SQL Coding Guidelines

1. Always specify SET NOCOUNT ON at the top of the stored procedure, this command suppresses the result set count information thereby saving some amount of time spent by SQL Server.

2. Always prefix all the database objects.
   For example:
   Use dbo.Order instead of Order.
   Use EXEC dbo.usr_sp_getOrders instead of EXEC usr_sp_getOrders.
   Use dbo.vw_OrderDetails instead of vw_OrderDetails.

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

3. In situations where a big table needs to be joined it is sometimes helpful to first store the required rows from the big table to a temp table and then use the smaller table in the join. This is far efficient than joining with a big table.

4. Always specify Header comments at the top of every stored procedure, function, view.
   Example

```
-- ====================================================
-- Author:           <Author Name>
-- Create date:      <date>
-- Description:      <Brief description>
-- EXEC [dbo].<SP Name>

/*
Change History
1) Added Admin related functionality for Template
*/
-- ===================================================
```

5. Using DISTINCT clause on large number of records is resource intensive and will cause performance issues, instead evaluate and see if the DISTINCT can be avoided using alternate coding practices like GROUP BY clause.

6. Avoid using SELECT * , it is not a good practice and can cause unnecessary Disk IO. Return only the columns that are necessary.

```
For example,

SELECT Column_1, Column_2 FROM TableName
```

7. Avoid using SQL reserved keywords such as SELECT, WHERE, FROM, TIME etc to name database objects like COLUMN NAME.

8. Use IF EXISTS clause to check for row existence instead of doing a SELECT COUNT(1)
   For example:

   ```
   SELECT @TotCount = COUNT(1) FROM dbo.Employee WHERE Emp_ID = 100

   IF @TotCount > 0
   BEGIN
   	--SQL Statement(s)
   END
   ```

   Above statement should be written as

   ```
   IF EXISTS (SELECT 1 FROM dbo.Employee WHERE Emp_ID = 100)
   BEGIN
   	--SQL Statement(s)
   END
   ```

   You can also use below statement just to verify row count of table but not in your script as it access system table. This is fastest ways to getting row count.

   ```
   SELECT TOP 1 rows from sys.partitions where object_id =
   object_id('TableName')
   ```

9. Always specify column names in your INSERT statements.

   ```
   For example,

   INSERT INTO TableName (
   	COLUMN_1,
   	COLUMN_2,
   	COLUMN_3,
   	COLUMN_4
   )
   VALUES(
   	1,
   	2,
   	3,
   	4
   )
   ```

10. When merging 2 result sets, try using UNION ALL instead of UNION clause if possible. UNION has to get distinct values from the result set and sort the final result set and it will add to the performance overhead.

11. By all means avoid using cursor, instead use straightforward SQL. 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.

a. If a cursor is unavoidable, use a WHILE loop instead. Testing and concluded that a WHILE loop is always faster than a cursor is easy. But for a WHILE loop to replace a cursor you need a column (primary key or unique key) to identify each row uniquely. SQL is a set based language and works on a set of rows. In situations we have to use loop it can be implemented using WHILE loop.

12. Use LIKE clause efficiently wherever possible. Do not use wild card characters at the beginning of word while search using LIKE keyword as it results in Index scan.

    For example:

    ```
    SELECT Address, State, ZipCode FROM dbo.ZipCode WHERE City LIKE '%Burbank%'
    SELECT Address, State, ZipCode FROM dbo.ZipCode WHERE City LIKE 'Burbank%'
    ```

    Second statement is more efficient than first.

13. Using SQL Join is better than writing Sub-queries. SQL join simplifies the cardinality between tables and improves the query performance.

14. When working with branch conditions or complicated expressions, use parenthesis to increase readability.

    ```
    IF (   (SELECT 1 FROM TableName WHERE 1=2) ISNULL)
    ```

15. Always indent SQL Code for better readability and also helps make the code look clean. It also helps to understand the other person who supports the code.

16. Avoid SQL Server / User defined functions in the WHERE clause wherever possible to avoid Index scan.
    For example, the below query would cause Index scan.

