**MAGNA PROJECT DATABASE CODING STANDARD**

**Bow Valley College**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Revision | Description | Date | Created by | Reviewed By | Approved By |
| Rev A | Issued for Review | May 11, 2020 | Adebayo Adeniji | Adil Rauf | Mauro Domingos |
| Rev B | Issued for Design | May 12, 2020 | Adebayo Adeniji | Adil Rauf | Mauro Domingos |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

**Table of Contents**

[1 Advantages of Database Coding Standard 1](#_Toc40108203)

[2 Naming Conventions 1](#_Toc40108204)

[2.1 Naming Tables: 1](#_Toc40108205)

[2.2 Naming Stored Procedures: 1](#_Toc40108206)

[2.3 Naming Views: 1](#_Toc40108207)

[2.4 Naming Triggers: 1](#_Toc40108208)

[2.5 Naming Indexes: 2](#_Toc40108209)

[2.6 Naming Primary Keys: 2](#_Toc40108210)

[2.7 Naming Foreign Keys: 2](#_Toc40108211)

[2.8 Naming Default Constraints: 2](#_Toc40108212)

[2.9 Naming Reference Columns 2](#_Toc40108213)

[3 Structure 2](#_Toc40108214)

[4 General Formatting 3](#_Toc40108215)

[4.1 SQL keywords 3](#_Toc40108216)

[4.2 Indentations 3](#_Toc40108217)

[4.3 Commenting: 3](#_Toc40108218)

[4.4 Parentheses 4](#_Toc40108219)

[4.5 Spacing / Code Section Separations 4](#_Toc40108220)

[4.6 Expression Spacing 4](#_Toc40108221)

[4.7 ANSI Joins 4](#_Toc40108222)

[4.8 SET Statements 5](#_Toc40108223)

[4.9 Nesting 5](#_Toc40108224)

[5 Coding 5](#_Toc40108225)

# Advantages of Database Coding Standard

Clean code is code that is focused and understandable, which means it must be readable, logical, and changeable. Remember – good code is not the one [computers understand](https://365datascience.com/introduction-machine-learning/); it is the one humans can understand.

# Naming Conventions

## Naming Tables:

Rules:        Pascal notation; end without “s”

Examples:

ProjectInfo

Organization

Note: Do not use spaces in the name of database objects

Do not use SQL keywords as the name of database objects, In cases where this is  necessary, surround the object name with brackets, such as **[Year].**

## Naming Stored Procedures:

Rules:               USP<TableName>\_<Action>\_[By]\_<ColumnName>

Do not prefix stored procedures with ‘**sp\_**’, prefix it with “USP\_”(user defined stored procedure). The prefix sp\_ is reserved for system stored procedures that ship with SQL Server. Whenever SQL Server encounters a procedure name starting with sp\_, it first tries to locate the procedure in the master database, then it looks for any qualifiers (database, owner) provided, then it tries dbo as the owner. Time spent locating the stored procedure can be saved by avoiding the "sp\_" prefix.

Examples:

USP\_ProjectInfo\_Insert

USP\_ProjectInfo\_Delete

USP\_ProjectInfo\_Select\_By\_ProjectID

## Naming Views:

Rules:         VW\_<TableName >\_<Usage/Description>

Examples:

VW\_ ProjectInfo\_Report

## Naming Triggers:

Rules:       TR\_<TableName>\_<action>

Examples:

TR\_ ProjectInfo\_Delete

TR\_ ProjectInfo\_Insert

Note: The use of triggers is discouraged or not advisable unless until that is must.

## Naming Indexes:

Rules:      IX\_<TableName>\_<columns separated by \_>

Examples:

IX\_Interaction\_ InteractionID

## Naming Primary Keys:

Rules:        PK\_<TableName>

Examples:

PK\_ ProjectInfo

PK\_Manager

## Naming Foreign Keys:

Rules:       FK\_<TableName1>

Example:

FK\_ProjectInfo

## Naming Default Constraints:

Rules:      DF\_<TableName>\_<ColumnName>

Example:

DF\_Interaction\_LikelihoodOfProjectApproval

## Naming Reference Columns

Columns assigned as foreign key in a table must be given the same column name assigned in the reference table where it is primary key.

