


The DNS course
for developers



Made by me!



I've spent 2 years learning DNS while building NSLookup.io. Now, I'm teaching everything I know.

ADS VIA CARBON

- RESOURCES
- [About Me](#)
[discourse.org](#)
[stackexchange.com](#)
[Learn Markdown](#)
[Recommended Reading](#)

-  [Subscribe in a reader](#)
-  [Subscribe via email](#)

Coding Horror has been continuously published since 2004

Copyright Jeff Atwood © 2023
Logo image © 1993 Steven C. McConnell
Proudly published with  Ghost

11 Oct 2007

A Visual Explanation of SQL Joins

I thought Ligaya Turmelle's [post on SQL joins](#) was a great primer for novice developers. Since SQL joins *appear* to be set-based, the use of [Venn diagrams](#) to explain them seems, at first blush, to be a natural fit. However, like the commenters to her post, I found that the Venn diagrams didn't quite match the [SQL join syntax](#) reality in my testing.

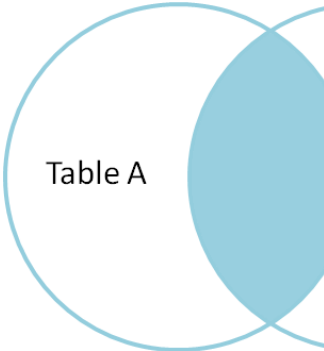
I love the concept, though, so let's see if we can make it work. Assume we have the following two tables. **Table A** is on the left, and **Table B** is on the right. We'll populate them with four records each.

id	name	id	name
--	----	--	----
1	Pirate	1	Rutabaga
2	Monkey	2	Pirate
3	Ninja	3	Darth Vader
4	Spaghetti	4	Ninja

Let's join these tables by the name field in a few different ways and see if we can get a conceptual match to those nifty Venn diagrams.

```
SELECT * FROM TableA
INNER JOIN TableB
ON TableA.name = TableB.name
```

id	name	id	name
1	Pirate	2	Pirate
3	Ninja	4	Ninja



Inner join produces only the set of records that match in both Table A and Table B.

```
SELECT * FROM TableA
FULL OUTER JOIN TableB
ON TableA.name = TableB.name
```

id	name	id	name
1	Pirate	2	Pirate
2	Monkey	null	null
3	Ninja	4	Ninja
4	Spaghetti	null	null
null	null	1	Rutabaga
null	null	3	Darth Vader



Table A

Full outer join produces the set of all records in Table A and Table B, with matching records from both sides where available. If there is no match, the missing side will contain null.

```
SELECT * FROM TableA
LEFT OUTER JOIN TableB
ON TableA.name = TableB.name
```

id	name	id	name
--	----	--	----
1	Pirate	2	Pirate
2	Monkey	null	null
3	Ninja	4	Ninja
4	Spaghetti	null	null



Table A

Left outer join produces a complete set of records from Table A, with the matching records (where available) in Table B. If there is no match, the right side will contain null.

```
SELECT * FROM TableA
LEFT OUTER JOIN TableB
ON TableA.name = TableB.name
WHERE TableB.id IS null
```

id	name	id	name
----	------	----	------

2	Monkey	null	null
---	--------	------	------

4	Spaghetti	null	null
---	-----------	------	------



Table A

To produce the set of records only in Table A, but not in Table B, we perform the same left outer join, then **exclude the records we don't want from the right side via a where clause.**

```
SELECT * FROM TableA
FULL OUTER JOIN TableB
ON TableA.name = TableB.name
WHERE TableA.id IS null
OR TableB.id IS null
```

id	name	id	name
----	------	----	------

2	Monkey	null	null
---	--------	------	------

4	Spaghetti	null	null
---	-----------	------	------

null	null	1	Rutabaga
------	------	---	----------

null	null	3	Darth Vader
------	------	---	-------------



Table A

To produce the set of records unique to Table A and Table B, we perform the same full outer join, then **exclude the**

records we don't want from both sides via a where clause.

