Database Structure and Coding Guidelines for Online Transactional Processing (OLTP) Databases Implemented on Microsoft SQL Server

Bradley Don Morris
bradleydonmorris@hotmail.com
http://bradleydonmorris.me

Contents

I. Database Access Methods
   a. Stored Procedures – As a general rule all calls to a database should be made by way of stored procedures. This will permit easier troubleshooting, exception handling, assist with standardizing security, improve performance over non parameterized dynamic SQL, and allow for proper source code control of databases.
   b. .Net Entity Framework – To ease the development burden of the application developers who are not familiar with TSQL or good database query design, the use of .Net Entity Framework will be permitted without using stored procedures. *In other words, it was 6 against 2, and this is how Bradley chose to compromise.* The caveat is that the application developer will be solely responsible for the poor performance of the code generated by Entity Framework. The only assistance that will be required from the database developers is to assist with the writing of stored procedures per application developer's specs once bad queries are discovered.
   c. Connection String – To assist with troubleshooting and to ensure all applications are connecting in like manner, all connection strings to SQL should include the following attributes.
      i. Data Source – Should be the fully qualified SQL server name, such as sql1.domain.local.
      ii. Initial Catalog – The main database calls will be made against.
      iii. Integrated Security – SSPI or True.
      iv. Application Name – The name of the application or web site making the call. In WinForm applications using the static property System.Windows.Forms.Application.ProgramName should suffice. If left unspecified, the .Net SQL Data Provider will pass in ".Net SqlClient Data Provider" for this parameter. This is not very helpful during troubleshooting.
      v. Workstation ID – The name of the computer making the call. Usually this is set by the .Net provider. However, using "System.Net.Dns.GetHostName()" in .Net applications will ensure that this is set properly.
      vi. Network Library – The following are all the available options for this parameter. However, only the first two options are to be used.
         1. dbmssocn (TCP/IP)
         2. dbmslpcn (Shared Memory)
            a. This is useful only during development, while an application and database may be debugged on the developer's workstation.
         3. dbnmpntw (Named Pipes)
            a. Though usable on a TCP/IP based network, Named Pipes require additional resolution and can be the source of a myriad of issues.
         4. dbmsspxn (IPX/SPX)
         5. dbmsvinn (Banyan Vines)
         6. dbmsrpcn (Multi Protocol)
         7. dbmsadsn (Apple Talk)
         8. dbmsgnet (VIA)

d. Sample Connection String section from an app.config or web.config file:

```
<configuration>
	<connectionStrings>
		<clear />
		<add
			name="TotallyAwesomeDatabaseConnection"
			providerName="System.Data.SqlClient"
			connectionString="Data Source=sqlserver.domain.local; Initial
Catalog=TotallyAwesomeDatabase; Integrated Security=SSPI; Network Library=dbmssocn;
Workstation ID=SOMEWEBSERVER1; Application Name=&quot;The Totally Awesome Super Hyper
Global Application&quot;;"
		/>
	</connectionStrings>
</configuration>
```

e. Sample Connection string using SqlConnectionStringBuilder:

```
System.Data.SqlClient.SqlConnectionStringBuilder sqlConnectionStringBuilder = new
System.Data.SqlClient.SqlConnectionStringBuilder();
sqlConnectionStringBuilder.DataSource = "sqlserver.domain.local";
sqlConnectionStringBuilder.InitialCatalog = "TotallyAwesomeDatabase";
sqlConnectionStringBuilder.IntegratedSecurity = true;
sqlConnectionStringBuilder.NetworkLibrary = "dbmssocn";
sqlConnectionStringBuilder.WorkstationID = System.Net.Dns.GetHostName();
sqlConnectionStringBuilder.ApplicationName = "The Totally Awesome Super Hyper Global
Application";
```

f. Sample Connection string using AMEN.Models.ConnectionString:

```
AMEN.Models.ConnectionString connectionString = new AMEN.Models.ConnectionString();
connectionString.Name = "TotallyAwesomeDatabaseConnection";
connectionString.DataSource = "sqlserver.domail.local";
connectionString.InitialCatalog = "TotallyAwesomeDatabase";
connectionString.DefaultSchema = "Global";
connectionString.IntegratedSecurity = true;
connectionString.NetworkLibrary = NetworkLibrary.TCPIP;
//connectionString.WorkstationID = System.Net.Dns.GetHostName();
connectionString.ApplicationName = "The Totally Awesome Super Hyper Global Application";
```

