DATABASE ADMINISTRATION SQL SERVER STANDARDS

SQL Server Naming Conventions and Standards 3
1.0 Databases, Files, and File Paths 3
2.0 Tables and Views 3
3.0 Columns 3
4.0 Indexes 3
5.0 Stored Procedures 4
6.0 Triggers 4
7.0 Variables 4

SQL Server Programming Guidelines 5
1.0 Introduction 5
2.0 Code Readability and Format 5
3.0 Datatypes 7
4.0 Stored Procedures 7
5.0 Performance Considerations 8
6.0 Miscellaneous Topics 10

SQL Server Security Model 12
1.0 General Access Requirements 12
2.0 SQL Server Roles 12

SQL Server Migration 14
1.0 RACF Requirements 14
2.0 Development Environment 14
3.0 Production Migration 14

SQL Server Execution Environment 14
1.0 Non-Web Applications 15
2.0 Web Applications 15

SQL Server Naming Conventions and Standards

1.0 Databases, Files, and File Paths

- The database name should accurately reflect the database content and function. All database names must be prefixed with the originating component's acronym--e.g., CBC_Medicare_Compare or EDG_Database_Tracking. Use both upper and lower case letters as appropriate. Do not use spaces in the name.

- File names must match the database name.

- If the database is being developed off site from CMS's central office, please create your databases in the default Microsoft directory structure. E.g.,

  C:\Program Files\Microsoft SQL Server\
  MSSQL\data\OIS_Personnel_Master_Data.MDF

2.0 Tables and Views

- Table names should accurately reflect the table's content and function. Do not use spaces in the name.

- View names follow the same conventions as table names, but should be prefixed with the literal 'VW'. E.g.,

  vw1999NewRegulations

3.0 Columns

The standards below are applicable to all column names:

- Each column name must be unique within its table.
- Each column name must be derived from the business name identified during the business/data analysis process. For more information on deriving column names from business names, see *Creating Physical Names for Elements and Columns* in the *Data Administration standards*. If the column was not identified during the analysis of business data, it must still be given a spelled-out logical name and an abbreviated physical name.
- Do not use reserved or key words as object names.

In addition, if the data is going to be brought in-house to interact with other CMS computer systems, the following standards are applicable:

- The name can have a maximum of 18 characters.
- The name must include acceptable class and modifying words as specified in CMS's *Data Administration standards*.

4.0 Indexes

Indexes are named to indicate the table they are attached to and the purpose of the index.

- Primary keys have a suffix of '_PK'.
- Foreign keys have a suffix of '_FKx' where x is a number that is incrementally assigned.
- Clustered indexes have a suffix of '_IDX'.
- All other indexes have a suffix of '_NDXx' where x is incrementally assigned.

Only one suffix per index may be appended. The application of the appropriate suffix should follow the following hierarchy: primary key, clustered index, foreign key, other index. E.g., an index that is both a primary key and clustered should have a suffix of '_PK'. It is good practice to index columns that are frequently used in a query's selection criteria.

5.0 Stored Procedures

- System level stored procedures are named using a prefix 'SP__' (two underscores) and a description of what the stored procedure does.
- All application level and user defined stored procedures are prefixed with the constant 'USP' with a description of what the stored procedure does. E.g., UspGetLastModifiedDate

6.0 Triggers

Triggers are named to indicate the table they are for and the type of trigger. The purpose of the trigger is identified in the prefix to the name. All triggers should be prefixed with the letter 'T', a letter(s) designating the type, an underscore, and the table name. The type should be designated as 'I' = insert, 'U' = update, 'D' = delete. E.g., ti_Orders (Insert trigger)

7.0 Variables

Variable identifiers for datatypes should consist of two parts:

- The *base*, which describes the content of the variable;
- The *prefix*, which describes the datatype of the variable

Correct prefixes for each datatype are shown in the table below.

