# 10 Database and ADO.NET Fallacies

dotNet > ADO.Net

**Submitted By:** Tim Cassidy, Deloitte & Touche LLP
**Posted On:** 9/9/2004 12:00:00 AM

**Description:**

## Database and ADO.NET Fallacies

## Tim Cassidy

## Introduction

I wrote this document simply because I've witnessed first hand some dreadful fallacies emerge in the IT industry. The fallacies have likely been adopted largely because even though the developer is writing inefficient code, the code still works relatively quickly thanks, in most, to behind the scenes optimizations and ever faster hardware. But should the application be run on a lesser machine or more likely should the database grow in size – the application will be become unusable simply due to those same misguided individual(s).

A warning to those who shudder when well established nomenclature isn't used and instead less conventional language or slang is used in its place:
It is important to recognize that rows are not records, fields are not columns and tables are not files. However, in some instances it is easier to speak of a single row in a table as being a single record and so on. So in some sections of this document I have abused conventional database terminology by using record to refer to a row or an attribute to refer to a column, etc.

## Ten Database and ADO.NET Fallacies

The following comprise the list of what I would consider the most misguided fallacies in regards to designing a data model to be used in a SQL Server database and the subsequent use of that database in ADO.NET – though some of the concepts are applicable across DBMS's and across application programming languages. Lastly, two of these fallacies are only applicable if the front end of your application happens to be ASP.NET.

### *1 Use Multi-Value Attributes Instead of Junction Tables*

Some people have been informed of the dangers/cost of table joins. This has caused them to implement their own "light-weight" solution to table joins. Unfortunately this usually results in the "light-weight" solution costing many times more than the traditional table join. Relational databases have existed for decades and have been refined over those years. It would be folly to think that one could come up with a better solution in a much shorter span of time without understanding the underlying mechanics of those databases.

In one instance I met someone who was convinced that using a comma-delimited string as an attribute was a better solution than using a junction table. To illustrate what they proposed I have created the following simplistic example:

There is a table “student”:

| student_num | first_name | last_name | classes |
|---|---|---|---|
| 123456 | John | Smith | 1,2,4 |
| 123457 | Jane | Doe | 1,3,5 |

And a table “class”:

| Uid | name | building | room_number |
|---|---|---|---|
| 1 | Biology | Grant Hall | 123 |
| 2 | History | McConnell Hall | 231 |
| 3 | Geology | Jeffrey Hall | 312 |
| 4 | Math | Richardson Hall | 213 |
| 5 | Philosophy | Dunning Hall | 321 |

Effectively anytime that you wanted to “select” which students attended a particular class, you would have to select all student records and parse the classes attribute of each record.

As opposed to:
The new “student” table (notice the missing column “classes”):

| student_num | first_name | last_name |
|---|---|---|
| 123456 | John | Smith |
| 123457 | Jane | Doe |

The same “class” table shown previously and lastly the junction table:

| student_num | class_uid |
|---|---|
| 123456 | 1 |
| 123456 | 2 |
| 123456 | 4 |
| 123457 | 1 |
| 123457 | 3 |
| 123457 | 5 |

Now, to get all students registered to a particular class, an inner join can be done between the student table and the junction table. In many instances, the class table wouldn’t need to be included in the join since the class data – including the UID - would likely be stored in memory when the system starts up. This in-memory storage would likely take the form of a hashtable data structure to increase efficiency.

In the above section I have informally stated the reason why using a comma delimited list is an ineffective solution, but more formally:

1. The comma delimited list can’t effectively be indexed.
2. Any select queries done on the mapping for the comma delimited list would require string parsing.
3. It removes the possibility of database optimizations on relationships between tables.

Another approach that people may be tempted to adopt is to use XML in a single column to store

different attributes. Typically people will come to the conclusion that using mark-up is an efficient alternative to additional columns in a table. This is likely because of .Net's XMLReader and XMLWriter classes which, through various sources, have become associated with the definitive way to access data – whether that data's source is a database or a file.

In some instances, storing XML in a single column is useful if there is no expectation that this data will be used in any database queries. Therefore, when in doubt use well understood database concepts (additional columns, junction tables, etc) to store the data.

## 2 Store XML to Capture Attributes Instead of Using Columns

As previously mentioned, whether it be the use of XML or some other mark-up, if queries are likely to involve these attributes, using XML instead of a separate column is only going to slow those queries.

