SQL Injections by truncation

Bala Neerumalla
Microsoft

Introduction

- Who am I?
  - Security Engineer at Microsoft
  - Worked on SQL Server 2000 SP3 and SP4
  - Worked on SQL Server 2005
  - Working in Exchange Hosted Services
- Why am I here?
  - Talk about new vulnerabilities we encountered
  - Talk about mitigation techniques

Agenda

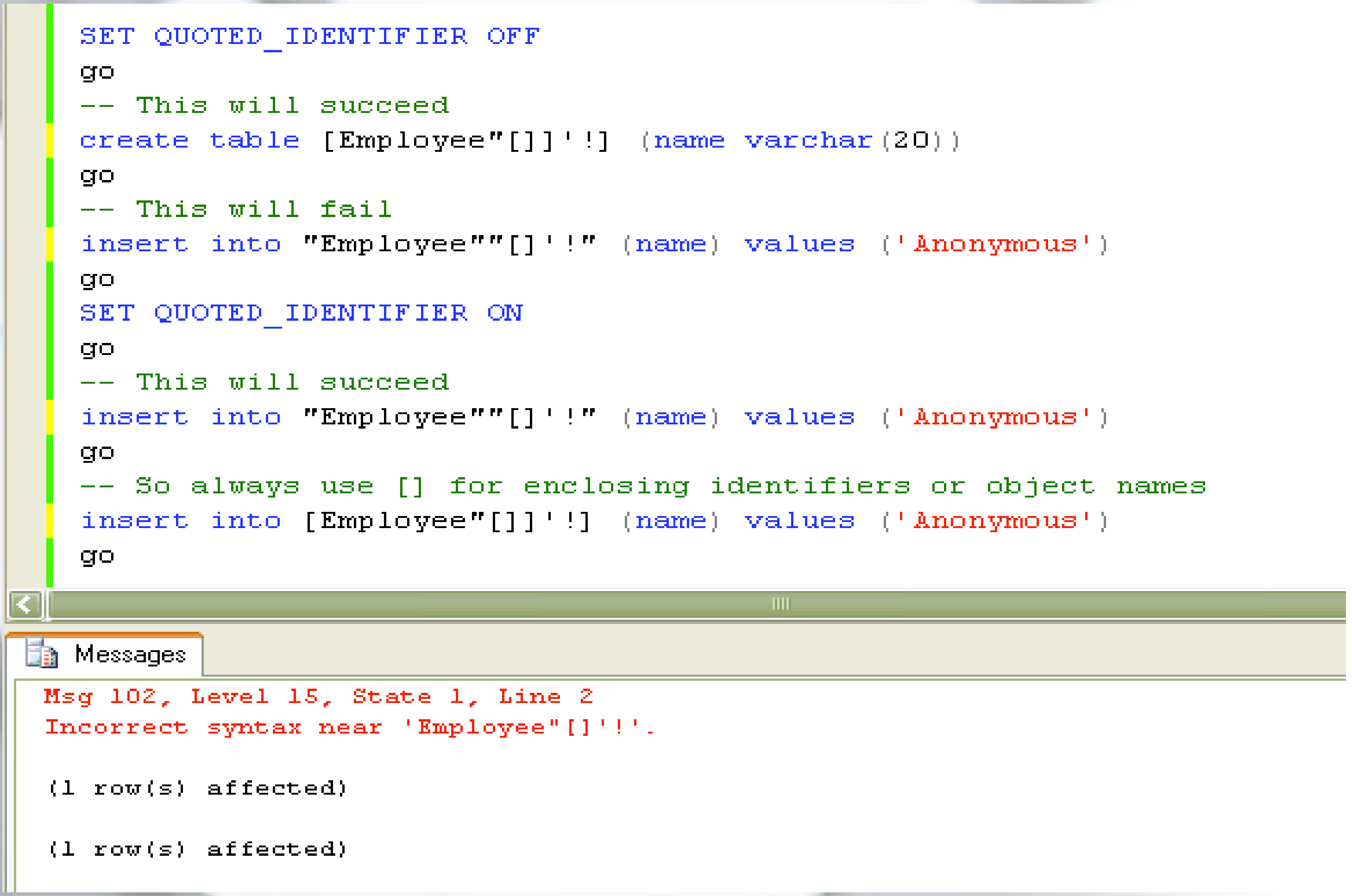
- Best practices for constructing dynamic TSQL
  - Delimiting Identifiers and Character Strings
  - SQL functions
- Truncation Issues
  - SQL modification by truncation
  - SQL injection by truncation
  - Finding and Mitigating truncation issues

Best practices for constructing dynamic TSQL

Delimiting database object names

- Use delimited Identifiers
  - When reserved words are used for object names.
  - When you are using characters that are not listed as qualified identifiers
- Double quotes can be used to delimit identifiers based on where QUOTED_IDENTIFIER is ON or OFF.
- Never use single quotes to delimit identifiers.
- Always use square brackets (‘[‘ and ‘]’) to delimit identifiers.
- Double up all occurrences of right square brackets (]) in the object name.

Create a table with name Employee”[]’!



```
SET QUOTED_IDENTIFIER OFF
go
-- This will succeed
create table [Employee"[]]'!] (name varchar(20))
go
-- This will fail
insert into "Employee""[]'!" (name) values ('Anonymous')
go
SET QUOTED_IDENTIFIER ON
go
-- This will succeed
insert into "Employee""[]'!" (name) values ('Anonymous')
go
-- So always use [] for enclosing identifiers or object names
insert into [Employee"[]]'!] (name) values ('Anonymous')
go
```

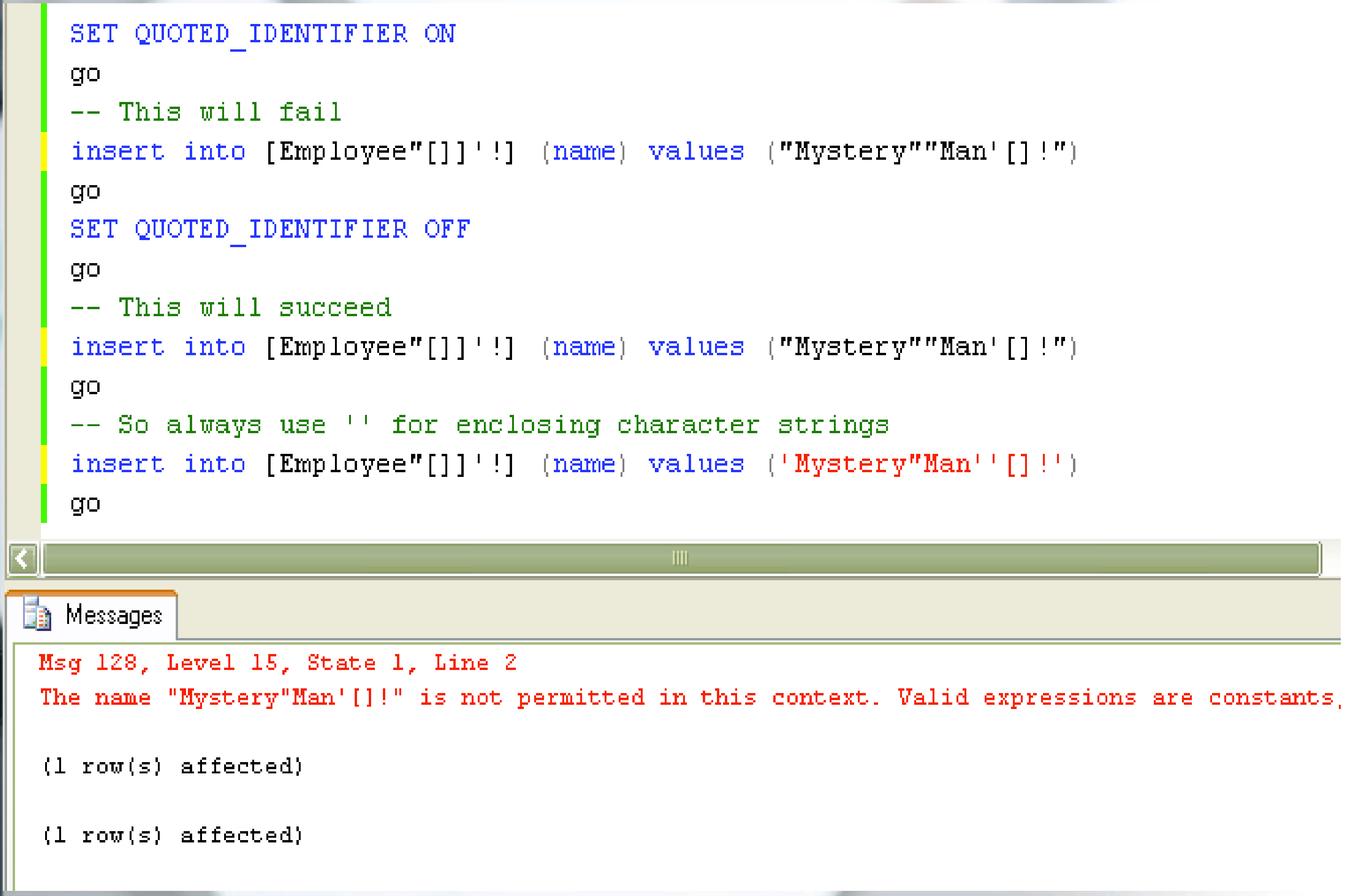
Messages

```
Msg 102, Level 15, State 1, Line 2
Incorrect syntax near 'Employee"[]'!'.

(1 row(s) affected)

(1 row(s) affected)
```

Delimiting character strings

- Double quotes can be used to delimit character strings based on where QUOTED_IDENTIFIER is OFF or ON.
- Always use single quotes to delimit character strings.
- Double up all occurrences of single quotes in the character strings.

Insert the name Mystery"Man'[]!



```
SET QUOTED_IDENTIFIER ON
go
-- This will fail
insert into [Employee"[]]'!] (name) values ("Mystery""Man'[]!")
go
SET QUOTED_IDENTIFIER OFF
go
-- This will succeed
insert into [Employee"[]]'!] (name) values ("Mystery""Man'[]!")
go
-- So always use '' for enclosing character strings
insert into [Employee"[]]'!] (name) values ('Mystery"Man''[]!')
go
```

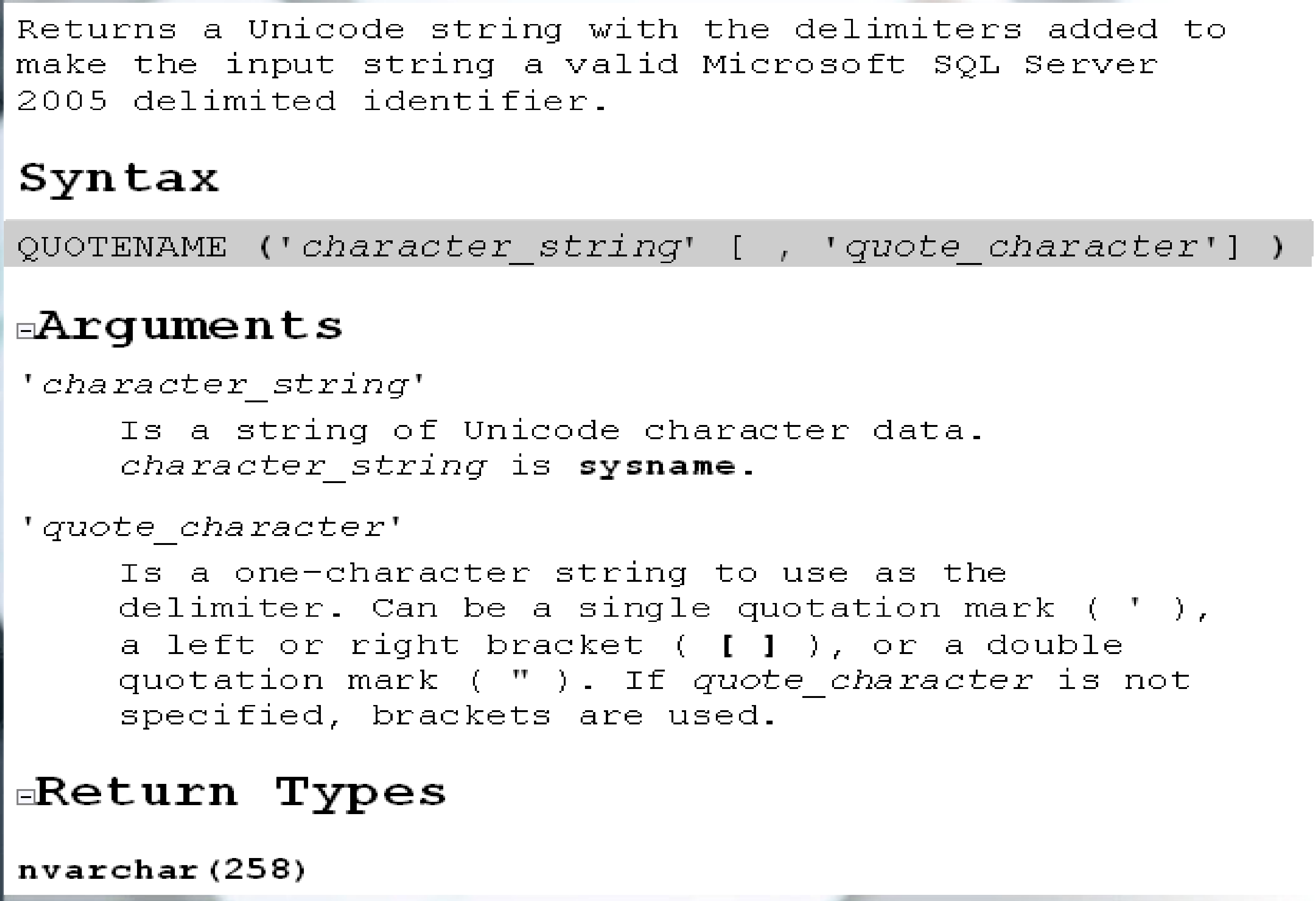
Messages

```
Msg 128, Level 15, State 1, Line 2
The name "Mystery"Man'[]!" is not permitted in this context. Valid expressions are constants,

(1 row(s) affected)

(1 row(s) affected)
```

SQL Functions

- quotename()
- replace()

quotename() function



Returns a Unicode string with the delimiters added to make the input string a valid Microsoft SQL Server 2005 delimited identifier.

