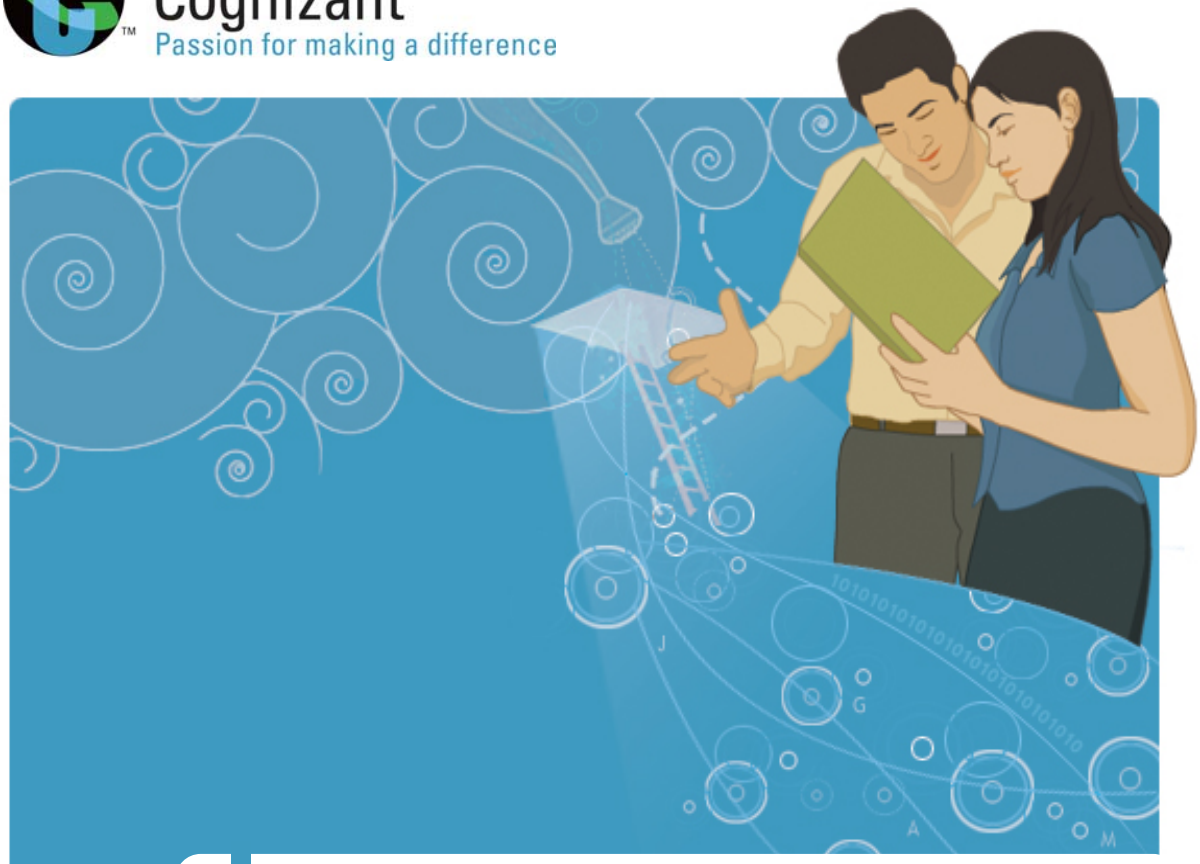
Cognizant
Passion for making a difference



Coding Standards: SQL Server 2005

Version: SQLSERVER2005/Coding Standards/0408/1.0
Date: 30-04-08

Cognizant
500 Glen Pointe Center West
Teaneck, NJ 07666
Ph: 201-801-0233
www.cognizant.com

TABLE OF CONTENTS

1. Naming Conventions

Tables

Rules: Pascal notation; end with a 's'
Examples: Products, Customers
Group related table names1

