# Week 5 SQL Tutorial: Joins and Multi-table Queries

Often it is necessary to pull together attributes from multiple tables to answer a question you have about the data in a RDBMS. In such cases, you need to perform a *join*. The following details why joins are necessary and how to do them.

## The Scenario

Imagine we have the following two tables about U.S. Presidents:  
(Note that this database does NOT exist in your MySQL installation. We’ll get to a follow-along example using the ‘employees’ database at the end of this tutorial.)

MYSQL> SELECT \* FROM FIRSTNAME;

+----+---------+

| ID | FNAME |

+----+---------+

| 1 | GEORGE |

| 2 | ABRAHAM |

| 3 | RICHARD |

| 4 | BARACK |

+----+---------+

4 ROWS IN SET (0.00 SEC)

MYSQL> SELECT \* FROM LASTNAME;

+----+------------+

| ID | LNAME |

+----+------------+

| 1 | WASHINGTON |

| 2 | LINCOLN |

| 3 | NIXON |

| 4 | OBAMA |

+----+------------+

4 ROWS IN SET (0.00 SEC)

Note that this would *NOT* be a sound way to store these data. It *does* make for a good illustration of how joins work, however, which is why we are using it.

Imagine our goal is to reunite presidential first names together with their corresponding last names.

## What NOT to Do

Bringing data from two or more tables together is unfortunately NOT as simple as specifying a comma separated list of the tables. Doing that WILL generate a result, but that result is almost never very useful. Take a look at the following:

MYSQL> SELECT \* FROM FIRSTNAME, LASTNAME;

+----+---------+----+------------+

| ID | FNAME | ID | LNAME |

+----+---------+----+------------+

| 1 | GEORGE | 1 | WASHINGTON |

| 2 | ABRAHAM | 1 | WASHINGTON |

| 3 | RICHARD | 1 | WASHINGTON |

| 4 | BARACK | 1 | WASHINGTON |

| 1 | GEORGE | 2 | LINCOLN |

| 2 | ABRAHAM | 2 | LINCOLN |

| 3 | RICHARD | 2 | LINCOLN |

| 4 | BARACK | 2 | LINCOLN |

| 1 | GEORGE | 3 | NIXON |

| 2 | ABRAHAM | 3 | NIXON |

| 3 | RICHARD | 3 | NIXON |

| 4 | BARACK | 3 | NIXON |

| 1 | GEORGE | 4 | OBAMA |

| 2 | ABRAHAM | 4 | OBAMA |

| 3 | RICHARD | 4 | OBAMA |

| 4 | BARACK | 4 | OBAMA |

+----+---------+----+------------+

16 ROWS IN SET (0.00 SEC)

This query just takes each record in the firstname table and combines it in turn with each record in the lastname table – not very useful if you want accurate data, given that most resultant records are bogus.

Go back to the list and take a look at the records that actually are valid:

+----+---------+----+------------+

| ID | FNAME | ID | LNAME |

+----+---------+----+------------+

| 1 | GEORGE | 1 | WASHINGTON |

| 2 | ABRAHAM | 1 | WASHINGTON |

| 3 | RICHARD | 1 | WASHINGTON |

| 4 | BARACK | 1 | WASHINGTON |

| 1 | GEORGE | 2 | LINCOLN |

| 2 | ABRAHAM | 2 | LINCOLN |

| 3 | RICHARD | 2 | LINCOLN |

| 4 | BARACK | 2 | LINCOLN |

| 1 | GEORGE | 3 | NIXON |

| 2 | ABRAHAM | 3 | NIXON |

| 3 | RICHARD | 3 | NIXON |

| 4 | BARACK | 3 | NIXON |

| 1 | GEORGE | 4 | OBAMA |

| 2 | ABRAHAM | 4 | OBAMA |

| 3 | RICHARD | 4 | OBAMA |

| 4 | BARACK | 4 | OBAMA |

+----+---------+----+------------+

These two tables share an attribute in common, namely ID. Where the IDs match, the records match, and the data is accurate. By adding a “WHERE” condition that specifies those values have to be equal, we can get the non-bogus info we want and eliminate the junk:

MYSQL> SELECT \* FROM FIRSTNAME, LASTNAME WHERE FIRSTNAME.ID = LASTNAME.ID;

+----+---------+----+------------+

| ID | FNAME | ID | LNAME |

+----+---------+----+------------+

| 1 | GEORGE | 1 | WASHINGTON |

| 2 | ABRAHAM | 2 | LINCOLN |

| 3 | RICHARD | 3 | NIXON |

| 4 | BARACK | 4 | OBAMA |

+----+---------+----+------------+

4 ROWS IN SET (0.00 SEC)

Note that the shared attribute has exactly the same name in both tables: ID. Because of this, we need to put the table names in front of the ID attribute and add the dot operator like this:

table\_name.attribute\_name,

because if we don’t, the specific attribute we are referring to will be ambiguous, and the SQL interpreter can’t handle ambiguity:

MYSQL> SELECT \* FROM FIRSTNAME, LASTNAME WHERE ID=ID;

ERROR 1052 (23000): COLUMN 'ID' IN WHERE CLAUSE IS AMBIGUOUS

If the shared ID attribute had different names in each of the two tables (which is not uncommon, for example, ID in one table and name\_ID in the other), you could skip that syntax if you wanted.