There's also a cartesian product or **cross join**, which as far as I can tell, can't be expressed as a Venn diagram:

```
SELECT * FROM TableA  
CROSS JOIN TableB
```

This joins "everything to everything", resulting in $4 \times 4 = 16$ rows, far more than we had in the original sets. If you do the math, you can see why this is a *very* dangerous join to run against large tables.

NEXT

Mouse Ballistics

PREVIOUS

A Lesson in Control Simplicity

Written by Jeff Atwood

Indoor enthusiast. Co-founder of Stack Overflow and Discourse. Disclaimer: I have no idea what I'm talking about. Find me here: <http://twitter.com/codinghorror>

**ShawnW**

Oct '07

Funny, I just explained this to a co-worker in the same manner earlier in the week. I guess it never dawned on me that others may have never thought about joins in terms of these diagrams. Good post, Jeff!

**capdog**

Oct '07

Hey Jeff, thanks for the great blog - I thoroughly enjoy reading every single one of your posts.

Even though I often am familiar with the concepts you talk of, I find the simple manner in which you break down the issues is always a great refresher.

Keep up the great work.

**hacktick**

Oct '07

linked from join (SQL) en wikipedia to your blog entry ... your post is much better (or simplistic) than the techtalk on that wikipedia entry 😊

**Weeble**

Oct '07

The post is much simpler than the Wikipedia entry because it omits all the non-trivial cases. All of these examples assume that each entry in table A will be matched by the join predicate to at most one entry in table B, and vice-versa.

**Brad**

Oct '07

I'm not even sophisticated enough to use databases anymore (my career as a developer has devolved), but when I was, 99% of the time I used inner joins and just thought of it as the data common to both tables.

Every now and then I wanted some data that didn't fit that kind of join and I'd do a left outer join to get the data that was "left out".

**GintsP**

Oct '07

Speaking about Oracle (+) syntax and LEFT/RIGHT [OUTER] JOIN syntax there is a difference. And it is nicely described here

<http://www.oreillynet.com/pub/a/network/2002/10/01/whatsinacondition.html>

Speaking about ANSI syntax where everyone has to explicitly say to make CROSS JOIN to get Cartesian product at least in Oracle it is not true, using stupid (at least to my mind) NATURAL JOIN clause one can easily get CROSS JOIN and also JOIN on such columns he never thought it really is. I've blogged about it here

<http://gplivna.blogspot.com/2007/10/natural-joins-are-evil-motto-if-you.html>

Don't know whether it is just Oracle specific or as per ANSI SQL standard.

Speaking about INNER JOINS as a Cartesian product and then eliminating all unnecessary rows, at least the real mechanics in Oracle is absolutely different, there are following possibilities for doing them:

NESTED LOOPS (OUTER)

HASH JOIN (OUTER)

MERGE JOIN (OUTER)

and for Cartesian product it uses MERGE JOIN CARTESIAN, and that's when usually the real trouble (filling temp) starts 😊

The real choice among Nested loops, hash join or merge join at least depends on data, statistics, available memory for a session, explicitly given available memory for hash joins or merge joins (btw there is possibility to give for one but not for other), system workload, hints, initialization parameters allowing certain join types (at least for hash joins) and probably something other 😊

**dt11**

Oct '07

Hi Jeff,

I think your illustrations make one BIG assumption: Table A and Table B contain unique rows. If either table contained duplicate matching records, then the whole Venn diagram will not hold (the cardinality of the sets will not add up). I am afraid this visual explanation will lead some to think that you can use SQL joins to do filter operations which shouldn't be done this way.

Although it's nice attempt to explain SQL joins, overall I think it is misleading.

-dt

**AlexC**

Oct '07

One of the worst three months of my CS training was a class ostensibly on database theory which was really a boot-camp in SQL syntax, taught by a Johnny-one-note professor who thought SQL and RDMSs were the greatest invention of mankind.

This experience turned me off databases for 15 years until one day in the shower I realized that joins were analogous to set theory, in a rough way. (Yes they are, all you naysayers! "Give me everything that's there AND there" or "Give me everything that's there OR there" may not handle all the possible inputs, but it's a good jumping off point for explaining "inner" and "outer." And who came up with that stupid, arbitrary terminology anyway?)

