FROM product p
  JOIN category c
  ON p.category_id = c.category_id
   JOIN brand b
   ON p.brand_id = b.brand_id
WHERE category_name = "Children Bicycles";
```

But remember if you use a column in the WHERE or SELECT that could be in two different tables you have to use the the alias qualifiers there as well. For example if you wanted to see the category_id and filter it by brand_id.

```
SELECT product_name, c.category_id, brand_name, list_price
FROM product p
  JOIN category c
  ON p.category_id = c.category_id
   JOIN brand b
   ON p.brand_id = b.brand_id
WHERE b.brand_id = 2;
```

If you leave them off you will get ambiguous errors.

OK try one on your own. We want to get a list of staff (first and last name) from the Rowlett Bikes Store. Pause the video and see if you can get this one on your own. So the first and last name of the staff from the Rowlett Bikes store and you can't use store_id #3 but the actual name of the store.

```
SELECT first_name, last_name
FROM staff JOIN store
  ON staff.store_id = store.store_id
WHERE store_name = "Rowlett Bikes";
```

Here's what I came up with. The two columns that relate the two tables of staff and store is the store_id column. You could have also use the LIKE keyword in the WHERE if you didn't know exactly how the store name was spelled.

If I did use the store_id in the WHERE clause I'd have to qualify that table as well. For example if we use a condition in the WHERE clause that says store_id = 3; since you have two tables joined and they both have a store_id column in them it will give you and error saying that store_id in the WHERE clause is amibiguous.

```
SELECT first_name, last_name
FROM staff JOIN store
   ON staff.store_id = store.store_id
WHERE store_id = 3;
```

That's because it's not sure which table the store_id is referring to. In our case it doesn't really matter so I can put a table qualifier of staff or store in front of the store_id in the WHERE clause.

```
SELECT first_name, last_name
FROM staff JOIN store
   ON staff.store_id = store.store_id
WHERE staff.store_id = 3;
```

If I place store_id in the SELECT clause as well. I have to qualify it there as well.

```
SELECT first_name, last_name, staff.store_id
FROM staff JOIN store
   ON staff.store_id = store.store_id
WHERE staff.store_id = 3;
```

Let's look at joining more than 2 tables. Let's join 3 tables together. We want to show not only the bike product name, but the brand name and category name of each bike. If we look at the ERD of the product, brand and category. The product_name is in the product table, the category_name is in the category table and the brand_name is in the brand table. So let's code this.

```
SELECT product_name, category_name, brand_name, list_price
FROM product JOIN category
   ON product.category_id = category.category_id
   JOIN brand
   ON product.brand_id = brand.brand_id;
```

You will just add another JOIN and ON for each new table you want to add.

```
SELECT product_name, category_name, brand_name, list_price
FROM product JOIN category
   ON product.category_id = category.category_id
   JOIN brand
   ON product.brand_id = brand.brand_id
WHERE category_name = "Children Bicycles";
```

Let’s switch to the v_art database and try a multiple table example. We want to show all the artwork titles that use the keyword ‘water’.

```
SELECT title
FROM artwork
   JOIN artwork_keyword
   ON artwork.artwork_id = artwork_keyword.artwork_id
   JOIN keyword
   ON artwork_keyword.keyword_id = keyword.keyword_id
WHERE keyword = "water";
```

In the SELECT we want a column from the artwork table and in the WHERE we refer to a column from the keyword table. That’s two tables so why do we need to JOIN three together. Because if you look at the ERD the artwork table doesn’t connect or relate directly with keyword, there is a linking table between them. So there wouldn’t be any column that would be the same for artwork and keyword you have to join everything together that connects those tables. So the linking table of artwork_keyword has to be added as well.

Again, you can use table aliases for your tables so you can use them elsewhere when referring to that table

