# Understanding JOINs in MySQL and Other Relational Databases

Craig Buckler (http://www.sitepoint.com/author/craig-buckler/)
UK Web Developer and Writer

(http://www.sitebuckler/)



Published May 19, 2011

This article was written in 2011 and remains one of our most popular posts. If you're keen to learn more about MySQL, you may find this recent article on administering MySQL (http://www.sitepoint.com/dbninja-mysql-client/) of great interest.

"JOIN" is an SQL keyword used to query data from two or more related tables. Unfortunately, the concept is regularly explained using abstract terms or differs between database systems. It often confuses me. Developers cope with enough confusion, so this is my attempt to explain JOINs briefly and succinctly to myself and anyone who's interested.

## Related Tables

MySQL, PostgreSQL, Firebird, SQLite, SQL Server and Oracle are relational database systems. A well-designed database will provide a number of tables containing related data. A very simple example would be users (students) and course enrollments:

**'user' table:**

| id | name | course |
|---|---|---|
| 1 | Alice | 1 |
| 2 | Bob | 1 |
| 3 | Caroline | 2 |
| 4 | David | 5 |
| 5 | Emma | (NULL) |

MySQL table creation code:

```
CREATE TABLE `user` (
 `id` smallint(5) unsigned NOT NULL AUTO_INCREMENT,
 `name` varchar(30) NOT NULL,
 `course` smallint(5) unsigned DEFAULT NULL,
 PRIMARY KEY (`id`)
) ENGINE=InnoDB;
```

The course number relates to a subject being taken in a course table…

**'course' table:**

| id | name |
|---|---|
| 1 | HTML5 |
| 2 | CSS3 |
| 3 | JavaScript |
| 4 | PHP |
| 5 | MySQL |

MySQL table creation code:

```
CREATE TABLE `course` (
 `id` smallint(5) unsigned NOT NULL AUTO_INCREMENT,
 `name` varchar(50) NOT NULL,
 PRIMARY KEY (`id`)
) ENGINE=InnoDB;
```

Since we're using InnoDB tables and know that user.course and course.id are related, we can specify a foreign key relationship:

```
ALTER TABLE `user`
ADD CONSTRAINT `FK_course`
FOREIGN KEY (`course`) REFERENCES `course` (`id`)
ON UPDATE CASCADE;
```

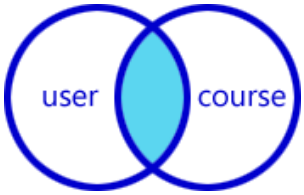
In essence, MySQL will automatically:

- re-number the associated entries in the user.course column if the course.id changes
- reject any attempt to delete a course where users are enrolled.

**important:** This is terrible database design!
This database is not efficient. It's fine for this example, but a student can only be enrolled on zero or one course. A real system would need to overcome this restriction — probably using an intermediate 'enrollment' table which mapped any number of students to any number of courses.

JOINs allow us to query this data in a number of ways.

## INNER JOIN (or just JOIN)



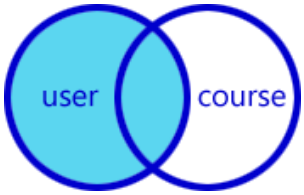

The most frequently used clause is INNER JOIN. This produces a set of records which match in both the user and course tables, i.e. all users who are enrolled on a course:

```
SELECT user.name, course.name
FROM `user`
INNER JOIN `course` on user.course = course.id;
```

**Result:**

| user.name | course.name |
|---|---|
| Alice | HTML5 |
| Bob | HTML5 |
| Carline | CSS3 |
| David | MySQL |

## LEFT JOIN



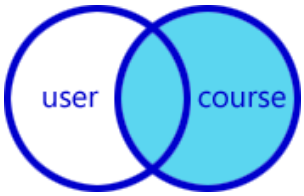

What if we require a list of all students and their courses even if they're not enrolled on one? A LEFT JOIN produces a set of records which matches every entry in the left table (user) regardless of any matching entry in the right table (course):

```
SELECT user.name, course.name
FROM `user`
LEFT JOIN `course` on user.course = course.id;
```

**Result:**

| user.name | course.name |
|---|---|
| Alice | HTML5 |
| Bob | HTML5 |
| Carline | CSS3 |
| David | MySQL |
| Emma | (NULL) |

