#1 - Stored Procedures (SPs) Inside SQL Server

* Stored Procedures are precompiled Transact-SQL statements stored in a SQL Server database.

* Stored Procedures are one of the most powerful pieces of *programming* you will ever see. When you start out, you will see them as a way to return a record set, or do some small update on your data. As you learn more about SPs you will understand why there are entire books written on the subject. SQL Server compiles the Proc so that when you run it, it runs as fast as possible. Once you write a couple of complicated SPs, you will be convinced. This paper only covers the tip of the Stored Procedure iceberg.

* I will refer to Stored Procedures in this document as SP and Proc - get use to it.

* Stored Procedures return *read only data* and can have
  > Input parameters
  > Output parameters
  > Parameters that are both input and output
  > Can have 1 or more recordsets
#2 - Simple Recordset with a Input Parameter

* Figure 2-1 shows a simple stored procedures with that has in input parameter and returns a recordset. When we run it from the Query Analyzer (Figure 2-2) we get the following results.

* If you notice in Figure 2-2, it shows “(3 row(s) affected)”. If you don’t set “set nocount on” in a SP, when you run the SP in the Query Analyzer, you will get back a message “X rows affected”. By setting nocount on, it stops SQL Server from doing some work, that you don’t care about. This will cause the SP to run just a little faster.

* You need to learn about sp_Help and other system stored procedures. Works with or without the single quotes.

* You can also run the query in an Access Pass-through query.
In figure 2-4 we use ADO code and the command object to get a recordset from the Stored Procedure on SQL Server.

Note: You certainly can do this many different ways, however I do want to point out the difference between the While, Wend Loop as opposed to the GetString. You will probably want to use the GetString in testing.

```vbs
Public Function ex_SP_ReadRecords()
    '>-- Uses the Command Object
    Dim Cmd1 As ADODB.Command
    Dim lngRecordsAffected As Long
    Dim rs1 As ADODB.Recordset
    Dim intRecordCount As Integer

    Dim cnnTemp As ADODB.Connection
    Set cnnTemp = New ADODB.Connection
    cnnTemp.ConnectionString = "Provider=SQLOLEDB.1;" & _
        "DRIVER=SQL Server;SERVER=bcnt;" & _
        "Trusted_Connection=Yes;UID=;PWD=;" & _
        "DATABASE=MWData;"
    cnnTemp.Open

    'Open Connection
    Set Cmd1 = New ADODB.Command
    Cmd1.ActiveConnection = cnnTemp

    '---
    With Cmd1
        .CommandText = "z_sp_SimpleReadTable"
        .CommandType = adCmdStoredProc
        .Parameters.Refresh
        .Parameters("@vcCompanyName").Value = "bus"
    End With

    Set rs1 = Cmd1.Execute()
    'While Not rs1.EOF
    '    intRecordCount = intRecordCount + 1
    '    Debug.Print rs1.Fields(1), intRecordCount
    '    rs1.MoveNext
    'Wend

    The following lines shows all the records and all fields from the above recordset
    Debug.Print rs1.GetString(adClipString, , ";")

    rs1.Close

    Finish_Up:
        ex_SP_ReadRecords = True

ProcedureDone:
    On Error Resume Next
    rs1.Close
    Set Cmd1 = Nothing
    Set rs1 = Nothing

Exit Function

HandleError:
    Debug.Print Err.Number, Err.Description
    Resume ProcedureDone
End Function
```

Figure 2-4  ADO using Proc for recordset
#3 - Simple Input & Output Parameters

* Figure 3-1 shows another example of a simple SP with input and output parameter. In the SP we input a company Id (@vcCo_IdT) and return the company name in the output parameter. We run the SP with ADO Code. (see figure-6)

The Proc simply takes the input from the ADO code, runs the T-SQL statement using the input parameter, and returns the answer to the ADO code.

* Notice the line in Figure-3-2. This is a remark. You can put in a remark with “/*” and end with “*/”. You can also use two dashes “--this is a test” for a single line.

* The ADO code (figure-3-3) Opens the connection, sets the command, refreshes the parameters, and set the value of the parameter, and then executes the proc. It then reads the output parameter from the proc.

