# Parameterize an SQL IN clause

1000

381

How do I parameterize a query containing an `IN` clause with a variable number of arguments, like this one?

```
SELECT * FROM Tags
WHERE Name IN ('ruby','rails','scruffy','rubyonrails')
ORDER BY Count DESC
```

In this query, the number of arguments could be anywhere from 1 to 5.

I would prefer not to use a dedicated stored procedure for this (or XML), but if there is some elegant way specific to SQL Server 2008, I am open to that.

sql sql-server parameters

edited Mar 26 '18 at 11:05
Nisarg
11.3k 5 23 41

asked Dec 3 '08 at 16:16
Jeff Atwood
46.7k 45 140 149

7 For MySQL, see MySQL Prepared statements with a variable size variable list. – outis Apr 4 '12 at 12:02

Similar: Passing array parameters to a stored procedure, PreparedStatement IN clause alternatives. – Vadzim Apr 29 at 22:33

## 39 Answers

1 2 next

Here's a quick-and-dirty technique I have used:

So here's the C# code:

```
string[] tags = new string[] { "ruby", "rails", "scruffy", "rubyonrails" };
const string cmdText = "select * from tags where '|' + @tags + '|' like '%|' + Name +
'|%'";

using (SqlCommand cmd = new SqlCommand(cmdText)) {
   cmd.Parameters.AddWithValue("@tags", string.Join("|", tags);
}
```

Two caveats:

- The performance is terrible. `LIKE "%...%"` queries are not indexed.
- Make sure you don't have any `|`, blank, or null tags or this won't work

There are other ways to accomplish this that some people may consider cleaner, so please keep reading.

edited Oct 26 '17 at 20:56
user8839064
17 3

answered Dec 3 '08 at 16:41
Joel Spolsky ♦
29.4k 16 77 96

111 That will be hella slow – Matt Rogish Dec 3 '08 at 16:43

9 Yes, this is a table scan. Great for 10 rows, lousy for 100,000. – Will Hartung Dec 3 '08 at 16:48

14 Make sure you test on tags that have pipes in them. – Joel Coehoorn Dec 3 '08 at 17:16

14 This doesn't even answer the question. Granted, it's easy to see where to add the parameters, but how can you accept this a solution if it doesn't even bother to parameterize the query? It only looks simpler than @Mark Brackett's because it isn't parameterized. – tvanfosson Dec 3 '08 at 20:14

19 What if your tag is 'ruby|rails'. It will match, which will be wrong. When you roll out such solutions, you need to either make sure tags do not contain pipes, or explicitly filter them out: select * from Tags where '|ruby|rails|scruffy|rubyonrails|' like '%|' + Name + '|%' AND name not like '%!%' – A-K Aug 19 '09 at 22:21

You can parameterize *each* value, so something like:

```
    (s, i) => "@tag" + i.ToString()
).ToArray();

string inClause = string.Join(", ", paramNames);
using (SqlCommand cmd = new SqlCommand(string.Format(cmdText, inClause))) {
    for(int i = 0; i < paramNames.Length; i++) {
       cmd.Parameters.AddWithValue(paramNames[i], tags[i]);
    }
}
```

Which will give you:

```
cmd.CommandText = "SELECT * FROM Tags WHERE Name IN (@tag0, @tag1, @tag2, @tag3)"
cmd.Parameters["@tag0"] = "ruby"
cmd.Parameters["@tag1"] = "rails"
cmd.Parameters["@tag2"] = "scruffy"
cmd.Parameters["@tag3"] = "rubyonrails"
```

No, this is not open to SQL injection. The only injected text into CommandText is not based on user input. It's solely based on the hardcoded "@tag" prefix, and the index of an array. The index will *always* be an integer, is not user generated, and is safe.

The user inputted values are still stuffed into parameters, so there is no vulnerability there.

Edit:

> Injection concerns aside, take care to note that constructing the command text to accomodate a variable number of parameters (as above) impede's SQL server's ability to take advantage of cached queries. The net result is that you almost certainly lose the value of using parameters in the first place (as opposed to merely inserting the predicate strings into the SQL itself).

Not that cached query plans aren't valuable, but IMO this query isn't nearly complicated enough to see much benefit from it. While the compilation costs may approach (or even exceed) the execution costs, you're still talking milliseconds.

If you have enough RAM, I'd expect SQL Server would probably cache a plan for the common counts of parameters as well. I suppose you could always add five parameters, and let the unspecified tags be NULL - the query plan should be the same, but it seems pretty ugly to me and I'm not sure that it'd worth the micro-optimization (although, on Stack Overflow - it may very well be worth it).

**76.8k** 15 95 147

2 Basically the same as my answer to the "related" question and obviously the best solution since it is constructive and efficient rather than interpretive (much harder). – tvanfosson Dec 3 '08 at 16:53

47 This is how LINQ to SQL does it, BTW – Mark Cidade Dec 18 '08 at 18:55

3 @Pure: The whole point of this is to avoid SQL Injection, which you would be vulnerable to if you used dynamic SQL. – Ray Feb 4 '09 at 23:27

4 @God of Data - Yes, I suppose if you need more than 2100 tags you'll need a different solution. But Basarb's could only reach 2100 if the average tag length was < 3 chars (since you need a delimiter as well). msdn.microsoft.com/en-us/library/ms143432.aspx – Mark Brackett Feb 11 '10 at 12:17

2 @bonCodigo - your selected values are in an array; you just loop over the array and add a parameter (suffixed with the index) for each one. – Mark Brackett Jun 27 '14 at 13:42

---

242

For SQL Server 2008, you can use a table valued parameter. It's a bit of work, but it is arguably cleaner than my other method.

