

SQL Server Security for the Access Developer

Mary Chipman and Andy Baron

2000

There's a lot to learn when working with SQL Server security from an Access application. In this article, Mary Chipman and Andy Baron introduce some basic SQL Server security concepts and discuss the choices you have when implementing SQL Server security in your Access-SQL Server application.

ONE of the reasons you're moving to SQL Server might well be that you need data security you can count on. Access security has always been a hit-or-miss proposition—flaws and security holes have existed in every version of Access, and will no doubt continue to exist in future versions. This can make any Jet database vulnerable to any user who cares to research the subject on the Internet. Adequate security is hard in any application—it's very difficult to secure anything against all attackers. However, an Access/Jet database application is simply impossible to secure in any meaningful sense of the word. A determined hacker can simply walk away with your mdb and mdw files and poke at them until they yield their secrets. SQL Server is a tougher nut to crack—the physical files aren't vulnerable in this way because there's no need to grant users permissions on the network shares where the files reside. However, you must take steps to guarantee that your database security scheme isn't vulnerable in other ways because most so-called "security holes" are self-inflicted.

This article presents an overview of the core concepts behind SQL Server security and how you can implement them with an Access front end. Fortunately, SQL Server security is fully documented in SQL Server Books Online, and we encourage you to turn there for more detailed information.

Authentication and permissions validation

SQL Server security can be integrated with the security system in Windows 2000 or Windows NT 4. The advantages of using integrated Windows security are many—users don't have to log on twice, password ageing is supported, and logging of user activity is available. As an alternative, however, you can choose to use SQL Server for creating and storing all of your security accounts.

SQL Server 2000 operates under two distinct modes

of user authentication:

- **SQL Server and Windows (mixed)**—Both SQL Server and Windows NT/Windows 2000 logins are supported.
- **Windows Only**—Only Windows NT/Windows 2000 logins are allowed. In this mode, logins created and stored in SQL Server aren't supported.

When installing SQL Server 2000, you'll need to decide whether or not you want to allow SQL Server logins. Using integrated security with Windows authentication only is the most secure option. For smaller, less critical database applications, you might find SQL Server authentication easier to maintain and support. If you change your mind later, you can always change the authentication mode in the Server Properties dialog box in the SQL Server Enterprise Manager.

The first step a user goes through when attempting to use a SQL Server database is authentication, which happens at the server level prior to any database access. The second step is permissions validation, which happens at the database level. Users might be authenticated and still not be able to work with any databases if they aren't explicitly granted access to a database and then granted permission to interact with database objects. Unlike Access security, the default state of affairs in SQL Server is that users have no access to any databases, and no permissions once they're inside. Database access and permissions can be explicitly granted to users, or they can be granted to Windows groups or SQL Server roles to which the users belong.

Permissions

Assigning permissions directly to users can create an administrative nightmare. Users come and go, and adjusting their permissions for each object in each database can be a tedious task. Database roles provide a convenient way of managing permissions on database objects for multiple users or Windows groups. You can create as many roles as you need for each database. It even makes sense to create a role that contains only one user because, if that user leaves and is replaced, all you need to do is remove the old user from the role and add

the new user. That's much easier than reassigning all of the permissions individually. Once users are assigned to a role, they inherit all of the permissions granted to that role. If you assign a Windows group to a role, then all of the members of that group automatically inherit the permissions of the role.

SQL Server roles

Database roles aren't the only roles supported by SQL Server. Here's a rundown of the types of roles SQL Server supports:

- **Fixed server roles** allow users to perform server-level tasks. These roles can't be modified or deleted. Members of fixed server roles can perform all of the tasks defined for the role. At the apex of the fixed server role hierarchy is the sysadmin, or system administrator, fixed server role. A member of sysadmin has irrevocable administrative access to all aspects of SQL Server and all databases installed on a SQL Server instance. Examples of other fixed server roles are Disk Administrators (or diskadmin), members of which can manage the disk files, and Database Creators (or dbcreator), members of which can create and alter databases.
- **Fixed database roles** simplify database administration and grant or revoke basic permissions on all objects in a database. These roles can't be modified or deleted. Users added to these roles inherit the permissions granted to the roles. Examples of fixed database roles include db_datareader, db_backupoperator, and public. Public is a role that every database user is a member of. The public role is the only fixed database role you can configure—permissions granted to public are granted to all database users.
- **User-defined database roles** are ones that you create in each database. User-defined database roles can be made members of fixed database roles or other database roles. Users inherit the permissions granted to the roles.
- **Application roles** are used for an application or connection. You don't add users to an application role; the application connecting

to the database supplies a password, which activates the role. Once an application role is activated for a connection, any and all other permissions for that connection are then suspended for the duration of the connection.

