COMMONWEALTH OF PENNSYLVANIA
DEPARTMENT'S OF HUMAN SERVICES, INSURANCE, AND AGING

INFORMATION TECHNOLOGY STANDARD

Name Of Standard: **MS SQL Server 2012 / 2014 Naming and Coding Standard**	Number: **STD-DMS009**
Domain: **Data**	Category:
Date Issued:	Issued By Direction Of:
Date Revised: **9/15/2016**	Clifton Van Scyoc, Dir of Division of Technical Engineering

Abstract:
The purpose of this document is to detail and give examples of Microsoft Structured Query Language (MS SQL) development standards followed at the Departments of Human Services (DHS), Insurance (PID) and Aging (PDA).

General:
Coding standards are conventions and methods developers follow when developing, editing or maintaining program code. Better programming style, developer understanding, readability and reduced application development time are the results of following coding standards.

Database Modeling Environment

Computer Associates (CA) ERwin Data Modeler is the standard tool for designing and modeling SQL Server 2008/2012/2014 databases.

All models are reviewed by the Database Management Section.

All completed models are stored in the Microsoft Visual Studio Team Foundation Server.

Database Data Definition Environment

Microsoft SQL Server Management Studio is the standard tool to use for the execution and modification of Data Definition Language (DDL) for the development of SQL Server 2008/2012/2014 databases. The general process is:

1. Initially generate a DDL from ERwin using forward engineering, or if required, create DDL statements manually within SQL Server Management Studio.
2. Execute the DDL within SQL Server Management Studio and, if necessary, edit and/or tune the statements using the Management Studio environment.
3. Reverse engineer the database back into ERwin to keep the database and data model synchronized.

Store revised data models and all DDL scripts in Microsoft Visual Studio Team Foundation Server. Store the DDL scripts in a consistent manner under the project folder in Team Foundation Server.

Database Programming Environment

Use SQL Server 2008/2012/2014 Management Studio to code and test stored procedures and triggers.

Prototype every process using Transact-SQL (T-SQL) within SQL Server Management Studio prior to creating a new stored procedure, view, function or trigger. This results in ease of debugging.

Use Execution Plans

Estimated Execution plans need to be used while developing T-SQL.

Actual Execution plans need to be included in the migration packet.

Use Database Engine Tuning Advisor

SQL Server DBA's and analysts use this tool to determine if additional keys or indexes may be required. Creation of the additional objects are determined and executed by database administrators and analysts.

General Import/Export File Rules

All files imported to and exported from SQL Server 2008/2012/2014 preferred delimited. Although commas, hyphens, semicolons and other characters can be used to delimit files, **only a vertical bar (| or 'pipe') is allowed, no exceptions**. Using other characters introduces the possibility of that character being embedded as valid data within a text or large variable character column. A vertical bar is less likely to be valid data.

Commenting Guidelines

These rules apply in general to all source files.

Object Header Usage

Include object headers in every source code file.

Object Maintenance Documentation

Include object maintenance documentation in every source code file. Keep this documentation current on all revisions after moving any given source file into production. Before initial migration into production, revision information is not required.

Comment Quality

Comments must contain a description of the process the SQL module (source file), procedure, or trigger accomplishes. Write the description in clear, concise, non-colloquial English, using business terms to describe the processes.

Do not use comments such as:

```
--
--    Declare the Customer Name Variable
--
DECLARE @CustomerName AS VARCHAR(32)
```

Describe the purpose of the process and how it works, in non-technical terms. This is important because most developers can read the SQL and see line-by-line what it does mechanically, but may not be able to easily grasp the purpose (business logic) behind the code. Write comments to aid the reader of the code. Assume that the reader is not well versed in Transact-SQL and try always to describe the “how” and “why” of the process clearly.

Comment Quantity

More comments are better than using fewer comments. Remember that the code is reviewed first, then, over time (perhaps after many years), require revisions and enhancements. Providing adequate comments allows the database administrator, developer, and the developer’s peers to quickly understand the logic and perform necessary revisions, and so forth.

Standard Object Header

Every object contains an object header. An object header contains the database name, object name, creation date, author, a brief description of the object, parameters passed, returns, calling mechanism, tables and aliases used in code, other procedures called, DPSR #, notes, and any special comments and warnings. Module headers remain open on the right-hand side.

Documentation for Object Maintenance

This section describes how to document changes in the Transact-SQL source code. The method involves using a maintenance documentation heading and comments within the source code. A common developer signature provides these together.

Object Maintenance Documentation

Immediately following the object header there is a skeletal maintenance block. Use this skeletal block as a template for maintenance done to the object code. Copy, paste, and fill in this block when you alter the object code. The block copy marks are on the lines with the equal signs. Include the lines with the block copy marks.

Make the insertion of the copied block directly after the skeleton block. Thus, the history is recorded from earliest to latest.

Example of OBJECT HEADER AND MAINTENANCE DOCUMENTATION

```
/****************************************************************************
*      DATABASE:                  OMAPINTERNET
*      PROCEDURE NAME:            usp_SelectAllResults
*      DATE:                      04/03/2009
*      AUTHOR:                    Homer Simpson
*      PROCEDURE DESC:            This proc returns all the results for a specific NDC,
*                                 ShortName or market basket code or their combination
*      WorkOrder#                 13798
*****************************************************************************
*      DATE:           Developer        Change
----------      --------------- ----------------------------------------
   08/01/2013        Joe Developer      Add Indicator

*****************************************************************************/
```

Minor Line-level Commenting

Do not put comments on the same line as the code. Examples of this illegal style follow:

```
DECLARE @IsNew AS Integer           -- Is New Flag
DECLARE @Firstname AS VARCHAR(32)   -- First Name Field
```

Insert all comments for a particular section of code before the code block, with one blank comment line before and after the comment. An example follows:

```
--
--   Declare the local working variables
--
DECLARE @IsNew  AS Integer
DECLARE @Firstname AS VARCHAR(32)

--
--   Return identity value.
--
SET @IDENTITY_CONTACT = CAST(SCOPE_IDENTITY() AS INT)

    SELECT @IDENTITY_CONTACT
```

Formatting Guidelines

Font Style, Size, and Capitalization

Font Style and Size

Write all SQL code in 10 point Courier New font.

Capitalization

Please make sure that Transact-SQL keywords are opposite case from variables, field names, and object names when coding SQL statements,.

Tab Stops

Set tabs to either four spaces or eight spaces (default).

Line Length

Do not write a Transact-SQL line that exceeds 132 characters in length, unless there is no other way to properly format the source.

Line Count—needs moved to store proc section.

Do not write procedures that exceed four pages in length excluding comments, headers, and maintenance log, without written justification to the data base administrators. Exceptions to rule are procedures used in conversion or loads of data. Code exceeding this limit should be looked at closer and possibly split into smaller stored procedures or modularized.

Star "*" Usage

Do not use the star symbol "*" in a Transact-SQL statement, as this is very inefficient.

This is prohibited .In all instances, include the appropriate fields in the statements where needed. An example of the wrong way to write a Transact-SQL statement follows:

```
SELECT  *
FROM    T_VENDOR
```

If you must return a count, select one field in the table or use 1. Example below.

```
Select count (1)
from T_VENDOR
```

Examples of the right way to write a Transact-SQL statement follows. Specify each field as in the examples:

```
SELECT     Field1,
           Field2,
           Field3,
```

```
          Field4
FROM   TABLE_A
WHERE  nbr_vt = @nbr_vt_recent
SELECT PMT.IDN_APPLN_PMT,
  PMT.AMT_BNFT
FROM   T_APPLN_PMT PMT

INNER JOIN  T_VOU_TRNSMTL VT
   ON  VT.IDN_TRNSMTL_VOU = PMT.IDN_TRNSMTL_VOU
```

Compound Statements

Compound statements are those that are encased within a **BEGIN**…**END** structure. Formatting rules for this structure are as follows:

1. The **BEGIN** statement aligns directly under the statement above.
2. The **END** statement is aligned in the same position as its corresponding **BEGIN** statement.
3. Within the **BEGIN**…**END** structure, statements are indented and should be in alignment.