II. Object Naming Convention

a. Schema Scoping – All objects will be schema scoped, that is to say they will be defined within a specific schema, other than "dbo". Think of database schemas as namespaces. Objects (tables, stored procedures, etc.) of a specific scope should be grouped into a schema representing that scope. For example, tables and stored procedures that relate to employees may be placed in a schema named Employees. Tables and stored procedures related to employee benefits, even though having a heavy dependency on objects in Employees schema, may stand alone in their own schema named Benefits or EmployeeBenefits.

i. Pluralizing the name of a schema is permissible, as it refers not only to the collection of objects but also the collection of entities within those objects.

ii. A schema named Global will be used for objects that are relevant to all areas of the database. This is not to be used as a "catch-all", but as a location to place objects that are truly global. Peer review should bring to light any object that has been placed in the Global schema inappropriately.

b. Prefixes – Given the segregated nature of the display of objects in the management and development UIs used for Microsoft SQL Server, there is no need to prefix object names. However to ease the management burden from a scripting standpoint and organization from a development standpoint, the following prefixes have been deemed appropriate.

Object Type	**Prefix**
Tables	None
Table Columns	None
Primary Key	PK_
Non Clustered Index	IX_
Unique Index	UX_
~~Clustered Index~~	~~CX_ (Cluster indexing should be handle by the table's primary key. Therefore this prefix should never be used. See IV, b)~~
Check Constraints	CK_
Default Constraints	DF_
Foreign Keys Constraints	FK_
~~Unique Constraints~~	~~UQ_ (This should be avoided. A unique index should be used instead.)~~
Triggers	Should be avoided at all cost.
Views	None
Functions	None
Stored Procedures	None. However application specific prefixes (e.g. Report_ for stored procedures used by SSRS reports or Job_ for stored procedures used in SQL Agent jobs) may be used to assist in organization from a development standpoint.

c. Capitalization Styles – Pascal casing should be the overriding style used throughout SQL objects and code. All object names (tables, columns, stored procedures, constraints, etc) as well as variables in stored procedures, and functions should be in Pascal casing. The one exception is where an abbreviated form of a word must be used.

d. Abbreviated Words – In Microsoft SQL 2000 and moving forward the sysname data type used for object names is derived from nvarchar(128). This means that object names can contain up to one hundred twenty-eight (128) characters. Please consider carefully the use of abbreviations in an object names. Peer review may bring to light a reason to not use an abbreviation, resulting in refactoring work. If an abbreviated form of a word is common to the industry and well known, then it will likely be acceptable. For example, it is very common to use "Id" as a suffix for identity fields in Microsoft SQL Server databases. "Id" is, therefore, an acceptable abbreviation. If an abbreviated form must be used, it should conform to the following guideline:

i. Acronyms – Since an acronym is a word formed from the initial letters of the words in a name or phrase, all characters should be upper cased instead of a mix of casing or all lower case. The exception is when the abbreviation will contain an unimportant word, such as in the "Proof of Concept" example given below.

Expression	**Good Examples**	**Bad Examples**
Three Letter Acronym	TLA	Tla, tla, tLA

Expression	Good Examples	Bad Examples
Standard Query Language	SQL	Sql, sql, sQL
Hyper Text Markup Language	HTML	Html, html, hTML
Extensible Markup Language (It is common, therefore acceptable to use "X" when abbreviating a word that begins with "ex".)	XML	Xml, xml, xML
Proof of Concept (The word "of" is generally included when abbreviating Proof of Concept, but it is an unimportant word in the title.)	PoC, POC	poc, Poc, pOC

ii. Abbreviations – Since an abbreviation is a shortened or contracted form of a word, the first character should be upper cased and the remaining characters lower cased.

Expression	Good Examples	Bad Examples
Mister	Mr	MR, mr, mR
Identifier	Id	ID, id, iD

e. Tables, Views, Table Valued Functions, Cursors, Temporary Tables, and Variable Tables
   i. Tables and Views – Tables represent the instances of an entity. They are "noun" oriented. Therefore, the overriding rule for table names should be that the name represents the "entity" (a single row in the table). Tables and views are objects that define entities. Their names should represent a singular entity that they define.
      1. Singular Form Example – You store all your employee information in a table. Here, "employee" is an entity and all the rows in the employee table represent the instances of the entity "employee". Therefore the table name should be "Employee".
      2. Plural Form Example – You need to store the intranet preferences for an employee. The list of preferences is predefined and seldom changing, and each preference will be represented by a field in the table. Each row in the table will have a one-to-one relationship with a row in the "Employee" table. In this case the table defines the preferences of the employee and should thus be named in plural form, for example "EmployeePreferences".

