# T-SQL Coding Standard

**- for programming the Microsoft Sql Server 2005**

Version 0.5 (work in progress)

September 2006

Author:
Casper.Nielsen@gmail.com

# Table of Contents

# Preface

This standard is meant as an appendix to the **IDesign C# Coding Standard**[i] - for use in our organisation.
For a meaningful introduction into why a standard is necessary see the preface of that document.
**This T-SQL coding standard has in no way been endorsed by the IDesign group or the Microsoft Cooperation and the author is not affiliated with either of these groups.**
The author grants the public rights to freely use alter and redistribute the contents of this T-SQL coding standard. If it is altered in any way this preface should be removed and the naming of the document should be changed.

Searching the web for T-SQL Coding Standards yields a few result, either very formalized documents that gives directions for everything - or documents that focus primarily on optimizations. This document have another fundamental viewpoint: It concerns itself very little with optimization details and very heavily with the structure that programmers interact with, giving more heed to the clarity of the code rather than the speed in which it executes. This is not saying that speed of execution is not a matter of importance, but other documents such as this one on MSDN[ii] have already covered that aspect intensely. Also see this[iii] document for a nice guide to Sql Server 2000 programming.

Writing T-SQL is a highly creative process and real-life code can never be as formalised as the examples in this guide - use it as a style guideline.

We can view a database as having two parts:

- The structure (defined by Data Definition Language) and
- The means to interact with this structure (defined by the Data Manipulation Language).

What this standard proposes is to keep all the DML isolated in the database, enforcing a multilayered approach to programming, never exposing more than a calling interface and the data sets themselves to other layers.

Consistency is the key for good programming: When the programmer does not have to constantly search for names, but instead intuitively knows what objects are called, the task of coding becomes easier and less errorprone.

# 1

# Naming Conventions and Style

1. Use upper case for all T-SQL constructs, except Types:

```
SELECT MAX(MyField) FROM MyTable
```

2. User lower case for all T-SQL Types and usernames:

```
DECLARE @MyVariable int
```

3. Use Camel casing for all UDO's:

```
CREATE TABLE dbo.MyTable
(
   MyField int
)
```

4. Avoid abbreviations and single character names

```
--Correct
DECLARE @Counter int

--Avoid
DECLARE @C int
```

5. UDO naming must confer to the following regular expression (`[a-zA-Z][a-zA-Z0-9]+`) - in short don't use any special or language dependent characters to name objects. Constraints can use the underscore character.

```
--Avoid
CREATE TABLE dbo.[User Information]
```

6. Use the following prefixes when naming objects:
   - `usp` - User Stored Procedures
   - `svf` - Scalar Valued Functions
   - `tvf` - Table Valued Functions
   - `vi` - Views
   - `FK_` - Foreign keys
   - `DF_` - Default constraints
   - `IX_` - Indexes

```
CREATE PROCEDURE dbo.uspMyProcedure AS (...)

CREATE FUNCTION dbo.svfMyFunction
(...)
RETURNS int
AS
(...)

CREATE FUNCTION dbo.tvfMyFunction
(...)
RETURNS TABLE
AS
(...)

CREATE VIEW dbo.viMyView AS (...)
```

7. Name tables in the singular form:

```
--Correct
CREATE TABLE dbo.Address

--Avoid
CREATE TABLE dbo.Addresses
```

8. Tables that map many-to-many relationships should be named by concatenating the names of the tables in question, starting with the most central table's name.

9. Primary and Foreign key fields are postfixed with ID.

```
--Correct
CREATE TABLE dbo.[User]
(
    UserID int NOT NULL,
    AddressID int NOT NULL --Foreign key
)

--Avoid
CREATE TABLE dbo.[User]
(
    UserID int NOT NULL,
    AddressFK int NOT NULL --Fieldname indicates its use as a foreign key
)
```

10. Avoid naming fields in a way that indicates its use as a foreign key.

```
--Avoid
CREATE TABLE dbo.[UserAddress]
(
    UserFK int NOT NULL,
    AddressFK int NOT NULL
)
```

11. Name Stored Procedures as [schema] .[usp][Object][Operation].
When creating Procedures to wrap single INSERT/UPDATE/DELETE statements, *operation* should be Insert, Update and Delete respectively.

12. Always assign schema to UDO's when defining.

```
--Correct
CREATE TABLE dbo.MyTable (...)

--Avoid
CREATE TABLE MyTable (...)
```

13. Always include the schema when referencing an object:

```
--Correct
SELECT * FROM dbo.MyTable (...)

--Avoid
SELECT * FROM MyTable (...)
```

14. Properly arrange statements: Either use one-liners without indentation or multi-liners with indentation. Don't mix the two.

```
--Correct one-liner
SELECT * FROM dbo.MyTable

--Correct multi-liner
SELECT *
FROM   dbo.MyTable
WHERE  MyTableID IN
(
   SELECT MyForeignTableID
   FROM   dbo.MyForeignTable
)
AND    MyColumn > 1
```

```
--Avoid
SELECT *
FROM dbo.MyTable --Missing indentation
WHERE   MyField > 1 AND --Misplaced AND
        Myfield < 3

--Avoid mixing multiline and singleline expressions
SELECT * FROM dbo.MyTable
WHERE   MyField > 1
```