    ```
    SELECT EmailAddress FROM dbo.Contact WHERE LTRIM(TRIM(EmailAddress)) =
    'test@yahoo.com'
    ```

    If you are sure that the column will not have leading or trailing spaces, do not use the TRIM functions. The trimming of spaces and other validations should be handled on the application side.

    Some more examples:

    - Avoiding ISNULL function
      ```
      WHERE ISNULL(FullName,'') = 'Robert Fisher'
      ```
      Can be re-written efficiently as
      ```
      WHERE ( (FullName = 'Robert Fisher') OR (FullName IS NULL) )
      ```

- Avoiding DATE functions

  ```
  WHERE DATEDIFF(mm,DateOrderPlaced,GETDATE()) >= 30
  ```

  Can be re-written efficiently as

  ```
  WHERE DateOrderPlaced < DATEADD(mm,-30, GETDATE())
  ```

17. Whenever possible avoid using in-line UDF (User defined function) in SELECT statements. Evaluate and see if the UDF could be mimicked as a Table using Derived table or by using CROSS APPLY clause in the join. Functions operate on row by row basis and will cause performance issues especially on large table.

18. State all the DDL statements at the top of the stored procedure, having interleaved DDL statements for example CREATE #Temp or CREATE INDEX on the #temp table in the code will cause the stored procedure to recompile. In case you require to write DLL statement make sure to dispose it once work is done.

    ```
    IF OBJECT_ID ('tempdb..#tmp_001') IS NOT NULL
       DROP TABLE #tmp_001

    CREATE TABLE #tmp_001(
       ID INT,
       NAME VARCHAR(100)
    )

    --Add your logic to store information

    IF OBJECT_ID ('tempdb..#tmp_001') IS NOT NULL
       DROP TABLE #tmp_001
    ```

19. Use Dynamic SQL only when required and if it is absolutely required use EXEC sp_executesql @sql instead of EXEC @sql statement.

20. Indexing:

    - Make Sure All JOIN Columns are indexed. Ensure all the foreign keys have Non-Clustered Index created on them.
    - Use WHERE, JOIN, ORDER BY, SELECT Column Order When Creating Indexes
    - Make Sure All Tables Have a Clustered Index Defined
    - Consult with your DBA before you create any index as in case of huge volume of data index creation takes considerable amount of time and could impact server performance.

21. Ensure you add prefix N' to any Unicode data string or Unicode column whenever you insert or retrieve data.

22. Whenever creating #Temp tables, always specify the column collation as 'COLLATE DATABASE_DEFAULT' for all varchar, nvarchar, char, nchar columns. This will allow the temp table columns to pick the database collation rather than picking up the tempdb db collation which may be different from user defined database collation. Having different collations in joins can cause un-desired data results.

23. Use table variables instead of #temp tables in stored procedures when the data set is small, say less than 100 records.

```
DECLARE @tblExample TABLE(ID INT, NAME VARCHAR(100)
```

24. Beware when using database transactions. Keep transactions as short as possible in the stored procedures and exit gracefully on failure or success. Long transactions tend to cause all sorts of performance problems including blocking and deadlocking. Also open transactions could lock up the entire table thereby making a particular functionality unresponsive.

25. Also avoid searching using not equals operators (<> and NOT) as they result in table and index scans. Alternatives to NOT IN and NOT Exists

    SQL Server 2008 introduced the Merge command for bulk insert / update / delete operations which can be used with the EXCEPT clause to minimize its performance hit.

26. Use MERGE Statement to implement multiple DML operations instead of writing separate INSERT, UPDATE, DELETE statements.

```
MERGE Target1 AS T
USING Source1 AS S
ON (T.EmployeeID = S.EmployeeID)
WHEN NOT MATCHED BY TARGET AND S.EmployeeName LIKE 'S%'
          THEN   INSERT(EmployeeID, EmployeeName)
                 VALUES(S.EmployeeID, S.EmployeeName)
WHEN MATCHED
          THEN   UPDATE SET T.EmployeeName = S.EmployeeName
WHEN NOT MATCHED BY SOURCE AND T.EmployeeName LIKE 'S%'
          THEN   DELETE;
```

