

Jeff's SQL Server Blog

Random Thoughts & Cartesian Products with Microsoft SQL Server

[<< How to format a Date or DateTime in SQL Server](#) | [Home](#) | [Sorting Columns with the C# Pivot Function >>](#)

Better Alternatives to a FULL OUTER JOIN

As many of you know, I strongly recommend that you avoid using [RIGHT OUTER JOINS](#), since they make your SQL code less readable and are easily rewritten as [LEFT OUTER JOINS](#). In addition, I have yet to find a situation where a [FULL OUTER JOIN](#) makes sense or is necessary -- I have found that in just about every case other techniques work better. (Feel free to suggest some FULL OUTER JOIN situations in the comments and we can discuss)

Let's take a common situation where you need to "merge" the data from two tables, and rows may or may not exist in *either* table. For example, suppose we have budgets in one table and actuals in another and we wish to present them side by side. Many people would use a FULL OUTER JOIN to accomplish this, perhaps writing something like this:

```
SELECT
    coalesce(a.company,b.company) as company,
    coalesce(a.account, b.account) as account,
    coalesce(a.year,b.year) as year,
    coalesce(a.month, b.month) as month,
    coalesce(a.amount,0) as Actual,
    coalesce(b.amount,0) as Budget
FROM
    Actuals a
FULL OUTER JOIN
    Budgets b on
    a.company = b.company and
    a.account = b.account and
    a.year = b.year and
    a.month = b.month
```

The above FULL OUTER JOIN will effectively "merge" the two tables and allow you to see all actuals and budgets for every company/account/year/month, and it will show values from either table even if there is not a matching value in the other table. Essentially, it gets the job done.

There are some important things to note in the above, however:

- There is *no relation* between the Budgets and Actuals table (i.e., they do not have a 1:1 or 1:M relationship to each other) yet *we are joining them together!* To me, this does not make logical sense.
- Suppose that the PK of the Budgets table allows for more than 1 budget row per Company/Account/Year/Month -- this would result in two Budget rows matching a single Actual row, duplicating the Actual amount. You must be absolutely sure when doing a FULL OUTER JOIN with any sort of totaling that the two tables will never have more than 1 matching row to the other table.

- What is the "driving" table behind this SQL statement? Even though we are clearly selecting FROM the Actuals table, the Actuals do not "drive" our results. And even though we are joining FROM the Actuals TO the Budgets, some rows will have data only from the Budget table without matching Actual rows. To me, this is difficult to read and logically interpret; it doesn't follow the standard "select from table A join to table B" logic that is clear and easy to understand and work with. (note: this is basically my same argument against RIGHT OUTER JOINS.)
- *Every column* that is returned from *either table* is potentially nullable. All of them. You cannot reference a column in either table without handling the case where that column might have a null value. Thus, *every column* in either table *must* be wrapped in an ISNULL(), CASE, or COALESCE() expression. Most importantly: *this includes all non-nullable primary key columns in both tables!*
- Once you've wrapped every column in an expression, no further use of indexes from those columns can be used. For example, if I join the result of this FULL OUTER JOIN to other tables to show Account or Company names or information, *no existing indexes on the Actual or Budget table can be used on that join since every column is an expression!* Thus, as soon as you use a FULL OUTER JOIN, you completely eliminate all indexes from both tables involved.
- Since no indexes are usable in the results, any sorting done on the results is done on expressions and it will not be optimally efficient as well.

So, based on the above, I feel there's two basic issues with FULL OUTER JOINS: The code itself is difficult to interpret and isn't especially clear, and the result returned is just a big mess of nullable columns all wrapped in expressions which isn't a clean and efficient set of data to work with.

I have always felt that it is very important in any SELECT to clearly establish your FROM clause as the primary, "driving" data source, and then to join from that primary source to the auxiliary tables or SQL statements via JOINS. However, in our example and with FULL OUTER JOINS in general, we don't really want to relate one table to another, we want to MERGE these two tables together. When merging the data from two tables, doesn't a [UNION](#) make more sense? Why express our intentions as a (FULL OUTER) JOIN when the true operation you are after is actually a UNION?

Consider this SQL statement as an alternative:

```
SELECT
    company,
    account,
    year,
    month,
    sum(actual) as actual,
    sum(budget) as budget
FROM
    (
        SELECT
            company, account, year, month, amount as actual, 0 as budget
        FROM
            Actuals
        UNION ALL
        SELECT
```

```

        company, account, year, month, 0 as actual, amount as budget
FROM
    Budgets
) x
GROUP BY
    company, account, year, month

```

(note: we are basically using the technique I described [here](#)).

Some notes:

- Since there is technically no direct relation between Budgets and Actuals (i.e., there is not a 1:1 or 1:M relation between the two tables) this SQL make more sense because it is not implying or stating that we are joining these two tables together.
- If there are multiple budget amounts that match a single Actual amount (or vice versa) for a given company/account/year/month, the correct totals are still calculated -- since we are not joining the two tables, the Actual row will not be duplicated if it matches multiple Budget rows.
- The UNION makes it very clear that we are taking the results from two tables together, and merging them into 1 set of rows. The FROM clause truly and accurately describes the primary data that this SELECT will use.
- The primary key columns returned are not wrapped in a COALESCE() functions; they can never be NULL since they come directly from one of the two tables.
- Joins or sorts on our indexed columns can now be used.

Even if the UNION ended up being a little longer or even in some cases slightly less efficient, I still feel it is clearer and more readable to use a UNION instead of a FULL OUTER JOIN and it is usually worth the cost.

Other ways of handling this situation would be to use a CROSS JOIN as well, which might even be necessary depending on the results that we want. If we want to always return *all* combinations of Companies/Accounts/Months/Years even if there is no Budget or Actual data available for that combination, a CROSS JOIN is the answer you need:

```

SELECT
    All.Company,
    All.Account,
    All.Year,
    All.Month,
    coalesce(Actuals.Amount,0) as Actual,
    coalesce(Budgets.Amount,0) as Budget
FROM
    (
        SELECT
            C.Company, A.Account, M.Year, M.Month
        FROM
            Companies C
        CROSS JOIN
            Accounts A
        CROSS JOIN

```

```

        Months M
WHERE
    M.Year = 2006    -- or whatever criteria you need here ....
) All
LEFT OUTER JOIN
    Actuals
        ON Actuals.company = All.Company and
        Actuals.Account = All.Account and
        Actuals.Year = All.Year and
        Actuals.Month = All.Month
LEFT OUTER JOIN
    Budgets
        ON Budgets.company = All.Company and
        Budgets.Account = All.Account and
        Budgets.Year = All.Year and
        Budgets.Month = All.Month

```

(The technique shown above is discussed further [here](#).)

In the above, our primary "driving" data source is a series of CROSS JOINS that produces the combination of rows we need to return, and then from there we do two separate LEFT OUTER JOINS to our transactional tables. (Note that, as with a FULL OUTER JOIN, if the primary key of either the Budgets or the Actuals table isn't Co/Acct/Year/Month, you should join instead to summarized derived tables that group on those key columns to prevent duplicating rows in the result.)

Thus, even though it seems that maybe a FULL OUTER JOIN or UNION is necessary to "merge" or two tables, sometimes a CROSS JOIN/LEFT OUTER JOIN can work as well, and it indeed fulfills a different requirement that is not possible (or easily achieved) with those other techniques by guaranteeing the exact set of rows to be returned, even if no matching transactions exist.

In conclusion, think carefully before unleashing your next FULL OUTER JOIN. I'm sure it has its place, and someday I'll see a situation where it makes sense for me, but for now, I'm going to stick with UNION, CROSS JOINS and LEFT OUTER JOIN to solve these SQL problems. After all, who wants to wrap every single column in a coalesce() function when it's so damn hard to spell !