         (Note: In practice this specific scenario may be better handled with an Entity-Attribute-Value (EAV or attribute) table structure using sql_variant for the value field to account for the dynamic nature of the values. This example has been left in this document for illustration purposes only.)

   ii. Table Valued Functions, Cursors, Temporary Tables, and Variable Tables – Objects of this nature, while taking on the form of tabular data, represent a

subset of entities. Plural form is acceptable and permitted. Peer review should bring to light any areas where an improper form might have been chosen.

f. Constraint Names
    i. Primary keys should be in the PK_TableName format.
    ii. Foreign keys should be in the FK_ForeignKeyTableName_PrimaryKeyTableName format.
    iii. Unique constraints should be in the UQ_TableName_DescriptiveName format.
    iv. Check constraints should be in the CK_TableName_DescriptiveName format.
g. Index and Statistics Names
    i. All single column indexes, regardless of their use, should be in the IX_TableName_ColumnName format.
    ii. All multi column indexes (composite indexes), regardless of their use, should be in the IX_TableName_DescriptiveName format.
    iii. All single column statistics should be in the ST_TableName_ColumnName format.
    iv. All multi column statistics should be in the ST_TableName_DescriptiveName format.
    v. In default configuration of SQL Server statistics are created when an index is created. A statistics created in this manor will have the same name as the index. This is expected and acceptable behavior.
h. Stored Procedures and Function – Stored Procedures and Functions are actionable items and should end with an appropriate verb (e.g. Get, Post, Put, Build, Delete, etc.)

III. Schema Design
a. Schemas will be created to house objects for the sake of organization and segregation of data. In Microsoft SQL Server "dbo" is the default schema. To assist in data organization dbo will not be used for any user defined objects. A schema will be created for each grouping of data, such as an "Employees" schema for all objects related to employees.
b. Many development systems, including .Net, prevent object name/container name collision. For this reason, and many other undisclosed reasons, Schema names should be pluralized. If a need arises that a schema name should not be pluralized, it is acceptable to use a singular name form, with the understanding that this may hinder proper object/class naming in .Net development languages.
c. For each schema created in a database at least two file groups will also be created. One will house data and the other will house indexes. Their names will be “{SchemaName}_Data” and “{SchemaName}_Indexes”, respectfully.
d. If a file stream file group is needed for a schema, it will follow the format of “{SchemaName}_FileStream”.
e. If a memory optimized file group is needed for a schema, it will follow the format of “{SchemaName}_MemoryOptimized”.

IV. Physical File Structure
a. The first file created for a database will follow the format “{DatabaseName}_PRIMARY_01.mdf”. It will be a member of the PRIMARY file group and will be placed on the appropriate drive in the appropriate folder for the specified

SQL instance. No user defined objects will be stored on this file. There should be no reason for more than one file within the PRIMARY file group.

b. The second file in a database will be the transaction log file. The transaction log file will be named "{DatabaseName}_TLOG_01.ldf", and will be placed on the appropriate drive in the appropriate folder for the specified SQL instance. If for performance or storage reasons it is deemed necessary to add additional transaction log files, each additional log file will be named "{DatabaseName}_TLOG_xx.ldf" (where "xx" is the number of the file in sequence).
c. For each schema that is created at least two file groups will be created and named according to the schema name. Within each file group there will be at least one file. The names will follow the format of "{SchemaName}_Data_xx.ndf" and "{SchemaName}_Indexes_xx.ndf", respectfully (where "xx" is the number of the file in sequence).
d. If a file stream file group is created for a schema, it will contain at least one "file" that follows the format of "{DatabaseName}_{SchemaName}_FileStream_xx" (where "xx" is the number of the file in sequence). Note: file stream files are actually folders on the drive. They will not have a file extension.
e. If a memory optimized file group is needed for a schema it will follow the format of "{SchemaName}_MemoryOptimized_xx" (where "xx" is the number of the file in sequence). Note: memory optimized files are actually folders on the drive. They will not have a file extension.