First, you have to create a type

```
CREATE TYPE dbo.TagNamesTableType AS TABLE ( Name nvarchar(50) )
```

Then, your ADO.NET code looks like this:

```
string[] tags = new string[] { "ruby", "rails", "scruffy", "rubyonrails" };
cmd.CommandText = "SELECT Tags.* FROM Tags JOIN @tagNames as P ON Tags.Name = P.Name";

// value must be IEnumerable<SqlDataRecord>
cmd.Parameters.AddWithValue("@tagNames", tags.AsSqlDataRecord("Name")).SqlDbType =
SqlDbType.Structured;
cmd.Parameters["@tagNames"].TypeName = "dbo.TagNamesTableType";

// Extension method for converting IEnumerable<string> to IEnumerable<SqlDataRecord>
public static IEnumerable<SqlDataRecord> AsSqlDataRecord(this IEnumerable<string>
values, string columnName) {
    if (values == null || !values.Any()) return null; // Annoying, but SqlClient wants
null instead of 0 rows
```

```
            return r;
        });
    }
```

edited May 23 '17 at 12:10
Community ♦
1 1

answered Dec 3 '08 at 16:53
Mark Brackett
76.8k 15 95 147

39 we tested this and table valued parameters are DOG slow. It is literally faster to execute 5 queries than it is to do one TVP. – Jeff Atwood Apr 4 '11 at 5:19

4 @JeffAtwood - Have you tried reshuffling the query to something like `SELECT * FROM tags WHERE tags.name IN (SELECT name from @tvp);` ? In theory, this really should be the fastest approach. You can use relevant indexes (e.g. an index on tag name that `INCLUDE` s count would be ideal), and SQL Server should be doing a few seeks to grab all the tags and their counts. What does the plan look like? – Nick Chammas Oct 13 '11 at 19:47

9 I've also tested this and it is FAST AS LIGHTNING (compared to constructing a large IN string). I had some problems setting the parameter though since I was constantly getting "Failed to convert parameter value from a Int32[] to a IEnumerable`1.". Anyway, solved that and here's a sample I made pastebin.com/qHP05CXc – Fredrik Johansson May 2 '13 at 13:49

6 @FredrikJohansson - Out of 130 upvotes, you may be the only run that's actually tried to run this! I made a mistake reading the docs, and you actually need an IEnumerable<SqlDataRecord>, not just any IEnumerable. Code has been updated. – Mark Brackett May 2 '13 at 18:17

3 @MarkBrackett Great with an update! Accually this code really saved the day for me since I'm quering a Lucene search-index and it sometimes returns more than 50.000 or so hits that need to be doublechecked against SQL server - So I create an array of int[] (document/SQL keys) and then the code above comes in. The whole OP now takes less than 200ms :) – Fredrik Johansson May 3 '13 at 6:57

182

The original question was **"How do I parameterize a query ..."**

Let me state right here, that this is **not an answer** to the original question. There are already some demonstrations of that in other good answers.

With that said, go ahead and flag this answer, downvote it, mark it as not an answer... do whatever you believe is right.

See the answer from Mark Brackett for the preferred answer that I (and 231 others) upvoted. The approach given in his answer allows 1) for effective use of bind variables, and 2) for predicates that are sargable.

Joel Spolsky's approach is clever. And it works reasonably, it's going to exhibit predictable behavior and predictable performance, given "normal" values, and with the normative edge cases, such as NULL and the empty string. And it may be sufficient for a particular application.

But in terms generalizing this approach, let's also consider the more obscure corner cases, like when the `Name` column contains a wildcard character (as recognized by the LIKE predicate.) The wildcard character I see most commonly used is `%` (a percent sign.). So let's deal with that here now, and later go on to other cases.

**Some problems with % character**

Consider a Name value of `'pe%ter'`. (For the examples here, I use a literal string value in place of the column name.) A row with a Name value of `'pe%ter' would be returned by a query of the form:

```
select ...
 where '|peanut|butter|' like '%|' + 'pe%ter' + '|%'
```

But that same row will **not** be returned if the order of the search terms is reversed:

```
select ...
 where '|butter|peanut|' like '%|' + 'pe%ter' + '|%'
```

The behavior we observe is kind of odd. Changing the order of the search terms in the list changes the result set.

It almost goes without saying that we might not want `pe%ter` to match peanut butter, no matter how much he likes it.

**Obscure corner case**

(Yes, I will agree that this is an obscure case. Probably one that is not likely to be tested. We wouldn't expect a wildcard in a column value. We may assume that the application prevents such a value from being stored. But in my experience, I've rarely seen a database constraint that specifically disallowed characters or patterns that would be considered wildcards on the right side of a `LIKE` comparison operator.

**Patching a hole**

One approach to patching this hole is to escape the `%` wildcard character. (For anyone not familiar with the escape clause on the

Now we can match the literal %. Of course, when we have a column name, we're going to need to dynamically escape the wildcard. We can use the `REPLACE` function to find occurrences of the `%` character and insert a backslash character in front of each one, like this:

```
select ...
 where '|pe%ter|'
  like '%|' + REPLACE( 'pe%ter' ,'%','\%') + '|%' escape '\'
```

So that solves the problem with the % wildcard. Almost.

### Escape the escape

We recognize that our solution has introduced another problem. The escape character. We see that we're also going to need to escape any occurrences of escape character itself. This time, we use the ! as the escape character:

```
select ...
 where '|pe%t!r|'
  like '%|' + REPLACE(REPLACE( 'pe%t!r' ,'!','!!'),'%','!%') + '|%' escape '!'
```

### The underscore too

Now that we're on a roll, we can add another `REPLACE` handle the underscore wildcard. And just for fun, this time, we'll use $ as the escape character.

```
select ...
 where '|p_%t!r|'
  like '%|' + REPLACE(REPLACE(REPLACE( 'p_%t!r' ,'$','$$'),'%','$%'),'_','$_') + '|%'
escape '$'
```

I prefer this approach to escaping because it works in Oracle and MySQL as well as SQL Server. (I usually use the \ backslash as the escape character, since that's the character we use in regular expressions. But why be constrained by convention!

### Those pesky brackets

SQL Server also allows for wildcard characters to be treated as literals by enclosing them in brackets `[]`. So we're not done fixing yet.

Finding matching pairs of brackets shouldn't be that hard. It's a little more difficult than handling the occurrences of singleton % and _. (Note that it's not sufficient to just escape all occurrences of brackets, because a singleton bracket is considered to be a literal, and doesn't need to be escaped. The logic is getting a little fuzzier than I can handle without running more test cases.)

**Inline expression gets messy**

That inline expression in the SQL is getting longer and uglier. We can probably make it work, but heaven help the poor soul that comes behind and has to decipher it. As much of a fan I am for inline expressions, I'm inclined not use one here, mainly because I don't want to have to leave a comment explaining the reason for the mess, and apologizing for it.

**A function where ?**

Okay, so, if we don't handle that as an inline expression in the SQL, the closest alternative we have is a user defined function. And we know that won't speed things up any (unless we can define an index on it, like we could with Oracle.) If we've got to create a function, we might better do that in the code that calls the SQL statement.

And that function may have some differences in behavior, dependent on the DBMS and version. (A shout out to all you Java developers so keen on being able to use any database engine interchangeably.)

**Domain knowledge**

We may have specialized knowledge of the domain for the column, (that is, the set of allowable values enforced for the column. We may know *a priori* that the values stored in the column will never contain a percent sign, an underscore, or bracket pairs. In that case, we just include a quick comment that those cases are covered.

The values stored in the column may allow for % or _ characters, but a constraint may require those values to be escaped, perhaps using a defined character, such that the values are LIKE comparison "safe". Again, a quick comment about the allowed set of values, and in particular which character is used as an escape character, and go with Joel Spolsky's approach.

But, absent the specialized knowledge and a guarantee, it's important for us to at least consider handling those obscure corner cases, and consider whether the behavior is reasonable and "per the specification".

**Other issues recapitulated**

I believe others have already sufficiently pointed out some of the other commonly considered areas of concern:

- optimizer plan using index scan rather than index seeks, possible need for an expression or function for escaping wildcards (possible index on expression or function)
- using literal values in place of bind variables impacts scalability

**Conclusion**

I like Joel Spolsky's approach. It's clever. And it works.

But as soon as I saw it, I immediately saw a potential problem with it, and it's not my nature to let it slide. I don't mean to be critical of the efforts of others. I know many developers take their work very personally, because they invest so much into it and they care so much about it. So please understand, this is not a personal attack. What I'm identifying here is the type of problem that crops up in production rather than testing.

Yes, I've gone far afield from the original question. But where else to leave this note concerning what I consider to be an important issue with the "selected" answer for a question?

edited Aug 14 '15 at 20:25

Jean-François Savard
**17.8k** 5 34 63

answered May 29 '09 at 23:18

spencer7593
**88.3k** 13 85 98

can you please let us know if you use or like parameterized querys? in this particular case is it correct to jump over de rule of 'use parameterized querys' and sanitize with the original language? THANKS a lot – Luis Siquot Apr 19 '12 at 14:18

2 @Luis: yes, i prefer using bind variables in SQL statements, and will only avoid bind variables when using them causes a performance problem. my normative pattern for the original problem would be to dynamically create the SQL statement with the required number of placeholders in the IN list, and then bind each value to one of the placeholders. See the answer from Mark Brackett, which is the answer that I (and 231 others) upvoted. – spencer7593 Apr 23 '12 at 20:55

130

You can pass the parameter as a string

So you have the string

```
where Name in (SELECT item from fnSplit(@tags, '|'))
order by Count desc
```

Then all you have to do is pass the string as 1 parameter.

Here is the split function I use.

```
CREATE FUNCTION [dbo].[fnSplit](
    @sInputList VARCHAR(8000) -- List of delimited items
  , @sDelimiter VARCHAR(8000) = ',' -- delimiter that separates items
) RETURNS @List TABLE (item VARCHAR(8000))

BEGIN
DECLARE @sItem VARCHAR(8000)
WHILE CHARINDEX(@sDelimiter,@sInputList,0) <> 0
 BEGIN
 SELECT
  @sItem=RTRIM(LTRIM(SUBSTRING(@sInputList,1,CHARINDEX(@sDelimiter,@sInputList,0)-1))),

@sInputList=RTRIM(LTRIM(SUBSTRING(@sInputList,CHARINDEX(@sDelimiter,@sInputList,0)+LEN(@sD


 IF LEN(@sItem) > 0
  INSERT INTO @List SELECT @sItem
 END

IF LEN(@sInputList) > 0
 INSERT INTO @List SELECT @sInputList -- Put the last item in
RETURN
END
```

answered Dec 3 '08 at 16:27

David Basarab
**43.6k** 41 120 151

2 You can also join to the table-function with this approach. – Michael Haren Dec 4 '08 at 3:06

returned – adolf garlic Apr 1 '09 at 9:55

65

I heard Jeff/Joel talk about this on the podcast today (*episode 34*, 2008-12-16 (MP3, 31 MB), 1 h 03 min 38 secs - 1 h 06 min 45 secs), and I thought I recalled Stack Overflow was using LINQ to SQL, but maybe it was ditched. Here's the same thing in LINQ to SQL.

```
var inValues = new [] { "ruby","rails","scruffy","rubyonrails" };

var results = from tag in Tags
              where inValues.Contains(tag.Name)
              select tag;
```