IF…ELSE Statement Blocking

Treat all "**IF**" statements (and block "ifs") as if they were compound statements using the **BEGIN**…**END** structure. No exceptions are permitted.

An example follows:

```
      IF @@ROWCOUNT < 1
BEGIN
      RETURN -1
END
      ELSE
BEGIN
      SELECT @RowReturned = @@ROWCOUNT
 RETURN 0
END
```

Multi-line Parameter Style

When you must expand a statement to more than one line (because it exceeds the 80-character line limit standard), use the style in the following example, for consistent readability. It is best to apply this style even when you do not exceed the 80-character limit (though this rule is not strictly enforced).

```
              EXECUTE     usp_MyStoredProcedure
```

```
                    @MyParameter1,
                    @MyParameter2,
                    @MyParameter3 OUTPUT,
                    @MyParameter4 OUTPUT
```

Indenting and Formatting SQL Statements

The styling presented in this standard provides for a more consistent look to all stored procedures and triggers. The guidelines presented here (by example) apply to all Transact-SQL statements. Use of white space is highly recommended.

Example of Select Statement

```
SELECT DISTINCT  t1.nbr_rcpt,
                 t2.nam_rcpt

FROM    T_TABLE1 t1,
         T_TABLE2 t2

WHERE    t1.nbr_value = t2.nbr_value
  AND    t1.cde_value = @ParmKeyVal
  AND    (t1.nbr_value < 1  OR   t1.nbr_value > 99)

ORDER BY t2.nam_rcpt DESC
```

From the example, it can be seen that:

1. Individual elements are placed in separate lines
2. Elements are indented one tab stop
3. Keywords are to the left whenever possible
4. If the length of the keyword or keywords (ex: "**SELECT DISTINCT**" and "**ORDER BY**") equals or exceeds the tab stop setting (8), the parameter is placed on the subsequent line at the proper indent position
5. Parentheses are used to distinguish logical blocks and are indented at the same level as any other parameter.

Other examples follow:

Example of Select Statement:

```
SELECT DISTINCT t1.nbr_rcpt,
                t2.nam_rcpt

FROM            T_TABLE1 t1,T_TABLE2 t2
```

```
WHERE           t1.cde_value = t2.cde_value
        AND     t1.nbr_rcpt = @ParmKeyVal
        AND     (t1.cde_value < 1 OR t1.cde_value > 99)

ORDER BY    t2.nam_rcpt DESC
```

Examples of Insert Statements

```
INSERT
INTO    T_VENDOR
(
   nam_vendor,
   cde_active,
   dte_crtd,
   idn_user_crtd
)
VALUES
(
   @Name,
   @Active,
   GETDATE(),
   @ModifiedUser
)
```

Example of Update Statement

```
UPDATE      T_VENDOR

    SET     nam_vendor = @Name,
            cde_active = @Active,
            dte_crtd = GETDATE(),
            idn_user_crtd = @ModifiedUser

    WHERE   nbr_vendor = @VendorID
```

Example of Delete Statement

```
DELETE
FROM        T_VENDOR
WHERE       nbr_vendor = @VendorID
```

Example of Case Statement

```
CASE @myfield
    WHEN 'A' THEN 'ABC '
    WHEN 'B' THEN 'BCD '
    WHEN 'C' THEN 'CDE '
ELSE  'JKL'
END
```

Naming Conventions

For table and column names please reference the Data Administration Standards on the Data Domain page under Business and Technical Standards on the BIS web site.

Table Naming

All table names follow the current BIS naming conventions and are prefaced with a T_.

Example: T_FIPS_CODE

Column Naming

All column names follow the current DHS naming conventions.

View Naming

All view names consist of the prefix VW_ followed by business function they perform or business rule they enforce.

Example: VW_SEL_T_FIPS_CODE

Variable Formats

All variable names are defined in a consistent manner across all the programs. To ensure consistency parameters are descriptive and define correctly according to their usage.

```
@p_IDN_RECON      integer,
@p_IDN_ALJ        integer,
@p_HONORABLE      varchar(20)= NULL,
```

Parameters can be set to NULL.

Instead of using an IF block to check for NULL parameters, NULLS are checked like the example below. Second example is to check a range.

```
(B.CDE_PARTY_RQNG = @p_CDE_PARTY_RQNG OR @p_CDE_PARTY_RQNG IS NULL)
```

```
((DTE_FAA_BHA >=  @p_DTE_FAA_BHA_FROM  AND DTE_FAA_BHA <=  @p_DTE_FAA_BHA_TO)
  OR (@p_DTE_FAA_BHA_FROM  Is NUll AND @p_DTE_FAA_BHA_TO Is NUll))
```

If you have to use a dynamic order by (sort results) pass in the field to sort and the sort order.

```
@p_field_order_sort   varchar(50)=null,
@p_order_sort         varchar(4)=null

IF @p_order_sort = 'ASC'

      BEGIN
            SELECT   m.ProviderName as NAM_PROVR,
                     m.NAM_BUSNS_MP,
                     m.NAM_PROVR_ALT,

           FROM     TBLMEDICHECK m

           WHERE     m.Providername = @p_nam_provr

           ORDER BY
                      CASE @p_field_order_sort
                      WHEN 'NAM_PROVR' THEN m.ProviderName
                      WHEN 'NAM_BUSNS_MP' THEN m.NAM_BUSNS_MP
                      WHEN 'NAM_PROVR_ALT' THEN NAM_PROVR_ALT
                      END

                      ASC;

        END

   ELSE

       BEGIN

         SELECT     m.ProviderName as NAM_PROVR,
                    m.NAM_BUSNS_MP,
                    m.NAM_PROVR_ALT,

          FROM     TBLMEDICHECK m

          WHERE     m.Providername = @p_nam_provr

          ORDER BY
                     CASE @p_field_order_sort
                     WHEN 'NAM_PROVR' THEN m.ProviderName
                     WHEN 'NAM_BUSNS_MP' THEN m.NAM_BUSNS_MP
                     WHEN 'NAM_PROVR_ALT' THEN NAM_PROVR_ALT
                     END

                     DESC;
       END
```

Stored Procedures

When coding enterprise applications, the use of stored procedures to separate application logic from database code has been found beneficial. Stored procedures are used for any data access (SELECT) or manipulation (INSERT, DELETE, UPDATE).

The advantages include:

- Ease of Maintenance – When supporting database structure or application logic changes, the DML is easily accessible directly from the database, can be easily scanned using SQL, and eliminates the need to interrogate each COM or .NET object when searching for a particular database reference. As such, it is recommended that all web-based applications use stored procedures for the majority, if not all, database access.
- Enhanced Performance – Stored procedures run directly on the database server, which is optimally tuned to perform such operations.
- Reusability – Stored procedures can be called from multiple applications, thereby reducing the time required to design, code and test commonly used application functions.

Stored procedures are named according to the business function they perform or business rule they enforce. The name should include a prefix of USP and a business description of the action performed. The name is appropriately abbreviated with items found in the standard abbreviation list. An example of a procedure name is USP_ADD_NEW_T_INDIV. As the name implies, this procedure adds a new row to the T_INDIV table.