Using the "student" table from the previous section:

| student_num | first_name | last_name |
|---|---|---|
| 123456 | John | Smith |
| 123457 | Jane | Doe |

And the alternative of using XML in a single "content" column in the "student" table:

| content |
|---|
| <student_num>123456</student_num><br><first_name>John<br></first_name ><br><last_name>Smith</last_name<br>> |
| <student_num>123457</student_num><br><first_name>Jane</first_name<br>><br><last_name>Doe</last_name<br>> |

Or

| content |
|---|
| <student first_name="John" last_name="Smith" student_num="123456" /> |
| <student first_name="Jane" last_name="Doe" student_num="123457" /> |

In either of the above two alternatives the same problems arise as mentioned in Section 1 – "Using Multi-Value Attributes Instead of Junction Tables".

## 3 Filtering of Data in .Net DataSets is Inefficient

There are two basic approaches when using .Net DataSets:

1. Getting a massive amount of data from the database and then performing what amounts to filtering on that data in your application, or
2. Using customized database queries based on the data that you want, such that the DataSet will only contain the data that you require, in the order that you require it.

The real question is "Is there ever an instance when it is more efficient to rely on DataView's RowFilter property or GetRow(s) methods [1] instead of just issuing a separate SQL query?"

In most cases, the correct approach is to use a mix of the above two. If you rely too heavily on SQL queries to give you exactly the data you require in exactly the order you require it, communication with the Database can become the bottleneck for the program. Conversely, if too much reliance is placed on filtering using the DataView property RowFilter or the DataView method GetRow(s), the bottleneck will become filtering vast quantities of data.

As previously mentioned DataView's RowFilter property can be used to filter data contained within a DataSet object. In the case that you would like to navigate through the records contained in a single table:
Suppose you have a table with a primary field (named "UID"), a foreign key attribute (named "parent_UID") that references the table's primary key and you need all the records that have a parent_UID of "2". You could retrieve some subset of the table's data into a DataSet object and then use (the following code is written in VB.NET but can easily be translated into one of the other .Net languages):

```
Dim ChildView as DataView = new DataView (dataSetObj.Table("some_table"))
ChildView.RowFilter = "parent_UID = '2'"
```

and then iterate through the ChildView object to get all records that have a parent_UID of "2".

In the case, that the child records were in a different table from the parent, you would just need to create the DataView on the other table – like so:

```
Dim ChildView as DataView = new DataView (dataSetObj.Table("another_table"))
```

The method GetRow or GetRows can be used in a similar manner to the above RowFilter property.

```
Dim ChildView as DataView = new DataView (dataSetObj.Table("some_table"))
ChildView.GetRow("parent_UID = '2'")
```

## 4 .Net Connection Pooling doesn't require any Thought

While it is true .Net connection pooling is automatically enabled – it is important that your application is designed to take full advantage of its use. I once witnessed an application that used only a single shared (in C# syntax this is known as "static") connection object. This effectively resulted in the entire application using only a single connection across all threads/contexts (i.e. no connection pooling). In other words all users of the application would have to be queued up in order to submit their queries through this connection object. For some applications having a single connection shared across the entire application can be useful. However, in most cases, it is important to exploit the benefits of connection pooling.

In case there are concerns that somehow connection pooling is disabled, simply add the following string:
Pooling=true;
to the connection string for the SQLConnection Object.

Conversely, as previously mentioned, in some instances it is appropriate to disable connection pooling, in these instances add the following connection string to the SQLConnection object:
Pooling=False;
For an OLEDBConnection object, simply add the following:
OLE DB Services=-4;

By doing this the OLE DB data provider will ensure that no connection pooling is used for the connection object.

## 5 "Null" Should always be used for Undefined Values

There is no clear-cut answer to this. Debates have been raging on this topic for almost as many years as

there have been databases – specifically the use of Null in relational databases is the cause the of the most heated debates.

In my opinion, unless you are a database expert and therefore fully understand the ramifications of using Null (storage space in fixed length columns, indexing difficulties, processing time involved in decoding bitmap values, three-valued logic in aggregate functions, etc), don't use it. The alternative to supporting Nulls is to make use of defaults. You can find out more about defining default values for missing or unknown entries by reading "Inside SQL Server 2000" by Kalen Delaney.

It is also important to be aware that the default values for a table can be retrieved using a SQL query on the system tables. Defaults values of columns use the character mapping "D" in the table "dbo.sysobjects.xtype". Therefore, the application developer should not hard-code the default values for the database into the application – since these defaults could change as the data model evolves.

