SOFTWARE ENGINEERING STANDARDS & PRACTICES

SQL Server Coding Standards

9AAD134842

Department	GF-IS ADM Applications Performance Excellence (APE)
Approver	Giulio Bitella, Global Department Manager
Owner	Tomasz Jastrzębski, Global Leader for Software Engineering (SE)

For the latest distributable version of this and other Software Engineering standards please visit this link to ABB Library.

WHAT IS THIS?

This document contains SQL Server programming standards.

WHAT IS THE PURPOSE OF THIS DOCUMENT?

The purpose of the document is to establish basic SQL Server programming standards, practices and guidelines.

TABLE OF CONTENTS

DATABASE DESIGN

1. SQL Server is highly sophisticated product and should be used to its full potential, rather just as a simple table store.
2. Pay attention to proper database normalization. OLTP database should satisfy at least Third Normal Form (3NF) – see references for details.
3. Do not mix OLTP and OLAP tables in the same database schema.
4. SQL Server is multi user server, always consider other users may try to modify the same data the same time and use transactions whenever appropriate. Within transaction lock as little data as possible, particularly in long-lasting transactions using large datasets where transaction can block multiple other transactions. Use appropriate transaction Isolation Level and Lock Level (e.g. `tablock` vs. `rowlock`, vs. `nolock`).
   (ref. Table Hints, Transaction Isolation Levels)
5. Make sure transactions follow ACID (Atomicity, Consistency, Isolation, Durability) principle.
6. Consider using stored procedures for time sensitive routines which would otherwise require numerous database call and round trips. Use of temporary, especially in-memory tables within stored procedures is encouraged.
7. To ensure data integrity use table relations (foreign keys) whenever appropriate. Whenever feasible, use the same column name as primary and foreign key column. E.g. `Customers.CustomerId` ↔ `Orders.CustomerId`.
8. Consider using table/view triggers and relationships (foreign keys) with "`on delete/update cascade`" option. Such approach greatly reduces number of required database calls and, albeit particular database operation takes longer, overall system performance is increased.
9. Building triggers always bear in mind trigger is fired just once for multiple inserted, deleted or updated records.

SECURITY

1. Grant database object permissions to roles, rather than particular users. Define database roles matching access needs. In simple cases use bulit-in roles like: `db_datareader`, `db_datawriter`, `db_denydatawriter`.
2. Always follow "the least privilege principle". Under no circumstances application or its middle layer should access database as administrator or database owner (`db_owner`).
3. Define database users using "contained user" model without server login for better database portability, rather than traditional model.

4. Avoid using passwords for authentication. Use Managed Service Identity/Principal and Azure AD (for Azure PaaS) or certificates and Azure AD (for IaaS).

DATA TYPES

1. Data types used in data store should match types used in higher application layers as closely as possible.
2. Avoid using `varchar(max)`, `nvarchar(max)` and similar without real need to store very large amounts of data.
3. Avoid using Unicode `nvarchar` and `nchar` types where ASCII `varchar` and `char` types are sufficient.
4. Do not use datetime type. Use `datetime2()` instead, typically `datetime2(0)` – no second fraction.
5. Storing `date` and `datetime2` use UTC date/time. If timezone information is required use `datetimeoffset` type. If column always contains date/datetime other than UTC add timezone code as column name suffix. Example: name field containing CET timezone `DateCreated`**`Cet`**.
6. Avoid using float point numerical types (`real`, `float`) unless stored values really are float point. Use `decimal` type instead. Always use `decimal` type to store cost, price, quantity and other values of defined/limited precision.

NULL VALUES

1. Allow for NULL values in columns only when NULL values are required.
2. Typically in `text`, `ntext`, `char`, `nchar`, `varchar`, `nvarchar` columns NULL values are not required and empty string can be used as a default value.

INDEXES

1. Define table indexes wisely as they slow down insert/update operations.
2. Field sequence in indexes does matter. Typically, the most specific fields should be specified first. Example: `LastName`, `FirstName` (in this order).
3. Make informed decision whether to use the clustered index as table primary key PK. It is often the best choice, however in cases of e.g. sequential range scans, keeping often scanned ranges in clusters may be by far the best choice performance-wise.
   Example: `OrderDetails` records are usually read by a given order. Therefore it may be a good idea to create clustered, non-unique index on `OrderId` column, rather than `OrderDetailId` (PK).

NAMING CONVENTIONS

1. Use lowercase for all keywords to provide similar coding experience for full-stack developers.

2. Literals (table names, column names etc., stored procedure and function public parameters) should follow Pascal casing, local procedure/function parameters: camel casing. It is especially important for DB-first approach that literals follow C# Coding Standards. (ref. 9AAD134036)
3. Use plural noun form for table names. E.g. `Sessions`.
4. Record ID field (usually primary key) should be named "`Id`".
5. Index names should consist of:
   a. `IX_` or `PK_` prefix for primary key indexes
   b. Table name
   c. Column names separated by "`-`"
   Example: `PK_Sessions`, `IX_Sessions_TimeStart-TimeFinish`
6. View names should contain plural noun and end with "`View`" suffix.
   Example: `ActiveUsersView`
7. Trigger names should consist of table name and trigger type.
   Example: `Users_InsertTigger`
8. No prefixes or suffixes should be used for functions or stored procedures.