Updates:

- I have published a test SQL script [here](#) that you can play around with to see some of the performance differences between a FULL OUTER JOIN and an UNION ALL (more on this in the comments). Preliminary testing shows that UNION ALL is about twice as fast.
- There is also a script [here](#) comparing a FULL OUTER JOIN with an equivalent CROSS JOIN. If the columns you are joining/grouping on are foreign key references to indexed tables in your database, a CROSS JOIN also can be slightly more efficient (in addition to being shorter and clearer) than using a FULL OUTER JOIN.


see also:

- [Taking a look at CROSS APPLY](#)
- [The "Nested WHERE-IN" SQL Anti-Pattern](#)
- [Using GROUP BY to avoid self-joins](#)
- [Criteria on Outer Joined Tables](#)
- [Better Alternatives to a FULL OUTER JOIN](#)

- [Conditional Joins in SQL Server](#)
- [How to JOIN Multiple Transactional Tables in SQL](#)
- [The power of the Cross Join](#)

Related Links

[Using GROUP BY to avoid self-joins](#) (6/12/2007)
[Be Careful When Mixing INNER and OUTER Joins](#) (10/11/2007)
[Criteria on Outer Joined Tables](#) (5/14/2007)
[Taking a look at CROSS APPLY](#) (10/18/2007)
[The "Nested WHERE-IN" SQL Anti-Pattern](#) (7/12/2007)

 [Print](#) | posted on Thursday, April 19, 2007 11:56 AM | Filed Under [[T-SQL Techniques](#) [Efficiency](#) [Report Writing](#) [Joins/Relations](#) [GROUP BY](#)]

Feedback



re: FULL OUTER JOIN

I've seen good full outer join situations. Suppose you have a parent table (A), and two child tables (B) and (C). Further suppose that the Primary key of B and C is the IDENTITY column from A.

You are given a report to write which should show a header record for A, and underneath it, a column for B, and a column for C. An easy way to do this is to create a sub-query that FULL OUTER JOINS B and C on the primary key values and some other sequencing value, then INNER JOIN table A to the sub-query results.

I'd be interested in any performance analysis between this and alternative methods.

4/19/2007 1:22 PM | [Chris McKenzie](#)



re: FULL OUTER JOIN

Sorry, the Primary Key in B AND C should be a combination of the IDENTITY column from A, and a sequence number.

4/19/2007 1:24 PM | [Chris McKenzie](#)



re: FULL OUTER JOIN

Chris --Can you give an example? Sounds like all you need is this:

```
select id, sequence, sum(bvalue), sum(cvalue)
from
(
select id, sequence, value as bvalue, 0 as cvalue
from b
union all
select id, sequence, 0 as bvalue, value as cvalue
from c
```

) x
group by x.id, x.sequence

You can just join that to table a if necessary, of course.

If that isn't what you had in mind, an example would be great.

4/19/2007 1:58 PM | [Jeff](#)

re: FULL OUTER JOIN

There's a problem with your RSS feed, it seems to only show the first paragraph, which renders it pretty useless.

4/19/2007 2:22 PM | [Ian](#)



re: Alternatives to a FULL OUTER JOIN

Ian -- An RSS feed that doesn't give you the entire article is useless?? I think there's a problem with your RSS reader. The feed contains the summary and a link to the article. Any good RSS reader should let you view the page the feed comes from via a simple click. If yours doesn't, you should look into a new one because that's pretty a basic feature! I also double-checked the RSS page itself, the feed is fine.

4/19/2007 5:00 PM | [Jeff](#)



re: Alternatives to a FULL OUTER JOIN

Example:

```
SELECT *  
FROM tblA  
INNER JOIN  
(  
  SELECT *  
  FROM tblB  
  FULL OUTER JOIN tblC ON tblB.ID = tblC.ID  
  AND tblB.SequenceNumber = tblC.SequenceNumber  
) SubQuery ON tblA.ID = SubQuery.ID
```

I suppose you could use a union now that I think about it, although I'm not sure how that compares in terms of performance. I'd have similar concerns about the CROSS JOIN. Sounds like a benchmark test is called for.

Also, I'd about the possibility of losing data due to the union's side-effect of removing duplicate rows.

4/20/2007 8:22 AM | [Chris McKenzie](#)

re: Alternatives to a FULL OUTER JOIN



Chris -- we are using UNION ALL, not union, which doesn't remove duplicates. But either way, that is not factor since no rows can be duplicates, see the SQL that I wrote and the way it is structured.

You didn't write it out fully, but as mentioned in the article, you'd need to wrap EVERY COLUMN in your FULL OUTER JOIN derived table in a COALESCE() function, and if any rows in table C or B match more than 1 row in the other table, you will get duplicate rows which a union all avoids.

In fact, your example is **exactly** the same as the example I used in my article, so pretty much you can just refer to the article for all the reasons not to write that particular sql statement with a FULL OUTER JOIN. A union really truly works perfectly for this type of thing, if you haven't tried it you really should.

As for cross joins, I put them in there as an example, but they serve a different purpose: when you need to ensure that all combinations of rows are returned, even when there is no matching transactional data. (see the article I linked to, as well) You'd only use a CROSS JOIN when you have this requirement; if you don't, there's no reason to. If you do have this requirement, the CROSS JOIN is actually your most efficient and clear way to get this result.

4/20/2007 8:51 AM | [Jeff](#)



re: Alternatives to a FULL OUTER JOIN

Chris -- a simple comparison of the two techniques which you can test for performance and adjust as necessary is here:

<http://weblogs.sqlteam.com/jeffs/articles/60180.aspx>.

Preliminary testing shows that the UNION is about twice as fast.

4/20/2007 9:17 AM | [Jeff](#)



re: Better Alternatives to a FULL OUTER JOIN

Just added a CROSS JOIN comparison here:

<http://weblogs.sqlteam.com/jeffs/archive/2007/04/20/60181.aspx>.

Even a CROSS JOIN, if you have properly indexed tables and established foreign key constraints, is faster and clearer. Those results might not be typical in every case, of course, but it is always worth considering a CROSS JOIN solution as well (especially when you have the requirement to still show \$0 rows even when no transactions exist).

4/20/2007 10:57 AM | [Jeff](#)



re: Better Alternatives to a FULL OUTER JOIN

I gave the wrong example. Your criticisms stand.

Given:

One parent table with two child tables, each with a sequence number identifier.

Requirement:

A report that shows the parent in one column, childA in column2, and childB in column3.

Analysis:

There is no direct relationship between ChildA and ChildB, except that they are both children of Parent, and that they each have a sequence number.

The problem with a union is that it "stacks" the results of one query with the results of another. We need results from tblChildA in one column (or set of columns), and results of childB in another (set of) column(s).

Solution:

```
SELECT
Parent.ParentID
, ChildAID
, ChildASequenceNumber
, ChildBID
, ChildBSequenceNumber
FROM tblParent
INNER JOIN
(
SELECT
ParentID = COALESCE(tblChildA.ParentID, tblChildB.ParentID)
, ChildAID = tblChildA.ID
, ChildASequenceNumber = tblChildA.SequenceNumber
, ChildBID = tblChildB.ID
, ChildBSequenceNumber = tblChildB.SequenceNumber
FROM tblChildA
FULL OUTER JOIN tblChildB ON tblChildA.ParentID = tblChildB.ParentID
AND tblChildA.SequenceNumber = tblChildB.SequenceNumber
) SubQuery ON tblParent.ParentID = SubQuery.ParentID
```

Notice that there is only one COALESCE() function. This query suited my needs perfectly. I attempted to construct the subquery using a CROSS JOIN combined with a WHERE clause, but the issue I ran into with that was that if one table had no records for the same Parent-Sequence as the other, the record was excluded from the result set.

Still, your point that a CROSS JOIN is worth considering is well taken.

4/20/2007 11:13 AM | [Chris McKenzie](#)

re: Better Alternatives to a FULL OUTER JOIN

First off, before I forget -- Thanks for the feedback and ideas, I really appreciate



people taking interest in this and taking the time to provide some ideas and alternative approaches and all that. It's much more fun when I throw some of my crazy ideas out there and I get some good, intelligent responses.