Stored Procedures

Rules: `sp<App Name>_[<Group Name >_]<Action><table/logical instance>`
Examples: `spOrders_GetNewOrders`, `spProducts_UpdateProduct`

Triggers

Rules: `TR_<TableName>_<action>`
Examples: `TR_Orders_UpdateProducts`

Indexes

Rules: `IX_<TableName>_<columns separated by _>`
Examples: `IX_Products_ProductID`

Primary Keys

Rules: `PK_<TableName>`
Examples: `PK_Products`

Foreign Keys

Rules: `FK_<TableName1>_<TableName2>`
Example: `FK_Products_Orderss`

Defaults

Rules: `DF_<TableName>_<ColumnName>`
Example: `DF_Products_Quantity`

Columns

If a column references another table's column, then name it `<table name>ID`
Example: The `Customers` table has an `ID` column.
The `Orders` table should have a `CustomerID` column.

General Rules

- 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].
- Do not prefix stored procedures with 'sp_'2.
- Prefix table names with the owner names.

Structure

- Each table must have a primary key
- In most cases it should be an IDENTITY column named `ID`
- Normalize data to third normal form
- Do not compromise on performance to reach third normal form. Sometimes, a little denormalization results in better performance
- Do not use TEXT as a data type; use the maximum allowed characters of VARCHAR.
- In VARCHAR data columns, do not default to `NULL`, use an empty string instead
- Columns with default values should not allow `NULL`s
- As much as possible, create stored procedures on the same database as the main tables they will be accessing

Variables and Parameters

- All variables should be explicitly declared at the top of each stored procedure. Place each declaration on its own line. Explain what each parameter is for:
  - **`DECLARE @lCaseStart integer:`** Starting case number
  - **`DECLARE @dStartDate datetime:`** First posted date of cases
  - **`DECLARE @dEndDate datetime:`** Last posted date of cases
  - **`DECLARE @lCaseCount integer:`** Running total of cases
- Do not make a variable name that begins with '@@'. Although it is a legitimate thing to do, the '@@' is typically reserved for SQL Server global variables. This makes readers confused in thinking a system variable when it is not.
  - **`DECLARE @sGoodName varchar(50):`** Good name
  - **`DECLARE @@sBadName varchar(50):`** Bad name
- Never name a variable name by its data type or by a reserved word. Use a valid business explanation of what the variable name is for.
  - **`DECLARE @dCaseOpen datetime:`** Date the case opened
  - **`DECLARE @datetime datetime:`** Bad naming
- A prefix for variables should contain the data type prefix, followed by what the variable or parameter is for. For user-defined types, use the prefix where the type came from acceptable prefixes:
  - `I:` integer, smallint, tinyint
  - `S:` char, varchar
  - `D:` datetime, smalldatetime
  - `C:` money, smallmoney

- `B:` bit, binary, varbinary
- `F:` float, real
- `n:` decimal, numeric

User-Defined functions

A user defined function should prefixed by "udf_" and then a description that logically follows the function process. The description should be proper case words with no spaces.

Examples:

- `udf_GetNextID`
- `udf_SumOrderLines`

Others

All the database objects like tables, stored procedures, views, functions and so on should be referenced in their fully qualified name like `[database].[owner].[ObjectName]. ]`

Example:

`Sales.dbo.SearchCustomer Sales.dbo.Customer`. If database name is not fixed, then use `[owner].[ObjectName]`.

2. T-SQL Guidelines

T-SQL Guidelines

Retrieve only the data that is required

- A SELECT statement should retrieve only data that is necessary
- Columns that are not used or already known should not be returned
- SELECT * is not a good coding practice as it:
  - Increases network traffic
  - Requires more buffers and processing
  - Could prove error prone, if the table or view definition changes

Minimize the use of not equal operations, <> or !=.

- SQL Server has to scan a table or index to find all values to see if they are not equal to the value given in the expression.
- Try rephrasing the expression to use ranges.

