Dinamički SQL

Dinamički SQL se obično koristi kada je potrebno kreirati i izvršiti deo koda koji zavisi od unosa korisnika. Za tu namenu SQL Server obezbeđuje dve komande koje pozivaju programski kod koji se nalazi u stringu, a to su EXEC (odnosno EXECUTE) i sp_executesql.

Po pravilu, preporučeno je da se koristi sp_executesql zato što ima interfejs (odnosno, ulazne i izlazne parametre) pa je samim tim otporniji na SQL Injection napade. Takođe, ta procedura omogućava veću efikasnost ukoliko se jedan string izvršava više puta, jer dolazi do ponovo iskorišćavanja plana izvršavanja. Sa druge strane, EXEC iako manje bezbedan, ponekad fleksibilniji.

Važni aspekti pri upotrebi dinamičkog SQL su:

1. Dinamički SQL zahteva da korisnik koji izvršava kod ima direktne privilegije da ga izvrši, iako je kod unutar neke uskladištene procedure (drugim rečima, iako korisnik ima execute privilegiju nad procedurom koja u sebi sadrži dinamički SQL, korisnik ipak mora ima direktne privilegije nad objektima koji se koriste u tom dinamičkom SQL kodu). Međutim, ovo se može zaobići ukoliko se koristi impersonifikacija koristeći klauzulu EXECUTE AS.
2. Dinamički SQL se izvršava kao zaseban batch, koji je odvojen od pozivajućeg batch. To znači da se dinamički batch parsira, razrešava i optimizuje kao zasebna jedinica, a samim tim postaje posebna jedinica kompilacije.
3. Sva podešavanja okruženja koja važe za pozivajući batch (npr. kontekst baze koja se koristi, opcije sesije, pa čak i privremene tabele) važiće i unutar dinamičkog batch. Međutim, sve promene koje se odnose na okruženje, a izvrše se unutar dinamičkog batch (npr. privremen tabela napravljena unutar dimamičkog batch nestaje nakon što se on izvrši), nisu vidljive i nemaju efekta u pozivajućem batch.
4. Promenljive napravljene u pozivajućem batch, nisu vidljive unutar dinamičkog batch.

EXEC

Na primer, ukoliko imamo više tabele Orders, tj. za svaku godinu po jednu, onda bi kreiranje nove tabele za tekuću godinu moglo da bude ovako urađeno.

```
DECLARE @year AS CHAR(4) = CAST(YEAR(CURRENT_TIMESTAMP) AS CHAR(4));
EXEC('IF OBJECT_ID(''dbo.Orders' + @year + ''', ''U'') IS NOT NULL
DROP TABLE dbo.Orders' + @year + ';
CREATE TABLE dbo.Orders' + @year + '(orderid INT PRIMARY KEY);');
```

Što bi u slučaju da je tekuća godina 2018, kreiralo kod:

```
IF OBJECT_ID('dbo.Orders2018', 'U') IS NOT NULL
DROP TABLE dbo.Orders2018;
CREATE TABLE dbo.Orders2018(orderid INT PRIMARY KEY);
```

Da bi se selektovao sadržaj te tabele, bio bi potreban sledeći kod:

```
DECLARE @year AS CHAR(4) = CAST(YEAR(CURRENT_TIMESTAMP) AS CHAR(4));
EXEC('SELECT orderid FROM dbo.Orders' + @year + ';');
```

Važno je primetiti da se EXEC komandi sme proslediti samo string, tako da poyivi funkcija nisu dozvoljeni.

Na primer, sledeći poziv bi bio neuspešan:

```
EXEC('SELECT orderid FROM dbo.Orders'
+ CAST(YEAR(CURRENT_TIMESTAMP) AS CHAR(4)) + ';');
```

Odnosno, bilo bi neophodno uraditi sledeće:

```
DECLARE @sql AS VARCHAR(500) =
'SELECT orderid FROM dbo.Orders'
+ CAST(YEAR(CURRENT_TIMESTAMP) AS CHAR(4)) + ';';

EXEC(@sql);
```

Takođe, s obzirom da EXEC nema interfejs, već je jedini ulazni parametar string, to znači da dinamički batch nema pristup lokalnim promenljivim. Tako bi, na primer, sledeći kod prijavio grešku:

```
DECLARE @lastname AS NVARCHAR(40) = N'Davis';
DECLARE @sql AS NVARCHAR(500) =
N'SELECT empid, firstname, lastname
FROM HR.Employees
WHERE lastname = @lastname;';
EXEC(@sql);
```

