Advanced Microsoft Jet SQL for Access 2000

Acey James Bunch
Microsoft Corporation

February 2000

Applies To: Microsoft® Access 2000

Summary: Third in a series of three articles, Advanced SQL builds on the concepts covered in the Fundamental and Intermediate articles, this time focusing on SQL syntax that is most often found in a multi-user environment. If you've read the previous two articles in an effort to become a well-rounded SQL user, this article will provide the final tools you will need to accomplish your goal. (24 printed pages)

Download AcAdvSQL.exe

Contents

Introduction
Advanced SQL Defined
  The Multiuser Model
  Jet vs. MSDE
  SQL Coding Conventions
Advanced Data Definition Language
  Views
  Procedures
  Managing Security
  Share-Level Security
  User-Level Security
Advanced Data Manipulation Language
  Transactions
  With OwnerAccess Statement
Using Advanced SQL in Access
  Sample Database
  Setting the Database Password
  Using a Procedure
One Last Comment
Additional Resources

Introduction

This is the third in a series of articles that explains what Microsoft® Jet SQL is and how you can use it in your Access 2000 applications. There are three articles in all: a fundamental, an intermediate, and an advanced article. The articles are designed to progressively show the syntax and methods for using Jet SQL, and to demonstrate those features of Jet SQL that are new to Access 2000. For the remainder of this article, all references to Structured Query Language (SQL) are meant to be to SQL statements as they are used in the Microsoft Jet database engine.

Advanced SQL Defined

As in the previous two articles, Jet SQL features fall into two different categories: Data Definition Language (DDL) and Data Manipulation Language (DML). In this article, we look at the advanced SQL statements that you can use to manage database objects and manipulate the data that they contain.

For advanced DDL, we examine how to use views, procedures, and database security to add more power and control over a database system. We look at much more generic ways to get at the data contained in the tables; and with regard to managing security, we look at ways to establish permissions and access rights to the objects that the database contains.

For advanced DML, we examine syntax that can be used to manage the SQL statements as they are executed, and syntax that will allow special permissions on certain queries.

The advanced DDL statements in this article are presented in the context of a multiuser application, but that is not to say that they can't be used in a single-user application. You could have an application that is deployed on a single workstation, but that is used by multiple people and therefore needs multiuser-style security. When we think of multiuser applications in Access, the definition that usually comes to mind is a database application that is designed to accommodate one or more users at any one time. This means that more than one user can be using the same application, and accessing the same data at the same time. There are a variety of SQL statements that are useful when developing multiuser Access applications, and many of the advanced SQL statements are new to the Jet 4.0 database engine that is included with Access 2000.

It is very important to note that some of the advanced SQL syntax is currently available in code only when using ActiveX® Data Objects (ADO) and the Jet OLE DB provider, and is not currently available through the Access SQL View user interface or through Data Access Objects (DAO). This article points out when a certain SQL statement is available only through the Jet OLE DB provider and ADO.

The Multiuser Model

Before you use some of the advanced SQL statements, it is helpful to understand exactly what a multiuser database application is and, equally important, what it is not. When you use Access and the Jet database engine to create an application, you are using what is known as a file-server solution. A typical scenario is to split the Access application into two separate .mdb files. One file resides on a networked file server and it contains only the data tables for the application. The other .mdb file resides on users' workstations and it contains all of the other database objects such as forms, macros, modules, and reports. When users request some type of database action, all of the processing is handled locally and the back-end tables are simply linked to the .mdb file on each user's local workstation.

This file-server-based model should not be confused with the client/server-based model. In a client/server-based model, most of the data processing occurs on the database server, not on the local workstation. In addition, client/server database systems like Microsoft SQL ServerÂ™ provide extra power and data integrity, such as transaction recovery in the case of system failure.

That said, there are many powerful features in Jet SQL that will allow you to administer and maintain multiuser database solutions within the file-server database model.

Jet vs. MSDE

New to Access 2000 is Microsoft Data Engine (MSDE). MSDE is a true client/server database engine that is compatible with the SQL Server database system. MSDE is included with Office 2000 Professional and Premium Editions, and is also available with the Access 2000 stand-alone edition. Although this paper is focusing on the SQL syntax as it is implemented in the Jet 4.0 database engine, thanks to the stronger ANSI-92 compliance of Jet 4.0, many of the SQL statements are equally compatible and scalable to the SQL syntax implemented by the MSDE, and also by Microsoft SQL Server. If you are using MSDE or thinking about moving an existing application to it, be sure to check the differences between Jet and MSDE SQL syntax.