Datatype	Prefix	Example
Char	chr	@chrFirstName
Varchar	chv	@chvActivity
Nchar	chrn	@chrnLastName
Nvarchar	chvn	@chvnLastName
Text	txt	@txtNote
Ntext	txtn	@txtnComment
Datetime	dtm	@dtmTargetDate
Smalldatetime	dts	@dtsCompletedDate
Tinyint	iny	@inyActivityID

Smallint	ins	@insEquipmentTypeID
Integer	int	@intAsset
Bigint	inb	@inbGTIN
Numeric or Decimal	dec	@decProfit
Real	rea	@reaVelocity
Float	flt	@fltLength
Smallmoney	mns	@mnsCost
Money	mny	@mnyPrice
Binary	bin	@binPath
Varbinary	biv	@bivContract
Image	img	@imgLogo
Bit	bit	@bitOperational
Timestamp	tsp	@tspOrderID
Uniqueidentifier	guid	@guidPrice
sql_variant	var	@varInventory
Cursor	cur	@curInventory
Table	tbl	@tblLease

SQL Server Programming Guidelines

1.0 Introduction

This section provides guidelines and best practices for SQL Server programming.

Guidelines and best practices should be followed as a general rule, but it is understood that exception situations may exist. Developers must be prepared to provide a justification for any exceptions.

2.0 Code Readability and Format

- Write comments in your stored procedures, triggers and SQL batches generously, whenever something is not very obvious. This helps other programmers understand your code. Don't worry about the length of the comments, as it won't impact the performance, unlike interpreted languages (e.g., ASP 2.0).

- Always use case consistently in your code. On a case insensitive server, your code might work fine, but it will fail on a case sensitive SQL Server if the code is not consistent in case. For example, if you create a table in SQL Server or a database that has a case-sensitive or binary sort order, all references to the table must use the same case that was specified in the CREATE TABLE statement. If you name the table "MyTable" in the CREATE TABLE statement and use "mytable" in the SELECT statement, you get an "object not found" error.

- Do not use column numbers in the ORDER BY clause. In the following examples, note that the second query is more readable than the first.

  Example 1:

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

  Example 2:

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

- Use the more readable ANSI-Standard Join clauses instead of the old style joins. With ANSI joins, the WHERE clause is used only for filtering data. With older style joins, the WHERE clause handles both the join condition and filtering data. The first of the following two examples shows the old style join syntax, while the second one shows the new ANSI join syntax.

  Example 1:

  ```
  SELECT a.au_id, t.title
  FROM titles t, authors a, titleauthor ta
  WHERE
  a.au_id = ta.au_id AND
  ta.title_id = t.title_id AND
  t.title LIKE '%Computer%'
  ```

  Example 2:

  ```
  SELECT a.au_id, t.title
  FROM authors a
  INNER JOIN
  titleauthor ta
  ON
  a.au_id = ta.au_id
  INNER JOIN
  titles t
  ON
  ta.title_id = t.title_id
  WHERE t.title LIKE '%Computer%'
  ```

- To make SQL statements more readable, start each clause on a new line and indent when needed. E.g.:

```
SELECT title_id, title
FROM titles
WHERE title LIKE '%Computer%' AND
title LIKE '%cook%'
```

- As is true with any other programming language, do not use GOTO, or use it sparingly. Excessive usage of GOTO can lead to hard-to-read-and-understand code.

3.0 Datatypes

- Use User Defined Datatypes if a particular column repeats in multiple tables so that the datatype of that column is consistent across all your tables.

- Use the CHAR data type for a column only when the column is non-nullable. If a CHAR column is nullable, it is treated as a fixed length column in SQL Server 7.0+. So, a CHAR(100), when NULL, will eat up 100 bytes, resulting in space wastage. Use VARCHAR(100) in this situation. Of course, variable length columns do have a very little processing overhead over fixed length columns. Carefully choose between CHAR and VARCHAR depending upon the length of the data you are going to store.

- Use Unicode datatypes, like NCHAR, NVARCHAR, or 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 datatypes only when they are absolutely needed as they use twice as much space as non-Unicode datatypes.

- Try not to use TEXT or NTEXT datatypes for storing large blocks of textual data. The TEXT datatype has some inherent problems associated with it. For example, 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. There are also a lot of bugs associated with replicating tables containing text columns. So, if you don't have to store more than 8KB of text, use CHAR(8000) or VARCHAR(8000) datatypes instead.

