Coding / Naming Standards

SQL Server

DB Gurus

Document History

This section describes the changes of the document by version number.

Version	Date	Initials	Description of change
1	27 Aug 2008	JB	Initial Draft
2	5 Jan 2009	JB	Added clarifications
3	17 Sep 2010	JB	TOC and updated
4	14 May 2019	JB	Updated with changes since 2010 and added a few new sections
5	24 July 2019	JB	Improved the Indent section

Phone (+61) 02 8001 6353 www.dbgurus.com.au

Table of contents

Introduction

This section orientates the user (context or business problem)

It is important to have standards when developing new software. This document covers the standards used by DB Gurus in MS SQL Server.

Note that DB Gurus has separate documents for:

- MS Access Coding Guidelines
- C# Coding Guidelines
- Version Numbering

And so on. If you need any of these please send an email to info@dbgurus.com.au

Capitalisation

- Reserved words (other than variable types) should be all capital letters

```
SELECT, CREATE, AND, IF, NULL, COUNT(), PRIMARY, GETDATE()
```

- Variable types should be in lower case

```
bit, int, varchar, date, datetime
```

- Table and variable names should be in mixed case

```
[Person], [Address], @Counter, @nNumberOfClaims
```

Brackets

- It is a suggested you use square brackets around table and column names

```
[Person], [Address]
```

- Brackets should only be used to make it clearer how the logic works and to avoid BODMAS

```
WHERE (a=1) AND (b=2)
WHERE ((a=1) AND (b=1))
       OR ((a=0) AND (b=0))
```

Indents and Line Breaks

If a command is short it can be put on one line:

```
SELECT * FROM [Person] WHERE [PersonID] = @PersonID
```

If a command is longer then it should be broken up so the reader can see it all and indent the subsequent lines:

```
SELECT [Name], [Address1], [Address2], [Suburb], [State], [Postcode],
       [Country], [Phone], [Email]
```

Except where the command is small put the keyword at the start of each line:

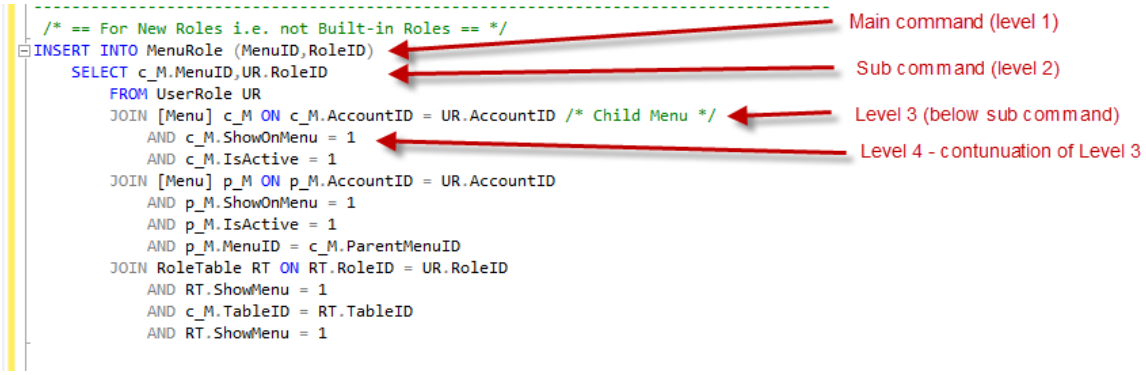
```
SELECT [Name], [Address1], [Address2], [Suburb], [State], [Postcode],
              [Country], [Phone], [Email]
       FROM [Person]
       WHERE [PersonType] = 1
```

Indent the subsequent lines so you can see the primary command:

```
INSERT INTO @PersonTable
       SELECT [Name], [Address1], [Address2], [Suburb], [State], [Postcode],
              [Country], [Phone], [Email]
       FROM [Person]
       WHERE [PersonType] = 1
```

(in the example above we are doing an insert. The SELECT is part of the INSERT)

Another example:



Tables

1. Table names should always describe the table's content / purpose. Some standard ones we user are SystemOption and LookupData

   Reason: Makes it easier for subsequent programmers to follow

2. Tables names should always be singular i.e. we have **Person** but not *People*, we have **Vehicle** not *Vehicles*.

   Reason: It can become confusing deciding when to use the singular or the plural plus when using singular the PK always sounds right "PersonID" (and as it is consistent can be guessed or scripted).

3. The first column of each table is normally:
   a. an identity column (1,1)
   b. the primary key
   c. called the same name as the table (without the prefix) plus 'ID' at the end. For example the Country table will have a primary key CountryID

```
CREATE TABLE Country
(
      CountryID int IDENTITY(1,1) NOT NULL PRIMARY KEY,
      CountryName varchar(100) NOT NULL,
      DateCreated datetime NOT NULL DEFAULT (GETDATE()),
      DateUpdated datetime NOT NULL
)
```

   Reason: Predictable and easy to understand.