Managing security with Enterprise Manager

The easiest way to manage security is to use the SQL Server Enterprise Manager. You can create and enable logins, assign fixed server roles, create and assign database roles, and set permissions from its graphical user interface. To create a SQL Server login or enable a Windows login, expand the Security node, right-click on Logins, select New Login, and fill in the information in the three tab pages.

Figure 1 shows the three tabs of the Login Properties dialog box: The General tab is shown enabling a Windows login on server MABEL for Windows user Andy. The default database is the Northwind database. The Server Roles tab displays the fixed server roles, which can also be assigned at this time. The Database Access tab lists the

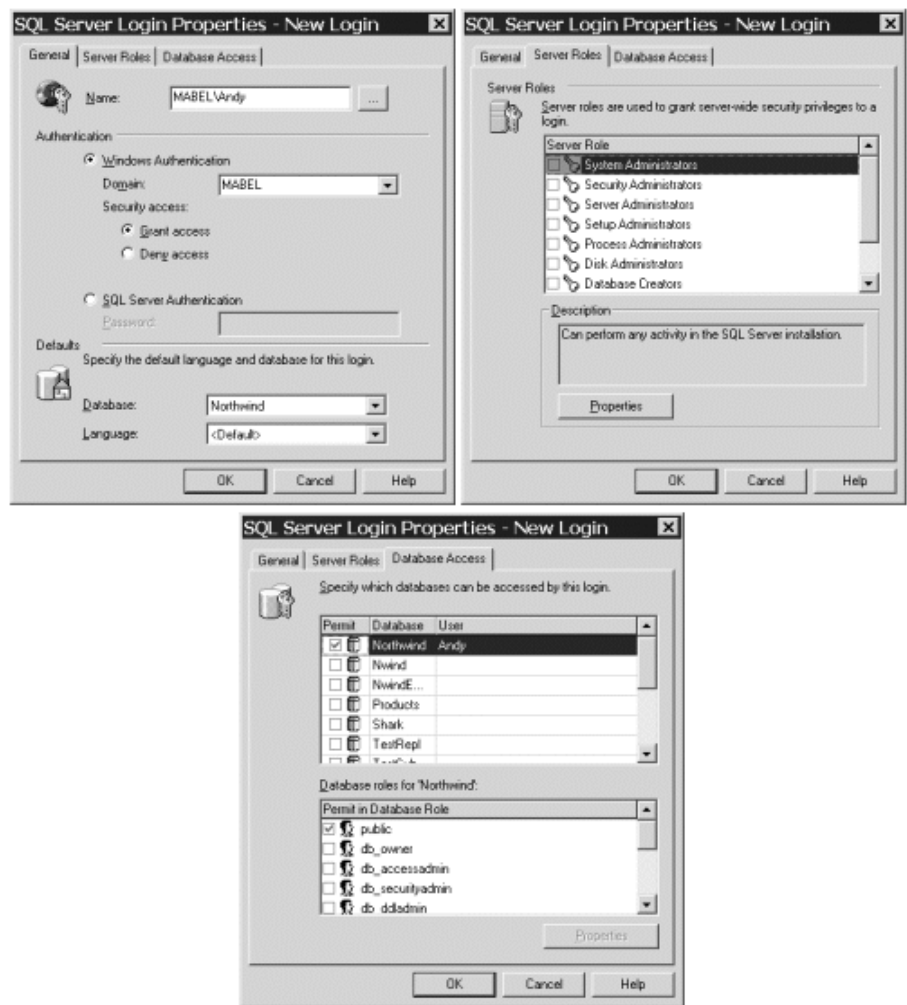


Figure 1. New Login—the General, Server Roles, and Database Access tabs.

databases installed on the MABEL SQL Server instance. You must grant database access to the database (Northwind) listed as the default database. Every database user is automatically added to the public role (which is analogous to Everyone in Windows security, or the Users group in Access security).

Regardless of how you allocate roles and permissions, there's one fixed server role that can override everything. If you add a user to the System Administrator fixed server role, that user can then perform any action in any database on your SQL Server.

The System Administrator, or sysadmin, fixed server role also has a SQL Server login—the sa login. If you enable SQL Server logins on your server, you should always set a password for this account. The first thing the bad guys will do when trying to break into your “secure” SQL Server is rattle the lock on the back door by attempting to log on as sa with no password. This is the SQL Server equivalent of leaving the Administrator account in Windows with a blank password. Once someone is logged on as a system administrator, he or she has complete control over everything. It's very easy to install SQL Server 7.0 without thinking about security and without noticing that sa has no password. SQL Server 2000 corrects this by adding a dialog box to the setup that encourages you to create a password for the sa user, if you've chosen to allow SQL Server logins.

There's no way to shield sensitive objects or data from a system administrator, so be very careful whom you admit as a member of the sysadmin role. Also bear in mind that any system administrator has access to the Windows administrative account that the SQL Server services use. This is the same as granting them administrative privileges on your Windows NT/Windows 2000 server since SQL Server system administrators have access to the xp_cmdshell system stored procedure, which executes commands at the operating system level.