All stored procedures, begin with one of the following:

USP_SELECT
USP_UPDATE
USP_INSERT
USP_DELETE

Stored Procedures performs only one function. This means one insert, one update or one delete statement per stored procedure. An “if” clause determining whether an insert or update should be performed, is acceptable. See the DBA team if there are any questions.
Multiple update\delete\insert statements can cause locking\blocking, orphan records and unhandled transactions.

Do not write store procedures that exceed four pages in length excluding comments, headers, and maintenance log, without written justification to the data base administrators. Exceptions to the rule are procedures used in conversion or loads of data. Analyze code exceeding this limit and split into smaller stored procedures or modularize.

For a SELECT, items in yellow are standard. The comment block is simple and contains a brief audit listing of changes that go to production; not a running summary of changes during development. Error Trapping is highly recommended with the Enterprise libraries, because if

you have an error, the first thing we ask is what the error log says. Please also list the fields. No SELECT * is allowed. Also, white space for readability is also highly desirable.

Example

```
USE [RECON]
GO
/****** Object:  StoredProcedure [RECONSchema].[USP_SELECT_BACK_END_RECON]
Script Date: 09/10/2015 09:23:38 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
/*****************************************************************************
-- Author:        George Washington
-- Create date:   07/04/2012
-- Description:   get all fields for the Back End Screen from the RECON table

*****************************************************************************

DATE:           Developer             Change
----------      ---------------       -----------------------------------------
01/05/2012      George Washington     New
03/25/2013      Tom Jefferson         Added DTE_CLER_RTND
*****************************************************************************/

ALTER PROCEDURE [RECONSchema].[USP_SELECT_BACK_END_RECON]
      (
                        @IDN_RECONS       Integer
       )
AS
BEGIN

    SET NOCOUNT ON;

    BEGIN TRY


SELECT [IDN_RECONS]
      ,[TXT_CASE_NUM]
      ,[TXT_APPEAL_NUM]
      ,[TXT_BHA_FILE_NUM]
      ,[DTE_RCVD_BHA]
      ,[DTE_MLNG_ORDER_FINAL_SCTRY]
      ,[DTE_CREATN]
      ,[NAM_BY_CRTD]
      ,[DTE_MODFD]
      ,[NAM_BY_MODFD]

  FROM [RECON].[dbo].[T_RECON]
  where IDN_RECONS = @IDN_RECONS

      END TRY

      BEGIN CATCH
            SELECT
```

```
                  ERROR_NUMBER() AS ErrorNumber,
                  ERROR_SEVERITY() AS ErrorSeverity,
                  ERROR_STATE() AS ErrorState,
                  ERROR_PROCEDURE() AS ErrorProcedure,
                  ERROR_LINE() AS ErrorLine,
                  ERROR_MESSAGE() AS ErrorMessage;
      END CATCH
END
```

For an UPDATE, items in yellow are standard. The comment block is simple and contains a brief audit listing of changes that go to production; not a running summary of changes during development. Error Trapping is highly recommended with the Enterprise libraries, because if you have an error, the first question is what the error log says. Please also list the fields. No SELECT * is allowed. Also, white space for readability is also highly desirable.

Example

```
USE [StateFacilityTrackingSystem]
GO
/****** Object:  StoredProcedure [PTS].[USP_UPDATE_BSU]    Script Date: 09/10/2015
09:52:53 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO

/*****************************************************************************
Database:         StateFacilityTrackingSystem
Procedure Name:   USP_UPDATE_BSU
Date:             10/21/2014
Author:           Harry Potter

Procedure Desc:   This procedure creates new bsu for existing admission.

Parameters: see below

*****************************************************************************
DATE:           Developer        Change
----------      ---------------- ----------------------------------------
02/02/2014      PETER PAN        New

*****************************************************************************/

ALTER PROCEDURE [PTS].[USP_UPDATE_BSU]

      @IDN_PTN_ADMSN AS INT=0,
      @IDN_BSU AS INT,
      @PNT_BSU_CODE AS VARCHAR(7),
      @PNT_BSU_NUMBER AS VARCHAR(7),
      @PNT_BSU_EFF_DTE AS DATETIME,
      @CWOPA_ID AS VARCHAR(12)=NULL
AS
SET NOCOUNT ON;

BEGIN TRY
```

```
UPDATE PTS.T_PNT_BSU

SET CDE_BSU=@PNT_BSU_CODE,
    DTE_EFFV_BSU=@PNT_BSU_EFF_DTE,
    NBR_CASE_BSU=@PNT_BSU_NUMBER,
    DTE_UPDTD_RECORD=GETDATE(),
    IDN_USER_UPDTD_RECORD=@CWOPA_ID

WHERE IDN_BSU_PNT=@IDN_BSU

 END TRY

      BEGIN CATCH
            SELECT
                  ERROR_NUMBER() AS ErrorNumber,
                  ERROR_SEVERITY() AS ErrorSeverity,
                  ERROR_STATE() AS ErrorState,
                  ERROR_PROCEDURE() AS ErrorProcedure,
                  ERROR_LINE() AS ErrorLine,
                  ERROR_MESSAGE() AS ErrorMessage;
      END CATCH
```

An INSERT/UPDATE is also acceptable as long as you are doing either an INSERT or an UPDATE – not both! This is acceptable:

```
IF @P_IDN_USER = 'cwopa\user'

        BEGIN

            UPDATE PIOS.dbo.T_CW_ASGMT

                SET IDN_CW = @P_IDN_CW,
                    DTE_ASGMT_CASE = @P_DTE_ASGMT_CASE,
                    IDN_USER_CHANGE_LAST = @P_IDN_USER,
                    DTE_RECORD_CHANGE_LAST  = GetDate()

                WHERE IDN_ASGMT_CW = @P_IDN_ASGMT_CW
        END
ELSE

        BEGIN

            INSERT INTO PIOS.dbo.T_CW_ASGMT
                                    (IDN_CW
                                    ,IDN_STATUS
                                    ,DTE_CHANGE_STATUS)

                                VALUES
                                    (@P_IDN_CW
                                    ,@P_IDN_STATUS
                                    ,NULL)
        END
```

THIS IS NOT ACCEPTABLE: An Insert or UPDATE stored procedure performs one UPDATE or INSERT, and rarely more than that. Check parameters in your .NET code and then an INSERT or UPDATE is called.

A Transaction can be set in the .NET code to either commit all INSERTS or none of them if the code has multiple INSERTS/UPDATES. In this case a collection in .NET can be used and the same INSERT could be called as many times as needed as you loop through the collection.

```
USE [RECON]
GO
/****** Object:  StoredProcedure [PTS].[USP_INSERT_DIAGNOSTICS]    Script Date:
09/10/2015 10:18:52 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO

/*****************************************************************************
**
Database          :     RECON
Procedure Name    :     USP_INSERT_DIAGNOSTICS
Date              :     10/14/2014
Author            :     Tom Sawyer

Procedure Desc    :     This procedure inserts records into Diagnostics table.

Parameters: see below

*****************************************************************************
*
DATE:           Developer                Change
----------      ---------------    ----------------------------------------
10/14/2014      Will Robinson            New
*****************************************************************************
*/

ALTER PROCEDURE [PTS].[USP_INSERT_DIAGNOSTICS]
@idn_pnt AS INT,
@idn_admsn_pnt AS INT,
@ambulationCode AS VARCHAR(1),
@psych_Code_1 AS VARCHAR(7) ,
@psych_Code_Date_1 AS DATETIME,
@psych_Code_NBR_Order_1 AS VARCHAR(1),
@psych_Code_2 AS VARCHAR(7),
@psych_Code_Date_2 AS DATETIME,
@psych_Code_NBR_Order_2 AS VARCHAR(1),
@psych_Code_3 AS VARCHAR(7),
@psych_Code_Date_3 AS DATETIME,
@psych_Code_NBR_Order_3 AS VARCHAR(1),
@psych_Code_4 AS VARCHAR(7),
@psych_Code_Date_4 AS DATETIME,
@psych_Code_NBR_Order_4 AS VARCHAR(1),
@psych_Code_5 AS VARCHAR(7),
```

```
@psych_Code_Date_5 AS DATETIME,
@psych_Code_NBR_Order_5 AS VARCHAR(1),
@psych_Code_6 AS VARCHAR(7),
@psych_Code_Date_6 AS DATETIME,
@psych_Code_NBR_Order_6 AS VARCHAR(1),
@psych_Code_7 AS VARCHAR(7),
@psych_Code_Date_7 AS DATETIME,
@psych_Code_NBR_Order_7 AS VARCHAR(1),
@substance_Code_1 AS VARCHAR(7),
@substance_Code_Date_1 AS DATETIME,
@substance_Code_NBR_Order_1 AS VARCHAR(1),
@substance_Code_2 AS VARCHAR(7),
@substance_Code_Date_2 AS DATETIME,
@substance_Code_NBR_Order_2 AS VARCHAR(1),
@substance_Code_3 AS VARCHAR(7),
@substance_Code_Date_3 AS DATETIME,
@CWOPA_ID AS VARCHAR(12)
AS

SET NOCOUNT ON;

BEGIN TRY

      BEGIN TRANSACTION;

            INSERT INTO PTS.T_PNT_AMBTN
            (
                  CDE_AMBTN,
                  IDN_ADMSN_PNT,
                  DTE_CREATN_RECORD,
                  IDN_USER_CREATN_RECORD,
                  DTE_UPDTD_RECORD,
                  IDN_USER_UPDTD_RECORD
            )
            VALUES
            (
                  @ambulationCode,
                  @idn_admsn_pnt,
                  GETDATE(),
                  @CWOPA_ID,
                  GETDATE(),
                  @CWOPA_ID
            )

            if @psych_Code_1 IS NOT NULL and LTRIM(RTRIM(@psych_Code_1)) <> ''
            begin

                  INSERT INTO PTS.T_PNT_DGNSTC
                  (
                        DTE_EFFV_DGNSTC,
                        IND_RECORD_ARCD,
                        NBR_ORDER_DGNSTC,
                        DTE_CREATN_RECORD,
                        IDN_USER_CREATN_RECORD,
                        DTE_UPDTD_RECORD,
                        IDN_USER_UPDTD_RECORD,
                        IDN_ADMSN_PNT,
```

```
                IDN_DX,
                IDN_TYPE_DGNSTC
        )
        VALUES
        (
                @psych_Code_Date_1,
                0,
                @psych_Code_NBR_Order_1,
                GETDATE(),
                @CWOPA_ID,
                GETDATE(),
                @CWOPA_ID,
                @idn_admsn_pnt,
                @psych_Code_1,
                1
        )
end

if @psych_Code_2 IS NOT NULL and LTRIM(RTRIM(@psych_Code_2)) <> ''
begin

        INSERT INTO PTS.T_PNT_DGNSTC
        (
                DTE_EFFV_DGNSTC,
                IND_RECORD_ARCD,
                NBR_ORDER_DGNSTC,
                DTE_CREATN_RECORD,
                IDN_USER_CREATN_RECORD,
                DTE_UPDTD_RECORD,
                IDN_USER_UPDTD_RECORD,
                IDN_ADMSN_PNT,
                IDN_DX,
                IDN_TYPE_DGNSTC
        )
        VALUES
        (
                @psych_Code_Date_2,
                0,
                @psych_Code_NBR_Order_2,
                GETDATE(),
                @CWOPA_ID,
                GETDATE(),
                @CWOPA_ID,
                @idn_admsn_pnt,
                @psych_Code_2,
                1
        )
end

if @psych_Code_3 IS NOT NULL and LTRIM(RTRIM(@psych_Code_3)) <> ''
begin

        INSERT INTO PTS.T_PNT_DGNSTC
        (
                DTE_EFFV_DGNSTC,
                IND_RECORD_ARCD,
                NBR_ORDER_DGNSTC,
```

```
                DTE_CREATN_RECORD,
                IDN_USER_CREATN_RECORD,
                DTE_UPDTD_RECORD,
                IDN_USER_UPDTD_RECORD,
                IDN_ADMSN_PNT,
                IDN_DX,
                IDN_TYPE_DGNSTC
        )
        VALUES
        (
                @psych_Code_Date_3,
                0,
                @psych_Code_NBR_Order_3,
                GETDATE(),
                @CWOPA_ID,
                GETDATE(),
                @CWOPA_ID,
                @idn_admsn_pnt,
                @psych_Code_3,
                1
        )
end

if @psych_Code_4 IS NOT NULL and LTRIM(RTRIM(@psych_Code_4)) <> ''
begin

        INSERT INTO PTS.T_PNT_DGNSTC
        (
                DTE_EFFV_DGNSTC,
                IND_RECORD_ARCD,
                NBR_ORDER_DGNSTC,
                DTE_CREATN_RECORD,
                IDN_USER_CREATN_RECORD,
                DTE_UPDTD_RECORD,
                IDN_USER_UPDTD_RECORD,
                IDN_ADMSN_PNT,
                IDN_DX,
                IDN_TYPE_DGNSTC
        )
        VALUES
        (
                @psych_Code_Date_4,
                0,
                @psych_Code_NBR_Order_4,
                GETDATE(),
                @CWOPA_ID,
                GETDATE(),
                @CWOPA_ID,
                @idn_admsn_pnt,
                @psych_Code_4,
                1
        )
end

if @psych_Code_5 IS NOT NULL and LTRIM(RTRIM(@psych_Code_5)) <> ''
begin
```

```
		INSERT INTO PTS.T_PNT_DGNSTC
		(
			DTE_EFFV_DGNSTC,
			IND_RECORD_ARCD,
			NBR_ORDER_DGNSTC,
			DTE_CREATN_RECORD,
			IDN_USER_CREATN_RECORD,
			DTE_UPDTD_RECORD,
			IDN_USER_UPDTD_RECORD,
			IDN_ADMSN_PNT,
			IDN_DX,
			IDN_TYPE_DGNSTC
		)
		VALUES
		(
			@psych_Code_Date_5,
			0,
			@psych_Code_NBR_Order_5,
			GETDATE(),
			@CWOPA_ID,
			GETDATE(),
			@CWOPA_ID,
			@idn_admsn_pnt,
			@psych_Code_5,
			1
		)
	end

	if @psych_Code_6 IS NOT NULL and LTRIM(RTRIM(@psych_Code_6)) <> ''
	begin

		INSERT INTO PTS.T_PNT_DGNSTC
		(
			DTE_EFFV_DGNSTC,
			IND_RECORD_ARCD,
			NBR_ORDER_DGNSTC,
			DTE_CREATN_RECORD,
			IDN_USER_CREATN_RECORD,
			DTE_UPDTD_RECORD,
			IDN_USER_UPDTD_RECORD,
			IDN_ADMSN_PNT,
			IDN_DX,
			IDN_TYPE_DGNSTC
		)
		VALUES
		(
			@psych_Code_Date_6,
			0,
			@psych_Code_NBR_Order_6,
			GETDATE(),
			@CWOPA_ID,
			GETDATE(),
			@CWOPA_ID,
			@idn_admsn_pnt,
			@psych_Code_6,
			1
		)
```

```
		end

		if @psych_Code_7 IS NOT NULL and LTRIM(RTRIM(@psych_Code_7)) <> ''
		begin

			INSERT INTO PTS.T_PNT_DGNSTC
			(
				DTE_EFFV_DGNSTC,
				IND_RECORD_ARCD,
				NBR_ORDER_DGNSTC,
				DTE_CREATN_RECORD,
				IDN_USER_CREATN_RECORD,
				DTE_UPDTD_RECORD,
				IDN_USER_UPDTD_RECORD,
				IDN_ADMSN_PNT,
				IDN_DX,
				IDN_TYPE_DGNSTC
			)
			VALUES
			(
				@psych_Code_Date_7,
				0,
				@psych_Code_NBR_Order_7,
				GETDATE(),
				@CWOPA_ID,
				GETDATE(),
				@CWOPA_ID,
				@idn_admsn_pnt,
				@psych_Code_7,
				1
			)
		end

		if @substance_Code_1 IS NOT NULL and LTRIM(RTRIM(@substance_Code_1)) <>
''
		begin

			INSERT INTO PTS.T_PNT_DGNSTC
			(
				DTE_EFFV_DGNSTC,
				IND_RECORD_ARCD,
				NBR_ORDER_DGNSTC,
				DTE_CREATN_RECORD,
				IDN_USER_CREATN_RECORD,
				DTE_UPDTD_RECORD,
				IDN_USER_UPDTD_RECORD,
				IDN_ADMSN_PNT,
				IDN_DX,
				IDN_TYPE_DGNSTC
			)
			VALUES
			(
				@substance_Code_Date_1,
				0,
				@substance_Code_NBR_Order_1,
				GETDATE(),
				@CWOPA_ID,
```

```