Syntax

```
QUOTENAME ('character_string' [ , 'quote_character'] )
```

Arguments

'*character_string*'

Is a string of Unicode character data. *character_string* is **sysname**.

'*quote_character*'

Is a one-character string to use as the delimiter. Can be a single quotation mark ('), a left or right bracket (**[ ]**), or a double quotation mark ("). If *quote_character* is not specified, brackets are used.

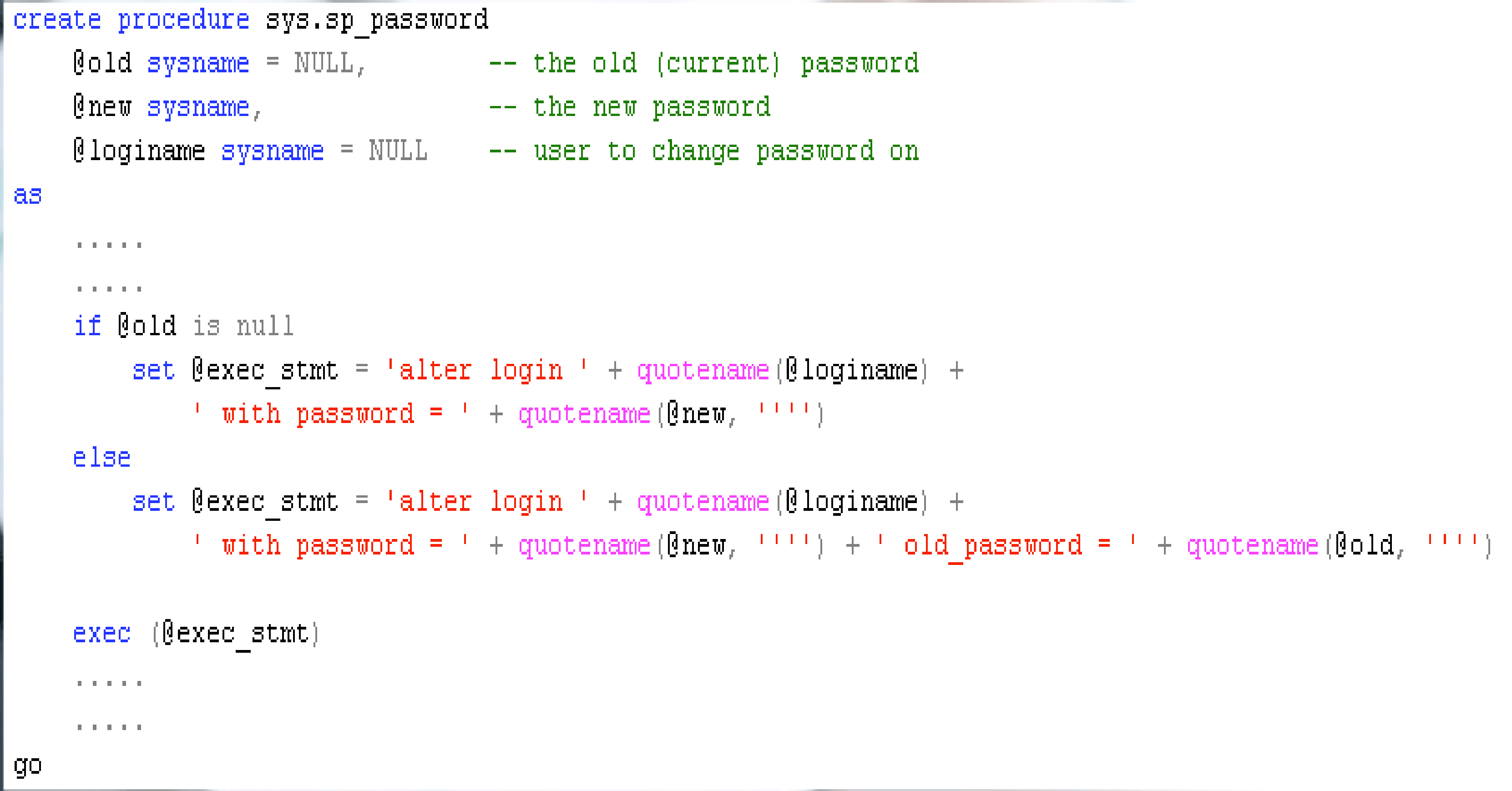
Return Types

nvarchar(258)

Delimiting object names with quotename()

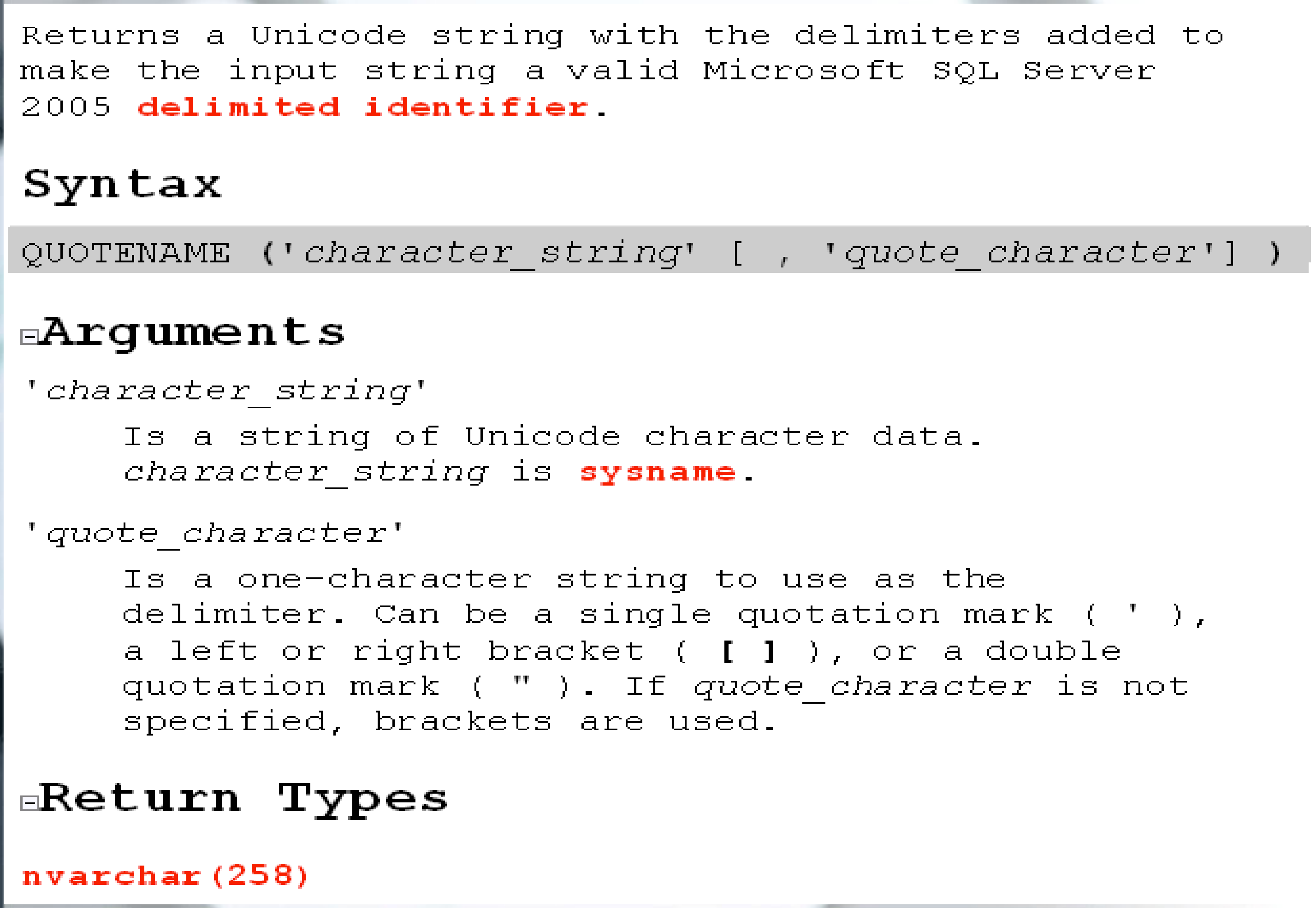
```
create procedure sys.sp_droplogin
    @loginame sysname
as


    ......
    ......
    set @exec_stmt = 'drop login ' + quotename(@loginame)
    exec (@exec_stmt)

    if @@error <> 0
        return (1)

    -- SUCCESS MESSAGE --
    return (0)  -- sp_droplogin
go
```

Delimiting character strings with quotename()



```
create procedure sys.sp_password
    @old sysname = NULL,        -- the old (current) password
    @new sysname,               -- the new password
    @loginame sysname = NULL    -- user to change password on
as
    .....
    .....
    if @old is null
        set @exec_stmt = 'alter login ' + quotename(@loginame) +
            ' with password = ' + quotename(@new, '''')
    else
        set @exec_stmt = 'alter login ' + quotename(@loginame) +
            ' with password = ' + quotename(@new, '''') + ' old_password = ' + quotename(@old, '''')

    exec (@exec_stmt)
    .....
    .....
go
```

quotename() function



Returns a Unicode string with the delimiters added to make the input string a valid Microsoft SQL Server 2005 **delimited identifier**.

Syntax

```
QUOTENAME ('character_string' [ , 'quote_character'] )
```

Arguments

'character_string'

Is a string of Unicode character data. *character_string* is **sysname**.

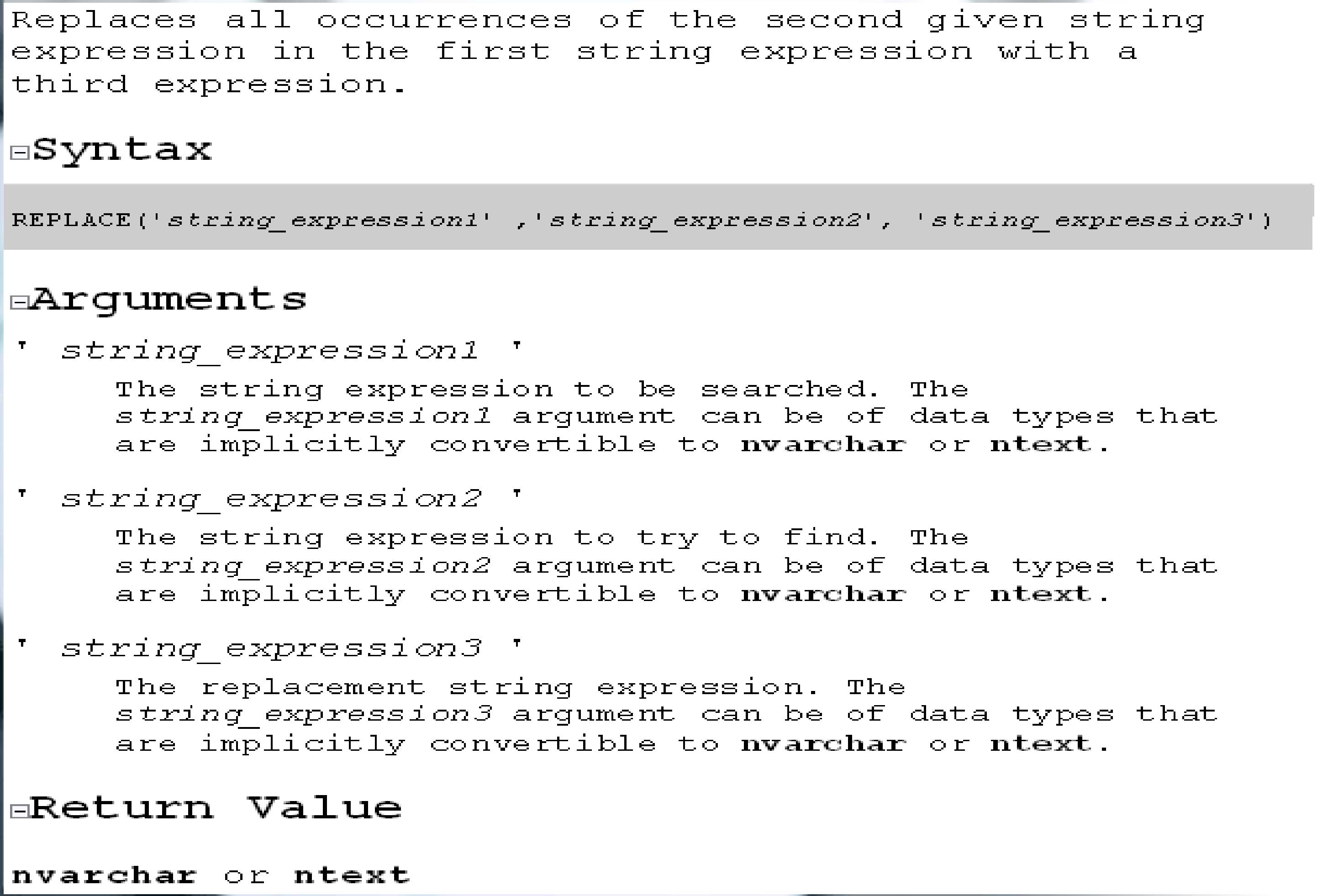
'quote_character'

Is a one-character string to use as the delimiter. Can be a single quotation mark ('), a left or right bracket (**[ ]**), or a double quotation mark ("). If *quote_character* is not specified, brackets are used.