In addition to the system administrator, object owners also have irrevocable permissions on the database objects that they own.

Object ownership

Whoever creates an object in SQL Server owns that object. This is particularly important to understand because SQL Server will allow duplicate object names as long as those objects have different owners. When referring to objects, you specify the owner name in front of the object name. For example, both Andy and Mary can have their own Customers table in the same database, and users would need to be granted permissions to use those tables by their owner, by the database owner, or by the system administrator. Users would refer to the tables in a SELECT statement as follows:

```
SELECT * FROM Andy.Customers
```

```
SELECT * FROM Mary.Customers
```

The dbo user

For simplicity's sake, most database administrators elect to have all objects owned by a special user that exists in all databases, called dbo. Any members of the sysadmin fixed server role are automatically mapped to the dbo user when they create objects in a database, and all of those objects have dbo as their owner. Don't confuse dbo with the fixed database role db_owner. Bob could be a member of the db_owner role, but any objects Bob creates will have Bob, not dbo, as their owner, unless Bob is also a system administrator.

Life is a lot less complicated if everything is owned by dbo. You can transfer ownership of non-dbo-owned objects by running the sp_changeobjectowner or the sp_changedbowner system stored procedures. If individual database users own objects in a database, those users can't be removed from the database and can't have their logins deleted. And to refer to any of those objects in a query, you'd need to preface the object name with the name of the appropriate object owner.

The guest account

In any database, you can create a special account named guest that isn't mapped to any particular login. The guest account is used to enable anonymous access to a database, and it's useful for Internet connections or other anonymous connections where you don't need or want to track individual user activity. By default the guest account isn't present; you need to make the decision that you want to allow anonymous database access and manually create it. When you create the guest account, don't associate it with any specific login. The guest account inherits permissions from the public database role, to which all users belong, so make sure that you don't grant public any permissions you don't want guests to have.

Guest users will still need to be authenticated by the server. For Internet users, you do this by creating a server login for the Iusr_ServerName account. For Windows users, you can do this by creating a login for a Windows group to which all of your guest users belong.

Application roles

Application roles are designed to allow an application to log on independently of the users who are running the application. An application (Access, Word, Excel, a VB ActiveX DLL, or whatever) activates the application role by running the sp_setapprole system stored procedure, supplying the name of the application role and its password:

```
EXEC sp_setapprole 'AppRoleName', 'AppRolePassword'
```

Once activated, users of the application work with database objects using the permissions granted to the application role. Any permissions a user might possess independently of the application role are suspended, and only the application role's permissions are evaluated. For example, if Andy is a system administrator and connects using an application role that has read-only privileges, then Andy loses his system administrative powers for the duration of that connection. The key point about application roles is that they're connection-specific and persist until the connection that activated them terminates. Another key point for the Access developer is that they're difficult to implement whenever an application uses connection pooling, which both ODBC and OLE DB do by default. You can disable OLE DB connection pooling, but not ODBC pooling in the case of a linked table application. In an Access-SQL Server linked table application, ODBC opens multiple connections behind the scenes that you have no control over. There's no way to ensure that an application role is activated for these secondary connections—the only way to use an application role in that scenario would be from within ADO code, where you could establish your own connection and run the `sp_setapprole` stored procedure.

Assigning permissions

As noted earlier, the default state of affairs in SQL Server is for users not to have any permissions with respect to databases until you grant them. Because most databases have many users and many objects, it makes sense to grant permissions only to roles and to let users inherit those permissions by being members of roles. There are three Transact-SQL statements that cover the types of permissions you can assign:

- GRANT grants a permission.
- REVOKE revokes, or removes, a permission. This is the default state of affairs for all users and roles when you create a new database. A revoked permission for a user can still be inherited through membership in a role that's been granted that permission. In other words, if you revoke Mary's permission to read data in the Customers table, but you also assign Mary to a role that's been granted permission to read customer data, then Mary will be able to read data in the Customers table.
- DENY goes beyond revoke and prevents inheriting that

permission from another role. DENY takes precedence over all other permissions, except permissions inherited from membership in the sysadmin role or from being an object owner. So, if you deny Mary permission to read customer data, she won't be able to read the data even if she's a member of a role that's been granted that permission—unless she's also a member of sysadmin or is herself the owner of the table.

The most common strategy when data security is important is to revoke or deny all permissions to all users and roles for tables. This means that users must go through views, user-defined functions, or stored procedures that you create for data access. For example, if you deny all permissions on the Customers table, and you grant EXECUTE permissions on a set of stored procedures that reference the Customers table, a user will be forced to work with customer data through the stored procedures, where you can limit the available options to ones you consider safe. To make this strategy work, however, you must ensure an unbroken chain of ownership from your exposed objects—the stored procedures, views, and functions—to the underlying base tables. As long as all of the objects have the same owner, permissions on the underlying tables aren't checked, and the user can work with the data using the stored procedures, views, and user-defined functions.