CODING STYLE

1. Stored procedure, view, trigger, function header should contain copyright and description.
2. Every "`select`" statement clause (`select`, `from`, `where`, `order by`) on a separate line. Exception: derived tables with short definition.
3. Long predicates in "`where`" clause should be placed on a separate line.
4. Use Inline comments wherever it is helpful and applicable.
5. Indent content of control blocks (statement block). Place if/while and begin on the same line. Example:
```
if @isActive = 1 begin
   […]
end else begin
   […]
end
```
6. Indent content of transaction block (statement block), place `begin transaction` and `commit` on separate line.
7. Table/view/function schema must be always provided (e.g. "`dbo`") – it is very important for performance since otherwise SQL Server is unable to share execution plans between sessions and requests.
8. Do not select all the columns in views, stored procedures or queries using `select * […].` Explicit field list is always required. Adding a new column or changing table/view column order should not be a breaking change.
9. Column names in views, stored procedures, triggers and functions should be prefixed with table alias. Example: `s.SessionId`

10. Long calculated fields should be placed on a separate line starting with field name. Example:
    `[Minutes] = datediff(mi, s.TimeStart, s.TimeFinish)`
11. Reserved names used as field name should be placed in square brackets, so they do not get highlighted in source code editors.

DATABASE ACCESS

For SQL Server database access guidelines refer to:

1. Best Coding Practices - 9AAD135446
2. C# Best Practices - 9AAD134037
3. Java Coding Standards - 9AAD135383

SAMPLE CODE - VIEW

```
-- Copyright © ABB Inc. 2018
-- Returns session list with details.
create view dbo.SessionsView
as
select top 100 percent s.SessionId, u.LoginName, s.TimeStart, s.TimeFinish, c.ParticipantCount,
   [Minutes] = datediff(mi, s.TimeStart, s.TimeFinish)
from dbo.[Sessions] s
inner join Users u on u.UserId = s.HostUserId
-- include session participants count (c)
left join (
   select su.SessionId, count(0) ParticipantCount
   from dbo.SessionUsers su
   group by su.SessionId
   ) c on c.SessionId = s.SessionId
where
   -- exclude sessions hosted by test users
   u.LoginName not like 'testuser%'
   -- exclude deleted sessions
   and s.IsDeleted = 0
order by s.TimeStart
```

SAMPLE CODE – STORED PROCEDURE

```
-- Author: Tomasz Jastrzębski, 2008
-- CreativeCommons CC0 Universal License
-- Procedure defragments table indexes.
create procedure dbo.DefragmentIndexes
   @Table nvarchar(128),
   @IndexCount int = null out,
   @DefragCount int = null out
as

set nocount on;

declare row_cursor cursor fast_forward for
   select [name], avg_fragmentation_in_percent
   from sys.dm_db_index_physical_stats(db_id(), object_id(@Table), null, null, null) as a
   inner join sys.indexes as b on a.object_id = b.object_id and a.index_id = b.index_id

declare @name sysname = null;
declare @fragmentation float = 0;

set @IndexCount = 0
set @DefragCount = 0

open row_cursor;
fetch next from row_cursor into @name, @fragmentation;

while @@fetch_status = 0 begin
   set @IndexCount = @IndexCount + 1

   if @fragmentation <= 5 begin
      print @name + ' fragmented ' + cast(@fragmentation as varchar) + '% - skipping';
   end

   if @fragmentation > 5 and @fragmentation <= 30 begin
      print @name + ' fragmented ' + cast(@fragmentation as varchar) + '% - reorganizing';
      set @DefragCount = @DefragCount + 1
      exec (N'alter index [' + @name + '] on ' + @Table + ' reorganize;');
   end

   if @fragmentation > 30 begin
      print @name + ' fragmented ' + cast(@fragmentation as varchar) + '% - rebuilding';
      set @DefragCount = @DefragCount + 1
      exec (N'alter index [' + @name + '] on ' + @Table + ' rebuild;');
   end

   fetch next from row_cursor into @name, @fragmentation;
end

close row_cursor;
deallocate row_cursor;
```

REFERENCES

1. Transaction Isolation Levels
   https://docs.microsoft.com/en-us/sql/database-engine/transaction-isolation-levels?view=sql-server-2014
2. Table Hints
   https://docs.microsoft.com/en-us/sql/t-sql/queries/hints-transact-sql-table?view=sql-server-2017
3. Index Architecture and Design Guide
   https://docs.microsoft.com/en-us/sql/relational-databases/sql-server-index-design-guide?view=sql-server-2017
4. Database normalization
   https://en.wikipedia.org/wiki/Database_normalization
5. IS Standard for Managing Transactional Data - 9AAD135312
6. Best Coding Practices - 9AAD135446
7. C# Best Coding Practices - 9AAD134037
8. C# Coding Standards - 9AAD134036
9. Java Coding Standards - 9AAD135383
10. Source Code Management Standards - 9AAD134843

REVISION HISTORY

Rev.	Page	Change Description	Author(s)	Date
A	all	approved	Tomasz Jastrzębski et al.	2019-04-09

Dev

Information Systems

Applications Performance Excellence Department (APE)

Software Engineering Standards & Practices