                        GETDATE(),
                        @CWOPA_ID,
                        @idn_admsn_pnt,
                        @substance_Code_1,
                        2
                )
        end

        if @substance_Code_2 IS NOT NULL and LTRIM(RTRIM(@substance_Code_2)) <>
''
        begin

                INSERT INTO PTS.T_PNT_DGNSTC
                (
                        DTE_EFFV_DGNSTC,
                        IND_RECORD_ARCD,
                        NBR_ORDER_DGNSTC,
                        DTE_CREATN_RECORD,
                        IDN_USER_CREATN_RECORD,
                        DTE_UPDTD_RECORD,
                        IDN_USER_UPDTD_RECORD,
                        IDN_ADMSN_PNT,
                        IDN_DX,
                        IDN_TYPE_DGNSTC
                )
                VALUES
                (
                        @substance_Code_Date_2,
                        0,
                        @substance_Code_NBR_Order_2,
                        GETDATE(),
                        @CWOPA_ID,
                        GETDATE(),
                        @CWOPA_ID,
                        @idn_admsn_pnt,
                        @substance_Code_2,
                        2
                )
        end

        if @substance_Code_3 IS NOT NULL and LTRIM(RTRIM(@substance_Code_3)) <>
''
        begin

                INSERT INTO PTS.T_PNT_DGNSTC
                (
                        DTE_EFFV_DGNSTC,
                        IND_RECORD_ARCD,
                        NBR_ORDER_DGNSTC,
                        DTE_CREATN_RECORD,
                        IDN_USER_CREATN_RECORD,
                        DTE_UPDTD_RECORD,
                        IDN_USER_UPDTD_RECORD,
                        IDN_ADMSN_PNT,
                        IDN_DX,
                        IDN_TYPE_DGNSTC
                )
```

```
		VALUES
		(
			@substance_Code_Date_3,
			0,
			@substance_Code_NBR_Order_3,
			GETDATE(),
			@CWOPA_ID,
			GETDATE(),
			@CWOPA_ID,
			@idn_admsn_pnt,
			@substance_Code_3,
			2
		)
	end

	if @substance_Code_4 IS NOT NULL and LTRIM(RTRIM(@substance_Code_4)) <>
''
	begin

		INSERT INTO PTS.T_PNT_DGNSTC
		(
			DTE_EFFV_DGNSTC,
			IND_RECORD_ARCD,
			NBR_ORDER_DGNSTC,
			DTE_CREATN_RECORD,
			IDN_USER_CREATN_RECORD,
			DTE_UPDTD_RECORD,
			IDN_USER_UPDTD_RECORD,
			IDN_ADMSN_PNT,
			IDN_DX,
			IDN_TYPE_DGNSTC
		)
		VALUES
		(
			@substance_Code_Date_4,
			0,
			@substance_Code_NBR_Order_4,
			GETDATE(),
			@CWOPA_ID,
			GETDATE(),
			@CWOPA_ID,
			@idn_admsn_pnt,
			@substance_Code_4,
			2
		)
	end

	if @medical_Code_1 IS NOT NULL and LTRIM(RTRIM(@medical_Code_1)) <> ''
	begin

		INSERT INTO PTS.T_PNT_DGNSTC
		(
			DTE_EFFV_DGNSTC,
			IND_RECORD_ARCD,
			NBR_ORDER_DGNSTC,
			DTE_CREATN_RECORD,
			IDN_USER_CREATN_RECORD,
```

```
                DTE_UPDTD_RECORD,
                IDN_USER_UPDTD_RECORD,
                IDN_ADMSN_PNT,
                IDN_DX,
                IDN_TYPE_DGNSTC
        )
        VALUES
        (
                @medical_Code_Date_1,
                0,
                @medical_Code_NBR_Order_1,
                GETDATE(),
                @CWOPA_ID,
                GETDATE(),
                @CWOPA_ID,
                @idn_admsn_pnt,
                @medical_Code_1,
                3
        )
end

if @medical_Code_2 IS NOT NULL and LTRIM(RTRIM(@medical_Code_2)) <> ''
begin

        INSERT INTO PTS.T_PNT_DGNSTC
        (
                DTE_EFFV_DGNSTC,
                IND_RECORD_ARCD,
                NBR_ORDER_DGNSTC,
                DTE_CREATN_RECORD,
                IDN_USER_CREATN_RECORD,
                DTE_UPDTD_RECORD,
                IDN_USER_UPDTD_RECORD,
                IDN_ADMSN_PNT,
                IDN_DX,
                IDN_TYPE_DGNSTC
        )
        VALUES
        (
                @medical_Code_Date_2,
                0,
                @medical_Code_NBR_Order_2,
                GETDATE(),
                @CWOPA_ID,
                GETDATE(),
                @CWOPA_ID,
                @idn_admsn_pnt,
                @medical_Code_2,
                3
        )
end

if @medical_Code_3 IS NOT NULL and LTRIM(RTRIM(@medical_Code_3)) <> ''
begin

        INSERT INTO PTS.T_PNT_DGNSTC
        (
```

```
                DTE_EFFV_DGNSTC,
                IND_RECORD_ARCD,
                NBR_ORDER_DGNSTC,
                DTE_CREATN_RECORD,
                IDN_USER_CREATN_RECORD,
                DTE_UPDTD_RECORD,
                IDN_USER_UPDTD_RECORD,
                IDN_ADMSN_PNT,
                IDN_DX,
                IDN_TYPE_DGNSTC
        )
        VALUES
        (
                @medical_Code_Date_3,
                0,
                @medical_Code_NBR_Order_3,
                GETDATE(),
                @CWOPA_ID,
                GETDATE(),
                @CWOPA_ID,
                @idn_admsn_pnt,
                @medical_Code_3,
                3
        )
end

if @medical_Code_4 IS NOT NULL and LTRIM(RTRIM(@medical_Code_4)) <> ''
begin

        INSERT INTO PTS.T_PNT_DGNSTC
        (
                DTE_EFFV_DGNSTC,
                IND_RECORD_ARCD,
                NBR_ORDER_DGNSTC,
                DTE_CREATN_RECORD,
                IDN_USER_CREATN_RECORD,
                DTE_UPDTD_RECORD,
                IDN_USER_UPDTD_RECORD,
                IDN_ADMSN_PNT,
                IDN_DX,
                IDN_TYPE_DGNSTC
        )
        VALUES
        (
                @medical_Code_Date_4,
                0,
                @medical_Code_NBR_Order_4,
                GETDATE(),
                @CWOPA_ID,
                GETDATE(),
                @CWOPA_ID,
                @idn_admsn_pnt,
                @medical_Code_4,
                3
        )
end
```

```
if @medical_Code_5 IS NOT NULL and LTRIM(RTRIM(@medical_Code_5)) <> ''
begin

	INSERT INTO PTS.T_PNT_DGNSTC
	(
		DTE_EFFV_DGNSTC,
		IND_RECORD_ARCD,
		NBR_ORDER_DGNSTC,
		DTE_CREATN_RECORD,
		IDN_USER_CREATN_RECORD,
		DTE_UPDTD_RECORD,
		IDN_USER_UPDTD_RECORD,
		IDN_ADMSN_PNT,
		IDN_DX,
		IDN_TYPE_DGNSTC
	)
	VALUES
	(
		@medical_Code_Date_5,
		0,
		@medical_Code_NBR_Order_5,
		GETDATE(),
		@CWOPA_ID,
		GETDATE(),
		@CWOPA_ID,
		@idn_admsn_pnt,
		@medical_Code_5,
		3
	)
end

if @medical_Code_6 IS NOT NULL and LTRIM(RTRIM(@medical_Code_6)) <> ''
begin

	INSERT INTO PTS.T_PNT_DGNSTC
	(
		DTE_EFFV_DGNSTC,
		IND_RECORD_ARCD,
		NBR_ORDER_DGNSTC,
		DTE_CREATN_RECORD,
		IDN_USER_CREATN_RECORD,
		DTE_UPDTD_RECORD,
		IDN_USER_UPDTD_RECORD,
		IDN_ADMSN_PNT,
		IDN_DX,
		IDN_TYPE_DGNSTC
	)
	VALUES
	(
		@medical_Code_Date_6,
		0,
		@medical_Code_NBR_Order_6,
		GETDATE(),
		@CWOPA_ID,
		GETDATE(),
		@CWOPA_ID,
		@idn_admsn_pnt,
```

```
                        @medical_Code_6,
                        3
                  )
            end

            if @medical_Code_7 IS NOT NULL and LTRIM(RTRIM(@medical_Code_7)) <> ''
            begin

                  INSERT INTO PTS.T_PNT_DGNSTC
                  (
                        DTE_EFFV_DGNSTC,
                        IND_RECORD_ARCD,
                        NBR_ORDER_DGNSTC,
                        DTE_CREATN_RECORD,
                        IDN_USER_CREATN_RECORD,
                        DTE_UPDTD_RECORD,
                        IDN_USER_UPDTD_RECORD,
                        IDN_ADMSN_PNT,
                        IDN_DX,
                        IDN_TYPE_DGNSTC
                  )
                  VALUES
                  (
                        @medical_Code_Date_7,
                        0,
                        @medical_Code_NBR_Order_7,
                        GETDATE(),
                        @CWOPA_ID,
                        GETDATE(),
                        @CWOPA_ID,
                        @idn_admsn_pnt,
                        @medical_Code_7,
                        3
                  )
            end

      COMMIT;
END TRY
BEGIN CATCH

      ROLLBACK;
      SELECT
            ERROR_NUMBER() AS ErrorNumber,
            ERROR_SEVERITY() AS ErrorSeverity,
            ERROR_STATE() AS ErrorState,
            ERROR_PROCEDURE() AS ErrorProcedure,
            ERROR_LINE() AS ErrorLine,
            ERROR_MESSAGE() AS ErrorMessage;
END CATCH
```