## RIGHT JOIN




Perhaps we require a list all courses and students even if no one has been enrolled? A RIGHT JOIN produces a set of records which matches every entry in the right table (course) regardless of any matching entry in the left table (user):

```
SELECT user.name, course.name
FROM `user`
RIGHT JOIN `course` on user.course = course.id;
```

**Result:**

| user.name | course.name |
|---|---|
| Alice | HTML5 |
| Bob | HTML5 |
| Carline | CSS3 |
| (NULL) | JavaScript |
| (NULL) | PHP |
| David | MySQL |

RIGHT JOINs are rarely used since you can express the same result using a LEFT JOIN. This can be more efficient and quicker for the database to parse:

```
SELECT user.name, course.name
FROM `course`
LEFT JOIN `user` on user.course = course.id;
```

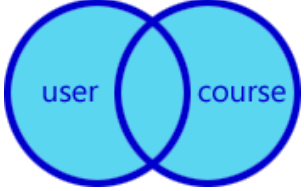
We could, for example, count the number of students enrolled on each course:

```
SELECT course.name, COUNT(user.name)
FROM `course`
LEFT JOIN `user` ON user.course = course.id
GROUP BY course.id;
```

**Result:**

| course.name | count() |
|---|---|
| HTML5 | 2 |
| CSS3 | 1 |
| JavaScript | 0 |
| PHP | 0 |
| MySQL | 1 |

## OUTER JOIN (or FULL OUTER JOIN)




Our last option is the OUTER JOIN which returns all records in both tables regardless of any match. Where no match exists, the missing side will contain NULL.

OUTER JOIN is less useful than INNER, LEFT or RIGHT and it's not implemented in MySQL. However, you can work around this restriction using the UNION of a LEFT and RIGHT JOIN, e.g.

```
SELECT user.name, course.name
FROM `user`
LEFT JOIN `course` on user.course = course.id

UNION

SELECT user.name, course.name
FROM `user`
RIGHT JOIN `course` on user.course = course.id;
```

**Result:**

| user.name | course.name |
|---|---|
| Alice | HTML5 |
| Bob | HTML5 |
| Carline | CSS3 |
| David | MySQL |
| Emma | (NULL) |
| (NULL) | JavaScript |
| (NULL) | PHP |

I hope that gives you a better understanding of JOINs and helps you write more efficient SQL queries.

If you enjoyed reading this post, you'll love Learnable (https://learnable.com/); the place to learn fresh skills and techniques from the masters. Members get instant access to all of SitePoint's ebooks and interactive online courses, like PHP & MySQL Web Development for Beginners (https://learnable.com/courses/php-mysql-web-development-for-beginners-13).

Comments on this article are closed. Have a question about MySQL? Why not ask it on our forums (http://www.sitepoint.com/forums/forumdisplay.php?88-Databases-amp-MySQL?utm_source=sitepoint&utm_medium=link&utm_campaign=forumlink)?




## 33 Reader comments

**Latz** | May 19, 2011 at 9:49 am

I never really understood JOINs and never used them therefore. This article really encourages me to over-think some of my previous coded SQL queries.

**Sparda** | May 19, 2011 at 10:18 am

Right article at the right time. Really helpful for my semester exams.
Thanks Craig.

**Craig Buckler (http://www.sitepoint.com/author/craig-buckler/)** | May 20, 2011 at 12:29 am

Best of luck – remember me when you pass and go on to become an internet billionaire!

**Mike A** | May 19, 2011 at 11:23 am

The following might help someone new the relational databases: A relation is defined as a set of tuples that have the same attributes. For years I erroneously thought that it was the capability for things like one-to-many and many-to-many relationships that put the 'relational' in relational database.

**Richard Sweeney** | May 19, 2011 at 1:18 pm

What a fantasic, clear explanation! I've just got started with MySQL and have been scratching my head a little with regards JOINs. All is clear now. The graphics really helped to visualise the query too. Many thanks.

**Kyle** | May 19, 2011 at 3:09 pm

Great overview, very helpful to see it explained so clearly. One question, Was show this just yesterday, what about ailiasing where there is no Join? Not sure which is better or why.

**BlaineSch** | May 19, 2011 at 3:59 pm