Return Types

nvarchar(258)

replace() Function



Replaces all occurrences of the second given string expression in the first string expression with a third expression.

Syntax

```
REPLACE('string_expression1' ,'string_expression2', 'string_expression3')
```

Arguments

- *string_expression1*

  The string expression to be searched. The *string_expression1* argument can be of data types that are implicitly convertible to **nvarchar** or **ntext**.

- *string_expression2*

  The string expression to try to find. The *string_expression2* argument can be of data types that are implicitly convertible to **nvarchar** or **ntext**.

- *string_expression3*

  The replacement string expression. The *string_expression3* argument can be of data types that are implicitly convertible to **nvarchar** or **ntext**.

Return Value

nvarchar or **ntext**

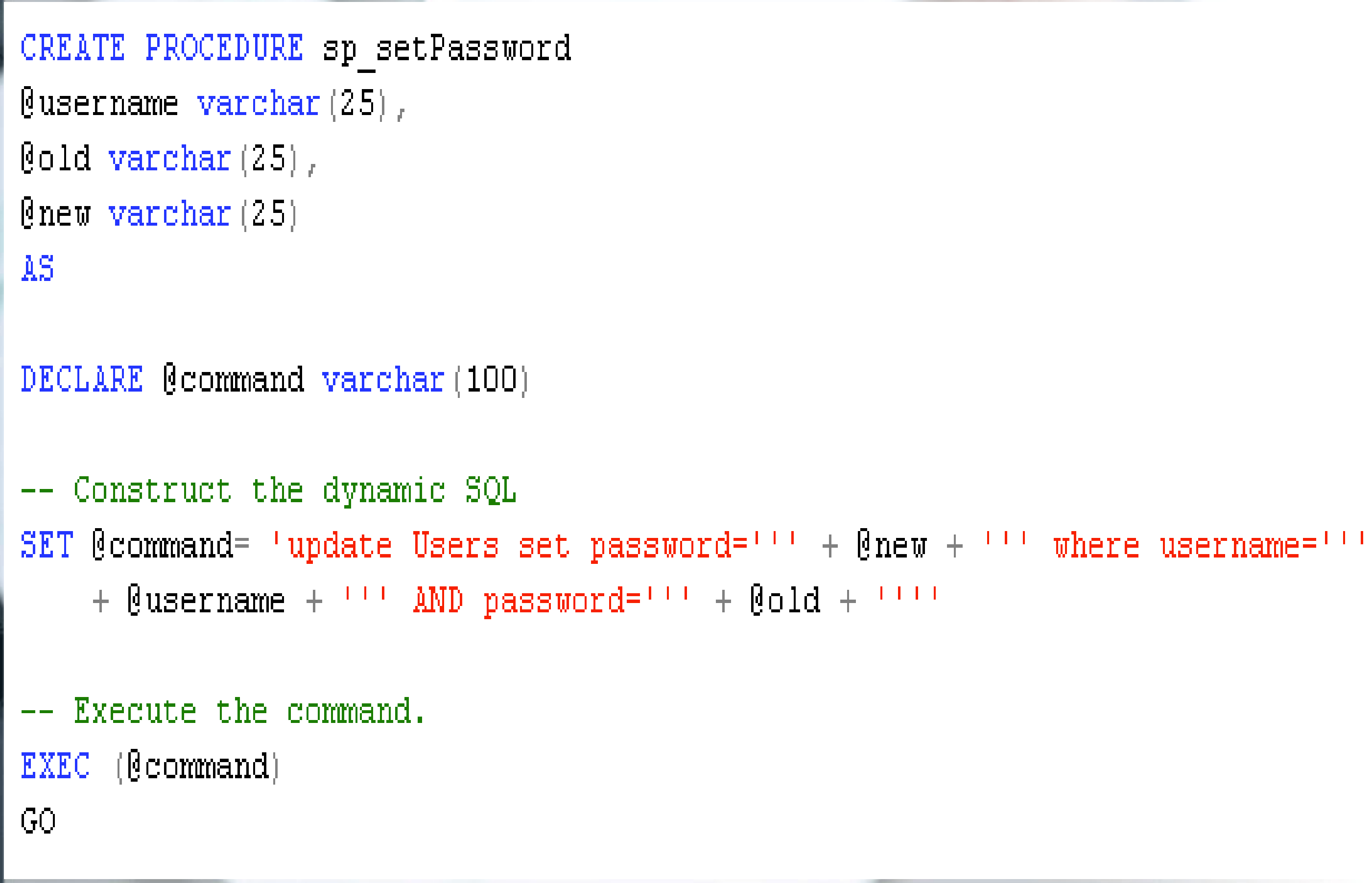
replace() function cont…

```
create procedure sys.sp_attach_single_file_db
    @dbname sysname,
    @physname nvarchar(260)
as
    .....
    .....
    select @execstring = 'CREATE DATABASE '
        + quotename( @dbname , '[')
        + ' ON (FILENAME ='
        + ''''
        + REPLACE(@physname,N'''',N'''''')
        + ''''
        + ' ) FOR ATTACH'
    EXEC (@execstring)
    .....
    .....
    return (0) -- sp_attach_single_file_db
go
```

quotename() vs replace()

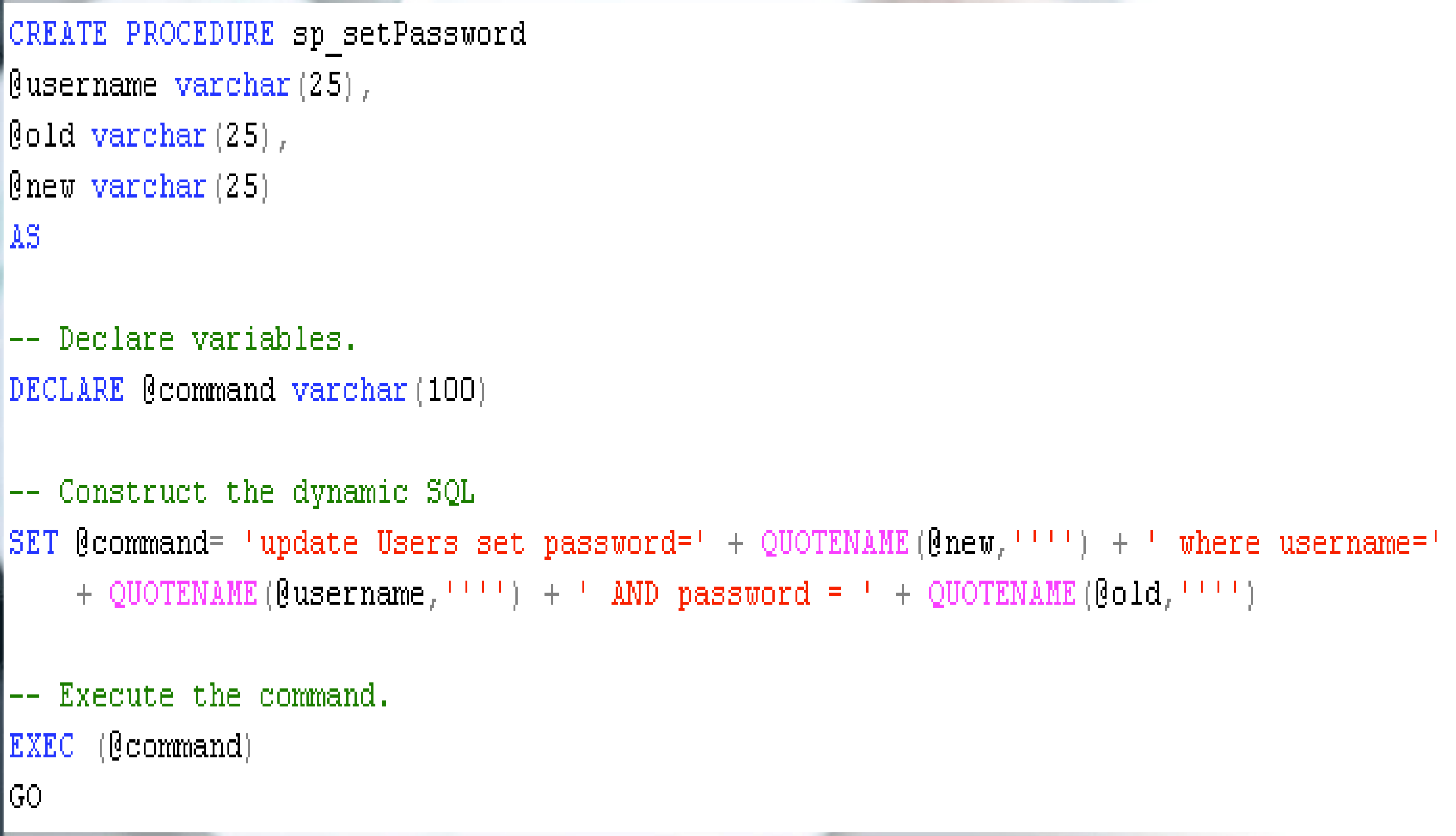
- QUOTENAME works for character strings of length less than or equal to 128 characters.
- Use QUOTENAME for quoting all SQL object names.
- Use REPLACE for character strings of lengths greater than 128 characters.
- Quotename() = delimiter + replace() + delimiter
  - Quotename(@var) = ‘[‘ + replace(@var,’]’,’]]’) + ‘]’
  - Quotename(@var,’”’) = ‘”’ + replace(@var,’”’,’””’) + ‘”’

Dynamic SQL in Stored Procedures



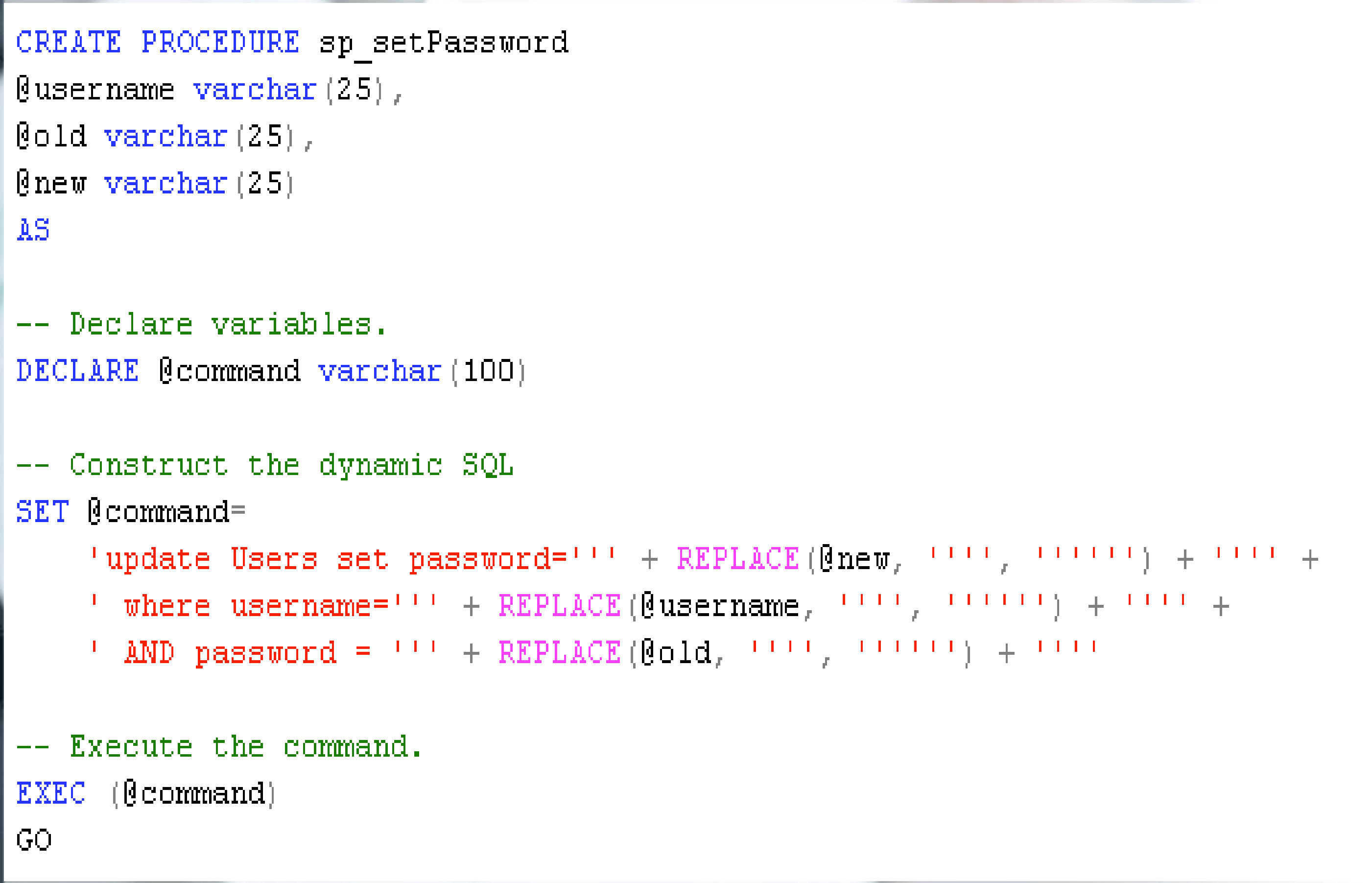
```
CREATE PROCEDURE sp_setPassword
@username varchar(25),
@old varchar(25),
@new varchar(25)
AS

DECLARE @command varchar(100)

-- Construct the dynamic SQL
SET @command= 'update Users set password=''' + @new + ''' where username='''
    + @username + ''' AND password=''' + @old + ''''

-- Execute the command.
EXEC (@command)
GO
```

