SQL-frågor och parametrar

Upphovsrätt för detta verk

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

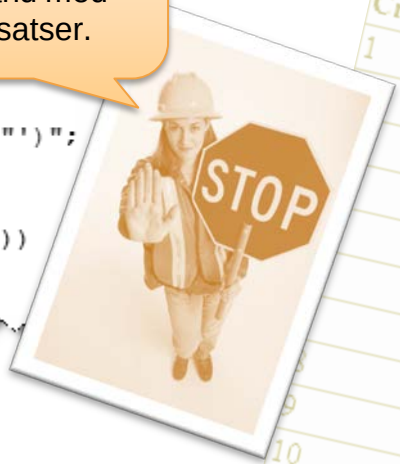
Du får använda detta verk så här:

"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, Ca
    "ExpMonth, ExpYear, ModifiedDate "
    "FROM          Sales.CreditCard " +
    "WHERE      (CardNumber = N'" + cardNumber + "')";
SqlCommand cmd = new SqlCommand(sql, con);

using (SqlDataReader reader = cmd.ExecuteReader())
{
    CreditCardGr  View.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.

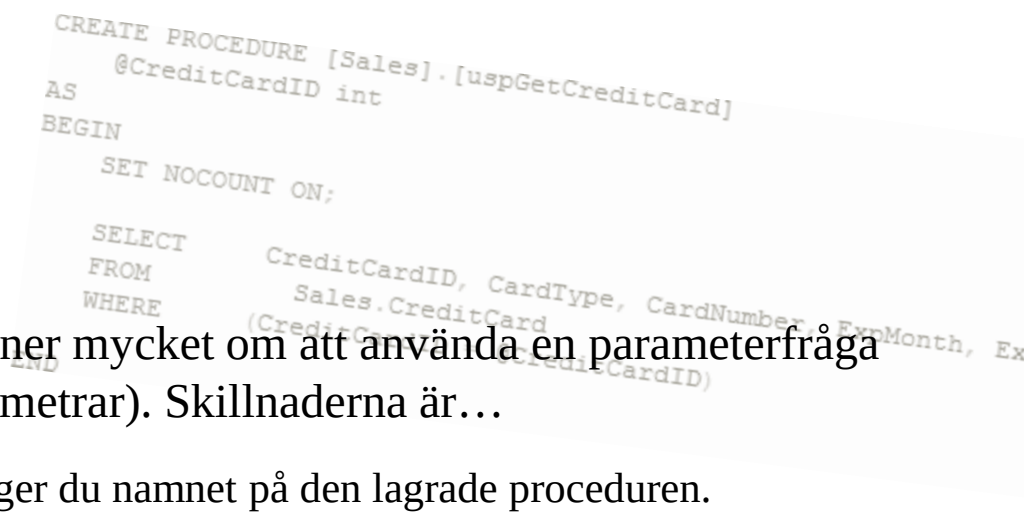
Kommandon och parametrar

- `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;

SqlDataReader reader = cmd.ExecuteReader();
```

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`.



Anropa lagrad procedur med parameter

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

// Lägger till den paramter 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 sätttet 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 sätttet 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 parametern 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
            // parametern kommer att ha EFTER att den lagrade proceduren exekverats är primärnycklens 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 primärnyckelns värde för den nya posten och tilldelar Customer-objektet värdet.
            customer.CustomerId = (int)cmd.Parameters["@CustomerId"].Value;
        }
        catch
        {
            // Kastar ett eget undantag om ett undantag kastas.
            throw new ApplicationException("An error occured 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.