Dakle, da bi se koristila promenljiva, njen sadržaj mora biti konkateniran:

```
DECLARE @lastname AS NVARCHAR(40) = N'Davis';
DECLARE @sql AS NVARCHAR(500) =
N'SELECT empid, firstname, lastname
FROM HR.Employees
WHERE lastname = ' + QUOTENAME(@lastname, N'''') + N';';
EXEC(@sql);
```

Funkcija QUOTENAME je jedan od načina za odbranu od SQL Injection napada. Ta funkcija govori čime treba da bude ograđen sadržaj promenljive (npr. jednostruki znaci navoda, dvostruki znaci navoda, ili ono što je podrazumevano, a to su uglaste zagrade).

Ukoliko bi dinamički SQL trebao da kreira povratnu vrednost, onda bi to moglo da bude urađeno:

```
DECLARE @sql AS VARCHAR(500) =
'DECLARE @result AS INT = 42;
SELECT @result AS result;';
EXEC(@sql);
```

Ali bi dohvatanje povratne vrednosti moralo da ide preko privremene tabele:

```
DECLARE @sql AS VARCHAR(500) =
'DECLARE @result AS INT = 42;
SELECT @result AS result;';
DECLARE @myresult AS INT;
CREATE TABLE #T(result INT);
INSERT INTO #T(result) EXEC(@sql);
SET @myresult = (SELECT result FROM #T);
SELECT @myresult AS result;
DROP TABLE #T;
```

Takođe, moglo je prvo da se kreira privremena tabele, koja bi se, pošto je vidljiva unutar dinamičkog batch, mogla popunjvati direktno iz njega:

```
DECLARE @sql AS VARCHAR(500) =
'DECLARE @result AS INT = 42;
INSERT INTO #T(result) VALUES(@result);'
DECLARE @myresult AS INT;
CREATE TABLE #T(result INT);
EXEC(@sql);
SET @myresult = (SELECT result FROM #T);
SELECT @myresult AS result;
DROP TABLE #T;
```

sp_executesql

Uskladištena procedura za izvršavanje dinamičkog SQL se pojavila kasnije u odnosu na komandu EXEC, i to uglavnom sa razlogom da obezbedi bolju podršku za ponovnu upotrebu planova izvršavanja.

Za razliku od EXEC, procedura sp_executesql ima interfejs koji podržava i ulazne i izlazne parametre. Ovo omogućava kreiranje upita sa argumentima.

Sintaksa sp_executesql glasi:

```
EXEC sp_executesql
@stmt = <statement>, -- kao telo procedute
@params = <params>, -- kao deklaracija parametara procedure
{<params assignment> [,]} -- kao poziv procedure, svaki parametar po jedan argument
```

Za isti primer kao kod EXEC komande, kod bi izgledao ovako:

```
DECLARE @mylastname AS NVARCHAR(40) = N'Davis';
DECLARE @sql AS NVARCHAR(500) =
N'SELECT empid, firstname, lastname
FROM HR.Employees
WHERE lastname = @lastname';
EXEC sp_executesql
@stmt = @sql,
@params = N'@lastname AS NVARCHAR(40)',
@lastname = @mylastname;
```

Dakle, u ovom slučaju umesto da se radi konkatenacija sadržaja promenljive, ovaj kod definiše ulazni parametar.

Druga važna mogućnost prilikom upotrebe sp_executesql odnosi se na podršku za izlazne parametre. Na primer:

```
DECLARE @sql AS NVARCHAR(500), @myresult AS INT
SET @sql = N'SET @result = 42;';
EXEC sp_executesql
@stmt = @sql,
@params = N'@result AS INT OUTPUT',
@result = @myresult OUTPUT;
SELECT @myresult;
```

SQL Injection

Jedan od najvećih sigurnosnih rizika prilikom upotrebe dinamičkog SQL jesu SQL Injection napadi. Napadi mogu biti zasnovani na strani klijenta ili na strani servera.

Na primer, ukoliko bi postojao login ekran, na kome se od korisnika očekuje da unese korisničko ime i lozinku, pa da aplikacija te unete vrednosti prosledi do dinamičkog SQL radi autentifikacije, može doći do SQL Injection napada (zasnovanog na klijentskoj strani).

