Ironhack Student Portal

preview.my.ironhack.com/lms/courses/course-

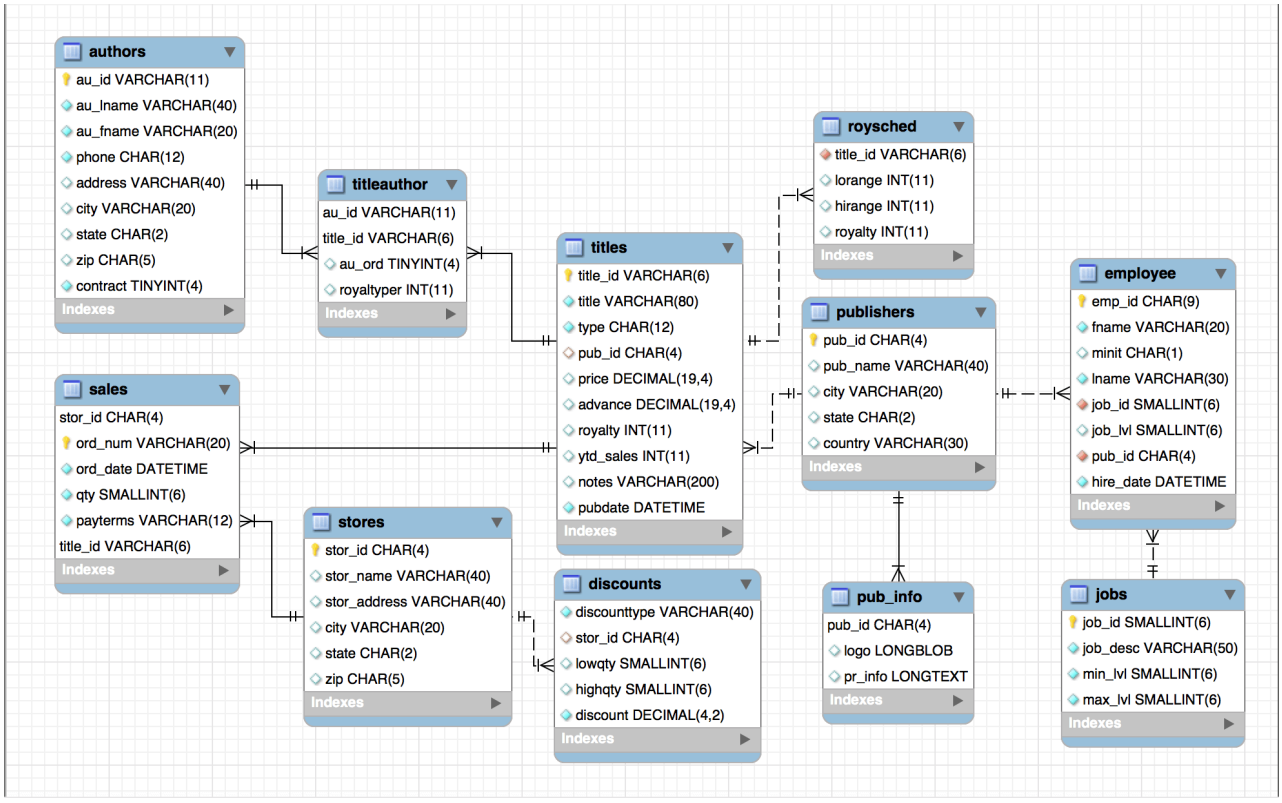
Lesson Goals

- Learn about the concept of joins.
- Learn about the different types of joins and what subsets of the data each returns.
- Learn how to write queries that join tables.

Introduction

In the last lesson, we learned about relational database concepts and structure, including the different types of relationships that can exist in a database (one-to-one, one-to-many, and many-to-many). In this lesson, we will learn how to leverage that structure and those relationships to write more robust queries with SQL. We will do this by using *joins*, which will allow us to specify relationships between entities from within our queries.