4.0 Stored Procedures

- Always add an @Debug parameter to your stored procedures. This can be a BIT data type. When a '1' is passed for this parameter, print all the intermediate results, variable contents using SELECT or PRINT statements. When '0' is passed do not print anything. This helps in quickly debugging stored procedures as you don't have to add and remove these PRINT/SELECT statements before and after troubleshooting problems.

- Do not call functions repeatedly within your stored procedures, triggers, functions and batches. For example, 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.

- Make sure your stored procedures always return a value indicating their 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 handles output parameters faster than resultsets returned by SELECT statements.

- Do not prefix your stored procedure names with 'sp_'. The prefix 'sp_' is reserved for system stored procedure 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. You can save time in locating the stored procedure by avoiding the 'sp_' prefix.

- Do not let your front-end applications query/manipulate the data directly using SELECT or INSERT/UPDATE/DELETE statements. Instead, create stored procedures and let your applications access these stored procedures. This keeps the data access clean and consistent across all the modules of your application, while at the same time centralizing the business logic within the database.

5.0 Performance Considerations

- While designing your database, keep performance in mind. You can't really tune performance later when your database is in production as it involves rebuilding tables and indexes, re-writing queries, etc. 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 should be avoided where possible. Choose the right indexes on the right columns.

- Initially, your data should be normalized at least to the third normal form. If you then need to denormalize some of the data to improve performance, you may do so. There should be a documented rationale for all denormalization activities.

- Do not use 'SELECT *' in your queries. Always write the required column names after the SELECT statement, as in the following example:

  ```
  SELECT CustomerID, CustomerFirstName, City
  ```

  This technique results in reduced disk I/O and better performance.

- Avoid the creation of temporary tables while processing data as much as possible, as creating a temporary table means more disk I/O. Consider using advanced SQL, views, SQL Server 2000 table variable, or derived tables instead of temporary tables.

- Try to avoid wildcard characters at the beginning of a word while searching using the LIKE keyword as that results in a full table scan, which defeats the purpose of an index. The first example below results in an index scan, while the second example results in an index seek.

Example 1:

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

Example 2:

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

The use of functions in SELECT statements will not take advantage of indexing.

- Also avoid searching using not equals operators (<> and NOT) as they result in table and index scans.

- Use derived tables wherever possible, as they perform better. Consider the following query to find the second highest salary from the Employees table:

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

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

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

  This is just an example and your results might differ in different scenarios depending on the database design, indexes, volume of data, etc. So, test all the possible ways a query could be written and go with the most efficient one.

- 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 improves the performance of stored procedures by reducing network traffic.

- Perform all your referential integrity checks and data validations using constraints (foreign key and check constraints) instead of triggers, as they are faster. Use triggers only for auditing, custom tasks and validations that cannot be performed using constraints. Constraints save you time as well, as you don't have to write code for these validations, allowing the RDBMS to do all the work for you.

6.0 Miscellaneous Topics

- 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. A WHILE loop is always faster than a cursor. For a WHILE loop to replace a cursor you need a column (primary key or unique key) to identify each row uniquely. Every table must have a primary or unique key in any case.

- Views are generally used to show specific data to specific users based on their interest. Views are also used to restrict access to the base tables by granting permission only on views. Yet another significant use of views is that they simplify your queries. Incorporate your frequently required, complicated joins and calculations into a view so that you don't have to repeat those joins/calculations in all your queries. Instead, just select from the view.

- If you have a choice, do not store binary or image files (Binary Large Objects or BLOBs) inside the database. Instead, store the path to the binary or image file in the database and use that as a pointer to the actual binary file stored elsewhere on a server. Retrieving and manipulating these large binary files is better performed outside the database. Keep in mind that a database is not meant for storing files.

- 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. IF and CASE statements come in handy to avoid dynamic SQL. Another major disadvantage of using dynamic SQL is that it requires users to have direct access permissions on all accessed objects, like tables and views. Generally, users are given access to the stored procedures which reference the tables, but not directly on the tables. In this case, dynamic SQL will not work.

- Consider the following drawbacks before using the IDENTITY property for generating primary keys. IDENTITY is very much SQL Server specific, and you will have problems porting your database application to some other RDBMS. IDENTITY columns have other inherent problems. For example, IDENTITY columns can run out of numbers at some point, depending on the data type selected; numbers can't be reused automatically, after deleting rows; and problems may arise if you are using replication.

