Parameterisering af databasekald med ASP og ADO

Jeg har efterhånden set en del spørgsmål her på Eksperten vedr. SQL injections og hvordan man kan beskytte sig. Efter min opfattelse er det rigtige svar: Parameterisering! Dette vil jeg forsøge at beskrive hvordan gøres med Command-objektet (ADO og ASP).

Skrevet den **13. Apr 2010** af **softspot** I kategorien **Programmering / ASP** | ⭐⭐⭐☆☆

Ændringslog
18-11-2008: Artikel oprettet.
24-11-2008: Tilføjelse af nyt afsnit om genbrug af command-objektet, samt sektion med kommentar til kommentar.
17-06-2009: Ombrydning af kodeeksempler, så de ikke skaber scrollbars i rammerne til kodeeksemplerne.
13-04-2010: Tilføjelse af en kort udgave til kald af Command-objektet, samt eksempel på indpakning af Command-objekt oprettelsen.

Den gængse metode
En gængs metode at lave databasekald i ASP, er at sammensætte en SQL-streng af nogle SQL kommandoer og nogle parametre. Dette er dog en kilde til problemer på flere områder. Dels risikerer man at typen af en parameter ikke stemmer overens med det felt man forsøger at gemme data i, dels bliver man sårbar overfor "SQL-injections". Ydermere har strengsammensætningsmetoden også en tendens til at blive uoverskuelig, hvis man ikke holder en stram disciplin.

Eksempel på strengsammensat SQL:

```
dim sql
sql = "SELECT * " & _
      "FROM tabel " & _
      "WHERE kodeord = '" & request.querystring("kodeord")& "' " & _
      "AND brugernavn = '" & request.querystring("brugernavn") & "'"
```

Når du har sådan en konstruktion i din kode, skal alarmklokkerne begynde at ringe, fordi du har åbnet døren på vid gab for SQL-injections. Request.QueryString er som bekendt værdier der sendes via url'en og er derfor noget af det simpleste at overføre til en side. Det bliver ikke meget bedre af at det er Request.Form der benyttes! En begynder kan relativt let POSTe en form til en url.

Nu kunne jeg begynde at skrive en masse om hvordan man modellerer sine input, så de er sikret mod SQL-injections, men det vil jeg ikke (det er der skrevet mange andre artikler om på internettet). Nej, jeg vil i stedet slå et slag for parametre og Command-objekter i ADO!

Parametre og Command-objektet
Med parametre kan du være nogenlunde sikker på, at databasekaldet (hvis det når databasen) er korrekt og fri for injections.

Før jeg går igang med at eksemplificere brugen af Command, vil jeg starte med at få en ting på plads, nemlig de konstanter som benyttes i forbindelse med ADO. Konstanterne gør koden lettere og mere

intuitiv at læse.

Der er umiddelbart 3 muligheder for at få disse konstanter ind i koden (faktisk 4, men at sidde og definere dem selv, synes jeg er lidt omsonst, så den vil jeg ikke foreslå):

1. Du kan inkludere en fil (adovbs.inc) i den side som benytter ADO og dermed få adgang til konstanterne i den specifikke side. Dette synes jeg dog ikke specielt godt om, da man dels skal sørge for at den include er med hver gang man skal bruge ADO, dels lægger den en ekstra belastning på serveren, i og med den skal indlæses i den samlede side hver gang siden kaldes. Du kan læse lidt mere om denne metode her: http://msdn.microsoft.com/en-us/library/ms676526(vs.85).aspx

2. Et noget bedre alternativ er at erklære et METADATA-element i siden, som peger på TypeLibrary til ADO. På denne måde skal serveren ikke bruge kræfter på at indlæse en ekstra fil hver gang siden vises. Dog har denne metode stadig den svaghed, at man skal huske dette METADATA-element hver gang ADO skal bruges.

3. Den bedste løsning er efter min mening, at erklære førnævnte METADATA-element i applikationens global.asa-fil, da man så har adgang til ADO-konstanterne fra alle sider i applikationens scope. Applikationens scope omfatter altså alle filer og mapper (og undermapper) som ikke definerer en ny virtuel applikation. Virtuelle applikationer har deres eget scope og skal derfor erklære METADATA-elementet i sin egen global.asa for at få adgang til konstanterne.