Lets fix it with quotename()



```sql
CREATE PROCEDURE sp_setPassword
@username varchar(25),
@old varchar(25),
@new varchar(25)
AS

-- Declare variables.
DECLARE @command varchar(100)

-- Construct the dynamic SQL
SET @command= 'update Users set password=' + QUOTENAME(@new,'''') + ' where username='
    + QUOTENAME(@username,'''') + ' AND password = ' + QUOTENAME(@old,'''')

-- Execute the command.
EXEC (@command)
GO
```

Fix it with replace()



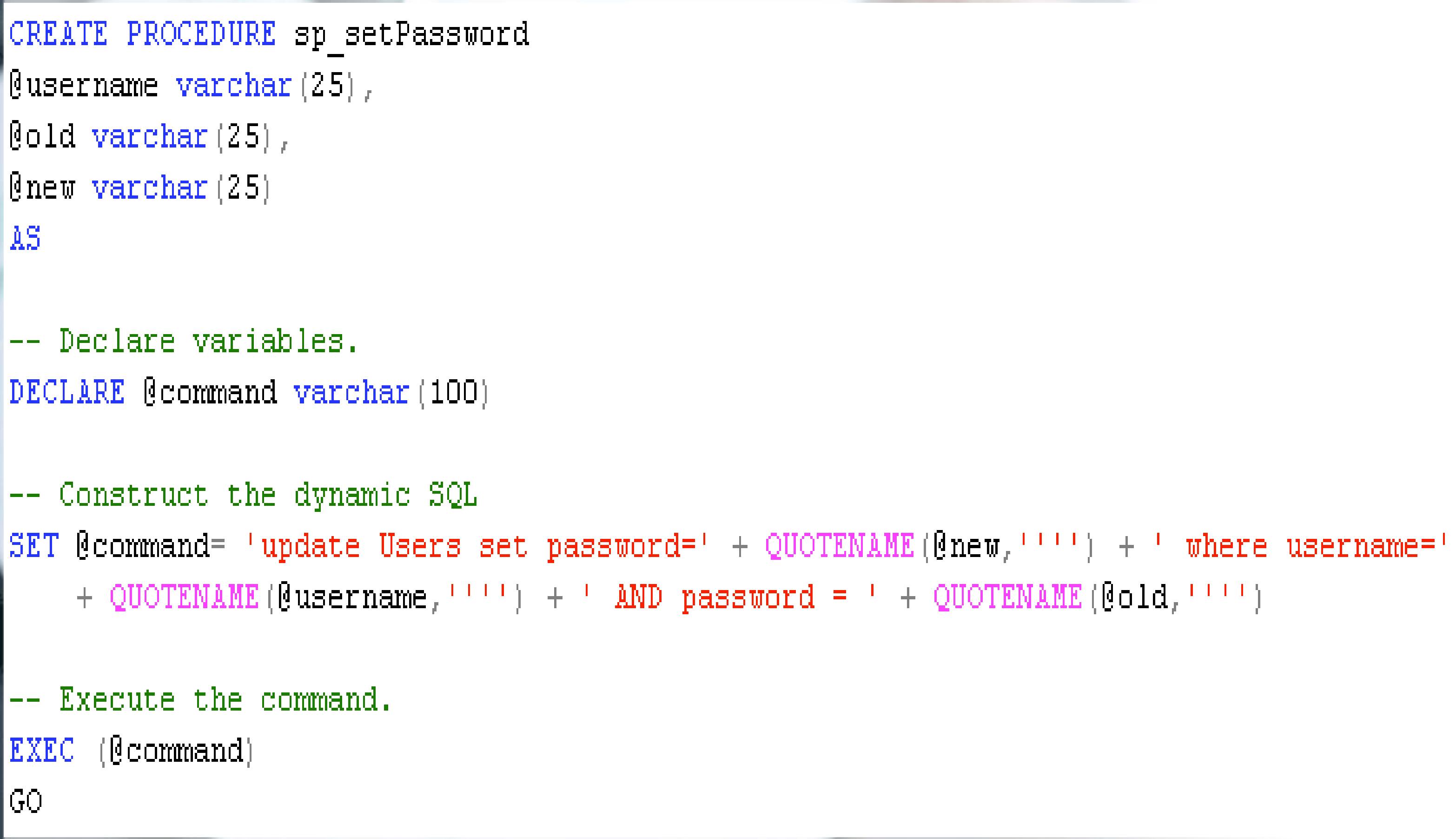
```sql
CREATE PROCEDURE sp_setPassword
@username varchar(25),
@old varchar(25),
@new varchar(25)
AS

-- Declare variables.
DECLARE @command varchar(100)

-- Construct the dynamic SQL
SET @command=
    'update Users set password=''' + REPLACE(@new, '''', '''''') + '''' +
    ' where username=''' + REPLACE(@username, '''', '''''') + '''' +
    ' AND password = ''' + REPLACE(@old, '''', '''''') + ''''

-- Execute the command.
EXEC (@command)
GO
```

Part 1: Key points

- Double up] (right brackets) in SQL Identifiers and delimit them with []s.
- Double up ‘s (single quotes) in character strings and delimit them with single quotes.
- We can use quotename() or replace() to mitigate SQL injections.
- The only difference between these functions is that quotename() adds the beginning and ending delimiters and in case of replace() we will need to add them explicitly.

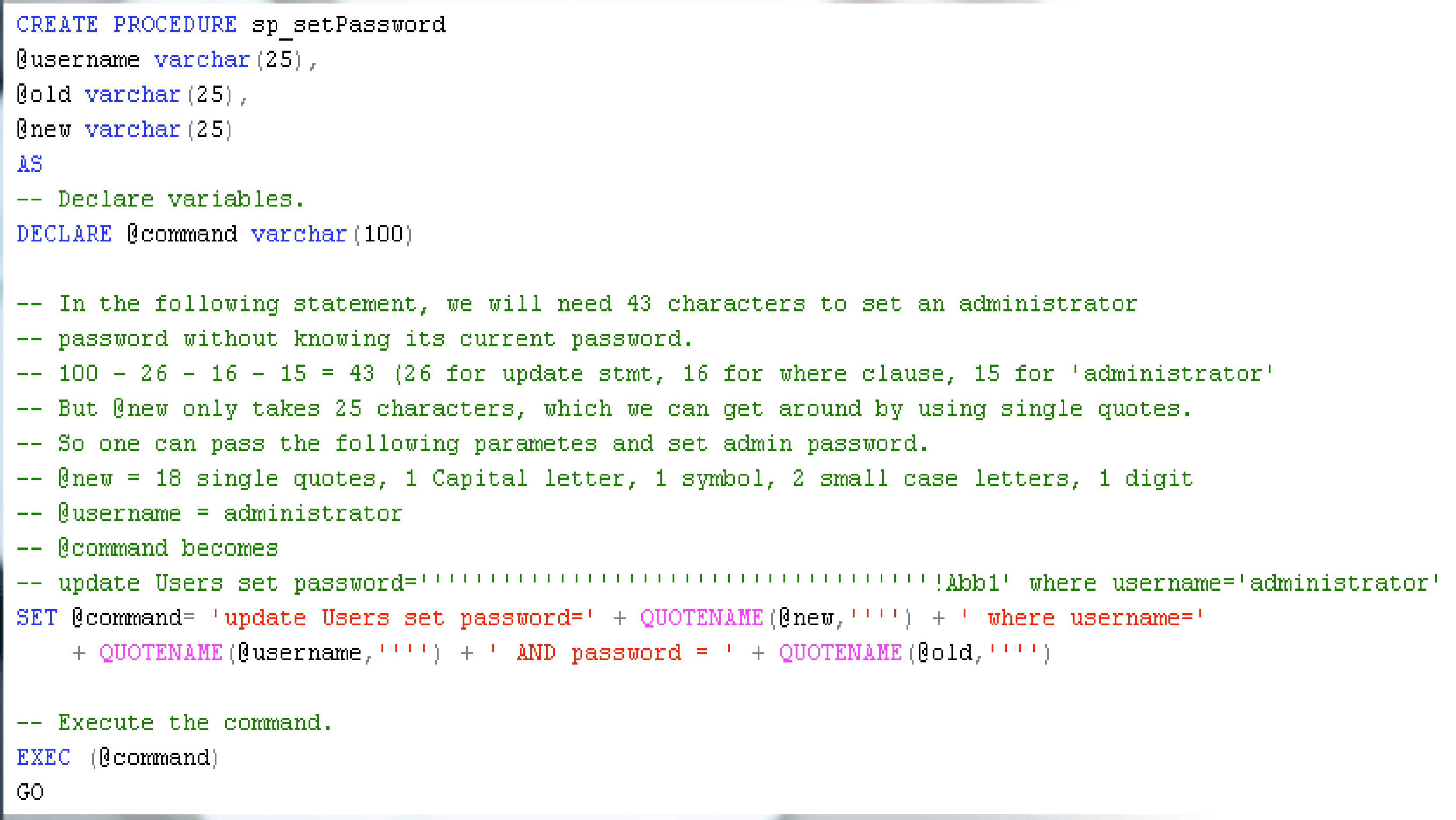
Truncation Issues

What did we fix?



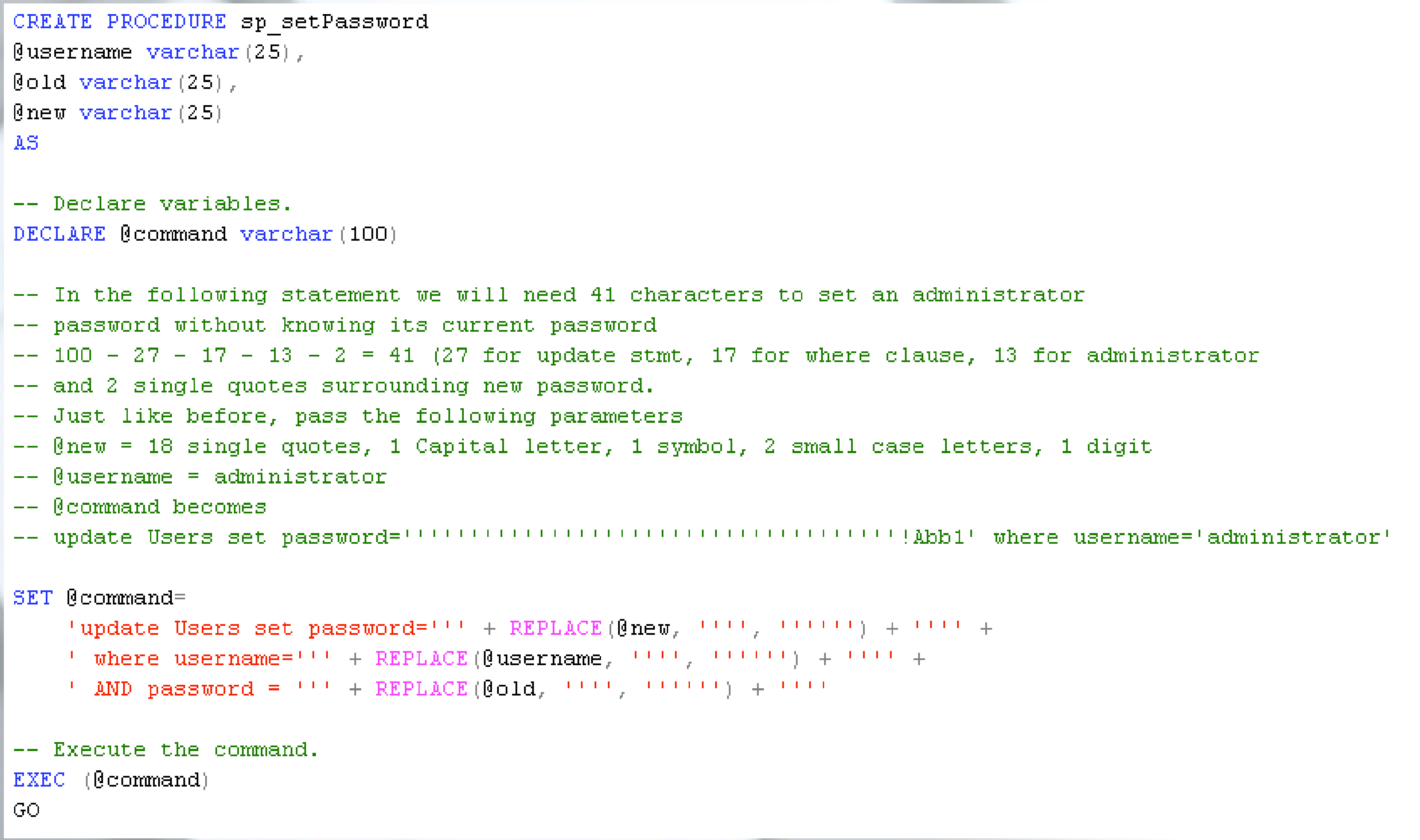
```sql
CREATE PROCEDURE sp_setPassword
@username varchar(25),
@old varchar(25),
@new varchar(25)
AS

-- Declare variables.
DECLARE @command varchar(100)

-- Construct the dynamic SQL
SET @command= 'update Users set password=' + QUOTENAME(@new,'''') + ' where username='
    + QUOTENAME(@username,'''') + ' AND password = ' + QUOTENAME(@old,'''')

-- Execute the command.
EXEC (@command)
GO
```