Now let’s go from this specific example to the general rules to follow for successful joins.

## Rules for Joining Tables

The process for joining any two tables is very much the same as we just detailed above. The rules include the following:

* The tables to be joined *must* share an attribute, which will be the PK in one table and a FK in the other, related table.
* The query must specify that the value of this shared attribute be *equal* in both tables
* General form:

SELECT \* from Table1, Table2 WHERE Table1.PK = Table2.FK;

### Example

SELECT \* from employee, department WHERE department.dept\_id = employee.dept\_id;

This query will give you all of the data from employee, together with all data from department, where department’s PK (dept\_id) is equal to employee’s FK (also dept\_id). The WHERE condition assures that only the department that each employee *actually works in* will be matched with that employee.

## More than Two Tables

Queries that involve more than two tables repeat the basic pattern above, requiring an additional join conditions ANDed together in the WHERE clause. You need to specify 1 fewer join condition than the number of tables you have: 3 tables, 2 joins; four tables, 3 joins, etc.

From our presidential name example, let’s put a “MIDDLENAME” table in play as well, with the same format as the other two, and let’s join all 3 tables together:

MYSQL> SELECT \* FROM FIRSTNAME, MIDDLENAME, LASTNAME WHERE FIRSTNAME.ID = MIDDLENAME.ID AND MIDDLENAME.ID = LASTNAME.ID;

+----+---------+----+-----------+----+-------+

| ID | FNAME | ID | MNAME | ID | LNAME |

+----+---------+----+-----------+----+-------+

| 3 | RICHARD | 3 | MILLHOUSE | 3 | NIXON |

| 4 | BARACK | 4 | HUSSEIN | 4 | OBAMA |

+----+---------+----+-----------+----+-------+

2 ROWS IN SET (0.00 SEC)

The syntax is the same, we just need to do the joins 1st to 2nd and then 2nd to 3rd.

You may notice something weird here though: We didn’t get any result for George Washington or for Abraham Lincoln! That’s because we are doing *inner joins* here. In an inner join, any time there is no corresponding FK to match the PK value (such as for Washington and Lincoln given they have no middle names) the results are omitted. You can fix that behavior by doing an *outer join* instead. Outer joins are beyond the scope of this lesson, mainly because they require a slightly different join syntax (mentioned briefly below).

## Adding Additional WHERE Criteria

Say for example we had forgotten President Obama’s first and/or last name (an unlikely example Id’ hope) and wanted ONLY his information and not anyone else’s. We can simply add an additional WHERE criteria to those involving the joins:

MYSQL> SELECT \* FROM FIRSTNAME, MIDDLENAME, LASTNAME WHERE FIRSTNAME.ID = MIDDLENAME.ID AND MIDDLENAME.ID = LASTNAME.ID AND LNAME = "OBAMA";

+----+--------+----+---------+----+-------+

| ID | FNAME | ID | MNAME | ID | LNAME |

+----+--------+----+---------+----+-------+

| 4 | BARACK | 4 | HUSSEIN | 4 | OBAMA |

+----+--------+----+---------+----+-------+

1 ROW IN SET (0.00 SEC)

## Additional Join Syntax Alternative

Note that the syntax we’ve discussed in this tutorial is NOT the most conventional MySQL join syntax, because it is harder to read and debug for experienced coders. It IS, however, the easiest to understand, which is why we cover it first in this class. If you are interested in the alternative join syntax, take a look at the video at <https://www.youtube.com/watch?v=tgwnXhiIhE0&t=4m39s>. Note that performance is EXACTLY the same between these two, which are evaluated and executed in exactly the same fashion behind the scenes.

## Follow-Along example from the “employees” database

Suppose we wanted to create a salary history for employee Leon Huttel that listed Leon’s salary over time. We’d need information from two tables: employee for first and last name, and salary for salary information. The two tables share an attribute in common -- emp\_no -- which acts as the primary key in employee and as a foreign key in salary. By using that key, we can join the information in the two tables together to get what we need:

SELECT first\_name, last\_name, salary, from\_date, to\_date FROM

employee, salary WHERE employee.emp\_no = salary.emp\_no AND

first\_name = "Leon" and last\_name = "Huttel";

+------------+-----------+--------+------------+------------+

| first\_name | last\_name | salary | from\_date | to\_date |

+------------+-----------+--------+------------+------------+

| Leon | Huttel | 69387 | 1992-05-24 | 1993-05-24 |

| Leon | Huttel | 72298 | 1993-05-24 | 1994-05-24 |

| Leon | Huttel | 74106 | 1994-05-24 | 1995-05-24 |

| Leon | Huttel | 75070 | 1995-05-24 | 1996-05-23 |

| Leon | Huttel | 75789 | 1996-05-23 | 1997-05-23 |

| Leon | Huttel | 75723 | 1997-05-23 | 1998-05-23 |

| Leon | Huttel | 76876 | 1998-05-23 | 1999-05-23 |

| Leon | Huttel | 78443 | 1999-05-23 | 2000-05-22 |

| Leon | Huttel | 80212 | 2000-05-22 | 2001-05-22 |

| Leon | Huttel | 83194 | 2001-05-22 | 2002-05-22 |

| Leon | Huttel | 84841 | 2002-05-22 | 0000-00-00 |

+------------+-----------+--------+------------+------------+

11 rows in set (0.00 sec)

Leon received some good raises over his career!