  So, come up with an algorithm to generate a primary key in the front-end or from within the inserting stored procedure. There still could be issues with generating your own primary keys too, like concurrency while generating the key, or running out of values. So, consider both options and choose the one that is most appropriate for your circumstances.

- Minimize the use 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. ISNULL and COALESCE functions are helpful in dealing with NULL values.

- 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).

- Always access tables in the same order in all your stored procedures and triggers consistently. This helps in avoiding deadlocks. Other things to keep in mind to avoid deadlocks are:
    - Keep your transactions as short as possible.
    - Touch the least amount of data possible during a transaction.
    - Do not wait for user input in the middle of a transaction.
    - Do not use higher level locking hints or restrictive isolation levels unless they are absolutely needed.
    - Make your front-end applications deadlock-intelligent, that is, these applications should be able to resubmit the transaction in case the previous transaction fails with error 1205.
    - In your applications, process all the results returned by SQL Server immediately so that the locks on the processed rows are released, hence no blocking.

- Offload 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. Also try to do basic validations in the front-end itself during data entry. This saves unnecessary network roundtrips.

- Always check the global variable @@ERROR immediately after executing a data manipulation statement (like INSERT/UPDATE/DELETE), so that you can rollback the transaction in case of an error (@@ERROR will be greater than 0 in case of an error). This is important because, by default, SQL Server will not rollback all the previous changes within a transaction if a particular statement fails. This behavior can be changed by executing SET XACT_ABORT ON. The @@ROWCOUNT variable also plays an important role in determining how many rows were affected by a previous data manipulation (also, retrieval) statement, and based on that you could choose to commit or rollback a particular transaction.

- Always store 4 digit years instead of 2 digit years in dates (especially when using CHAR or INT datatype columns) to avoid any confusion and problems. This is not a problem with DATETIME columns, as the century is stored even if you specify a 2 digit year. But it's always a good practice to specify 4 digit years even with DATETIME datatype columns.

- Do not forget to enforce unique constraints on your alternate keys.

SQL Server Security Model

1.0 General Access Requirements

- To be able to access data from a database, a user must pass through two stages of authentication, one at the SQL Server level and the other at the database level. These two stages are implemented using Login names and User accounts respectively. A valid login is required to connect to SQL Server and a valid user account is required to access a database.

- Login: A valid login name is required to connect to a SQL Server instance. At CMS a valid login is approved by the RACF group and entered into their system. These login names are also maintained within the master database of the SQL Server they are authorized to access.

- User: A valid user account within a database is required to access that database. User accounts are specific to a database. All permissions and ownership of objects in the database are controlled by the user account. SQL Server logins are associated with these user accounts. A login can have associated users in different databases, but only one user per database.

- Controlling access to objects within the database and managing permissions: Apart from managing permissions at the individual database user level, SQL Server 7.0/2000 implements permissions using roles. See the next section for information on types of roles.

2.0 SQL Server Roles

A role is nothing but a group to which individual logins/users can be added, so that the permissions can be applied to the group, instead of applying the permissions to all the individual logins/users. There are three types of roles in SQL Server 7.0/2000:

- Fixed server roles will be given to project leaders and local DBAs.

- Fixed database roles will be given to database users.

- Application roles will be assigned to Web applications.

2.1 Fixed Server Roles

These are server-wide roles. Logins can be added to these roles to gain the associated administrative permissions of the role. Fixed server roles cannot be altered and new server roles cannot be created.

Here are the fixed server roles and their associated permissions in SQL Server 2000:

Fixed Server Role	Permissions
sysadmin	Can perform any activity in SQL Server

serveradmin	Can set server-wide configuration options, shut down the server
setupadmin	Can manage linked servers and startup procedures
securityadmin	Can manage logins and CREATE DATABASE permissions, also read error logs and change passwords
processadmin	Can manage processes running in SQL Server
dbcreator	Can create, alter, and drop databases
diskadmin	Can manage disk files
bulkadmin	Can execute BULK INSERT statements

2.2 Fixed Database Roles

Each database has a set of fixed database roles, to which database users can be added. These fixed database roles are unique within the database. While the permissions of fixed database roles cannot be altered, new database roles can be created.