Note: If you run this procedure from the query analyzer, you will need to put in a false parameter for the output parameter, and probably put a print statement inside the proc to show the output parameter in the query analyzer.

```
Figure 3-1 Stored Procedure Input and Output Parameters

Text:
CREATE Procedure z_sp_In_Out_Parameters_Simple
    @vcCo_IdT as varchar(10),
    @vcOutPut1 AS varchar[100] output
As
    set nocount on

    /* Input @vcCo_Idt = 266
      OutPut @vcOutPut1 = Business & Computers, Inc */

    select @vcOutPut1 = Co_AliName from tbl_Companies where Co_IdT = @vcCo_IdT
    return
```

```
Figure 3-2 Comment

/*Input @vcCo_Idt =266
OutPut @vcOutPut1 = Business & Computers, Inc */
```

```
Figure 3-3 ADO Code to run Proc

Public Function ex_SP_In_Out_Parameters_Simple_2()
    'On Error GoTo HandleError

    Dim Cmd1 As ADODB.Command
    Dim lngRecordsAffected As Long
    Dim cnnTemp As ADODB.Connection

    Set cnnTemp = New ADODB.Connection
    cnnTemp.ConnectionString = "Provider=SQLOLED.B1;" & _
    "DRIVER=SQL Server;SERVER=bcnt;" & _
    "Trusted_Connection=Yes;UID=;PWD=;" & _
    "DATABASE=MWData;"
    cnnTemp.Open

    '----
    'Open Command Object
    Set Cmd1 = New ADODB.Command
    Cmd1.ActiveConnection = cnnTemp

    '----
    'With Cmd1
    .CommandText = "z_sp_In_Out_Parameters_Simple"
    .CommandType = adCmdStoredProc
    .Parameters.Refresh
    .Parameters("@vcCo_IdT").Value = 266
    .Execute , lngRecordsAffected, adExecuteNoRecords
    End With

    Debug.Print Cmd1.Parameters("@vcOutPut1").Value

    Set Cmd1 = Nothing

    ProcedureDone:
    Exit Function

    HandleError:
```
CREATE PROCEDURE z_sp_In_Out_Parameters_Complex
    @vcWhere AS varchar(8000),
    @vcTableFromName AS varchar(255),
    @vcIDName AS varchar(55),
    @vcTableInToName AS varchar(255) = 'tbl_zs_StartID',
    @btNumericId_YN as bit=0,
    @vcOutPut1 AS varchar(255) output
AS
    SET NOCOUNT ON

    /*--------------------------------------------------------------------
     --> Purpose: This is an Example of In & Out Parameters
     See ex_SP_In_Out_Parameters_Complex in Mod_ADo_SQL<- Purpose
     --------------------------------------------------------------------
     --> Required_Elements: tbl_zs_StartID <- Required_Elements
     --------------------------------------------------------------------
     -->Returns: Recordset
     --------------------------------------------------------------------
    Documentation Above */

Declare @vcSQL AS varchar(255)
Declare @vcIDField AS varchar(55)

set @vcOutPut1 = '-100'

Select @vcSQL = 'delete from ' + @vcTableInToName
exec(@vcSQL)

set @vcOutPut1 = '-90'

--What field do we put the data into
If @btNumericId_YN =0
Begin
    Select @vcIDField = 'IdT'
End
else
Begin
    Select @vcIDField = 'Id'
End

set @vcOutPut1 = '-80'

--Put the Id from the records in the current form into the table
Select @vcSQL = 'INSERT INTO ' + @vcTableInToName + '( ' + @vcIDField + ')'
    + ' SELECT ' + @vcIDName + ' FROM ' + @vcTableFromName + ' Where ' + @vcWhere
exec(@vcSQL)

set @vcOutPut1 = '-70'

--SELECT @chmsg = 'We are Done.'
--select @vcOutPut1 = str(@@rowcount)

SELECT vw_Companies.Co_Alpha_Name, vw_Companies.Bill_Cty,
    vw_Companies.Bill_St
FROM vw_Companies RIGHT OUTER JOIN
    tbl_zs_StartID ON vw_Companies.Co_IdT = tbl_zs_StartID.IdT
select @vcOutPut1 = ' --->' + Ltrim(str(@@rowcount) + ' Records')
return 10
#4 - What Access Calls Action Queries
(Delete data, Append Data, Update Data, Make Tables)

* In MS Access we have select queries that would return a result set similar to figure 2-5. In addition we have the following type of queries that manipulates the data in the tables.