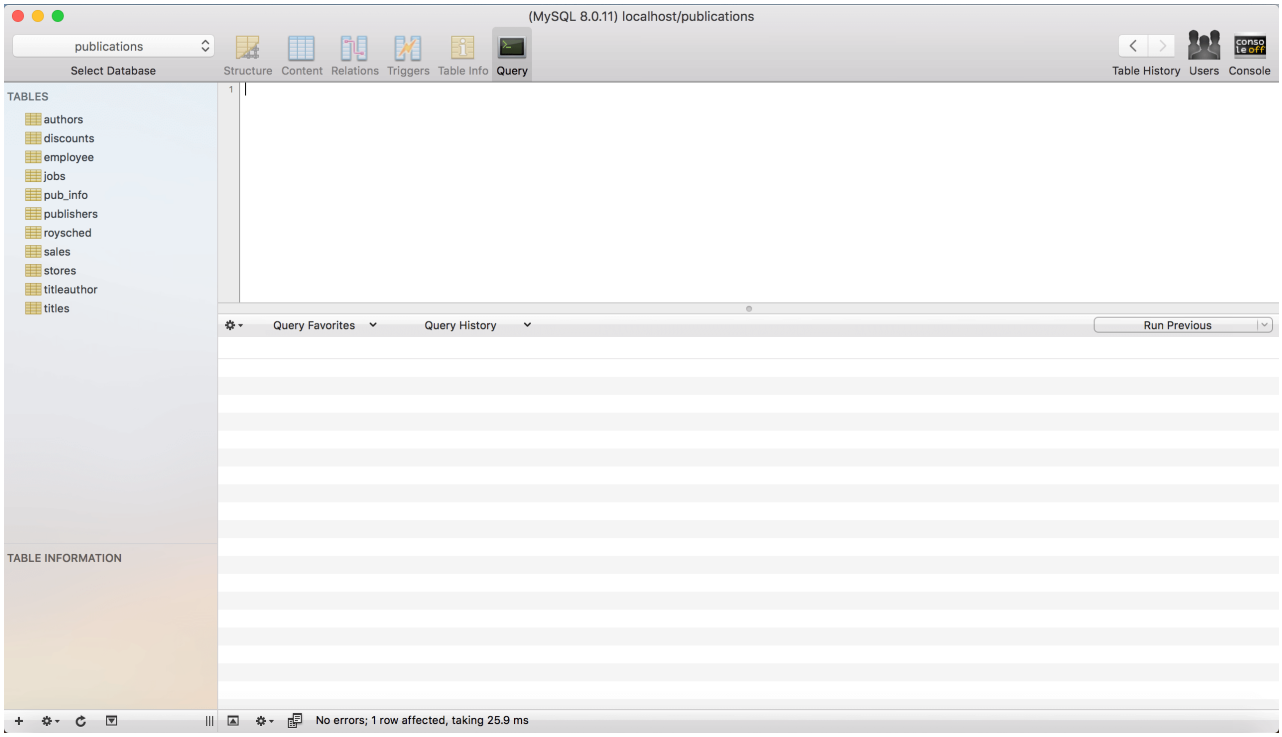
For this lesson, we will revisit the publications data set that we examined the ERD diagram in the previous lesson. Because we know how that database is designed and what entity tables relate in what ways to other entity tables, we will be able to easily create queries that join those entities together. As a reminder, below is how the database is structured and the relationships that exist.



To follow along, you must obtain the publications data set and import it into MySQL. Below are step-by-step instructions for how to do that.

1. Download publications.sql.zip.
2. Unzip the downloaded file and extract the database dump file on your machine.
3. In Sequel Pro, create a new database called "publications" by running the following command: `CREATE DATABASE publications;` .
4. From the *Select Database* dropdown menu on the upper left, select the publications database.
5. From the top menu, select *File* and then the *Import* menu option.
6. Navigate to the publications.sql dump file and import it.

You should now see the database tables listed under *TABLES* in the left sidebar listed.



Join Basics

Before we jump into writing queries with joins on our data set, let's go over some basic concepts as well as the different types of joins you will need to know.

When you join two tables, you join them based on a field that they have in common. For example, if you wanted to join the `publishers` table and the `employees` table in our database, you would join on the `pub_id` fields in each table since they both reference the same thing (the publisher's unique ID).

