SQL SERVER STANDARDS DATABASE ADMINISTRATION

Table of Content

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.

Example:- Project for Vehicle Maintenance.

Not Good:

```
CREATE DATABASE testDB;
CREATE DATABASE Vehicle;
CREATE DATABASE VehicleLive;
CREATE DATABASE VehicleProd;
```

Good:

```
CREATE DATABASE VehicleMaintenance
```

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'.

Example:- Vehicle Entity

Not Good:

- Vehicle
- Vehi cle

Good:-

- tblVehicles
- Vehicles

- Programming languages have their best practices when it comes to case types like camelCase, PascalCase, etc.

View names follow the same conventions as table names, but should be prefixed with the literal 'VW'. **Example:- Vehicle Records**

Not Good:

- VehicleRecords
- VWvahiclerecords
- vwvahiclerecords

Good:-

- vwVehicleRecords

3.0 Columns

The standards below are applicable to all column names:

- Each column name must be unique within its stable.
- Each column name should be logical.
- Do not use reserved or keywords as object names.
- The name can have a maximum of 25 characters.

Example:- Employee

Not Good:

```
SELECT   EMP.Id,
         EMP.Name,
         EMP.Dept
FROM Employee EM
```

Good:

```
SELECT   emp.Id,
         emp.Name,
         emp.Dept
FROM Employee emp
```

4.0 Indexes

Indexes is 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, and another 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.

Example:-

Good: Vehicle Table

- **PK_Vehicle**
- **IX_ Vehicle_Name_Age**
- **CL_Vehicle**
- **UX_Vehicle_Name_Age**

5.0 Stored Procedures

- Ensure your stored procedure returns an answer demonstrating their status. Institutionalize the arrival estimates of the stored procedures for success and failure. The Return statements for returning the execution status, not information.

- System-level stored procedures are named using the prefix 'sp ' (two underscores) and a description of what the stored procedure does.

- Stored Procedure name formate:-

  sp_[table-name]_[action]

- **Example:- Sp for Get Record By Id**

```
CREATE  PROCEDURE
[dbo].[sp_MagazineTags_GetById]

    @Id INT = 0

AS

BEGIN

    SET TRANSACTION ISOLATION LEVEL
SNAPSHOT

    SELECT *

    FROM MagazineTags WITH(nolock)

    WHERE (ISNULL(@Id,0) = 0 OR (Id =
@Id))

    ORDER BY CreationDate DESC

END
```

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.

The naming convention for triggers is

1. Each trigger name should use the syntax "TR_<TableName>_<ActionName>.
2. Each trigger name should have the prefix "TR_".
3. Each table name and action name should start with a capital letter.
4. The table name should end with the letter "s" (or "es") to indicate plural.

Example:-

- TR_Employees_ AfterInsert
- TR_OrderDetails_AfterUpdate
- TR_Employees_InstedOfDelete

7.0 Use Common Table Expression(CTE)

A CTE allows you to define and execute a query, in which the result exists temporarily and can be used within a larger query. Using CTE improves the readability of your query. A CTE is defined once and then can be referred to multiple times.

Example :-

```
WITH my_cte AS
(
  SELECT col1, col2 FROM
table
)
SELECT * FROM my_cte
```

8.0 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

9.0 Declare variable

In spite of the fact that T-SQL has no understanding of constants(like the ones in the C language), the variable can fill a similar need. Using variables rather than steady values inside your query enhances the comprehensibility and viability of your code.

Example :-

```
DECLARE @Id, @status

SELECT @Id = 5 , @status = 1

SELECT id, name

FROM Table

WHERE Status IN (@status, @id)
```

10.0 @Debug parameter

Continuously add a @Debug parameter to your stored procedure. This can be of BIT information sort. At the point when a 1 is passed for this parameter, print all the moderate results, and variable substance using SELECT or PRINT articulations, and when 0 is passed, don't print anything. This aids in the fast investigation of stored procedure systems, as you don't need to include and expel these PRINT/SELECT articulations prior to and then after investigating issues.

11.0 Output parameter

If any stored procedure returns a single row of data, consider returning the data using OUTPUT parameters in the place of a SELECT query and the Output parameter is faster than the data returned by a SELECT query.

12.0 Use TRY-CATCH block for error handling

Sometimes our queries can return errors. We need to handle these errors. Especially when we get an error in transaction blocks, we have to roll back this transaction. Because open transactions may cause serious problems. Therefore, we need to use TRY-CATCH blocks.