V. Table Design
   a. Normalization – The overriding philosophy in table design is, as the saying goes: Normalize till it hurts; denormalize till it works. A table should define a unique entity and its direct attributes. For example, a table containing a list of persons should only contain information specific to a person (First Name, Middle Name, and Last Name). It should not contain indirect attributes (address[es], email address[es], phone number[s], etc.)
   b. Primary Keys Constraints, Clustered Indexes, and Identity Fields – For the sake of establishing good referential integrity and to assist in index performance analysis, an identity field should be included on every table and be placed in ordinal position one. The identity field should be defined as the Primary Key and Clustered Index. It should be used to create relationships between other tables. The identity field should be created by combining the table name and the suffix "Id", such as PersonId for the identity field on a Person table.
      i. Note: This is in opposition to the idea of using natural keys as the clustered index and/or primary key. Candidate/natural keys may, and should, still be defined as unique indexes. However, candidate keys will not be used for referential integrity.
      ii. If a record does not have a decent single field to use as a surrogate key, and it needs to be directly referenced from application code, the identity field should never be use. Identity fields are database specific and must be handled in an appropriate way as not to break referential integrity. An alternate surrogate key

should be defined that is system or entity specific that can be exposed outside the database. Options for generating such a key are as follows:

1. GUID – If a surrogate key field is defined in this manner, it should be named by combining the table name and the suffix "GUID". Such as ContactGUID for the surrogate GUID field on a Contact table. The ROWGUIDCOL attribute should also be used to denote the field's purpose.
    a. Some methods available for creating a truly global unique identifier include: NEWID() in SQL and System.Guid.NewGuid() in C#. These methods present a difficult issue for SQL in regards to indexes and their use should be weighed very carefully.
    b. Other methods include: NEWSEQUENTIALID() in SQL, and a Comb GUID algorithm (cf. Jimmy Nilsson's Comb GUID algorithm in C# and SQL). These methods overcome the index performance issue by creating sequential identifiers, but uniqueness cannot be guaranteed.
2. Entity Number Generation – Can be implemented SQL side (see http://bradleydonmorris.me/EntityNumberGeneration.html) – If a surrogate key field is defined in this manner, it should be named in such a way as to denote its true purpose. Such as AccountNumber on an Account table.

iii. An example of a misuse of the identity filed would be that of Lease Number. The user community has come to know this identity field as "Lease Number" and it is used throughout the enterprise. This has or will create many issues with integration between databases. **AN IDENTITY FIELD SHOULD NEVER BE EXPOSED OUTSIDE OF THE DATABASE NOR TO THE USER COMMUNITY.**

c. Foreign Keys Constraints – When a table needs to reference another table, a field and a foreign key constraint must be created.
    i. Field – A field on the referencing table will be used to contain the value of the primary key identity field of the referenced table. The field in the foreign table must be named the same as primary key identity field of the referenced table and be defined to use the same data type. It must also not be nullable, as nullability prevents a foreign key constraint from properly constraining the data during Insert, Update, and Delete operations. In the event the foreign key must be null, a many-to-many relationship table will be used so as not to violate the non-nullable key principal in SQL Server. In the case where a table must be referenced twice, such as a UserId stored on one field when a record is created and in a separate field when a record is update, the field names on the referencing table will be prefixed with a descriptive title (no underscore between the descriptive title and the field name). For example, if the referenced table was named "User" and its primary key field UserId, the fields in the referencing table for creating user and modifying user might be appropriately name CreateUserId and ModifyUserId.

ii. Constraint – A foreign key constraint on the referencing table will be used to enforce referential integrity. The constraint will follow the naming convention mentioned earlier in this document. In the case where a table must be referenced twice, such as a UserId stored in one field when a record is created and in a separate field when a record is update, the foreign key names will be suffixed with the descriptive title used for the field names. An underscore will be placed before the suffix. For example, if the referenced table was named "User" and its primary key field UserId, the foreign keys constraint on the referencing table for creating user and modifying user might be appropriately name FK_ReferencingTableName_User_Create and FK_ReferencingTableName_User_Modify. Note the full field name with the "Id" suffix is not used in the foreign key constraint name. This is because foreign keys according to this document are always single field references and that field is always the identity field. The foreign key primarily denotes the relationship between the entities, not the fields.

iii. Index – A "backing" index will be created on the field in the referencing table for every foreign key constraint, until such time as production index analysis deems the index to be a hindrance to performance. If, in the case of hierarchical tables, the foreign key field must also be included in unique index, then the backing index would be redundant and should not be used.

