

Upsize ASP From Access To SQL Server

Scale your Web apps up to SQL Server.

by A. Russell Jones

WHAT YOU NEED

Microsoft Access 95
or 97

SQL Server 6x

Access Upsizing Wizard

You have an Active Server Pages (ASP) application that's been performing yeoman service for your department. The application is so good you've been asked to deploy it company-wide. You developed the application using a Microsoft Access database and you've heard it's easy to "upsized" your Access database to Microsoft SQL Server using the free upsizing wizard, so you'll just get the wizard, upsize your database, and be done. Hey! Not so fast. If you don't want to get a 6 a.m. call from your boss who's unable to demo your application to the CEO because you didn't upsize your database correctly, you should consider some issues first. This column will review these issues and guide you down a successful migration path from Access to SQL Server.

Why move from Access? Access is a desktop and workgroup database that was designed to be robust, powerful, and easy to use—but Microsoft didn't design it to be used by many people simultaneously. SQL Server is an all-around better choice for a Web database than Access (see Table 1).

There are major differences in the way the two databases work internally, but from a development viewpoint, the external interfaces—creating a connection and retrieving or updating data—are similar. Both databases use SQL as the query language and are generally ANSI-89 and ANSI-92 compliant, but they use different "flavors" of SQL. The basic differences are in naming conventions, in reserved words (which can cause problems with object names), in defined types, and in parameter syntax and types for stored procedures (SQL Server) and queries (Access).

The easiest way to upsize is to use an Access add-in program available from Microsoft called the Access Upsizing Wizard (see the "Where to Find the Wizard" sidebar). Alternatively, you can use the Database Documenter tools to print the structure of your database, then create SQL Data Definition Language

(DDL) statements to create the tables for your database. However, this is labor-intensive, and you need a strong knowledge of SQL to be successful. Unfortunately, Microsoft has not yet released a Wizard version that upsizes directly to SQL Server 7, so you need to upsize to version 6.5, then use the transfer tools to move the database to SQL Server 7.

Prepare for Upsizing

Check your Access database table, field, index, and query names for SQL Server reserved words—you need to rename any reserved words you find before upsizing (see the Transact-SQL Help topic "reserved words" for a list). Make sure all your table, field, index, and query names are no longer than 30 characters; contain no spaces; start with a letter; and contain no non-alphanumeric characters except the dollar sign (\$) and the underscore (_).

Some Access data types have no counterpart in SQL Server. Yes/No, True/False, On/Off, Hyperlink, ReplicationID, Memo, and OLE Object data types aren't supported under those names. The wizard will migrate these to appropriate (but not always equivalent) SQL Server data types (see Table 2).

Open the new copy of your Access database and compress it. Next, calculate the approximate size of your SQL Server database. A good rule of thumb—the database's SQL Server version needs to be at least twice the size of your Access database (check the Access database file size in Windows Explorer).

SQL Server (up to version 7.0) handles its own I/O, so you need to create database "devices" on which SQL Server stores objects and data. Although the Upsizing Wizard can create devices for you, I recommend you create them manually using SQL Server Enterprise Manager because it provides more control and reliability for device creation. You should create at least two devices—one for the database and

one for the transaction log. For maximum performance and for recovery in case of a disk failure, create the log device on a different physical disk than the device for the database. The database device must be at least as large as the required database size you just calculated. The log device must be at least as large as your largest Access table; otherwise you need to perform several passes with the Upsizing Wizard. In general, don't skimp on device or database sizes. Once the devices are complete, create the database on the database device you made for the database and the transaction log on the log device.

One more step, then you can run the Upsizing Wizard. From Settings | Control Panel | ODBC, create a new system Data Source Name (DSN) on your machine with the Access database, that points to the new database on SQL Server. Create a second DSN on your Web server. For both DSNs, specify

that you want to run TCP/IP Sockets as the client Net Library. Make sure the connections work by clicking on the Test Data Source button at the end of the ODBC Data Source Wizard.