## 6 ViewState is the Definitive Caching Structure

Though the ViewState is not a part of ADO.NET or Databases (it is an ASP.NET class), caching structures/mechanisms can often be used effectively to reduce the requirements on the data layer of your application. The ViewState is used on a per page basis (i.e. when requests are resubmitted to the same page) to cache information on the client computer in the form of a hidden input field. However, this caching is done by serializing the object and then outputting the object to the page's hidden input field. Then when the page is resubmitted, the value stored in the hidden input field must be deserialized which can end up being more expensive then simply reissuing a SQL query (or whatever mechanism is being used to access data).

ViewState objects definitely have a valid use just like any mechanism in .Net. However, they should only be used sparingly and only in very select circumstances. Using the Northwind database example, the removal of ViewState caching from one page resulted in the results being displayed over 100 times quicker (difference between less than a second and over a minute and a half).

The reason for this dramatic timing difference is due to the fact that there is a large amount of overhead associated with serializing/deserializing any object (especially as the object size increases). Good candidates for storage in the ViewState are primitive types, String, ArrayList, and HashTable types. These types are serialized using an optimized object serialization format.

## 7 Normalization Always Results in a Better Database

Sometimes strict adherence to the normalization process doesn't equate to a well designed data model. That is, the requirements of an application can be in violation with the process of normalization.

In this instance, just because you *can* save space by normalizing data and extracting it out to other tables, doesn't mean you should. As every attentive student remembers from their school days, if data of one type is likely to be repeated across records, then this data can be extracted into a separate table. Thus when a set of records reference common data, they can reference the same record in the different table.

However, when speed is more crucial than space, it is better to simply aggregate all of that data into the same table and take the hit on storing data inefficiently. This avoids having to perform table joins and thus increases the throughput of queries that make use of those joins.

Using the "student" table mentioned previously, we might want to add campus address information to the student records. We may also realize that many students tend to live together – for instance, on

average three to four students would end up having duplicate addresses. Therefore, we might see this as an opportunity to normalize the data using second normal form.

Thus, our student table would look like this:

| student_num | first_name | last_name | campus_address_uid |
|---|---|---|---|
| 123456 | John | Smith | 3 |
| 123457 | Jane | Doe | 4 |

Plus the additional table called “campus_address”:

| uid | street_address | apt_num |
|---|---|---|
| 1 | 123 Oak Road | 42 |
| 2 | 123 Elm Street | 1 |
| 3 | 321 Elm Street | 2 |
| 4 | 231 Pine Street | 42 |

However, the above data model would require a join every time we wanted to see the campus address of a particular student [2]. Therefore, if processing speed was our primary concern and storage space was in large supply, the following table would be an improvement (even though it is a violation of second normal form):

| student_num | first_name | last_name | campus_street_address | apt_num |
|---|---|---|---|---|
| 123456 | John | Smith | 321 Elm Street | 2 |
| 123457 | Jane | Doe | 231 Pine Street | 42 |

## 8 Recursive Problems Require Recursive Stored Procedures

For the purposes of this paper, a recursive problem is simply any problem whose system has a state space of arbitrary depth. More specifically in regards to databases it concerns data models that support an arbitrary number of associations between entities/records. While the solution to these systems typically requires a recursive solution – this doesn’t mean that these systems require a recursive stored procedure to navigate those associations. One problem with using recursive procedures is that it is not possible to union the results of a stored procedure that has been invoked by another stored procedure [3].

Often the more natural solution is to simply make use of temporary tables. That is, to store the results of each iteration of a looping structure into a temporary table. Thus each iteration of the looping structure will add to the existing set of data in the table. The index into this data can be achieved using a cursor object.

## 9 Conditional Statements in Stored Procedures Won’t Affect Performance

Just because SQL supports them doesn’t mean it is a good idea to use them. In general the fewer conditional statements used the better. Obviously you need conditional statements to control your application – however, there are ways of reducing the number of conditional statements used by a program – thereby increasing the performance of your application.

For one application, a stored procedure similar to the following stored procedure was being invoked on average 100 times per user. This means that every user was causing 200 conditional statements to be executed in this one stored procedure each time they accessed the application:

```
CREATE PROC get_immediate_children_sections @section_UID int = NULL, @role_UID int = NULL
```

```
AS
If (@section_UID is NULL)
BEGIN
if (@role_UID is Null)
BEGIN
SELECT *
FROM tc_section
WHERE tc_section.parent_section_UID is NULL
END
else
BEGIN
SELECT *
FROM tc_section, tc_role_section_junction
WHERE tc_section.parent_section_UID is NULL
AND tc_role_section_junction.section_UID = tc_section.UID
AND (tc_role_section_junction.role = @role_UID
OR tc_role_section_junction.role = 0)
END
END
Else
BEGIN
if (@role_UID is Null)
BEGIN
SELECT *
FROM tc_section
WHERE tc_section.parent_section_UID = @section_UID
END
else
BEGIN
SELECT *
FROM tc_section, tc_role_section_junction
WHERE tc_section.parent_section_UID = @section_UID
AND tc_role_section_junction.section_UID = tc_section.UID
AND (tc_role_section_junction.role = @role_UID
OR tc_role_section_junction.role = 0)
END
END

GO
```

It should also be noted that by using the ANSI NULL setting in SQL Server you can remove the necessity to use special conditional terminology in handling of Nulls [4]. Or even better use the suggestion in Section 5 - ""Null" Should Always be Used for Undefined Values" to remove the requirement to support Nulls in your data model.