That's it. And, yes, LINQ already looks backwards enough, but the `Contains` clause seems extra backwards to me. When I had to do a similar query for a project at work, I naturally tried to do this the wrong way by doing a join between the local array and the SQL Server table, figuring the LINQ to SQL translator would be smart enough to handle the translation somehow. It didn't, but it did provide an error message that was descriptive and pointed me towards using *Contains*.

Anyway, if you run this in the highly recommended LINQPad, and run this query, you can view the actual SQL that the SQL LINQ provider generated. It'll show you each of the values getting parameterized into an `IN` clause.

edited Jan 8 '12 at 9:08
Peter Mortensen
**14.2k** 19 88 114

answered Dec 19 '08 at 5:40
Peter Meyer
**21.9k** 1 29 51

47

If you are calling from .NET, you could use Dapper dot net:

```
string[] names = new string[] {"ruby","rails","scruffy","rubyonrails"};
var tags = dataContext.Query<Tags>(@"
select * from Tags
where Name in @names
order by Count desc", new {names});
```

```
            orderby tag.Count descending
            select tag;
```

edited Apr 21 '17 at 13:28

Sam Saffron
**91.4k** 69 287 481

answered Jun 15 '11 at 11:04

Marc Gravell ♦
**807k** 205 2195 2588

11 which happens to be what we use on this page, for the actual question asked (dapper) i.stack.imgur.com/RBAjL.png – Sam Saffron Jun 15 '11 at 11:09

3 Note that dapper now also supports Table Valued Parameters as first class citizens – Marc Gravell ♦ Jul 16 '14 at 16:48

This falls over if names is long – cs0815 Oct 20 '14 at 14:41

---

27

This is possibly a half nasty way of doing it, I used it once, was rather effective.

Depending on your goals it might be of use.

1. Create a *temp table* with one column.
2. `INSERT` each look-up value into that column.
3. Instead of using an `IN` , you can then just use your standard `JOIN` rules. ( Flexibility++ )

This has a bit of added flexibility in what you can do, but it's more suited for situations where you have a large table to query, with good indexing, and you want to use the parametrized list more than once. Saves having to execute it twice and have all the sanitation done manually.

I never got around to profiling exactly how *fast* it was, but in my situation it was needed.

edited May 31 '15 at 10:12

shA.t
**13.4k** 4 39 75

answered Dec 3 '08 at 17:04

Kent Fredric
**50.8k** 14 98 147

23

We have function that creates a table variable that you can join to:

```
ALTER FUNCTION [dbo].[Fn_sqllist_to_table](@list  AS VARCHAR(8000),
                                          @delim AS VARCHAR(10))
RETURNS @listTable TABLE(
  Position INT,
  Value    VARCHAR(8000))
AS
  BEGIN
      DECLARE @myPos INT

      SET @myPos = 1

      WHILE Charindex(@delim, @list) > 0
        BEGIN
            INSERT INTO @listTable
                        (Position,Value)
            VALUES     (@myPos,LEFT(@list, Charindex(@delim, @list) - 1))

            SET @myPos = @myPos + 1

            IF Charindex(@delim, @list) = Len(@list)
              INSERT INTO @listTable
                          (Position,Value)
              VALUES     (@myPos,'')

            SET @list = RIGHT(@list, Len(@list) - Charindex(@delim, @list))
        END

      IF Len(@list) > 0
        INSERT INTO @listTable
                    (Position,Value)
        VALUES     (@myPos,@list)

      RETURN
  END
```

So:

```
@Name varchar(8000) = null // parameter for search values
```

edited Dec 21 '14 at 14:37

Pரதீப்
**76.9k** 12 84 122

answered Dec 3 '08 at 17:11

David Robbins
**7,861** 7 45 79

---

18

This is gross, but if you are guaranteed to have at least one, you could do:

```
SELECT ...
       ...
 WHERE tag IN( @tag1, ISNULL( @tag2, @tag1 ), ISNULL( @tag3, @tag1 ), etc. )
```

Having IN( 'tag1', 'tag2', 'tag1', 'tag1', 'tag1' ) will be easily optimized away by SQL Server. Plus, you get direct index seeks

edited Sep 22 '10 at 21:20

answered Dec 3 '08 at 16:31

Matt Rogish
**18.8k** 10 69 90

1 Optional parameters with Null checks spoil performance, since the optimizer requires the number of parameters used to create efficient queries. A query for 5 parameters may need a different query plan than one for 500 parameters. – Erik Hart Jan 11 '14 at 15:33

---

17

In my opinion, the best source to solve this problem, is what has been posted on this site:

Syscomments. Dinakar Nethi

```
CREATE FUNCTION dbo.fnParseArray (@Array VARCHAR(1000),@separator CHAR(1))
RETURNS @T Table (col1 varchar(50))
AS
BEGIN
 --DECLARE @T Table (col1 varchar(50))
 -- @Array is the array we wish to parse
 -- @Separator is the separator charactor such as a comma
 DECLARE @separator_position INT -- This is used to locate each separator character
 DECLARE @array_value VARCHAR(1000)     this holds each array value as it is returned
```

```
 WHILE PATINDEX('%' + @separator + '%', @array) <> 0
 BEGIN
    -- patindex matches the a pattern against a string
    SELECT @separator_position = PATINDEX('%' + @separator + '%',@array)
    SELECT @array_value = LEFT(@array, @separator_position - 1)
    -- This is where you process the values passed.
    INSERT into @T VALUES (@array_value)
    -- Replace this select statement with your processing
    -- @array_value holds the value of this element of the array
    -- This replaces what we just processed with and empty string
    SELECT @array = STUFF(@array, 1, @separator_position, '')
 END
 RETURN
END
```

Use:

```
SELECT * FROM dbo.fnParseArray('a,b,c,d,e,f', ',')
```

**CREDITS FOR: Dinakar Nethi**

answered Jul 22 '10 at 14:47

Paulo Henrique

**657** 2 18 30

Great answer, clean and modular, super fast execution except for the initial CSV parsing into a table (one time, small number of elements). Although could use simpler/faster charindex() instead of patindex()? Charindex() also allows argument 'start_location' which may be able to avoid chopping input string each iter? To answer the original question can just join with function result. – crokusek Dec 14 '11 at 0:37

17

I would pass a table type parameter (since it's SQL Server 2008), and do a `where exists` , or inner join. You may also use XML, using `sp_xml_preparedocument` , and then even index that temporary table.

edited Jan 13 '12 at 20:59

Peter Mortensen

answered Dec 3 '08 at 16:30

eulerfx

15

In `SQL Server 2016+` you could use `STRING_SPLIT` function:

```
DECLARE @names NVARCHAR(MAX) = 'ruby,rails,scruffy,rubyonrails';

SELECT *
FROM Tags
WHERE Name IN (SELECT [value] FROM STRING_SPLIT(@names, ','))
ORDER BY Count DESC;
```

or:

```
DECLARE @names NVARCHAR(MAX) = 'ruby,rails,scruffy,rubyonrails';

SELECT t.*
FROM Tags t
JOIN STRING_SPLIT(@names,',')
  ON t.Name = [value]
ORDER BY Count DESC;
```

LiveDemo

The accepted answer will of course work and it is one of the way to go, but it is anti-pattern.

> **E. Find rows by list of values**
>
> This is replacement for common anti-pattern such as creating a dynamic SQL string in application layer or Transact-SQL, or by using LIKE operator:

```
SELECT ProductId, Name, Tags
FROM Product
WHERE ',1,2,3,' LIKE '%,' + CAST(ProductId AS VARCHAR(20)) + ',%';
```

11

The proper way IMHO is to store the list in a character string (limited in length by what the DBMS support); the only trick is that (in order to simplify processing) I have a separator (a comma in my example) at the beginning and at the end of the string. The idea is to "normalize on the fly", turning the list into a one-column table that contains one row per value. This allows you to turn

> in (ct1,ct2, ct3 ... ctn)

into an

> in (select ...)

or (the solution I'd probably prefer) a regular join, if you just add a "distinct" to avoid problems with duplicate values in the list.

Unfortunately, the techniques to slice a string are fairly product-specific. Here is the SQL Server version:

```
with qry(n, names) as
      (select len(list.names) - len(replace(list.names, ',', '')) - 1 as n,
              substring(list.names, 2, len(list.names)) as names
       from (select ',Doc,Grumpy,Happy,Sneezy,Bashful,Sleepy,Dopey,' names) as list
       union all
       select (n - 1) as n,
              substring(names, 1 + charindex(',', names), len(names)) as names
       from qry
       where n > 1)
select n, substring(names, 1, charindex(',', names) - 1) dwarf
from qry;
```

The Oracle version:

```
select n, substr(name, 1, instr(name, ',') - 1) dwarf
from (select n,
             substr(val, 1 + instr(val, ',', 1, n)) name
     from (select rownum as n,
                  list.val
```

and the MySQL version:

```
select pivot.n,
      substring_index(substring_index(list.val, ',', 1 + pivot.n), ',', -1) from (select
1 as n
     union all
     select 2 as n
     union all
     select 3 as n
     union all
     select 4 as n
     union all
     select 5 as n
     union all
     select 6 as n
     union all
     select 7 as n
     union all
     select 8 as n
     union all
     select 9 as n
     union all
     select 10 as n) pivot,    (select ',Doc,Grumpy,Happy,Sneezy,Bashful,Sleepy,Dopey,'
val) as list where pivot.n <  length(list.val) -
                  length(replace(list.val, ',', ''));
```

(Of course, "pivot" must return as many rows as the maximum number of items we can find in the list)

edited Feb 4 '09 at 18:54 — answered Feb 4 '09 at 18:51

Jeff Atwood
**46.7k** 45 140 149

SFA

---

10

If you've got SQL Server 2008 or later I'd use a Table Valued Parameter.