I still think SQL is an awful language, though, and relational databases are an ugly hack mandated by hardware limitations trumping elegant design. OLAP "cubes" are so clearly a better solution—so intuitively clear and obvious—the natural generalization to higher dimensions of the flat-file database.

**Kevin**

Oct '07

To me, a Cartesian Product would be best illustrated by both circles being blank.

Granted, all the SQL gurus will probably spit feathers at my outrageous suggestion, but for me, it's a beautifully simple method of explaining simple joins and I'll be using it in my technical documentation!

Thanks for this article, Jeff.

**Steve**

Oct '07

Jeff, you are *the* master of well crafted blog posts that sometimes don't say much, but somehow gather scads of inane comments! This one and the previous post (on exercise) are great examples.

Now, not all of your posts are in this category, thank God! But I'm watching you...

**Catto**

Oct '07

Hey Now Jeff,
These diagrams make me think of when learning of classic boolean logic.
Coding Horror Fan,
Catto

**Andreask**

Oct '07

Well, the database practitioner rejoice, and the set relationists cringe.

There is only one join, which is the cross product (each row of the first table concatenated to each row of the second table). The INNER JOIN example only discards some rows by the where clause, otherwise it is (or should be) the same as the CROSS JOIN. The outer join is a hack that isn't very firmly rooted in relational algebra.

**jan_g7**

Oct '07

Nice Jeff, thank you so much for this post!

I always got confused with JOINS and how they work, that's one of these things that are hard to be explained in a book or a manual.

But these diagrams make it pretty clear, they are great!

**MikeW**

Oct '07

Don't forget EXCEPT! Which was (and additionally still is) MINUS in Oracle. For years I think Oracle was the only RDBMS that implemented it and I had, in nearly 20 years, used it all of, oh, once. Until last week, when I used it the second time.

It's the same as your fourth example: instead of

```
SELECT * FROM TableA
LEFT OUTER JOIN TableB
ON TableA.name = TableB.name
```

```
WHERE TableB.id IS null
```

we can say

```
SELECT * FROM TableA
MINUS
SELECT * FROM TableB
```

(I used the old Oracle operator - it's more expressive to my eyes)

Not every platform implements the operator yet: MySQL (which I just checked) doesn't, for example. MS SQL Server 2005 does. SQLite? Hang on ... yup, that works in v3 at least.



KG145

Oct '07

Very strange. In my "Database Management" class at Arizona State University - a student asked about Join's and my professor said "I never use them. Just stick with the WHERE syntax".

This was a more theoretical course though. We spent FAR more time in relational algebra / calculus and algorithms for ensuring atomic transactions than we did in more practical stuff like database normalization.

Hmm... lots of people say the theoretical stuff is a waste of time. For me, well, I almost became a math major instead of CS, so I find the theoretical stuff more interesting.



joske

Oct '07

If you don't use any join-feature and just do "select * from TableA, TableB", you have your cross-join.



david16

Oct '07

Venn diagrams.

wow, so simple, so obvious; how come I haven't seen this anywhere else?



Rock

Oct '07

Hi Jeff

that's an nice diagram and a new way of presenting the sql join function.



BuggyF

Oct '07

Yet another case of the blind leading the blind (down the blind alley of ignorance and arrogance); the original, BTW. At least you made an attempt to fix it up. But reading all those "now I get it" comments, makes it clearer why Yahoos like GW Bush get away with bald faced lies. "People believe what they want to believe, when it makes no sense at all"/Mellencamp.

Now we'll have another gaggle of folks running around fawning on Ambler and Fowler. Gad. The Horror, the Horror.



Chris

Oct '07

Thanks to this I'm now skipping merrily down the path of BOM management and stock control, rather than flicking furiously through Teach-yourself books, thanks



Rasmus

Oct '07

Excellent post.

Everyone who has followed (and passed) a computer science beginner course should be able to understand this.











CR13

Oct '07

It's important to note that when joining two tables on non-indexed columns most databases perform the equivalent of a cross join and then weed out the rows that don't match the join/where criteria. While this doesn't really impact the query results, it has a big impact on query performance.

