|  |  |
| --- | --- |
|  | T-SQL code standard |
| C:\Users\speterso\AppData\Local\Microsoft\Windows\INetCache\Content.Word\BobbyTables.jpg | |
| 6/18/2015 | Standards for 1-800 CONTACTS T-SQL code |
|  | The standard helps to enforce best practices and avoid pitfalls, and makes knowledge dissemination across the team easier. The coding standard presented next contains best practices, dos and don’ts, pitfalls, guidelines and recommendations, as well as naming conventions and styles. Writing T-SQL is a highly creative process and real-life code can never be as formalized as the examples in this guide - use it as a style guideline. |

T-SQL code standard

Standards for 1800 CONTaCTs T-SQL code

Table of Contents

[T-SQL code standard 3](#_Toc422391065)

[Standards for 1-800 CONTaCTs T-SQL code 3](#_Toc422391066)

[What is a database? 3](#_Toc422391067)

[We can view a database as having two parts: 3](#_Toc422391068)

[Naming Conventions and Style 3](#_Toc422391069)

[Naming Conventions 3](#_Toc422391070)

[Coding Style 5](#_Toc422391071)

[Best Practices 7](#_Toc422391072)

[Database Design and Architecture 7](#_Toc422391073)

[Transactions 10](#_Toc422391074)

[Queries and Optimizations 10](#_Toc422391075)

[Release Spreadsheet – Example upon request 10](#_Toc422391076)

[Database T-SQL Standards check list: 11](#_Toc422391077)

T-SQL code standard

Standards for 1-800 CONTaCTs T-SQL code

# What is a database?

## We can view a database as having two parts:

* The structure (defined by Data Definition Language [DDL]) and
* The means to interact with this structure (defined by the Data Manipulation Language [DML]).

What this standard proposes is to keep all the DML (Data Manipulation Language) 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 error prone.

# Naming Conventions and Style

## Naming Conventions

* Use upper case for all T-SQL constructs:
  + SELECT MAX(MyField) FROM MyTable
* User upper case for all T-SQL Types and usernames:
  + DECLARE @MyVariable INT
* Use Camel casing for all UDO’s (User Defined Objects):  
  + CREATE TABLE dbo.MyTable  
    (  
     MyField INT  
    )
* Avoid abbreviations and single character names
  + Correct
    - DECLARE @Counter INT
  + Avoid
    - DECLARE @C INT
* UDO (User Defined Object) 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 or spaces to name objects.
  + Correct
    - CREATE TABLE dbo.UserInformation
  + Avoid
    - CREATE TABLE dbo.[User Information]
* Use the following prefixes when naming objects:
  + usp - User Stored Procedures
  + svf - Scalar Valued Functions
  + tvf - Table Valued Functions
  + vw – Views
  + FK\_ - Foreign keys
  + DF\_ - Default constraints
  + IX\_ - Indexes
  + CREATE PROCEDURE dbo.usp\_MyProcedure AS
  + CREATE FUNCTION dbo.svf\_MyFunction
  + CREATE FUNCTION dbo.tvf\_MyFunction
  + CREATE VIEW dbo.vw\_MyView AS (...)
* Name tables in the singular form:
  + Correct
    - CREATE TABLE dbo.Address
  + Avoid
    - CREATE TABLE dbo.Addresses
* 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.
* Primary and Foreign key fields are post fixed with ID.
  + Correct
    - CREATE TABLE dbo.[User]  
      (  
       UserID INT NOT NULL,  
       AddressID INT NOT NULL   
      )
  + Avoid
    - CREATE TABLE dbo.[User]  
      (  
       UserID INT NOT NULL,  
       AddressFK INT NOT NULL   
      )
* Avoid naming fields in a way that indicates its use as a foreign key.
  + Correct
    - CREATE TABLE dbo.[UserAddress]  
      (  
       UserID INT NOT NULL,  
       AddressID INT NOT NULL  
      )
  + Avoid
    - CREATE TABLE dbo.[UserAddress]  
      (  
       UserFK INT NOT NULL,  
       AddressFK INT NOT NULL  
      )
* Stored procedure naming must confer to the following regular expression (?<UserDefinedObject>(?:usp|svf|tvf|vw))\_(?<AppName>[A-Za-z]{2,})\_(?<TableList>(?:Multi|\w{2,64}))\_(?<TableFunction>Insert|Update|Delete|Select|Merge|Custom)
* Fully qualify Stored Procedures as [schema].<Validated Stored Procedure Name>.
  + Correct
    - CREATE PROCEDURE dbo.usp\_OMS\_OrderHD\_Insert
    - CREATE PROCEDURE OMS.usp\_OMS\_OrderHD\_Insert
      * If it is in a schema name the name can be omitted from the sproc name
        + I.E. OMS.usp\_OrderHD\_Insert
  + Avoid
    - CREATE PROCEDURE dbo.ap\_Order\_Insert
    - CREATE PROCEDURE usp\_OMS\_OrderHD\_Insert
* When creating Procedures to wrap single INSERT/UPDATE/DELETE statements, operation should be Insert, Update and Delete respectively.
* Always assign schema to UDO’s when defining.
  + Correct
    - CREATE TABLE dbo.MyTable (...)
  + Avoid
    - CREATE TABLE MyTable (...)
* Always include the schema when referencing an object:
  + Correct
    - SELECT ColumnName’s FROM dbo.MyTable (...)
  + Avoid
    - SELECT ColumnName’s FROM MyTable (...)