Neka je kreirana tabela:

```
IF OBJECT_ID('dbo.Users', 'U') IS NOT NULL DROP TABLE Users;
CREATE TABLE dbo.Users
(
username VARCHAR(30) NOT NULL PRIMARY KEY,
pass VARCHAR(16) NOT NULL
);
INSERT INTO Users(username, pass) VALUES('user1', '123');
INSERT INTO Users(username, pass) VALUES('user2', '456');
```

Neka se na primer za proveru autentifikacije koristi upit:

```
sql = "SELECT COUNT(*) AS cnt FROM dbo.Users WHERE username = '" _
& InputUserName & "' AND pass = '" & InputPass & "';"
```

Kada korisnik unese:

```
InputUserName = "user1"
InputPass = "123"
```

Ono što nastane je:

```
SELECT COUNT(*) AS cnt FROM dbo.Users WHERE username = 'user1' AND pass = '123';
```

Međutim, ukoliko bi korisnik uneo:

```
InputUserName = "' OR 1 = 1 --"
InputPass = ""
```

Nastao bi kod:

```
SELECT COUNT(*) AS cnt FROM dbo.Users WHERE username = '' OR 1 = 1 --' AND pass = '';
```

Ovaj kod zbog dodatog izraza 1=1 -- koji predstavlja uslov koji je uvek ispunjen, a iza njega je sve postalo komentar zbog upotrebe --.

Slično, SQL Injection napad, takođe, može biti zasnovan na osnovu koda koji se kreira na strani servera. Na primer, ukoliko bi se SQL Serveru i određenoj proceduri prosleđivao argument u vidu jednog stringa, a koji treba da sadrži niz argumenata razdvojenih zapetama, onda bi to moglo da izgleda ovako:

```
IF OBJECT_ID('dbo.GetOrders', 'P') IS NOT NULL DROP PROC dbo.GetOrders;
GO

CREATE PROC dbo.GetOrders
@orders AS VARCHAR(1000)
AS

DECLARE @sql AS NVARCHAR(1100);

SET @sql = N'SELECT orderid, shipcountry
FROM Sales.Orders
WHERE orderid IN(' + @orders + ');';

EXEC sp_executesql @sql;
GO
```

Pravilna upotreba procedure sa:

```
EXEC dbo.GetOrders '10248,10249,10250';
```

Bi kao rezultat dala:

```
orderid     shipcountry
---------   ---------------
10248       France
10249       Germany
10250       Brazil
```

Međutim, ukoliko bi poziv bio sa:

```
EXEC dbo.GetOrders ' --';
```

To bi kao rezultat dalo kod:

```
SELECT orderid, shipcountry
FROM Sales.Orders
WHERE orderid IN( --);
```

Odnosno poruku o grešci, koja ukoliko bude kao takva vraćena korisniku (hakeru), otkriva strukturu upita koji je upravo izvršen, odnosno da se radi o dinamičkom SQL kodu koji konkatenira ulazne vrednosti.

```
Msg 102, Level 15, State 1, Line 3
Incorrect syntax near '('.
```

Korisnik (haker) bi sledećim pozivom:

```
EXEC dbo.GetOrders '-1)
UNION ALL
SELECT object_id, QUOTENAME(SCHEMA_NAME(schema_id)) + ''.'' + QUOTENAME(name)
FROM sys.objects --';
```

Doveo do toga da procedura izvrši sledeći kod:

```
SELECT orderid, shipcountry
FROM Sales.Orders
WHERE orderid IN(-1)
UNION ALL
SELECT object_id, QUOTENAME(SCHEMA_NAME(schema_id)) + '.' + QUOTENAME(name)
FROM sys.objects --);
```

Rezultat tog koda bi mogao da bude:

```
orderid     shipcountry
---------   ------------------------------
...
149575571   [HR].[Employees]
165575628   [HR].[PK_Employees]
181575685   [HR].[FK_Employees_Employees]
197575742   [HR].[CHK_birthdate]
213575799   [Production].[Suppliers]
229575856   [Production].[PK_Suppliers]
389576426   [Sales].[Customers]
405576483   [Sales].[PK_Customers]
421576540   [Sales].[Shippers]
437576597   [Sales].[PK_Shippers]
453576654   [Sales].[Orders]
...
```

Odnosno, dalje bi unosom:

```
EXEC dbo.GetOrders '-1)
UNION ALL
SELECT column_id, name
FROM sys.columns
WHERE object_id = 389576426 --';
```

Nastao kod:

```
SELECT orderid, shipcountry
FROM Sales.Orders
WHERE orderid IN(-1)
UNION ALL
SELECT column_id, name
FROM sys.columns
WHERE object_id = 389576426 --);
```