Delete data
* Ok, so I made it a little more difficult than it had to be. To delete records from a table you can just have one line in the procedure:
   `delete tbl_City where City_Id = @intId`
You can pass an input parameter
Create Procedure abc
   `@intId as Int`
as
   `delete tbl_City where City_Id = @intId`
return

The Easy
* In figure 4-1 we pass a complete where statement in the input parameter, and are looking for a record count in the output parameter. We have to deal with the SQL statement as a string, and then execute it. You might consider using “With recompile” if you are passing a complete Where statement.

* In figure 4-2 we run the Proc with ADO Code.

Figure 4-1  Delete Records—Stored Procedure

Figure 4-2  ADO to Run the Above Delete Action

```
Public Function ex_SP_QueryDelete() As Boolean

    '>>> Stored Procedure & ADO are about the same Speed <<<
    '----
    ' Purpose: Use a stored Procedure to run the delete query
    ' Required Elements: Stored Procedure  -->  z_sp_qry_DeleteRecords_PassWhere
    ' Example: ex_SP_QueryDelete()
    '----
    ' Parameters:
    '----
    ' Returns:
    '---
    Dim Cmd1 As ADODB.Command
    Dim strWhereStatement As String
    '----
    strWhereStatement = "City like 'h%'"
    'Check the Connection - If no connection try to make one
    If Not CnnCheckConnection() Then GoTo ProcedureDone
    Set Cmd1 = New ADODB.Command
    Set Cmd1.ActiveConnection = cnn
    With Cmd1
        .CommandText = "z_sp_qry_DeleteRecords_PassWhere"
        .CommandType = adCmdStoredProc 'adCmdTable adCmdText adCmdStoredProc
        .Parameters.Refresh
        .Parameters("@vcWhere").Value = strWhereStatement
    End With
    Cmd1.Execute
    ex_SP_QueryDelete = Cmd1.Parameters("RETURN_VALUE").Value
    Debug.Print "Records Deleted: --> " & Cmd1.Parameters("@inRecCount").Value
    Set Cmd1 = Nothing
    ex_SP_QueryDelete = True
ProcedureDone:
```
Update data
* In figure 4-3 we see how to add 1 year to a date in a table using the T-SQL update process. Notice the SQL Server built-in Dateadd function. Look at the last 4 pages of this document for some additional SQL Server built-in functions.

* In figure 4-4 we see how to update a field in one row of data, with data from 1 row from another table.

Make a Table
* In figure 4-5 we are creating a table with data from another recordset. When we get finished data will be in the table.

* In figure 4-6 we create a table with no data.
Append data
* In figure 4-7 we are inserting rows of data from one table to another table.

* In figure 4-8 we get a little more complex. We can run this procedure with the code in Section A or Section B, not both. In Section A we “Select * Into tmpCity” which will create table Tmpcity and then put data into the table. In Section B we create the table, Set the Primary Key. Tell SQL Server don’t pay attention to Primary keys we insert, then we insert the data.

> We delete table tmpCity if it exist.

**Figure 4-7 Simple Append Proc**
Append data from 1 table to another table

```
CREATE Procedure z_sp_Qry_Append AS

SET NOCOUNT ON
-- We are using a perrinet table to append data - however a true temp table starts with # see note below
-- Use "#" before the table name for a temp table used only by the current user and deleted after The procedure is done
-- (if proc 1 calls proc 2 and proc 2 calls proc 3 you can use the #Table in any of the 3 procs)
-- Use "##" before the table name for a temp table used by all

-- delete Temp Table if it exists.
  if exists (SELECT * FROM sysobjects WHERE [id] = OBJECT_ID('dbo.tmpCity')) drop table dbo.tmpCity

-- Create the Temp Tables and put in records
  select * into tmpCity from tblCity
  where tbl.City.City like '%.2'
  Order by City
```