Here are the fixed database roles and their associated permissions in SQL Server 2000:

Fixed Database Role	**Permissions**
db_owner	Has all permissions in the database
db_accessadmin	Can add or remove user IDs
db_securityadmin	Can manage all permissions, object ownerships, roles and role memberships
db_ddladmin	Can issue ALL DDL, but cannot issue GRANT, REVOKE, or DENY statements
db_backupoperator	Can issue DBCC, CHECKPOINT, and BACKUP statements
db_datareader	Can select all data from any user table in the database
db_datawriter	Can modify any data in any user table in the database
db_denydatareader	Cannot select any data from any user table in the database
db_denydatawriter	Cannot modify any data in any user table in the database

2.3 Application Roles

Application roles are another way of implementing permissions. These are quite different from the server and database roles. After creating and assigning the required permissions to an application role, the client application needs to activate this role at run-time, using a password. By activating the role, the application inherits the permissions associated with that role. Application roles override standard permissions. Application roles are database specific.

SQL Server Migration

1.0 RACF Requirements

- All users of the database must be cleared by the RACF group.
- The users and database name will be passed on from RACF to DBA.

2.0 Development Environment

- SQL Server databases may be developed offsite, upsized from an Access database or developed in-house on a development server.
- Development of the physical model will occur after an ERWIN logical data model is submitted to the EDG/DDS DA staff. For more information on the logical model and DA standards, see *Data Administration Standards*.
- From the physical model the DDL will be generated to create your database on a QA/development server at CMS.
- If the database is to be accessed via the web, it must be tested on the QA/CMSnet site. Otherwise, it must be tested on the development test SQL SERVER within the CMS data center (currently CONDS04).
- The EDG/DDS SQL Server DBA team will build the database and establish the users.

3.0 Production Migration

- After the database has been tested and proven satisfactory to the user, it will be migrated to production.
- If the database is accessed via the web and the application has a sign-off from the web support team, the EDG/DDS SQL Server team will move the database to the production environment housed offsite by AT&T.
- If the database is not web related and the business office approves the test, the database is moved to the production SQL Server within the CMS data center (currently CONDS03).
- The database will be backed up and restored to production so that it is an exact duplicate of the test database in structure. The data will also be replicated to production if so desired.

SQL Server Execution Environment

1.0 Non-Web Applications

- Non-Web applications reside on local servers at the CMS site.
- The servers are CONDS03 (IP 158.73.105.23) and CONDS04 (IP 158.73.52.39). The first is the production box and the latter is development. Both of these servers are SQL SERVER version 7.0.
- Both boxes run on the Windows 2000 operating system. There are plans to upgrade both these boxes to SQL Server version 2000 in the year 2002.
- All new systems will first be placed on the test box and migrated to production following standard configuration management practices.

2.0 Web Applications

- Web applications reside on servers located at AT&T.
- These servers run under the Windows 2000 operating system and are maintained by AT&T personnel.
- The servers run SQL SERVER version 2000.

2.1 Intranet Applications

- The Intranet (or CMSNET) is accessed by CMS employees only.
- This site has two application servers: 7667www4 (IP 32.86.194.135) and 7667www5 (IP 32.86.194.136). The former is the production Intranet application server and the latter is for testing.
- Both application servers point to the database server 7667db3 (IP 32.86.194.136).

2.2 Internet Applications

- The Internet (or cms.gov) is accessed by the public.
- Each database server is clustered to ensure maximum up time.
- Two physically distinct locations ensure fail safe operations:

  The Ashburn Site
  - The application servers at the Ashburn site are 7667www0 (IP 32.86.180.134) and 7667www1 (IP 32.86.180.135).
  - They both point to the database servers that are clustered at IP 32.86.181.185.
  - For administrative purposes, these databases are read only. Applications that update data do not reside on the Ashburn database server.

The Koll Site

- The application servers at Koll are 7666www0 (IP 32.86.183.36) and 7666www1 (IP 32.86.183.37).
- Both these application servers point to a clustered database server whose IP address is 32.86.183.10. This database server houses all updateable databases.

2.3 Internet Staging Database

- A staging database server is available for Internet testing.
- This server is accessed by the staging Application server 7667www2 (IP 32.86.180.136).
- The database server is 7667db0 (IP 32.86.181.182).