Example:-

```
-- Create a procedure to retrieve error information.
CREATE PROCEDURE usp_GetErrorInfo
AS
SELECT
        ERROR_NUMBER() AS ErrorNumber
        ,ERROR_SEVERITY() AS ErrorSeverity
        ,ERROR_STATE() AS ErrorState
        ,ERROR_PROCEDURE() AS ErrorProcedure
        ,ERROR_LINE() AS ErrorLine
        ,ERROR_MESSAGE() AS ErrorMessage;
GO
BEGIN TRY
        -- Generate divide-by-zero error.
```

```
        SELECT 1/0;
END TRY
BEGIN CATCH
        -- Execute error retrieval routine.
        EXECUTE usp_GetErrorInfo;
END CATCH;
```

13.0 Functions

Functions usually perform simple calculations and return values. Therefore, the best way to name them would be to describe what the function does. I also love to put *f_* at the start of the name. In our database, one example is –

Example :-

- *f_east_from_long*.

14.0 Sql Commands

Structured Query Language(SQL) as we all know is the database language by the use of which we can perform certain operations on the existing database and also we can use this language to create a database. SQL uses certain commands like Create, Drop, Insert, etc. to carry out the required tasks.

These SQL commands are mainly categorized into four categories as:

1. **DDL – Data Definition Language**
2. **DQl – Data Query Language**
3. **DML – Data Manipulation Language**
4. **DCL – Data Control Language**
5. **TCL – Transaction Control Language**

1. DDL – Data Definition Language:-

- Ø Data Definition Language actually consists of the SQL commands that can be used to define the database schema. It simply deals with descriptions of the database schema and is used to create and modify the structure of database objects in the database.

- Ø DDL is a set of SQL commands used to create, modify, and delete database structures but not data. These commands are normally not used by a general user, who should be accessing the database via an application.

Ø List of DDL commands:

- **CREATE :-**
  - This command is used to create the database or its objects (like table, index, function, views, store procedure, and triggers).
  - **Example:-**

```
CREATE TABLE Subject
(
Sub_ID INT,
Sub_Name varchar(20)
);
```

- **DROP:-**
  - This command is used to delete objects from the database.
  - **Example:-**

```
DROP TABLE table_name;
```

- **ALTER:-**
  - This is used to alter the structure of the database.
  - **Example:-**

```
ALTER TABLE Student ADD (AGE number(3),COURSE varchar(40));
```

-

RENAME:-

- This is used to rename an object existing in the database.
- **Example:-**

ALTER TABLE Student RENAME TO Student_Details;

2. DQl – Data Query Language:-

Ø **DQL** statements are used for performing queries on the data within schema objects.

Ø The purpose of the DQL Command is to get some schema relation based on the query passed to it.

Ø We can define DQL as follows it is a component of SQL statement that allows getting data from the database and imposing order upon it.

Ø It includes the SELECT statement. This command allows getting the data out of the database to perform operations with it.

Ø When a SELECT is fired against a table or tables the result is compiled into a further temporary table, which is displayed or perhaps received by the program i.e. a front-end.

Ø List of DQl commands:

- **SELECT:-**
  - It is used to retrieve data from the database.
  - **Example:-**

SELECT ROLL_NO, NAME, AGE FROM Student;

3. DML – Data Manipulation Language:-

Ø The SQL commands that deal with the manipulation of data present in the database belong to DML or Data Manipulation Language and this includes most of the SQL statements. It is the component of the SQL statement that controls access to data and to the database. Basically, DCL statements are grouped with DML statements.

Ø List of DML commands:

- **INSERT:-**
  - It is used to insert data into a table.
  - **Example:-**

```
INSERT INTO Student VALUES (‘5’,’HARSH’,’WEST BENGAL’,’XXXXXXXXXX’,’19’);
```

- **UPDATE:-**
  - It is used to update existing data within a table.
  - **Example:-**

```
UPDATE Student SET NAME = 'PRATIK' WHERE Age = 20;
```

- **DELETE:-**
  - It is used to delete records from a database table.
  - **Example:-**

```
DELETE FROM Student WHERE NAME = 'Ram';
```

4. DCL – Data Control Language

Ø DCL includes commands such as GRANT and REVOKE which mainly deal with the rights, permissions, and other controls of the database system.

Ø List of DML commands:

- **GRANT:-**
  - This command gives users access privileges to the database.
  - **Syntax:-**

    GRANT privileges_names ON object TO user;
  - **Example:-**

    grant insert,

    select on accounts to Ram

- **REVOKE:-**
  - This command withdraws the user's access privileges given by using the GRANT command.
  - **Syntax:-**

    revoke privilege_name on object_name

    from {user_name | public | role_name}
  - **Example:-**

    revoke insert,

    select on accounts from Ram

5. TCL– Transaction Control Language:-

Ø Transactions group a set of tasks into a single execution unit. Each transaction begins with a specific task and ends when all the tasks in the group successfully complete. If any of the tasks fail, the transaction fails. Therefore, a transaction has only two results: **success** or **failure**.