SQL Coding Conventions

This article uses a consistent method of SQL coding conventions. As with all coding conventions, the idea is to display the code in such a way as to make it easy to read and understand. This is accomplished by using a mix of white space, new lines, and uppercase keywords. In general, use uppercase for all SQL keywords, and if you must break the line of SQL code, try to do so with a major section of the SQL statement. You'll get a better feel for it after seeing a few examples.

Poorly formatted SQL code

```
CREATE TABLE tblCustomers (CustomerID INTEGER NOT NULL, [Last Name]
TEXT(50) NOT NULL, [First Name] TEXT(50) NOT NULL, Phone TEXT(10), Email
TEXT(50))
```

Well-formatted SQL code

```
CREATE TABLE tblCustomers (
    CustomerID INTEGER NOT NULL,
    [Last Name] TEXT(5) NOT NULL,
    [First Name] TEXT(50) NOT NULL,
    Phone TEXT(10),
    Email TEXT(50))
```

Advanced Data Definition Language

For advanced Data Definition Language (DDL), we examine some of the newest additions to the SQL syntax that is implemented in Jet 4.0. These include views, procedures, and the syntax that is used to manage security. This new syntax aligns the Jet SQL implementation more closely to the ANSI-92 specification, and it provides for easier scaling or upsizing to Microsoft SQL Server or Microsoft Data Engine (MSDE).

Views

An SQL *view* is a database object that allows you to organize and look at data from one or more tables, and can be referenced as if it were a single, virtual table. It is similar to a Select query in Access in that it is based on a SELECT statement, but it is different because it cannot have parameters. The thing to remember about a view that differentiates it from a table is that it does not *store* data, it only *returns* data and may allow you to *update* data. To create a view, use the CREATE VIEW statement to name the view, define its field list, and associate the view with a SELECT statement.

```
CREATE VIEW MyCustomersNames (FirstName, LastName)
   AS SELECT [First Name], [Last Name]
   FROM tblCustomers
```

Then you can use the view in other SQL statements as if it were a table.

```
SELECT *
   FROM MyCustomersNames
```

Note, however, that a view can be updated only if the SELECT statement used in it is capable of being updated, and the view name cannot be the same as an existing table name. To drop a view, use the DROP VIEW statement.

```
DROP VIEW MyCustomersNames
```

> **Note** The CREATE VIEW and DROP VIEW statements can be executed only through the Jet OLE DB provider and ADO. They will return an error message if used through the Access SQL View user interface or DAO. Also note that views created with the CREATE VIEW statement are saved in the database, but are not exposed as saved queries in the Access user interface. You can work with them only in ADO and ADOX programming code.

For more information about views, type **create view statement** in the Office Assistant or on the **Answer Wizard** tab in the Microsoft Access Help window, and then click **Search**.

Procedures

A procedure is a database object that you can use to execute an SQL statement that is based on values that are passed to it. These values are often called *parameters*. You can think of a procedure as an SQL-based function. The procedure allows you to pass in parameters that are then used by the SQL statement, usually as part of a WHERE clause. The benefit is that you can write very generic procedures that can be used in a variety of ways, and they can be called or executed from many different places in your programming code.

In previous versions of Access, you could use either a procedure clause or a parameter query to create SQL statements that use parameters. Although both types of queries are still supported, another syntax you can use is the new CREATE PROCEDURE statement (its synonym is CREATE PROC). The new syntax is more ANSI-92 compliant, and when scaling up to a Microsoft SQL Server database or MSDE, the syntax can be converted more easily.

The basic structure of the procedure is this:

```
CREATE PROCEDURE ProcedureName
```

```
    (Parameter1 datatype, Parameter2 datatype) AS
    SQLStatement
```

Note that the name of the procedure must not be the same as the name of an existing table, the parameters must separated by commas in the parameter list, and the entire parameter list should be enclosed in parentheses. You can have up to 255 parameters in the parameter list, and any of the valid SQL data types can be used. (See the previous article, "Intermediate Microsoft Jet SQL for Access 2000," for a discussion of SQL data types.) The SQL statement itself can be any valid SQL statement that is recognized by the Jet database engine. In addition, you can also use the synonym CREATE PROC to create a new procedure.

Let's use the CREATE PROCEDURE statement with a DATETIME parameter, and follow it with the DELETE statement that will delete all invoices older than 1/1/1999.

```
CREATE PROCEDURE DeleteInvoices
   (InvoiceDate DATETIME) AS
   DELETE FROM tblInvoices
   WHERE tblInvoices.InvoiceDate < InvoiceDate
```

Note that because the parameter name is the same as the field name, the field name in the WHERE clause is qualified with the table name. To drop a procedure, use the DROP PROCEDURE statement.

```
DROP PROCEDURE DeleteInvoices
```

> **Note** The CREATE PROCEDURE and DROP PROCEDURE statements can be executed only through the Jet OLE DB provider and ADO. They will return an error message if used through the Access SQL View user interface or DAO. Also note that procedures created with the CREATE PROCEDURE statement are saved in the database, but are not exposed as saved queries in the Access user interface. You can work with them only in ADO and ADOX programming code.