Figure 2 shows the Permissions dialog box for the public role. Permissions have been denied on the tables,

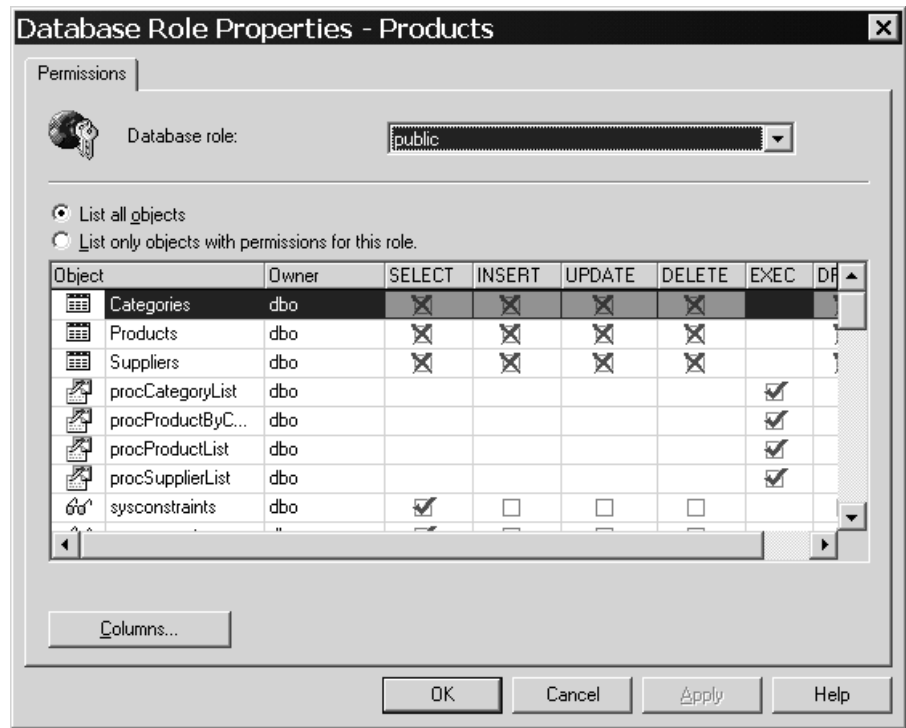


Figure 2. Permissions dialog box for the public role.

but granted on views and stored procedures. This means that database users will be unable to directly work with the tables in any way. They'll be forced to go through stored procedures, views, or user-defined functions where you have complete control over their activities. The obvious benefit is for action queries that modify data, but there's also the advantage of preventing expensive SELECT queries as well.

Working with SQL Server linked tables and views

The simplest and easiest way to work with Access and SQL Server is to simply link tables or views to an Access database (mdb). For security reasons, linking to views is a better choice if you want to update data. Working with linked views is the same as linking to tables, but it allows you the flexibility of partitioning data or aliasing column names. Views can include multiple joined tables, but only one of the tables can be updated (unless you use an instead-of trigger). When you want to update data using a view, you'll need to have previously specified which primary key or unique index to use when it updates data. A linked table or view without a primary key or unique index is always read-only in Access.

The links to SQL Server tables (or views) can be created easily in VBA. You'll want to use DAO for linked SQL Server tables because there's a bug with ADO/ADOX where your tables will link, but they'll be read-only. The following code fragment links to a SQL Server table using an ODBC DSN specified in the `strDSNname` variable using Windows authentication (or integrated security):

```
Dim db As DAO.Database
Dim tdf As DAO.TableDef
Set db = CurrentDb
Set tdf = db.CreateTableDef(strLinkName)
tdf.Connect = _
    "ODBC;Database=" & strDBName _
    & ";DSN=" & strDSNname _
    & ";Integrated Security=True"
tdf.SourceTableName = strTableName
db.TableDefs.Append tdf
```

If you programmatically create a link to a view, you can specify the unique index that Access should use to make the view updatable by executing a CREATE INDEX statement on the linked TableDef object:

```
db.Execute "CREATE INDEX " & strIndexName _
    & " ON " & tdf.Name & " (" _
    & strIndexFields & ") WITH PRIMARY"
```

A DSN is nothing more than a Registry entry or file that contains information about how to connect, and it's possible to link to SQL Server tables without a DSN by supplying the necessary connection information yourself. The following connection string supplies a SQL Server user ID and password, to connect to the Northwind

Why Use Stored Procedures?

