C# Database Project Concepts

Connecting a project to a database:

* Can add the database as a data source to the project. This creates a copy of the original database and stores it in the project directory. It also automatically makes your connection object for you, so the database is always available for processing. Handy for testing your application without actually changing the original database. *TableAdapterManager* objects can be used to fill a run-time dataset that contains the rows to be accessed.
* Instead of adding the database to the project, it can be accessed directly by the project from its original location. This can be done by creating a manual connection object and opening/closing the database whenever the application needs to. This gives ultimate control to the programmer, but requires somewhat more code due to the manual process. *OleDBDataAdapters* can be used to fill a run-time dataset that contains the rows to be accessed, or the data adapter can be used directly for processing tables as well.

Example of a manual connection (put statements in code where appropriate):

using System.Data.OleDb;

OleDbConnection conn = new OleDbConnection();

conn.ConnectionString = "PROVIDER=Microsoft.ACE.OLEDB.12.0;" + "Data Source = StudentDatabase.accdb";

conn.Open(); //open connection

OleDbDataAdapter da = new OleDbDataAdapter("SELECT \* FROM mytable", conn); //tie data adapter to the actual database

Dataset ds; //optionally, a dataset can be created and populated

da.Fill(ds, "MyDataSet"); //this dataset now holds all the rows from the query above

Example of a SELECT statement:

string sel = "SELECT \* FROM mytable WHERE mykey = mykeyvalue”; //sql select string

OleDbCommand command = new OleDbCommand(sel, conn); //make a command from the string, using your connection

OleDbDataReader odr = command.ExecuteReader(); //use the command to create a data reader, which will contain the rows

if (odr.HasRows) //boolean property: check to see if any rows were returned to the reader

Example of an INSERT statement:

string ins = "INSERT INTO mytable VALUES(myvalue1, myvalue2, etc.)"; //sql insert string

OleDbCommand command = new OleDbCommand(ins, conn); //make a command from the string, using your connection

int numRows = command.ExecuteNonQuery(); //use the command to insert the row

if (numRows > 0) // check to see if/how many rows were inserted (should be a value of 1)

Example of an UPDATE statement:

string upd = "UPDATE mytable SET mycolumn1 = myvalue1, mycolumn2 = myvalue2 WHERE mykey = mykeyvalue“; //sql update string

OleDbCommand command = new OleDbCommand(upd, conn); //make a command from the string, using your connection

int numRows = command.ExecuteNonQuery(); //use the command to update the row

if (numRows > 0) // check to see if/how many rows were update (should be a value of 1)

Example of a DELETE statement:

string del = "DELETE \* FROM mytable WHERE mykey = mykeyvalue“; //sql delete string

OleDbCommand command = new OleDbCommand(del, conn); //make a command from the string, using your connection

int numRows = command.ExecuteNonQuery(); //use the command to delete the row

if (numRows > 0) // check to see if/how many rows were delete (should be a value of 1)

Miscellaneous:

* Consider making your database objects public static variables so that multiple forms can use them.
* Open of the connection could be done at form load.
* Closing the connection ( conn.close() ) could be done at form exit.
* Be sure to place database statements in try/catch to catch potential exceptions.
* Important to use the debugger to determine which statement might be in err.