Database Standards & Naming Conventions

Table of Content

[1 Introduction 4](#_Toc376860110)

[2 Design Guidelines 4](#_Toc376860111)

[3 Programming Guidelines 5](#_Toc376860112)

[4 Naming Conventions 6](#_Toc376860113)

[5 General Guidelines 8](#_Toc376860114)

# Introduction

These standards are meant to be guidelines on development of software. They are not the ends all be all, this document is living and will grow and change as our development environment does. The overriding goal is to provide consistency. If there is a rule that does not fit with what must be accomplished functionality-wise, ask your team leader for advice. You may find you have found an exception to the rule and they will tell you to document the exception and move on. These rules are not here to hamper you creativity, but to enhance your productivity by providing consistency. How many times have you gotten into legacy code and had to take hours or days just to figure out what it does? These standards will hopefully help to reduce that time.

# Design Guidelines

* Make sure you normalize your data at least till 3rd normal form. At the same time, do not compromise on query performance. A little bit of de-normalization helps queries perform faster.
* A primary key is used to enforce entity integrity. Entity integrity is the very basic concept that every row is uniquely identifiable. Use appropriate data types and strive to keep the row size smaller to accommodate more rows in a page. It is recommended that the primary key should always be a numeric data type (INT, BIGINT) and never a GUID (UNIQUEIDENTIFIER data type).
* While designing your database, design it keeping 'performance' in mind. You can't really tune performance later, when your database is in production, as it involves rebuilding tables/indexes, re-writing queries. Use the graphical execution plan in Query Analyzer or SHOWPLAN\_TEXT or SHOWPLAN\_ALL commands to analyze your queries. Make sure your queries do 'Index seeks' instead of 'Index scans' or 'Table scans'. A table scan or an index scan is a very bad thing and should be avoided where possible (sometimes when the table is too small or when the whole table needs to be processed, the optimizer will choose a table or index scan).
* Perform all your referential integrity checks, data validations using constraints (foreign key and check constraints).

# Programming Guidelines

* To make SQL Statements more readable, start each clause on a new line and indent when needed. Use all reserved words in UPPER case.
* Do not use SELECT \* in your queries. Always write the required column names after the SELECT statement, like SELECT CustomerID, CustomerFirstName, City. This technique results in less disk IO and less network traffic and hence better performance.
* Try to avoid server side cursors as much as possible. Always stick to 'set based approach' instead of a 'procedural approach' for accessing/manipulating data. Cursors can be easily avoided by SELECT statements in many cases. If a cursor is unavoidable, use a simple WHILE loop instead, to loop through the table. A WHILE loop is faster than a cursor most of the times. But for a WHILE loop to replace a cursor you need a column (primary key or unique key) to identify each row uniquely.
* Avoid the creation of temporary tables while processing data, as much as possible, as creating a temporary table means more disk IO. Consider advanced SQL or views or table variables or derived tables, instead of temporary tables. Keep in mind that, in some cases, using a temporary table performs better than a highly complicated query.
* Use SET NOCOUNT ON at the beginning of your SQL batches, stored procedures and triggers in production environments, as this suppresses messages like '(1 row(s) affected)' after executing INSERT, UPDATE, DELETE and SELECT statements. This in turn improves the performance of the stored procedures by reducing the network traffic.
* Use the more readable ANSI-Standard Join clauses instead of the old style joins.
* Avoid dynamic SQL statements as much as possible. Dynamic SQL tends to be slower than static SQL, as SQL Server must generate an execution plan every time at runtime.
* Minimize the usage of NULLs, as they often confuse the front-end applications, unless the applications are coded intelligently to eliminate NULLs or convert the NULLs into some other form. Any expression that deals with NULL results in a NULL output. NVL, ISNULL and COALESCE functions are helpful in dealing with NULL values.
* Though T-SQL has no concept of constants (like the ones in C language), variables will serve the same purpose. Using variables instead of constant values within your SQL statements, improves readability and maintainability of your code.
* Use Unicode data types like nchar, nvarchar, ntext, if your database is going to store not just plain English characters, but a variety of characters used all over the world. Use these data types, only when they are absolutely needed as they need twice as much space as non-Unicode data types.
* To avoid deadloacks always access tables in the same order in all your stored procedures/triggers consistently.
* Make sure your stored procedures always return a value indicating the status. Standardize on the return values of stored procedures for success and failures. The RETURN statement is meant for returning the execution status only, but not data. If you need to return data, use OUTPUT parameters.
* If your stored procedure always returns a single row resultset, consider returning the resultset using OUTPUT parameters instead of a SELECT statement, as ADO.NET handles output parameters faster than resultsets returned by SELECT statements.
* Always check the global variable @@ERROR immediately after executing a data manipulation statement so that appropriate actions could be taken in case of an error. For databases like Oracle try to include EXCEPTION section to log all the exceptions.
* Do not put any business logic and complex processing in stored procedures. Remember it is difficult to add database servers but is relatively easy to add web/application servers.
* For better SELECT performance, index your JOIN columns
* When defining the table columns, make sure you correctly identify which fields require data and which do not. Don't set all your fields to not null, if you don't want to force users to enter data for them. On the other hand, defining your constraints very loosely or not defining any constraints at all will make your application vulnerable and ruin your data integrity
* Try not to use system tables directly as much as possible. System table structures may change in a future release. Wherever possible, use the sp\_help\* stored procedures or INFORMATION\_SCHEMA views.