If you're unlucky enough to be stuck on SQL Server 2005 you could add a CLR function like this,

```
    TableDefinintion = "s NVARCHAR(MAX)"]
public static IEnumerable Split(SqlChars seperator, SqlString s)
{
    if (s.IsNull)
        return new string[0];

    return s.ToString().Split(seperator.Buffer);
}

public static void SplitFillRow(object row, out SqlString s)
{
    s = new SqlString(row.ToString());
}
```

Which you could use like this,

```
declare @desiredTags nvarchar(MAX);
set @desiredTags = 'ruby,rails,scruffy,rubyonrails';

select * from Tags
where Name in [dbo].[Split] (',', @desiredTags)
order by Count desc
```

edited May 23 '17 at 11:54
Community ♦
1 1

answered Aug 15 '12 at 16:32
Jodrell
27.3k 3 59 104

---

9

I think this is a case when a static query is just not the way to go. Dynamically build the list for your in clause, escape your single quotes, and dynamically build SQL. In this case you probably won't see much of a difference with any method due to the small list, but the most efficient method really is to send the SQL exactly as it is written in your post. I think it is a good habit to write it the most efficient way, rather than to do what makes the prettiest code, or consider it bad practice to dynamically build SQL.

I have seen the split functions take longer to execute than the query themselves in many cases where the parameters get large. A stored procedure with table valued parameters in SQL 2008 is the only other option I would consider, although this will probably be slower in your case. TVP will probably only be faster for large lists if you are searching on the primary key of the TVP, because SQL will

make a noticeable difference. I highly recommend against having NULL in your IN lists, as if that gets changed to a NOT IN it will not act as intended. You could dynamically build the parameter list, but the only obvious thing that you would gain is that the objects would escape the single quotes for you. That approach is also slightly slower on the application end since the objects have to parse the query to find the parameters. It may or may not be faster on SQL, as parameterized queries call sp_prepare, sp_execute for as many times you execute the query, followed by sp_unprepare.

The reuse of execution plans for stored procedures or parameterized queries may give you a performance gain, but it will lock you in to one execution plan determined by the first query that is executed. That may be less than ideal for subsequent queries in many cases. In your case, reuse of execution plans will probably be a plus, but it might not make any difference at all as the example is a really simple query.

**Cliffs notes:**

For your case anything you do, be it parameterization with a fixed number of items in the list (null if not used), dynamically building the query with or without parameters, or using stored procedures with table valued parameters will not make much of a difference. However, my general recommendations are as follows:

**Your case/simple queries with few parameters:**

Dynamic SQL, maybe with parameters if testing shows better performance.

**Queries with reusable execution plans, called multiple times by simply changing the parameters or if the query is complicated:**

SQL with dynamic parameters.

**Queries with large lists:**

Stored procedure with table valued parameters. If the list can vary by a large amount use WITH RECOMPILE on the stored procedure, or simply use dynamic SQL without parameters to generate a new execution plan for each query.

edited Jun 9 '10 at 20:34

answered Jun 9 '10 at 20:28

Scott

**91** 1 2

What do you mean by "stored procedure" here? Could you post an example? – struhtanov Apr 26 '13 at 9:55

9

```
declare @x xml
set @x='<items>
<item myvalue="29790" />
<item myvalue="31250" />
</items>
';
With CTE AS (
     SELECT
        x.item.value('@myvalue[1]', 'decimal') AS myvalue
    FROM @x.nodes('//items/item') AS x(item) )

select * from YourTable where tableColumnName in (select myvalue from cte)
```

answered Oct 24 '11 at 18:41



MindLoggedOut
**160** 1 5

1 `CTE` and `@x` can be eliminated/inlined into the subselect, if done very carefully, as shown in this article. – robert4 Aug 20 '15 at 3:12

I'd approach this by default with passing a table valued function (that returns a table from a string) to the IN condition.

9

Here is the code for the UDF *(I got it from Stack Overflow somewhere, i can't find the source right now)*

```
CREATE FUNCTION [dbo].[Split] (@sep char(1), @s varchar(8000))
RETURNS table
AS
RETURN (
    WITH Pieces(pn, start, stop) AS (
      SELECT 1, 1, CHARINDEX(@sep, @s)
      UNION ALL
      SELECT pn + 1, stop + 1, CHARINDEX(@sep, @s, stop + 1)
      FROM Pieces
      WHERE stop > 0
    )
    SELECT
```

```
select * from Tags
where Name in (select s from dbo.split(';','ruby;rails;scruffy;rubyonrails'))
order by Count desc
```

Unless you have a ridiculously long string, this should work well with the table index.

If needed you can insert it into a temp table, index it, then run a join...

edited Jun 29 '15 at 18:05

answered Jun 11 '15 at 15:16

Eli Ekstein
**411** 7 18

---

8

Another possible solution is instead of passing a variable number of arguments to a stored procedure, pass a single string containing the names you're after, but make them unique by surrounding them with '<>'. Then use PATINDEX to find the names:

```
SELECT *
FROM Tags
WHERE PATINDEX('%<' + Name + '>%','<jo>,<john>,<scruffy>,<rubyonrails>') > 0
```

edited Jan 8 '12 at 9:04

Peter Mortensen
**14.2k** 19 88 114

answered Feb 12 '10 at 19:22

ArtOfCoding
**121** 1 4

---

8

Use the following stored procedure. It uses a custom split function, which can be found here.

```
create stored procedure GetSearchMachingTagNames
   @PipeDelimitedTagNames varchar(max),
   @delimiter char(1)
   as
   begin
        select * from Tags
```


Peter Mortensen
**14.2k** 19 88 114


mangeshkt
**2,771** 1 13 8

---

7

For a variable number of arguments like this the only way I'm aware of is to either generate the SQL explicitly or do something that involves populating a temporary table with the items you want and joining against the temp table.

answered Dec 3 '08 at 16:31


ConcernedOfTunbridge Wells
**53.2k** 13 126 189

---

7

In ColdFusion we just do:

```
<cfset myvalues = "ruby|rails|scruffy|rubyonrails">
    <cfquery name="q">
        select * from sometable where values in <cfqueryparam value="#myvalues#"
list="true">
    </cfquery>
```

edited Jan 23 '10 at 16:53


Peter Mortensen
**14.2k** 19 88 114

answered Dec 10 '08 at 21:54


rip747
**6,814** 6 28 42

---

7

Here's a technique that recreates a local table to be used in a query string. Doing it this way eliminates all parsing problems.