Building a stored procedure-driven application is not only good for security—it also provides a level of abstraction away from the database schema, separating interface from implementation. You can make changes to your database schema and to the internals of your stored procedures, even adding new optional parameters, without necessarily breaking existing client code. An additional benefit is that you can encrypt the definitions of views, stored procedures and user-defined functions by simply including the WITH ENCRYPTION option when defining them. The performance benefits of stored procedures are well known—they can take advantage of a pre-compiled execution plan that is cached on the server and re-used for subsequent calls to the same stored procedure from multiple clients.

database on a local SQL Server:

```
strConnectionString = "ODBC;Driver={SQL " _
    & "Server};Server=(local);Database=Northwind" _
    & ";UID=Bert;PWD=floom"
```

If the data you're fetching from SQL Server is destined for reports or for filling list or combo boxes, then using stored procedures or user-defined functions as the data source is your best choice. You simply execute the stored procedure using a read-only pass-through query that returns records. You then base your form, report, or combo box on the pass-through query in the same way you'd use a regular query. Pass-through queries are also easy to work with using DAO. The following code snippet modifies a saved pass-through query named `qryCustomerGet`, which executes a stored procedure named `procCustGet`, passing it an integer parameter value in `intID` so that only one customer record is retrieved:

```
Dim db As DAO.Database
Dim qdf As DAO.QueryDef
Set qdf = db.QueryDefs("qryCustomerGet")
qdf.Connect = strConnectionString
qdf.SQL = "EXEC procCustGet " & intID
```

One thing to bear in mind about pass-through queries is that they're not parsed by Jet, and they don't handle parameters in the same way that regular Access queries do. All SQL statements are passed directly to the SQL Server for execution, and parameter values must be explicitly passed in the string. No VBA functions or form references are allowed because SQL Server has no way to

process them.

Administering security

There are no graphical tools available in an Access database (mdb) for administering SQL Server security. Your best bet is to use Enterprise Manager to set up and configure security on your SQL Server. However, be aware that Enterprise Manager is simply a shell for executing Transact-SQL commands and system stored procedures. You can very easily wrap those same stored procedures in pass-through queries, which can be called from your form code, in order to handle day-to-day security administration tasks. For example, the `sp_password` system stored procedure will change a user's SQL Server password. A system administrator can pass null for the old password parameter (handy for resetting those forgotten passwords), or users can change their own passwords by supplying the old password value. Either way, the stored procedure can be executed from a pass-through query. All you need to do is provide the form and call the necessary code to modify the pass-through query's SQL property, as shown in the previous example. Here's the Transact-SQL syntax for resetting the password Rocky forgot to "password":

```
EXEC sp_password
  @old = NULL,
  @new = 'password',
  @loginame = 'Rocky'
```

A pass-through query isn't your only option. If you'd rather run everything from VBA code, then the ADO Connection, Command, and Recordset objects are all capable of executing stored procedures, or you can use ODBCDirect workspaces in DAO. The Transact-SQL statements and system stored procedures needed to administer security are all fully documented in SQL Server Books Online.

Application and code security in an Access database

The Jet database engine that Access uses handles permissions assigned in each database, and Jet is still present in an Access 2000 mdb. If you're so inclined, you can use Access security for forms, reports, and macro objects, although most find that it's difficult and tricky to get it right. Users will need two logins to work with the secured application: one for Access and one for SQL Server (or Windows if you're using integrated security). In addition, even correctly implemented Access security has proven vulnerable to attack. An Internet search will find many services offering to crack it for you. In Access 2000, modules are no longer included under the Jet umbrella—you set a separate password on your VBA project instead.

A less cumbersome solution is to save your Access front end as an MDE file. This strips out all of your source

code and prevents design changes from being made to any Access objects (forms, reports, and modules). It has no effect on data (tables and queries) or macros. The disadvantage is that users will be unable to create their own database objects. Don't lose your original mdb file, because you can't make any design changes to the MDE. Any design changes must be made in the original MDB database, and a new MDE must be compiled to distribute to your users.

Security with an Access project

In Access projects (ADPs), you can use stored procedures that return records as the record sources of your forms and reports. Normally, row-returning stored procedures aren't updateable, but Access enables you to add, update, and delete data on a form that's bound to a stored procedure, as long as the user doing the updating has ADD, UPDATE, and DELETE permissions on the underlying tables. The reason this works is that Access creates its own action query behind the scenes that attempts to make the data modification directly to the tables referenced in the stored procedure. Views are also updateable as long as the user has permissions on the single table being updated using the view. This means that the technique described earlier of removing all permissions from tables and using only views or stored procedures won't work for bound Access forms.