Triggers

Triggers are to be avoided as they have the potential to cause execute multiple SQL statements without other developer knowledge.

Constraints

Planning and creating tables requires identifying how to enforce integrity of the data stored within the columns of the tables. MS SQL Server has the following constraints to enforce integrity:

Primary Key Constraints are columns or a column that uniquely identifies a row within a table. All tables should have a primary key and composite primary keys should be avoided. Primary keys should be named PK_*tablename_columnname*. For most of your OLTP tables, an identity column is the primary key.

Example: PK_T_ADDRESS_IDN_ADDR.

Foreign Key Constraints are columns or a column that is used to enforce a relation between information in two tables. A link is defined between the two tables when a primary key is referenced by a column or columns in another table. The reference becomes a foreign key in the second table. Foreign keys are named as FK_*foreignkeytable_primarykeytable_columnname.*

Example: FK_T_NAME_T_ADDRESS_IDN_ADDR.

Indexes

An index is an on-disk structure associated with a table or a view that speeds retrieval of rows from the table or the view. An index contains keys built from one or more columns in the table or the view. These keys are stored in a structure (B-tree) that enables SQL Server to find the row or rows associated with the key values quickly and efficiently.

A table or view can contain the following types of indexes:

- Clustered
    - Clustered indexes sort and store the data rows in the table or view based on their key values. These are the columns included in the index definition. There can be only one clustered index per table, because the data rows themselves can be sorted in only one order.
    - The only time the data rows in a table are stored in sorted order is when the table contains a clustered index. When a table has a clustered index, the table is called a clustered table. If a table has no clustered index, its data rows are stored in an unordered structure called a heap.
- Nonclustered
    - Nonclustered indexes have a structure separate from the data rows. A nonclustered index contains the nonclustered index key values and each key value entry has a pointer to the data row that contains the key value.

- The pointer from an index row in a nonclustered index to a data row is called a row locator. The structure of the row locator depends on whether the data pages are stored in a heap or a clustered table. For a heap, a row locator is a pointer to the row. For a clustered table, the row locator is the clustered index key.

Indexes should be named IDX_ *tablename_columnname(s).*

Example: IDX_T_ADDR_STATE_STATE

Default Definitions

Default Definitions are predefined values created on columns within a table. The default value is used when inserting a new row in a table and a particular column in the insert statement does not have a value. For example you may have a default of all 99999999 that you may want to set up as a default end date. The default has a prefix of D_ followed by a description of what the default does.

Example: DF_OPEN_DATE

User Defined Functions

User defined functions can be created in SQL Server Management Studio to perform actions such as complicated calculations or character string manipulation. User defined functions are created and used for actions repeated within an application. An example is adding dashes to social security number. This user defined function could be created and used any time a correctly formatted social security number is required. User defined functions are named UDF_*function*.

Example: UDF_FORMATSSN

Transaction and Error Handling

Try…Catch

Try…Catch, is the preferred error handling method in SQL Server. This method is used in place of the old RAISEERROR.

Errors in Transact-SQL code can be processed by using a TRY…CATCH construct similar to the exception-handling feature of the Microsoft .NET and Microsoft Visual C# languages. A TRY…CATCH construct consists of two parts: a TRY block and a CATCH block. When an error condition is detected in a Transact-SQL statement that is inside a TRY block, control is passed to a CATCH block where the error can be processed.

After the CATCH block handles the exception, control is then transferred to the first Transact-SQL statement that follows the END CATCH statement. If the END CATCH statement is the last statement in a stored procedure or trigger, control is returned to the code that invoked the stored procedure or trigger. Transact-SQL statements in the TRY block following the statement that generates an error is not executed.

If there are no errors inside the TRY block, control passes to the statement immediately after the associated END CATCH statement. If the END CATCH statement is the last statement in a stored procedure or trigger, control is passed to the statement that invoked the stored procedure or trigger.

A TRY block starts with the BEGIN TRY statement and ends with the END TRY statement. One or more Transact-SQL statements can be specified between the BEGIN TRY and END TRY statements.

