



We Translate  
Business Processes

from the Mind  
to the Computer  
to the Bottom Line.

**BUSINESS & COMPUTERS, Inc.**  
13839 Mur-Len Rd, Suite M  
OLATHE, KANSAS 66062

Phone: (913) 764-2311  
Fax: 764 7515  
larryg@kcnet.com

# Stored Procedures for SQL Server

## The Basics

Copyright© 2001 Business & Computers, Inc.

**A note – the below is my humble opinion – with testing – If you use my ideas please test them and if you have problems or learn more let me know.**

### #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 an Input Parameter

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

**Figure 2-1** Stored Procedure with input parameter & recordset

```

Text:
CREATE PROCEDURE z_sp_SimpleReadTable
@vcCompanyName as varchar(50)
AS
--Set nocount on

SELECT Co_IdT, Co_Alpha_Name, Dt_Join
FROM dbo.tbl_Companies
Where not([Dt_Join] is null) and Co_Alpha_Name like @vcCompanyName + '%'

return
    
```

Labels in the screenshot: Stored Procedure Name, Parameter, Check Syntax, Run Button, Database running against, Result set.

Co_IdT	Co_Alpha_Name	Dt_Join
796	Bush Robert W Construction Company	1989-07-01 00:00:00.000
266	Business & Computers Inc	1979-07-01 00:00:00.000
1308	Busby Brothers	1993-07-01 00:00:00.000

(3 row(s) affected)

**Figure-2-2**  
Running a procedure on Query Analyzer

- \* 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.

**Figure 2-3**

```

z_sp_SimpleReadTable 'Bus'
sp_Help 'tmpcity'
    
```

- \* You can also run the query in an Access Pass-Through Query.

**Figure 2-4** MS Access Pass-Through Query

**Figure 2-5** Pass-Through Query Results

Co_IdT	Co_Alpha_Name	Dt_Join
796	Bush Robert W Construction Company	7/1/1989
266	Business & Computers Inc	7/1/1979
1308	Busby Brothers	7/1/1993

Record: 1 of 3

\* 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.

**Figure 2-4 ADO using Proc for recordset**

```
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 fro 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
```

### #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.

z\_sp\_In\_Out\_Parameters\_Simple '266', "

**print @vcOutPut1**

**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_Alpha_Name 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=SQLOLEDB.1;" & _
    "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:
    
```

**Figure 3-4** Complex Input and Output Parameters

\* This is a bit of a complex stored procedure. I won't go over it today, but some times we want to see a direction we are heading.

```

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

/* Documentation Below
'----
--> 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 '
    exec(@vcSQL + @vcWhere)

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

```
Text:  
CREATE PROCEDURE z_sp_Qry_DeleteRecords_PassWhere  
(@vcWhere AS varchar(8000), @inRecCount AS int output)  
AS  
  
    Declare @vcSQL AS varchar(255)  
    Declare @vcRowCnt as varchar(255)  
  
    -- Run the stored procedure zx_CreateTempTable_AddData to create the table  
  
    select @vcSQL = 'delete from tmpCity where '  
    exec(@vcSQL + @vcWhere)  
    --select vcRowCnt =str(@@rowcount)  
  
    Select @inRecCount =@@rowcount  
  
return 0
```

**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.

**Figure 4-3**

Update the date (dt\_expire) by 1 year

```
Text:
CREATE PROCEDURE z_sp_Qry_Update AS
update tmpTbl_Companies
set dt_Expire= dateadd(yy,1,dt_expire)
Where Co_Alpha_Name like 'b%' and dt_Expire is not null
```

**Figure 4-4**

Update the data in one table with data from another table.

```
Text:
-- Update 1 table using data in another table
update tbl_Permits
SET tbl_Permits.Invoice_No = tbl_zs_PermitInvoicing.Invoice_Number,
tbl_Permits.Billed = tbl_zs_PermitInvoicing.Billed,
tbl_Permits.Quarter = tbl_zs_PermitInvoicing.Quarter
FROM dbo.tbl_zs_PermitInvoicing JOIN dbo.tbl_Permits ON
dbo.tbl_zs_PermitInvoicing.Pmt_Idd = dbo.tbl_Permits.Pmt_Idd
```

## 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.

**Figure 4-5 Make a Table with Data from Another Recordset**

```
Text:
CREATE PROCEDURE z_sp_Qry_MakeTable
AS
-- If making a perminet table - you need go to the Properties for the database
-- and check Select into / Bulk Copy

if exists (select * from sysobjects where id=object_id('[(dbo].[AAA])') and OBJECTPROPERTY(id, 'IsTable')=1)
drop table [AAA]

SELECT Co_IdT, Co_Alpha_Name
into AAA
FROM tbl_Companies
WHERE (Co_Alpha_Name LIKE 'bus%')

Return
```