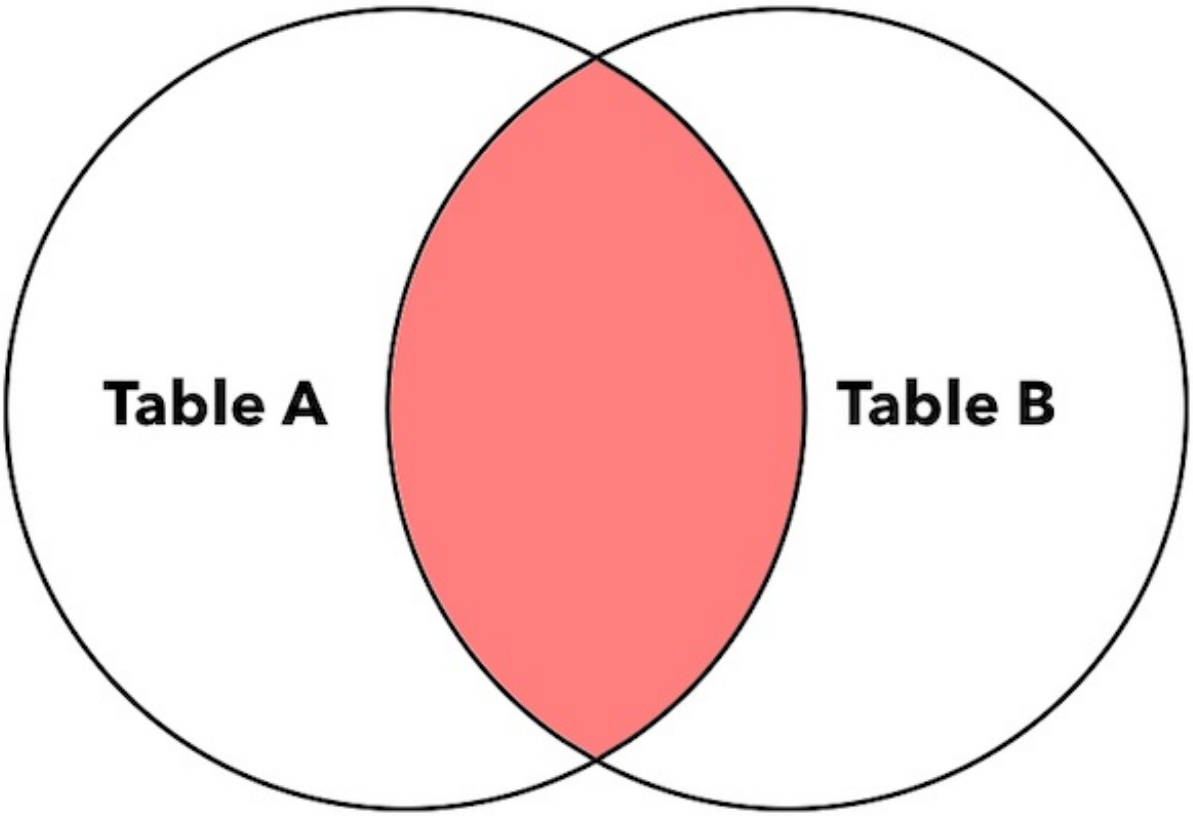
In a join, the table you declare first (we will call it Table A) will be the considered the *left* table and the second table you declare (Table B) will be the *right* table. You will want to keep track of which tables are left and right, as this will impact the results of your query for certain types of joins.

Below are explanations of the major types of joins you are likely to encounter as well as visual depictions of how their results differ.

INNER JOIN

The most basic type of join is the *inner join*. When you inner join two tables (Table A and Table B), the records returned are only those where there was a match in both tables.