**Figure 4-8 Complex Append Proc**

```
CREATE Procedure z_sp_Qry_AppendComplex AS

-- Create the Temp Tables and put in records
CREATE TABLE [dbo].[tmpCity] (  
    [City_Id] [int] IDENTITY (1, 1) NOT NULL,  
    [City] [varchar] (30) NOT NULL,  
    [County] [varchar] (11) NULL,  
    [Ab_State] [varchar] (2) NULL,  
    [State] [varchar] (20) NULL,  
    [Last_Updated] [datetime] NULL,  
    [Updated_By] [varchar] (50) NULL,  
    [City_ts] [timestamp] NOT NULL) ON [PRIMARY]

-- Set Primary Key
ALTER TABLE [dbo].[tmpCity] WITH NOCHECK ADD CONSTRAINT [PK_tmpCityNew] PRIMARY KEY NONCLUSTERED ([City_Id]) WITH FILLFACTOR = 50 ON [PRIMARY]

-- SET IDENTITY_INSERT to ON.
SET IDENTITY_INSERT tmpCity ON

-- Put in Records
INSERT INTO tmpCity (City_Id, City, County, Ab_State, State, Last_Updated, Updated_By)
SELECT City_Id, City, County, Ab_State, State, Last_Updated, Updated_By FROM dbo.tblCity
where tbl.City.City like '%.2'
Order by City

-- SET IDENTITY_INSERT to Off
SET IDENTITY_INSERT tmpCity Off

-- Show all records in the temp table
select * from tmpCity

return
```
If you are like me and use the “IIf” statement in Access queries, you are going to want to know what you can replace it with in SQL Server. There are no replacements in Views, however in SPS you can use the case statement. In figure 5-1 we have a SP that looks at the field Mail_St which is a 2 character field for the state. If it = KS we substitute Kansas, if MO we use Missouri, otherwise we use the actual value in the field Mail_St. You can see how it comes out in figure 5-2.
#6 - Additional Information

### Figure 6-1  
**>>Numeric Functions<<**

<table>
<thead>
<tr>
<th>SQL</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Floor(7.234)</td>
<td>Convert to integer</td>
</tr>
<tr>
<td>Round(765.4321, 2)</td>
<td>select Round(765.4321, 2) returns 765.43</td>
</tr>
</tbody>
</table>

### Figure 6-2  
**>>Type Conversions<<**

<table>
<thead>
<tr>
<th>SQL</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Convert(int, X)</td>
<td>Convert to Integer</td>
</tr>
<tr>
<td></td>
<td>Clnt(&quot;876.54&quot;) equals 877</td>
</tr>
<tr>
<td>Convert(float, X)</td>
<td>Convert to Double Precision</td>
</tr>
<tr>
<td>convert(money, X)</td>
<td>Convert to Currency</td>
</tr>
<tr>
<td>convert(varchar, X)</td>
<td>Convert to String</td>
</tr>
<tr>
<td>Convert(DateTime, X)</td>
<td>Convert to Date/Time</td>
</tr>
</tbody>
</table>

