|  |  |
| --- | --- |
| -  Sindy Saintclair  Tuesday, January 18, 2022  Lesson 2 - Joins | |
| **Learning Objectives and Questions** | **Notes and Answers** |
| **Join** | A frequent need is to combine data from multiple tables in a single result. Imagine a customer table and a separate order table. If you wanted to get all orders for a particular customer, you would need to join these two tables. In this lesson, we learn how to do just that.  A JOIN is an operand that allows you to select a row that contains columns from multiple tables. In the sakila database that you’ve been using, there are many tables, each with bits of information you might like to query.  By the end of the lesson, you will be able to:   * join multiple tables to run a single query   The lesson will culminate in a Hands-On in which you will join multiple tables to run specific queries. |
| **Inner Join** | When using INNER JOIN, it will select data that have matching values within 2 different tables. Below is the syntax for using INNER JOIN:  SELECT [columnName]  FROM [table1]  INNER JOIN [table2] ON table1.columnName = table2.columnName;  Look at a real-life example with your sample database. If you look at the sakila.film table and the sakila.film\_category table, you’ll see that both tables contain a column for film\_id.  sakila.film table data:  Sakila film table. Table of movies listed by their ID, title, description, release year, language, original language, rental duration, rental rate, length, and replacement costs.  Figure 2-1: sakila.film.Table  sakila.film\_category table data:  Sakila film category table. Table with the column headings film ID, category ID, and last update.  Figure 2-2: sakila.film\_category Table  Now, add a line to see the same data but only where the film\_id is 1:  SELECT title, release\_year, category\_id  FROM sakila.film  INNER JOIN sakila.film\_category  ON sakila.film.film\_id = sakila.film\_category.film\_id  WHERE film.film\_id = 1  Output in MySQL Workbench.  Inner Join Where. Table with the headings title, release year, and category ID. There is only one entry listed.  Figure 2-4: Inner Join Where  Now, you only get 1 row.  *Example*  Let’s look at another example of an INNER JOIN:  Take a look at the actor and film\_actor  tables. As you can see, there is an actor\_id column in each. Let’s join each of these tables using that column.  SELECT first\_name, last\_name, film\_id  FROM sakila.actor  INNER JOIN sakila.film\_actor  ON sakila.actor.actor\_id = sakila.film\_actor.actor\_id  After running the above query, now you can see the first and last name of the actors who have been in a film, along with their film ids. |
| **\*\*\*Outer Joins** | Having an inner join implies there is also an outer join, which is entirely correct. Furthermore, there is also a left and right variation. The difference between an inner and outer join has to do with when a row is selected to be included. Say there was a row in albums that was not linked to any rows in tracks. An inner join only includes the specified columns from a row if it can be paired up with a row in the joined table. So a row in albums that doesn’t match any row in tracks would not show up at all in the results. Outer joins change this. An outer join takes every row from either the left or right (hence the LEFT OUTER JOIN or RIGHT OUTER JOIN) and if a matching row exists in the other table, then those columns are included as well.  *LEFT OUTER JOIN*  A LEFT OUTER JOIN will return all records from the left table and all matched records from the right table. The result will be NULL on the right side if there is no match. The left table is the table name located right after the FROM keyword, and the right table is the table listed right after the INNER JOIN keywords.  Look at an example of the difference between an INNER JOIN and a LEFT OUTER JOIN:  SELECT \* FROM sakila.film  INNER JOIN sakila.film\_actor  ON sakila.film.film\_id = sakila.film\_actor.film\_id  Output in MySQL Workbench:  Join. Table with the headings film ID, title, description, release year, language, original language, rental duration, rental rate, length, and replacement cost, rating, and special.  Figure 2-5: JOIN  SELECT \* FROM sakila.film  LEFT OUTER JOIN sakila.film\_actor  ON sakila.film.film\_id = sakila.film\_actor.film\_id  Output in MySQL Workbench:  Left outer Join. Table with the headings film ID, title, description, release year, language, original language, rental duration, rental rate, length, and replacement cost, rating, and special.  Figure 2-6: Left Outer Join  *RIGHT OUTER JOIN*  A RIGHT OUTER JOIN accomplishes the same thing as the LEFT OUTER JOIN, but it will be returning all of the records of the right table instead of the left.  Below is what you will get when you try to run a RIGHT OUTER JOIN on the same above queries:  SELECT \* FROM sakila.film  RIGHT OUTER JOIN sakila.film\_actor  ON sakila.film.film\_id = sakila.film\_actor.film\_id  Right Outer Join. Table with the headings film ID, title, description, release year, language, original language, rental duration, rental rate, length, and replacement cost, rating, and special.  Figure 2-7: Right Outer Join |
