**Designing a Database**

Creating a database from a design involves a large number of tedious processes of creating tables, fields, indexes, relationships, default values, rules, and so on. While this can all be done with pure VB code, it's an an obscure and complex process. Here's a short example from the VB help file under the CreateTableDef topic that will create a table with a single field:

Dim MyTableDef As TableDef, MyField As Field

Dim MyDatabase As Database

Set MyDatabase = Workspaces(0).OpenDatabase("BIBLIO.MDB")

' Create new TableDef.

Set MyTableDef = MyDatabase.CreateTableDef("Title Detail")

' Add field to MyTableDef.

Set MyField = MyTableDef.CreateField("Comments",dbDate)

MyTableDef.Fields.Append MyField

' Save TableDef definition by appending it to TableDefs collection.

MyDatabase.TableDefs.Append MyTableDef

Now, you could skip using the CreateTableDef and CreateField methods and perform the intire process with a single call to the Execute method of the database object using a Jet DDL SQL statement. Here's the general format for a SQL CREATE TABLE statement:

CREATE TABLE table (field1 type [(size)] [index1] [, field2 type [(size)] [index2] [, ...]] [, multifieldindex [, ...]])

The code might look like this (I've modified some names from the previous example for simplicity):

Dim db As Database

Dim sSQLDDL As String

sSQLDDL = "CREATE TABLE TitleDetail ( Comments DATETIME );"

Set db = Workspaces(0).OpenDatabase(App.Path & "\" & App.EXEName & ".mdb")

db.Execute sSQLDDL

db.Close

Set db = Nothing

Keep in mind that in both of these examples, we have code that creates one table with one field. There are no rules, defaults, indexes, or primary or foreign keys involved. Additionally, the code contains no error handling. Now consider what the code would look like to create a large database design consisting of hundreds of fields in dozens of tables, complete with primary and foreign key constraints, rules, defaults, and so on.

In general, you'll find all aspects of designing a database considerably simpler in Access. Here's an example of what's required to accomplish some basic design tasks using the Access interface:

* Create a Table   
  Select the Table tab in the database window and click New.
* Create a Field   
  In table design view, type the name of the field in the grid, assign a datatype and set any other properties as necessary in the property sheet.
* Create an Index   
  In table design view, select the field and set the indexed property in the property sheet.
* Create a Relationship   
  Open the Relationships window and add the two tables involved in the relationship. Drag a field from one table onto a field in the other table. In the dialog box that appears, check the appropriate boxes to determine if the relationship will be enforced and if it will use cascading updates and deletes.

As you can see, the Access interface greatly simplifies the process of building a database.

**Building Queries**

Once you have built the basic database design, including tables, fields, indexes, relationships, rules, defaults, and so on, the most common task you will perform will be building queries. All queries are created with Structured Query Language (SQL), but Access provides a graphical design tool for building queries that makes the process easier, faster, and more reliable. It also gives you a chance to see what the SQL for a complex query looks like without needing to fully understand all of the SQL syntax involved. This makes the query design window both a design tool and a learning tool.

Here's an example of the SQL for a complex query that's used as the base source of several reports in a large database application:

SELECT DISTINCTROW

HLLink.HLID, HLLink.HLCaseID, HLLink.HLRemarks, HLLink.HLCRCInit,

CaseArchive.CaseStatusClass, CaseArchive.CaseStatusCategory,

Organizations.OrgName, Organizations.OrgDept,

Organizations.OrgAddress1, Organizations.OrgAddress2,

Organizations.OrgCity, Organizations.OrgState, Organizations.OrgZIP,

Organizations.OrgPhone, Organizations.OrgFax, Organizations.OrgCategory,

People.PersonPrefix,

People.PersonFirst, People.PersonMiddle, People.PersonLast,

People.PersonSuffix, People.PersonTitle,

People.PersonAddress1, People.PersonAddress2,

People.PersonCity, People.PersonState, People.PersonZIP,

People.PersonPhone, People.PersonPExt, People.PersonFax, People.PersonCategory,

TimeTable.TTBillingID, TimeTable.TTTaskID, TimeTable.TTUser,

TimeTable.TTStart, TimeTable.TTSeconds, TimeTable.TTAddMin,

Cases.CaseReportingRegion, Cases.CaseReportingState,

Cases.CaseTypeClass, Cases.CaseTypeCategory,

Cases.CaseStatusClass, Cases.CaseStatusCategory,

Cases.CaseDiagnosis, Cases.CaseIsOffLabel,

Plans.PlanClass, Plans.PlanCategory, Plans.PlanTitle,

Products.ProductName,

Diagnoses.Diagnosis

FROM Diagnoses

RIGHT JOIN (Organizations

RIGHT JOIN (TimeTable

INNER JOIN ((Plans

RIGHT JOIN (People

RIGHT JOIN (OPLink AS PhysicianOPLink

RIGHT JOIN (Products

RIGHT JOIN (Cases

RIGHT JOIN HLLink

ON Cases.CaseID = HLLink.HLCaseID)

ON Products.ProductID = Cases.CaseProductID)

ON PhysicianOPLink.OPID = Cases.CasePhysicianOPID)

ON People.PID = HLLink.HLPID)

ON Plans.PlanID = Cases.CasePlanID)

LEFT JOIN CaseArchive

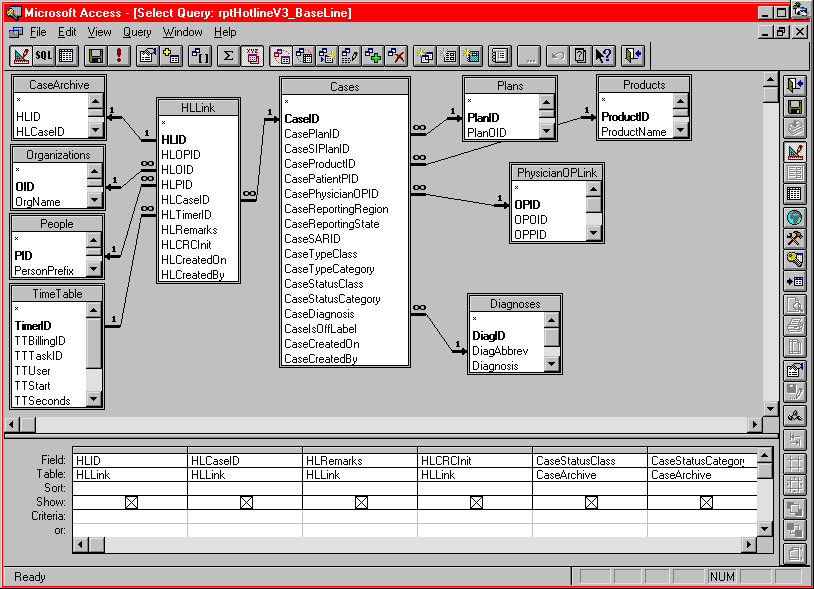
ON HLLink.HLID = CaseArchive.HLID)

ON TimeTable.TimerID = HLLink.HLTimerID)

ON Organizations.OID = HLLink.HLOID)

ON Diagnoses.DiagAbbrev = Cases.CaseDiagnosis;

Clearly this is a query that is more complex than what is typical in most applications. The effort required to build this query as a SQL statement would be considerable given the complexity, the number of joins, etc. Here's the same query seen in the Access query design window:



It doesn't take a user interface design expert to see that this query can be constructed and tested much faster using Access than by coding and testing it in Visual Basic. The process of adding tables and joining them using the query design window is as simple as drag and drop. If you've already defined the relationships between tables, Access will even add the joins for you.