Example:

ProjectInfo table has ProjectID column as primary key. Since interaction table has a relationship with ProjectInfo table via its primary key (ProjectID), Interaction table must have a foreign key column named exactly as ‘ProjectID’.

# Structure

1. Each table must have a primary key
   1. In most cases it should be an IDENTITY column named <TableName>ID

Example:

CREATE TABLE Manager (

ManagerID INT IDENTITY(1,1) PRIMARY KEY,

MFirstName VARCHAR(50),

MLastName VARCHAR(50),

MTitle VARCHAR(50),

);

1. Normalize data to third normal form. Do not compromise on performance to reach third normal form. Sometimes, a little denormalization results in better performance.
2. Do not use TEXT as a data type; use the maximum allowed characters of VARCHAR instead
3. In VARCHAR data columns, do not default to NULL; use an empty string instead.
4. Columns with default values should not allow NULLs.
5. As much as possible, create stored procedures on the same database as the main tables they will be accessing.

# General Formatting

## SQL keywords

Use upper case for all SQL keywords:

* 1. Such as CREATE, SELECT, INSERT, UPDATE, WHERE, AND, OR, LIKE, etc.

## Indentations

Indent code to improve readability:

Any information that goes into a bracket must be dropped one level and increase indent once. Every line instruction after a comma must be dropped one level afterwards until the bracket is closed. The closed bracket must be at the same indent level as the starting bracket i.e. decrease indent once (as shown below).

Example:

CREATE TABLE ProjectInfo(

ProjectInfoID INT NOT NULL,

ProjectName VARCHAR(50)

);

## Commenting:

Every section of code written must be preceded by a comment to explain the intent of that code. Comments shall be used appropriately. Unnecessary/redundant commenting shall be avoided.

1. For single line comments use single-line comment markers (--) as shown below

Example:

--create the project information table

CREATE TABLE Manager(

ManagerID INT IDENTITY(1,1) PRIMARY KEY,

MFirstName VARCHAR(50) NULL,

MLastName VARCHAR(50)

);

1. For multi-line comments, start with /\* and start writing the comments. End the comment section with \*/ and leave a blank line between comment and code as shown below.

Example:

/\*Comment line1 e.g. Create the ProjectInfo information table

Comment line2

Comment line3\*/

CREATE TABLE ProjectInfo(

ProjectInfoID INT NOT NULL,

ProjectName VARCHAR(50)

);

## Parentheses

Apply parentheses to increase readability.

Example:

SELECT MFirstName

FROM Manager

WHERE (MFirstName = ’Jack’ AND (BusinessUnitID = 1 OR BusinessUnitID = 2))

## Spacing / Code Section Separations

Introduce proper spacing between sections. Use two blank lines to separate code sections.

## Expression Spacing

Use spaces so that expressions read like sentences.

Example:

MFirstName = ’Jack’, not MFirstName=’Jack’

## ANSI Joins

Format JOIN operations using indents. Also, use ANSI Joins instead of old style joins.

Example:

**False code:**

SELECT column name1, column name2, ….

FROM Table1, Table2

WHERE Table1.d = Table2.c

**True code:**

SELECT column name1, column name2, ….

FROM Table1

INNER JOIN Table2 ON Table1.d = Table2.c

## SET Statements

Place SET statements before any executing code in the procedure.

## Nesting

Avoid unnecessary nested codes.

# Coding

1. Optimize queries using the tools provided by SQL Server.

Use the graphical execution plan in Query Analyzer or SHOWPLAN\_TEXT or SHOWPLAN\_ALL commands to analyze your queries. Make sure your queries do an "Index seek" instead of an "Index scan" or a "Table scan." A table scan or an index scan is a highly undesirable and should be avoided where possible.

1. Complexity - Code shall be writing and arrange in a neat and clear manner with less complexities.
2. Do not use SELECT \* to select all fields. Instead write required column names after the SELECT statement.

Example:

SELECT MFirstName, MLastName, MTitle

FROM Manager

1. Attention should be given to NOT NULL or NULL constraints. Data that is never NULL should have a NOT NULL constraint.
2. Return multiple result sets from one stored procedure to avoid trips from the application server to SQL server.
3. Avoid unnecessary use of temporary tables. Use 'Derived tables' or CTE (Common Table Expressions) wherever possible, as they perform better.
4. Avoid using <> as a comparison operator. Use ProjectID IN (1,3,4,5) instead of ProjectID <> 2
5. Use SET NOCOUNT ON at the beginning of stored procedures.