In MySql you can run: explain my query to determine if the query is using indexes and review performance of all your queries.

-
-  **Wombat** Oct '07
- Perfect timing. I just had a new project thrown on my desk that will require working with SQL - something I know little about and have avoided. Terms that have glazed my eyes in weeks past suddenly make sense.
-
-  **MattW** Oct '07
- dt is 100% right. You really need a disclaimer that indicates that your set diagrams only hold when there is at best a one-to-one correspondence between join criteria values in the tables.
- The second you have two or three pirates in one of the tables (an *extremely* common real-world scenario, probably much more common than the simplified case you show here) your Venn diagrams break down.
-
-  **ThomasB** Oct '07
- What a great way to explain JOINS, I'm certain there are thousands of students that would benefit from this... a picture is truly worth a thousand words (especially in cases like this).
-
-  **Joe_Beam** Oct '07
- I always use left outer joins and never really liked the right outer join. Is there anytime they are useful? It always seems bass ackwards to use a right outer join.
- Of course other developers use visual query designers which use right outer joins...
-
-  **Ricardo** Oct '07
- Is it just me or Oracle's syntax is much simpler than that? Like:
- ```
select * from TableA, TableB
where TableA.Name = TableB.Name
```
- I'm familiar with both, both I always liked that one better.
- 
-  **Jay\_R\_Wren5** Oct '07
- Jeff,
- This is great for those SQL types that didn't take a set theory, relational calculus or even discrete math in college.
- I'd love to see you visualize a cross product 😊
- 
-  **MattW** Oct '07
- Ricardo, that syntax is pre-ANSI. Most databases still support it but the advantages of the INNER JOIN, etc. syntax are: it makes the queries more portable across databases; it allows separation of the join criteria from the filter criteria, which improves readability; and allows full outer joins without syntactic hacks.
- Another huge benefit is that if you want to do a CROSS (Cartesian) JOIN, and usually you don't, you have to call it out specifically. In the old syntax, if you screwed up the WHERE clause you could get wildly huge resultsets due to inadvertent Cartesian joins.
- 
-  **Ed\_Kenny** Oct '07
- Joe Beam,
- The only use I have found for RIGHT OUTER is when i'm too lazy to reorder my query. 😞 Other then that it makes more sense to use only one type.
- I use LEFT JOIN too, as do my company.
- I agree with other responses here Jeff, while it's a really nice simple way to represent joins, it doesn't handle many to many or one to many very well and let's not talk about malformed joins.
- It's a good primer, but should perhaps contain a warning.
-





codinghorror

Oct '07

The commenters pointing out that the diagrams break down in case of multiple and or duplicate results, are absolutely right. I was actually thinking of joins along the primary key, which tends to be unique by definition, although the examples are not expressed that way.

Like the cartesian or cross product, anything that results in more rows than you originally started with does absolutely breaks the whole venn diagram concept. So keep that in mind.



Wyatt

Oct '07

Except that venn diagrams explain set logic, and SQL Joins have very little to do with set logic.

Scary how many people are agreeing with this.



Matt

Oct '07

Actually, I've always been a bit shaky on joins myself. I always have to look them up any time I need to using a join that isn't a standard filtered cross product. But seeing this visual guide has really helped me to grasp it much better. I just went back and read the Wikipedia entry for joins and it makes complete sense to me now. Before seeing this visual representation I might have just glazed over when reading the Wikipedia article.

So even if they aren't exactly right having a visual representation has really helped me to understand what is going on.

Thanks!



Joshua

Oct '07

A nice trick is you can write arbitrary filters in the ON clause of the join. I use this when writing truly awful joins or when I need the left join with an already filtered table.



JeffreyB

Oct '07

From one Jeff to another, this is a brilliant essay on a simple topic - simple but vexing to newbies. I am a DB2 DBA, and explaining outer joins to the newbie developers I encounter constantly will be much easier with materials like this. Thanks, keep it coming.



Jeff

Oct '07

I have never seen such a good illustration of SQL joins. Good work.