## Coding Style

* Properly arrange statements: Either use one-liners without indentation or multi-liners with indentation. Don’t mix the two.
  + Correct one-liner
    - SELECT ColumnName’s FROM dbo.MyTable
  + Correct multi-liner
    - SELECT ColumnName’s  
      FROM dbo.MyTable  
      WHERE MyTableID IN  
      (  
       SELECT ColumnName’s  
       FROM dbo.MyForeignTable  
      )  
      AND MyColumn > 1
  + Avoid mixing multi-line and single-line expressions
    - SELECT ColumnName’s FROM dbo.MyTable  
      WHERE MyField > 1
* When creating local scope always indent:  
   BEGIN  
   (...)  
   END
* When using IF statements, always BEGIN new scope:
  + Correct
    - IF(1 > 2)  
      BEGIN  
       (...)  
      END  
      ELSE  
      BEGIN  
       (...)  
      END
  + Avoid
    - IF(1 > 2)  
       (...)  
      ELSE  
       (...)
* Always create scope when defining Procedures and multi statement Functions:
  + Correct
    - CREATE PROCEDURE dbo.uspMyProcedure  
      AS  
      BEGIN  
       (...)  
      END
  + Avoid
    - CREATE PROCEDURE dbo.uspMyProcedure  
      AS  
      (...)
* When joining always identify all columns with aliases and always alias using the AS keyword. Never mix aliases with non-aliases for table names. If the table name is less than 15 characters long do not use aliases:
  + Correct
    - SELECT   
      User.Surname,  
      Address.Street  
      FROM dbo.[User]   
      JOIN dbo.Address ON User.AddressID = Address.AddressID
  + Avoid
    - SELECT   
      U.Surname,  
      Street –-Missing alias  
      FROM Users U –-Missing AS  
      JOIN dbo.Address ON U.AddressID = dbo.Address.AddressID –-Missing Alias
* 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
* Avoid using RIGHT joins – rewrite to LEFT joins.
* 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.
* Use comments only to illuminate things that are not obvious from reading the code.
  + Comment within TFS via the history.

# Best Practices

## Database Design and Architecture

* When creating a table decide between using integers with Identity or Sequences as artificial Primary keys. Avoid uniqueidentifier’s as the Primary key.
* 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
* Always have unique indexes or primary keys on all tables without exception – never leave the integrity up to the application.
* If you have enumerations in another layer that are persisted into the database, map these in a table and ensure consistency by constraints.
* 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 be constrained.
  + Either
    - CREATE TABLE dbo.ProductStatusCode  
      (  
       ProductStatusCodeID INT,  
       Description NVARCHAR(50)  
      )
    - Or  
      CREATE TABLE dbo.ProductStatusCode  
      (  
       ProductStatusCode CHAR(2),  
       Description NVARCHAR(50)  
      )
* If you allow NULL in a field, make sure it has a meaning apart from empty and zero.
* Image types are no longer allowed (not supported in the SQL Server engine).
  + VARBINARY should be used in its place.
* TEXT or NTEXT are no longer allowed try to use the MAX types (if no other option is available).
  + The use of VARCHAR(MAX) or VARBINARY(MAX) requires approval by a DBA or a Data Architect. Try to avoid this by designing a parent child system or keep the precision down to 4000 (VARCHAR(4000), VARBINARY(4000)).
  + Correct
    - CREATE TABLE dbo.MyTable  
      (  
       MyTextField VARCHAR(4000),  
       MyUnicodeTextField NVARCHAR(4000),

MyImage VARBINARY(4000)  
)

* + - Not Allowed
      * CREATE TABLE dbo.MyTable  
        (  
         MyTextField TEXT,  
         MyUnicodeTextField NTEXT  
        )
* SQLVarient Types are strictly forbidden and your table design will be rejected.
* Be cautious when using the BLOB (Binary Large Object [e.g. FileStream or FileTable) type for storing files. Weigh the structure benefit with the added database load before deciding which approach to use.
  + PLEASE CONSULT A DBA or a Data Architect BEFORE USING
* 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.
* Avoid triggers if possible. Using Procedures as access points should decrease the need for triggers considerably.
  + If trigger(s) are used for debug/History table purposes define the length of time the trigger will be used.
    - The data may not be current or active (don’t use in your applications)
* Avoid cascading updates and deletes. Instead use Procedures as access points to these operations.
* 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). This includes the use of Entity Framework.
* Be careful when using collations - they handle differently. Find the most general collation and use this as 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.
  + PLEASE CONSULT A DBA BEFORE SETTING UP A DIFFERENT COLLATION