To run the procedure, use the EXECUTE statement (or the EXEC synonym) and pass the required parameters. If there is more than one parameter, you must separate them with commas.

```
EXECUTE DeleteInvoices '1/1/1999'
```

> **Note** The EXECUTE statement above can be executed only through the Jet OLE DB provider and ADO. It will return an error message if used through the Access SQL View user interface or DAO.

For more information about procedures, type **create procedure statement** in the Office Assistant or on the **Answer Wizard** tab in the Microsoft Access Help window, and then click **Search**.

Managing Security

As mentioned in a previous section, some of the Jet DDL statements are used to set and maintain the access rights and permissions to your database application (or in other words, manage security). Although the Access user interface gives you the ability to manage security for a database, in reality the Jet database engine does all the work.

The Jet database engine provides two modes of security: share-level and user-level. *Share-level security* involves establishing a password for a database, which is the simplest form of security. Once this kind of security is set, anytime the database is opened the user will be prompted for the password, and all users will use that same password.

A more complex, and also more flexible, form of security is *user-level or workgroup-based security*. This involves establishing groups and users and giving each one permission to perform specific tasks in a database. Using this form of security will enable each user to have his or her own password and set of permissions to various database objects and activities.

It should be noted that some reference books refer to SQL statements that are used to manage security as Data Control Language (DCL). But since DCL is not a term that is defined by the ANSI standard, we will not refer to any of the Jet SQL syntax by that name.

Share-Level Security

To set share-level security on a Jet database, use the ALTER DATABASE statement to create, modify, or remove the database password. To use the ALTER DATABASE statement, you must open the database in exclusive mode. The general form of the ALTER DATABASE statement is as follows:

```
ALTER DATABASE PASSWORD NewPassword OldPassword
```

To set the database password for the first time, use the ALTER DATABASE statement with the NULL keyword as the OldPassword parameter. Let's suppose that we want to set the database password to "admin."

```
ALTER DATABASE
   PASSWORD admin NULL
```

> **Note** The ALTER DATABASE statement can be executed only through the Jet OLE DB provider and ADO. It will return an error message if used through the Access SQL View user interface or DAO.

To change the existing database password, use the ALTER DATABASE statement with the appropriate passwords. Let's change the database password to "administrator."

```
ALTER DATABASE
   PASSWORD administrator admin
```

To remove a database password, use the ALTER DATABASE statement with the NULL keyword as the NewPassword parameter. Let's remove the "administrator" database password.

```
ALTER DATABASE
   PASSWORD NULL administrator
```

For more information about using the ALTER DATABASE statement, type **alter database** in the Office Assistant or on the **Answer Wizard** tab in the Microsoft Access Help window, and then click **Search**.

User-Level Security

User-level security is a much more robust way to manage the access rights and permissions of a Jet database, but it is also more complicated than using just a single database password. Setting up user-level security involves establishing user and group accounts, which consist of user names and passwords, and then setting the database object permissions for those accounts. The user and group account information is stored in the workgroup information file, which can be created by using the Access User-Level Security Wizard, and the database object permissions are stored in the system tables of the application's database. Although a thorough discussion of the workgroup information file is outside the scope of this article, you can find more information by typing **workgroup** in the Office Assistant or on the **Answer Wizard** tab in the Microsoft Access Help window, and then clicking **Search**.

User and group accounts

Once you have established your workgroup information file and are properly opening your database through it, there are a variety of SQL statements that you can use to manage security. With advanced SQL, you can create, modify, and remove user and group accounts. In addition, you can add users to group accounts. The SQL statements used to manage the user-level accounts are the CREATE, ADD, ALTER, and DROP statements.

Let's suppose that we have an Invoices database that has three tables: tblCustomers, tblShipping, and tblInvoices. We want to set up security on the database for two different departments, billing and shipping, and each department has two employees. The following series of advanced SQL statements demonstrates how to implement our security scheme.

> **Note** The USER and GROUP statements can be executed only through the Jet OLE DB provider and ADO. They will return an error message if used through the Access SQL View user interface or DAO.