Now, back to your example:

While a cross join might be better in this case (assuming sequence has meaning and it is stored in a table, which I hope it is if you are using it to relate two OTHER tables!) a union all works beautifully here as well:

```
select
parentID, sequence, max(childAID), max(ChildBID)
from
(
select parentID, sequence, childID as childAID, null as ChildBID
from b
union all
select parentID, sequence, null, childID
from c
) x
group by parentID, sequence
```

It really is a great technique. To be honest and fair, these really are not good examples -- you would not have a sequence in these tables and join two transactional together like this, is just doesn't make much logical sense. A sequence is typically derived from data, not stored and use for relations between separate transactional tables. However, it is still a good exercise and it still demonstrates the benefits of alternative approaches to FULL OUTER JOINS nonetheless.

4/20/2007 11:23 AM | [Jeff](#)



re: Better Alternatives to a FULL OUTER JOIN

Actually, I see what you are doing there, and it's really interesting. I did a quick compare of the two methods on live data, and they seem to be relatively equivalent in performance. BTW, the example I gave is from a live production system, so it is a decent example to my way of thinking. I still personally prefer the syntax of the FULL OUTER JOIN, but you've shown me a real alternative to the FOJ, and for that I thank you.
Cheers!

4/20/2007 1:27 PM | [Chris McKenzie](#)



re: Better Alternatives to a FULL OUTER JOIN

Hello,

Take this following example to compare two tables.

```
CREATE TABLE A (PKCol int NOT NULL, DataCol char(20) NOT NULL)
CREATE TABLE B (PKCol int NOT NULL, DataCol char(20) NOT NULL)
```

```
-- same tables, same values
INSERT INTO A (PkCol, DataCol) SELECT 1, 'SameVal'
INSERT INTO B (PkCol, DataCol) SELECT 1, 'SameVal'

-- same tables, different values
INSERT INTO A (PkCol, DataCol) SELECT 2, 'DiffVal'
INSERT INTO B (PkCol, DataCol) SELECT 2, 'DiffVal'

-- only in 1 table
INSERT INTO A (PkCol, DataCol) SELECT 3, 'A'
INSERT INTO B (PkCol, DataCol) SELECT 4, 'B'

SELECT ISNULL(A.PKCol, B.PKCol) PKCol
, CASE WHEN A.PKCol IS NULL THEN 'Missing in A' WHEN B.PKCol IS NULL THEN 'Missing
in B' ELSE 'Available in Both' END Status
, CASE WHEN ISNULL(A.DataCol, '') = ISNULL(B.DataCol, '') THEN 'OK' ELSE
ISNULL(A.DataCol, '<dbnull>') + ' <> ' + ISNULL(B.DataCol, '<dbnull>') END DataCol
FROM A FULL OUTER JOIN B ON A.PKCol = B.PKCol

DROP TABLE A
DROP TABLE B
```

4/23/2007 6:18 AM | [Chris B](#)



re: Better Alternatives to a FULL OUTER JOIN

Thanks, Chris. In general, I have found it is much easier, shorter and more efficient to use the technique and results produced here (<http://weblogs.sqlteam.com/jeffs/archive/2004/11/10/2737.aspx>) to compare two tables (esp. when you have many columns to compare, not just a single value), but if you want to return results in the same format as you are, a UNION works perfectly as well, though it might be a tad longer. The resulting execution plan is more efficient, though not by a lot.

```
select pkcol, case when max(aVal) is null then 'Missing in A'
when max(bVal) is null then 'Missing in B'
else 'Available in Both' end as Status,
case when isnull(max(aVal), '') = isnull(max(bVal), '') then 'OK'
else isnull(max(aVal), '<dbnull>') + ' <> ' + isnull(max(bVal), '<dbnull>') end as DataCol
from
(
select pkcol, dataCol as aVal, null as bVal
from a
union all
select pkcol, null as aVal, dataCol as bVal
from b
) x
group by pkcol
order by pkCol
```

Note that I made two changes minor to your DDL: I added a primary key constraint to your tables (very important when comparing two techniques! we need constraints

and indexes, right?) and also I changed the second set of inserts to add different values in the DataCol (since they weren't really different values; they were inserting the same value). Neither affected performance, but it is important to have sample data that covers all possible results and have indexes in place when writing test code. Also, I added an ORDER BY pkCol to your results, which is the big performance killer for your technique for all the reasons mentioned in my article.

Thanks for the feedback!

4/23/2007 8:43 AM | [Jeff](#)



re: Better Alternatives to a FULL OUTER JOIN

I have three tables, each of whose rows can be associated to any number of rows in the other two (through reference tables). I need to be able to pick one row in table A and see all associations - or lack thereof - between those rows of tables B and C which are associated with the row in A.

If tables have an N:N relationship with each other, then how do you get around using a full outer join if you want to see all of their associations?

4/24/2007 6:57 PM | [Robert Notwicz](#)



re: Better Alternatives to a FULL OUTER JOIN

Rob -- If you can provide some DDL, some sample data, and expected results, I've love to help you out, but right now I have no idea specifically what you mean or how your tables and data is structured.

4/24/2007 7:13 PM | [Jeff](#)



re: Better Alternatives to a FULL OUTER JOIN

Jeff,

I agree in principal, that in an actual application FULL JOIN should be used rarely, if ever. However, I do find it useful in some external cases, such as ETL operations, or as a poor man's data diff. In the ETL case, if I have two tables from god-knows-where, that may or may not have some actual relationship, using a full join can help me visualize the relationship. As a poor man's diff -- if you have two identical or similar tables from different versions or instances, you can use the full join to say, "oh, there's 200 records for Key = 'Foo' on the left but 250 for Key = 'Foo' on the right."

5/31/2007 12:00 PM | [Matt N](#)



re: Better Alternatives to a FULL OUTER JOIN

I want to Convert this query from Sql to InterBase...

```
SELECT LabScreeningMap.Facility, LabScreeningMap.SubjectCode,  
LabScreeningMap.ResultCode, ScreeningTerm.Id, ScreeningMeasure.Prompt,  
ScreeningTerm.Description, LabScreeningMap.SubjectDesc,  
LabScreeningMap.ResultDesc FROM ScreeningMeasure INNER JOIN ScreeningTerm ON  
ScreeningMeasure.ScreenId = ScreeningTerm.Id CROSS JOIN LabScreeningMap
```

7/17/2007 6:01 AM | [Gani](#)



re: Better Alternatives to a FULL OUTER JOIN

Hello

Cross join keyWord is not supported by InterBase1.2 Version. here i have been mentioned a query below which is supported by SQL. Pls Find the solution for this query in InterBase and reply me as soon as possible...

```
SELECT LabScreeningMap.Facility, LabScreeningMap.SubjectCode,  
LabScreeningMap.ResultCode, ScreeningTerm.Id, ScreeningMeasure.Prompt,  
ScreeningTerm.Description, LabScreeningMap.SubjectDesc,  
LabScreeningMap.ResultDesc FROM ScreeningMeasure INNER JOIN ScreeningTerm ON  
ScreeningMeasure.ScreenId = ScreeningTerm.Id CROSS JOIN LabScreeningMap
```

7/17/2007 6:08 AM | [Gani](#)



re: Better Alternatives to a FULL OUTER JOIN

I have to side with all the non-Jeffs here and say that I like a FULL JOIN when I need it and I would also add that newbies should take your thoughts/discussions on using RIGHT and FULL JOINS with a grain of salt until they learn joins more deeply otherwise they'll make decisions for the potentially wrong reasons.

Regarding the FOJ, let's say that the UNION ALL is marginally faster. Even if it is, I favor ease-of-readability to 0.000001% of performance any day. I think that adding the complexity of the UNION ALL certainly falls under that heading (if, in fact, your UNION ALL is faster which wasn't proven conclusively to me in a test of 50 or so rows). And even if the UNION ALL is marginally faster, aren't you concerned that you are hacking around a limitation of the software? Ideally, SQL Server would perform the FULL JOIN at optimum speed and the UNION ALL would not be faster. Who's the say a future hotfix/service pack will not render your technique slower?