However, there's a workaround if you're using SQL Server 2000. You can create a view with the WITH VIEW_METADATA option, which tells SQL Server to return metadata information about the view to the client application, rather than metadata about the base tables. When you create a view with this option, you can use the view to update data even if you've denied all permissions to tables. Of course, you still need to grant the requisite UPDATE, INSERT, or DELETE permissions on the view itself. Since Access has the metadata information, it can then use the view to update the data, and it won't try to perform the update directly on the underlying tables. Here's what the Transact-SQL syntax looks like to create an Access-updateable view:

```
CREATE VIEW vwShippers
  WITH VIEW_METADATA
AS
  SELECT ShipperID, CompanyName, Phone
  FROM Shippers
```

Access users will be able to update data using the view even if they don't have any permissions on the Shippers table. If you left out the WITH VIEW_METADATA clause, the view would be read-only in Access. If you're using SQL Server 7.0, the WITH VIEW_METADATA option isn't available, and you're forced to grant users permissions on the underlying tables unless you move to creating an unbound application.

Application roles and Access projects

One insurmountable obstacle with using application roles from an Access database is that an Access database uses ODBC, which opens multiple connections that you have no control over. Since an application role requires activation of the role by running the `sp_setapprole` system stored procedure and supplying the role name and password, you can't ensure that the application role will always be active.

An Access project is a different story. It ostensibly uses a single OLE DB SQL Server connection. However, an ADP will also take advantage of OLE DB connection pooling, which means that you could end up using multiple different connections over the lifetime of one ADP instance. Therefore, if you want to use an application role, you need to turn off OLE DB connection pooling by setting the following option in the connection string used to connect to SQL Server:

```
OLE DB Services = -2
```

The following code fragment connects to a SQL Server database named `Nwind` and activates the `NwindApp` application role:

```
strConn = _
  "PROVIDER=SQLOLEDB.1;" _
  & "OLE DB Services = -2;" _
  & "INTEGRATED SECURITY=SSPI;" _
  & "PERSIST SECURITY INFO=FALSE;" _
  & "INITIAL CATALOG=Nwind;" _
  & "DATA SOURCE=(local)"

CurrentProject.OpenConnection strConn
CurrentProject.Connection.Execute _
  "EXEC sp_setapprole 'NwindApp', 'password'"
```

Administering security using an ADP

Access 2000 provides a graphical user interface for working with SQL Server security. However, when SQL Server 2000 was first released, Access 2000 didn't work well with SQL Server 2000. If you wanted to administer SQL Server from an Access project, you had to install the SQL Server client tools from the SQL Server setup CD to get the right bits on your machine. Microsoft has remedied this situation by releasing the Access 2000 and SQL Server 2000 Readiness Update, which you can download for free from Microsoft's Web site. Before you apply the Access 2000 and SQL Server 2000 Readiness Update, you must first apply the Office 2000 SR-1/SR-1a update.

Once you've installed the

Readiness Update, you can administer SQL Server security from Access as long as you have the necessary permissions to do so. From the Tools menu, choose Security | Database Security. This will load the SQL Server Security dialog box. There are three tabs, which allow you to drill down into every facet of security for the SQL Server database you're connected to:

- The Server Logins tab allows you to work with SQL Server logins. Use the Add button to create a new SQL Server login or to enable a Windows login. Use the Edit button to bring up the same dialog box you'd get in the Enterprise Manager for setting the default database, server roles, database access, and database roles. The Delete button removes the login. When you click the OK button here, changes are automatically saved back to SQL Server.
- The Database Users tab allows you to work with database users. Use the Add button to add a selected user to a database role. The Edit button is used either to add a user to a role or to assign permissions directly to the user by clicking the Permissions button. The Delete button removes the database user.
- The Database Roles tab allows you to work with database roles. Use the Add button to add a database role. The Edit button is used either to add users to the selected role or to assign permissions to the role.

Figure 3 shows the dialog boxes for working with the public role. You can view permissions assigned to public by clicking the Permissions button.

Application and code security in an Access project

The Jet database engine isn't present in an Access

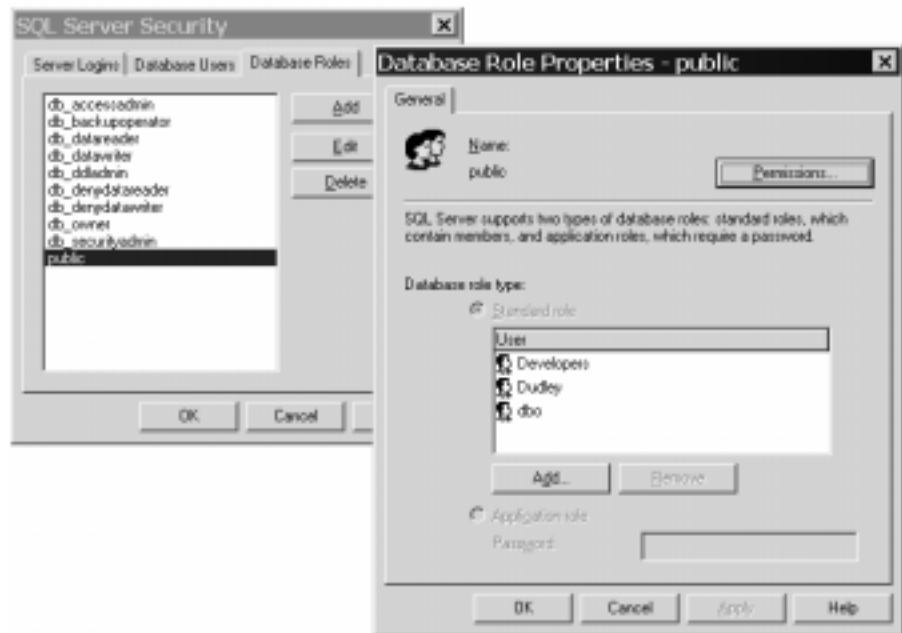


Figure 3. SQL Server security in an Access 2000 ADP file.