Koji kao rezultat, na primer, vraća:

```
orderid     shipcountry
--------    -----------------
1           custid
2           companyname
3           address
4           city
5           region
6           postalcode
7           country
8           phone
9           fax
```

Potom unosom:

```
EXEC dbo.GetOrders '-1)
UNION ALL
SELECT custid, companyname + '';'' + phone
FROM Sales.Customers --';
```

Nastaje kod:

```
SELECT orderid, shipcountry
FROM Sales.Orders
WHERE orderid IN(-1)
UNION ALL
SELECT custid, companyname + ';' + phone
FROM Sales.Customers --);
```

Što daje rezultat:

```
orderid     shipcountry
----------  ------------------------------------
1 Customer  NRZBB;030-3456789
2 Customer  MLTDN;(5) 789-0123
3 Customer  KBUDE;(5) 123-4567
4 Customer HFBZG;(171) 456-7890
```

Odbrana od SQL Injection napada

Sledeće mere mogu samo obezbediti neki nivo zaštite, ali nikako kompletnu zaštitu od SQL Injection napada:

1. Kako bi se smanjila ranjivost, potrebno je isključiti funkcionalnosti koje nisu potrebne, kao što su izvršavanje xp_cmdshell, ili SQL Server Agent, itd.
2. Korisniku koji izvršava kod, uvek davati minimalne privilegije. Konkretno, u primeru sa login ekranom, korisnik bi trebalo da ima privilegije samo da čita tabelu Users, ništa više od toga, čime bi haker bio sprečen da radi modifikacije podataka, ali bi ih i dalje mogao da pročita što i dalje predstavlja sigurnosni problem.
3. Uvek proveravati ulazne podatke koji dolaze od korisnika. Konkretno, u primeru sa listom porudžbina, sadržaj ulaza bi trebalo da budu samo brojevi i zapete, a ne i ostali karakteri.
4. Dužina ulaznih podataka treba da bude samo onoliko koliko je zaista neophodno (kako ne bi mogao da se smesti duži maliciozni kod). Konkretno, u primeru sa login ekranom, za podatke o korisničkom imenu i lozinci, nema potrebe dozvoljavati da podaci budu dugački stotinama karaktera.
5. Koristiti uskladištene procedure, koje zbog provere tipova ulaznih parametara (npr. celobrojne vrednosti, datum itd.) i mogućnosti podešavanja privilegija mogu obezbediti sveći nivo sigurnosti.
6. Izbegavati dinamički SQL kad god je to moguće, jer statički SQL je daleko sigurniji, a naročito ukoliko se pažnja posveti sigurnosni aspektima.
7. Kad god je potrebno da se ulazni podaci stave pod navodnike, to ne treba raditi eksplicitno, već upotrebom QUOTENAME funkcijom

Na primer (namerno je upotrebljen PRINT, a ne EXEC, kako bi se pokazalo šta bi bio rezultujući upit):

```
DECLARE @lastname AS NVARCHAR(40), @sql AS NVARCHAR(200);
SET @lastname = N'Davis';
SET @sql = N'SELECT * FROM HR.Employees WHERE lastname = N'''
+ @lastname + ''';';
PRINT @sql;
```

Za unos Devis, daje:

```
SELECT * FROM HR.Employees WHERE lastname = N'Davis';
```

Ali ukoliko bi haker pokusao unos:

```
DECLARE @lastname AS NVARCHAR(40), @sql AS NVARCHAR(200);
SET @lastname = N''' DROP TABLE HR.Employees --';
SET @sql = N'SELECT * FROM HR.Employees WHERE lastname = N'''
+ @lastname + ''';';

PRINT @sql;
```

To bi kao rezultat dalo:

```
SELECT * FROM HR.Employees WHERE lastname = N'' DROP TABLE HR.Employees --';
```

Međutim, ukoliko bi se koristila QUOTENAME funkcija:

```
DECLARE @lastname AS NVARCHAR(40), @sql AS NVARCHAR(200);
SET @lastname = N''' DROP TABLE HR.Employees --';
SET @sql = N'SELECT * FROM HR.Employees WHERE lastname = N'
+ QUOTENAME(@lastname, '''') + ';';
PRINT @sql;
```

Onda bi rezultat bio potpuno bezbedan upit:

```
SELECT * FROM HR.Employees WHERE lastname = N''' DROP TABLE HR.Employees --';
```