```
SELECT title
FROM artwork a
   JOIN artwork_keyword ak
   ON a.artwork_id = ak.artwork_id
   JOIN keyword k
   ON ak.keyword_id = k.keyword_id
WHERE keyword = "water";
```

OK let’s try an INNER JOIN on multiple tables inner Join before we go onto OUTER Joins. This is the employee ERD that you will work with. Say we need to find the name of an employee his salary and what department he works for. This is 3 different tables. The departments, employees, and salaries.

Pause the video and see if you could figure it out if you needed columns from department, employees and salary, how would you use joins so you could get all that you need.

OK, I’ll go through the process of how I’d do it. So I’ve got to join all these tables. Notice that to join between employees and departments, there is no primary and foreign key that could link them in an ON statement. So we will have to also join in the dept_emp table as well. This join will join 4 tables. So I’d start at one end or the other, it doesn’t matter (department or salaries). But say I start at department, I’d have FROM department JOIN dept_emp then the ON statement that joins those two tables. Then JOIN employees and the ON that joins those two and then another JOIN to salaries with the ON that joins those two. So now you could show any columns from those tables in the SELECT or WHERE or ORDER BY and it would work.

```
SELECT first_name, last_name, dept_name, salary
FROM departments d JOIN dept_emp de
        ON d.dept_no = de.dept_no
   JOIN employees e
   ON de.emp_no = e.emp_no
   JOIN salaries s
   ON e.emp_no = s.emp_no;
```

But what if I was filtering by from_date or wanting to show the from_date in the SELECT statement. I'd have to have a table qualifier because from_date is the same name of the column in two of the tables that I have joined. So be aware that any ambiguous error you get means you need to put a table qualifier before the column name.

```
SELECT first_name, last_name, dept_name, salary, s.from_date
FROM departments d JOIN dept_emp de
        ON d.dept_no = de.dept_no
   JOIN employees e
   ON de.emp_no = e.emp_no
   JOIN salaries s
   ON e.emp_no = s.emp_no
WHERE s.from_date > "2000-12-31";
```

Let's use a database where the primary key name is different from the foreign key name. Just because we've been using databases where the primary and foreign keys are exactly the same name, it doesn't mean you won't run into databases where those keys are different names. They still use the same values to relate the tables together but the actual name of the key headings are different. Look at the ERD and see that the primary key of magazine and subscribers is simply id which does follow some naming conventions of different systems. Then when it is used as a foreign key as in this linking table example of subscriptions, they will name it with the table it came from with id. This database also uses camelCase meaning column and table names with two words will have the second and more words have initial caps.

```
USE magazine;

SELECT * FROM magazine;

SELECT * FROM subscriptions;
```

We can see the primary keys of the magazines are now used as foreign keys.

If we want to get all the magazine names and subscriptionStartDates. We can use a join and now the ON clause doesn't need table qualifiers because the primary and foreign keys are different.

```
        SELECT magazineName, subscriptionStartDate
        FROM magazine
                JOIN subscription
                ON id = magazine_id;
```

However if we want to see names of all those who subscribe to the magazine we would need to add a table. I will also add aliases.

```
SELECT magazineName, subscriberLastName, subscriberFirstName
        FROM magazine m
                JOIN subscription sn
                ON id = magazine_id
                JOIN subscriber sr
                ON id = subscriber_id;
```

We will get an error saying that id is ambiguous because we have two tables that use id as the primary key so it does need to know which table we are referring to.

Now this time I did have to add table qualifiers in front of id in the on clause

```
SELECT magazineName, subscriberLastName, subscriberFirstName
        FROM magazine m
                JOIN subscription sn
                ON m.id = magazine_id
                JOIN subscriber sr
                ON sr.id = subscriber_id;
```

One more practice. Say we want to look up all of the products Mr. Baldwin has ordered. We want his last name, the product's names and the order date. You can pause the video and see if you can come up with just the product name and order dates of the customer with the last name of Baldwin.

Here's what I came up with.

```
SELECT product_name, order_date
FROM customer c
        JOIN cust_order co
  ON c.customer_id = co.customer_id
        JOIN cust_order_item coi
  ON co.cust_order_id = coi.cust_order_id
  JOIN product p
  ON coi.product_id = p.product_id
WHERE last_name = "Baldwin";
```

Looking at the ERD we see that we need his last name from the customer table to filter our results in the WHERE clause, We need the order date from the cust_order table and the product name from the product table to show in our results as part of the SELECT statement. So we join the customer, cust_order, cust_order_item, and product tables.

This will give us the results we need.

So these have all been inner joins getting just the data that has a relationship between the joined tables. Meaning there was a primary key that matched a related foreign key of the other table. INNER joins are the most common type of JOIN. Next video we will look at OUTER JOINs.