# SQL Coding Standards

|  |  |  |  |
| --- | --- | --- | --- |
| Reviews | | | |
| Date | Version | Description | Author |
| 08-October-2019 | 0.1 | Document Creation | Vadim Sudenko |

# Index

[1 T-SQL 5](#_Toc6229526)

[1.1 General T-SQL Practices 5](#_Toc6229527)

[1.1.1 Optimizer Hints 6](#_Toc6229528)

[1.1.2 Derived tables and temporary tables 6](#_Toc6229529)

[1.1.3 Table variables 6](#_Toc6229530)

[1.1.4 Recompiles 6](#_Toc6229531)

[1.1.5 User Defined Functions 7](#_Toc6229532)

[1.1.6 Indexed Views 7](#_Toc6229533)

[1.1.7 OUTPUT Clause 7](#_Toc6229534)

[1.1.8 Common Table Expressions 8](#_Toc6229535)

[1.1.9 CLR 8](#_Toc6229536)

[1.1.10 XML 8](#_Toc6229537)

[1.1.11 JSON 8](#_Toc6229538)

[2 SQL Server Code 9](#_Toc6229539)

[2.1 General Considerations 9](#_Toc6229540)

[2.2 Scripts 9](#_Toc6229541)

[2.2.1 Naming conventions 9](#_Toc6229542)

[2.2.2 Considerations 9](#_Toc6229543)

[2.2.3 Structure 9](#_Toc6229544)

[2.2.4 Representation SQL 10](#_Toc6229545)

[2.2.5 [LN-SQL-STRC-029] Object documentation section 11](#_Toc6229546)

[2.2.6 [LN-SQL-STRC-030] Script initialization section 11](#_Toc6229547)

[2.2.7 [LN-SQL-STRC-031] Object and parameter declaration section 11](#_Toc6229548)

[2.2.8 [LN-SQL-STRC-032] Local variables declaration section 11](#_Toc6229549)

[2.2.9 [LN-SQL-STRC-033] Local variables and DLL initialization section 11](#_Toc6229550)

[2.2.10 [LN-SQL-STRC-034] Implementation Section 11](#_Toc6229551)

[2.2.11 [LN-SQL-STRC-036] Catch Section 11](#_Toc6229552)

[2.2.12 [LN-SQL-STRC-037] Termination Section 11](#_Toc6229553)

[3 Templates 13](#_Toc6229554)

[3.1 Template Types 13](#_Toc6229555)

[3.2 Template Functions 13](#_Toc6229556)

[3.3 Template Procedures 14](#_Toc6229557)

[4 Comments 15](#_Toc6229558)

[4.1 Considerations 15](#_Toc6229559)

[4.2 Classification 15](#_Toc6229560)

[4.2.1 Object Documentation 15](#_Toc6229561)

[5 Data Structures 16](#_Toc6229562)

[5.1 Considerations 16](#_Toc6229563)

[5.2 Classification 16](#_Toc6229564)

[5.3 Variables 16](#_Toc6229565)

[5.3.1 [LN-SQL-VAR-055] Declaration 16](#_Toc6229566)

[5.3.2 Representation 16](#_Toc6229567)

[6 Using Cursors 17](#_Toc6229568)

[6.1 Considerations 17](#_Toc6229569)

[7 Data Paging 18](#_Toc6229570)

[7.1 Considerations 18](#_Toc6229571)

[7.1.1 Example: 18](#_Toc6229572)

# Introduction

The goal of this document is to establish data manipulation and data definition rules for the Microsoft SQL Server engine; this will allow us to obtain the following benefits:

* To improve performance applications
* To accredit the data paging strategy
* Easy code maintenance
* Increased code readability
* To have a defined programming style
* To be able to measure the quality of programming
* New members of the team will rapidly become familiar with the existing applications

The severity color code used is the following:

|  |  |
| --- | --- |
| **Color** | **Description** |
|  | Critical errors |
|  | Serious errors |
|  | Low errors |

# T-SQL

## General T-SQL Practices

* **[LN-SQL-TSQL-013]** Only relevant data should be returned to the client. This means that the use of “Select \*” is prohibited, you must specify the columns to be returned.