### Figure 6-3  
**>>Misc. Information<<**

<table>
<thead>
<tr>
<th>Explanation</th>
<th>SQL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Date Delimiter</td>
<td>SQL-&gt;Between ‘1/1/01’ and ‘12/31/01’</td>
</tr>
<tr>
<td>String Delimiter</td>
<td>SQL -&gt; ‘Gordon’ + ‘,’ + ‘Larry’</td>
</tr>
<tr>
<td>Concatenation Operator</td>
<td>SQL -&gt; ‘Gordon’ + ‘,’ + ‘Larry’</td>
</tr>
<tr>
<td>Wildcard Character (Any one character)</td>
<td>SQL -&gt; select last_Name from tbl_Individuals where last_name like ‘Gor_on’</td>
</tr>
<tr>
<td>Wildcard Character (Any group of characters)</td>
<td>SQL -&gt; select last_Name from tbl_Individuals where last_name like ‘Gor%d’</td>
</tr>
<tr>
<td>True/Yes Bit type data</td>
<td>1</td>
</tr>
<tr>
<td>False/No Bit type data</td>
<td>0</td>
</tr>
<tr>
<td>SQL</td>
<td>Explanation</td>
</tr>
<tr>
<td>-----</td>
<td>-------------</td>
</tr>
<tr>
<td>Replace('aabbccdd', 'bb', 'xx')</td>
<td>Replace all 'bb' in the original string with 'xx'</td>
</tr>
<tr>
<td>CharIndex(&quot;XYZ&quot;, &quot;Y&quot;)</td>
<td>Find a position of a particular string select CHARINDEX('Joe', 'Smith, Joe') returns 8</td>
</tr>
<tr>
<td>IsNull([Price], 0) IsNull([Price], 'Free')</td>
<td>If the price is null, return 0, else return the Price If the price is null, return Free, else return the Price</td>
</tr>
<tr>
<td>(([Dt_Join] IS NULL) Not ([Dt_Join] IS NULL))</td>
<td>Check to see if a value is null select * from tbl_Companies where not ([Dt_Join] IS NULL)</td>
</tr>
<tr>
<td>Left('ABCDE', 2)</td>
<td>Left characters of a string Left('ABCDE', 2) returns AB</td>
</tr>
<tr>
<td>Right('ABCDE', 2)</td>
<td>Right characters of a string Right('ABCDE', 2) returns DE</td>
</tr>
<tr>
<td>Substring(&quot;Test This&quot;,6, 20) Substring(Expression, Start, Length)</td>
<td>In SQL Server you have to put the length, however in Access you are not required to have the length. The secret in SQL Server is to put the maximum length it could ever be (if it's greater than string length, that's not a problem).</td>
</tr>
<tr>
<td>Ltrim(x)</td>
<td>Trim the spaces off the Left of a string Ltrim(&quot; SQL&quot;) returns “SQL”</td>
</tr>
<tr>
<td>Rtrim(x)</td>
<td>Trim the spaces off the Right of a string Rtrim(&quot;SQL “) returns “SQL”</td>
</tr>
<tr>
<td>Len(X) or DataLength(x)</td>
<td>select LEN('This is a test') returns 14</td>
</tr>
<tr>
<td>Space(X)</td>
<td>Give you X number of spaces e.g. Select Space(22) + 'aabbccdd'</td>
</tr>
<tr>
<td>Ascii(x)</td>
<td>Returns the ASCII value of a character Asc(&quot;A&quot;) will return 65</td>
</tr>
<tr>
<td>Char(x)</td>
<td>Returns a character associated with the specified character code. Chr(65) will return A</td>
</tr>
<tr>
<td>Str(X)</td>
<td>Converts a number to a string Str(1234) returns &quot;1234&quot;</td>
</tr>
<tr>
<td>Lower(x)</td>
<td>Change to lower case SELECT Lower('THIS IS HOW THE MAIN FRAME PROGRAMMERS USE TO DO IT')</td>
</tr>
<tr>
<td>Upper(x)</td>
<td>Change to UPPER case</td>
</tr>
</tbody>
</table>
## Date/Time Functions

<table>
<thead>
<tr>
<th>SQL</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Getdate</strong>()</td>
<td>SQL Server returns <code>2001-05-24 10:37:09.043</code>&lt;br&gt;<code>GetDate()</code> Gets Date &amp; Time - See “Style in Date Convert” below.</td>
</tr>
</tbody>
</table>

### Convert

```sql
Convert(data_type{}, expression [, style])
```

**In SQL Server**

```sql
select date_Invoice, convert(varchar, date_Invoice, 1) as x from tbl_invoice
```

Returns:

<table>
<thead>
<tr>
<th>Style</th>
<th>Date</th>
<th>Style</th>
<th>Date</th>
<th>Style</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4/12/01</td>
<td>101</td>
<td>4/12/2001</td>
<td>2</td>
<td>01.04.12</td>
</tr>
<tr>
<td>7</td>
<td>Apr 12, 01</td>
<td>107</td>
<td>Apr 12, 2001</td>
<td>0</td>
<td>Apr 12 2001 12:00AM</td>
</tr>
</tbody>
</table>

```sql
select convert(varchar, getdate(), 8) returns hh:mm:ss
```

13:02:57

### DatePart

```sql
DatePart(M, '5/22/99')
```

Get a part of a date - Select `DatePart(M, '5/22/99')` returns 5

### DateAdd

```sql
DateAdd(M, 2, '5/22/99')
```

Does Date addition and subtraction

```
DateAdd(interval, number, date) Interval - see the constants below
The number can be a positive or negative number
```

### DateDiff

```sql
DateDiff(M, pubdate, getdate())
```

Get the difference between 2 dates

```
DateDiff(interval, number, date) Interval - see the constants below
```

```sql
select date_Invoice, DATEDIFF(d, date_Invoice, getdate()) as x from tbl_invoice
```