Sven\_Groot

Oct '07

I must agree that while the diagrams are a fairly good illustrations of what happens with a join, they're not Venn diagrams in the strict sense because that's just not what's happening. What the first diagram says to me is that you get all records that are both in set A and set B (aka intersection). That's just not true. You get a set of new records, that are neither in set A nor in set B, but they contain a combination of the attributes of both sets.

Aaron G: joins are conceptually a special case of the cartesian product. How the database server actually computes them is another matter entirely.

Conceptually, in relational algebra, an inner join is a cross product followed by a selection on the join condition. That's a terribly inefficient way to compute it, but conceptually, that's what it is.



Lachlan\_McD

Oct '07

The morons running my database fundamentals class at my university thought that teaching the Cartesian product as a method of joining, then filtering the results, was a valid alternative to the insanely simple inner join. So many poor students are going to have a hard time in the industry as a result.



Powerlord

Oct '07

Jon Raynor:

Out of curiosity, but what databases other than Oracle support + for joins?

Anyway, I prefer the "TableA a INNER JOIN TableB b ON a.something = b.somethingelse" syntax over "TableA a, TableB b WHERE a.something = b.somethingelse"

(Most DBs also support "USING(something)" instead of "ON a.something = b.something" if the columns have the same name)

Many DBs will throw an error if I accidentally omit the ON statement instead of giving me a cross join that I didn't want. I imagine that the query parser would also parse it faster, as I specify up front which type of join I'm doing; the query parser doesn't have to figure it out.



**Steve**

Oct '07

Big deal, SQL finally caught up with the 1980's. Used a very fine DBMS and language, NOMAD2, on mainframes that did this stuff but only much better and more thoroughly.



**LigayaT**

Oct '07

nicely done. Though I do have to admit to finding it humorous that what started out as a fast, dirty way to try to clarify to someone (on an IRC channel) why they were getting the wrong information for a join now has others correcting and linking to me.



**Barry**

Oct '07

Bindun!

<http://www.khankennels.com/blog/index.php/archives/2007/04/20/getting-joins/>

Nice post though 😊



**RevMike**

Oct '07

"I was under the impression that INNER JOINS use a hash match internally (in fact, I'm quite certain of this). CROSS JOINS obviously don't. Therefore I don't see how it's reasonable to say that an IJ is just a special case of the CJ. I suppose in bread-and-butter set theory, where there's no notion of a hash or an index, this could work, but this is the world of CS, where the clean academic solution is almost never the most practical one. The IJ you speak of is the naivest possible algorithm, only used when there are no indexes (and if you're designing databases without any indexes, well... stop designing databases)." – Aaron G

An IJ is defined as being the equivalent of a filtered CJ. The fact that this would not be a reasonable implementation does not make a difference.

IJs are ABSOLUTELY NOT implemented exclusively with hash algorithms. Sort merge algorithms and nested loop algorithms are used frequently.



**MichaelC**

Oct '07

Jeff:

Nice illustration. I think your example would be minutely more readable if the left-hand table had an ID column containing values 1,2,3,4 (as it does) but the right-hand table had an ID column containing values A,B,C,D (instead of numbers). It just makes that tiny bit easier to tell what's what by avoiding the two similar-looking-but-different columns.



**JeffF**

Oct '07

Very nice article. I use a Sqlite database (via the command line) to manage my database of comics and this would have been a very nice thing to have when I started out. Hopefully, this will help people in grasping SQL joins.



**Shog9**

Oct '07

I love it. The grumpy comments, that is. Plenty of complaints that this is too simplistic a view to be useful, nary a link to anything meatier.

And SQL-addicts wonder why everyone else who has to deal with RDBS jumps at the first library that lets them treat them as anything other than RDBS...

(just finished re-working a big fat chunk of code originally written to pull all tables

into memory and iteratively query, join, and sort them - even an ultra-simplistic understanding would have done someone a ton of good)

**willd**

Oct '07

The Oracle (+), actually predates Oracle's usage, was because when implemented there was ANSI no outer join syntax.  
In Oracle 8, they added the ANSI syntax. No one should be using the old syntax so hopefully you will not be seeing it out in the wild.

