Programming (http://www.sitepoint.com/programming/)

Understanding JOINs in MySQL and Other Relational Databases

(http://www.sitepoint.com/author/craig-buckler/)

Craig Buckler (http://www.sitepoint.com/author/craig-buckler/)

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;
```

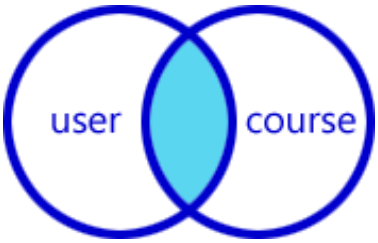
In essence, MySQL will automatically:

- 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)



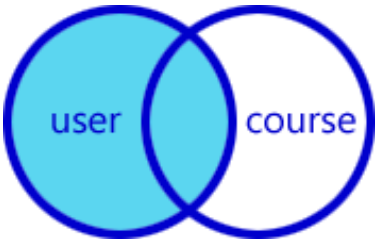
The most frequently used clause is INNER JOIN. This produces a set of records which match in both the user and course tables, i.e. all users who are enrolled on a course:

```
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):

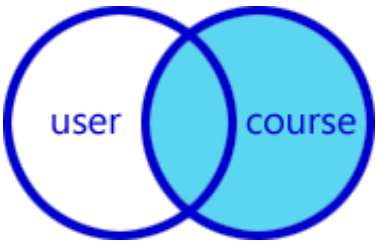
```
SELECT user.name, course.name
FROM `user`
LEFT JOIN `course` on user.course = course.id;
```

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;
```

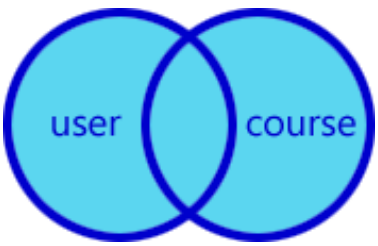
We could, for example, count the number of students enrolled on each course:

```
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)?


(http://www.sitepoint.com/author/craig-buckler/)

Craig Buckler (http://www.sitepoint.com/author/craig-buckler/)

Craig is a freelance UK web consultant who built his first page for IE2.0 in 1995. Since that time he's been advocating standards, accessibility, and best-practice HTML5 techniques. He's written almost 1,000 articles for SitePoint and you can find him @craigbuckler (http://twitter.com/craigbuckler)

(https://twitter.com/craigbuckler)
(http://www.facebook.com/craigbuckler)
(https://github.com/craigbuckler)
(http://www.linkedin.com/in/craigbuckler)
(http://plus.google.com/+CraigBuckler)

You might also like:

OSQuery: Explore your OS with SQL (http://www.sitepoint.com/osquery-explore-os-sql/)

Book: Jump Start Bootstrap (https://learnable.com/books/jump-start-bootstrap?utm_source=sitepoint&utm_medium=related-items&utm_content=js-bootstrap)

ORM in Ruby: An Introduction (http://www.sitepoint.com/orm-ruby-introduction/)

Latz · 4 years ago

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.

18 · Share ›

atman · 4 years ago

CLEAR, CONCISE & FIABLE.

Thank you!

8 · Share ›

sosydody · 4 years ago

Thank youuuuuuuuuuuu this article is very simple and clear and understandable

4 · Share ›

archana · a year ago

Thnxxx.....superb and .excelllent tutorial

6 · Share ›

Saravanan · a year ago

superb tutorial...very useful...thanks

3 · Share ›

Daniel S · 4 years ago

Great Article! I tried to reveal the difference between LEFT and RIGHT (and OUTER) joins long time ago, but your tutorial is VERY easy to understand and keeps the best explanation. Thumbs up!

Greetings from Germany..

2 · Share ›

Sean Ryder · 3 years ago

Excellent - thank you.

1 · Share ›

andrew · 4 years ago

excellent

1 · Share ›

northk · 4 years ago

Craig, best explanation I've ever seen of how joins work-- thank you! Never saw it explained so clearly in either a textbook or by one of my CS Profs. I hope you'll consider writing more like this as others have suggested.

· Share ›

Craig Buckler ➚ northk · 4 years ago

Thanks northk. JOINs are often explained using overly-complex terminology and abstract references. I suspect that's a reason many programmers prefer to code solutions in PHP rather than SQL.

There are several more MySQL and database articles coming soon. Keep an eye on SitePoint and follow me @craigbuckler.

· Share ›

biswa · 4 years ago

I would like to refer a book By Rudy Limeback 'Simply SQL'.
Read you will understand the core logic.

· Share ›

nkacharani · 4 years ago

right article at the right time. Another illustrative article on other issues like updating multiple tables would real help

tables would real help.

Share ›

Anonymous · 4 years ago

Nice article. Ven diagram's FTW. A good follow up article would be a little lesson on indexes and how they can be optimized for things like joins.

Share ›

ffrreeaakk · 4 years ago

Perfect! These images say everything.

Share ›

Dayo · 4 years ago

Very explicit explanation. I did loose out on a job bcos i did not understand this concept well enough. I think I will prolly be able to xplain this to someone else 50yrs from now, hopefully the world will still be in existence. Great job.. Hoping for more of this. Any possibilities of getting similar explanations on relationships. 1:1, 1:many, many:1 etc Thanks

Share ›

Corporate Boy · 4 years ago

I was wondering if there are any hard statistics in regards to any efficiency gains (speed/cpu/memory/network) when using JOINS in MySQL? Conservable you can get the same results with multiple queries, but JOINs make life simpler. Is the Database consuming more resources to perform the joins? Is there ever a case where it would be better to do multiple queries vs joins in order to avoid throttling CPU/Network resources of the database host?

Share ›

IT Mitică ➚ Corporate Boy · 4 years ago

At first, you need to focus on three things: tables, cursors, indexes.

You need to know that every SELECT statement opens implicit cursors, temporary tables, which it closes after the query result ends. The size of a cursor, the temporary table is a concern, but not the only one.

When you have a query involving big tables, if you have the luxury of enough database expanding tablespace, it can be faster to break the query down.

You can explicitly project the implicit cursors the database logic opens for your query, in temporary tables for which you explicitly declare indexes to help you in the next stage of your break down SELECT statement, since indexes make for a lot faster query. Thus, you take control over the internal mechanism of the JOIN.

The second thing you need to focus on: databases have to compile your JOIN query before execute it. It will also cache it, which makes for a faster second execution. It will also build an analysis on repeated runs of the said query to choose an optimal execution plan.

Those said, one look at one execution plan is not enough to decide if a JOIN query is fast enough or it needs to be divided in smaller parts. It's a statistic and indexes

game.

1 · Share ›

Corporate Boy ➚ Corporate Boy · 4 years ago

Thanks for the input. I'm considering solutions for a non-web oriented problem. We have several TBs of binary data that is used in requirement validation. We need to minimize the number of times we access this data, while also simplifying our approach to writing the validation logic that is accessing this data. Rules written in SQL could simplify our verification implementation if we could load the data in a database.

We can break this data into tables, fields, and databases. The question was asked how we would relate this data. If we didn't, then our validation might rely on multiple simple queries to prove the truth of requirement rules. We could achitect some type of relation to allow us to perform JOINs and minimize our queries to the database, but it wouldn't be completely clean. The database introduces problems into our environment and ultimately the network will be out limiting factor traffic.

Thanks again.

· Share ›

Craig Buckler ➚ Corporate Boy · 4 years ago

999 times out of 1,000, processing data in database will be more efficient -- assuming you have a well-designed schema, appropriate indexes, and well-written SELECTs.

Ultimately, though, it will depend on what you're doing and the quantities of data involved. I'd suggest you avoid premature optimization. Choose the quickest and easiest solution for the problem in hand then profile your app if performance becomes an issue.

· Share ›

John Eway · 4 years ago

This is quite a brave intro to joins. Excellent.

· Share ›

Fred · 4 years ago

Perfect ! It is the best explanation I see on JOIN !

Thanks !

· Share ›

TheMonk · 4 years ago

Good article but this leaves a question for me that I've always wondered. Does the left or right join have anything to do with the order in which you declare the tables?

For example here's the left join:
SELECT user.name, course.name
FROM `user`
LEFT JOIN `course` on user.course = course.id;

And here's the right:
SELECT user.name, course.name
FROM `user`
RIGHT JOIN `course` on user.course = course.id;

Is the first statement a "left" join simply because the user table is declared on the left in the SELECT? So if the tables were declared in reverse would my left join become a right join?

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

If my logic above is wrong, then what makes one decide if they should use a left or right join? It seems to me if there's no relation to the order tables are declared in why have two different kinds of joins if the syntax is purely arbitrary? Wouldn't one type of join simply do?

That is something which has always confused me and has limited my ability to use joins as effectively as I would like.

Craig Buckler ➦ TheMonk · 4 years ago

The essential difference between LEFT and RIGHT is how missing data is handled.

If you SELECT from tableA and LEFT JOIN tableB, you want all records in tableA regardless of any matches in tableB (missing data will be NULL).

If you SELECT from tableA and RIGHT JOIN tableB, you want all records in tableB regardless of any matches in tableA.

In effect, these produce identical results:
SELECT FROM tableA LEFT JOIN tableB
SELECT FROM tableB RIGHT JOIN tableA

So, yes, it can help to think about the table order in the SQL statement.

Where there's a choice, use LEFT JOIN. MySQL is optimized to handle those better and I suspect other DBs may too.

IT Mitică ➦ TheMonk · 4 years ago

I believe you're a bit confused.

When you say "the order in which you declare the tables" you seem to mean "the order in which you enumerate the fields (table attributes)".

The "the order in which you declare the tables" is represented in what you declare in FROM clause:

FROM `user`
LEFT JOIN `course`

where `user` is declared first and `course` is declared second. In this case, the `user` table is the main table for return select statement data. But...

The order in which you put the tables in the FROM clause it doesn't make any difference for the LEFT/RIGHT JOIN, but the LEFT JOIN/RIGHT JOIN clause in it self is the one making the difference.

If you were to write:

FROM `user`
RIGHT JOIN `course`

even though you declare `user` first, the bulk of return records' data will be from `course`.

The main point here is to remember that OUTER LEFT/RIGHT JOIN uses a main table to get its data from, putting NULL values in those fields where there is no corresponding data in the other table declared in the FROM...LEFT/RIGHT JOIN..ON clause. (BTW ON should also be in caps ;))

Share ›

Anonymous · 4 years ago

This topic helps me a lot especially with foreign keys even if I am familiar with JOINS.

Share ›

John Faulds · 4 years ago

What would also be useful would be covering statements that use WHERE instead of JOIN and how they compare.

Share ›

Dirk · 4 years ago

thanks! finally a nice and simple explanation that sticks to the basics. bookmarking this one as a cheat-sheet.

Share ›

stuw · 4 years ago

re: intermediate ‘enrollment’ table . i refer to this as a joiner table, because it joins rows in one table with another. i name a joiner table starting in alphabetical order, first table being joined, underscore letter j underscore, second table being joined.
i.e.
course_j_user

and be sure to make the user.id & course.id fields indexes so searches are found quickly.

i like the circles. they remind me of boobies. i like boobies.

now write some tuts about group by and having :)

Share ›

Craig Buckler ↗ stuw · 4 years ago

I'm sure Venn would have appreciated your comments about his diagrams!

GROUP BY and HAVING: good idea.

Share ›

BlaineSch · 4 years ago

I love the visual representation of how the data works. I've never seen it explained that way. Great job!

Share ›

Kyle · 4 years ago

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.

Share ›

Richard Sweeney · 4 years ago

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.

Share ›

Mike A · 4 years ago

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.

Share ›

(https://plus.google.com/+sitepoint)