First we need to create our two groups, Billing and Shipping. Use the CREATE GROUP statement to create a group.

```
CREATE GROUP Billing
```

To create more than one group at a time, separate the group names with a comma.

```
CREATE GROUP Billing, Shipping
```

To remove one group, use the DROP GROUP statement.

```
DROP GROUP Billing
```

To remove more than one group at a time, separate the group names with a comma.

```
DROP GROUP Billing, Shipping
```

Although you can create the group accounts with just a name, you should also include the optional argument known as a personal identifier*,* or PID. The PID is an extra string value that you can pass to the CREATE GROUP and CREATE USER statements. Jet will then combine the PID with the user or group name into a unique key value known as a security identifier*,* or SID. The SID is the value that Jet uses internally to identify and work with the corresponding user or group account. Specifying a PID when creating a user or group account ensures that the account is unique. Specifying a PID also allows you to re-create an identical account if the workgroup file becomes damaged, or if you need to move the account into another workgroup file. Once an account and its corresponding SID are created, you can never change or alter the PID value used to create it. You can, however, change the passwords for the user accounts.

The PID values you pass to the CREATE GROUP or CREATE USER statement can be 4 to 20 characters long, and they are case-sensitive. You should use PID values that will be difficult for someone else to guess; a combination of numbers and uppercase and lowercase characters is best. To really strengthen your security, you should also use different PIDs for each user and group. To create SID values for our two groups, let's give them their own PID unique values.

```
CREATE GROUP Billing Gu294JxP1m, Shipping Kl27c5sI9h
```

Now let's add our users. Tim and Sarah work in the Billing department, and Steve and Mary work in the Shipping department. Before we can add these users to their respective groups, we must first create their user accounts with the CREATE USER statement, passing the user name and password values to it. We will use "pwd" as the password for all users because they are allowed to change their own passwords in the Jet security model.

```
CREATE USER Tim pwd
```

To create more than one user account at a time, separate the user account information with commas.

```
CREATE USER Tim pwd, Sarah pwd, Steve pwd, Mary pwd
```

Here again, it is better to create the user accounts by using a PID. To use the PID with a CREATE USER statement, list the PID value after the password.

```
CREATE USER Tim pwd H3sJaZ9k2m
```

If you need to change a user's password, you can use the ALTER USER statement with the PASSWORD keyword. You must also supply the new and old password. In general, the form of the ALTER USER statement is as follows:

```
ALTER USER username PASSWORD NewPassword OldPassword
```

Let's suppose that management has asked you not to use "pwd" as the default password but to use the user names instead. Because you cannot change more than one user's password per statement, you will need to issue four separate ALTER USER statements.

```
ALTER USER Tim PASSWORD tim pwd
ALTER USER Sarah PASSWORD sarah pwd
ALTER USER Steve PASSWORD steve pwd
ALTER USER Mary PASSWORD mary pwd
```

To remove a user account, use the DROP USER statement.

```
DROP USER Tim
```

To remove more than one user account at a time, separate the user account names with commas.

```
DROP USER Tim, Sarah, Steve, Mary
```

Once the Group and User accounts have been created, you can use the ADD USER statement to add particular users to specific groups. Let's add our four users to the appropriate groups. Use the TO keyword with the name of the group in the ADD USER statement.

```
ADD USER Tim TO Billing
```

To add more than one user to a group at a time, separate the user account names with commas.

```
ADD USER Tim, Sarah TO Billing
ADD USER Steve, Mary TO Shipping
ADD USER Tim, Sarah, Steve, Mary TO Users
```

Note that you can only add users to one group at a time. Also note that you must add all users to the default group of Users; if you don't, they will not be able to log in through the Access user interface, because they will not have Read privileges to the Jet system tables that are automatically assigned to the default Users group. If you are adding users through the Access user interface, the users will be automatically included in the default Users group.

To remove a user from a group, use the DROP USER statement with the FROM keyword and specify the group name.

```
DROP USER Tim FROM Billing
```

Here again, note that you can remove a user (or users) from only one group at a time.

To remove more than one user from a group at a time, separate the user account names with commas.

```
DROP USER Tim, Sarah FROM Billing
DROP USER Steve, Mary FROM Shipping
DROP USER Tim, Sarah, Steve, Mary FROM Users
```

Note that removing a user from a group does not remove the user's account, it only removes that user from the group. But removing a user's account will automatically remove that user from any groups that he or she was associated with. To remove a user's account completely, use the DROP USER statement as discussed above.

For more information about user and group accounts, type **user accounts** in the Office Assistant or on the **Answer Wizard** tab in the Microsoft Access Help window, and then click **Search**.

Database object permissions

To fully implement the Jet security model, you must establish permissions on database objects for each user or group. There are two kinds of permissions that you can establish: *explicit* or *implicit* permissions. *Explicit permissions* are the privileges you assign or grant to a specific user, while *implicit permissions* are the privileges that you assign to a group and that are inherited by members of that group. *In general, user-level security is simpler to manage if you assign permissions only to groups, and then assign users to the appropriate group*.