# Naming Conventions

* Do not use numbers in the name. This is proof of poor design, indicating a badly divided table structure. If you need a many-to-many relation, the best way to achieve it is by using a separate linking table. Moreover, using numbers to differentiate between two columns that store similar information might be an indication that you need an extra table, storing that information.
* Do not use the dot (.) as a separator in names. This way you will avoid problems when trying to perform queries, as the dot is used to identify a field in a specific column.
* Do not use any of the reserved words as names of database elements. Each database language uses some words as names for internal functions, or as part of the SQL syntax. For instance, using order as the name of a table that stores product orders from an online shop is bad practice, because order is also used in SQL language to sort records (ascending or descending).
* When naming the elements, do not use long or awkward names. Keep them as simple as you can, while maintaining a clear meaning. It's also a good idea to use names which are close to the natural language

|  |  |  |  |
| --- | --- | --- | --- |
| Object | Convention | Example | Comments |
| Database | <database name> | Inventory | * Use the name of the product or application as the database name |
| Data Device | <database name>Data | InventoryData |  |
| Log Device | <database name>Log | InventoryLog |  |
| Table | <table name> | Product | * Use Noun * Use PascalCasing * Table name should be singular * Do not use reserve words |
| Column | <column name> | PoductName | * Use Noun * Use PascalCasing * Do not use reserved words |
| View | <view name> | TitleAuthor  YearlySalesSummary | * Use name of the tables involved OR * Use the name to indicate the data |
| Stored Procedure | usp[<group name>]<procedure name> | uspListProducts  uspInsertProduct  uspUpdateProduct  uspDeleteProduct | * Use a verb along with the table name * Optionally use a logical group name |
| Trigger | <type><table name> | InsertProduct  UpdateProduct  DeleteProduct | * Use PascalCasing |
| User Defined Type | udf<type name> | udfAmount | * Use PascalCasing for type name * The type name should not contain the name or length of the underlying data type |
| User Defined Function | fn<function name> | fnSplit | * Use a verb or verb phrase * Use PascalCasing |
| Primary Key Constraint | pk<table name><column name(s)> | pkProductProductID |  |
| Foreign Key Constraint | fk<table name><key table name><key name> | fkProductProductTypeProductTypeID |  |
| Unique Constraint | u<table name><column name> | uProductProductName |  |
| Check Constraint | chk<table name><column name> | chkProduct\_ProductType |  |
| Default Constraint | def<table name><column name> | defProductDateCreated |  |
| Index | Idx<type><table name> | idxnProductName | Use following types:  c = clustered  n = non clustered  u = unique |
| Fact | Fact<business group><table name> | FactBIProductSummary |  |
| Dimension | Dim<table name> | DimProduct |  |

# General Guidelines

Use following style to comment the stored procedures, triggers

**/**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Name of Procedure : clcRepProcChecksWithoutTrans

Author : SR <Initials of the developer>

Date of Creation : July 22, 2002 <Date on which it has been created>

Date of Modification : November 11,2002 2002 <Date on which it has been last updated>

Input Parameters :

1:@Param1 varchar(10) ->Date range start

2:@ Param2 varchar(10) -> Date range end

3:@ Param3 integer ->Control Id range start

Description : <Detailed description for procedure including purpose>

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/