15. When creating local scope always indent:

```
BEGIN
    (...)
END
```

16. When using parenthesis around multi-line expressions, always put them on their own lines:

```
--Correct
RETURN
(
    (...)
)

--Avoid
RETURN (
    (...) )
```

17. When using IF statements, always BEGIN new scope:

```
--Correct
IF(1 > 2)
BEGIN
    (...)
END
ELSE
BEGIN
    (...)
END

--Avoid
IF(1 > 2)
    (...)
ELSE
    (...)
```

18. Always create scope when defining Procedures and multi statement Functions:

```
--Correct
CREATE PROCEDURE dbo.uspMyProcedure
AS
BEGIN
    (...)
END

--Avoid
CREATE PROCEDURE dbo.uspMyProcedure
AS
    (...)
```

19. When joining always identify all columns with aliases and always alias using the AS keyword.

```
--Correct
```

```
SELECT  U.Surname,
        A.Street
FROM    dbo.[User] AS U
JOIN    dbo.Address AS A ON U.AddressID = A.AddressID

--Avoid
SELECT  U.Surname,
        Street --Missing alias
FROM    Users U --Missing AS
JOIN    dbo.Address ON U.AddressID = dbo.Address.AddressID --Missing Alias
```

20. Avoid joining in the where clause, instead use ANSI syntax for joining. Include the reference key last:

```
--Correct
SELECT  U.Surname,
        A.Street
FROM    dbo.[User] AS U
JOIN    dbo.Address AS A ON A.AddressID = U.AddressID

--Avoid
SELECT  U.Surname,
        A.Street
FROM    dbo.[User] AS U,
        dbo.Address AS A
WHERE   U.AddressID = A.AddressID --Joins in the WHERE clause
```

21. Avoid using RIGHT joins – rewrite to LEFT joins.

22. When doing INNER JOIN's, avoid using the INNER keyword:

```
--Correct
SELECT  U.Surname,
        A.Street
FROM    dbo.[User] AS U
JOIN    dbo.Address AS A ON A.AddressID = U.AddressID

--Avoid
SELECT  U.Surname,
        A.Street
FROM    dbo.[User] AS U
INNER JOIN dbo.Address AS A ON A.AddressID = U.AddressID
```

23. When defining Procedures and Functions, include a commented Test Harness. Declare used variables for usage in testing.
In Procedures include a transaction which is properly rolled back after checking values.
Skip this step if the Procedure is a simple INSERT/UPDATE/DELETE with no logic besides that.

```
--Correct
CREATE FUNCTION dbo.tvfMyFunction
(
    @MyParameter int
)
AS
/* TEST HARNESS
    DECLARE @MyParameter int
    SET @MyParameter = 1
    SELECT * FROM dbo.tvfMyFunction(@MyParameter)
*/
(...)

--Correct
CREATE PROCEDURE dbo.uspMyProcedure
```

```
(
    @MyParameter int
)
AS
/* TEST HARNESS
    DECLARE @MyParameter int
    SET @MyParameter = 1
    BEGIN TRAN
    SELECT * FROM dbo.MyTable –-MyTable before operation
    EXEC dbo.uspMyProcedure(@MyParameter)
    SELECT * FROM dbo.MyTable –-MyTable after operation
    ROLLBACK TRAN
*/
(...)

--Avoid
/* TEST HARNESS
    SELECT * FROM dbo.tvfMyFunction(1) –-argument not declared
*/
```

24. If you use designers to generate DML – reformat it using the design styles defined here. In effect it is disallowed to check in DML from designers into a project repository.
    Using designers to generate DDL however is allowed and encouraged.

25. Use comments only to illuminate things that are not obvious from reading the code.

# 2 Best Practices

## 2.1 Database Design and Architecture

1. Decide between using uniqueidentifier's (guid) or integers as artificial Primary keys. Generally use uniqueidentifier's for replicated scenarios.

2. Introduce a artificial Primary key on Tables, unless the table fall into one of the following categories:
   - Many-to-many relationship
   - Staging table
   - Imported External tables with reoccurring importing

3. Always have unique indexes or primary keys on all tables without exception – never leave the integrety up to the application.

4. If you have enumerations in another layer that are persisted onto the database, map these in a table and ensure consistency by constraints.

5. Decide whether to use abbreviations or integers as Primary keys on Enumeration mapping tables and stick with this choice throughout the design. Usage of these keys in others tables should of course is constrained.

```
--Either
CREATE TABLE dbo.ProductStatusCode
(
    ProductStatusCodeID int,
    Description nvarchar(50)
)

--Or
CREATE TABLE dbo.ProductStatusCode
(
    ProductStatusCode char(2),
    Description nvarchar(50)
)
```

6. If you allow NULL in a field, make sure it has a meaning apart from empty and zero.

7. Never use the TEXT or NTEXT types instead use the MAX types.

```
--Correct
CREATE TABLE dbo.MyTable
(
    MyTextField varchar(MAX),
    MyUnicodeTextField nvarchar(MAX)
)

--Avoid
CREATE TABLE dbo.MyTable
(
    MyTextField text,
    MyUnicodeTextField ntext
)
```

8. Be cautious when using the blob type for storing files. Weigh the structure benefit with the added database load before deciding which approach to use.
   Examine where the file logically belongs: If it have significance to the database, store it there, otherwise store it

on the file system.
If you choose the file system as the data store for files, remember to backup the file store simultaneously with the database to counter synchronization issues.