project (adp); neither is Jet's user-based security. All you have is SQL Server security on your database objects. Therefore, any kind of multi-level security for your forms, reports, data access pages, macros, and modules is simply not possible. However, you can compile your Access project as an ADE, which does the same thing that creating an MDE does for an Access database. All of your code, forms, and reports will be stripped of source code, and you won't be able to make any changes or add new objects. This will have no effect on your SQL Server database, or on the ADP connection to the SQL Server database.

Working with SQL Server object models to administer security

Whether you're using an Access database or an Access project, there are two additional ways to work with SQL Server security:

- *SQL Distributed Management Objects (SQL-DMO)* is a COM wrapper around an API that allows applications to administer all parts of a SQL Server installation, including security. This API then invokes the Transact-SQL commands and system stored procedures to do all of the actual work. Enterprise Manager itself uses SQL-DMO behind the scenes to do its job.
- *SQL Namespace (SQL-NS)* is also a set of COM interfaces. However, SQL-NS is different from SQL-DMO in that it allows your application to invoke wizards, dialog boxes, and other graphical elements of the Enterprise Manager user interface.

Both SQL-DMO and SQL-NS can be used from Access. However, SQL-NS requires that you have the SQL Server client tools installed on the computer that's using it. You need to set a reference to both libraries, as shown in [Figure 4](#).

To use SQL-DMO to change a user's password, create a SQL-DMO SQL Server object, connect to the server, and then create a Login object that points to the user's password you want to change. In the code fragment listed here, both the `SQLServer2` and `Login2` objects refer to a SQL Server 2000 instance—if you were using SQL Server 7.0, you'd leave off the "2":

```
Dim oSQLServer As SQLDMO.SQLServer2
Dim oLogin As SQLDMO.Login2

Set oSQLServer = New SQLDMO.SQLServer
With oSQLServer
    .LoginSecure = True
    .Connect "(local)"
End With

Set oLogin = oSQLServer.Logins("Rocky")
oLogin.SetPassword _
    OldPassword:=vbNullString, _
    NewPassword:="NewPassword"

oSQLServer.Disconnect
Set oLogin = Nothing
Set oSQLServer = Nothing
```

Using SQL-NS to change Rocky's password requires quite a bit more code. The SQL-NS object model is a reflection of the hierarchy of folders and nodes displayed in the Enterprise Manager. You have to start at the top, capturing references to the SQL Server, the Security folder, the Logins folder, and, finally, the Rocky login object. Once you've got the Login object, you can then execute a command on it. In this case, you want to bring up the Login Properties dialog box. Executing the command also requires that you pass it the `Hwnd` property of the form in order to display the dialog box. The Enterprise Manager's Login Properties dialog box is then displayed modally. When you dismiss it, the cleanup code at the end of the procedure runs.

```
Public Sub TestSQLNS()
    Dim oSQLNS As SQLNS.SQLNamespace
```

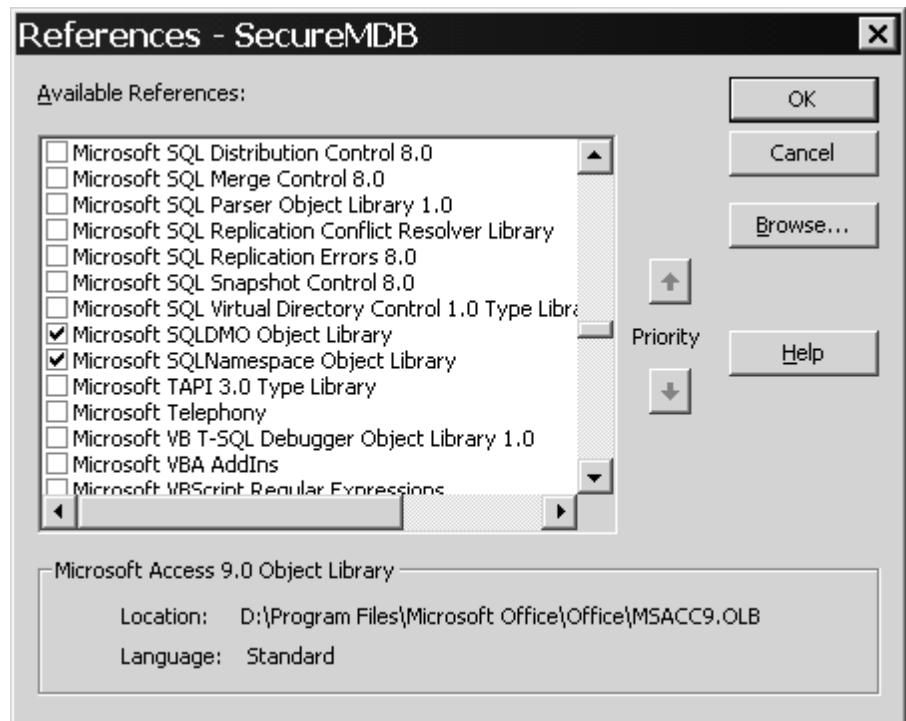


Figure 4. Setting references to the SQL-DMO and SQL-NS type libraries.

```

Dim oSQLNSObj As SQLNS.SQLNamespaceObject
Dim lngRoot As Long
Dim lngSec As Long
Dim lngLogins As Long
Dim lngLogin As Long

Set oSQLNS = New SQLNS.SQLNamespace
oSQLNS.Initialize _
    bstrAppname:="Test", _
    RootType:=SQLNSRootType_Server, _
    pRootInfo:"Server=MABEL;" _
    & "Trusted_Connection=Yes;", _
    Hwnd:=Me.Hwnd

' Get the root item (the SQL Server)
lngRoot = oSQLNS.GetRootItem

' Get the Security folder
lngSec = oSQLNS.GetFirstChildItem( _
    ItemIn:=lngRoot, _
    matchType:=SQLNSOBJECTTYPE_SECURITY)

' Get the Logins folder
lngLogins = oSQLNS.GetFirstChildItem( _
    ItemIn:=lngSec, _
    matchType:=SQLNSOBJECTTYPE_LOGINS)

' Get the Rocky login
lngLogin = oSQLNS.GetFirstChildItem( _
    ItemIn:=lngLogins, _
    matchType:=SQLNSOBJECTTYPE_LOGIN, _
    matchName:="Rocky")

' Set the SQLNamespaceObject object to the login
Set oSQLNSObj = oSQLNS.GetSQLNamespaceObject(lngLogin)

' Execute the command by its CommandID
oSQLNSObj.ExecuteCommandByID _
    CommandID:=SQLNS_CmdID_PROPERTIES, _
    Hwnd:=Me.Hwnd, _
    modality:=SQLNamespace_PreferModal

' Cleanup
Set oSQLNSObj = Nothing
Set oSQLNS = Nothing
End Sub

```

These two examples of SQL-DMO and SQL-NS merely scratch the surface of what's available to you. Both are documented in SQL Server Books Online, and you can obtain help through the VBA Object Browser once you've set a reference to their type libraries (there's no VBA Help available for SQL-NS in SQL Server 7.0). To examine additional examples of using SQL-DMO and SQL-NS, see the Visual Basic sample applications that ship on the SQL Server setup CD. The VB code requires little adaptation to run in Access, although it's helpful to run the sample projects in Visual Basic first so you can see what the examples are supposed to be doing.

If at this point you're asking yourself why you need either SQL-DMO or SQL-NS when you can simply execute Transact-SQL commands or system stored procedures from Access directly, the answer is, you don't. There's nothing to prevent you from creating your own forms for managing SQL Server security and executing the appropriate stored procedures yourself. However, many people prefer the convenience of programming in an object-oriented environment (SQL-DMO), and it's certainly easier and visually more consistent to call an

element of the Enterprise Manager user interface (SQL-NS) than it is to attempt to reproduce the look-and-feel of the Enterprise Manager user interface in your own forms.

In this article, you've seen some of the options available to you for implementing SQL Server security. Fortunately, SQL Server security itself is amply documented in SQL Server Books Online, which should always be your first stop when building a SQL Server application. A further resource is available in the Microsoft-sponsored SQL Server newsgroups at [news:msnews.microsoft.com](http://news.msnews.microsoft.com) and the SQL Server security white paper, which you can download from <http://www.microsoft.com/SQL/techinfo/2000securitywp.htm>. ▲

Mary Chipman (mChip@mcwtech.com) and Andy Baron (Andy_Baron@msn.com) are co-authors of the *Microsoft Access Developer's Guide to SQL Server* (Sams), from which parts of this article are excerpted. Mary and Andy are both senior consultants with MCW Technologies, a Microsoft Solution Provider. They're also Microsoft MVPs and trainers with Application Developers Training Company (<http://www.AppDev.com>) They co-authored AppDev's SQL Server 2000 courseware and videos. In addition, Mary is co-author of *SQL Server 7.0 in Record Time* (Sybex).