SQL Modification by Truncation



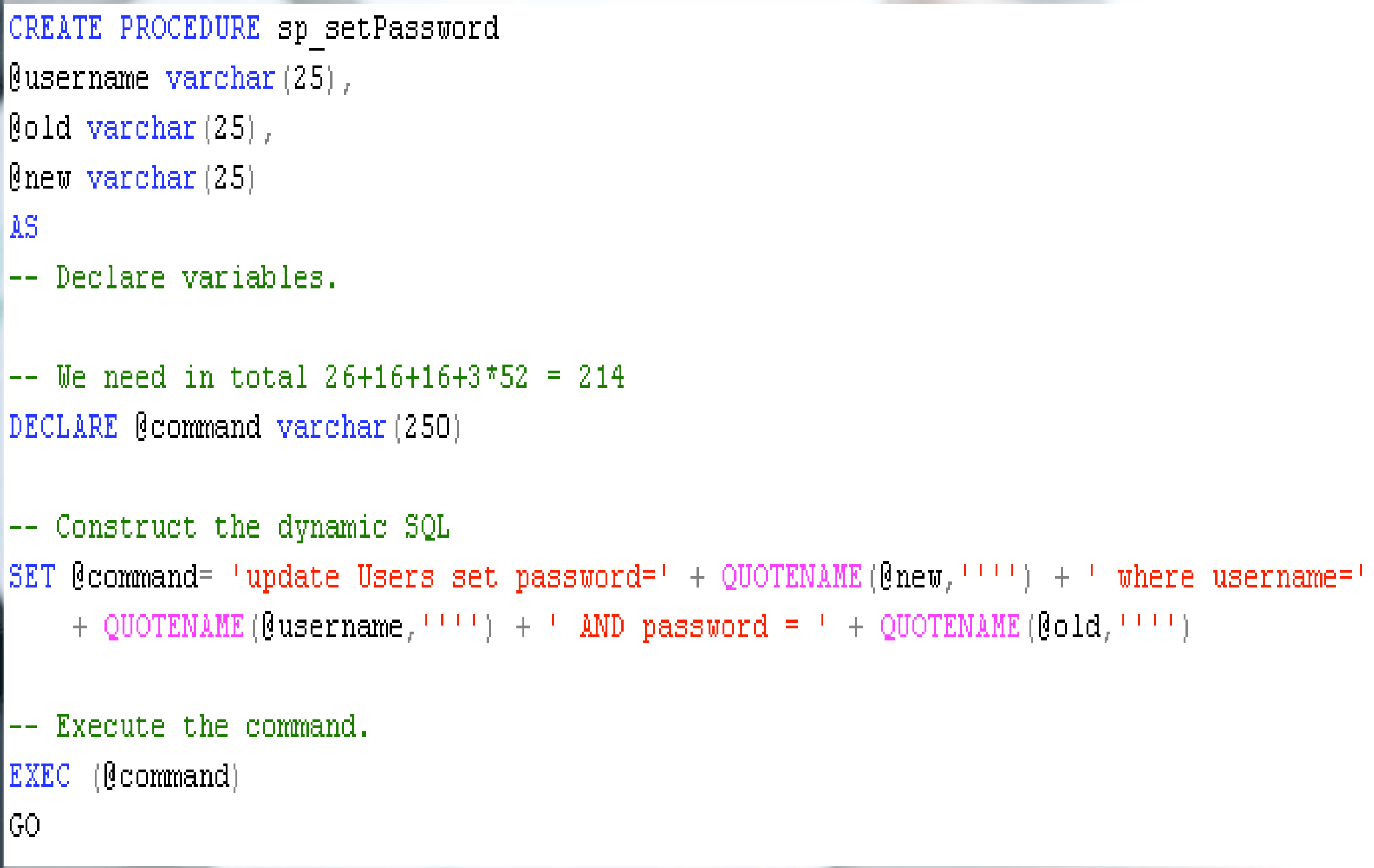
```sql
CREATE PROCEDURE sp_setPassword
@username varchar(25),
@old varchar(25),
@new varchar(25)
AS
-- Declare variables.
DECLARE @command varchar(100)

-- In the following statement, we will need 43 characters to set an administrator
-- password without knowing its current password.
-- 100 - 26 - 16 - 15 = 43 (26 for update stmt, 16 for where clause, 15 for 'administrator'
-- But @new only takes 25 characters, which we can get around by using single quotes.
-- So one can pass the following parametes and set admin password.
-- @new = 18 single quotes, 1 Capital letter, 1 symbol, 2 small case letters, 1 digit
-- @username = administrator
-- @command becomes
-- update Users set password='''''''''''''''''''''''''''''''''''''!Abb1' where username='administrator'
SET @command= 'update Users set password=' + QUOTENAME(@new,'''') + ' where username='
    + QUOTENAME(@username,'''') + ' AND password = ' + QUOTENAME(@old,'''')

-- Execute the command.
EXEC (@command)
GO
```

SQL Modification by Truncation



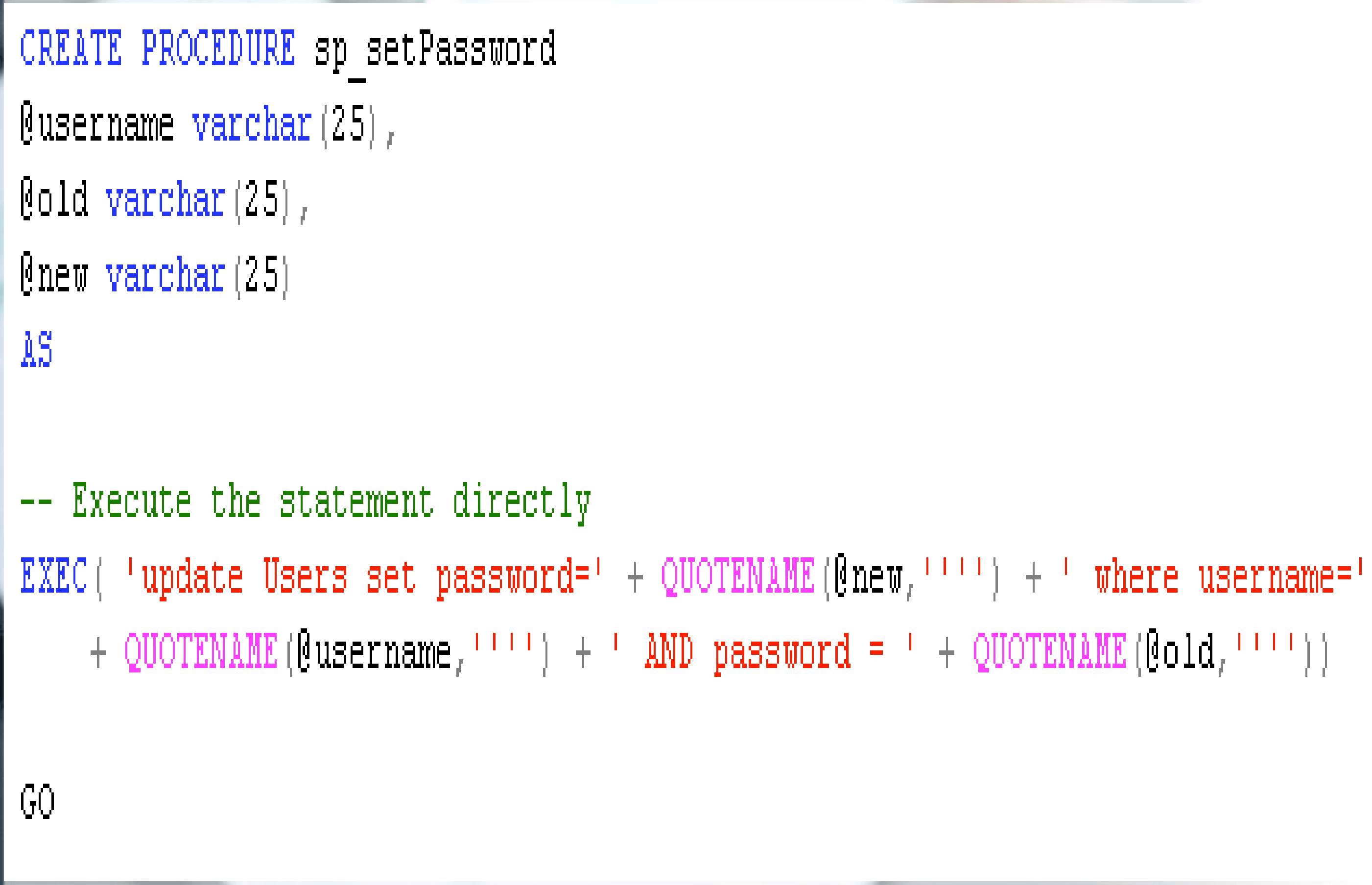
```
CREATE PROCEDURE sp_setPassword
@username varchar(25),
@old varchar(25),
@new varchar(25)
AS

-- Declare variables.
DECLARE @command varchar(100)

-- In the following statement we will need 41 characters to set an administrator
-- password without knowing its current password
-- 100 - 27 - 17 - 13 - 2 = 41 (27 for update stmt, 17 for where clause, 13 for administrator
-- and 2 single quotes surrounding new password.
-- Just like before, pass the following parameters
-- @new = 18 single quotes, 1 Capital letter, 1 symbol, 2 small case letters, 1 digit
-- @username = administrator
-- @command becomes
-- update Users set password='''''''''''''''''''''''''''''''''''''!Abb1' where username='administrator'

SET @command=
    'update Users set password=''' + REPLACE(@new, '''', '''''') + '''' +
    ' where username=''' + REPLACE(@username, '''', '''''') + '''' +
    ' AND password = ''' + REPLACE(@old, '''', '''''') + ''''

-- Execute the command.
EXEC (@command)
GO
```

Calculate the buffer lengths properly



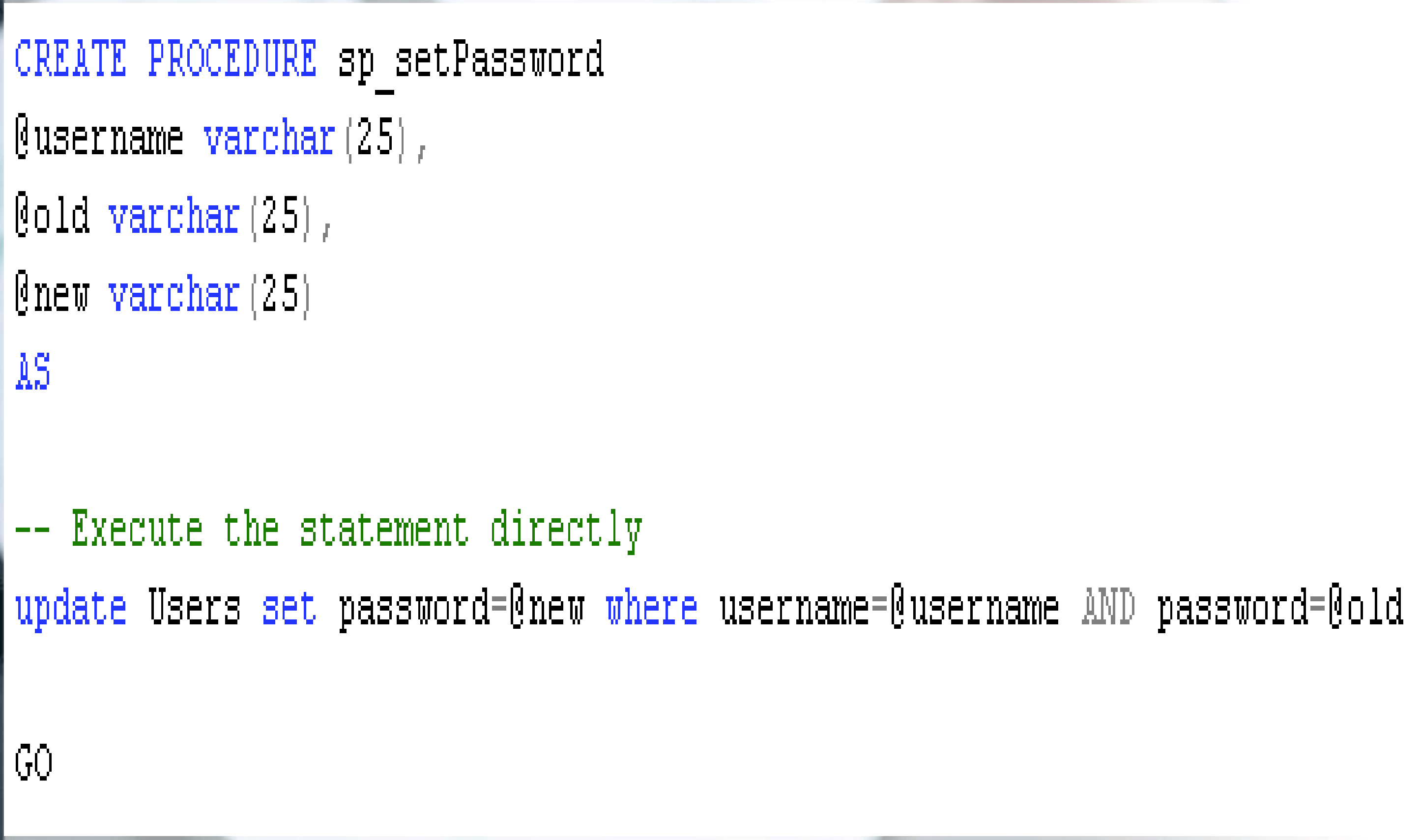
```
CREATE PROCEDURE sp_setPassword
@username varchar(25),
@old varchar(25),
@new varchar(25)
AS
-- Declare variables.

-- We need in total 26+16+16+3*52 = 214
DECLARE @command varchar(250)

-- Construct the dynamic SQL
SET @command= 'update Users set password=' + QUOTENAME(@new,'''') + ' where username='
    + QUOTENAME(@username,'''') + ' AND password = ' + QUOTENAME(@old,'''')

-- Execute the command.
EXEC (@command)
GO
```