9. Implement the database so it will keep itself consistent with the model it was designed for. This involves using referential integrity checks extensively. Implement these checks using constraints. As a minimum ID's with foreign keys and fields that have a constant limit should be constrained.

10. Avoid triggers if possible. Using Procedures as access points should decrease the need for triggers considerably.

11. Avoid cascading updates and deletes. Instead use Procedures as access points to these operations.

12. Unless a management decision have been taken to allow the usage of Table Adapters in the other .Net layers, the Table Adapters should exist only in the data access-layer. Typed Datasets can be used in all layers, if they themselves do not include Table Adapters. Keep in mind that even when using Table Adapters T-SQL code should be kept on the database, i.e. the adapter only wraps stored procedures, functions and views.

13. Be careful when using collations - they handles differently. Find the most general collation and use this is the database default, for most European language scenarios this will be the SQL_Latin1_General_CP1_CI_AS (Case insensitive/Accent sensitive). When the needs arise to use a specialized collation, do so on the specific field.

## 2.2 Coding

1. Use Procedures for functionality with side-effects - Functions for everything else requiring parameters.
   Sole exception is if the Function requires a specific database state to run correctly - in this case write it as a Procedure.

2. Avoid the use of GOTO. They are basically not needed after the introduction of TRY/CATCH and often indicate sloppy coding.

```
--Avoid
GOTO MyHackLocation
```

3. Avoid using the @@ERROR function to handle exceptional states. Instead use the TRY/CATCH construct.

4. Avoid using DTS packages – use SSIS instead.

5. Avoid T-SQL code anywhere but in Views, Procedures, Functions and Batches. This includes SQL Agent Jobs, DTS, SSIS and .Net code. Only use the following T-SQL in other layers:

```
SELECT * FROM dbo.tvfMyFunction(someParameter)

SELECT dbo.svfMyFunction(someParameter)

EXEC dbo.uspMyProcedure(someParameter)
```

6. Avoid using batches (scripts) for recurring tasks, except creating and initializing the database. Use SSIS for transferring batches of data.

7. If you feel like using a CURSOR to solve a particular problem, first consider finding a set-based solution. Most of the times the problem can be solved using CASE.
   If this approach is not viable try solving it with a WHILE loop on the primary key.

8. Use Table-valued variables or the new WITH construct instead of creating Temporary Tables. If you need advanced indexing functionality on the Temporary Table, create the Table as a physical entity on the database used, prefixing it with Temporary.

9. Always individually name fields in INSERT and UPDATE statements. Never use the * operator in such statements.

```
--Correct
INSERT INTO dbo.[User] (FirstName, LastName)
VALUES (@FirstName, @LastName)

--Avoid
INSERT INTO dbo.[User]
VALUES (@FirstName, @LastName)
```

10. Set the NOCOUNT state as the first statement in all Procedures where you don't specifically need the count returned.

```
--Correct
CREATE PROCEDURE dbo.uspMyProcedure (...)
AS
BEGIN
    SET NOCOUNT ON
    (...)
```

11. Avoid using dynamic SQL in Procedures.

12. Avoid using PRINT statements in Procedures; instead use the debug functionality of the Visual Studio.

13. Try avoiding prefixing any UDO beyond what is described here. If you need to further group objects, consider using a schema different from dbo to create the objects.

## 2.3 Transactions

1. Always set XACT_ABORT.
   If you have more than one database modifying action in a Procedure, decide if it needs to run atomically (most probably it does) – if so set XACT_ABORT ON and encapsulate all statements in a transaction. If it is not required to run atomically, explicitly set XACT_ABORT OFF.

```
--Correct
CREATE PROCEDURE dbo.uspMyProcedure (...)
AS
BEGIN
    SET XACT_ABORT ON
    BEGIN TRAN
    (...) --Transaction will be rolled back if anything fails here
    COMMIT TRAN
```

## 2.4 Queries and Optimizations

1. Using SELECT * will slightly reduce performance:
   - The Sql Engine will need to lookup the fields involved.
   - In many cases more IO than needed is performed.

2. It is important to make sure that when running a query, the order of tables accessed is vital for speed. Important is also that if a composite index is referred to in a query, the keys must be referred to in the proper order and in the proper sort order. If content is looked up in reverse, use DESC on a column when creating indexes. The driving table in a query (in the FROM clause) should be the table to return the fewest rows seen in percentage.

[i] http://www.idesign.net/idesign/download/IDesign%20CSharp%20Coding%20Standard.zip

[ii] http://msdn.microsoft.com/sql/default.aspx?pull=/library/en-us/dnsqlpro04/html/sp04l9.asp

[iii] http://www.sql-server-performance.com/vk_sql_best_practices.asp