*SELECT CompanyName*

*FROM Customers*

*WHERE CustomerID=1*

**Avoid queries with the wildcard \***

*SELECT \**

*FROM Customers*

*WHERE CustomerID=1*

* **[LN-SQL-TSQL-014]** The use of USE NOCOUNT ON at the beginning of the stored procedure or session is required in order to suppress the “row affected” messages sent to the client.
* **[LN-SQL-TSQL-015]** The choice of the operators that participates on a WHERE clause is important. This is the order in which operators must be used.
  + =
  + >, >=, <, <=
  + LIKE
  + <>
* **[LN-SQL-TSQL-016]** The operations performed on a WHERE clause is also important. This is the order in which operations must be used.
  + A single literal used by itself on one side of an operator.
  + A single column name or single parameter used by itself on one side of an operator.
  + A multi-operand expression on one side of an operator.
  + A single exact number on one side of an operator.
  + Other numeric number (other than exact), date and time.
  + Character data, NULLS.
* **[LN-SQL-TSQL-017]** Avoid WHERE clauses that are non-sargable (search argument, which refers to a WHERE clause that compares a column to a constant value). Non-sargable clauses cannot take advantage of existing indexes. Non-sargables clauses includes: "IS NULL", "<>", "!=", "!>", "!<", "NOT", "NOT EXISTS", "NOT IN", "NOT LIKE", and "LIKE '%500'", expressions that includes a function on a column, expressions that have the same column on both sides of the operator, or comparisons against a column (not a constant).

|  |  |
| --- | --- |
| *SELECT member\_number, first\_name, last\_name  FROM members  WHERE DATEDIFF(yy,datofbirth,GETDATE()) > 21*  *SELECT member\_number, first\_name, last\_name  FROM members  WHERE dateofbirth < DATEADD(yy,-21,GETDATE())* | *SELECT member\_number, first\_name, last\_name  FROM members*  *WHERE NOT column\_name > 5*  *SELECT member\_number, first\_name, last\_name  FROM members*  *WERE column\_name <= 5* |

* **[LN-SQL-TSQL-018] When having a choice of using the IN or the EXISTS clause**, the EXISTS clause should generally be used, as it is usually more efficient and performs faster.
* **[LN-SQL-TSQL-019] When you having a choice of using the IN or the BETWEEN clauses** in your Transact-SQL, the BETWEEN clause should generally be used, as it is much more efficient.
* If OR clauses are being used, it may be a good practice to create an index for each referenced column in the WHERE clause.
* **[LN-SQL-TSQL-020] A query with one or more OR clauses can sometimes be rewritten as a series of queries combined with a UNION ALL statement** in order to boost the performance of the query.

|  |
| --- |
| *SELECT employeeID, firstname, lastname  FROM names  WHERE dept = 'prod' or city = 'Orlando' or division = 'food'*  *SELECT employeeID, firstname, lastname FROM names WHERE dept = 'prod'  UNION ALL  SELECT employeeID, firstname, lastname FROM names WHERE city = 'Orlando'  UNION ALL  SELECT employeeID, firstname, lastname FROM names WHERE division = 'food'* |

### Optimizer Hints

* **[LN-SQL-HINT-021]** Optimizer hints should be avoided in the WHERE clauses. If a hint is included in the query, this forces the Query Optimizer to become static, preventing the Query Optimizer from dynamically adapting to the current environment for the given query.
* If a hint might be necessary to optimize a query, be sure to perform all of the following first:
  + Update the statistics on the relevant tables.
  + If the problem query is inside a stored procedure, recompile it.
  + Review the search arguments to see if they are sargable, and if not, try to rewrite them so that they are sargable.
  + Review the current indexes, and make changes if necessary. If this does not resolve the issue, then an optimizer hint can be used.
* **[LN-SQL-HINT-022]** The only query hint that may be used freely is NOLOCK.
* JOIN hints (loop, merge and hash) should only be used as a last resort.

### Derived tables and temporary tables

* Temporary tables provide a useful way to create intermediate results, but they cause recompilations, need additional disk resources and depending on the recovery model, operations on them are fully logged causing performance to be reduced.
* [LN-SQL-TT-023] When creating a temporary table, always use the SELECT…INTO statement as it does not log the operation, this increase the speed of the query.
* Temporary tables can be avoided using derived tables (the result of using another SELECT statement in the FROM clause of another SELECT statement), this increases the performance of the query. The rule is when a sub select is used more than once in the query use temporary tables, otherwise use derived tables.