**Figure 4-6 Make a Table with No Data**

```
Text:
CREATE PROCEDURE aa3
AS

if exists (select * from sysobjects where id = object_id(N'[(dbo).[aa_tbl_City_States]') and OBJECTPROPERTY(id, N'IsUserTable') =
drop table [(dbo).[aa_tbl_City_States]

CREATE TABLE [(dbo).[aa_tbl_City_States] (
[St_Abbrev] [varchar] (2) NOT NULL ,
[State_Name] [varchar] (50) NOT NULL ,
[CityState_ts] [timestamp] NULL
) ON [PRIMARY]

ALTER TABLE [(dbo).[aa_tbl_City_States] WITH NOCHECK ADD
CONSTRAINT [aaa_tbl_City_States] PRIMARY KEY NONCLUSTERED
([St_Abbrev]) WITH FILLFACTOR = 90 ON [PRIMARY]
```



## 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.

**Figure 4-7 Simple Append Proc**

Append data from 1 table to another table

Text:

```
CREATE Procedure z_sp_Qry_Append As
INSERT INTO tmpCity (City, County)
SELECT City, County FROM tbl_City
where tbl_City.City like 'H%'
Order by City
```

**Figure 4-8 Complex Append Proc**

```
Text:
Create Procedure z_sp_Qry_AppendComplex
AS

SET NOCOUNT ON
-- We are using a perminet 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 your 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 tbl_City
where tbl_City.City like 'k%'
Order by City */

-- 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 = 90 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.tbl_City
where tbl_City.City like 'k%'
Order by City

-- SET IDENTITY_INSERT to Off
SET IDENTITY_INSERT tmpCity Off

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

return
```

> We delete table tmpCity if it exist.

Section A

Section B



## #5 - Case Statement

- \* 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*.

Figure 5-1

```
cc: Stored Procedure
Alter Procedure cc
As
set nocount on
SELECT Co_Alpha_Name, Mail_City,
State =
    Case Mail_St
    When 'KS' then 'Kansas'
    When 'MO' then 'Missouri'
    else Mail_St
    end
FROM vw_Companies where not (Mail_St is null)
return
```

Figure 5-2

Co Alpha Name	Mail City	State
Welcome Home Properties	St Charles	Missouri
Carpet Alliance	Lee's Summit	Missouri
Valley Floor Covering	Lee's Summit	Missouri
Gibson Companies Inc The	Kansas City	Missouri
Remodelers Council	Kansas City	Missouri
Burke Construction	Lee's Summit	Missouri
Leader Mortgage	Lenexa	Kansas
RARE Enterprise	Kansas City	Missouri
Blind Guy, The	Harrisonville	Missouri
Veronicas Construction	Kansas City	Missouri
Witkin Custom Homes Inc	Englewood	CO
Gach Mark Construction Inc	St Joseph	Missouri
Keach & Grove Real Estate	Bedford	IN

## #6 - Additional Information

Figure 6-1 >>Numeric Functions<<	
SQL	Explanation
Floor(7.234)	Convert to integer
Round(765.4321, 2)	select Round(765.4321, 2) returns 765.43

Figure 6-2 >>Type Conversions<<	
SQL	Explanation
Convert(int, X)	Convert to Integer CInt("876.54") equals 877
Convert(float, X)	Convert to Double Precision
convert(money, X)	Convert to Currency
convert(varchar, X)	Convert to String
Convert(DateTime, X)	Convert to Date/Time

Figure 6-3 >>Misc. Information<<		
Explanation		SQL
Date Delimiter	SQL->Between '1/1/01' and '12/31/01'	'
String Delimiter	SQL -> 'Gordon' + ', ' + 'Larry'	'
Concatenation Operator	SQL -> 'Gordon' + ', ' + 'Larry'	+
Wildcard Character (Any <i>one</i> character) Where Last like "Gor_on"	SQL -> select last_Name from tbl_Individuals where last_name like 'Gor_on'	_
Wildcard Character (Any <i>group</i> of characters)	SQL -> select last_Name from tbl_Individuals where last_name like 'Gord%'	%
True/Yes	Bit type data	1
False/No	Bit type data	0