The string can be built in any language. In this example I used SQL since that was the original problem I was trying to solve. I needed a clean way to pass in table data on the fly in a string to be executed later.

Using a user defined type is optional. Creating the type is only created once and can be done ahead of time. Otherwise just add a full table type to the declaration in the string.

```
-- Create a user defined type for the list.
CREATE TYPE [dbo].[StringList] AS TABLE(
    [StringValue] [nvarchar](max) NOT NULL
)

-- Create a sample list using the list table type.
DECLARE @list [dbo].[StringList];
INSERT INTO @list VALUES ('one'), ('two'), ('three'), ('four')

-- Build a string in which we recreate the list so we can pass it to exec
-- This can be done in any language since we're just building a string.
DECLARE @str nvarchar(max);
SET @str = 'DECLARE @list [dbo].[StringList]; INSERT INTO @list VALUES '

-- Add all the values we want to the string. This would be a loop in C++.
SELECT @str = @str + '(''' + StringValue + '''),' FROM @list

-- Remove the trailing comma so the query is valid sql.
SET @str = substring(@str, 1, len(@str)-1)

-- Add a select to test the string.
SET @str = @str + '; SELECT * FROM @list;'

-- Execute the string and see we've pass the table correctly.
EXEC(@str)
```

edited May 30 '12 at 0:16

answered May 29 '12 at 23:49

Rockfish

**109** 1 2

---

7

Here is another alternative. Just pass a comma-delimited list as a string parameter to the stored procedure and:

```
CREATE PROCEDURE [dbo].[sp_myproc]
    @UnitList varchar(MAX) = '1,2,3'
AS
select column from table
where ph.UnitID in (select * from CsvToInt(@UnitList))
```

```
CREATE Function [dbo].[CsvToInt] ( @Array varchar(MAX))
returns @IntTable table
(IntValue int)
AS
begin
    declare @separator char(1)
    set @separator = ','
    declare @separator_position int
    declare @array_value varchar(MAX)

    set @array = @array + ','

    while patindex('%,%' , @array) <> 0
    begin

        select @separator_position = patindex('%,%' , @array)
        select @array_value = left(@array, @separator_position - 1)

        Insert @IntTable
        Values (Cast(@array_value as int))
        select @array = stuff(@array, 1, @separator_position, '')
    end
    return
end
```

edited Aug 30 '16 at 14:11

answered Apr 6 '13 at 2:39

Metaphor

**3,408** 5 32 61

---

7

If we have strings stored inside the IN clause with the comma(,) delimited, we can use the charindex function to get the values. If you use .NET, then you can map with SqlParameters.

**DDL Script:**

```
CREATE TABLE Tags
    ([ID] int, [Name] varchar(20))
```

```
    (2, 'rails'),
    (3, 'scruffy'),
    (4, 'rubyonrails')
;
```

**T-SQL:**

```
DECLARE @Param nvarchar(max)

SET @Param = 'ruby,rails,scruffy,rubyonrails'

SELECT * FROM Tags
WHERE CharIndex(Name,@Param)>0
```

You can use the above statement in your .NET code and map the parameter with SqlParameter.

Fiddler demo

**EDIT:** Create the table called SelectedTags using the following script.

DDL Script:

```
Create table SelectedTags
(Name nvarchar(20));

INSERT INTO SelectedTags values ('ruby'),('rails')
```

T-SQL:

```
DECLARE @list nvarchar(max)
SELECT @list=coalesce(@list+',','')+st.Name FROM SelectedTags st

SELECT * FROM Tags
WHERE CharIndex(Name,@Param)>0
```

edited Oct 26 '17 at 22:05 answered Nov 23 '12 at 18:13

@JohnSaunders, I have edited the script without using any hardcoded list. Please verify. – Gowdhaman008 Dec 3 '12 at 13:32

3 One limitation with this option. CharIndex returns 1 if the string is found. IN returns a match for an exact terms. CharIndex for "Stack" will return 1 for a term "StackOverflow" IN will not. There is a minor tweek to this answer using PatIndex above that encloses names with '<' % name % '>' that overcomes this limitation. Creative solution to this problem though. – Richard Vivian May 17 '14 at 7:38

6

I have an answer that doesn't require a UDF, XML Because IN accepts a select statement e.g. SELECT * FROM Test where Data IN (SELECT Value FROM TABLE)

You really only need a way to convert the string into a table.

This can be done with a recursive CTE, or a query with a number table (or Master..spt_value)

Here's the CTE version.

```
DECLARE @InputString varchar(8000) = 'ruby,rails,scruffy,rubyonrails'

SELECT @InputString = @InputString + ','

;WITH RecursiveCSV(x,y)
AS
(
    SELECT
        x = SUBSTRING(@InputString,0,CHARINDEX(',',@InputString,0)),
        y = SUBSTRING(@InputString,CHARINDEX(',',@InputString,0)+1,LEN(@InputString))
    UNION ALL
    SELECT
        x = SUBSTRING(y,0,CHARINDEX(',',y,0)),
        y = SUBSTRING(y,CHARINDEX(',',y,0)+1,LEN(y))
    FROM
        RecursiveCSV
    WHERE
        SUBSTRING(y,CHARINDEX(',',y,0)+1,LEN(y)) <> '' OR
        SUBSTRING(y,0,CHARINDEX(',',y,0)) <> ''
)
SELECT
    *
FROM
```

answered May 13 '11 at 15:03

Runonthespot
650 7 15

---

6

I use a more concise version of the top voted answer:

```
List<SqlParameter> parameters = tags.Select((s, i) => new SqlParameter("@tag" +
i.ToString(), SqlDbType.NVarChar(50)) { Value = s}).ToList();

var whereCondition = string.Format("tags in ({0})", String.Join(",",parameters.Select(s
=> s.ParameterName)));
```

It does loop through the tag parameters twice; but that doesn't matter most of the time (it won't be your bottleneck; if it is, unroll the loop).