Anyway, I felt compelled to reply since I don't at all agree with statements like, "I strongly recommend that you avoid using RIGHT OUTER JOINS, since they make your SQL code less readable and are easily rewritten as LEFT OUTER JOINS. " and "...I have yet to find a situation where a FULL OUTER JOIN makes sense or is necessary." I do find plenty of use of a RIGHT JOIN and I've written tons of FULL JOINS. I'm not a

perfect SQL programmer by any means but I'll hold my code up with anyone's when it comes to performance and time-to-develop and my attitude is that I write what is necessary and logical to achieve the results I need.

RIGHT JOINS and FULL JOINS aren't common unless you are doing complex reporting, ETL (as MattN pointed out), or are dealing with a poorly designed database. My first thought, when I read this post, was, "Just because *you* don't write those types of queries daily doesn't mean that they are not commonplace" (with "commonplace" being a relative term to the type of data/application you are working with). Writing UNION ALL and rearranging my queries to fit some model like this just doesn't seem logical and I can imagine the questions I would receive in a peer review. Oh, and if I didn't want those questions, I would have to write a paragraph of comments at the beginning of each query explaining why I was not using a RIGHT or FULL JOIN. Ugh.

Anyway, I appreciate the idea and I always love to see/learn new approaches but I felt compelled to let anyone who is a newbie and is reading this that there are plenty of experienced, professional programmers who would disagree with Jeff's statements about RIGHT and FULL JOINS. The title of this could be, "A different Approach to FULL JOINS" or "Why I don't Use Full Joins" but the title of "Why to avoid FULL OUTER JOINS in SQL -- and what to use instead" (the title in reddit) is just sensationalist and, to someone like me who has been a SQL programmer for a decade, clearly nothing more than an attempt to garner attention regardless of facts and content.

7/30/2007 10:44 AM | [Scott Whigham](#)



re: Better Alternatives to a FULL OUTER JOIN

Scott -- I appreciate your feedback!

I think you missed all of the points of coalesce() functions, the lack of index usage, and the clarity of the code. FULL OUTER JOINS do not result in shorter code, or more efficient code. I can only suggest that you read the article a little more closely.

As for RIGHT OUTER JOINS, let me state it plainly: There is NEVER a need or reason to use them. NEVER. Feel free to give me an example if you disagree. They can and should always be written properly and logically as LEFT joins.

A good, solid, clean sql statement works like this:

- 1) Select FROM your starting point, your base set of data
- 2) outer join TO your auxillary data.

FULL and RIGHT OUTER JOINS break this model. There is no query that you cannot write that doesn't follow this model, and it will always be clearer and easier to work with. As I wrote earlier, a badly designed database does NOT require you to suddenly use FULL/RIGHT OUTER JOINS; all the more reason to write good, solid code.

As for "Agreeing with the others", in every case I was able to demonstrate that other techniques work more efficeintly and/or are shorter and easier to use than FULL OUTER JOINS. Care to submit your own example?

You made a great point about beginners, and thats part of who I am addressing: It is

a bad habit to start using RIGHT/FULL outer joins because they do not teach good, solid, clean SELECT writing; they do not follow the "select from primary data, join to auxillary data" model which is crucial to use, especially as your code and databases become larger and more complex.

So, I definitely appreciate your feedback, but it may pay off for you to read the article a little more closely and also the comments. And, please, I invite you to submit your own examples of where you feel a FULL/RIGHT outer join makes sense and is more efficient or cleaner overall.

Thanks again!

7/30/2007 11:21 AM | [Jeff](#)



re: Better Alternatives to a FULL OUTER JOIN

Oh, I almost forgot to respond to this:

>>Regarding the FOJ, let's say that the UNION ALL is marginally faster. Even if it is, I favor ease-of-readability to 0.000001% of performance any day. I think that adding the complexity of the UNION ALL certainly falls under that heading (if, in fact, your UNION ALL is faster which wasn't proven conclusively to me in a test of 50 or so rows). And even if the UNION ALL is marginally faster, aren't you concerned that you are hacking around a limitation of the software? Ideally, SQL Server would perform the FULL JOIN at optimum speed and the UNION ALL would not be faster. Who's the say a future hotfix/service pack will not render your technique slower?

First of all, UNION ALL is not only faster, but often shorter, logically cleaner, and returns more accurate results. read the article for all of the reasons why; no need to repeat that all again here.

As for "hacking around a limitation of software" -- what? Can you explain? These FULL OUTER JOIN drawbacks are NOT a limitation of SQL Server -- it is a limitation to how they work. They ALWAYS will return NULLs in all columns potentially, they ALWAYS will require you to state you are joining two non-related tables, and they ALWAYS will result in COALESCE() or similar functions that wrap all columns resulting in a lack of index usage. that's a simple fact and a result of how they work. This isn't because SQL Server isn't coded "properly" or anything like that; a service pack will not fix it!

It's simply good SQL to write your queries in a clean, well structured manner. FULL OUTER JOINS are not clean or well structured, in regards to how you logically write them or in the results that they return.

I suspect that you read this article from a viewpoint of "I use FULL joins, let me try to prove this guy wrong and defend my code" instead of objectively reading it and carefully considering the reasons I've given and the examples. Perhaps try reading it again, but this time with an open mind?

7/30/2007 11:34 AM | [Jeff](#)



re: Better Alternatives to a FULL OUTER JOIN

If you don't scatter the data across a bunch of tables you don't need to use any kind of join. Try normalizing the data into fewer tables. See my article on Object-Oriented Databases here:

<http://typicalprogrammer.com/programming/object-oriented-databases/>

7/30/2007 1:50 PM | [Greg Jorgensen](#)



re: Better Alternatives to a FULL OUTER JOIN

Greg, that'd be denormalizing, not normalizing. I shan't bother pointing out what normalization helps with, if you're dead-set against it, you won't be convinced.

7/30/2007 11:23 PM | [NevDull](#)



re: Better Alternatives to a FULL OUTER JOIN

why doesn't the following work?

```
mysql> select * from girls full outer join boys where girls.city = boys.city;
```

where :

```
drop table girls;
drop table boys;
create table girls (name varchar(12), city varchar(12));
create table boys (name varchar(12), city varchar(12));
insert into girls values('Mary', 'Boston');
insert into girls values('Nancy', null);
insert into girls values('Susan', 'Chicago');
insert into girls values('Betty', 'Chicago');
insert into girls values('Anne', 'Denver');
insert into boys values('John', 'Boston');
insert into boys values('Henry', 'Boston');
insert into boys values('George', null);
insert into boys values('Sam', 'Chicago');
insert into boys values('James', 'Dallas');
```

If needed, I can go into insert and create for you!

BTW left and right work OK!

BTW what do P1(T1,T2), P1(T1,T2) and R(mean in the following

```
SELECT * T1 LEFT JOIN T2 ON P1(T1,T2)
WHERE P(T1,T2) AND R(T2)
```

my academic theory book doesn't cover this!

8/27/2007 8:42 PM | [london kelsey](#)



re: Better Alternatives to a FULL OUTER JOIN

I have two results sets that I want to compare. In an ideal world, they would have been identical, but I don't live in an ideal world. They represent the amount of resources used by various workloads. The "correct" workload identification was phased in over time, with a "stop gap" measure in place before it, and the two methods now overlap. So - I want to see how well correlated, and by how much, the resource usage numbers are for each workload since both have been in place, so I know how much confidence I can put in the results from only the "stop gap" measure before the "correct" scheme was available. I need to allow for the fact that a workload might be mis-named in one result set or the other, and I don't want to miss a row in either result set.

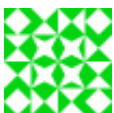
I was planning on doing a full outer join of these two result sets as derived tables. As derived tables, there's no indexes for them, anyway, so that should be a moot point.