A TRY block must be followed immediately by a CATCH block. A CATCH block starts with the BEGIN CATCH statement and ends with the END CATCH statement. In Transact-SQL, each TRY block is associated with only one CATCH block.

Working with TRY…CATCH

When using the TRY…CATCH construct, consider the following guidelines and suggestions:

Each TRY…CATCH construct must be inside a single batch, stored procedure, or trigger. For example, you cannot place a TRY block in one batch and the associated CATCH block in another batch. The following script would generate an error:

```
BEGIN TRY

	SELECT	IDN_AGENCY, NAM_AGENCY
	 FROM	dbo.T_AGENCY
	ORDER BY	NAM_AGENCY

END TRY

	BEGIN CATCH
		SELECT
			ERROR_NUMBER() AS ErrorNumber,
			ERROR_SEVERITY() AS ErrorSeverity,
			ERROR_STATE() AS ErrorState,
			ERROR_PROCEDURE() AS ErrorProcedure,
			ERROR_LINE() AS ErrorLine,
			ERROR_MESSAGE() AS ErrorMessage;
	END CATCH
```

Transact-SQL Restrictions

Following is SQL Server functionality that **is not** used when developing procedures or triggers.

Cursors

Cursors are not allowed.

Looping within Stored Procedures.

Looping within stored procedures is **PROHIBITED.** Looping is only performed in developer code and not in SQL stored procedures. An infinite loop can bring down SQL Server.

THIS IS NOT ACCEPTABLE!

```
WHILE len(@IDN_USER) > 0
   BEGIN
      SET @ID_USER =LEFT(@IDN_USER, charindex(',', @IDN_USER+',')-1)
       INSERT INTO   dbo.T_COURSE_INSTRR
                     (IDN_OFFNG_COURSE,
                      IDN_USER)

            VALUES   (@IDN_COURSE_OFFRNG,
                      @IDN_USER)
   END
```

Use of "*=" and "=*" and SQL JOINS- stopping point

Do not use of the "*=" and "=*" join operators in the **WHERE** clause. Use instead the **JOIN** keyword in the **FROM** clause. Microsoft does not recommend the use of "*=" and "=*" and may not provide support for these in future releases of MS SQL Server. Using this syntax for joins is discouraged (by Microsoft) because of the potential for ambiguous interpretation and because it is nonstandard. Be sure to use the join syntax. Instead of using where a = b and b = c, for example, use inner join on a = b, etc. Use left and right outer joins as well as **UNION ALLS** sparingly. Pick the join order carefully. The majority of outer joins can successfully be rewritten as inner joins with tremendous performance improvements.

Use of SELECT INTO

Do not use the **SELECT INTO** clause to create a table on the fly. Instead, create your table before the SELECT statement and use the **INSERT** statement followed by the appropriate **SELECT**.

Usage of Column Prefixes

It is generally recommended to use prefixes for columns appearing in the SELECT list. This is a coding best practice that leads to more maintainable applications.

As an example, it is preferred to have:

```
SELECT a.au_id, a.au_lname FROM dbo.authors a
```

Over this:

```
SELECT au_id, au_lname FROM dbo.authors
```

Database SQL Options

Database SQL Options should be configured as recommended to remove deprecated behaviors, be ANSI compliant, and be able to leverage the full feature set (indexed views and indexes on computed columns).

The following SQL options should be checked to see if they are configured properly as per Microsoft SQL Options control ANSI compliance options.

The following database SQL Options should be ON:

- ANSI_NULLS
- ANSI_PADDING
- ANSI_WARNINGS
- ARITHABORT
- CONCAT_NULL_YIELDS_NULL
- QUOTED_IDENTIFIERS

The following should be OFF:

- NUMERIC_ROUNDABOUT

Use of Defaults and Rules

Database Administrators check stored procedures, functions, views and triggers for existence of defaults and rules.

These objects have been deprecated in favor of CHECK constraints and are not supported in future releases of SQL Server.

Null Comparisons

Database Administrators check stored procedures, views, functions and triggers to flag the use of equality and inequality comparisons involving a NULL constant. These comparisons are undefined when ANSI_NULLS option is set to ON.

It is recommended to set ANSI_NULLS to ON and use the IS keyword to compare against NULL constants.

In a future version of SQL Server, ANSI_NULLS will always be ON and any applications that explicitly set the option to OFF will generate an error. Avoid using this feature in new development work, and plan to modify applications that currently use this feature.

String = Expression Aliasing

Database Administrators check stored procedures, functions, views and triggers for use of column aliases where the name of the expression uses a string value. It is recommended to use quoted identifiers instead. String aliases are not supported in future releases of SQL Server.

As an example, the following syntax is not recommended:

```
SELECT 'alias_for_col'=au_id+au_id FROM dbo.authors
```

Recommended alternatives are:

```
SELECT au_id+au_id as "alias_for_col" FROM dbo.authors
SELECT au_id+au_id as alias_for_col FROM dbo.authors
SELECT au_id+au_id as [alias_for_col] FROM dbo.authors
```

Primary Keys

Database Administrators check all user databases to ensure that all tables have defined either a primary key or have a column with a Unique Constraint defined. Tables without a Primary Key defined or a column with a unique constraint defined will not be approved.

Foreign Keys

Database Administrators check all user databases to ensure that proper foreign key/primary key relationships are created when needed. Foreign keys are also defined as indexes.

Order by Clause with Ordinals

Database Administrators check stored procedures, functions, views and triggers for use of ORDER BY clause specifying ordinal column numbers as sort columns.

As an example, the following syntax is not allowed:

```
SELECT au_id FROM dbo.authors ORDER BY 2, 1
```

The use of ordinal column numbers is not be allowed.

Schema Usage

Use Schema Usage only as needed; otherwise, the default schema ‘dbo’, is used. Database Administrators check stored procedures, functions, views and triggers for use of schema qualified names when referencing tables and views.

Unless specified otherwise, all Transact-SQL references to the name of a database object can be a four-part name in the form:

```
[
   server_name.[database_name].[schema_name].
   | database_name.[schema_name].
   | schema_name.
   ]
]
```

When referencing a specific object, it is not necessary to specify the server, database, and owner (schema) for SQL Server to identify the object. However, it is recommended that at least the schema name be specified to identify a table or view inside a stored procedure, function, view or trigger.

When SQL Server looks up a table/view without a schema qualification, it first looks in the default schema and then looks in the **'dbo'** schema. The default schema corresponds to the current user for adhoc batches, and corresponds to the schema of a stored procedure

when inside one. In either case, SQL Server incurs an additional runtime cost to verify schema binding of unqualified objects. Applications are more maintainable and may experience a slight performance improvement if object references are schema qualified.

Top without Order By

Database Administrators check stored procedures, functions, views and triggers for usages of TOP in queries without an ORDER BY clause.

It is recommended to specify sort criteria when using TOP clause. Otherwise, the results produced are plan dependent and may lead to undesired behavior.

Usually TOP without ORDER BY is meaningless.

Avoid Stored Procedure Recompiles

See next section.

Production Adhoc Querying

Real time adhoc reporting against the production database is not permitted. **Microsoft Access** is not allowed to connect to SQL. To do reporting as this causes table locks, performance degradation and creates a security risk.

Hard coded SQL

Hard coded SQL in web pages or applications is not permitted. Put T-SQL in stored procedures for security reasons, efficiency and ease in debugging**.**

Dynamic SQL is not allowed.

Performance Recommendations

General Considerations

Connections to the SQL Server database are to be brief. A connection is made, a process completed and the connection dropped. No persistent connections to SQL Server are allowed.