If you're really interested in performance and don't want to iterate through the loop twice, here's a less beautiful version:

```
var parameters = new List<SqlParameter>();
var paramNames = new List<string>();
for (var i = 0; i < tags.Length; i++)
{
    var paramName = "@tag" + i;

    //Include size and set value explicitly (not AddWithValue)
    //Because SQL Server may use an implicit conversion if it doesn't know
    //the actual size.
    var p = new SqlParameter(paramName, SqlDbType.NVarChar(50) { Value = tags[i]; }
    paramNames.Add(paramName);
    parameters.Add(p);
}

var inClause = string.Join(",", paramNames);
```

edited May 23 '17 at 12:10



Community ♦

answered Mar 12 '15 at 18:11

George Stocker ♦

6

This approach is blogged about in [OPENJSON - one of best ways to select rows by list of ids](#).

A full worked example below

```
CREATE TABLE dbo.Tags
  (
     Name  VARCHAR(50),
     Count INT
  )

INSERT INTO dbo.Tags
VALUES      ('VB',982), ('ruby',1306), ('rails',1478), ('scruffy',1), ('C#',1784)

GO

CREATE PROC dbo.SomeProc
@Tags VARCHAR(MAX)
AS
SELECT T.*
FROM   dbo.Tags T
WHERE  T.Name IN (SELECT J.Value COLLATE Latin1_General_CI_AS
                  FROM   OPENJSON(CONCAT('[', @Tags, ']')) J)
ORDER  BY T.Count DESC

GO

EXEC dbo.SomeProc @Tags = '"ruby","rails","scruffy","rubyonrails"'

DROP TABLE dbo.Tags
```

edited Nov 28 '15 at 18:24

answered Nov 7 '15 at 14:57

Martin Smith

**357k** 61 601 710

---

Here is another answer to this problem.

(new version posted on 6/4/13).

```
        using (var sqlConn = new SqlConnection(scsb.ConnectionString))
        {
            var sqlParameters = new List<SqlParameter>();
            var replacementStrings = new Dictionary<string, string>();
            if (pars != null)
            {
                for (int i = 0; i < pars.Length; i++)
                {
                    if (pars[i] is IEnumerable<object>)
                    {
                        List<object> enumerable = (pars[i] as
IEnumerable<object>).ToList();
                        replacementStrings.Add("@" + i, String.Join(",",
enumerable.Select((value, pos) => String.Format("@_{0}_{1}", i, pos))));
                        sqlParameters.AddRange(enumerable.Select((value, pos) => new
SqlParameter(String.Format("@_{0}_{1}", i, pos), value ?? DBNull.Value)).ToArray());
                    }
                    else
                    {
                        sqlParameters.Add(new SqlParameter(String.Format("@{0}", i),
pars[i] ?? DBNull.Value));
                    }
                }
            }
            strSql = replacementStrings.Aggregate(strSql, (current, replacementString)
=> current.Replace(replacementString.Key, replacementString.Value));
            using (var sqlCommand = new SqlCommand(strSql, sqlConn))
            {
                if (pars != null)
                {
                    sqlCommand.Parameters.AddRange(sqlParameters.ToArray());
                }
                else
                {
                    //Fail-safe, just in case a user intends to pass a single null
parameter
                    sqlCommand.Parameters.Add(new SqlParameter("@0", DBNull.Value));
                }
                using (var sqlDataAdapter = new SqlDataAdapter(sqlCommand))
                {
                    sqlDataAdapter.Fill(ds);
                }
            }
```

edited Jun 4 '13 at 13:58

answered Jun 3 '13 at 22:54

Darek
**3,580** 20 38

---

4

The only winning move is not to play.

No infinite variability for you. Only finite variability.

In the SQL you have a clause like this:

```
and ( {1}==0 or b.CompanyId in ({2},{3},{4},{5},{6}) )
```

In the C# code you do something like this:

```
int origCount = idList.Count;
if (origCount > 5) {
  throw new Exception("You may only specify up to five originators to filter on.");
}
while (idList.Count < 5) { idList.Add(-1); }  // -1 is an impossible value
return ExecuteQuery<PublishDate>(getValuesInListSQL,
             origCount,
             idList[0], idList[1], idList[2], idList[3], idList[4]);
```

So basically if the count is 0 then there is no filter and everything goes through. If the count is higher than 0 the then the value must be in the list, but the list has been padded out to five with impossible values (so that the SQL still makes sense)

Sometimes the lame solution is the only one that actually works.

answered Apr 28 '11 at 21:56

Jason Henriksen
**174** 1 1 6

---

Thank you for your interest in this question. Because it has attracted low-quality or spam answers that had to be removed, posting an answer now requires 10 reputation on this site (the association bonus does not count).

Would you like to answer one of these unanswered questions instead?