Example:

Use this:

```
SELECT Name, Description, OrderDate
FROM Orders
WHERE OrderDate < ‘2005-01-01’ OR OrderDate > ‘2005-01-01’
```

Instead of this:

```
SELECT Name, Description, OrderDate
FROM Orders
WHERE OrderDate <> ‘2005-01-01’
```

Make sure that there are at least n-1 join criteria if there are n tables.

Make sure that all tables included in the statement are joined. Make sure that only tables that

- Have columns in the select clause
- Have columns referenced in the where clause
- Allow two unrelated tables to be joined together are included

Ensure columns with similar data type are used when using JOINS

- Performance suffers when non-identical columns are compared.
- SQL Server needs an implicit conversion between the two types.
- Performance is better when the tables being joined are defined with identical types.
- This prevents implicit data conversions by the query engine.

Always use a column list in your INSERT statements.

- This helps in avoiding problems when the table structure changes (like adding or dropping a column).

Use sub query instead of a temporary table, unless strictly necessary.

- There are alternatives like the TABLE variable data type which can provide in-memory solutions for small tables inside stored procedures too.

Avoid Wildcard Search

- Avoid wildcards in search as much as possible.
- If not, then try to avoid wildcard characters at the beginning of a word while searching using the `LIKE` keyword.

Example:

Use this:

```
SELECT LocationID FROM Locations WHERE Specialities LIKE 'A%s'
```

Instead of this:

```
SELECT LocationID FROM Locations WHERE Specialities LIKE
'%pples'
```

Use 'Derived tables' wherever possible, as they perform better

Example:

Use this:

```
SELECT MIN(Salary)
FROM
(
SELECT TOP 2 Salary
FROM Employees
ORDER BY Salary DESC
) AS A
```

Instead of this:

```
SELECT MIN(Salary)
FROM Employees
WHERE EmpID IN
(
SELECT TOP 2 EmpID
FROM Employees
ORDER BY Salary Desc
)
```

Prefix the tables, views and SP names with the owner's name

- This improves readability and avoids any unnecessary confusion.
- Qualifying table names with owner names helps in execution plan reuse, which further boosts performance.
- Lack of owner-qualification can introduce delays in stored procedure execution and unnecessarily high CPU utilization since it forces SQL to perform a second cache lookup and acquire an exclusive compile lock before determining that the existing cached execution plan can be reused.
- It is, however not necessary to qualify the stored proc with the database name to prevent the additional cache lookup.

Use EXISTS clause instead of Count(*) for checking existence

- Use `EXISTS` instead of `COUNT(*)` when looking for the existence of one or more rows in a subquery. `EXISTS` cancels the subquery once the first existence of a record is found, while `COUNT(*)` forces processing of the entire query.

Do not use column numbers in the ORDER BY clause.

- Consider the following example in which the first query is more readable than the second one:

Example:

Use this

```
SELECT OrderID, OrderDate
FROM Orders
ORDER BY OrderDate
```

Instead of this

```
SELECT OrderID, OrderDate
FROM Orders
ORDER BY 2
```

Use UNION ALL over UNION wherever possible

- Use `UNION ALL` over `UNION` when the `SELECT` statement does not produce duplicate rows. `UNION` does an additional distinct operation over the accumulated result set which can be avoided in certain cases beforehand
- A `UNION` (without the ALL) performs an internal sorting to eliminate duplicate rows, thus creating a worktable hindering the performance

Do not use ORDER BY unless there is a requirement

- `ORDER BY` uses resources
- Try to `ORDER BY` the indexed columns to improve performance

Use SQLCLR stored procedures, functions, triggers with care

- Use `SQLCLR` stored procedures only for computational intensive algorithms
- Avoid calling a `SQLCLR` function for computations that can be performed in T-SQL

- Avoid using SQLCLR for set based computation. T-SQL is built for that and performs much faster
- Try to avoid SQLCLR User defined aggregates if possible. It uses and disposes lots of objects in the aggregation process and might hit the performance during high concurrency

Common Table Expressions versus CROSS APPLY in hierarchy generation

- It is best to go for a Common Table Expressions if the joining columns of the self join are indexed.
- If index cannot be created to the joining columns then its better to use `CROSS APPLY`.
- `CROSS APPLY` does not use indexes. So it's best to avoid this operator when indexes can be utilized in the query.
- `CROSS APPLY` is best suited if the number of rows returned is less and iteration depth < 5.