   Notes: User the same capitalization as SQL Server generates i.e. NULL is upper-case and datetime is lower case

4. Add DateCreated column with a default value of GETDATE() on commonly added data so that we can see when the rows are created. See Example 1 and 2.

   Reason: Allows us to see then a row is added.

5. Add DateUpdated so that we can see when the rows are updated. See Example 1 and 2. We then use a stored procedure called **dbgGenerateUpdateDateUpdatedTriggers** to create the triggers

   Reason: Provides some basic audit info.

```
CREATE TABLE State
(
	StateID int IDENTITY(1,1) NOT NULL PRIMARY KEY,
	StateName nvarchar(50) NOT NULL UNIQUE,
	CountryID int NOT NULL REFERENCES Country,
	DateCreated datetime NOT NULL DEFAULT GETDATE(),
	DateUpdated datetime NOT NULL
)
```

6. Preferred Data Types:

Data Type	**Usage**
bit	Checkboxes. Often use fieldname like "IsActive" or "IsMale"
int	Keys (primary/foreign keys). Normally ends in ID.
n/varchar Note if the client may use Chinese or Arabic characters use nvarchar, otherwise use varchar	Text boxes e.g. FirstName Typical values: • varchar(50) • varchar(100) • varchar(200) • varchar(MAX)
n/text I do not see the point of n/text now	Multi-line text boxes e.g. Notes

we have n/varchar(max)	
datetime	Dates e.g. StartDate
decimal (18,5)	Numbers, quantities and money. e.g. TotalCost

Note the we normally use SQL Server 2005 compatible datatypes unless there is a compelling reason not to.

7. Use NOT NULL when a column is mandatory.

Referential Integrity

Use referential integrity extensively because

- it makes coding easier – particularly in web development as the code generator can infer more information
- it ensures that only correct data is entered

```
CREATE TABLE Doctor
(
      DoctorID int IDENTITY(1,1) NOT NULL PRIMARY KEY,
      DivisionID int NOT NULL
      -- and so on
)
GO

ALTER TABLE Doctor ADD CONSTRAINT Doctor_Division
      FOREIGN KEY (DivisionID)
      REFERENCES llwDivision (DivisionID)
GO
```

1. Always add the constraint as a separate step.

   Reason: Allow you to provide more detail such as the name.

2. Name the FK constraint FromTable_ToTable

   Reason: Easy to understand/read

   Note: When referring to LookupData use the format FromTable_ToTable_ColumnName to differentiate.

3. Use Cascade deletes with caution / not at all

   Reason: Users find it hard to understand where there data has gone.

4. When using referential integrity without cascade delete you can often have the problem of not being able to delete the parent because the children still exist on the database. An alternative to deleting data is to have a IsActive bit DEFAULT(1) column and always filter on IsActive = 1. On delete just set IsActive=0 to do a "virtual delete".

Indexes and Unique Constraints

```
CREATE TABLE State
(
	StateID int IDENTITY(1,1) NOT NULL PRIMARY KEY,
	StateName nvarchar(50) NOT NULL UNIQUE,
	CountryID int NOT NULL REFERENCES Country,
	DateCreated datetime NOT NULL DEFAULT GETDATE(),
	DateUpdated datetime NOT NULL
)
```

5. Unique constraints. Note above that the primary key is StateID and there is a unique constraint on StateName to ensure that duplicates cannot be added. This is the preferred method rather than setting StateName as the PK.

   Reason: Follows other standards and easier for foreign keys. Neater.

6. Indexes should be added to other columns that will be queried a lot. Safest course of action is to index individual columns and let SQL Server optimise.

   Reason: System performance

7. Use NOT NULL when a column is mandatory.

   Reason: Tools can infer more – less manual coding. More robust.

Scripts and Deployment

1. During development the initial version can be developed using SQL Server Management Studio and either deployed using a backup/restore or using a SQL Server's script generation.

2. After the initial deployment to the test server all subsequent changes should be scripted so that the tester and/or the client does not lose their data.
    a. Save it as **[Number].sql**
    b. Save it into the SVN repository
    c. Ensure that the script can be run multiple times – see below.

    Reason: Makes deployment easy. Does not lose data.

3. Note it is good practice to put in print statements to show user what is happening as shown in the example above.

Running Scripts Multiple Times

You should always ensure that scripts can be run again and again – you will want to run them again and again during development to ensure that the database matches your code and the script has everything.

Also you do not want problems to occur if someone accidentally runs the script a 2nd time.

