

SQL-frågor och parametrar

Upphovsrätt för detta verk

Detta verk är framtaget i anslutning till kursen ASP.NET Web Forms vid Linnéuniversitetet.

Du får använda detta verk så här:

Allt innehåll i detta verk av Mats Looch, förutom Linnéuniversitetets logotyp och symbol samt ikoner, bilder och fotografier, är licensierad under:



Creative Commons Erkännande 4.0 Internationell licens.

<http://creativecommons.org/licenses/by/4.0>

Det betyder att du i icke-kommersiella syften får:

- kopiera hela eller delar av innehållet
- sprida hela eller delar av innehållet
- visa hela eller delar av innehållet offentligt och digitalt
- konvertera innehållet till annat format
- du får även göra om innehållet

Om du förändrar innehållet så ta inte med Linnéuniversitetets logotyp och symbol samt ikoner och fotografier i din nya version!

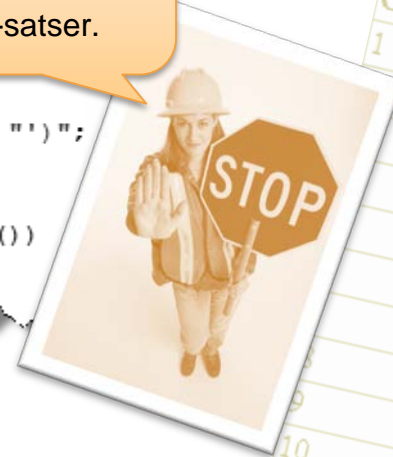
Vid all användning måste du ange källan: "Linnéuniversitetet – ASP.NET Web Forms" och en länk till <https://coursepress.lnu.se/kurs/aspnet-web-forms> och till Creative Common-licensen här ovan.

"SQL Injections"

- ✓ "SQL injection" är då en elak användare postar data som kan tolkas som SQL-satser av applikationen.

```
// Open the database connection.  
con.Open();  
  
string sql =  
    "SELECT CreditCardID, CardType, CardNum  
    "ExpMonth, ExpYear, ModifiedDate "  
    "FROM Sales.CreditCard "  
    "WHERE (CardNumber = N'" + cardNumber + "')";  
SqlCommand cmd = new SqlCommand(sql, con);  
  
using (SqlDataReader reader = cmd.ExecuteReader())  
{  
    CreditCardGridView.DataSource = reader;  
}
```

Konkatenera
ALDRIG strängar i
samband med
SQL-satser.



- ✓ Om användaren matar in
12345' OR '1'='1
i textrutan kommer alla poster i tabellen **Sales.CreditCard** att returneras! '1'='1'
är ju alltid sant!
- ✓ Genom att använda parameterfrågor eller lagrade procedurer kan detta förhindras.

Credit Card Number: 12345' OR '1'='1'

CreditCardID	CardType	CardNum
1	SuperiorCard	333326646953
2	Distinguish	5555212724972
3	ColonialVoice	77778344838353
4	ColonialVoice	77774915718248
5	Vista	11114404600042
6	Distinguish	55557132036181
7	Distinguish	55553635401028
8	SuperiorCard	33336081193101
9	Distinguish	55553465625901
10	SuperiorCard	333312586493
11	SuperiorCard	33335352517363
12	Vista	33334316194519
13	Distinguish	55553287727410
14	SuperiorCard	333312586493
15	Distinguish	55553287727410
16	SuperiorCard	333312586493

Kommandon och parametrar

- ✓ SqlCommand-objektet använder en parameter vilket effektivt förhindrar "SQL injection".

```
// Open the database connection
con.Open();

// Create and configure a new command containing 1 named parameter
string sql =
    "SELECT CreditCardID, CardType, CardNumber, " +
    "ExpMonth, ExpYear, ModifiedDate " +
    "FROM     Sales.CreditCard " +
    "WHERE     (CardNumber = @CardNumber)";
SqlCommand cmd = new SqlCommand(sql, con);

// Create and configure a SqlParameter object for the card number parameter, OR...
//SqlParameter param = new SqlParameter("@CardNumber", SqlDbType.NVarChar, 25);
//param.Value = CardNumberTextBox.Text;
//cmd.Parameters.Add(param);

// ...use shorthand syntax to do exactly the same thing.
cmd.Parameters.Add("@CardNumber", SqlDbType.NVarChar, 25).Value = CardNumberTextBox.Text;

//SqlParameter param = cmd.Parameters.Add("@CardNumber", SqlDbType.NVarChar, 25);
//param.Value = CardNumberTextBox.Text;
```



Lagrade procedurer

- ✓ Det finns egentligen bara fördelar med att använda lagrade procedurer jämfört med "hårdkodade frågor".
 - Enklare att underhålla.
 - Säkrare.
 - Effektivare.
- ✓ Att anropa en lagrad procedur påminner mycket om att använda en parameterfråga (om den lagrade proceduren har parametrar). Skillnaderna är...
 - Istället för att ange en SQL-fråga anger du namnet på den lagrade proceduren.
 - Du måste sätta SqlCommand-objektets egenskap CommandType till CommandType.StoredProcedure.

```
CREATE PROCEDURE [Sales].[uspGetCreditCard]
    @CreditCardID int
AS
BEGIN
    SET NOCOUNT ON;

    SELECT CreditCardID, CardType, CardNumber, ExpMonth, Ex
    FROM Sales.CreditCard
    WHERE (CreditCardID = @CreditCardID)
END
```