Avoid buffers if possible



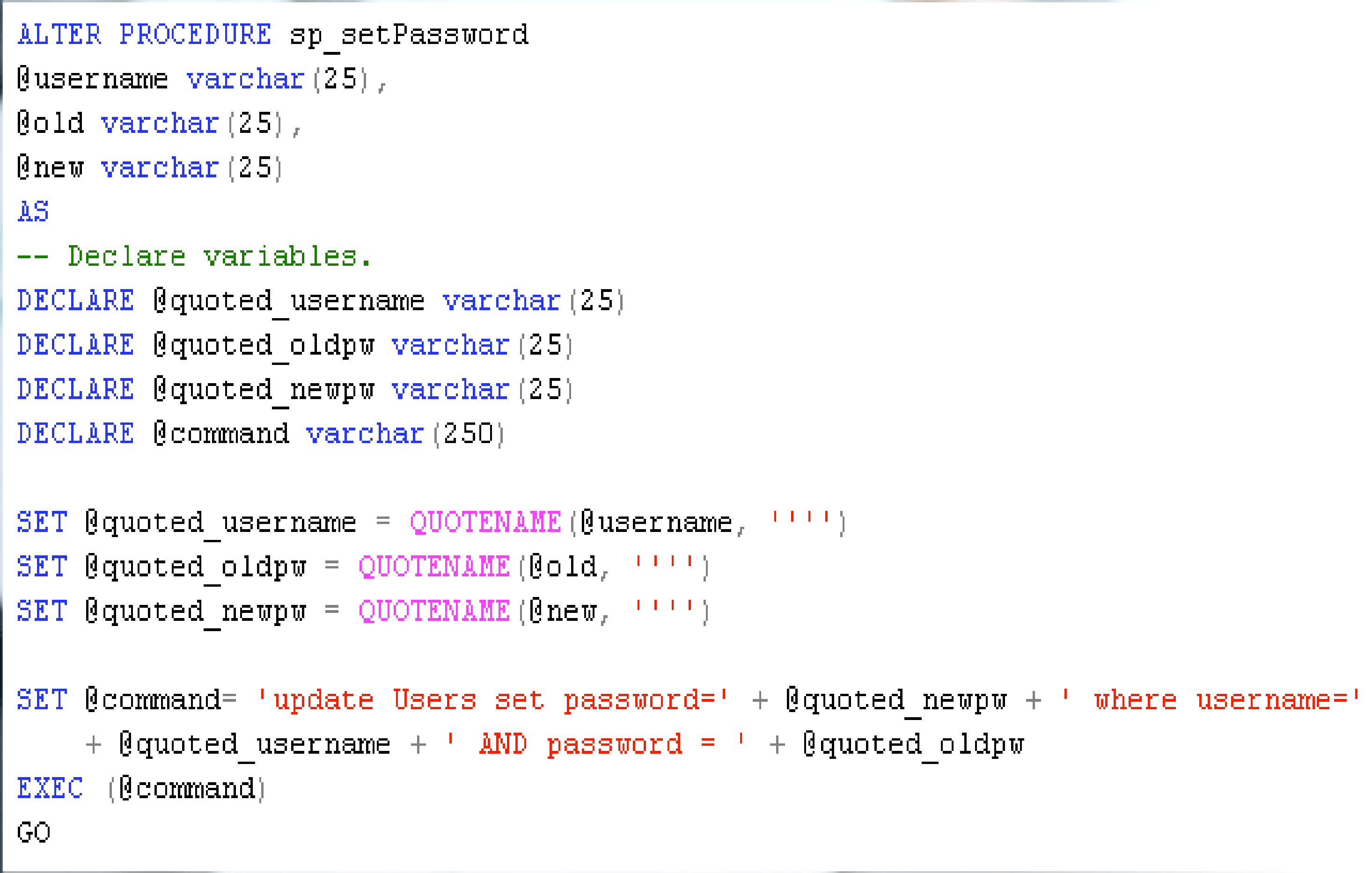
```
CREATE PROCEDURE sp_setPassword
@username varchar(25),
@old varchar(25),
@new varchar(25)
AS


-- Execute the statement directly
EXEC( 'update Users set password=' + QUOTENAME(@new,'''') + ' where username='
    + QUOTENAME(@username,'''') + ' AND password = ' + QUOTENAME(@old,''''))


GO
```

Avoid using dynamic SQL



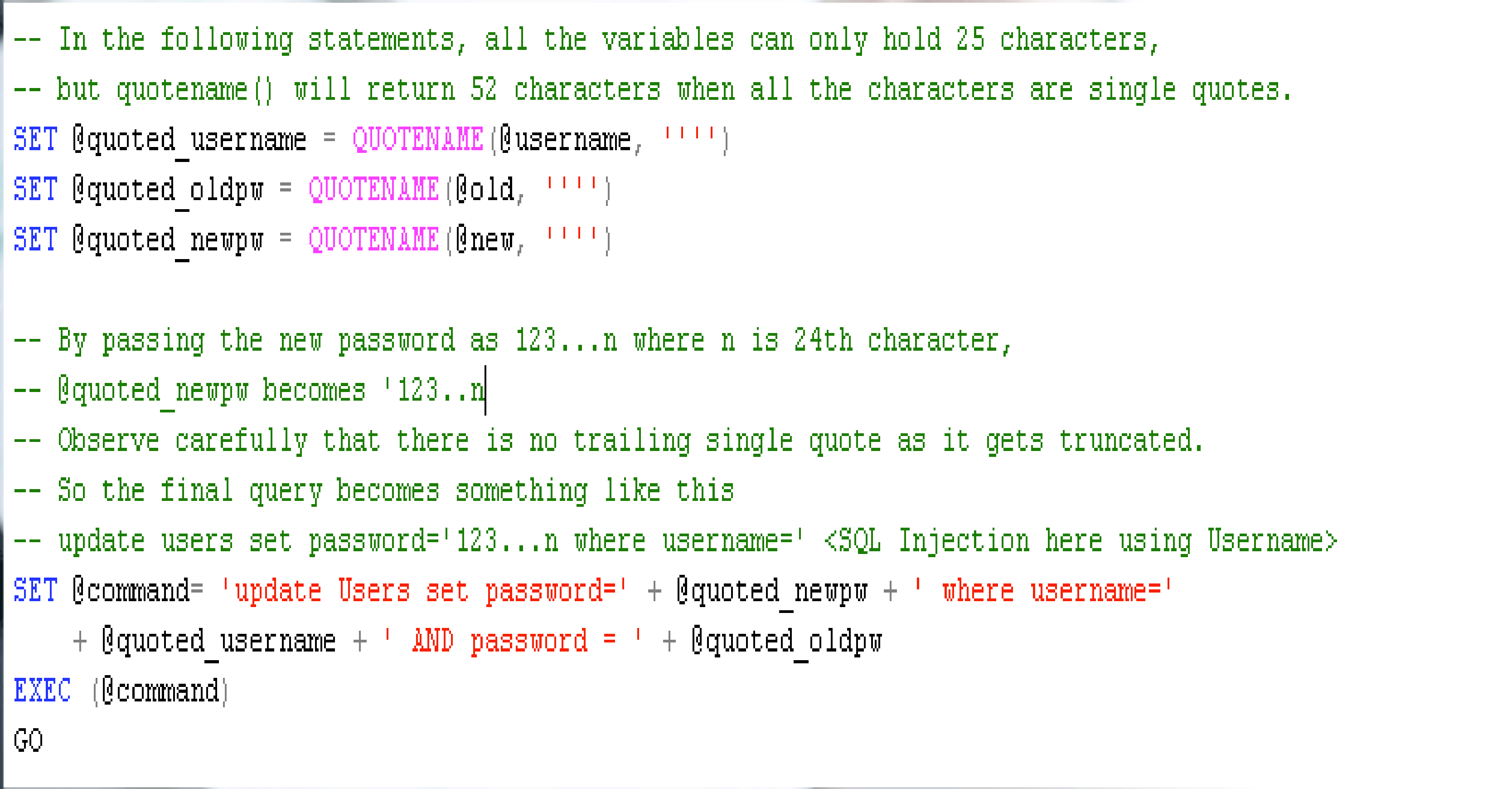
```
CREATE PROCEDURE sp_setPassword
@username varchar(25),
@old varchar(25),
@new varchar(25)
AS


-- Execute the statement directly
update Users set password=@new where username=@username AND password=@old


GO
```

One more variant



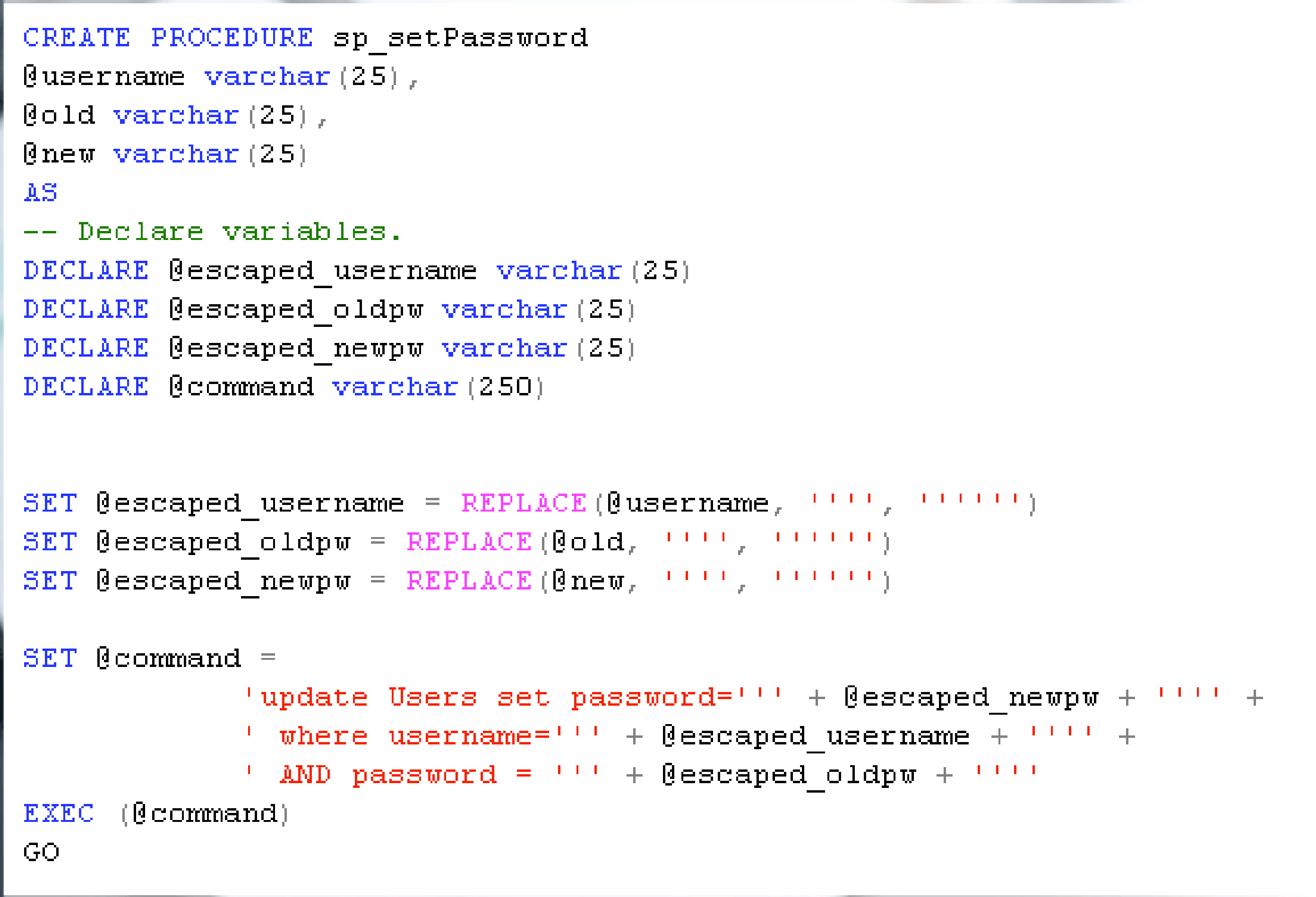
```
ALTER PROCEDURE sp_setPassword
@username varchar(25),
@old varchar(25),
@new varchar(25)
AS
-- Declare variables.
DECLARE @quoted_username varchar(25)
DECLARE @quoted_oldpw varchar(25)
DECLARE @quoted_newpw varchar(25)
DECLARE @command varchar(250)

SET @quoted_username = QUOTENAME(@username, '''')
SET @quoted_oldpw = QUOTENAME(@old, '''')
SET @quoted_newpw = QUOTENAME(@new, '''')

SET @command= 'update Users set password=' + @quoted_newpw + ' where username='
    + @quoted_username + ' AND password = ' + @quoted_oldpw
EXEC (@command)
GO
```