In 99% of case this can be done using the following techniques:

Tables

For tables this can be done in 2 ways:

a. Have a series of DROP TABLE statements at the top (put them in the reverse order of the CREATE TABLE statements).

```
IF EXISTS(SELECT * FROM INFORMATION_SCHEMA.TABLES WHERE TABLE_NAME
LIKE 'poState') DROP TABLE poState
IF EXISTS(SELECT * FROM INFORMATION_SCHEMA.TABLES WHERE TABLE_NAME
LIKE 'poCountry') DROP TABLE poCountry
```

Useful if you want to drop and recreate but dangerous for clients to use.

b. **Recommended**: Check the information schema to see if it exists and only add it if it does not.

```
IF NOT EXISTS(SELECT * FROM INFORMATION_SCHEMA.TABLES WHERE
TABLE_NAME = 'poState')
BEGIN
      -- Create table poState
END
```

Columns

```
IF NOT EXISTS(SELECT * FROM INFORMATION_SCHEMA.COLUMNS WHERE TABLE_NAME
= 'poReportTemplate' AND COLUMN_NAME = 'ReportName')
BEGIN
      -- Add column
END
```

Constraint

For a constraint we have a special stored procedure:

```
EXEC dbgDropConstraint
      @sTableName = 'poSupplier',
      @sColumnName = 'CountryID'
```

Adding data

```
IF EXISTS(SELECT * FROM poContent WHERE ContentKey = 'ConfirmEmail')
BEGIN
	PRINT '-------------------------------------------------------'
	PRINT '—Adding Contentkey ConfirmEmail '
	PRINT '-------------------------------------------------------'
	-- Add data
END
```

Note that we set identity insert on when we are using hard-coded values in the program for identity columns.

```
	SET IDENTITY_INSERT [poState] ON
	INSERT INTO [poState] ([StateID],[StateName])VALUES(1,'ACT')
	-- and so on
	SET IDENTITY_INSERT [poState] OFF
```

Views

- Name views with a v to differentiate them from tables.
- Check if exists before creating

```
IF EXISTS(SELECT * FROM INFORMATION_SCHEMA.VIEWS WHERE TABLE_NAME = 'vPersonFullName')
	DROP VIEW [dbo].[vPersonFullNameQuery]
GO
CREATE VIEW [dbo].[vPersonFullNameQuery]
AS
SELECT --
```

Stored Procedures

1. Stored procedures and functions are saved in a separate file Setup Code XXX.sql and also saved in the \SQL folder below the project.

   Reason: The structural changes should be made first, then the code run.

2. Functions are above stored procedures.

   Reason: The stored procedures may make use of the functions.

3. Example Function:

```
/* ---------------------------------------------------------------------*/
IF EXISTS (SELECT * FROM INFORMATION_SCHEMA.ROUTINES
   WHERE ROUTINE_NAME = 'fnUserHasRole' AND ROUTINE_TYPE = 'FUNCTION')
      DROP FUNCTION fnUserHasRole
GO
/* ---------------------------------------------------------------------*/
CREATE FUNCTION fnUserHasRole
(
      @nUserID int,
      @nRole int
)
RETURNS bit
AS
/*
      UNIT TESTING
      ============
      SELECT dbo.fnUserHasRole(1, 3)
*/

BEGIN
      DECLARE @bResult bit
      IF EXISTS(SELECT * FROM poUserRole
         WHERE UserID = @nUserID AND RoleID = @nRole)
            SET @bResult = 1
      ELSE
            SET @bResult = 0
      RETURN @bResult
END
GO
```

   a. If exists drop it first so that it is always the right version.

   b. Include a unit testing section so you can test, others can test it, everyone can see how it runs

c. Parameters start with a letter indicating type

d. Using the naming convention fnDoSomething

4. Example procedure:

```
/* ---------------------------------------------------------------------
--------------------------------*/
IF EXISTS (SELECT * FROM INFORMATION_SCHEMA.ROUTINES
   WHERE ROUTINE_NAME = 'spCancelRequest' AND ROUTINE_TYPE = 'PROCEDURE')
      DROP PROCEDURE spCancelRequest
GO
/* ---------------------------------------------------------------------
--------------------------------*/
CREATE PROCEDURE spCancelRequest
      @nOrderID int
AS
/*
      UNIT TESTING
      ============
      exec spCancelRequest @nOrderID = 5
*/
BEGIN
      DECLARE @nResult int
      UPDATE Order
            SET OrderStatusID = 8
            WHERE OrderID = @nOrderID
      SELECT @nResult = @@ERROR
      RETURN @nResult
END
GO
```

a. If exists drop it first so that it is always the right version.

b. Include a unit testing section so you can test, others can test it, everyone can see how it runs

c. Return the error code

d. Using the naming convention spDoSomething

Triggers

1. Add triggers for all DateUpdated columns:

```
CREATE TRIGGER trCompanyUpdated ON Company AFTER UPDATE AS
      UPDATE poCompany
            SET DateUpdated = GETDATE()
            FROM inserted
            WHERE (Company.CompanyID = inserted.CompanyID)
```

2. Do not use triggers for anything else unless you have permission from Jon or Jarrod as they are very hard to diagnose and should only be used in special situations