Coding Standards Document - Database

I. Naming

I. Tables:
Rules: <App Name>_<TableName>
Table name in Pascal Case ending with an 's'
Table Names should be nouns
Examples: Census_Products, Customers

II. Stored Procs:
Rules: <App Name>_<Method Name >_<Action>

a. There should not be any underscore (_) in the method name
b. Do not prefix stored procedures with 'sp_'

Examples
Correct :
Census_AnimalDetails_Get
Census_NewOrders_Get,
OSP_Product_Update
InCorrect:
Census_Animal_Details_Get

III. Triggers:
Rules: TR_<TableName>_<action>
Examples
TR_Orders_Update
Notes: The use of triggers is discouraged

IV. Indexes:
Rules: IX_<TableName>_<columns separated by _>
Examples: IX_Products_ProductID

V. Primary Keys:
Rules: PK_<TableName>
Examples: PK_Products

VI. Foreign Keys:
Rules: FK_<TableName1>_<TableName2>
Example: FK_Products_Orders

VII. Defaults:
Rules: DF_<TableName>_<ColumnName>
Example: DF_Products_Quantity

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

IX. General Rules:

a. Do not use spaces in the name of database objects

b. Do not use SQL keywords as the name of database objects
c. Table Names should be in Pascal Case
d. Parameter Names should be in Pascal Case

II. Structure

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

III. Formatting

I. Use upper case for all SQL keywords, SELECT, INSERT, UPDATE, WHERE, AND, OR, LIKE, etc.
II. Indent code to improve readability
III. Comment code blocks that are not easily understandable
    a) Use single-line comment markers(--)
    b) Reserve multi-line comments (/*.. ..*/) for blocking out sections of code
IV. Use single quote characters to delimit strings.
    a) Nest single quotes to express a single quote or apostrophe within a string
       For example, SET @sExample = 'SQL''s Authority'
V. Use parentheses to increase readability,
    a) WHERE (color='red' AND (size = 1 OR size = 2))
VI. Use BEGIN..END blocks only when multiple statements are present within a conditional code segment.
VII. Use one blank line to separate code sections.
VIII. Use spaces so that expressions read like sentences.
      Example fillfactor = 25, not fillfactor=25
IX. Format JOIN operations using indents,Also, use ANSI Joins instead of old style joins
X. Place SET statements before any executing code in the procedure.

IV. Coding

I. Optimize queries using the tools provided by SQL Server
II. Do not use SELECT *
III. Return multiple result sets from one stored procedure to avoid trips from the application server to SQL server

IV. Avoid unnecessary use of temporary tables,Use 'Derived tables' or CTE (Common Table Expressions) wherever possible, as they perform better

V. Avoid using<> as a comparison operator, use ID IN(1,3,4,5) instead of ID <> 2

VI. Use SET NOCOUNT ON at the beginning of stored procedures7

VII. Do not use cursors or application loops to do inserts,Instead, use INSERT INTO

VIII. Fully qualify tables and column names in JOINs

IX. Fully qualify all stored procedure and table references in stored procedures.

X. Do not define default values for parameters. If a default is needed, the front end will supply the value.

XI. Do not use the RECOMPILE option for stored procedures.

XII. Place all DECLARE statements before any other code in the procedure.

XIII. Do not use column numbers in the ORDER BY clause.

XIV. Do not use GOTO.

XV. 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

XVI. 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

XVII. Always use a column list in your INSERT statements. This helps avoid problems when the table structure changes (like adding or dropping a column).

XVIII. 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.

   a. Any expression that deals with NULL results in a NULL output.
   b. The ISNULL and COALESCE functions are helpful in dealing with NULL values.

XIX. Do not use the identitycol or rowguidcol.

XX. Avoid the use of cross joins, if possible.

XXI. When executing an UPDATE or DELETE statement, use the primary key in the WHERE condition, if possible. This reduces error possibilities.

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

XXIII. Avoid dynamic SQL statements as much as possible.

XXIV. Access tables in the same order in your stored procedures and triggers consistently.

XXV. Do not call functions repeatedly within your stored procedures, triggers, functions and batches.

XXVI. Default constraints must be defined at the column level.

XXVII. 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.

XXVIII. Define all constraints, other than defaults, at the table level.

XXIX. When a result set is not needed, use syntax that does not return a result set.

XXX. 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.

XXXI. Constraints that apply to more than one column must be defined at the table level.

XXXII. Use the CHAR data type for a column only when the column is non-nullable.

XXXIII. Do not use white space in identifiers.

XXXIV. The RETURN statement is meant for returning the execution status only, but not data.

V. Useful Tips

i. 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.

ii. 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.

iii. 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.

iv. Qualifying table names with owner names helps in execution plan reuse, further boosting performance.

v. Join Tables
Incorrect:
SELECT * FROM Table1, Table2 WHERE Table1.d = Table2.c
Correct:
SELECT * FROM Table1 INNER JOIN Table2 ON Table1.d = Table2.c

vi. 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.

vii. Consider the following query to find the second highest offer price from the Items table:

SELECT MAX(Price)

```
FROM Products
WHERE ID IN
(
SELECT TOP 2 ID
FROM Products
ORDER BY Price Desc
)
```

The same query can be re-written using a derived table, as shown below, and it performs generally twice as fast as the above query:

```
SELECT MAX(Price)
FROM
(
SELECT TOP 2 Price
FROM Products
ORDER BY Price DESC
)
```

viii. 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.

ix. Try to avoid server side cursors as much as possible. Always stick to a 'set-based approach' instead of a 'procedural approach' for accessing and manipulating data. Cursors can often be avoided by using SELECT statements instead. If a cursor is unavoidable, use a WHILE loop instead. For a WHILE loop to replace a cursor, however, you need a column (primary key or unique key) to identify each row uniquely.

x. 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.

xi. This helps to avoid deadlocks. Other things to keep in mind to avoid deadlocks are:
   a. Keep transactions as short as possible.
   b. Touch the minimum amount of data possible during a transaction.
   c. Never wait for user input in the middle of a transaction.
   d. Do not use higher level locking hints or restrictive isolation levels unless they are absolutely needed.

xii. If 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.

xiii. Use the following
IF EXISTS (SELECT 1 FROM Products WHERE ID = 50)
Instead Of:
IF EXISTS (SELECT COUNT(ID) FROM Products WHERE ID = 50)