**Andrew**

Oct '07

While this is an interesting example of these kinds of join, I have found that in practice I almost never need anything other than left/inner join. 10 years of practice, and it's not the world's most complicated SQL most of the time, but there it is. Thank GOD that stupid thing didn't illustrate "RIGHT" joins too!

The first "left outer join" is the same as just "left join". The second one is the same as well, just put "**b.id** is null" in the where. And please don't anyone tell me that the optimizer isn't going to figure that out (or do the same thing with the outer.) Well, maybe it won't if it is MySQL, but that's why you don't use that.

Die "outer", die!

**ShawnW**

Oct '07

Excellent visual representation. This should help the novice see a concrete picture in their mind of the different joins. While LINQ isn't quite set based (but close) these same visualizations would work there too.

**Mandar**

Oct '07

Excellent article.

Thanks a lot.

can you write something similar for indexes.

mpunaskar at gmail dot com

**DavidG**

Oct '07

Something you may or may not know. . . .

During the late 70's Maths was re-packaged to include

Venn Diagrams  
Matrices

and so on,  
with the idea of preparing for the coming of computers.  
At least in England that happened...

What they teach now is hard for me to determine, as most victims of the educational system can not count out the correct change in a shop.

**Will\_Harris**

Oct '07

Wow man, your design is stellar. Content is great too...you've got a new reader.

**Terry\_Smith**

Oct '07

Holy crap this is WAY over due! I've read TONS of SQL books and NONE of them had this simple diagram!

**Tokes**

Oct '07

I think the Cartesian equation or Cross Product in Venn Diagram form would be both circles combining to form a sphere.

**smackfu1**

Oct '07

Interesting. I find joins to be pretty obvious personally. Fundamentally, you have two sets of rows, and you're matching them up by some criteria. In an inner join, you only take rows that match up. In a left outer join, you take all the ones from the left, plus

take rows that match up. In a left outer join, you take all the ones from the left, plus the ones from the right that match. In a right outer join, the opposite. If you have multiple matches, you take all possible combinations of them.



**MitchellH**

Oct '07

Thank you! I've been using SQL for so long and have many books on SQL and I have never seen such simple examples. I love it!



**Steve**

Oct '07

The reason many people who do DB development have such an inadequate understanding of this simple topic is because the SQL language is an inadequate tool to try to absorb it from. So, if you never were taught this in school or on the job, it's kind of a surprise.

Some DBMS' has syntax with more descriptive verbs (e.g., 'Merge', 'Subset', 'Reject'), and Venn diagrams as well in the vendor documentation.

SQL 2005 is an improvement over 2000. But the sad fact is many people learn about relational algebra by writing SQL statements.

A less pretty version of much of what Jeff is saying can be found here:

<http://db.grussell.org/section010.html>



**Matt**

Oct '07

With all respect to Jeff Atwood, beware if you've read this post and think, "wow, that's easy – why did noone explain it like that before?" Well, there's a reason why so many SQL tutorials and books don't use these diagrams – they're an inadequate explanation of what database joins are all about.

Joins are indeed analogous to set operations, but it is only an analogy. Understand what's really going on, or you will get burned when you actually have to use the things.



**RobCromar**

Oct '07

Very nice first post on the subject. I think many of the comments are valid and I think it would be good to continue this as a series that builds up the more complex issues of SQL joins.

I, who learned this stuff the hard way, really appreciate these primers!



**Steve**

Oct '07

If only someone had explained SQL to me like this when I started... great blog.



**raveman**

Oct '07

very cool visualisation, please do more of that



**WesleyT**

Oct '07

the problem with venn diagrams is that a rdbms table is not the same thing as a set.



**Andy**

Oct '07

Great visual explanation. I've added that to my list of reference bookmarks for when I (often) forget the nature of each SQL join type.



**WaterBreath**

Oct '07

You're right Jeff.

This is a useful explanation, as far as getting people to understand joins in terms of something they already understand (rudimentary set theory). However, it should be made clear that this is not what is meant by the statement that "the relational database model is based on set theory". For a long time, I thought I understood that statement because I had this Venn-diagram understanding of joins.

Joins do not map directly to the basic additive operations of set theory (union, intersect, difference), which create new sets by way of simply including or excluding elements of other sets. There are ways to get those in SQL, but it involves chaining

UNION and MINUS statements.

Joins, rather, are mapping operations, which create new sets by taking an element from each set, according to a set of rules, and lumping them together in a new element for the new set that is actually itself a small set containing the original two elements from the other sets.

If A is an additive set operation (representable by a Venn diagram), then the result of A on sets X and Y is a set of members of X and Y.

$$A(X, Y) = \{x_1, y_1, x_2, x_3, y_4, y_5, \dots\}$$

In database terms, each element of the resulting set is simply a row from either X or Y.

In real-world terms, it's like sorting jelly beans. Throw them all into a bucket, through a sieve. The sieve eliminates all but the appropriate shaped ones, but what gets through is still what you started with: an assortment of jellybeans.

If B is a mapping set operation (representable by a line-drawing matching question), then the result of B on sets X and Y is a set of sets each containing a member from X and a member of Y.

$$B(X, Y) = \{\{x_1, y_1\}, \{x_2, y_2\}, \{x_3, y_3\}, \dots\}$$

In database terms, each element of the resulting set is a new type of row that is a lumping together of a row from X and a row from Y.

In real-world terms, an example would be taking a bucket of oranges and a bucket of apples, and pulling out pairs of one of each that are the same size, putting the pairs together in little bags, and then putting those bags into a third bucket. You don't get a bucket of apples and oranges. You get a bucket of pairs of an apple and an orange.

Looking at it this way, it should be reasonably easy to see that there is a fundamental difference between joins and the simple union/intersect/difference set operations.



**cloud9ine**

Oct '07

Except that venn diagrams explain set logic, and SQL Joins have very little to do with set logic.

Scary how many people are agreeing with this.

My thoughts exactly!



**JF15**

Oct '07

Jeff, this is great! How about visually explaining why you have SELECT all the columns you are GROUP-ing by? I always have a hard time explaining that one to people!



**Robin\_Day**

Oct '07

I'm afraid this concept is misleading on the reasons mentioned by Sven Groot.

The result of a join is not those items in table A and those items in table B, it is a "joined" combination of the two. The diagrams you have drawn are set diagrams and are created in SQL using the set operators UNION and INTERSECT.

Joins are a completely different concept which I believe are best explained using just the table outputs you have above without adding the incorrect venn diagrams.

JF, the explanation behind SELECT and GROUP BY can be described simply.

```
SELECT COUNT(*) FROM People
-- This will return a count of 100 people
```

```
SELECT EyeColour, COUNT(*) FROM People GROUP BY EyeColour
-- This will return Blue 20, Green 50, Brown 30
```

```
SELECT COUNT(*) FROM People GROUP BY EyeColour
-- This will return 20, 50, 30 whilst this contains the counts that
you are looking for, they are useless as there is no relation to
each count and what the represent.
```



**james\_maida**

Oct '07

Assuming Pirate and Ninja are identical records in both tables, the very first thing that came to mind as what result I would want when I join Tables A B was

Pirate  
Monkey  
Ninja  
Spaghetti  
Rutabaga  
Darth Vader

This seems to be the intuitive meaning of join. Venn's don't work.



**Daniel**

Oct '07

This is a very likable article, I personally really like the way you've done this...



**Dad**

Oct '07

It never ceases to amaze me how many folks become "experts" on a technical topic once somebody takes the time to write about one.

I give props to the author of the article presented here. I needed to understand this information, and it was put to me visually in a way I could parse quite easily.

As for the folks dissecting this article, nit-picking even the premise, I ask that you please write a better one, and post the link instead of complaining. I will then drop by and pick it apart out of professional discourtesy.

Thanks.



**Barfo\_Rama1**

Oct '07

The Cartesian product of 2 sets is the set of all combinations of ordered pairs which can be produced from the elements of both sets. So the Venn representation would look like two piles of poker chips next to each other, each pile numbering the number of data points in the other bottom chip.



**Leon**

Oct '07

Thanks for this great Tutorial 😊



**Satheesh**

Oct '07

Nice way to explain things in joins. By seeing the example nobody cannot forget joins.