### Table variables

* Table variables use tempdb and disk resources, but on the other hand operations performed on these variables are not logged, these in fact offer a significant performance over other alternatives.
* Procedures with table variables do not require recompiles, a precompiled script offer a performance advantage over procedures that need to be constantly recompiled.
* Transactions on table variables are different from temporary tables, table variables are not affected by rollbacks. Every operation on a table variable is committed immediately.
* Table variables exists only in the same scope as variables (not visible to inner stored procedures).
* There are no statistics based recompiles for table variables.
* [LN-SQL-TV-024] Use temporary tables when operating on large datasets and table variables for small datasets with frequent updates.
* Table variables can only be indexed by using a primary key or a constraint, while temporary tables can be indexed as a normal table.

### Recompiles

* Recompilation of the stored procedures, triggers, functions, etc. is a required operation inside any database. This recompilations occurs for a number of reasons, but there are certain tips that must be follow.
  + Minimize the amount of schema changes during stored procedure execution.
  + If dynamic T-SQL is needed instead of a stored procedure, avoid using EXECUTE, instead use the SP\_EXECUTESQL stored procedure.
  + Establish the correct SET options during at connection time and ensure that they do not change during the connection. (ANSI\_NULL\_DFLT\_OFF, ANSI\_NULL\_DFLT\_ON, ANSI\_NULLS, ANSI\_PADDING, ANSI\_WARNINGS, ARITHABORT, CONCAT\_NULL\_YIELDS\_NULL, DATEFIRST, DATEFORMAT, FORCEPLAN, LANGUAGE, NO\_BROWSETABLE, NUMERIC\_ROUNDABORT, QUOTED\_IDENTIFIERS causes recompilation).
  + Sometimes, depending on the variety of parameters received by a stored procedure using “CREATE PROCEDURE… WITH RECOMPILE” can help.
  + Because a change in cardinality of a table variable does not cause recompilations, consider using a table variable instead of a temporary table.
  + The KEEP PLAN query hint changes the recompilation thresholds for temporary tables, and makes them identical to those for permanent tables. Therefore, if changes to temporary tables are causing many recompilations, this query hint can be used.

*SELECT B.col4, sum(A.col1)*

*FROM dbo.PermTable A INNER JOIN #TempTable B ON A.col1 = B.col2*

*WHERE B.col3 < 100*

*GROUP BY B.col4*

*OPTION (KEEP PLAN)*

* + To avoid recompilations due to plan optimality-related (statistic update-related) reasons totally, KEEPFIXED PLAN query hint can be specified.

*SELECT c.TerritoryID, count(\*) as Number, c.SalesPersonID*

*FROM Sales.Store s INNER JOIN Sales.Customer c*

*ON s.CustomerID = c.CustomerID*

*WHERE s.Name LIKE '%Bike%' AND c.SalesPersonID > 285*

*GROUP BY c.TerritoryID, c.SalesPersonID*

*ORDER BY Number DESC*

*OPTION (KEEPFIXED PLAN)*

* + Turning off automatic updates of statistics for indexes and statistics defined on a table or indexed view will ensure that plan optimality-related recompilations caused by those objects will stop.

### User Defined Functions

* [LN-SQL-UDF-025] Try to avoid User Defined Scalar Functions in SELECT statements as they will be evaluated as many times as the SELECT statement.

### Indexed Views

* **[LN-SQL-IV-026]** When accessing VIEWS inside a database, it should be consider adding a unique clustered index to them, in order to significantly improve the performance. You can also create non-clustered indexes.
* When modifying any table of the view, the clustered index and any non-clustered indexes of the view are also modified so that it is always up to date.
* Use Indexed Views on:
  + Data marts, data warehouses, decision support, data mining, OLAP applications.
  + Views that join two or more large tables.
  + Views that aggregate data.
  + Repeated patterns of queries.
  + For tables that would benefit from multiple clustered indexes.

### OUTPUT Clause

* OUTPUT clause allow the developer to have access to the virtual tables INSERTED and DELETED, so it can return values to the client clause or can be used during write operations to identify the actual rows affected by the statement.
* The OUTPUT clause may be useful to retrieve the value of identity or computed columns after an INSERT or UPDATE operation.

