SQL SERVER STANDARDS DATABASE ADMINISTRATION

SQL Server Naming Conventions and Standards

1.0 Databases, Files, and File Paths

• The database name should reflect the project.

• File names must match the database name.

2.0 Tables and Views

• Table names should accurately reflect the table's content and function. Do not use spaces in the name. Always should be plural ending with ‘s’. For Example, tblUsers

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

3.0 Columns

The standards below are applicable to all column names:

• Each column name must be unique within its table.
• Each column name should be logical.
• Do not use reserved or key words as object names.
• The name can have a maximum of 25 characters.

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 'FK'
• Clustered indexes have a suffix of IDX'.
• All other indexes have a suffix of 'NDX'

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 do.
• All application level and user defined stored procedures are prefixed with the constant 'u' with a description of what the stored procedure does. Use following naming convention whenever possible :

u<tablename><action type(get,insert,delete,update etc)>
E.g., u_UserdetailsGet

• All local variables in stored procedure should start with ‘l’.
E.g., @lintTempUser

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_tblOrders (Insert trigger)

7.0 Variables

Datatype	Prefix	Example
Char	str	@strFirstName
Varchar	str	@strActivity
Nchar	str	@strLastName
Nvarchar	str	@strLastName
Text	str	@strNote
Ntext	str	@strComment
Datetime	dtm	@dtmTargetDate
Smalldatetime	dtm	@dtmCompletedDate
Tinyint	int	@intActivityID
Smallint	int	@intEquipmentTypeID
Integer	int	@intAsset
Bigint	int	@intGTIN
Numeric or Decimal	dec	@decProfit
Real	dbl	@dblVelocity
Float	Flt	@fltLength
Smallmoney	mon	@monCost
Money	mon	@monPrice
Binary	bin	@binPath
Varbinary	bin	@binContract
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.: 6SELECT 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 Data types

• Use the CHAR data type for a column only when the column is non-null able. 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 data types, like NCHAR, NVARCHAR, or NTEXT, if your database is going to store non English characters.

• Try not to use TEXT or NTEXT data types for storing large blocks of textual data. The TEXT data type 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) data types instead.

4.0 Stored Procedures

• 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.

• If your stored procedure always returns one or two values, consider returning the result set using OUTPUT parameters instead of a SELECT statement, as ADO handles output parameters faster than result sets returned by SELECT statements.

5.0 Performance Considerations

• While designing your database, keep performance in mind. 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 renormalize some of the data to improve performance, you may do so. There should be a documented rationale for all renormalization 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 disks I/O. Consider using advanced SQL, views, 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 those results in a full table scan, which defeats the purpose of an index. The first example below results in an index scans, 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

• 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).

6.0 Miscellaneous Topics

• Try to avoid server side cursors as much as possible.

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.

• 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 on a server. Retrieving and manipulating these large binary files is better performed outside the database.

• 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.

• 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.

• 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.

• 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).

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