The only way I can see of creating a driving table would be to do THIRD derived table which does a "union" of the two sets of workload names, and then do a left out join of that to both of my previous derived tables. In fact, I considered doing that, but it seems so costly as each of the original results sets takes some significant system resources to generate. The driving table would basically be made up from the union of the "stripped down" SQL for each of the original result sets to only return the workload name.

Your thoughts on this subject would be very welcome.

Thanks!

9/26/2007 6:17 PM | [John Lind](#)



re: Better Alternatives to a FULL OUTER JOIN

I have a case where full outer join or any other join will do the trick. Please help.

I have created a table called Calender to help me with the issue I am having after reading the following post: <http://sqlserver2000.databases.aspfaq.com/why-should-i-consider-using-an-auxiliary-calendar-table.html>

The table will be very usefull once i lean how to use it properly.

Here is my issue.

I have a table that contains of all my orders. Each order as an order date and each order has a customer. The goal is to show the number of order by month by customer. I was able to do it in a cross tab in Crystal report but I can't show it on a graph. Crystal does not dispaly 0 for the months where there are no orders.

As a workaround, if I can figure out how to query my table and show the total number of orders by customer by month into a view, crystal will behave. So if I have a customer that ahs no order in january I want my view to show january 01, 2007 (for example) 0.

I hope I am making sense and someone can help me with this one.

Thanks

11/26/2007 5:04 PM | [Frenchie](#)



re: Better Alternatives to a FULL OUTER JOIN

Frenchie --

You need to cross join Customers and Months, and then OUTER JOIN to your data. This is discussed here:

<http://weblogs.sqlteam.com/jeffs/archive/2005/09/12/7755.aspx>

11/29/2007 4:15 PM | [Jeff Smith](#)



re: Better Alternatives to a FULL OUTER JOIN

OK -- I'll toss a log on the fire. This is a piece of dynamically constructed SQL from a stored procedure I was looking at this morning:

```
SET @execstr =  
'SELECT ' + @topX + ' ISNULL(U.Dimension, T.Dimension) AS Dimension,  
ISNULL(T.Measure, 0) AS Measure1, ISNULL(T.Cnt, 0) AS Cnt1, ISNULL(U.Measure, 0)  
AS Measure2, ISNULL(U.Cnt, 0) AS Cnt2' +  
' FROM (SELECT ' + @dimension + ' AS Dimension, ' + @formula + ' AS Measure,  
Count(*) AS Cnt' +  
' FROM Opportunities ' + @whereClause1 +  
' GROUP BY ' + @dimension + ') AS T FULL OUTER JOIN' +  
' (SELECT ' + @dimension + ' AS Dimension, ' + @formula + ' AS Measure, Count(*)  
AS Cnt' +  
' FROM Opportunities ' + @whereClause2 +  
' GROUP BY ' + @dimension + ') AS U ON T.Dimension = U.Dimension' +  
' ORDER BY U.Measure DESC, T.Measure DESC'
```

This stored procedure is generating a data set to populate a chart for comparing performance of members of a dimension across two time periods -- the where clauses include distinct date intervals. Notably, neither of the sets being joined is truly auxiliary to the other -- if U has fewer rows than the top X condition calls for, it's desirable for the table to fill in with the top rows from T.

The API for the chart software makes it necessary for each member of the dimension to have one row in a table with both sets of data elements. Replacing NULLs with 0 is also a requirement of the chart software API.

Obviously, I'm in SQL Server land -- ISNULL vs. COALESCE can be left for another time.

Curious how you would see this working without the FULL OUTER JOIN.



re: Better Alternatives to a FULL OUTER JOIN

Richard -- this is pretty bad code; you probably don't even need a JOIN at all since the data is coming from the same table. However, since it is all dynamic it is hard to tell for sure.

So, you have two simple options to fix this monstrosity:

1) use case expressions and a simple group by, no joins required at all

or

2) use a union all and follow the examples shown in my article.

Both will be shorter, cleaner, faster and easier to read/write than a FULL OUTER JOIN.

example using UNION ALL:

```
select <dimension>, sum(measure1), sum(cnt1), sum(measure2), sum(cnt2)
from
(
select <dimension>, <measure1> as measure1, <cnt1> as cnt1, 0 as measure2, 0 as
cnt2
from Opportunities
where <where1>

union all

select <dimension>, 0,0, <measure2> as measure2, <cnt2> as cnt2
from Opportunities
where <where2>
) x
group by <dimension>

...
```

In this case, I would probably just recommend using CASE:

```
select <dimension>,
(case when <where1> then <measure1> else 0 end) as measure1,
(case when <where1> then 1 else 0 end) as cnt1,
(case when <where2> then <measure2> else 0 end) as measure2,
(case when <where2> then 1 else 0 end) as cnt2
from
Opportunities
group by
<dimension>
where
<where1> or <where2>
```

Either way, there is no reason to use a FULL OUTER JOIN. Both of the above make

much more sense and have all of the benefits I explained in the article in regard to clarity, use of indexes, performance, etc..

12/13/2007 2:04 PM | [Jeff Smith](#)



re: Better Alternatives to a FULL OUTER JOIN

Jeff,

I'm curious on how you would solve this problem. I usually use a FULL OUTER JOIN for this. Lets say you have two sets of customers and you want to find out which states do not have customers in one set of data but not the other.

I usually do this with a left join like this

```
SELECT COALESCE(a.state,b.state) as thestate, CASE WHEN a.state IS NULL THEN 0 ELSE 1 END As in_A, CASE WHEN b.state IS NULL THEN 0 ELSE 1 END As in_B
FROM (SELECT state FROM Customers_2007 GROUP BY state) A FULL JOIN
(SELECT state FROM Customers_2006 GROUP BY state) B ON A.state = B.state
WHERE a.state IS NULL or b.state IS NULL
```

2/13/2008 8:24 PM | [Regina](#)



re: Better Alternatives to a FULL OUTER JOIN

Oops I had a couple of typos. I meant to say I use a FULL JOIN and the question.

You have two sets

of customers and you want to find out which states have customers in one set of data but not the other.

2/13/2008 8:28 PM | [Regina](#)



re: Better Alternatives to a FULL OUTER JOIN

As the article explains, just use a UNION. However, I must tell you that it looks like you have a bad data model if you have tables like Customers_2007 and Customers_2006.

Anyway, all you need is something like this:

```
select state, max([In2007]) as [In2007], max([In2006]) as [In2006]
from
(
select state, 1 as [In2007], 0 as [In2006]
from customers_2007
union all
select state 0,1
```

from customers_2006
) x
group by state

2/13/2008 8:32 PM | [Jeff](#)



re: Better Alternatives to a FULL OUTER JOIN

Actually that was assuming some sort of table partitioning in place but customer was a bad example for partitioning (or using a subquery but I didn't want to needlessly complicate the example).

I think you are missing a HAVING CLAUSE and a comma to make your example complete. Fair enough.

2/13/2008 8:43 PM | [Regina](#)



re: Better Alternatives to a FULL OUTER JOIN

>>I think you are missing a HAVING CLAUSE and a comma to make your example complete.

Ah yes, that's exactly right! UNION ALL is so overlooked, it is an amazingly powerful way to combine tables. Combine a UNION ALL with a GROUP BY and a few aggregates of columns that are spread out, and you can do really amazing things. Very efficiently, too.

2/13/2008 8:59 PM | [Jeff](#)



re: Better Alternatives to a FULL OUTER JOIN

When doing audits on sensitive data, say campaign value, on which commission will be paid, we'll have a table
campaign_audit with value_before, value_after, who_dun_it, when_they_dun_it

Then you'll trigger out
Insert into the above

```
SELECT
d.value,
i.value,
SYSTEM_USER,
GETDATE()
FROM
inserted AS i
FULL OUTER JOIN
deleted AS d ON
i.tables_unique_identifier = d.tables_unique_identifier
```

obviously you'll have null values on the before and after on inserts and deletes respectively.