The loss in performance is especially evident in stored procedures because all applications supported by a database will eventually have to make a query to the database – i.e. very often when the objective is to speed up an application, the first place to look is at the data layer.

## 10 Strong Security is Difficult to Achieve in an ASP.NET Application Using a Database

Building security into any system is never easy because the process of securing a system is essentially just making that system less accessible. In order to properly secure a system an appropriate amount of thought and time must be spent on determining how to build that security into the system. The earlier in the project that this time is spent planning for security, the more time you will save when building that security into the system.

Often we can go too far and make the application non-functional because of an attempt to restrict access too far. On the flip side, we can end up leaving the system unintentionally wide open. Often the best practice is to follow standard security methodologies and lock-down the system in that manner instead of attempting to invent your own methods. That said, adding your own layer(s) to an existing security methodology is a good example of layered security.

Microsoft has released a paper on how to encrypt the credentials of an account being impersonated by an ASP.NET application. That paper can be found here:
http://support.microsoft.com/default.aspx?scid=kb;en-us;Q329290

To summarize the above paper, by using the ASP.NET utility Aspnet_setreg.exe you can effectively encrypt the account name and password of an account on your system. Then place that encrypted text into the registry. After which you simply need to grant read access to that small portion of your registry to the ASP.NET account and your application is now making use of encrypted credentials [5].

After this has been completed, SQL Server allows the administrator to restrict access to chosen stored procedures, selects from chosen tables, etc for any account on the server (technically this is directly done through SQL Server roles). In other words, you can set security for the data layer to a very granular level on a per role basis. In addition, regular windows security can be employed to restrict that account's access to the file system as well.

## Summary

In the above paper, I have discussed some database and ADO.NET fallacies that I have encountered in my career as a developer. Most of the sections simply provide guidance rather than any hard set rules. I certainly wouldn't claim that this comprises the total sum of misunderstandings in ADO.NET or Database development - but hopefully by making yourself aware of the above fallacies – you can avoid falling victim to them.

## References

N. Saluja. ADO.NET Connection Pooling FAQ, May 12, 2004. http://www.c-sharpcorner.com/Code/2004/May/PoolingNS.asp.

Microsoft, How To Use the ASP.NET Utility to Encrypt Credentials and Session State Connection Strings. 2004. http://support.microsoft.com/default.aspx?scid=kb;en-us;Q329290.

T. Landgrave. Client-side state management techniques for .NET architects. http://builder.com.com/5100-6389-1058704.html.

M. Chapple. Database Normalization Basics. 2004. http://databases.about.com/library/weekly/aa080501a.htm, http://databases.about.com/library/weekly/aa081901a.htm, http://databases.about.com/library/weekly/aa090201a.htm, http://databases.about.com/library/weekly/aa091601a.htm

K. Delaney. “Inside SQL Server 2000”, Microsoft Press, 2001.

C. Kempster. Data Type Performance Tuning Tips for Microsoft SQL Server. June 24, 2003. http://www.sql-server-performance.com/datatypes.asp

J. Danna. Documentum. Personal Communication, 2004.

---

[1] Another possibility is to use DataTable’s Select Method – which returns an array of DataRow objects.

[2] Depending on the number of records in the associated table it might also be feasible to simply keep the secondary table in memory. For example, in our first example we used classes as the secondary table – in this instance the data from this table would likely be stored in an efficient data structure like a hashtable and indexed on the UID for each record.

[3] It is possible to achieve this in an indirect manner by using cursors or table objects.

[4] The SQL-92 standard requires that an equals (=) or not equal to (<>) comparison against a null value evaluates to FALSE. When SET ANSI NULLS is ON, a SELECT statement using WHERE column name = NULL returns zero rows even if there are null values in column name. A SELECT statement using WHERE column name <> NULL returns zero rows even if there are nonnull values in column name.

[5] There are some additional settings that have to be configured for the account to be impersonated but essentially those are the steps.