d. Check Constraints – Check constraints, though useful in some cases, must be avoided.
   i. Data Validation Check Constraint – (E.g. [FlagField] & 16 = 16) – Using a check constraint in this manner may indicate a flaw in the data model. A lookup table with proper referential integrity would likely provide better validation and performance and would allow the database model to more appropriately define the data. While bitwise operations are generally extremely fast from a basic programming concept, in SQL this principal does not hold to be true.
   ii. Security Check Constraint – (E.g. IS_MEMBER('WindowsDomain\Group') = 1) – Using a check constraint in this manner moves security from the built-in security mechanisms into a custom, and difficult to document mechanism. This might better be handled through stored procedure or view security.
   iii. Ad Hoc Prevention Check Constraint – (E.g. @@PROCID IS NOT NULL) – Using a check constraint in this manner moves security from the built-in security mechanisms into a custom, and difficult to document mechanism. This might better be handled through stored procedure or view security.

e. Unique Constraints and Unique Indexes
   i. Unique Constraints – As a general rule there is not a major difference between unique constraints and unique indexes. The differences that do exist are only beneficial when working outside the policies outlined in this document. For example, a unique constraint can be referenced by foreign key constraints, whereas a unique index cannot. Since, according to the policies of this document, a foreign key should only reference a primary key, a unique constraint will likely serve no use. A unique constraint will create a unique index

to be used during duplicate key detection, so the performance hit incurred will be the same for a constraint or an index.

ii. Unique Indexes – A unique index should be defined on columns that represent natural keys and surrogate keys where duplicates are not to be allowed (e.g. Social Security Number on an Employee table).

iii. Non-Clustered Indexes - While in development non clustered indexes should be created where appropriate to approximate user activity. As user behavior and usage patterns change over time, non-clustered indexes may be changed, added and/or dropped based on information discovered from routine monitoring, Database Tuning Advisor, etc. When an index change is called for, the change should be developed and tested based on current development and deployment standards and procedures.

VI. Coding Style (Indentation, qualification, bracketization, and aliasing)

Tabs as referenced below should be true tabs, not spaces. This will allow each developer to choose their most comfortable tab size without affecting other developers. It will also make for a small code base.

In code (for stored procedures, views, function, triggers, etc.) all keywords (SELECT, UPDATE, etc.) should have their child keywords (SET, FROM, WHERE, etc.) appear on the next line from the keyword and should be indented one tab over from the parent keyword.

For example, a SELECT statement has many clauses (FROM, WHERE, ORDER BY, etc). All of its clauses should be indented one tab over as in the example shown below. A JOIN clause would also have a child clause, ON, that should be indented one tab over from its parent. In the case where there are object names, criteria, and the like following an attribute, if only one item is needed after the clause then that item should appear on the same line as the clause, separated by a space. If more than one item is needed after the clause, each item should appear on a new line and be indented one over form the parent clause.

All reserved words (SELECT, SET, DELETE, etc.) and built in functions (GETDATE(), CHAR(), CAST(), etc.) should appear in upper case characters. All object names (tables, views, schemas, stored procedures, functions, data types) should appear in brackets ("[", "]"), and should be cased according to how the actual object is cased. This is to aid in differentiating between the two where ambiguity might exist.

All tables, stored procedures, functions, and views must be qualified with their parent schema. If accessing an object in another database on the same server, the object must be qualified with the database name and the schema name. If accessing an object in a database on another server, the object must be qualified with the linked server name, the database and the schema name. It must also be aliased according to the aliasing standards outlined later in this document.

All objects (tables, schemas, views, data type, functions, etc.) should be enclosed in brackets as shown in the example below, and be cased according to the case used in the physical objects.

For example if a field were FirstName in a table, then it should appear as [FirstName] in code. Not as firstname, FIRSTNAME, or any variation thereof.

Aliasing, while good in all cases, is necessary in some cases. When it is necessary to alias a table, view, or function in a SQL statement, the object name should be used as a prefix to the alias name with an underscore separating the two. The alias name should be descriptive enough to easily identify the entities provided by the table or view. For example, when joining a Contact table from a database called Accounting to a contact table in a database called Marketing the aliases might appear as follows:

- [Accounting].[dbo].[Contact] AS [Contact_Accounting]
- [Marketing].[dbo].[Contact] AS [Contact_Marketing]

Cursors should be avoided if possible. However, there are times when a cursor is the most appropriate way of solving a dilemma. If the situation arises where a cursor makes the most sense, it should be declared as FAST_FORWARD or FORWARD_ONLY, and be a READ_ONLY cursor. The cursor name should be descriptive enough to easily identify the result set it is handling. The name should always be preceded with an underscore to facilitate easily identifying it at a cursor. Cursor names in plural form are acceptable as they identify a collection of entities.