Anropa lagrad procedur med parameter

```
// Ändrar kommandotypen så att den lagrade proceduren kan exekveras av SqlCommand-objektet
SqlCommand cmd = new SqlCommand("app.uspGetCustomer", conn);
cmd.CommandType = CommandType.StoredProcedure;

// Lägger till den parameter den lagrade proceduren kräver. AddWithValue är
// inte den effektivaste metoden att använda - men den enklaste...
cmd.Parameters.AddWithValue("@CustomerId", customerId);

// Öppnar anslutningen till databasen.
conn.Open();
```

- ✓ Metoden `AddWithValue()` är ytterligare ett sätt (kanske det enklaste MEN det mest ineffektivaste) att lägga till en parameter till ett kommando.
- ✓ Bästa metoden att använda för att lägga till en parameter är `Add`.

```
cmd.Parameters.Add("@CustomerId", SqlDbType.Int, 4).Value = customerId;
```

- ✓ **OBS!** Glöm inte att ändra `CommandType` till `StoredProcedure`. Standardvärdet är `Text`.

...ett effektivare sätt.

```
// Skapar ett SqlCommand-objekt som kan exekvera den lagrade proceduren uspInsertCustomer.  
SqlCommand cmd = new SqlCommand("app.uspDeleteCustomer", conn);  
  
// Ändrar kommandotypen så att den lagrade proceduren kan exekveras av SqlCommand-objektet.  
cmd.CommandType = CommandType.StoredProcedure;  
  
// Lägger till de paramterar den lagrade proceduren kräver. Använder här det effektiva sättet att  
// göra det på - något "svårare" men ASP.NET behöver inte "jobba" så mycket.  
cmd.Parameters.Add("@CustomerId", SqlDbType.Int, 4).Value = customerId;  
  
// Öppnar anslutningen till databasen.  
conn.Open();  
  
// Den lagrade proceduren innehåller en DELETE-sats och returnerar inga poster varför metoden  
// ExecuteNonQuery används för att exekvera den lagrade proceduren.  
cmd.ExecuteNonQuery();
```

- ✓ Ovan ser du det effektivaste sättet att skapa, lägga till och initiera ett parameterobjekt.

OUTPUT-parameter

```
/// <summary>
/// Skapar en ny post i tabellen Customer.
/// </summary>
/// <param name="customer">Kunduppgifter som ska läggas till.</param>
public void InsertCustomer(Customer customer)
{
    // Skapar och initierar ett anslutningsobjekt.
    using (SqlConnection conn = CreateConnection())
    {
        try
        {
            // Skapar och initierar ett SqlCommand-objekt som används till att
            // exekveras specifierad lagrad procedur.
            SqlCommand cmd = new SqlCommand("app.uspInsertCustomer", conn);
            cmd.CommandType = CommandType.StoredProcedure;

            // Lägger till de paramterar den lagrade proceduren kräver. Använder här det effektiva sättet att
            // göra det på - något "svårare" men ASP.NET behöver inte "jobba" så mycket.
            cmd.Parameters.Add("@Name", SqlDbType.VarChar, 30).Value = customer.Name;
            cmd.Parameters.Add("@Address", SqlDbType.VarChar, 30).Value = customer.Address;
            cmd.Parameters.Add("@PostalCode", SqlDbType.VarChar, 6).Value = customer.PostalCode;
            cmd.Parameters.Add("@City", SqlDbType.VarChar, 30).Value = customer.City;

            // Den här paramtern är lite speciell. Den skickar inte något data till den lagrade proceduren,
            // utan hämtar data från den. (Fungerar ungerfär som ref- och out-prameterar i C#.) Värdet
            // paramtern kommer att ha EFTER att den lagrade proceduren exekverats är primärnyckels värde
            // den nya posten blivit tilldelad av databasen.
            cmd.Parameters.Add("@CustomerId", SqlDbType.Int, 4).Direction = ParameterDirection.Output;

            // Öppnar anslutningen till databasen.
            conn.Open();

            // Den lagrade proceduren innehåller en INSERT-sats och returnerar inga poster varför metoden
            // ExecuteNonQuery används för att exekvera den lagrade proceduren.
            cmd.ExecuteNonQuery();

            // Hämtar primärnyckels värde för den nya posten och tilldelar Customer-objektet värdet.
            customer.CustomerId = (int)cmd.Parameters["@CustomerId"].Value;
        }
        catch
        {
            // Kaster ett eget undantag om ett undantag kastas.
            throw new ApplicationException("An error occurred in the data access layer.");
        }
    }
}
```

✓ Har den lagrade proceduren en parameter som är av typen **OUTPUT** måste du skapa ett SqlParameter-objekt av motsvarande typ. Du gör det genom att använda egenskapen **Direction**.

✓ **OUTPUT**-parameterns värde får du tillgång till via egenskapen **Value** som är av typen **object** varför dess värde måste typomvandlas.