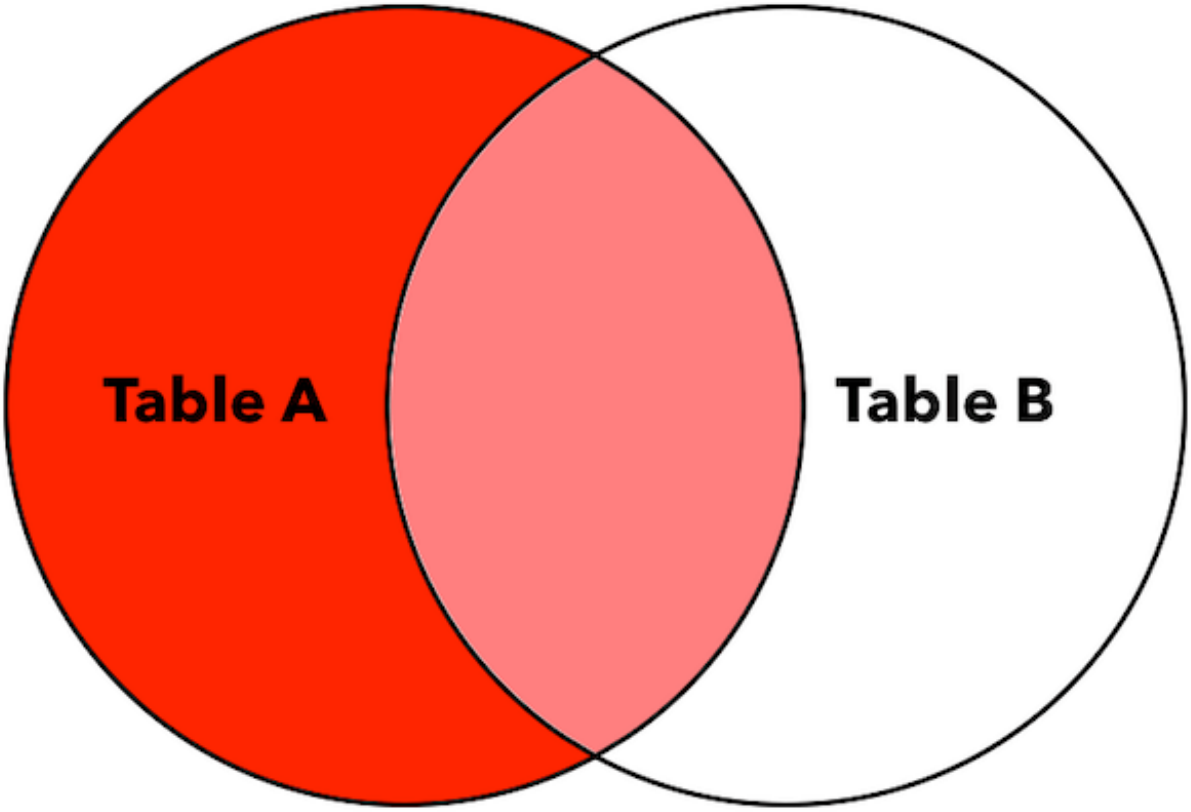
```
SELECT *
FROM Table_A
INNER JOIN Table_B
ON Table_A.Key = Table_B.Key
```



LEFT JOIN

With a *left join*, the query will return all the records in the left table (the table declared first) and the matched records from the right table (the table declared second).

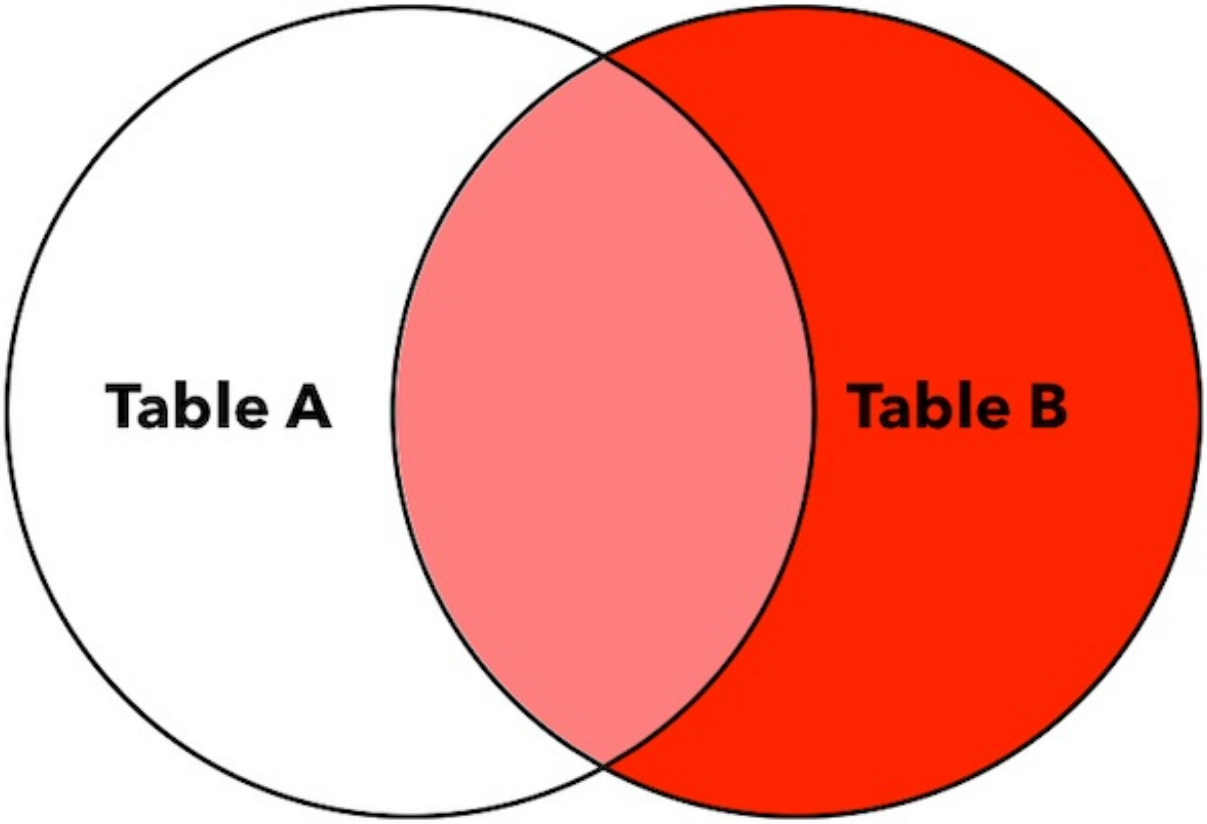
```
SELECT *
FROM Table_A
LEFT JOIN Table_B
ON Table_A.Key = Table_B.Key
```