METADATA-elementet ser således ud:

```
<!-- METADATA TYPE="typelib"
       uuid="00000205-0000-0010-8000-00AA006D2EA4"  -->
```

Igang med Command-objektet
Nu er vi klar til at kaste os over Command-udgaven af tidligere viste databaseopslag. Det kunne tage sig således ud:

```
dim sql, cmd, rs
dim kodeord, brugernavn

kodeord = request.querystring("kodeord")
brugernavn = request.querystring("brugernavn")

sql = "SELECT * " & _
      "FROM tabel " & _
      "WHERE kodeord = ? " & _
      "AND brugernavn = ?"
set cmd = Server.CreateObject("ADODB.Command")
set cmd.ActiveConnection = conn
cmd.CommandText = sql
cmd.CommandType = adCmdText
cmd.Parameters.Append cmd.CreateParameter("@kodeord", _
      adVarChar, adParamInput, 20, kodeord)
cmd.Parameters.Append cmd.CreateParameter("@brugernavn", _
      adVarChar, adParamInput, 20, brugernavn)
```

```
set rs = cmd.Execute()
'
' ... og resten af databaseoperationerne her!
'
```

Eller en quick and dirty udgave som kan bruges i mange tilfælde i stedet for ovenstående:

```
dim sql, cmd, rs
dim kodeord, brugernavn

kodeord = request.querystring("kodeord")
brugernavn = request.querystring("brugernavn")

sql = "SELECT * " & _
      "FROM tabel " & _
      "WHERE kodeord = ? " & _
      "AND brugernavn = ?"
set cmd = Server.CreateObject("ADODB.Command")
set cmd.ActiveConnection = conn
cmd.CommandText = sql
cmd.CommandType = adCmdText
set rs = cmd.Execute(, array(kodeord,brugernavn))
'
' ... og resten af databaseoperationerne her!
'
```

Denne metode kan som nævnt bruges i mange tilfælde og er noget mere kompakt og tilgængelig end versionen med eksplicitte parametererklæringer. Der kan dog være tilfælde hvor det er nødvendigt med mere kontrol over parametrenes typer og så må man "tilbage" til den lange udgave.

Regler der skal overholdes
Der er nogle ting man skal være opmærksom på i forbindelse med parameterisering via Command-objektet. Jeg vil prøve at komme ind på de vigtigste herunder.

Angivelse af parameterværdier (pladsholdere)
Der hvor man ønsker at indsætte en parameter i SQL-sætningen, placeres et spørgsmålstegn. Der skal ikke tages hensyn til typen af parameteren i SQL-sætningens syntaks, dvs. der skal ikke apostroffer (') omkring tekst-parametre eller hash (#) omkring datoer (i Access), da dette styres under oprettelsen af parameteren i command-objektets parameter-collection.

Sekvens af parametre
Det er vigtigt, at parametrene oprettes i command-objektetks parameter-collection, i den rækkefølge de forekommer i SQL-sætningen, da ADO ikke har nogen anden måde at koble parametre til SQL-sætningen end sekvensen (i og med pladsholdere for parametre udgøres af et spørgsmålstegn har de ikke nogen værdi i forhold til at identificere feltet).

Navngivning af parametre
Det er ikke så vigtigt hvad man kalder sine parametre, blot man sørger for at sende dem i den rækkefølge de er angivet i SQL-sætningen. For læsbarhedens og gennemskuelighedens skyld vil jeg da anbefale, at man benytter feltets navn til parameteren, så der er mindre tvivl om koblingen af en parameter til SQL-sætningen.