In advanced SQL, permissions are managed with the GRANT and REVOKE SQL statements.

> **Note** The GRANT and REVOKE statements can be executed only through the Jet OLE DB provider and ADO. They will return an error message if used through the Access SQL View user interface or DAO.

The GRANT statement assigns or allows a user or group to perform some type of action with a specified database object, and its general forms are as follows:

```
GRANT privilege ON TABLE tablename TO grantee
Or
GRANT privilege ON OBJECT objectname TO grantee
Or
GRANT privilege ON CONTAINER containername TO grantee
Or
GRANT privilege ON DATABASE TO grantee
```

The REVOKE statement removes or disallows a user or group from performing some type of action, and its general forms are as follows:

```
REVOKE privilege ON TABLE tablename FROM grantee
Or
REVOKE privilege ON OBJECT objectname FROM grantee
Or
REVOKE privilege ON CONTAINER containername FROM grantee
Or
REVOKE privilege ON DATABASE TO grantee
```

The keywords used with the GRANT and REVOKE statements are as follows:

- The TABLE keyword is used when granting permissions on one or more tables in a database.
- The OBJECT keyword is used to specify any other non-table object within a database. Non-table objects include forms, queries, reports, macros, views, and procedures.
- The CONTAINER keyword is used to specify any of the container objects, such as tables, relationships, forms, or reports. Setting permissions on a container object allows you to specify permissions that will be inherited when a new object of that type is created. In earlier versions of Access, you can set user-level permissions on individual modules and the modules container. In Access 2000, you use VBA project password protection to set security for all modules.
- When using the DATABASE keyword, the current database is implied so it does not need to be specified.

The grantee can be a user or group account name, or you can use the PUBLIC or ADMIN keywords, with the following caveats.

- Using PUBLIC will set permissions for the default Users group account so that everyone will get the assigned privileges.
- Using ADMIN will set permissions for the default Admin single-user account. In general, you should disable all permissions for the Admin single-user account because it is the default account that is used to open all databases. That is, in an "out-of-the-box" installation of Access, all users are automatically logged on by the default workgroup information file as the Admin user, so any permissions that you grant to the Admin single-user account will be available to all Access users. If you want to establish an administrative account, you should do so explicitly by creating a separate account.

The privileges that you can grant or revoke are summarized in the following table.

Privilege	**Applies To**	**Description**
SELECT	Tables, Objects, Containers	Allows a user to read the data and read the design of a specified table, object, or container.
DELETE	Tables, Objects, Containers	Allows a user to delete data from a specified table, object, or container.
INSERT	Tables, Objects, Containers	Allows a user to insert data into a specified table, object, or container.
UPDATE	Tables, Objects, Containers	Allows a user to update data in a specified table, object, or container.
DROP	Tables, Objects, Containers	Allows a user to remove a specified table, object, or container.

SELECTSECURITY	Tables, Objects, Containers	Allows a user to view the permissions for a specified table, object, or container.
UPDATESECURITY	Tables, Objects, Containers	Allows a user to change the permissions for a specified table, object, or container.
UPDATEIDENTITY	Tables	Allows a user to change the values in auto-increment columns.
CREATE	Tables, Objects, Containers	Allows a user to create a new table, object, or container.
SELECTSCHEMA	Tables, Objects, Containers	Allows a user to view the design of a specified table, object, or container.
SCHEMA	Tables, Objects, Containers	Allows a user to modify the design of a specified table, object, or container.
UPDATEOWNER	Tables, Objects, Containers	Allows a user to change the owner of a specified table, object, or container.
ALL PRIVILEGES	All	Allows a user all permissions, including administrative, on a specified table, object, container, or database.
CREATEDB	Database	Allows a user to create a new database.
EXCLUSIVECONNECT	Database	Allows a user to open a database in exclusive mode.
CONNECT	Database	Allows a user to open a database.
ADMINDB	Database	Allows a user to administer a database.

Looking at our invoices database example, let's assign permissions to the Billing and Shipping groups. First we need to give everyone permission to open the database, so we will use the PUBLIC keyword in our GRANT statement. This will allow all members of the Users default group to open the database.

```
GRANT CONNECT
   ON DATABASE
   TO PUBLIC
```

For the invoices table, let's give read-only permissions to the Shipping group, and all data manipulation permissions to the Billing group.

```
GRANT SELECT
   ON TABLE tblInvoices
   TO Shipping

GRANT SELECT, INSERT, UPDATE, DELETE
   ON TABLE tblInvoices
   TO Billing
```

If you log on to the database as a member of the Shipping group, you will be able to open the Invoices table, but you will not be able to insert, update, or delete any records in it.