RIGHT JOIN

A *right join* does the exact opposite of a left join. It returns all the records from the right table and just the matching records from the left table.

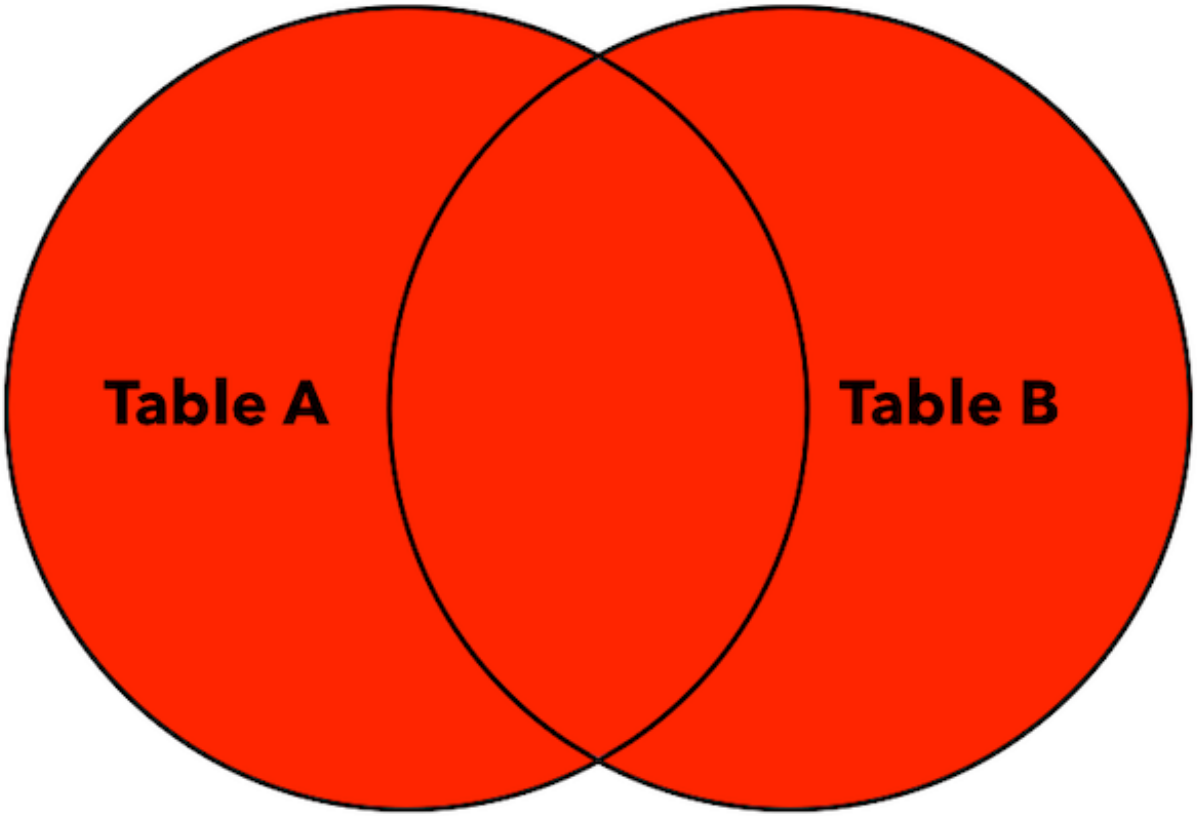
```
SELECT *
FROM Table_A
RIGHT JOIN Table_B
ON Table_A.Key = Table_B.Key
```



OUTER JOIN

An *outer join* will return every record that there is a match for in both the left and right tables.

```
SELECT *
FROM Table_A
OUTER JOIN Table_B
ON Table_A.Key = Table_B.Key
```



Creating Queries with Joins

Now that we know about the different types of joins and have our data loaded in to MySQL, we are ready to start writing queries with joins in them.