SQL Injection by truncation



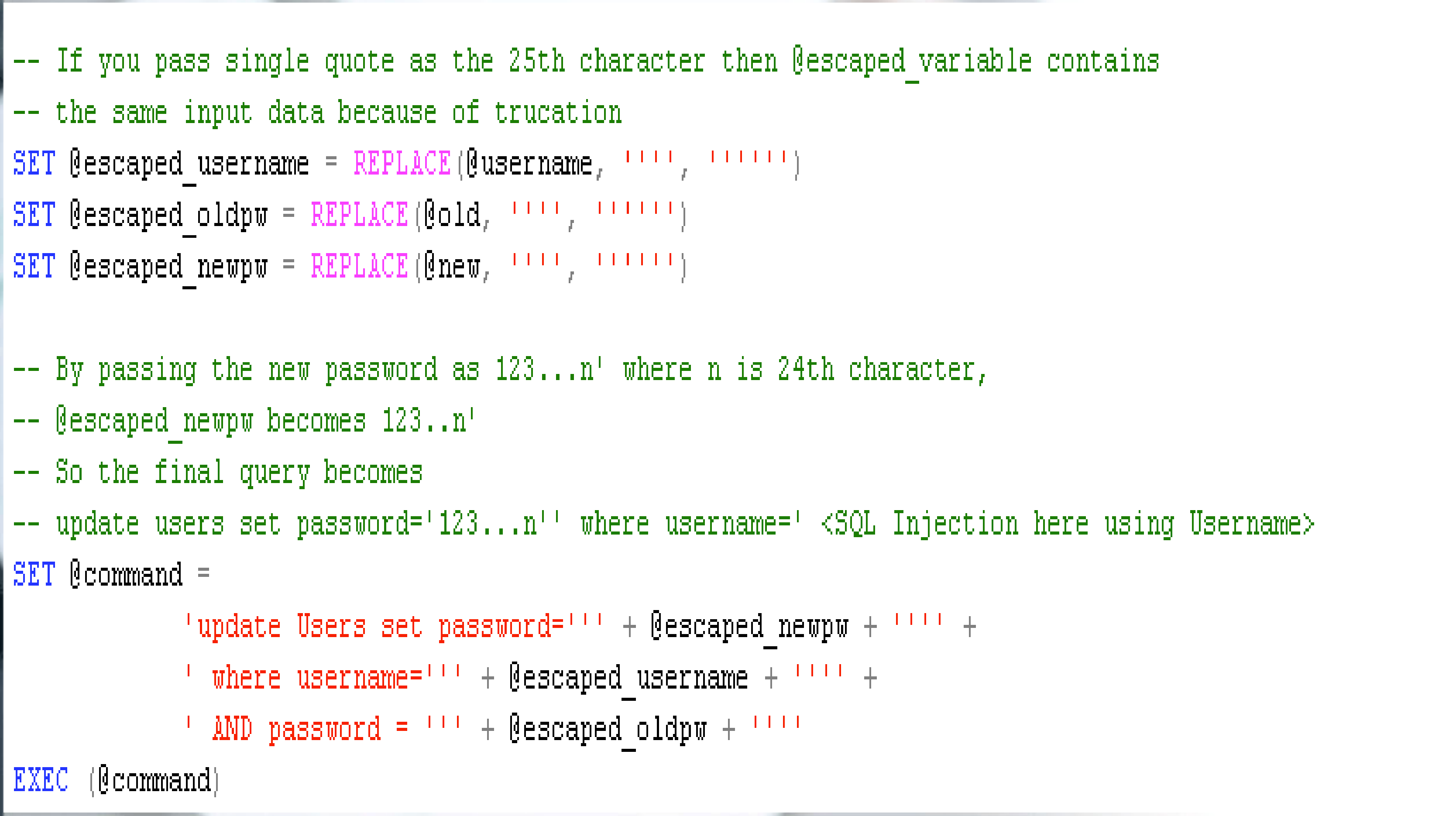
```
-- In the following statements, all the variables can only hold 25 characters,
-- but quotename() will return 52 characters when all the characters are single quotes.
SET @quoted_username = QUOTENAME(@username, '''')
SET @quoted_oldpw = QUOTENAME(@old, '''')
SET @quoted_newpw = QUOTENAME(@new, '''')

-- By passing the new password as 123...n where n is 24th character,
-- @quoted_newpw becomes '123..n
-- Observe carefully that there is no trailing single quote as it gets truncated.
-- So the final query becomes something like this
-- update users set password='123...n where username=' <SQL Injection here using Username>
SET @command= 'update Users set password=' + @quoted_newpw + ' where username='
    + @quoted_username + ' AND password = ' + @quoted_oldpw
EXEC (@command)
GO
```

SQL Injection by truncation



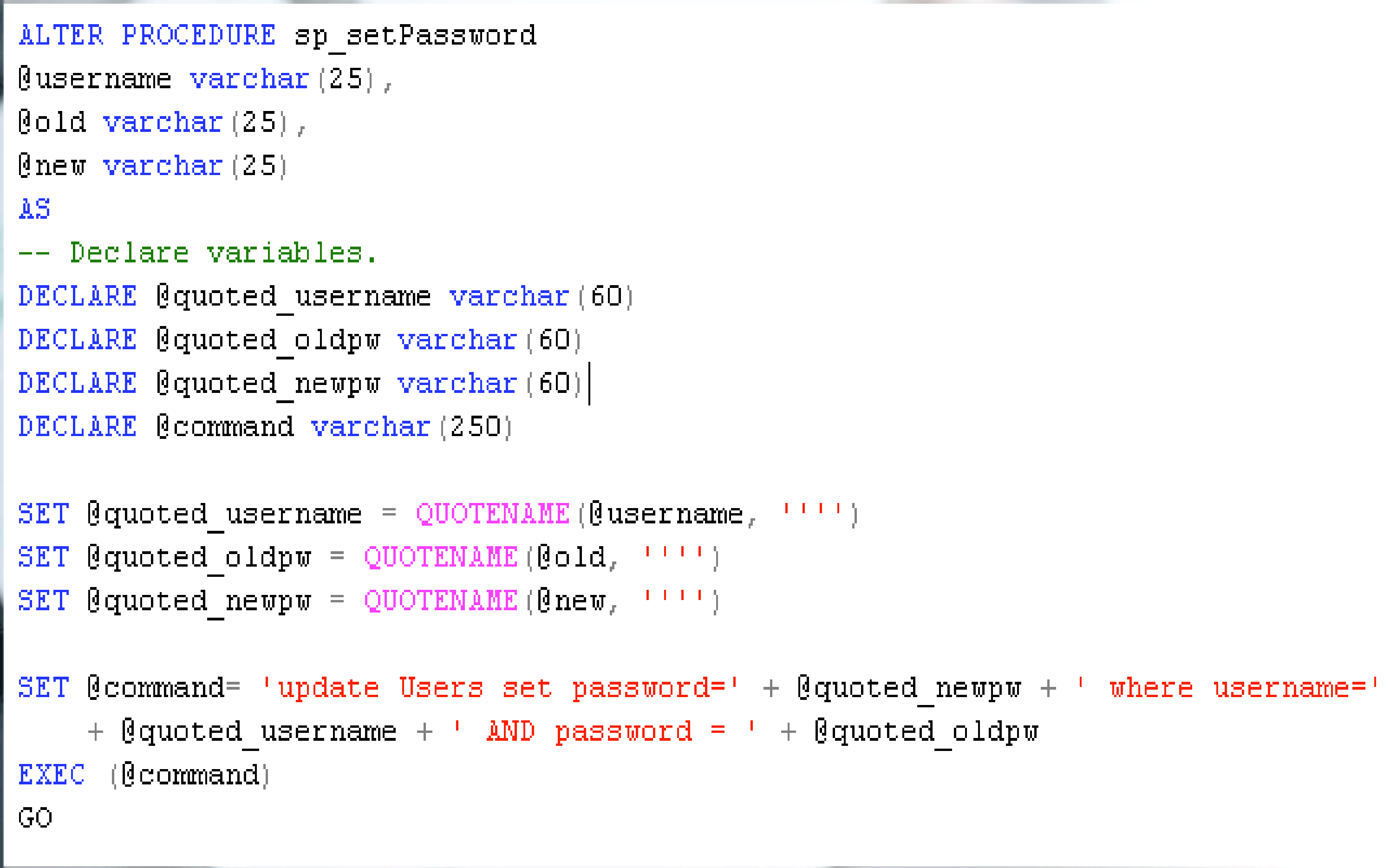
```
CREATE PROCEDURE sp_setPassword
@username varchar(25),
@old varchar(25),
@new varchar(25)
AS
-- Declare variables.
DECLARE @escaped_username varchar(25)
DECLARE @escaped_oldpw varchar(25)
DECLARE @escaped_newpw varchar(25)
DECLARE @command varchar(250)


SET @escaped_username = REPLACE(@username, '''', '''''')
SET @escaped_oldpw = REPLACE(@old, '''', '''''')
SET @escaped_newpw = REPLACE(@new, '''', '''''')

SET @command =
            'update Users set password=''' + @escaped_newpw + '''' +
            ' where username=''' + @escaped_username + '''' +
            ' AND password = ''' + @escaped_oldpw + ''''
EXEC (@command)
GO
```

SQL Injection by truncation



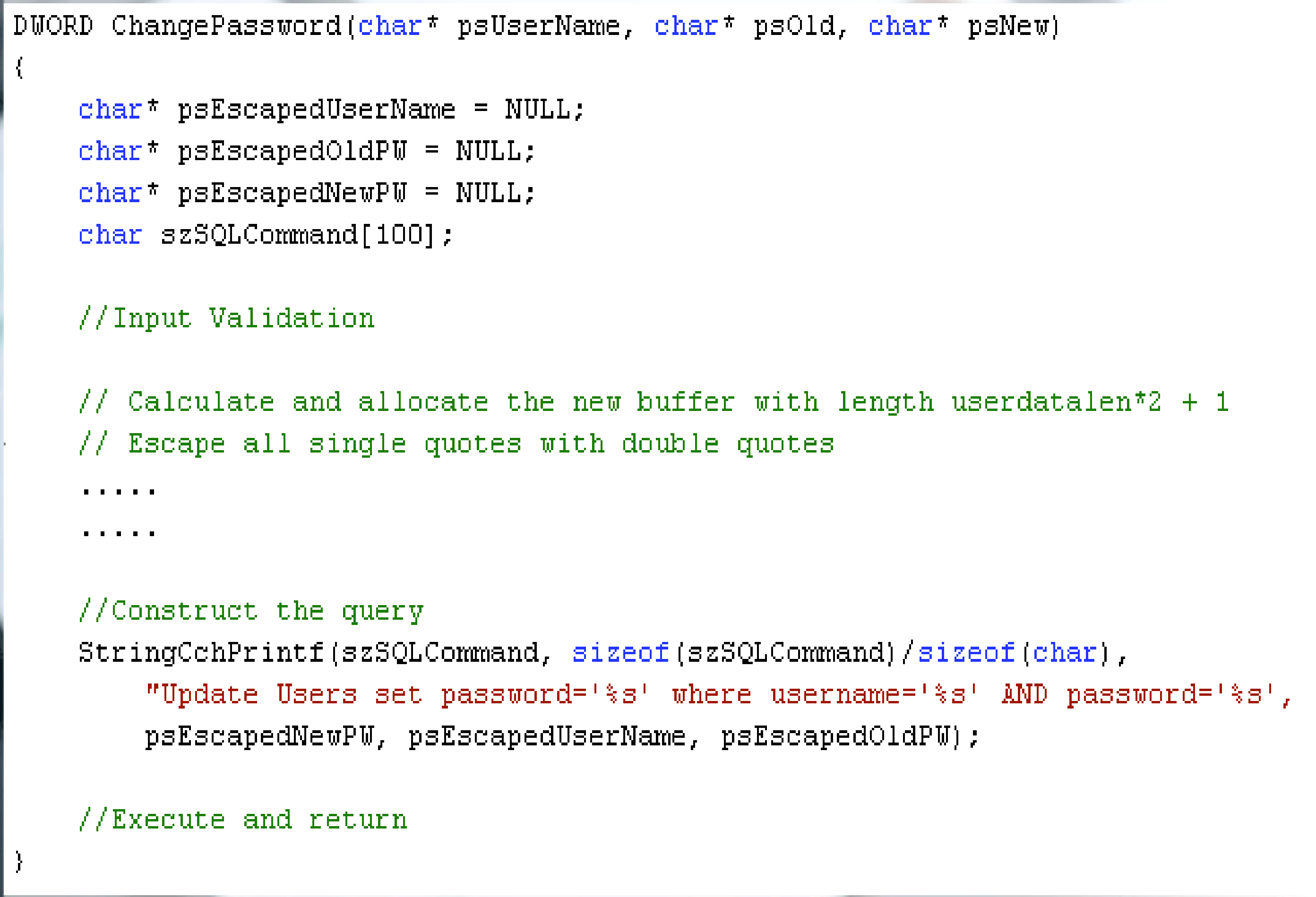
```
-- If you pass single quote as the 25th character then @escaped_variable contains
-- the same input data because of trucation
SET @escaped_username = REPLACE(@username, '''', '''''')
SET @escaped_oldpw = REPLACE(@old, '''', '''''')
SET @escaped_newpw = REPLACE(@new, '''', '''''')

-- By passing the new password as 123...n' where n is 24th character,
-- @escaped_newpw becomes 123..n'
-- So the final query becomes
-- update users set password='123...n'' where username=' <SQL Injection here using Username>
SET @command =
            'update Users set password=''' + @escaped_newpw + '''' +
            ' where username=''' + @escaped_username + '''' +
            ' AND password = ''' + @escaped_oldpw + ''''
EXEC (@command)
```