**Figure 6-4****>>String Functions<<**

SQL	Explanation
Replace('aabbccdd', 'bb', 'xx')	Replace all 'bb' in the original string with 'xx'
CharIndex("XYZ", "Y")	Find a position of a particular string <code>select CHARINDEX('Joe', 'Smith, Joe')</code> returns 8
IsNull([Price], 0) IsNull([Price], 'Free')	If the price is null, return 0, else return the Price If the price is null, return Free, else return the Price
([Dt_Join] IS NULL) Not ([Dt_Join] IS NULL)	Check to see if a value is null <code>select * from tbl_Companies where not ([Dt_Join] IS NULL)</code>
Left('ABCDE', 2)	Left characters of a string <code>Left('ABCDE', 2)</code> returns AB
Right('ABCDE', 2)	Right characters of a string <code>Right('ABCDE', 2)</code> returns DE
Substring("Test This",6, 20) Substring(Expression, Start, Length)	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).
Ltrim(x)	Trim the spaces off the Left of a string <code>Ltrim(" SQL")</code> returns "SQL"
Rtrim(x)	Trim the spaces off the Right of a string <code>Rtrim("SQL ")</code> returns "SQL"
Len(X) or DataLength(x)	<code>select LEN('This is a test')</code> returns 14
Space(X)	Give you X number of spaces e.g. <code>Select Space(22) + 'aabbccdd'</code>
Ascii(x)	Returns the ASCII value of a character <code>Asc("A")</code> will return 65
Char(x)	Returns a character associated with the specified character code. <code>Chr(65)</code> will return A
Str(X)	Converts a number to a string <code>Str(1234)</code> returns "1234"
Lower(x)	Change to lower case <code>SELECT Lower('THIS IS HOW THE MAIN FRAME PROGRAMMERS USE TO DO IT')</code>
Upper(x)	Change to UPPER case

**Figure 6-5**

**>> Date/Time Functions <<**

SQL	Explanation
Getdate()	SQL Server returns 2001-05-24 10:37:09.043 GetDate() Gets Date & Time - See "Style in Date Convert" below.
Convert( <i>data_type</i> ( <i>length</i> ), <i>expression</i> [, <i>style</i> ])	<b>In SQL Server</b> select date_Invoice, convert(varchar, date_Invoice, 1) as x from tbl_invoice Returns: 2001-04-12 00:00:00.000 4/12/01 2001-04-04 00:00:00.000 4/04/01 Style Date Style Date Style Date 1 4/12/01 101 4/12/2001 2 01.04.12 7 Apr 12, 01 107 Apr 12, 2001 0 Apr 12 2001 12:00AM  select convert(varchar, getdate(), 8) returns hh:mm:ss 13:02:57
DatePart(M, '5/22/99')	Get a part of a date - Select DatePart(M, '5/22/99') returns 5
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(M, pubdate, getdate() ( ))	Get the difference between 2 dates DateDiff(interval, number, date) Interval - see the constants below  select date_Invoice, DATEDIFF(d, date_Invoice, getdate()) as x from tbl_invoice
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

**Figure 6-6 SQL Data Type**

SQL Data Type	Explanation Of SQL Data Type
bit	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.
int	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
smallint	Integer data from $2^{15}$ (-32,768) through $2^{15} - 1$ (32,767).
tinyint	Integer data from 0 through 255.
decimal	Fixed precision and scale numeric data from $-10^{38} - 1$ through $10^{38} - 1$ .
numeric	same as decimal
money	Monetary 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.
Small money	Monetary data values from -214,748.3648 through +214,748.3647, with accuracy to a ten-thousandth of a monetary unit.
float	Floating precision number data from $-1.79E + 308$ through $1.79E + 308$ .
real	Floating precision number data from $-3.40E + 38$ through $3.40E + 38$ .
datetime	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.
small-datetime	Date and time data from January 1, 1900, through June 6, 2079, with an accuracy of one minute.
timestamp	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.
uniqueidentifier	A globally unique identifier (GUID).
char	Fixed-length non-Unicode character data with a maximum length of 8,000 characters.
varchar	Variable-length non-Unicode data with a maximum of 8,000 characters.
text	Variable-length non-Unicode data with a maximum length of $2^{31} - 1$ (2,147,483,647) characters.
nchar	Fixed-length Unicode data with a maximum length of 4,000 characters.
nvarchar	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.
ntext	Variable-length Unicode data with a maximum length of $2^{30} - 1$ (1,073,741,823) characters.
binary	Fixed-length binary data with a maximum length of 8,000 bytes.
varbinary	Variable-length binary data with a maximum length of 8,000 bytes.
image	Variable-length binary data with a maximum length of $2^{31} - 1$ (2,147,483,647) bytes.