If you want to remove permissions, use the REVOKE statement.

```
REVOKE SELECT
   ON TABLE tblInvoices
   FROM Shipping

REVOKE SELECT, INSERT, UPDATE, DELETE
   ON TABLE tblInvoices
   FROM Billing
```

Note that instead of the TO keyword, the REVOKE statement uses the FROM keyword. For more information about the GRANT or REVOKE statements, type **GRANT statement** or **REVOKE statement** in the Office Assistant or on the **Answer Wizard** tab in the Microsoft Access Help window, and then click **Search**.

Advanced Data Manipulation Language

For advanced DML, there are only two SQL concepts to discuss. One is the use of transactions and the other is the use of the owner access option.

Transactions

A *transaction* is a logical grouping of work, or a collection of SQL statements, that must be completed successfully as a group or not at all. This logical grouping has a beginning, an ending, and a way to cancel the work, should there be a reason to. For example, let's suppose that you have a series of UPDATE statements that you want to execute, but you want either all or none of them to be applied. If an error occurs with one of them, you do not want any of them to be saved to the database.

In Jet SQL syntax, you would use the BEGIN TRANSACTION statement to start the transaction, and either the COMMIT TRANSACTION statement to save the data, or the ROLLBACK TRANSACTION statement to cancel any changes. Transactions can be nested five levels deep, and you should take care that all transactions are either committed or rolled back.

> **Note** The BEGIN, COMMIT, and ROLLBACK TRANSACTION statements above can be executed through the Jet OLE DB provider and ADO. They will return error messages if used though the Access SQL View user interface or DAO. In addition, you should not mix Jet SQL transactions with ADO transaction methods. If you do, you may get unpredictable results.

The following table lists each of the TRANSACTION statement keywords and their synonyms.

Keywords	**Synonyms**	**Description**
BEGIN TRANSACTION	BEGIN WORK	Begins the transaction.
COMMIT TRANSACTION	COMMIT, COMMIT WORK	Saves the transaction.
ROLLBACK TRANSACTION	ROLLBACK, ROLLBACK WORK	Cancels the transaction.

Let's suppose that you are cleaning up old data and want to delete all invoices dated before January 1 of a particular year, but you want to add a step that will allow users to cancel before actually deleting them. First, let's issue the BEGIN TRANSACTION statement.

```
BEGIN TRANSACTION
```

Now let's issue the DELETE statement.

```
DELETE FROM tblInvoices
   WHERE InvoiceDate < #1/1/1999#
```

If you open the Invoices table, you will notice that the records have not yet been deleted. To cancel the changes before they are committed, issue the ROLLBACK TRANSACTION statement.

```
ROLLBACK TRANSACTION
```

To save the changes to the database, issue the COMMIT TRANSACTION statement.

```
COMMIT TRANSACTION
```

For more information about transactions, type **transaction statement** in the Office Assistant or on the **Answer Wizard** tab in the Microsoft Access Help window, and then click **Search**.

With OwnerAccess Statement

The WITH OWNERACCESS OPTION clause is applied to queries, and it is most often used to present data to users who lack the sufficient permissions to normally access the data in the underlying tables. In other words, you can specify that a certain query is always executed with the permissions of the query owner, not the permissions of the user trying to open the query.

For example, let's say that the shipping department needs to know about customer invoices, but you do not want them to know the invoice amounts. Create a new query specifying all fields *except* the Amount field. Use the WITH OWNERACCESS OPTION clause at the end of your SELECT statement, and name the query Limited Invoices. Note that this statement can be executed in the Access SQL View user interface.

```
SELECT InvoiceID, CustomerID, InvoiceDate
   FROM tblInvoices
   WITH OWNERACCESS OPTION
```

Next, make sure that the shipping department has Read permissions on the Limited Invoices query.

```
GRANT SELECT
   ON OBJECT [Limited Invoices]
   TO Shipping
```

> **Note** The GRANT statement above can be executed only through the Jet OLE DB provider and ADO. It will return an error message if used through the Access SQL View user interface or DAO.

If you now log in as a member of the shipping group, you cannot open the Invoices table, but you can run the Limited Invoices query. In addition to permissions for SELECT statements, you can also use the WITH OWNERACCESS OPTION clause with INSERT INTO and SELECT INTO statements.

For more information about the WITH OWNERACCESS OPTION clause, type **owneraccess** in the Office Assistant or on the **Answer Wizard** tab in the Microsoft Access Help window, and then click **Search**.

Using Advanced SQL in Access

Now that we've had a discussion of the advanced SQL syntax, let's look at some of the ways we can use it in an Access application.

Sample Database

Along with this article, there is a sample database called acAdvSQL.mdb.

Everything in acAdvSQL is based on all the topics covered in this article, and it demonstrates the different SQL statements discussed through queries and sample code. However, it is important to note that *you must first create a* ***workgroup information file*** *for the acAdvSQL database before you can use the user-level security SQL statements.*

Many of the sample queries used in acAdvSQL depend on certain tables existing and containing data, or on certain other preexisting database objects. If you are having difficulty running one of the queries due to missing data, open the Reset Tables form and click the Reset Tables button. This will re-create the tables and their original default data. To step through the reset table process manually, execute the following queries in this order:

1. Drop Table Shipping
2. Drop Table Invoices
3. Drop Table Customers
4. Create Table Customers
5. Create Table Invoices
6. Create Table Shipping
7. Populate Customers
8. Populate Invoices
9. Populate Shipping

Setting the Database Password

In this example, we create a form that will allow a user to change the database password. Let's say that we are creating a database that needs share-level security. You decide to implement this with Jet's database password capability, but you do not want to show the user interface for database passwords that is currently in Access, because it only allows you to set or remove the database password. By using inline SQL and a custom form, you can enable users to manage their own share-level security.

1. Create a new Access database and name it Password.mdb.
2. Create a new form and call it frmDBPassword.
3. Create three text boxes with matching labels and name the text boxes txtOldPwd, txtNewPwd, and txtConfirmPwd, and set all of their InputMask properties to Password.
4. Create three command buttons and name them cmdCancel, cmdClear and cmdOK. Once completed, your custom form should look similar to the following (Figure 1):

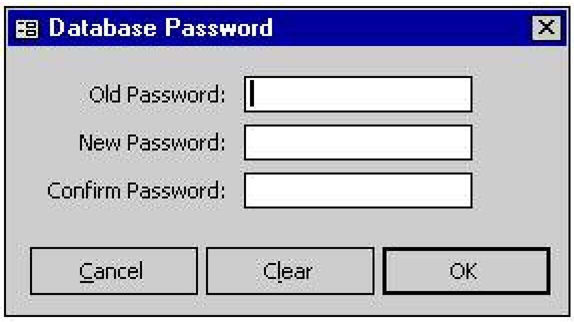


Figure 1. Database Password form

5. In the Click event of cmdCancel, enter the following code:

```
DoCmd.Close acForm, "frmDBPassword"
```

6. In the Click event of cmdClear, enter the following code:

```
Dim conDatabase As ADODB.Connection
Dim SQL As String

On Error GoTo Error_Handler

Set conDatabase = Application.CurrentProject.Connection

'Clear the password only if the old one is given.
If Not IsNull(txtOldPwd) Then
SQL = "ALTER DATABASE PASSWORD NULL " & txtOldPwd
conDatabase.Execute SQL
MsgBox "The password has been successfully cleared.", _
       vbInformation
Else
MsgBox "Please enter the current (old) password.", vbExclamation
txtOldPwd.SetFocus
Exit Sub
End If

'Close out the form and all object variables.
conDatabase.Close
Set conDatabase = Nothing
DoCmd.Close acForm, "frmDBPassword"

Exit Sub
Error_Handler:
MsgBox Err.Description, vbCritical
```

7. In the Click event of cmdOK, enter the following code:

```
Dim conDatabase As ADODB.Connection
Dim SQL As String
Dim intCheckPwd As Integer

On Error GoTo Error_Handler

Set conDatabase = Application.CurrentProject.Connection

'Check that the new and confirmed passwords match.
intCheckPwd = StrComp(txtNewPwd, txtConfirmPwd, vbBinaryCompare)

'Determine if we are changing/clearing the current password
'or setting a new one.
If txtOldPwd.Enabled = False And intCheckPwd = 0 Then
SQL = "ALTER DATABASE PASSWORD " & txtNewPwd & " NULL"
conDatabase.Execute SQL
MsgBox "The new password was successfully set.", vbInformation

'Change the current password.
ElseIf intCheckPwd = 0 Then
```

```
'Check that the old password was given.
If IsNull(txtOldPwd) Then
MsgBox "Please enter the current (old) password.", _
       vbInformation
txtOldPwd.SetFocus
Exit Sub
End If

SQL = "ALTER DATABASE PASSWORD " & txtNewPwd & " " & txtOldPwd
conDatabase.Execute SQL
MsgBox "The password was successfully changed.", vbInformation

'Password was not confirmed.
Else
MsgBox "The password was not confirmed.  Please try again.", _
       vbExclamation
txtConfirmPwd = vbNullString
txtConfirmPwd.SetFocus
Exit Sub
End If

'Close out the form and all object variables.
conDatabase.Close
Set conDatabase = Nothing
DoCmd.Close acForm, "frmDBPassword"

Exit Sub
Error_Handler:
MsgBox Err.Description, vbCritical
```

