# The City Lit Institute

##### Department of Computing

## Keeley Street, Holborn, London WC2B 4BA

**MySQL**

**(with Apache server)**

Performing JOINS

**LECTURER** : **ALEXANDER ADU-SARKODIE**

MSc. Telematics (IT & Telecom), MSc. Eng., Dip. Russ. Lang., Teach. Cert, AMIAEng (UK), MBCS (UK), MIfL(UK

**Biopic :** [**http://goldhawk-college.com/istudio/**](http://goldhawk-college.com/istudio/)

**Blog :** [**http://www.blogger.com/profile/14800490193632788559**](http://www.blogger.com/profile/14800490193632788559)

**Email :** [**aadusarkodie1@gmail.com**](mailto:aadusarkodie1@gmail.com)

**Performing a FULL Join**

The beauty of joins resides in its ability to fetch data from two or more tables using a single query. Considering that I defined only two tables, the first example that I want to show you is how to use the tables in question for performing the simplest kind of join, which turns out to be one called the **FULL** join or **Cartesian** product.

Simply put, a FULL join returns a new table containing all the possible combinations between the rows of the first and the second table. The best way to understand this process is with functional code, so be sure to check the following query, which performs a FULL join with the aforementioned tables:

**SELECT customers.first\_name, customers.last\_name, books.title FROM customers, books**

As you can see, the syntax of a FULL join is pretty intuitive, as you only need to specify which columns should be retrieved, followed by the names of the joined tables separated by a comma (notice the use of the dot notation, in order to avoid possible clashing with fields that have been named the same).

In addition, it’s possible to run the previous join using explicitly the JOIN keyword, something depicted in the following example:

**SELECT customers.first\_name, customers.last\_name, books.title FROM customers JOIN books**

Effectively, both expressions are equivalent, even when the latter uses the JOIN clause instead of a comma to specify which tables should be joined. What’s more, regardless of the syntax that you pick up for executing the query, it should produce the table shown in the below snapshot:

FULL joins aren’t very useful in practice, as they only return a bunch of rows that in most cases don’t meet any specified condition. You can easily address this issue by using the **INNER** join.

**Working with INNER Joins**

Again, an example is the most useful ally to make things clear; so take a peek at the one below, which uses an INNER join to retrieve the customers that purchased a book along with the book’s title:

**SELECT customers.first\_name, customers.last\_name, books.title FROM customers INNER JOIN books ON customers.book\_id=books.id ORDER BY customers.first\_name**

As one might expect, the above query will return only the rows in both tables that match the given condition.

As with a FULL join, it’s possible to use an alternative syntax for an INNER join. Since in this particular case, we’re looking for customers that did purchase a book, the previous query could be rewritten by using a WHERE clause, as shown below:

**SELECT customers.first\_name, customers.last\_name, books.title FROM customers, books WHERE customers.book\_id=books.id ORDER BY customers.first\_name**

While the “INNER” keyword hasn’t been explicitly coded in the above statement, if you try it using your own MySQL client program, you should get exactly the same output that you saw before. Considering that both expressions are equivalent, simply feel free to use the one that best suits your preferences.