Evaluate in T-SQL

Asked 15 years, 9 months ago Modified 12 years, 9 months ago Viewed 18k times

9

I've got a stored procedure that allows an IN parameter specify what database to use. I then use a pre-decided table in that database for a query. The problem I'm having is concatenating the table name to that database name within my queries. If T-SQL had an evaluate function I could do something like

```
eval(@dbname + 'MyTable')
```

Currently I'm stuck creating a string and then using `exec()` to run that string as a query. This is messy and I would rather not have to create a string. Is there a way I can evaluate a variable or string so I can do something like the following?

```
SELECT *
FROM eval(@dbname + 'MyTable')
```

I would like it to evaluate so it ends up appearing like this:

```
SELECT *
FROM myserver.mydatabase.dbo.MyTable
```

sql-server-2005 t-sql evaluation

Share Improve this question Follow

asked Mar 27, 2009 at 3:27

Joe Phillips
51k ● 33 ● 108 ● 165

d03boy - I deleted my suggestion as I forgot that "Use" cannot be used in a stored procedure. I should have known better, too, as I ran into this a few months back. Sorry if I led you down the wrong path. – Mark Brittingham Apr 5, 2009 at 13:39

I didn't know it couldn't be used so I at least learned that much :) – Joe Phillips Apr 5, 2009 at 18:05

d03boy - I deleted my third suggestion which was to use OPENROWSET as you cannot use variables in the connection string or the query. Bummer. – Hafthor Apr 6, 2009 at 22:58

11 Answers

Sorted by: Highest score (default)

16

Read this... The Curse and Blessings of Dynamic SQL, help me a lot understanding how to solve this type of problems.

Share Improve this answer Follow

answered Apr 1, 2009 at 20:59

Alan Featherston
1,086 ● 3 ● 14 ● 27

+50

9

There's no "neater" way to do this. You'll save time if you accept it and look at something else.

EDIT: Aha! Regarding the OP's comment that "We have to load data into a new database each month or else it gets too large.". Surprising in retrospect that no one remarked on the faint smell of this problem.

SQL Server offers native mechanisms for dealing with tables that get "too large" (in particular, partitioning), which will allow you to address the table as a single entity, while dividing the table into separate files in the background, thus eliminating your current problem altogether.

To put it another way, this is a problem for your DB administrator, not the DB consumer. If that happens to be you as well, I suggest you look into partitioning this table.

Share

Improve this answer

Follow

edited Apr 6, 2009 at 22:33

answered Mar 27, 2009 at 3:40

harpo
43.1k ● 14 ● 100 ● 133

Amen. Sounds like his db schema is the route of all evil here. – Paul Suart Apr 2, 2009 at 12:09

Why is that? We have to load data into a new database each month or else it gets too large. – Joe Phillips Apr 3, 2009 at 18:37

It's not actually our software. We just create new "projects" each month within the software. We can do some work to the database but it is limited by our lack of knowledge of the system. – Joe Phillips Apr 5, 2009 at 2:53

Plus, this spans across many projects which will be in different DBs no matter what. So the code will still have to be dynamic. – Joe Phillips Apr 5, 2009 at 2:55

In that case, I stand by my first answer. – harpo Apr 5, 2009 at 19:39

6

try the sp_executesql built in function. You can basically build up your SQL string in your proc, then call

```
exec sp_executesql @SQLString.

DECLARE @SQLString nvarchar(max)
SELECT @SQLString = '
SELECT *
FROM  ' +  @TableName

EXEC sp_executesql @SQLString
```

Share

Improve this answer

Follow

edited Mar 22, 2012 at 13:18

Taryn

247k ● 57 ● 370 ● 408

answered Mar 27, 2009 at 3:29

Scott Ferguson

7,810 ● 7 ● 43 ● 64

this is what he's doing currently and does not like. – Learning Mar 27, 2009 at 3:35

1 That's what I'd like to avoid :) – Joe Phillips Mar 27, 2009 at 3:39

2

You can't specify a dynamic table name in SQL Server.

There are a few options:

1. Use dynamic SQL
2. Play around with synonyms (which means less dynamic SQL, but still some)

You've said you don't like 1, so lets go for 2.

First option is to restrict the messyness to one line:

```
begin transaction t1;
declare @statement nvarchar(100);

set @statement = 'create synonym temptablesyn for db1.dbo.test;'
exec sp_executesql @statement

select * from db_syn

drop synonym db_syn;

rollback transaction t1;
```

I'm not sure I like this, but it may be your best option. This way all of the SELECTs will be the same.

You can refactor this to your hearts content, but there are a number of disadvantages to this, including the synonym is created in a transaction, so you can't have two of the queries running at the same time (because both will be trying to create temptablesyn). Depending upon the locking strategy, one will block the other.

Synonyms are permanent, so this is why you need to do this in a transaction.

Share Improve this answer Follow

answered Apr 1, 2009 at 11:16

Matthew Farwell

61.7k ● 18 ● 132 ● 173

1

There are a few options, but they are messier than the way you are already doing. I suggest you either:
(1) Stick with the current approach
(2) Go ahead and embed the SQL in the code since you are doing it anyway.
(3) Be extra careful to validate your input to avoid SQL Injection.

Also, messiness isn't the only problem with dynamic SQL. Remember the following:
(1) Dynamic SQL thwarts the server's ability to create a reusable execution plan.
(2) The ExecuteSQL command breaks the ownership chain. That means the code will run in the context of the user who calls the stored procedure NOT the owner of the procedure. This might force you to open security on whatever table the statement is running against and create other security issues.

Share Improve this answer Follow

answered Apr 1, 2009 at 21:17

JohnFx

34.9k ● 18 ● 107 ● 166

1

Just a thought, but if you had a pre-defined list of these databases, then you could create a single view in the database that you connect to to join them - something like:

```
CREATE VIEW dbo.all_tables
AS

SELECT  your_columns,
        'db_name1' AS database_name
FROM    db_name1.dbo.your_table

UNION ALL

SELECT  your_columns,
        'db_name2'
FROM    db_name2.dbo.your_table
```

```
etc...
```

Then, you could pass your database name in to your stored procedure and simply use it as a paramater in a WHERE clause. If the tables are large, you might consider using an indexed view, indexed on the new database_name column (or whatever you call it) and the tables' primary key (I'm assuming from the question that the tables' schemas are the same?).

Obviously if your list of database changes frequently, then this becomes more problematic - but if you're having to create these databases anyway, then maintaining this view at the same time shouldn't be too much of an overhead!

Share Improve this answer Follow

answered Apr 4, 2009 at 7:20

David M
72.8k • 13 • 163 • 187

These databases are added so often that it would be difficult to keep track and maintain a proper list. – Joe Phillips Apr 5, 2009 at 2:56

Then make a database maintenance script that creates the view (nightly?) by going through all the databases on the server and checking if they are of the appropriate type. You could also then automatically insert a text column in your selects that is the name of the database (so you can pass it in) – Stephen Apr 6, 2009 at 20:47

1

I think Mark Brittingham has the right idea (here: http://stackoverflow.com/questions/688425/evaluate-in-t-sql/718223#718223), which is to issue a `use database` command and write the sp to NOT fully qualify the table name. As he notes, this will act on tables in the login's current database.

Let me add a few possible elaborations:

From a comment by the OP, I gather the database is changed once a month, when it gets "too large". ("We have to load data into a new database each month or else it gets too large. – d03boy")

1. User logins have a default database, set with sp_defaultdb (deprecated) or ALTER LOGIN. If each month you move on to the new database, and don't need to run the sp on the older copies, just change the login's default db monthly, and again, don't fully qualify the table name.
2. The database to use can be set in the client login: `sqlcmd -U login_id -P password -d db_name`, then exec the sp from there.
3. You can establish a connection to the database using the client of your choice (command line, ODBC, JDBC), then issue a `use database` command, the exec

the sp.

use database bar; exec sp_foo;

Once the database has been set using one of the above, you have three choices for executing the stored procedure:

1. You could just copy the sp along with the database, in to the new database. As long as the table name is NOT fully qualified, you'll operate on the new database's table.

   exec sp_foo;

2. You could install the single canonical copy of the sp in its own database, call it `procs`, with the tablename not fully qualified, and then call its fuly qualified name:

   exec procs.dbo.sp_foo;

3. You could, in each individual database, install a stub `sp_foo` that execs the fully qualified name of the real sp, and then exec `sp_foo` without qualifying it. The stub will be called, and it will call the real procedure in `procs`. (Unfortunately, `use database dbname` cannot be executed from within an sp.)

   ```
   --sp_foo stub:
   create proc bar.dbo.sp_foo
    @parm int
   as
   begin
     exec procs.dbo.sp_foo @parm;
   end
   go
   ```

However this is done, if the database is being changed, the real sp should be created with the `WITH RECOMPILE` option, otherwise it'll cache an execution plan for the wrong table. The stub of course doesn't need this.

Share Improve this answer Follow

answered Apr 5, 2009 at 13:26

tpdi

35.1k ● 11 ● 82 ● 123

1

You could create a SQL CLR Table-Valued UDF to access the tables. You have to tie it to the schema because TV-UDFs don't support dynamic schema. (My sample includes an ID and a Title column - modify for your needs)

Once you've done this, you should be able to do the follow query:

```
SELECT * FROM dbo.FromMyTable('table1')
```

You can include a multipart name in that string too.

```
SELECT * FROM dbo.FromMyTable('otherdb..table1')
```

to return the ID,Title columns from that table.

You will likely need to enable SQL CLR and turn on the TRUSTWORTHY option:

```
sp_configure 'clr enabled',1
go
reconfigure
go
alter database mydatabase set trustworthy on
```

Create a C# SQL Project, add a new UDF file, paste this in there. Set Project Property, Database, Permission Level to external. Build, deploy. Can be done without VisualStudio. Let me know if you need that.

```
using System;
using System.Data.SqlTypes;
using Microsoft.SqlServer.Server;
using System.Collections;
using System.Data.SqlClient;

[assembly: CLSCompliant(true)]
namespace FromMyTable
{
    public static partial class UserDefinedFunctions
    {
        [Microsoft.SqlServer.Server.SqlFunction(DataAccess =
DataAccessKind.Read, IsDeterministic = true, SystemDataAccess =
SystemDataAccessKind.Read, IsPrecise = true, FillRowMethodName = "FillRow",
            TableDefinition = "id int, title nvarchar(1024)")]
        public static IEnumerable FromMyTable(SqlString tableName)
        {
            return new FromMyTable(tableName.Value);
        }

        public static void FillRow(object row, out SqlInt32 id, out SqlString
title)
        {
            MyTableSchema v = (MyTableSchema)row;
            id = new SqlInt32(v.id);
            title = new SqlString(v.title);
        }
    }

    public class MyTableSchema
    {
        public int id;
        public string title;
        public MyTableSchema(int id, string title) { this.id = id; this.title =
title; }
    }
```

```
    internal class FromMyTable : IEnumerable
    {
        string tableName;

        public FromMyTable(string tableName)
        {
            this.tableName = tableName;
        }

        public IEnumerator GetEnumerator()
        {
            return new FromMyTableEnum(tableName);
        }
    }

    internal class FromMyTableEnum : IEnumerator
    {
        SqlConnection cn;
        SqlCommand cmd;
        SqlDataReader rdr;
        string tableName;

        public FromMyTableEnum(string tableName)
        {
            this.tableName = tableName;
            Reset();
        }

        public MyTableSchema Current
        {
            get { return new MyTableSchema((int)rdr["id"],
(string)rdr["title"]); }
        }

        object IEnumerator.Current
        {
            get { return Current; }
        }

        public bool MoveNext()
        {
            bool b = rdr.Read();
            if (!b) { rdr.Dispose(); cmd.Dispose(); cn.Dispose(); rdr = null;
cmd = null; cn = null; }
            return b;
        }

        public void Reset()
        {
            // note: cannot use a context connection here because it will be
closed
            // in between calls to the enumerator.
            if (cn == null) { cn = new
SqlConnection("server=localhost;database=mydatabase;Integrated
Security=true;"); cn.Open(); }
            if (cmd == null) cmd = new SqlCommand("select id, title FROM " +
tableName, cn);
            if (rdr != null) rdr.Dispose();
            rdr = cmd.ExecuteReader();
        }
```

```
    }
}
```

Share

Improve this answer

Follow

edited Apr 6, 2009 at 23:49

answered Apr 6, 2009 at 18:48

Hafthor

16.9k ● 9 ● 58 ● 66

0

```
declare @sql varchar(256);
set @sql = 'select * into ##myGlobalTemporaryTable from '+@dbname
exec sp_executesql @sql

select * from ##myGlobalTemporaryTable
```

copies into a global temporary table which you can then use like a regular table

Share Improve this answer Follow

answered Apr 5, 2009 at 2:12

Hafthor

16.9k ● 9 ● 58 ● 66

This would very likely use too many resources in my specific case. Huge tables.
– Joe Phillips Apr 5, 2009 at 2:59

0

If you have a reasonably manageable number of databases, it may be best to use a pre-defined conditional statement like:

```
if (@dbname = 'db1')
  select * from db1..MyTable
if (@dbname = 'db2')
  select * from db2..MyTable
if (@dbname = 'db3')
  select * from db3..MyTable
```

...

you can generate this proc as part of your database creation scripts if you are changing the list of databases available to query.

This avoids security concerns with dynamic sql. You can also improve performance by replacing the 'select' statements with stored procedures targeting each database (1 cached execution plan per query).

Share Improve this answer Follow

answered Apr 5, 2009 at 5:30

John

0

```
if exists (select * from master..sysservers where srvname = 'fromdb')
    exec sp_dropserver 'fromdb'
go

declare @mydb nvarchar(99);
set @mydb='mydatabase'; -- variable to select database

exec sp_addlinkedserver @server = N'fromdb',
    @srvproduct = N'',
    @provider = N'SQLOLEDB',
    @datasrc = @@servername,
    @catalog = @mydb
go

select * from OPENQUERY(fromdb, 'select * from table1')
go
```

Share Improve this answer Follow

answered Apr 8, 2009 at 21:42

Hafthor

16.9k ● 9 ● 58 ● 66