8. In the OnLoad event of frmDBPassword, enter the following code:

```
Dim conDatabase As ADODB.Connection
Dim SQL As String

On Error Resume Next

'Check to see if the database password is enabled.
Set conDatabase = Application.CurrentProject.Connection
SQL = "ALTER DATABASE PASSWORD NULL NULL"
conDatabase.Execute SQL

'If the SQL statement did not cause an error the database
'password is not currently enabled.
If Err.Number = 0 Then
txtOldPwd.Enabled = False
cmdClear.Enabled = False
Me.Caption = "New Database Password"
Err.Clear
Else
Me.Caption = "Change Database Password"
End If

'Close out all object variables.
conDatabase.Close
Set conDatabase = Nothing
```

Note the trick used in the code sample above to determine if a database password is currently set on the database. The SQL statement uses two NULL values. If the SQL statement causes an error, it means that there is already a database password set for the database.

Using a Procedure

One of the great things about using procedures is that because they are built with parameters, you can use them in very generic ways. For example, in the example used in the procedures section above, a DELETE statement was used to delete all invoices older than a certain given date. Because the procedure used a parameter for the date to delete from, we can prompt the user for the date and simply pass it to the procedure.

The following code uses the **InputBox** function to prompt the user for a date:

```
Private Sub cmdExecuteProc_Click()
Dim conDatabase As ADODB.Connection
Dim RtnValue As String
Dim SQL As String

On Error GoTo Error_Handler

Set conDatabase = Application.CurrentProject.Connection
RtnValue = InputBox("Enter the date to delete from:", _
           "Delete Invoices", "1/1/1999")

If IsDate(RtnValue) Then
SQL = "EXECUTE DeleteInvoices '" & RtnValue & "'"
conDatabase.Execute SQL
MsgBox "The invoices have been deleted.", vbInformation
Else
MsgBox "An invalid date was given.  Please try again.", vbInformation
End If

conDatabase.Close
Set conDatabase = Nothing

Exit Sub
Error_Handler:
   MsgBox Err.Description, vbInformation
End Sub
```

One Last Comment

Although this article has focused on SQL statements that are typically found in multiuser solutions, that is not to say that the statements discussed cannot be used in single-user solutions. Where and when you use Data Definition Language and Data Manipulation Language SQL statements should depend on the requirements of your application.

Additional Resources

Resource	**Description**
Fundamental Microsoft Jet SQL for Access 2000	This article discusses the basic mechanics of using Jet SQL to work with data in an Access 2000 database. It also delves into using SQL to create and alter a database's structure. If you're new to manipulating data with SQL in an Access database, this article is a great place to start.
Intermediate Microsoft Jet SQL for Access 2000	Second in the series of SQL articles, this one shows how using intermediate SQL can add greater power and flexibility to your Access applications. More complex statements will expand the range of ways to access and process the information in your database. Using intermediate SQL will also enable you to take greater control of how your database is used and maintained.
Microsoft Jet SQL Reference Help	This is the definitive source for the SQL language as it applies to Access 2000. It can be found in the Contents section of Microsoft Access 2000 Help.
Microsoft Jet Database Engine Programmer's Guide	This book is everything you wanted to know about programming with the Microsoft Jet Database Engine.
Building Applications with Forms and Reports	Here you can find solid development techniques for developing an Access 2000 application.
Microsoft Access 2000 Help	An irreplaceable source of Access 2000 programming topics.
Microsoft Office 2000/Visual Basic Programmer's Guide	This comprehensive book covers how to use VBA to program Office 2000 applications.
MSDN, http://msdn.microsoft.com/	This Web site always has the latest information for developing solutions with Microsoft platforms and languages.
Microsoft Office Developer, http://msdn.microsoft.com/office/	This Web site contains the latest information about developing applications with Microsoft Office.
MSDN Training, Career, and Events	Look to Microsoft's MSDN Training courses to provide the soundest techniques for developing Access 2000 applications.

Print E-Mail Add to Favorites

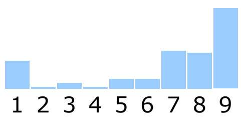
How would you rate the quality of this content?

	1	2	3	4	5	6	7	8	9	
Poor	○	○	○	○	○	○	○	○	○	Outstanding

Tell us why you rated the content this way. (optional)

Submit

Average rating:
7 out of 9



234 people have rated this page