Some aspects of building SQL queries are fairly simple to create by just entering the SQL statement. If you're doing a simple select of a few fields from a single table, it's easy enough to just type:

SELECT FirstName, LastName FROM Customers WHERE LastName LIKE 'Smith';

If, however, you're building a query with several tables and multiple outer joins, the query design window will make the process of building and debugging the query much easier. A final bit of advice I'll offer is that you should not use the query design window as a substitute for understanding Jet SQL. If you're building a serious database application, at some point you'll probably need to dynamically generate the SQL for a query in your code. Study the SQL generated by the query designer and the help files to learn the syntax for builing inner and outer joins, setting criteria and sorting records.

**Managing Security**

Setting up a secured database is one of the most misunderstood topics in Visual Basic. With VB 4.0 and Jet 3.0, you at last have the capability of programmatically assigning permissions to database objects. However, the process is complex and you will still need a copy of Access to create the system database file which is the cornerstone of Jet security. Unfortunately, the Access interface for managing object permissions isn't that much of an improvement over assigning them programmatically.

Properly securing a database isn't difficult, but there are several steps involved which must be followed for the database to be secure.

Setting the permissions for an object using VB code consists of getting a reference to a Document object and assigning the appropriate value to the permissions property. Here's a fairly useless example from the VB help file:

Dim MyDB As Database, MyDoc As Document

Set MyDB = Workspaces(0).OpenDatabase("MYDB.MDB")

Set MyDoc = MyDB.Containers(0).Documents("MyDoc")

Debug.Print MyDoc.Permissions

The reason the example is useless is that it doesn't actually set any permissions. There's only one key thing to remember when setting the permissions property and that is that the property is a bit field containing all the permissions in a single value. While you can replace the entire set by assigning a value to the property, if you wish to add or remove a permission, you must use the bitwise operators. In the preceding code, here's how you would add the Modify permission to the object:

MyDoc.Permissions = MyDoc.Permissions Or dbSecWriteDef

To add a permission to an object, Or the appropriate constant with the existing permissions property.

Here's how you would remove the same permission:

MyDoc.Permissions = MyDoc.Permissions And Not dbSecWriteDef

To remove a permission, use the bitwise And Not operator.

The reason you must use the bitwise operators rather than simply adding or subtracting the constant is that if you happen to add a constant to a permissions property where the permission has already been assigned, the operation will improperly modify the value. The bitwise operators will simply have no effect if the value has already been set (or removed). This applies to bitfields in general - always use the bitwise operators when working with bitfields.

By comparison, the Access user interface provides relatively easy to use dialogs for assigning permissions. Simply select the object from the appropriate object list, select the user or group from the users and groups list, and check or clear the permissions as appropriate. When you're done, click the Assign button.

For what its worth, the Access interface for assigning permissions could use some work. Here's some of the reasons why:

* The small, fixed size dialog box makes it difficult to work with a large number of objects simultaneously.
* The dialog presents a list of users rather than groups by default. As a matter of habit, you should always assign permissions to groups rather than users in a multiuser situation - whether it be a database application or anything else.
* In order to manage users and groups, you need to open a separate, modal dialog box.
* There's no means provided to see inherited permissions for an object when looking at a user account.
* The means by which you assign default permissions for new objects is less than obvious and poorly documented.
* Some permissions cannot be modified at all using the user interface. For example, there is no means provided to grant or revoke the ability to create new objects. This permission is granted by default and you must use DAO code to revoke it. If you wish to prevent users from creating new tables or other database objects, you'll need to write code for it.
* You need to open a database file to create and manage user and group accounts, even though the accounts are bound to a system database file (Access calls this a workgroup) rather than a particular database.

Overall, the entire set of security management tools in Access are a bit clumsy, but still better than writing a lot of code for something that's done only once or at best infrequently. An improved user interface for managing security with Jet databases would be a fine small project that would be much appreciated by many Access and Visual Basic developers. If you're interested in programming Jet security, this would be a good starter project. Perhaps you can beat me to the punch - my version is definitely on the back burner.

*Note: Keep in mind that if you will be creating secured applications with Visual Basic, you must have a copy of Access available to create the necessary system database. There is no means available with Visual Basic code to create the required file. Only the Access Workgroup Administrator can create a system database.*

Originally written by Joe Garrick