This suppresses messages like '(1 row(s) affected)' after executing INSERT, UPDATE, DELETE and SELECT statements. Performance is improved due to the reduction of network traffic.

1. Do not use cursors or application loops to do inserts. Instead, use INSERT INTO.
2. Fully qualify tables and column names in JOINs.
3. Fully qualify all stored procedure and table references in stored procedures. Modifications to stored procedures should be briefly documented within the procedure if changed after release for DEMO testing.
4. Do not define default values for parameters. If a default is needed, the front end will supply the value.
5. Do not use the RECOMPILE option for stored procedures.
6. Place all DECLARE statements before any other code in the procedure.
7. Do not use column numbers in the ORDER BY clause.
8. Do not use GOTO.
9. Check the global variable @@ERROR immediately after executing a data manipulation statement (like INSERT/UPDATE/DELETE), so that you can rollback the transaction if an error occurs. Or use TRY/CATCH.
10. Do basic validations in the front-end itself during data entry.
11. Off-load tasks, like string manipulations, concatenations, row numbering, case conversions, type conversions etc., to the front-end applications if these operations are going to consume more CPU cycles on the database server.
12. Always use a column list in your INSERT statements. This helps avoid problems when the table structure changes (like adding or dropping a column).
13. Minimize the use of NULLs, as they often confuse front-end applications, unless the applications are coded intelligently to eliminate NULLs or convert the NULLs into some other form. Note that any expression that deals with NULL results in a NULL output.
14. The ISNULL and COALESCE functions are helpful in dealing with NULL values.
15. Avoid the use of cross joins, if possible.
16. When executing an UPDATE or DELETE statement, use the primary key in the WHERE condition, if possible. This reduces error possibilities.

Example:

DELETE FROM ProjectInfo

WHERE ProjectID = 1

1. Avoid using TEXT or NTEXT datatypes for storing large textual data.
   1. Use the maximum allowed characters of VARCHAR instead

You cannot directly write or update text data using the INSERT or UPDATE statements. Instead, you have to use special statements like READTEXT, WRITETEXT and UPDATETEXT. So, if you don't have to store more than 8KB of text, use the CHAR(8000) or VARCHAR(8000) datatype instead.

1. Avoid dynamic SQL statements as much as possible.
2. Dynamic SQL tends to be slower than static SQL, as SQL Server must generate an execution plan at runtime. IF and CASE statements come in handy to avoid dynamic SQL.
3. Access tables in the same order in your stored procedures and triggers consistently. This helps to avoid deadlocks. Other things to keep in mind to avoid deadlocks are:
4. Keep transactions as short as possible.
5. Touch the minimum amount of data possible during a transaction.
6. Never wait for user input in the middle of a transaction.
7. Do not use higher level locking hints or restrictive isolation levels unless they are absolutely needed.
8. Do not call functions repeatedly within your stored procedures, triggers, functions and batches.
9. You might need the length of a string variable in many places of your procedure, but don't call the LEN function whenever it's needed. Instead, call the LEN function once and store the result in a variable for later use.
10. Default constraints must be defined at the column level.
11. Avoid wild-card characters at the beginning of a word while searching using the LIKE keyword, as these results in an index scan, which defeats the purpose of an index.
12. Define all constraints, other than defaults, at the table level.
13. When a result set is not needed, use syntax that does not return a result set.
14. Avoid rules, database level defaults that must be bound or user-defined data types. While these are legitimate database constructs, opt for constraints and column defaults to hold the database consistent for development and conversion coding.
15. Constraints that apply to more than one column must be defined at the table level.
16. Use the CHAR data type for a column only when the column is non-nullable.
17. CHAR(100), when NULL, will consume 100 bytes, resulting in space wastage. Preferably, use VARCHAR(100) in this situation. Variable-length columns have very little processing overhead compared with fixed-length columns.
18. Do not use white space in identifiers.
19. The RETURN statement is meant for returning the execution status only, but not data.
20. Unused part of the code shall be removed, either identifiers/variables or part of code.
21. Redundancy- Remove redundant codes and extra spaces within the code.
22. Flexibility - Code shall be flexible enough which can be easily modified to handle larger system and new environments.