Ø List of TCL commands

- **COMMIT:-**
  - Commits a Transaction.
  - **Example:-**

```
DELETE FROM Student WHERE AGE = 20;
COMMIT;
```

- **ROLLBACK:-**
  - Rollbacks a transaction in case of any error occurs.
  - **Example:-**

```
DELETE FROM Student WHERE AGE = 20;
ROLLBACK;
```

- **SAVEPOINT:-**
  - Sets a save point within a transaction.
  - **Example:-**

```
SAVEPOINT SP1;
//Savepoint created.
DELETE FROM Student WHERE AGE = 20;
//deleted
SAVEPOINT SP2;
//Savepoint created.
```

15.0 Sequences

Sequence is a set of integers 1, 2, 3… that are generated and supported by some database systems to produce unique values on demand.

- A sequence is a user-defined schema-bound object that generates a sequence of numeric values.
- Sequences are frequently used in many databases because many 65 applications require each row in a table to contain a unique value and sequences provide an easy way to generate them.
- The sequence of numeric values is generated in an ascending or descending order at defined intervals and can be configured to restart when it exceeds max_value.
- **Syntax:-**

```
CREATE SEQUENCE sequence_name

START WITH initial_value

INCREMENT BY increment_value

MINVALUE minimum value

MAXVALUE maximum value

CYCLE|NOCYCLE ;
```

Ø **sequence_name:** Name of the sequence.

Ø **initial_value:** starting value from where the sequence starts. Initial_value should be greater than or equal to the minimum value and less than equal to the maximum value.

Ø **increment_value:** Value by which sequence will increment itself. Increment_value can be positive or negative.

Ø **minimum_value:** Minimum value of the sequence.

- **maximum_value:** Maximum value of the sequence.
- **cycle:** When the sequence reaches its set_limit

  it starts from the beginning.
- **nocycle:** An exception will be thrown

  if the sequence exceeds its max_value.
- **Example -** Following is the sequence query creating a sequence in ascending order.

```
CREATE SEQUENCE sequence_1
start with 1
increment by 1
minvalue 0
maxvalue 100
cycle;
```

16.0 Pivot and Un-pivot in SQL

- In SQL, Pivot, and Un-pivot are relational operators that are used to transform one table into another in order to achieve a simpler view of the table. Conventionally we can say that the **Pivot** operator converts the rows data of the table into the column data. The **Un-pivot** operator does the opposite, that is it transforms the column-based data into rows.
- **Syntax**

1. **Pivot:-**

```
SELECT (ColumnNames)
FROM (TableName)
PIVOT
 (
   AggregateFunction(ColumnToBeAggregated)
   FOR PivotColumn IN (PivotColumnValues)
 ) AS (Alias) //Alias is a temporary name for a table
```

2. **Un-pivot:-**

```
SELECT (ColumnNames)
FROM (TableName)
 UNPIVOT
(
   AggregateFunction(ColumnToBeAggregated)

   FOR PivotColumn IN (PivotColumnValues)
 ) AS (Alias)
```

Ø **Example:-**

```
SELECT CourseName, PROGRAMMING, INTERVIEWPREPARATION
FROM geeksforgeeks
PIVOT
(
SUM(Price) FOR CourseCategory IN (PROGRAMMING,
INTERVIEWPREPARATION )
) AS PivotTable
```

TatvaSoft
sculpting thoughts...

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 exceptional 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).

- Make Sure that you are applying the Normalization rules. Your data must be of at least the 3rd normal form. At the same time, please do not try to compromise your query performance. A little bit of de-normalization in your query, helps the system perform faster.

- Choose a database naming convention, institutionalize it over your association, and be reliable in tailing it. It makes your code more lucid and justifiable.

- Once you write the "SELECT" command in SQL, don't use SELECT * in your queries. Always try to write the only needful column names after the SELECT statements.

- Always use cases 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 the 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 "my table" 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 BY2
```

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 show 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.: 6 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 Data types

- 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 very little processing overhead over fixed-length columns. Carefully choose between CHAR and VARCHAR depending on 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.

- Utilizing SET NOCOUNT ON towards, puts away the system, and triggers underway situations, as this stifles messages, like '(1 row(s) influenced)' in the wake of executing INSERT, UPDATE, DELETE and SELECT statements. This enhances the execution of put-away techniques by lessening the system movements.

- Comments might be useful in some situations. But you should definitely avoid the pitfall of commenting too much.

5.0 Constraint

This one is very important, Once you go to create or alter any constraint in SQL Server. In that part of SQL queries, most of the developers create constraints without constraint names. In that case, SQL will create the default key name for the constraint, e.g. default_abc.

6.0 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 re-normalize some of the data to improve performance, you may do so. There should be a documented rationale for all re-normalization 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 variables, 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 result 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 the 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.
- 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 run time. 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 round trips.
- Always check the global variable @@ERROR immediately after executing a data manipulation statement (like INSERT/UPDATE/DELETE); so that you can roll back 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.