Stored Procedures (SPs)

Use Stored Procedures (SPs) wherever you can.

- Execution plans may be cached for improved performance
- Separate the business logic from presentation logic
- Limit multiple network calls by encapsulating complex business logic

Views

Do not use indexed views unless you have one of the following scenarios:

- Joins and aggregations of large tables
- Repeated patterns of queries
- Repeated aggregations on the same or overlapping sets of columns
- Repeated joins of the same tables on the same keys
- Combinations of the above
- The query takes more than a few seconds to run. The index in the view is ignored unless SQL Server anticipates a real performance hit

Transaction Handling

- Keep transaction boundary short.
- Avoid nested transactions as much as possible.
- Do not use transactions inside stored procedures unless the stored procedure directly or indirectly executes more than one insert, update or delete statement.
- While using transactions inside stored procedures, name your transactions.
- Before exiting the stored procedure you must commit or rollback any transactions you began. Do not use implicit transactions ("`SET IMPLICIT_TRANSACTIONS ON`").

CLR Integration guidelines

- The CLR stored procedures are efficient when the vector operations or complex mathematical operations are to be performed. For set based operations T-SQL is the best.
- Avoid data access logic inside CLR procedures.
- When there is a need to use extended stored procedures, it is better to go for CLR procedures as an alternative.

XML Usage Guidelines

Usage of OPENXML

Stored procedures have the ability to accept XML and parse it into individual insert or update statements. This XML can contain many rows across many tables to affect in one document. This provides a significant performance increase because you only have the overhead of one network round trip for the entire commit rather than one round trip per modification to the database. However it is not the right choice for all kinds of applications. Some guidelines on the usage of OPENXML are given as follows:

- OPENXML is not recommended for online transactions with large number of concurrent users. This is typically recommended for batch processes where you may need to process and send large number of records for update into the database
- It is recommended to keep the size of the XML small and not process a huge XML document as this puts the SQL Server memory under pressure due to the recursive nature of processing the XML. Typical tests show that it is better to keep it under 300K, but this number can vary based on the hardware and usage scenarios
- Bulk DML operations using OPENXML is not recommended if the requirement is to track individual rows for failure or success
- Attribute centric XML structure is preferred since the size is small

XML or Relational Storage

Follow these guidelines when deciding the choice of storage (XML or relational)

- If your data is highly structured with known schema, then the relational model is likely to work best for data storage.
- XML is a good choice if you want a platform-independent model in order to ensure portability of the data by using structural and semantic markup. Additionally, it is an appropriate option if some of the following properties are satisfied:
  - Your data is sparse or you do not know the structure of the data, or the structure of your data may change significantly in the future.
  - Your data represents containment hierarchy, instead of references among entities, and may be recursive.
  - Order is inherent in your data.
  - You want to query into the data or update parts of it, based on its structure.

References

Websites

Cognizant Coding Standards / Guidelines can be accessed from the site:

- https://dotnet/Developercenter/Site%20Pages/Templates.aspx?RootFolder=%2fDevelopercenter%2fTemplatesGuidelines%2fDevelopment%20Guidelines&FolderCTID=&View=%7bBEA05050%2d5EEE%2d4896%2d97E5%2d6C3797634C8A%7d

Others URLs:

- http://blogs.msdn.com/steven_bates/archive/2006/03/20/Database-Object-Naming-Rules.aspx
- http://www.SQLAuthority.com

STUDENT NOTES: