How can I add user-supplied input to an SQL statement?

Asked 3 years, 6 months ago Active 1 year, 5 months ago Viewed 5k times

44

8

I am trying to create an SQL statement using user-supplied data. I use code similar to this in C#:

```
var sql = "INSERT INTO myTable (myField1, myField2) " +
          "VALUES ('" + someVariable + "', '" + someTextBox.Text + "');";

var cmd = new SqlCommand(sql, myDbConnection);
cmd.ExecuteNonQuery();
```

and this in VB.NET:

```
Dim sql = "INSERT INTO myTable (myField1, myField2) " &
          "VALUES ('" & someVariable & "', '" & someTextBox.Text & "');"

Dim cmd As New SqlCommand(sql, myDbConnection)
cmd.ExecuteNonQuery()
```

However,

- this fails when the user input contains single quotes (e.g. `O'Brien`),
- I cannot seem to get the format right when inserting DateTime values and
- people keep telling me that I should not do this because of "SQL injection".

How do I do it "the right way"?

c# sql vb.net ado.net sql-injection

edited Mar 27 '18 at 7:31

asked Feb 2 '16 at 20:39

Heinzi

127k 42 281 421

2 If you would like a more indepth look at what "SQL Injection" is and why it is dangerous see the question: "How can I explain SQL injection without technical jargon?" from our Information Security sister site. – Scott Chamberlain Feb 2 '16 at 21:11

1 You should wiki this, btw. – Will Feb 2 '16 at 21:12

2 @Will: Won't CW'ing the question also CW all future answers, and, thus, discourage others from contributing better answers than mine? – Heinzi Feb 2 '16 at 21:19

1 @Igor: Good idea, done. I have also moved the VB version of the question code directly to the question, to make it obvious that this is about VB as well. – Heinzi Mar 27 '18 at 7:33

2 Answers

50

Use parameterized SQL.

Examples

(These examples are in C#, see below for the VB.NET version.)

Replace your string concatenations with `@...` placeholders and, afterwards, add the values to your SqlCommand. You can choose the name of the placeholders freely, just make sure that they start with the `@` sign. Your example would look like this:

```
var sql = "INSERT INTO myTable (myField1, myField2) " +
          "VALUES (@someValue, @someOtherValue);";

using (var cmd = new SqlCommand(sql, myDbConnection))
{
    cmd.Parameters.AddWithValue("@someValue", someVariable);
    cmd.Parameters.AddWithValue("@someOtherValue", someTextBox.Text);
    cmd.ExecuteNonQuery();
}
```

The same pattern is used for other kinds of SQL statements:

```
var sql = "UPDATE myTable SET myField1 = @newValue WHERE myField2 = @someValue;";

// see above, same as INSERT
```

```
var sql = "SELECT myField1, myField2 FROM myTable WHERE myField3 = @someValue;";

using (var cmd = new SqlCommand(sql, myDbConnection))
{
    cmd.Parameters.AddWithValue("@someValue", someVariable);
    using (var reader = cmd.ExecuteReader())
    {
        ...
    }
    // Alternatively: object result = cmd.ExecuteScalar();
    // if you are only interested in one value of one row.
}
```

A word of caution: `AddWithValue` is a good starting point and works fine in most cases. However, that the value you pass in needs to exactly match the data type of the corresponding database field. Otherwise, you might end up in a situation where the conversion prevents your query from using an index. Note that some SQL Server data types, such as char/varchar (without preceding "n") or date do not have a corresponding .NET data type. In those cases, `Add` with the correct data type should be used instead.

Why should I do that?

- It's more secure: It stops SQL injection. (Bobby Tables won't delete your student records.)
- It's easier: No need to fiddle around with single and double quotes or to look up the correct string representation of date literals.
- It's more stable: `O'Brien` won't crash your application just because he insists on keeping his strange name.

Other database access libraries

- If you use an OleDbCommand instead of an SqlCommand (e.g., if you are using an MS Access database), use `?` instead of `@...` as the placeholder in the SQL. In that case, the first parameter of `AddWithValue` is irrelevant; instead, you need to add the parameters in the correct order. The same is true for OdbcCommand.
- Entity Framework also supports parameterized queries.

edited Mar 27 '18 at 7:33

community wiki
7 revs
Heinzi

Careful with the "other kind" statement -- SELECT won't work with `cmd.ExecuteNonQuery()` – Hogan Feb 2 '16 at 20:56

@Hogan: True. I thought about giving a more complete example, but since this answer is mainly about transitioning from string-concatenated SQL to

@Heinze - I think I'd like a short bullet point under special cases saying something like "using parameters will work even if you are doing 'ExecuteQuery(), ExecuteReader(), ExecuteScalar()` or others. – Hogan Feb 2 '16 at 22:06

@Hogan: I've expanded it a bit. I'm still trying to keep it short. There is no SqlCommand.ExecuteQuery(). – Heinzi Feb 4 '16 at 7:28

2

VB.NET Example Code

This is the example code for the wiki answer in `vb.net`, assuming `Option Strict On` and `Option Infer On`.

INSERT

```
Dim sql = "INSERT INTO myTable (myField1, myField2) " &
          "VALUES (@someValue, @someOtherValue);"

Using cmd As New SqlCommand(sql, myDbConnection)
    cmd.Parameters.AddWithValue("@someValue", someVariable)
    cmd.Parameters.AddWithValue("@someOtherValue", someTextBox.Text)
    cmd.ExecuteNonQuery()
End Using
```

UPDATE

```
Dim sql = "UPDATE myTable SET myField1 = @newValue WHERE myField2 = @someValue;"

' see above, same as INSERT
```

SELECT

```
Dim sql = "SELECT myField1, myField2 FROM myTable WHERE myField3 = @someValue;"

Using cmd As New SqlCommand(sql, myDbConnection)
    cmd.Parameters.AddWithValue("@someValue", someVariable)
    Using reader = cmd.ExecuteReader()
        ' ...
    End Using
    ' Alternatively: Dim result = cmd.ExecuteScalar()
    ' if you are only interested in one value of one row
```

edited Mar 27 '18 at 7:36

Heinzi
127k 42 281 421

answered Mar 26 '18 at 19:48

Igor
45.3k 4 56 116