Typen af parametre
Det er vigtigt, at typen på parameteren stemmer overens eller er kompatibel med den type feltet har i databasen, da ADO ellers vil kvittere med en fejl. Ligeledes er det vigtigt, at typen på værdien af parameteren er kompatibel med den type, som parameteren er erlæret med. Hvis ikke den er det, vil ADO ligeledes kvittere med en fejl. Typiske fejl er integer-parametre som fødes med tomme strenge og dermed ikke er gyldige numeriske værdier, men også tomme dato'er eller strenge der er for lange til at passe ned i den definerede parameter giver problemer. Man slipper altså ikke for at validere brugerens data, selvom man benytter Command-objektet!

Understøttelse af parametre
Undestøttelsesgraden af parameterisering er forskellig fra "provider" til "provider" og man skal derfor undersøge, om den databasedriver man benytter, understøtter de features man benytter. Der findes muligheder for at spørge ADO om disse ting, men det er ikke noget jeg har benyttet mig af, da jeg altid har arbejdet i et homogent miljø og dermed altid har vidst hvad den driver jeg benytter kan og ikke kan. Derfor vil jeg henvise dig til MSDN eller Google for at finde mere avancerede features i forbindelse med ADO og Command-objektet.

Med disse ting på plads kan jeg prøve at demonstrere et par andre typiske databaseoperationer hvor Command-objektet er handy.

Oprettelse af nye data
Oprettelse af data i en tabel er et typisk scenarie, hvor brugeren er leverandør af data og et andet typisk eksempel på, hvor man skal være ekstra opmærksom på injectionforsøg.

En insert kunne håndteres således ved brug af command:

```
dim sql, cmd, recordCount
dim kodeord, brugernavn, email

kodeord = request.form("kodeord")
brugernavn = request.form("brugernavn")
email = request.form("email")

sql = "INSERT INTO tabel (kodeord, brugernavn, email) VALUES(?,?,?)"
set cmd = Server.CreateObject("ADODB.Command")
set cmd.ActiveConnection = conn
cmd.CommandText = sql
cmd.CommandType = adCmdText
cmd.Parameters.Append cmd.CreateParameter("@kodeord", _
      adVarChar, adParamInput, 20, kodeord)
cmd.Parameters.Append cmd.CreateParameter("@brugernavn", _
      adVarChar, adParamInput, 20, brugernavn)
cmd.Parameters.Append cmd.CreateParameter("@email", _
      adVarChar, adParamInput, 255, email)
```

```
recordCount = 0

cmd.Execute recordCount

if recordCount = 1 then
  response.write "Dine oplysninger er nu gemt."
else
  response.write "Dine oplysninger blev IKKE gemt."
  response.write " Forsøg evt. igen eller kontakt support."
end if
```

Som det ses, er der ikke meget forskel på hvordan Command-objektet benyttes mht. initiering og opsætning af parametre. Selve kaldet returnerer, i dette tilfælde, ikke noget brugbart recordset, så derfor undlader jeg at gemme det i en variabel. I stedet sender jeg en parameter med til cmd.Execute for at få oplysninger om, hvormange rækker det blev påvirket af min INSERT. Dette benytter jeg til at sandsynliggøre, at indsættelsen gik godt. I dette tilfælde skal recordsCount gerne indholde 1, da der oprettes netop én ny række i tabellen.

Samme mønster vil gøre sig gældende for UPDATE og DELETE.

Stored Procedues
Hidtil har jeg kun behandlet tekstbaserede commands, men hvis jeg har en stored procedure, som skal udføres, er opskriften nogenlunde den samme, blot med den ændring, at cmd.CommandType skal sættes til adCmdStoredProc i stedet for adCmdText.

Eksempel, hvor jeg har en stored procedure som sletter en bruger fra databasen og til dette formål tager den brugerens navn som parameter:

```
sql = "spSletBruger"
set cmd = Server.CreateObject("ADODB.Command")
set cmd.ActiveConnection = conn
cmd.CommandText = sql
cmd.CommandType = adCmdStoredProc
cmd.Parameters.Append cmd.CreateParameter("@brugernavn", _
      adVarChar, adParamInput, 20, brugernavn)

recordCount = 0

cmd.Execute recordCount

if recordCount = 1 then
  response.write "Dine oplysninger er nu slettet."
else
  response.write "Dine oplysninger blev IKKE slettet."
  response.write " Forsøg evt. igen eller kontakt support."
end if
```

Som med INSERT-eksemplet er der heller ikke umiddelbart noget brugbart resultat efter denne operation, så jeg vælger bare at aflæse antallet af rækker som blev påvirket af denne handling.