I'm not sure how a UNION would be a viable alternative.

2/15/2008 3:30 AM | [piersy](#)



re: Better Alternatives to a FULL OUTER JOIN

piersy -- I can't make heads or tails of what you are doing there-- a bad side effect of using FULL OUTER JOINS, of course. Apparently, you have written a SELECT that is expressing a JOIN between two tables that will NEVER match under any circumstance Why would you do this? If you really wanted to do a SELECT that gave you your resultset, isn't a UNION much clearer as to your intentions?

```
select id as inserted, null as deleted, system_user, getDate()  
from inserted  
union all  
select null as inserted, id as deleted, system_user, getDate()  
from deleted
```

Isn't that a lot easier to read and work with and understand? Better yet, wouldn't a separate SQL statement for INSERT and DELETE triggers make a lot more sense?

2/15/2008 1:19 PM | [Jeff](#)



re: Better Alternatives to a FULL OUTER JOIN

Sorry, it wasn't the clearest writing, hands up on that one.

It's done in a single trigger for INSERT DELETE UPDATE on one table, say for example:

```
campaign (campaign_id, sales_rep_id, client_id, campaign_value, ...)
```

Now the campaign value can be changed, but this can also be abused by staff to try and up their commission, so we need a log to track this

```
campaign_audit (campaign_id, value_before, value_after, who, when)
```

Thus joining on the id from inserted to deleted on campaign you will only not have the matching record on the insert or delete. You will always have a match on the update (well, as long as we don't go changing the unique identifier, but that wouldn't make much sense).

so in the above trigger we have

```
INSERT INTO  
campaign_audit  
(  
campaign_id,  
value_before,  
value_after
```

```
who,  
when  
)  
SELECT  
coalesce(i.campaign_id, d.campaign_id)  
d.campaign_value,  
i.campaign_value,  
SYSTEM_USER,  
GETDATE()  
FROM  
inserted AS i  
FULL OUTER JOIN deleted AS d ON  
i.campaign_id = d.campaign_id
```

So you can track changes, plus its obvious when you have a new record or a deletion.

Three seperate triggers for such a simple operation seems rather over the top to me.

Different triggers for different semantic operations, yes, but why split out a simple audit trigger into 3 just because they are insert update or delete when the above handles all 3 scenarios perfectly well? A UNION just makes no sense to me here as you're ending up with two records which would make for much more complicated code to handle.

2/18/2008 4:44 AM | [piersy](#)



re: Better Alternatives to a FULL OUTER JOIN

piersy -- OK, I think that helped to clear it up a little. Plus I forgot that for an UPDATE both the deleted and inserted table will have values. For a trigger like this, I suppose a FULL OUTER JOIN is fine -- we are not concerned with generating an efficient resultset, and the very nature of the query itself isn't a standard SELECT or JOIN in the traditional sense, it is simple "grab what you can and insert it into a table" statement. Ugly, but works well for simple tasks like this and is simple to write. Still logically tough to get a handle of, though.

The problem is: your audit table doesn't make a lot of sense. It is not clear when a row is deleted or just has a NULL value updated. Or when a row is added, or just has a NULL value changed to a non-NULL value. Those events look the same in your audit table.

I think it is cleaner and clearer to simply insert 3 possibilities to your audit table, with a clear "type" column to indicate what occurred: Update, delete or insert.

It all comes back to this: FULL OUTER JOIN == hard to read, generates odd results with nulls everywhere, hard to maintain. Well formed, clean, efficient SQL = none of those problems. Your example, while perhaps a viable use for FULL OUTER JOINS in your situation, only serves to strengthen my arguments about why to avoid them.

2/21/2008 12:48 PM | [Jeff](#)

re: Better Alternatives to a FULL OUTER JOIN



I'll take a yeah but no but any day :)

In all fairness these audit examples *are* pretty much the only examples of FULL OUTER JOIN in the entire codebase.

My rule of thumb is that there are a few things that are usually symptomatic of someone doing something wrong.

Cursors, DISTINCT* and FULL OUTER JOIN. While all of them have their uses, in 90% of cases where I find them they've been used as hack, a misunderstanding of the data, or because the writer didn't know of a better way of doing things.

*Yes DISTINCT can be very useful, but so often, so hacky...

2/22/2008 4:09 AM | [piersy](#)



re: Better Alternatives to a FULL OUTER JOIN

Well said, piersy, I definitely agree... thanks for your comments!

- Jeff

2/22/2008 8:08 AM | [Jeff](#)



re: Better Alternatives to a FULL OUTER JOIN

Hi Jeff,

I have two tables whose structures are like this:

TableA(Key1, Key2, Key3, Col4, Col5, Col6, SequenceNo)

```
1 1 1 c1114 c1115 c1116 1
1 1 2 c1124 c1125 c1126 2
1 1 3 c1134 c1135 c1136 3
1 1 5 c1154 c1155 c1156 4
1 1 6 c1164 c1165 c1166 5
```

TableB(Key1, Key2, Key3, ColA, ColB, SequenceNumber)

```
1 1 1 c111A c111B 1
1 1 3 c113A c113B 2
1 1 4 c114A c114B 3
1 1 6 c116A c116B 4
```

I need to get the result set like this.

```
Key1 key2 key3 col4 col5 col6 ColA ColB TableA.SequenceNo TableB.SequenceNo
1 1 1 c1114 c1115 c1116 C111A C111B 1 1
1 1 2 c1124 c1125 c1126 null null 2 null
1 1 3 c1134 c1135 c1136 C113A C113B 3 2
```

```
1 1 4 null null null C114A C114B null 3
1 1 5 c1154 c1155 c1156 null null 4 null
1 1 6 c1164 c1165 c1166 C116A C116B 5 4
```

How can you achieve this? Thanks

3/14/2008 6:17 PM | [Fred](#)



re: Better Alternatives to a FULL OUTER JOIN

By the way, I need to main the order just like above.

3/14/2008 6:27 PM | [Fred](#)



re: Better Alternatives to a FULL OUTER JOIN

looking for sql commands that work for SQL2005, SSCE3.0 and/or Ado.net
DataSet/DataTable

3/14/2008 6:47 PM | [Fred](#)



re: Better Alternatives to a FULL OUTER JOIN

Fred -- I recommended UNION in the article above -- did you try it?

I have no idea of your specifics, but just guessing, I would do it like this:

First, focus on getting a set of keys which will be your "driving" row source, or the "FROM" clause in your query:

```
select key1,key2,key3
from tableA
union
select key1,key2,key3
from TableB
```

From there, you just left outer join to each table:

```
select *
from
(the above SQL here ) x
left outer join TableA on ...
left outer join TableB on ...
```

you could also just union the two tables together, group on the KEY columns, and take the MAX() of the others -- filling out missing columns with NULL.

If this doesn't help you figure it out, don't post further questions here -- post a clear, complete question with your DDL at the SqlTeam forums.



re: Better Alternatives to a FULL OUTER JOIN

Thanx for you good article

I have mdb file include 2 tables and 3 queries , i will give you the file , I want from you to convert my queries to full outer join method..

Ok?

Sorry for my bad english

8/15/2008 10:41 AM | [akmalkady](#)



re: Better Alternatives to a FULL OUTER JOIN

Thanx for you good article

I have mdb file include 2 tables and 3 queries , i will give you the file , I want from you to convert my queries to full outer join method..

Ok?

Sorry for my bad english

8/15/2008 10:41 AM | [akmalkady](#)



re: Better Alternatives to a FULL OUTER JOIN

I have an example. We have an ancient text (dead language) in poor condition for which we created high res images under various lighting conditions (UV infra red...), which we then subdivided into arbitrary chapter and verse (due to image size, each verse is a separate image). We contracted with two experts to transliterate the text on the images. We loaded each experts results into separate tables, and calculated checksums for each transliterated verse.