Stored procedures perform one process and not a series of processes. It is better to create a stored procedure that performs an insert rather than a stored procedure that performs an insert, update and a delete. Executing multiple stored procedures is preferred over executing one, unwieldy, multi-task stored procedure.

Stored Procedure Recompiles

No stored procedures are to interleave Data Definition Language (DDL) and Data Manipulation Language (DML) operations. In other words, do all **CREATE/DROP** statements separate from **SELECT** and **UPDATE** statements. If data operations are mixed,

the SQL server must recompile procedures each time to determine the best plan to use for each new or dropped object. Therefore, coding all creates and drops together reduces the amount of stored procedure compiles. Avoid creating a temporary table in a control-of-flow statement such as an **IF…ELSE** or **WHILE** statement. Avoid using a **DROP** statement for temporary tables in stored procedures, because tables are automatically dropped when the procedure is completed. Specifying the owner of objects such as dbo.Tablename and dbo.Stored procedure name cuts down on recompiles.

Temporary Tables

Often procedure throughput can be dramatically improved by the use of temporary tables. Restructuring a Transact-SQL statement that involves lengthy and/or complicated joins to perform the same functionally by multiple actions against a temp table will improve performance. Please note: use a temporary table to manipulate small amounts of data, and if data exceeds 500 records add an index. Also if you have > 6 data changes in a temporary table consider using the option (keep plan) to avoid unnecessary recompiles.

Table Variable Type

The table variable type functions exactly like the temporary table, with the exception that all data is loaded directly in random access memory.

Temporary Tables vs. Table Variable

Database Administrators check stored procedures and triggers for usages of temporary tables that may be replaced by use of table variables.

When a procedure creates a temporary table and has no CREATE INDEX issued on it, and it is dropped all in the same procedure, consider using table variables instead to potentially observe fewer recompilations.

Note that if large data volumes are inserted in the temporary table it may still be preferred to use temporary tables over table variables due to parallel execution restrictions and statistics maintenance.

Indexes

Examine the performance of the statements within the procedure and the entire application to determine if an additional index will justify its overhead. Consider overall index usage all along the development path. Indexes, or lack of indexes, are the direct cause of poor performance 90% of the time. As per Microsoft's suggestion, all tables that have more than 500 records are to have a clustered index, even if a field is created just to hold the index.

Please note when setting an index, pay close attention to the fill factor for the given index. The fill factor is the amount space consumed by the index and more importantly the amount of space that will be kept free for expansion. For an example a 90% fill factor is 90% full with 10% left for expansion. For tables that do a high volume of inserts, updates, deletes consider a lower fill factor, or perhaps not having a clustered index at all. A clustered index slows down inserts but speed up selects. The developer determines what the best tradeoff is.

Ultimately, it is the responsibility of the developer coding the stored procedures and the SQL code to determine what the best indexes are and how they are used.

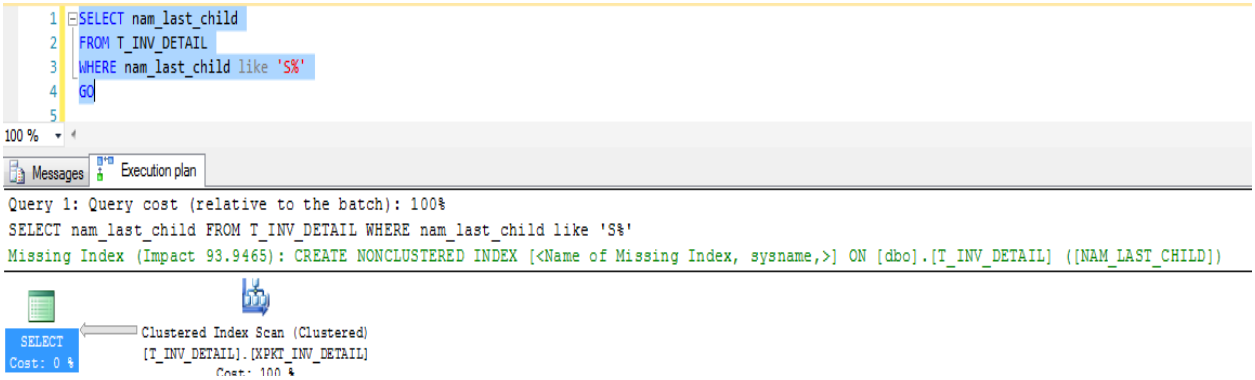
Execution Plan

When debugging and tuning a stored procedure, examine the execution plan for clues as to the performance of the procedure and how it may be improved. The reason for execution plan is to determine whether an index is required.

In the example below, execution identifies a missing index that could improve performance.

From this, one asks if another index should be created to avoid the scan.

If yes, create an index and rerun.



SQL Profiler/Database Engine Tuning Advisor

Two valuable and necessary tools when tracking and or monitoring performance are SQL Profiler and Database Engine Tuning Advisor. Only the DBA has permissions to run these two tools. The DBA can analyze the output from these tools and give suggestions on what tables or objects have problems and can benefit from index creation.

Programmatic Recommendations

This section provides recommendations to developers for improving the reliability of procedures and triggers.

Use SET NOCOUNT

When developing procedures that return a result set and use intermediate processing, be sure to set the **SET NOCOUNT ON** option to prevent the generation of unwanted result set count information. Use the **SET NOCOUNT OFF** statement to restore result counting before executing a **SELECT** statement to return results.

Stylistic Recommendations

This section provides recommendations to developers for improving the readability of procedures and triggers.

Nesting IF statements

Avoid, wherever possible, nesting **IF** statements. Use compound Boolean expressions instead.

No more than 4 nested **IF** statements are allowed.

SQL Server 2014 Integration Services

SQL Server Data Tools for Visual Studio 2013 is required when performing extract, transform and load (ETL) processing. Data Tools replaces SQL Server Business Intelligence Development Studio and brings enhancements and improvements to creating and executing packages.

Naming Packages

The name of the SQL Server Integration Services package indicates the database the package belongs to as well as the purpose of the SSIS package. In a development environment, the developer testing the package may own the package but in all other environments SA (System Administrator) owns the package. The package may very well have the same name as the job. The difference between a job and a package is that a job is a scheduled package. A package on its own must be started manually. The package names can be mixed case for ease of readability.

Example: CUSTOMERVendorExport

SQL Server 2014 Reporting Services

SQL Server 2014 Reporting Services (SSRS) is required when creating any reports for use at the DHS.

Naming Reports

The name of the SQL Server 2014Reporting Services object should indicate the database the report belongs to as well as the purpose of the SSRS package. The report names can be mixed case for ease of readability.

Example: CUSTOMERNameAndAddresses

Job Naming Standards

A job consists of a series of SQL Statements executed as a transaction (as one complete unit). The name of a SQL SERVER job indicates the database to which the job belongs as well as the purpose of the job. The job name is the same name as the SSIS package.

SA owns the job in all environments. All jobs need DBA approval prior to being scheduled to run. The job names are mixed case for ease of readability.

Example: ACCOUNTINGImportDailyFiles

Exemptions from this Standard:

There will be no exemptions to this standard.

Refresh Schedule:

All standards and referenced documentation identified in this standard will be subject to review and possible revision annually or upon request by the DHS Information Technology Standards Team.

References:

Microsoft SQL Server 2008/2012/2014 Books On Line

Standard Revision Log:

Change Date	Version	Change Description	Author and Organization
	1.0	New document	Kiley Milakovic
12/11/2007	1.1	SQL Server 2005 revisions	Matt Leitzel
01/13/2011	1.2	SQL Server 2008 revisions	Matt Leitzel
09/15/2016	1.3	SQL Server 2012/2014 revisions	Steve Isleib and Michael Kraus