Run the Upsizing Wizard

Finally, open your Access database and run the Upsizing Wizard. Click on Tools | AddIns and select the Upsizing Wizard to begin. If you don't see it, click on the AddIn Manager and check the Upsizing Wizard. Select the "Use Existing Database" radio button, and then select the DSN you just created. SQL Server 6x can use declarative referential integrity (DRI) to maintain relationships between tables, but triggers are much more flexible: They're code you can modify to suit your needs. I recommend you select the "Use triggers" option to maintain data relationships. Select "No, never"

Feature Support	Access	SQL Server
Multuser capable	Designed for workgroup-level multuser access, at maximum	Can scale to hundreds of simultaneous users
Data services	A file-serving database that returns both index info and data. Not only must more info pass between the calling program and the database, but some data is useless to anything but the Jet DLLs. If you have more than one client, it adds up in a hurry.	A data server—returns data only
Multithreaded data engine	Jet engine isn't multithreaded, whereas the Web server is—causing problems when users try to access the database simultaneously. Access databases are uncertain for more than one user (in read/write mode).	Capable of servicing multiple simultaneous threads
Transactions	Yes, but not through ADO version 1.5	Yes
Nested Transactions	Yes	Yes
Triggers (event code that fires when rows are inserted, updated, or deleted)	No—table "relationships" can enforce data integrity	Supports triggers
Stored procedures	Supports "queries," limited to a single statement, which are similar to SQL Server views	Stored procedures may have multiple statements
Conditional statements	Supported only through VBA	Yes
Views	Supports "queries," which can return updatable recordsets	Yes
Cascading deletes (to eliminate "orphaned" data)	Yes, with relationships	Yes, with triggers
Dependency information	Indirectly, through the Database Documenter add-in	Yes; you can generate dependency information for any object (table, view, index, stored procedure)

Table 1 Access Versus SQL Server. SQL Server is a better database choice than Access. Here's why.

LINKS

www.microsoft.com/AccessDev/Articles/autwp.htm

from the “Add timestamp fields to tables?” combo box. You don’t need to link the new SQL Server tables, so uncheck that option. Click on “Next” and check the “Create Upsizing Report” checkbox, then click on “Finish” to begin the upsizing process. When the wizard is finished, check the error log for warnings or errors.

Now that you have a SQL Server database, you’re done, right? No. Sorry, but the Upsizing Wizard doesn’t translate your queries. But don’t worry—it’s not difficult to translate most queries. In fact, most queries without parameters run without modification.

In Access SQL, parameter queries begin with the keyword PARAMETERS, then a space, the first parameter name, another space, then the parameter type. Commas separate multiple parameters. A semicolon separates the parameter definition from the rest of the query. Square brackets surround parameters in the body of the query:

```
PARAMETERS Signon Text, Password Text;
SELECT Users.Signon, Users.Password,
       Users.LastName, Users.FirstName,
       Users.LastOn, Users.TimedOut
FROM Users
WHERE (((Users.Signon)=[Signon]) AND
      ((Users.Password)=[Password]));
```

In Transact-SQL, you declare parameters immediately after the procedure’s name, and there’s no semicolon. All parameter names begin with an “@” symbol, both in the parameter definition and in the procedure body:

```
CREATE Procedure
    getUserBySignon_Password @UserID
    varChar(20), @Password varChar(20)
AS
SELECT Users.Signon, Users.Password,
       Users.LastName, Users.FirstName,
       Users.LastOn, Users.TimedOut
FROM Users
WHERE (((Users.Signon)=@Signon) AND
      ((Users.Password)=@Password));
```

The Select statements are exactly the same. Only the parameter declaration is different, so for many queries, replacing the PARAMETER statement and the parameter references suffices. Use the data type translation table to select which Transact-SQL data type to use (see Table 2).

You might also have a problem with True/False, Yes/No, and On/Off values. The equivalent data type in SQL Server is

called “bit.” In Access, these data types have values of either –1 (True) or 0 (False); but in Transact-SQL, 1 is True and 0 is False. To make things worse, the keywords True and False are undefined. If you have code that explicitly tests for a Boolean value of True, False, or –1, you need to modify it. The best way is to test for zero or test for not zero:

```
' this select statement will not work
' in SQL Server
SELECT * FROM Orders WHERE
    OrderShipped=True
' this one will
SELECT * FROM Orders WHERE
    OrderShipped <> 0
```

In Access, create complex queries by selecting fields from one or more base queries to obtain a resultset. For example, if you have a query that selects a specific customer, you can join that with a query that selects all that customer’s orders along with the customer information:

```
PARAMETERS CustID Long;
SELECT getCustomer.*,
       getOrdersForCustomer.* FROM
getOrdersForCustomer INNER JOIN
```

```
getCustomer ON
getOrdersForCustomer.CustID=
getCustomer.CustID
```

In SQL Server, you can’t directly execute another stored procedure inside a Select statement, but you can create a view or write subqueries to accomplish the same task:

Where to Find the Wizard

Wondering where you can find the Access Upsizing Wizard? Here’s where to look:

- Free from Microsoft at www.microsoft.com/AccessDev/ProdInfo/AUT97dat.htm
- From the Access Developers Kit (ADK) available from Microsoft and from resellers
- For Access 95 and Access 2.0, contact Weir Performance Technologies at www.weirperf.com/upsiz4.htm

Access Data Type	SQL Server Data Type	Bytes
Yes/No, True/False, On/Off	Bit	1 (8 per byte)
Byte	Smallint	1
Integer	Smallint	2
Long Integer	Int	4
Single	Real	4
Double	Float	8
Text, Hyperlink	Varchar	Actual data size
Text, Hyperlink	Char	Defined field size
Memo	Text	2 + 2K minimum for each non-null value
ReplicationID	Varbinary	16
Date/Time	Datetime	8
Currency	Money	8
AutoNumber (Long Integer)	Int (Identity)	4
OLE Object	Image	2 + 2K minimum for each non-null value

Table 2 Map Access to SQL Data Types. This table shows how data types in Access map to SQL Server, as well as the physical size of each data type. Use this table to calculate row sizes and to translate parameter definitions in stored procedures.

```

CREATE PROCEDURE GetCustomerWithOrders
    @CustID integer
AS
SELECT * FROM
    -- subquery "A" is the equivalent
    of getCustomer
(SELECT * FROM Customers WHERE
    Customers.CustID = @CustID) A
INNER JOIN
-- subquery "B" is the equivalent of
getOrdersForCustomer
(SELECT * FROM Orders WHERE CustID =
    @CustID) B ON A.CustID = B.CustID

```

Optimize, Simplify, and Trap Those Errors

You've had some practice writing Transact-SQL by translating your queries, but SQL Server can go far beyond simple queries. Access queries can contain only a single SQL statement. SQL Server stored procedures can contain multiple statements. You can often optimize and simplify your application code by combining multiple queries into a single stored procedure. Stored procedures can also return values. Rather than opening a recordset to retrieve a single value—for example, an order number—given a customer and a date, you can use an output parameter to return the value:

```

CREATE PROCEDURE getOrderNumber
    @CustID int, @OrderDate datetime,
    @OrderNumber int output
AS
SELECT @OrderNumber = SELECT
    OrderNumber FROM Orders WHERE
    CustID = @CustID AND OrderDate =
    @OrderDate

```

Study the triggers the Upsizing Wizard creates. You can often eliminate some application code by moving the logic into triggers. Because the triggers fire automatically when rows are updated, inserted, or deleted, you can use triggers to maintain data integrity and eliminate “bad” data entirely within SQL Server itself.

Check the indexes on your tables. SQL Server supports “clustered indexes.” A clustered index stores data on disk in the same order as the index itself, which can speed up data retrieval for large tables, those where the indexed field is often queried with BETWEEN (for example, SELECT * FROM Orders WHERE OrderDate BETWEEN '1/1/98' AND '12/30/98'). By default, the Upsizing Wizard creates non-clustered in-

dexes. You can improve performance by creating a clustered index instead. Don't create a clustered index on an automatically incremented (AutoNumber in Access, Identity column in SQL Server) field because it dramatically slows down insert operations. In addition, I've found the Upsizing Wizard names indexes in a manner that's useful if you continue to use Access as a front end, but not particularly user-friendly if you're upsizing for the Web. It also sometimes creates duplicate indexes. You should delete the duplicate indexes if you find any.

The wizard creates triggers to enforce non-null values in fields marked “Required” in Access. It's more efficient to define the fields to accept or reject null values when the table is created. It is easiest to do this with the database tools in Visual Studio, but you can do it with DDL statements once you've seen the syntax. Fortunately, SQL Server makes it easy to see the SQL syntax behind any object. Using SQL Enterprise Manager, select any table or stored procedure, then click on the Object menu and select “Gen-

erate SQL Scripts.” Select the objects for which you want to generate SQL scripts. You can cut-and-paste the scripts into other programs for modifications.

Finally, take advantage of SQL Server's error-handling and logging features. You can use these to provide robust error messages to application programs that can speed up development and simplify finding and fixing bugs.

You've seen why and how to upsize your Access database to SQL Server. If you follow the guidelines in this column, your applications will become more scalable, more robust, and easier to maintain. Good luck! **VBPJ**

About the Author

A. Russell Jones, Ph.D., is vice president of InterCom Inc., creators of multimedia training and applications software located in The Woodlands, Texas. He's a former reptile keeper and professional musician who now composes computer applications. Reach him by e-mail at arjonesiii@flex.net or find him answering questions in the microsoft.public.vinterdev newsgroup.