Since the text is so large, we only wish to examine results where the experts disagreed, so that we can assemble a team to resolve the differences. If an expert could make out no text at all, that chapter and verse were left out when the data was returned.

The set of differences is the following. All chapter verses found by expert 1 which were not found by expert 2, all verses found by expert 2 which were not found by expert one, and all verses for which the experts both found, but disagreed on transliteration.

Below is the query used to return the set of differences using FULL OUTER JOIN, and then using your UNION ALL solution.

The FULL OUTER JOIN looks cleaner to me, and performed better.

```

SET NOCOUNT ON
GO
SET STATISTICS IO ON
GO
IF OBJECT_ID('tempdb.dbo.#ancienttext1') IS NOT NULL DROP table #ancienttext1
IF OBJECT_ID('tempdb.dbo.#ancienttext2') IS NOT NULL DROP table #ancienttext2

CREATE TABLE #ancienttext1 (Chapter int, Verse int, Versechecksum int)
CREATE TABLE #ancienttext2 (Chapter int, Verse int, Versechecksum int)

INSERT #ancienttext1 values (1, 1, 23)
INSERT #ancienttext1 values (1, 2, 2322)
INSERT #ancienttext1 values (1, 3, 2323)
INSERT #ancienttext1 values (1, 4, 2323)

INSERT #ancienttext2 values (1, 1, 23)
INSERT #ancienttext2 values (1, 2, 2322)
INSERT #ancienttext2 values (1, 3, 232)
INSERT #ancienttext2 values (2, 3, 2323)

CREATE UNIQUE CLUSTERED INDEX PK_T1
ON #ancienttext1
(
Chapter,
Verse
)

CREATE UNIQUE CLUSTERED INDEX PK_T2
ON #ancienttext2
(
Chapter,
Verse
)

SELECT
T1.*,
T2.*
FROM
#ancienttext1 T1

FULL OUTER JOIN #ancienttext2 T2
ON T1.Chapter = T2.Chapter
AND T1.Verse = T2.Verse
WHERE
(T1.chapter IS NULL)
OR
(T2.Chapter IS NULL)
OR
(T1.Versechecksum <> T2.VerseChecksum)

SELECT 7
SELECT 8

```

```

SELECT
T1.*
FROM
#ancienttext1 T1

LEFT OUTER JOIN #ancienttext2 T2
ON T1.Chapter = T2.Chapter
AND T1.Verse = T2.Verse
WHERE
(T2.Chapter IS NULL)

UNION ALL

SELECT
T2.*
FROM
#ancienttext2 T2

LEFT JOIN #ancienttext1 T1
ON T1.Chapter = T2.Chapter
AND T1.Verse = T2.Verse
WHERE
(T1.chapter IS NULL)

UNION ALL

SELECT
T1.*
FROM
#ancienttext1 T1

INNER JOIN #ancienttext2 T2
ON T1.Chapter = T2.Chapter
AND T1.Verse = T2.Verse
AND T1.Versechecksum <> T2.VerseChecksum

```

8/15/2008 4:55 PM | [Ben Tennen](#)



re: Better Alternatives to a FULL OUTER JOIN

Jeff,

Forgive the SELECT 7 SELECT 8 in my example above...they are just dummy SELECTs to more cleanly separate reads when SET STATISTICS IO ON is used.

Thanks for the blog...
Ben

8/15/2008 4:58 PM | [Ben Tennen](#)

re: Better Alternatives to a FULL OUTER JOIN



I have mdb file include 2 tables and 3 queries , i will give you the file , I want from you to convert my queries to full outer join method..

this is the file .. click at the link to download the database file

<http://www.arabteam2000-forum.com/index.php?act=attach&type=post&id=71194>

8/27/2008 12:16 PM | [akmalkady](#)



re: Better Alternatives to a FULL OUTER JOIN

akmalkady -- Do I have to?

8/27/2008 1:49 PM | [jeff](#)



re: Better Alternatives to a FULL OUTER JOIN

Just came across this page. I have a scenario where full outer join seems to be the best fit. Basically need to do a self-join to determine some changed columns. Here's an example (not complete, w/o constraints etc):

```
CREATE TABLE TestStatus (RunID integer NOT NULL, BugID integer NOT NULL, BugStatus char(20) NOT NULL)
```

Here BugStatus can be, for example, one of new/closed/reopened.

The requirement is to compare two test runs to see what bugs have changed between two runs, including newly opened, no longer reproducing etc. Here is an example query using full outer join (shortened for clarity):

```
select <runID-new> as RunID, (case when test1.BugID is null 'Not reproducible' else 'New bug' end) as BugStatus
from TestStatus as test1 full outer join TestStatus as test2
on (test1.BugID = test2.BugID and test1.RunID = <runID-new> and test2.RunID = <runID-previous>)
where (test1.BugID is null or test2.BugID is null)
```

Here <runID-new> and <runID-previous> are the two runs being compared. I can see that it can be done using a union of two left outer join queries or exists but that does not seem to be very efficient.

9/30/2008 8:59 AM | [sumwale](#)



re: Better Alternatives to a FULL OUTER JOIN

Just came across this page. I have a scenario where full outer join seems to be the best fit. Basically need to do a self-join to determine some changed columns. Here's an example (not complete, w/o constraints etc):

```
CREATE TABLE TestStatus (RunID integer NOT NULL, BugID integer NOT NULL, BugStatus char(20) NOT NULL)
```

Here BugStatus can be, for example, one of new/closed/reopened. The requirement is to compare two test runs to see what bugs have changed between two runs, including newly opened, no longer reproducing etc. Here is an example query using full outer join (shortened for clarity):

```
select <runID-new> as RunID, (case when test1.BugID is null 'Not reproducible' else 'New bug' end) as BugStatus
from TestStatus as test1 full outer join TestStatus as test2
on (test1.BugID = test2.BugID and test1.RunID = <runID-new> and test2.RunID = <runID-previous>)
where test1.BugID is null or test2.BugID is null
```

Here <runID-new> and <runID-previous> are the two runs being compared. I can see that it can be done using a union of two left outer join queries or exists but that does not seem to be very efficient.

9/30/2008 9:01 AM | [sumwale](#)



re: Better Alternatives to a FULL OUTER JOIN

sumwale -- first, you don't need a full outer join at all. You can use an INNER JOIN. You are comparing two runs, so they both should exist in the table, correct?

Even then, you don't need a join or a union, just GROUP BY, since you are comparing two things in the same table.

```
select
@RunNew as NewRun, @RunPrevious as PreviousRun,
case when min(BugStatus)=Max(BugStatus) then 'New bug' else 'Not reproducible'
end as BugStatus
from TestStatus
where runID in (@runNew, @runPrevious)
```

... or something like that. Absolutely no need for a FULL OUTER JOIN. Again, I had to stare at your FOJ for a few minutes just to figure out what you are trying to return. Here it is simple -- get two rows from the table for two different runs, and determine if the BugStatus is the same for both.

9/30/2008 9:17 AM | [jeff](#)



re: Better Alternatives to a FULL OUTER JOIN

Ben Tennen -- please read the article. this exact situation is demonstrated.

9/30/2008 9:18 AM | [jeff](#)



re: Better Alternatives to a FULL OUTER JOIN

Thanks. Not sure how it can be done using an INNER JOIN. You are right that GROUP BY can get it done with something like (not clear how your query will work):

```
SELECT @runNew AS NewRun, @runPrevious as PreviousRun, BugID, (CASE WHEN
MAX(RunID) = @runPrevious THEN 'Not reproducible' ELSE 'New bug' END) AS
BugStatus
FROM TestStatus
WHERE RunID IN (@runNew, @runPrevious)
GROUP BY BugID
HAVING COUNT(*) = 1
```

Is there a way to get rid of the redundant MAX(RunID)?

9/30/2008 10:20 AM | [sumwale](#)



re: Better Alternatives to a FULL OUTER JOIN

sumwale -- this would be a lot easier if you post some sample data and expected results. I am not following your logic of what you are trying to return.