| **USING** | Consider a shortcut, USING is a great way to cut down your query. You can use USING instead of ON. Consider the following query:  SELECT first\_name, last\_name, film\_id  FROM sakila.actor  INNER JOIN sakila.film\_actor  ON sakila.actor.actor\_id = sakila.film\_actor.actor\_id  We worked with this query earlier in this lesson. What this query is doing is joining the two tables using the actor\_id column which both tables have. The ON statement is defining each table name then the column name we want (sakila.actor.actor\_id or sakila.film\_actor.actor\_id). Since the column is the same name for both tables, you can use USING. Consider below:  SELECT first\_name, last\_name, film\_id  FROM sakila.actor  INNER JOIN sakila.film\_actor  USING(actor\_id)  Both queries will return the same amount of data, as they are the same. One query is just using shorthand. The USING keyword assumes that both tables defined (actor and film\_actor) have the column name of actor\_id. Therefore, you don’t need to define the table name when looking for the same column in two tables.  WORKSHOP JOINS |
| **ALIASES** | Aliases are used to give tables or columns different names than what is currently in the database. This is accomplished by using the AS keyword. Let’s explore a scenario that could potentially require an alias.  You are working with a database, but someone else compiled the database and the column and table names. However, what you are doing with the data doesn’t quite line up with the names of the columns for what you are trying to do. You could change the name of the columns using AS so it lines up better with your project. Consider below:  SELECT title AS filmTitle FROM sakila.film  The output in MySQL Workbench:  A S. Table with the heading film title. There is a listing of several film titles below it.  Figure 2-8: AS  In this case, we are changing the title  column in the film  table to filmTitle which makes the column name a bit clearer about what it is. If you explore the tables, there are multiple columns named title which can get a bit confusing. So, you can query that column but name it something different so you can easily identify it after the query. |
| **Multiple Joins** | Now that you have learned how to join two tables, you will learn how to join from multiple. You will use the same concept you learned from doing a singular join, except the main difference will be finding multiple columns that connect the tables together.  Look at the example below:  SELECT \* from sakila.actor  JOIN sakila.film\_actor USING (actor\_id)  JOIN sakila.film\_category USING (film\_id)  In the query above, you are joining the actor, film\_actor, and film\_category tables together. First, you join the actor table and the film\_actor table by using their common column, actor\_id. After these two tables are joined, you are then able to join the film\_category table due to the fact that the film\_actor table had a common column ( film\_id) with the film\_category table. Without joining the film\_actortable to the actor table first, you would have not been able to join the actor table to the film\_category table because they have no common columns.  \*\*\*Notice how we are only using the “JOIN” keyword? That is shorthand for “INNER JOIN”, and it is common to use only the word “JOIN” when doing an inner join.  The output of the above query will look like below:  Multiple. Table with the headings film id, actor ID, first name, last name, last update, last update, category ID, and last update. There are several entries below the headings with the appropriate information.  Figure 2-9: Multiple  Once you can join across more than two tables, there’ not any join you can’t write. Experiment by looking at the tables in the database and extracting interesting data across at least two tables.  WORKSHOP: ADVANCED JOINS |