Calculate the buffers properly



```
ALTER PROCEDURE sp_setPassword
@username varchar(25),
@old varchar(25),
@new varchar(25)
AS
-- Declare variables.
DECLARE @quoted_username varchar(60)
DECLARE @quoted_oldpw varchar(60)
DECLARE @quoted_newpw varchar(60)
DECLARE @command varchar(250)

SET @quoted_username = QUOTENAME(@username, '''')
SET @quoted_oldpw = QUOTENAME(@old, '''')
SET @quoted_newpw = QUOTENAME(@new, '''')

SET @command= 'update Users set password=' + @quoted_newpw + ' where username='
    + @quoted_username + ' AND password = ' + @quoted_oldpw
EXEC (@command)
GO
```

SQL modification by truncation

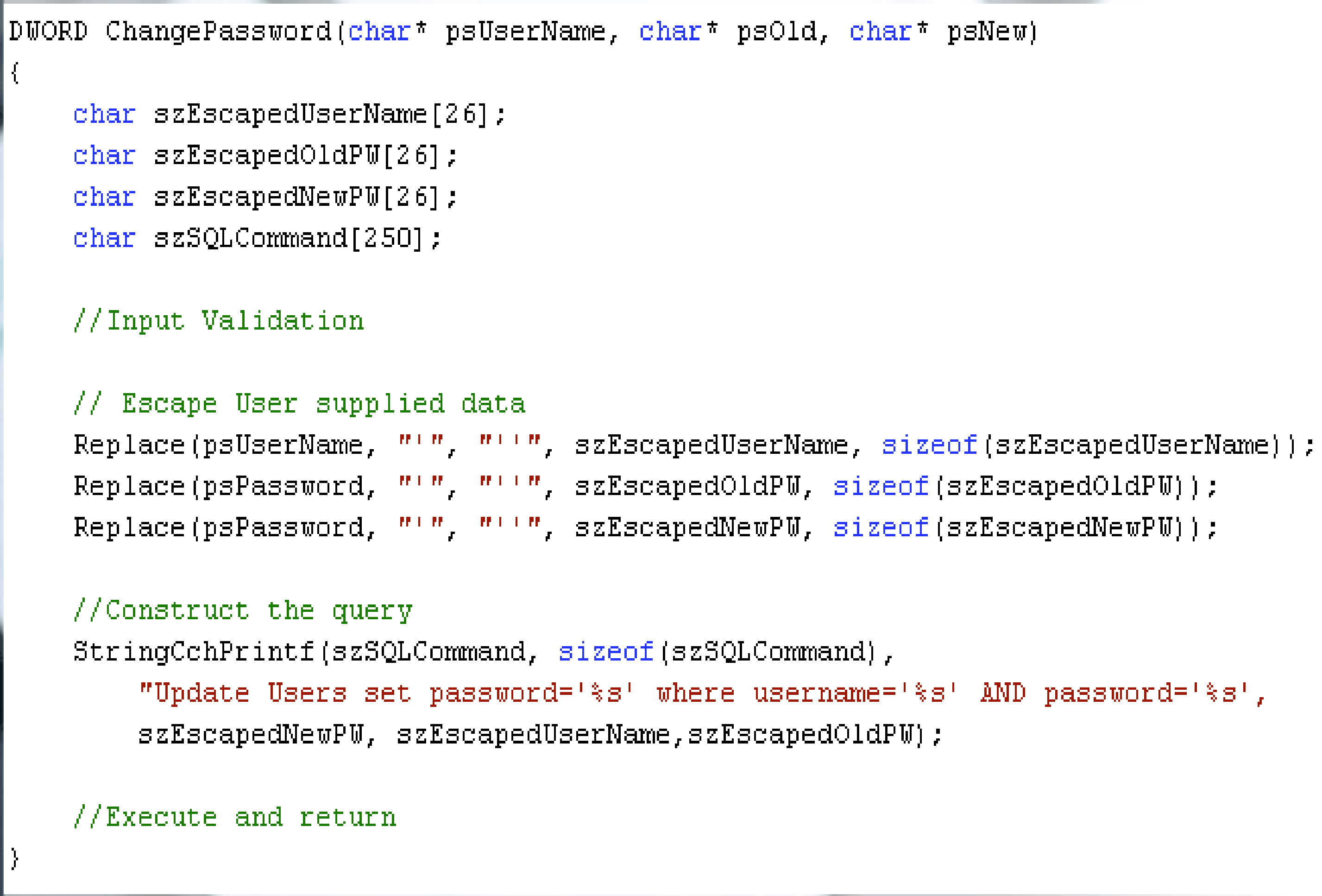


```
DWORD ChangePassword(char* psUserName, char* psOld, char* psNew)
{
    char* psEscapedUserName = NULL;
    char* psEscapedOldPW = NULL;
    char* psEscapedNewPW = NULL;
    char szSQLCommand[100];

    //Input Validation

    // Calculate and allocate the new buffer with length userdatalen*2 + 1
    // Escape all single quotes with double quotes
    .....
    .....

    //Construct the query
    StringCchPrintf(szSQLCommand, sizeof(szSQLCommand)/sizeof(char),
        "Update Users set password='%s' where username='%s' AND password='%s',
        psEscapedNewPW, psEscapedUserName, psEscapedOldPW);

    //Execute and return
}
```

Check for return values

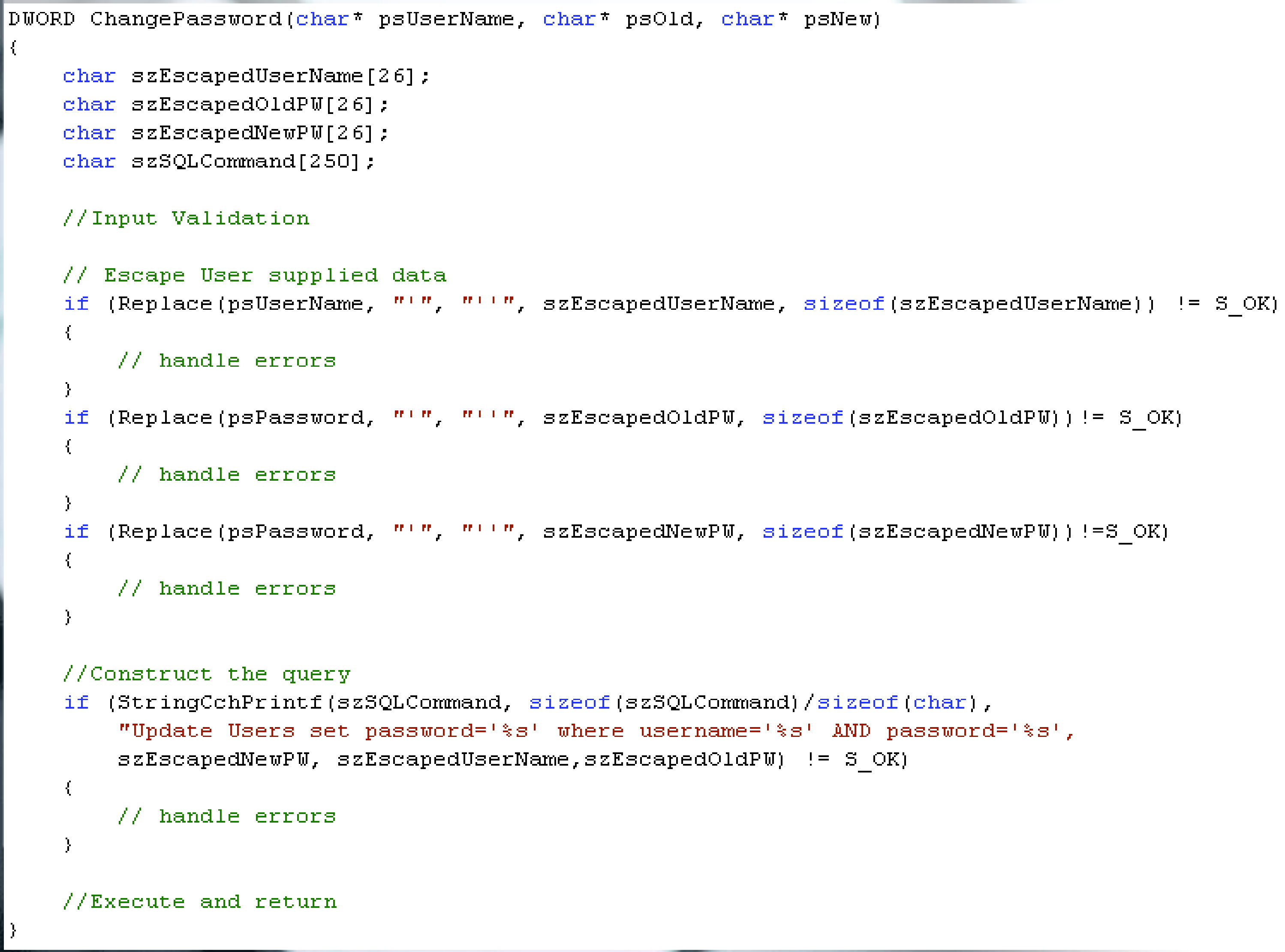
```
DWORD ChangePassword(char* psUserName, char* psOld, char* psNew)
{
    char* psEscapedUserName = NULL;
    char* psEscapedOldPW = NULL;
    char* psEscapedNewPW = NULL;
    char szSQLCommand[100];
    HRESULT hr=0;
    //Input Validation

    // Calculate and allocate the new buffer with length userdatalen*2 + 1
    // Escape all single quotes with double quotes
    .....
    .....

    //Construct the query
    hr = StringCchPrintf(szSQLCommand, sizeof(szSQLCommand)/sizeof(char),
        "Update Users set password='%s' where username='%s' AND password='%s',
        psEscapedNewPW, psEscapedUserName, psEscapedOldPW);

    if (S_OK != hr)
    {
        // handle error cases
    }

    //Execute and return
}
```

SQL Injection by truncation



```
DWORD ChangePassword(char* psUserName, char* psOld, char* psNew)
{
    char szEscapedUserName[26];
    char szEscapedOldPW[26];
    char szEscapedNewPW[26];
    char szSQLCommand[250];

    //Input Validation

    // Escape User supplied data
    Replace(psUserName, "'", "''", szEscapedUserName, sizeof(szEscapedUserName));
    Replace(psPassword, "'", "''", szEscapedOldPW, sizeof(szEscapedOldPW));
    Replace(psPassword, "'", "''", szEscapedNewPW, sizeof(szEscapedNewPW));

    //Construct the query
    StringCchPrintf(szSQLCommand, sizeof(szSQLCommand),
        "Update Users set password='%s' where username='%s' AND password='%s',
        szEscapedNewPW, szEscapedUserName,szEscapedOldPW);

    //Execute and return
}
```

Check for return values



```
DWORD ChangePassword(char* psUserName, char* psOld, char* psNew)
{
    char szEscapedUserName[26];
    char szEscapedOldPW[26];
    char szEscapedNewPW[26];
    char szSQLCommand[250];

    //Input Validation

    // Escape User supplied data
    if (Replace(psUserName, "'", "''", szEscapedUserName, sizeof(szEscapedUserName))  != S_OK)
    {
        // handle errors
    }
    if (Replace(psPassword, "'", "''", szEscapedOldPW, sizeof(szEscapedOldPW))!= S_OK)
    {
        // handle errors
    }
    if (Replace(psPassword, "'", "''", szEscapedNewPW, sizeof(szEscapedNewPW))!=S_OK)
    {
        // handle errors
    }

    //Construct the query
    if (StringCchPrintf(szSQLCommand, sizeof(szSQLCommand)/sizeof(char),
        "Update Users set password='%s' where username='%s' AND password='%s',
        szEscapedNewPW, szEscapedUserName,szEscapedOldPW)  != S_OK)
    {
        // handle errors
    }

    //Execute and return
}
```

Key points

- SQL modification is enabled by truncating the command string.
- SQL injection is enabled by truncating the quoted string.
- Truncation issues are not specific to PL/SQL code.

Affected Applications

- Applications written in TSQL and C/C++
  - Web Applications
  - Mid-tier Applications
  - Backend Applications
  - Tools and client applications
  - Internal Maintenance Scripts.

Finding SQL injections

- Identify the calls that execute dynamic SQL
- Review the construction of dynamic SQL
- Review the buffers used for the variables

Mitigating SQL Injections by truncation

- If possible, call QUOTENAME() or REPLACE() directly inside the dynamic Transact-SQL.
- Calculate the buffer lengths properly.
- Check the return values for truncation errors.

Resources

- http://msdn2.microsoft.com/en-us/library/ms161953(SQL.90).aspx

Questions ?