Genbrug af command-objektet
Skulle man have behov for at udføre den samme databaseoperation flere gange efter hinanden, kan man med fordel benytte det samme command-objekt og blot udskifte parametrenes værdier (det er vel en af de andre vigtige årsager til at benytte command-objektet, da man på denne måde kan vinde noget performance i og med man slipper for at oprette og nedlægge command-objekter for hver iteration). Det kunne tage sig nogenlunde således ud:

```
dim sql, cmd, recordCount
dim arrBrugere, bruger

arrBrugere = array( _
        array("bruger1","kode1","mail1@example.dk"), _
        array("bruger2","kode2","mail2@example.dk"), _
        array("bruger3","kode3","mail3@example.dk"), _
        array("bruger4","kode4","mail4@example.dk"), _
        array("bruger5","kode5","mail5@example.dk") _
      )

sql = "INSERT INTO tabel (kodeord, brugernavn, email) VALUES(?,?,?)"
set cmd = Server.CreateObject("ADODB.Command")
set cmd.ActiveConnection = conn
cmd.CommandText = sql
cmd.CommandType = adCmdText
cmd.Parameters.Append cmd.CreateParameter("@kodeord", _
      adVarChar, adParamInput, 20)
cmd.Parameters.Append cmd.CreateParameter("@brugernavn", _
      adVarChar, adParamInput, 20)
cmd.Parameters.Append cmd.CreateParameter("@email", _
      adVarChar, adParamInput, 255)

for each bruger in arrBrugere
  cmd.Parameters("@kodeord").Value = bruger(1)
  cmd.Parameters("@brugernavn").Value = bruger(0)
  cmd.Parameters("@email").Value = bruger(2)

  recordCount = 0

  cmd.Execute recordCount

  if recordCount = 0 then
    response.write "Oplysninger om " & bruger(0)
    response.write " kunne IKKE gemmes!"
  end if
next
```

Lidt dovenskab skader vel ikke
Som mange nok har luret, er selve oprettelsen af command-objektet en relativ triviel opgave og der kan spares en del liniers kode (set på applikationsplan), hvis man lige pakker den del lidt ind. Dette har jeg lavet et lille eksempel på herunder.

```
function CreateTextCommand(conn, sql)
  dim cmd
  set cmd = Server.CreateObject("ADODB.Command")
  set cmd.ActiveConnection = conn
  cmd.CommandText = sql
  cmd.CommandType = adCmdText
  set CreateTextCommand = cmd
end function
```

Funktionen opretter, ikke overraskende, et command-objekt og returnerer dette til den kaldende part. Den kaldende part sparer altså tre liniers triviel kode hver gang, hvilket dels gør tingene hurtigere at kode, men også nemmere at overskue (når først man har lært hvad funktionen CreateTextCommand gør ;-)).

Funktionen kan bruges således i praksis (her anvendes quick and dirty-metoden også for at demonstrere hvordan dette kan se ud i en opdateringssituation):

```
dim sql, cmd, recordCount
dim kodeord, brugernavn, email

kodeord = request.form("kodeord")
brugernavn = request.form("brugernavn")
email = request.form("email")

sql = "INSERT INTO tabel (kodeord, brugernavn, email) VALUES(?,?,?)"
set cmd = CreateTextCommand(conn, sql)

recordCount = 0

cmd.Execute recordCount, array(kodeord, brugernavn, email)

if recordCount = 1 then
  response.write "Dine oplysninger er nu gemt."
else
  response.write "Dine oplysninger blev IKKE gemt."
  response.write " Forsøg evt. igen eller kontakt support."
end if
```

Konklusion
Nu er command-objektets mest basale muligheder introduceret og der er rig mulighed for at udbygge sin viden om de features som stilles til rådighed af ADO.