### Constants

- q, qq Quarter
- m, mm Month
- y, dy Day of Year
- d, dd Day
- ww, wk Week
- dw WeekDay
- hh Hour
- mi, n Minute
- s, ss Second
- ms millisecond
- yy, yyyy Year
<table>
<thead>
<tr>
<th>SQL Data Type</th>
<th>Explanation Of SQL Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>bit</td>
<td>Integer data with either a 1 or 0 value. Columns of type bit cannot have indexes on them. (It can be Null, but null can give you trouble later. I recommend you don't allow Nulls) Access stores True as –1 and False as 0 inside a Access table, however Access has no problems interpreting bit data - 1 = True and 0 = False.</td>
</tr>
<tr>
<td>int</td>
<td>Integer (whole number) data from -2^31 (-2,147,483,648) through 2^31 - 1 (2,147,483,647). About 2 billion minus to 2 billion plus</td>
</tr>
<tr>
<td>smallint</td>
<td>Integer data from 2^15 (-32,768) through 2^15 - 1 (32,767).</td>
</tr>
<tr>
<td>tinyint</td>
<td>Integer data from 0 through 255.</td>
</tr>
<tr>
<td>decimal</td>
<td>Fixed precision and scale numeric data from -10^38 -1 through 10^38 -1.</td>
</tr>
<tr>
<td>numeric</td>
<td>same as decimal</td>
</tr>
<tr>
<td>money</td>
<td>Monenary data values from -2^63 (-922,337,203,685,477.5808) through 2^63 - 1 (+922,337,203,685,477.5807), with accuracy to a ten-thousandth of a monetary unit.</td>
</tr>
<tr>
<td>Small money</td>
<td>Monenary data values from -214,748.3648 through +214,748.3647, with accuracy to a ten-thousandth of a monetary unit.</td>
</tr>
<tr>
<td>float</td>
<td>Floating precision number data from -1.79E + 308 through 1.79E + 308.</td>
</tr>
<tr>
<td>real</td>
<td>Floating precision number data from -3.40E + 38 through 3.40E + 38.</td>
</tr>
<tr>
<td>datetime</td>
<td>Date and time data from January 1, 1753, to December 31, 9999, with an accuracy of three-hundredths of a second, or 3.33 milliseconds.</td>
</tr>
<tr>
<td>small-datetime</td>
<td>Date and time data from January 1, 1900, through June 6, 2079, with an accuracy of one minute.</td>
</tr>
<tr>
<td>timestamp</td>
<td>A database-wide unique number. A table can have only one timestamp column. The value in the timestamp column is updated every time a row containing a timestamp column is inserted or updated.</td>
</tr>
<tr>
<td>uniqueidentifier</td>
<td>A globally unique identifier (GUID).</td>
</tr>
<tr>
<td>char</td>
<td>Fixed-length non-Unicode character data with a maximum length of 8,000 characters.</td>
</tr>
<tr>
<td>varchar</td>
<td>Variable-length non-Unicode data with a maximum length of 8,000 characters.</td>
</tr>
<tr>
<td>text</td>
<td>Variable-length non-Unicode data with a maximum length of 2^31 - 1 (2,147,483,647) characters.</td>
</tr>
<tr>
<td>nchar</td>
<td>Fixed-length Unicode data with a maximum length of 4,000 characters.</td>
</tr>
<tr>
<td>nvarchar</td>
<td>Variable-length Unicode data with a maximum length of 4,000 characters. sysname is a system-supplied user-defined data type that is a synonym for nvarchar(128) and is used to reference database object names.</td>
</tr>
<tr>
<td>ntext</td>
<td>Variable-length Unicode data with a maximum length of 2^30 - 1 (1,073,741,823) characters.</td>
</tr>
<tr>
<td>binary</td>
<td>Fixed-length binary data with a maximum length of 8,000 bytes.</td>
</tr>
<tr>
<td>varbinary</td>
<td>Variable-length binary data with a maximum length of 8,000 bytes.</td>
</tr>
<tr>
<td>image</td>
<td>Variable-length binary data with a maximum length of 2^31 - 1 (2,147,483,647) bytes.</td>
</tr>
</tbody>
</table>