9/30/2008 10:22 AM | [jeff](#)



re: Better Alternatives to a FULL OUTER JOIN

For data like:

```
INSERT INTO TestStatus SELECT 1, 100, 'New'
INSERT INTO TestStatus SELECT 1, 101, 'New'
INSERT INTO TestStatus SELECT 1, 102, 'New'
INSERT INTO TestStatus SELECT 2, 101, 'New'
INSERT INTO TestStatus SELECT 2, 103, 'New'
```

the expected output is:

```
NewRun PreviousRun BugID BugStatus
2 1 100 Not reproducible
2 1 102 Not reproducible
2 1 103 New bug
```

i.e. no output for bug #101 since it has remained unchanged for the two runs while others are included with appropriate status

9/30/2008 10:27 AM | [sumwale](#)



re: Better Alternatives to a FULL OUTER JOIN

Ah, the requirements make a little more sense now. Something like this is all you need:

```
declare @TestStatus table (RunID int, BugID int, BugStatus char(20))
```

```
INSERT INTO @TestStatus SELECT 1, 100, 'New'
```

```
INSERT INTO @TestStatus SELECT 1, 101, 'New'
INSERT INTO @TestStatus SELECT 1, 102, 'New'
INSERT INTO @TestStatus SELECT 2, 101, 'New'
INSERT INTO @TestStatus SELECT 2, 103, 'New'
```

```
declare @NewRun int, @PrevRun int
```

```
set @NewRun = 1
set @PrevRun = 2
```

```
select BugID, case when max(runID)=@newRun then 'New Bug' else 'Not reproducible'
end as Status
from @TestStatus
where ruNID in (@newRun, @PrevRun)
group by BugID
having count(*)=1
```

9/30/2008 10:55 AM | [jeff](#)



re: Better Alternatives to a FULL OUTER JOIN

... which is pretty much what you previously posted. So, what's wrong with the "redundant" max(runID)? Why would you want to get rid of it? It's not redundant or any issue/problem in any way, it's simply an aggregate calculation you use to help determine your result.

9/30/2008 10:57 AM | [jeff](#)



re: Better Alternatives to a FULL OUTER JOIN

Looks kind of redundant to me since it will always act on exactly one value, but will have to stay for group by (a distinct kind of aggregate would probably have made more logical sense).

9/30/2008 11:20 AM | [sumwale](#)



re: Better Alternatives to a FULL OUTER JOIN

sumwale -- not redundant, just arbitrary -- you can use MIN() or MAX(), no difference. Either way, it is necessary to aggregate the value since we don't want to GROUP on it. It's simply a logic issue; any non-grouped columns in a SQL statement must be aggregated. (see [this](#) for more on that if that doesn't make sense)

9/30/2008 11:22 AM | [jeff](#)



re: Better Alternatives to a FULL OUTER JOIN

I understand; maybe looking for a distinct() kind of aggregate (have had other occasions where it would have been useful).



re: Better Alternatives to a FULL OUTER JOIN

To clarify: what I mean is an aggregate that returns either the value if all values in a group are same, else returns a default value (e.g. null).

9/30/2008 11:47 AM | [sumwale](#)



re: Better Alternatives to a FULL OUTER JOIN

I have a full outer join query and it is slowing my job down, maybe some of you can help

I have two tables, let's say Tbl1 and Tbl2 which have two columns each, Col A being common for both and Col B and C the other columns which would look like these.

Tbl1
Col A Col B
123 A
123 B

Tbl2
Col A Col C
123 0
123 1

Would Union All be able to create my expected result which should be

Col A Col B Col C
123 A 0
123 A 1
123 B 0
123 B 1

10/1/2008 2:30 AM | [Vincent](#)



re: Better Alternatives to a FULL OUTER JOIN

Here's an example where a Full Outer Join gave me the results I needed and the Union did not. Perhaps I did not get the Union syntax correct, but it seems to me that this is a reason for a Full Outer Join:

A little background: The two views we are pulling data from are from two different applications, running on different servers. One app stores images, the other stores exam information. The exam information system has a table that links exams to images on the image storage system, however the image storage system has been in use for years longer than the exam info system, so has many records that do not have a cooresponding record in the exam info system. I have two views that pull together some basic information from each system (PatientID, PatientName, ApptDate, ExamID, Description and StudyUID) StudyUID is the field used to link the

two systems.

If I use a union as you suggest:

```
Select PatientID, PatientName, ApptDate, ExamID, Description, StudyUID)
From
(
Select PatientID, PatientName, ApptDate, ExamID, Description, StudyUID
From vwRISStudies where PatientID = '12345'
Union All
Select PatientID, PatientName, ApptDate, ExamID, Description, StudyUID
From vwPACSStudies where PatientID = '12345'
) x
```

I receive a result set with 20 records, some of which are in both systems, some of which are only in the Image system. Those that are in both systems are duplicated.

If on the other hand I run this query:

```
Select
isnull(ris.PatientID, pacs.PatientID) as PatientID,
isnull(ris.PatientName, pacs.PatientName) as PatientName,
isnull(ris.ApptDate, pacs.ApptDate) as ApptDate,
isnull(ris.ExamID, pacs.ExamID) as ExamID,
isnull(ris.Description, pacs.Description) as Description,
isnull(ris.StudyUID, pacs.StudyUID) as StudyUID
From vwRISStudies ris Full Outer Join
vwPACSStudies pacs on pacs.StudyUID = ris.StudyUID
Where isnull(ris.PatientID, '12345') = '12345' and isnull(pacs.PatientID, '12345') = '12345'
```

I get 14 records 7 of which reside in both systems and 7 that are older records that only exist in the Image System. Those that exist in both systems are not repeated.

I realize using all of the isnulls makes this harder to read, but I feel that is worth the issue of not having to filter out repeated records. I could of course use grouping to try to handle the repeated records, but since the Name and Description formatting is not the same between the systems it would make grouping difficult and also make for some ugly code.

All of this being said, this is the first example of a Full Outer Join that I have ever had to use in many years of SQL work. Which is why I was searching for help with Full Outer Joins and found this article.

3/2/2009 1:22 PM | [Neal](#)



re: Better Alternatives to a FULL OUTER JOIN

Awesome!

First clear solution that I've seen. Very helpful indeed. Thanks!



re: Better Alternatives to a FULL OUTER JOIN

Hi Jeff.

My problem can be stated as this. I need to compare literals in files against those stored in a translation database to identify new, matched, case mismatch, punctuation and obsolete literals.

Logically (well to me at least) a full outer join will return the result set we want as the null values will indicate which literals are new or obsolete and the matched, case mismatch and punctuation can be found by comparison.

My understanding of UNION [ALL] though is that both queries in the union need to have the same number/type of fields.
In my case, the source table/view contain different columns beyond the query keys.

I'd agree that you could do this job using UNION [ALL] but would you not have to create dummy columns to even up the tables/views? The FULL OUTER JOIN would do this for you.

Thanks for the assist
Peter

9/1/2009 12:02 AM | [Peter](#)



re: Better Alternatives to a FULL OUTER JOIN

I've just come across this article, having only recently discovered FULL OUTER JOIN. Personally, I think it's magic, but then I am no SQL expert by a long chalk.

Here's the situation I use it in. For years I have been generating lists of things on the AS/400 platform. The DB2/400 database is integrated into the system so if you ask for a list of anything, e.g. programs, users or database tables you get a real database table with the results. Very often I need to do a difference check between two different systems. Generate the lists on each system, send one over to the other and compare them.

For years I was doing three queries: What's on A but not on B, what's on B but not on A and, usually, what's on A and B but with different attributes. A FULL OUTER JOIN accomplishes the first two in one go with no fuss. The third can be managed with a little care in the same query.

Granted this scenario is not in an application and therefore performance is not an issue. I'm running these typically on an interactive SQL command line and looking directly at the results. FOJ FTW as far as I am concerned.

8/15/2010 7:23 PM | [Allister](#)