27. EXCEPT or NOT EXIST clause can be used in place of LEFT JOIN or NOT IN for better performance.

    Example:

```
SELECT EmpNo, EmpName
FROM EmployeeRecord
WHERE Salary > 1000 AND Salary
```

```
NOT IN (SELECT Salary
FROM EmployeeRecord
WHERE Salary > 2000);
```

Recommended:

```
SELECT EmpNo, EmpName
FROM EmployeeRecord
WHERE Salary > 1000
EXCEPT
SELECT EmpNo, EmpName
FROM EmployeeRecord
WHERE Salery > 2000
ORDER BY EmpName;
```

28. Schema design:
    Use DELETE CASCADE Option to Handle Child Key Removal in Foreign Key Relationships

29. 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.

    Therefore what are possible alternatives here to avoid using temp tables and table variables?
    1. CTE expressions.
    2. Sub queries.
    3. Physical table (Schema Table) created in the database schema, instead of creating a temp table at TempDB.
    4. Table Parameters which has been available from SQL Server 2008.

30. Try not to use TEXT or NTEXT datatypes for storing large 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.

31. Perform all your referential integrity checks and data validations using constraints (foreign key and check constraints) instead of triggers, as they are faster. Limit the 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.

32. 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.

33. While doing data transfer to temp tables or tables on another server/database for faster data import.
    a. Disable constraints and indexes for faster data transfer
    b. Use bcp command since it does not enforce constraints unless specified

34. After large data import of updates always:
    a. Reorg or reindex the table/Update statistics since it would cause a performance degrade

b. Use batch queries for updating large volume data.

35. Use Unicode datatypes, like NCHAR, NVARCHAR or NTEXT only if it’s needed, as they use twice as much space as non-Unicode datatypes.

36. If stored procedure always returns single row result set, then consider returning the result set using OUTPUT parameters instead of SELECT statement, as ADO handles OUTPUT parameters faster than result sets returned by SELECT statements.

37. The first step of ETL best practice is investing time to analyzing the requirements and having developed data model, which can reduce ETL challenges to a large extend.

38. Optimize SQL Data Source by using NOLOCK or TABLOCK hints to remove locking overhead.

39. To optimize memory usage, Select only columns you actually need. If you select unnecessary or all columns from table (e.g. SELECT * FROM) you will needlessly use memory and network bandwidth to store and retrieve columns.

40. If possible, perform your datatype conversions at your source or target databases, as it is more expensive to perform within Integration Services.

41. Most important if Integration Services and SQL Server run on the same server, use the SQL Server destination instead of the OLE DB destination to improve performance.

42. Commit size 0 is fastest on heap bulk targets, because only one transaction is committed. If you cannot use 0, use the highest possible value of commit size to reduce the overhead of multiple-batch writing. Commit size = 0 is a bad idea if inserting into a Btree – because all incoming rows must be sorted at once into the target Btree—and if your memory is limited, you are likely to spill. Batchsize = 0 is ideal for inserting into a heap. For an indexed destination, it is recommend testing between 1000 and 10000 as batch size.

43. Use a commit size of <5000 to avoid lock escalation when inserting.

44. Heap inserts are typically faster than using a clustered index. This means that you may want to drop indexes and rebuild if you are changing a large part of the destination table; you will want to test your inserts both by keeping indexes in place and by dropping all indexes and rebuilding to validate.

45. Tune your network by setting up packet size of your connection. By default this value is set to 4,096 bytes. This means a new network package must be assemble for every 4 KB of data. Increasing the packet size will improve performance because fewer network read and write operations are required to transfer a large data set. but use this wisely

46. Set-based UPDATE statements - which are far more efficient than row-by-row OLE DB calls

47. Aggregation calculations such as GROUP BY and SUM. These are typically also calculated faster using Transact-SQL instead of in-memory calculations by a pipeline.

48. When you insert data into your target SQL Server database, use minimally logged operations if possible. When data is inserted into the database in fully logged mode, the log will grow quickly because each row entering the table also goes into the log

49. Do not use Sort unless absolutely necessary.

50. Configure the package with parameters. This helps you to deploy the package across the environment.

51. If you place more than one task on control flow pane and do not connect them by using precedence constraint it will run in Parallel. This will help to speed up the process.