Keep it up. Hope you will publish many more article which will be useful to all.

Thanks,  
Satheesh.



**Brian**

Oct '07

thanks for this, I am in a class for crystal reports and I showed many and it helped them



**Jax**

Oct '07

nice quick lesson on joins. This is exactly what i wanted.



**Martin**

Nov '07

Great explanation of the basics of inner and outer joins.

I'll be referring it often and passing it on.

Also very brave to raise anything here as not 100% perfect equals rubbish

Martin



**Ghislain**

Nov '07

Very simple, I'm always confused up with sql operations, but I understand better with simple diagrams like yours.

Thks again for this nice job.



**Luke**

Nov '07

I hank god for venn diagrams! Good job mate.

**Clay**

Nov '07

Wow! Terrific tutorial. I've never understood JOINS beyond the concept of them. Thank you for demystifying!

Clay

**JamesP**

Nov '07

Thank you so much for making this available. Cheers from the UK

**Derek**

Dec '07

This article is great! A picture is worth a thousand words, and your pictures have helped me to understand SQL joins. Thanks.

**Hennie**

Dec '07

Hey that was a CLEAR explanation ... just started up with sqlite and didnt get why i was not getting the selection i wanted.

Now i do!

**Inquisitor**

Dec '07

how do you join two columns together and get one column in a query. For EX: i have lastname and firstname columns in a table. i want to join these two columns together and get lastname and firstname in one column ??

**NED1**

Jan '08

Although I'm familiar with the topic, I still like your nice and easy way of explaining things, nice work.

**SHALL**

Jan '08

This was a GREAT explanation. I am working with a junior DBA and this has helped tremendously. I will have to go back and read some of your other blogs.

AWESOME JOB!!!

**GintsP**

Jan '08

As someone suggested to created something themselves and then criticize 😊 I've tried to describe relationships among various join types using ER diagramm here: <http://gplivna.blogspot.com/2008/01/sql-join-types-im-studying-bit-sql.html> All comments welcome!

**Prajeesh**

Feb '08

It's a great way to explain things... thanks

**Andy**

Feb '08

Thanks for this Jeff, was struggling a bit with the concept but you've made it clear enough that even I can get my head around it 😊

**Arcond**

Mar '08

I really have to hand it to you Jeff, the way you break things down is just great. I read the blogs every time you post one. This one in particular came in handy just the other day. Our QA team was having trouble understanding what was going on in the database and we developers would send them queries to run to find the data they needed. The QA team has a rudimentary understanding of SQL, but couldn't quite grasp joins. I sent a link to this article to our lead QA, and now your article is firmly posted on her cube wall for reference. Just wanted to say thank you for the blogs!

**Ernest**

Mar '08

Years ago I had diagrams like this posted on my desk and I referred to it nearly every

other day. Thank you for posting this as I am sure I will be referring to it often as I try to understand and explain to others the impacts of SQL statements

**MattG**

Mar '08

This has been extremely helpful for a project that I am currently working on. I am by no means a coding expert(not a novice either), but there is a lot of stuff I have yet to learn. The way this is all displayed with the use of venn diagrams has made it very easy to understand, thanks for the help.

**AndrewM**

Mar '08

This page is so helpful!

I am constantly needing to build MySQL queries for my php and .net applications and I come back to this page time and time again to prevent me from having to think too hard!

Just that one less moment spent on these details when you're in the middle of solving big problems makes all the difference.

Thank you!

**gayathri**

Jun '08

hi jeff,

Thanks a lot for this ven diagram explanation. i opened the website without any understanding on sql joins and after reading ur explanation, i've understood it so well. really very helpfull.

Thanks a lot!

**Isa**

Jun '08

Great explain, thanks a lot.

**Jolly**

Jun '08

Its nice way of explanation of SQL Joins .Too easy!

**SivaAnanth**

Jun '08

Excellent way of exhibiting the concept.  
Simple , easy to understand.

**Charlie**

Jun '08

Great and very simply to understand!

**Dulcinea**

Jun '08

Great Work, Im teacher, and Iwill use this, to explain de joins.

**Pallavi**

Jul '08