Coding

* Use Procedures for functionality with side-effects - Functions for everything else requiring parameters. The sole exception is if the Function requires a specific database state to run correctly - in this case write it as a Procedure.
  + Scalar Functions will be questioned and should be written as a Table Function
  + This will increase PERFOMANCE and you want this in your application
* Use of GOTO is prohibited. They are basically not needed after the introduction of TRY/CATCH and often indicate sloppy coding.
  + Avoid
    - GOTO MyHackLocation
* Avoid using the @@ERROR function to handle exceptional states. Instead use the TRY/CATCH construct.
* Use of SSIS (SQL Server Integration Services) requires assistance by the Data Integration Team.
* Avoid T-SQL code anywhere but in Views, Procedures and Functions. This includes SQL Agent Jobs, SSIS, SSRS and .Net code. Only use the following T-SQL in other layers:
  + SELECT <col list> FROM dbo.tvf\_MyFunction(some Parameter)
  + SELECT dbo.svf\_MyFunction(some Parameter)
  + EXEC dbo.usp\_MyProcedure(some Parameter)
* Avoid using batches (scripts) for recurring tasks, except creating and initializing the database. Use SSIS for transferring batches of data.
* 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 no options are available please see a DBA
* Use the WITH construct (CTE: Common Table Expression) instead of creating Temporary Tables where available.
* If you need to use a Temporary Table you can add some indexing on it if needed.
* 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)
* 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  
      (...)
* Please set SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED; after the NOCOUNT to avoid locking tables in the select. This produces a read of the data at the current state “dirty read” unless you need to have data in a committed state this is our standard, please discuss any alternative with a DBA or Data Architect.
* Avoid using dynamic SQL in Procedures, it opens us up to SQL injection attacks and we don’t like that – this can be used if supervised by a DBA.
* Avoid using PRINT statements in Procedures; instead use the debug functionality of Visual Studio.
* 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.

## Transactions

* Always set XACT\_ABORT.  
  If you have more than one database modifying action in a Procedure, decide if it needs to run atomically (most likely it does) – if so set XACT\_ABORT ON and encapsulate all statements in an explicit transaction. If it is not required to run atomically, explicitly set XACT\_ABORT OFF. This ends execution immediately and caused the statement to throw and exception and automatically rollback explicit or implicit transactions (see a DBA for more information)
  + Correct
    - CREATE PROCEDURE dbo.usp\_MyProcedure (...)  
      AS  
      BEGIN  
      SET XACT\_ABORT ON  
      BEGIN TRAN  
      (...) –-Transaction will be rolled back if anything fails here  
      COMMIT TRAN

## Queries and Optimizations

* Using SELECT \* will slightly reduce performance, and is not allowed in production T-SQL Code:
  + The SQL Engine will need to look up the fields involved.
  + In many cases more IO than needed is performed.
* The DBA team will occasionally optimize queries in production and notify you of the changes to be added via the release process at your soonest convenience or release date.

## Release Spreadsheet – Example upon request

* The database spreadsheet requires the following items.
  + Run Order (in needed)
  + Staging Server Name or IP Address
  + Production Server Name
  + Database Name
  + File or Object that is changing (name of script or object name)
  + Access User (security object)
  + Notes
  + Developer(s) – please specify at least one full name
  + Date
* If you are specifying Permissions please include the following in a new section
  + Staging Server Name or IP Address
  + Production Server Name
  + Database Name
  + Permission – object name and type of permission (I.E. GRANT SELECT ON <object> TO <role>)
  + Notes
  + Developer(s) – please specify at least one full name
  + Date

## Database T-SQL Standards check list:

Here is a list of the most common staging release request issues:

* Please discuss all new or alter of table structures with the DBA team and/or Monte Hilton (Architecture) for approval before development.
* Triggers on core tables need to be discussed with the DBA team for approval before development/deployment.
* Existing stored procedures that are altered will need to be brought up to the T-SQL Standards to be released.
* Altering table(s) scripts should not have the data change and table change in the same script.

All Stored Procedures should have the following:

* USE [DatabaseName] clause at the top
  + All scripts/procedures/functions/TVPs should have this!
* BEGIN at the beginning of the procedure and END at the end of the procedure.
* SET XACT\_ABORT, NOCOUNT ON;  
  SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED;
* All objects (tables, views, functions, etc.) must be schema qualified even in in dbo schema.
  + No Database reference should be made within the stored procedure outside of the USE [Database] unless it is an object that lives outside of the database the stored procedure is in.
* All Updates should have a where clause as to avoid updating entire table.
* Notes should be included after the END (if you want to add comments)