From looking at our ERD, we know that publishers and titles have a one-to-many relationship. Let's join them and get a count of the number of titles each publisher has published.

```
SELECT pubs.pub_name, COUNT(titles.title_id) AS Titles
FROM publications.publishers pubs
INNER JOIN publications.titles titles
ON pubs.pub_id = titles.pub_id
GROUP BY pubs.pub_name;
```

pub_name	Titles
New Moon Books	5
Binnet & Hardley	7
Algodata Infosystems	6

Because we used an `INNER JOIN` for this query, it returns only results for publishers whose `pub_id` is in both tables. What if we wanted the results to return records for all the publishers, regardless of whether they had published any titles? This is exactly what a

left join can help us with. All we need to do is replace our `INNER JOIN` with a `LEFT JOIN` and it will return all the records in the `publishers` table and count zero records for the ones that do not have any titles published.

```
SELECT pubs.pub_name, COUNT(titles.title_id) AS Titles
FROM publications.publishers pubs
LEFT JOIN publications.titles titles
ON pubs.pub_id = titles.pub_id
GROUP BY pubs.pub_name;
```

pub_name	Titles
New Moon Books	5
Binnet & Hardley	7
Algodata Infosystems	6
Five Lakes Publishing	0
Ramona Publishers	0
GGG&G	0
Scootney Books	0
Lucerne Publishing	0

What if we wanted to analyze how many units were sold for each title? We could declare our `sales` table first, our `titles` table second, and use a `RIGHT JOIN` to ensure that our query returns a record for every title (even the ones that did not have any sales).

```
SELECT titles.title, titles.type, titles.price, SUM(sales.qty) AS units_sold
FROM publications.sales sales
RIGHT JOIN publications.titles titles
ON sales.title_id = titles.title_id
GROUP BY titles.title, titles.type, titles.price;
```

title	type	price	units_sold
The Busy Executive's Database Guide	business	19.99	15
Cooking with Computers: Surreptitious Balance Sheets	business	11.95	25
You Can Combat Computer Stress!	business	2.99	35
Straight Talk About Computers	business	19.99	15

title	type	price	units_sold
Silicon Valley Gastronomic Treats	mod_cook	19.99	10
The Gourmet Microwave	mod_cook	2.99	40
The Psychology of Computer Cooking	UNDECIDED	NULL	NULL
But Is It User Friendly?	popular_comp	22.95	30
Secrets of Silicon Valley	popular_comp	20	50
Net Etiquette	popular_comp	NULL	NULL
Computer Phobic AND Non-Phobic Individuals: Behavior Variations	psychology	21.59	20
Is Anger the Enemy?	psychology	10.95	108
Life Without Fear	psychology	7	25
Prolonged Data Deprivation: Four Case Studies	psychology	19.99	15
Emotional Security: A New Algorithm	psychology	7.99	25
Onions, Leeks, and Garlic: Cooking Secrets of the Mediterranean	trad_cook	20.95	40
Fifty Years in Buckingham Palace Kitchens	trad_cook	11.95	20
Sushi, Anyone?	trad_cook	14.99	20

We can see from the results that there is one title that has sold very well (*Is Anger the Enemy?*) and two titles that have not had sales (which show NULL in the `units_sold` field).

The command for an outer join is typically `FULL OUTER JOIN`. However, MySQL does not support full outer joins, so we must improvise and perform them using a combination of the `LEFT JOIN`, `RIGHT JOIN`, and `UNION` commands. The left and right joins will collectively return all the records from each of the joined tables. The `UNION` command combines the results of two queries, and it should be placed directly between them. For example, if we wanted to see what employees were assigned to which jobs, ensuring that the query returns both employees not assigned to a job and jobs not assigned to any employee, we would write our query as follows.

```
SELECT *
FROM publications.employee emp
RIGHT JOIN publications.jobs job
ON emp.job_id = job.job_id
UNION
SELECT *
FROM publications.employee emp
LEFT JOIN publications.jobs job
ON emp.job_id = job.job_id;
```

Summary

In this lesson, we learned how joins help bring data about separate entities together. We also learned about the different types of joins, the subsets of the data we can expect queries that use them to return, and how to incorporate them into our queries. By combining joins with the SQL commands you learned in the pre-work for the program, you should now be able to construct fairly complex queries that bring information from disparate tables together.