Du kan også se, at der ikke umiddelbart er mulighed for at SQL-injicere når der benyttes parametre sammen med Command-objektet, da mulighederne for at snyde med formatet af input, så de påvirker den genererede SQL-sætning, er fjernet.

Kravet om at validere input består (som det også gør med strengsammensætningsmetoden), da det stadig kan opstå fejl, hvis data har forkert format i forhold til parametertypen. Fejlhåndtering kan være med til at afbøde dette lidt, men det er som udgangspunkt bedre, at sikre datavaliditet frem for at fejlhåndtere sig

ud af datainvaliditet.

Min egen primære kilde til at undersøge mulighederne ved ADO er MSDN, så derfor vil jeg referere til følgende:

http://msdn.microsoft.com/en-us/library/ms675532(VS.85).aspx som er programmørens guide og referencemanualen til ADO.

http://msdn.microsoft.com/en-us/library/ms676571(VS.85).aspx som er en lille samling af sider der beskriver hvordan man benytter Command-objektet.

Der er ingen tvivl om at der findes meget mere om emnet rundt omkring, men mine egne behov er typisk af opslagskarakter, så derfor er det ovenstående materialer jeg benytter mest.

Kommentar til kommentar til artiklen
arne_v >> Naturligvis! En rigtig vigtig pointe hvis man arbejder med batchoperationer. Jeg har tilføjet et afsnit om dette ("Genbrug af command-objektet").

thesurfer >> Jeg er enig i at man skal beskytte sine kodefiler og vil i princinppet også vælge at beskytte mine filer. Dog er den refererede inc-fil public domain, så den er der nok ikke så stor risiko ved at lade folk hente (hvis bare man lader være med at tilføje sin egen kode i den :-)).

erikjacobsen >> Enig. Jeg har desværre kun kendskab til .NET ud over ASP, men vil da gerne bekræfte at det også her kan lade sig gøre. Det kan være der herfra kommer en artikel om det på et tidspunkt (hvis der ikke er nogen som kommer mig i forkøbet... UPS! Jeg kan se der allerede er lavet et par stykker af arne_v :-)).

Kommentar af erikjacobsen d. 20. Nov 2008 | 1

Og man kan tilføje, at i alle andre frameworks og lignende, findes en tilsvarende mulighed.

Kommentar af thesurfer d. 20. Nov 2008 | 2

Meget god artikel. Men man må aldrig bruge ".inc"-filer (f.eks. adovbs.inc), da disse filer er ren tekst som ikke bliver afviklet. Dvs, hvis man kan gætte filnavnet (f.eks. hvis man kan genkende systemet/forum/osv), kan man få fat på indholdet af ".inc"-filen (database-oplysninger, brugernavn, kodeord osv).. Brug heller sprogets eget filtype (f.eks. adovbs.asp) og definer variablerne der.

Kommentar af arne_v d. 24. Nov 2008 | 3

Glimrende. Man kan også assigne værdier til parametrene efter at man har created og appended dem.

Kommentar af trer d. 07. Dec 2008 | 4

Fin artikel.

Kommentar af windcape d. 22. Nov 2008 | 5

Kommentar af erizias (nedlagt brugerprofil) d. 07. Jul 2011 | 6

Kanon artikel, dog savner jeg lidt "hvad gør denne, og hvad gør denne" :)

F.eks. hvordan følgende 2 statements kan være lig hinanden:

```
  cmd.Parameters.Append cmd.CreateParameter("@kodeord", adVarChar, adParamInput, 20, kodeord)
  cmd.Parameters.Append cmd.CreateParameter("@brugernavn", adVarChar, adParamInput, 20,
brugernavn)
  set rs = cmd.Execute()
```

OG

```
  set rs = cmd.Execute(, array(kodeord,brugernavn))
```

For mig at se, får den nederste metode jo ikke at vide om det er varchar, ints eller lignende samt hvor mange tegn det indeholder, som den øverste metode gør :)
Det savner jeg lidt i artiklen, men ellers en kanon artikel som fortæller om grundprincipperne af parameteriseringen :)