*--Creating the table which will store permanent table*

*CREATE TABLE TestTable (ID INT, TextVal VARCHAR(100))*

*--Creating temp table to store ovalues of OUTPUT clause*

*DECLARE @TmpTable TABLE (ID INT, TextVal VARCHAR(100))*

*--Insert values in real table as well use OUTPUT clause to insert*

*--values in the temp table.*

*INSERT TestTable (ID, TextVal)*

*OUTPUT Inserted.ID, Inserted.TextVal INTO @TmpTable*

*VALUES (1,'FirstVal')*

*INSERT TestTable (ID, TextVal)*

*OUTPUT Inserted.ID, Inserted.TextVal INTO @TmpTable*

*VALUES (2,'SecondVal')*

*--Check the values in the temp table and real table*

*--The values in both the tables will be same*

*SELECT \* FROM @TmpTable*

*SELECT \* FROM TestTable*

*GO*

***--ResultSet 1:***

ID TextVal

———– ————

1 FirstVal

2 SecondVal

ID TextVal

———– ———–

1 FirstVal

2 SecondVal

### Common Table Expressions

* Common Table Expressions (CTE) help to perform several aggregate functions on a single data set or to define views that need to be stored during the execution of a batch.
* Its scope is limited to the statement that defines it.
* CTE’s can be recursive (hierarchical data) or non-recursive (replacement for views and derived tables).
* It provides a convenient tool for creating queries by building tables as and when they are required within nested SELECT statements.

*WITH MyCTE (LP, SP) AS   
(   
  SELECT LP, LP \* .95 FROM itemlist.item   
)*

*SELECT \* FROM My\_CTE*

*WITH DictRep(Log\_ID, Mng\_ID, Emp\_ID) AS   
(   
    SELECT Log\_ID, Mng\_ID, Emp\_ID   
    FROM HRs.Emp   
    WHERE Mng\_ID IS NULL   
    UNION ALL   
    SELECT e.Log\_ID, e.Mng\_ID, e.Emp\_ID   
            FROM HRs.Emp e   
    INNER JOIN DictRep d   
    ON e.Mng\_ID = d.Emp\_ID   
)   
SELECT \* FROM DictRep ;*

### CLR

* CLR provides various functions and services required for program execution, including just-in-time (JIT) compilation, allocating and managing memory, enforcing type safety, exception handling, thread management, and security.
* Advantages of CLR integration:
  + Better programming model.
  + Improved safety and security.
  + Ability to define data types and aggregate functions.
  + Streamlined development through a standardized environment.
  + Potential for improved performance and scalability.

### XML

* A new XML type is introduced.
* The new XML type can be used to process xml data inside the database.
* **[LN-SQL-XML-027]** The use of XQuery is now supported and should be preferred instead of the OPENXML or FOR XML functions.

### JSON

* The use of JSON is not allowed for the formatting of the return value of a stored procedure. Only a tabular or integer value are allowed.

# SQL Server Code

The code within SQL Server is organized in text files which contain:

* Scripts
* Commentaries
* Data Structures
* Data Definition Language (DDL)
* Control Structures
* Data Manipulation Language (DML)
* Error handling
* Data types
* Nomenclature

## General Considerations

* Use consistent and descriptive identifiers and names
* Make judicious use of white space and indentation to make code easier to read.
* Try to use only standard SQL functions instead of vendor specific functions for reasons of portability.
* Include comments in SQL code where necessary. Use the C style opening “/\*” and closing “\*/” where possible otherwise precede comments with “--” and finish them with a new line.
* Ensure the name is unique and does not exist as a reserved keyword.
* Keep the names length to a maximum of 30 bytes—in practice this is 30 characters unless you are using multi-byte character set.
* Names must begin with a letter and may not end with an underscore.
* Only use letters, numbers and underscores in names.
* Avoid abbreviations and if you have to use them make sure they are commonly understood.
* The script name should have the same name of the type, function or procedure created.
* An error handling routines should be defined in case an error arises or may arise.
* All the names of the objects should be defined as **dbo**.*Name* .

## Scripts

Each object of an application within SQL Server must be programmed using a script, which is a text file that contains the necessary code for the definition and use of the object.

### Naming conventions

* **Type**
  + ty\_typename
* **Functions**
  + fn\_functionName
  + The name must contain a verb ex. “get, set, etc.”.
* **Procedures**
  + usp\_procedureName
  + The name must contain a verb ex. “get, ins, del, etc.”.

### Considerations

* Each script must contain a single functionality.
* Each script must contain the code necessary to assure that if the object already exists, it is previously deleted to avoid errors.
* Each script must contain the code necessary to create the necessary temporary or intermediate objects.
* Each code file should be properly standardized and comply with the norms established in this document of standards.
* The script should use the environment variables as required and make sure to reestablish them at the end of the script.
* If the script uses DDL (Data Definition Language), it is very important to define DDL at the beginning of the script before any logic, otherwise, the data base engine will have to recreate the execution plan every time the script is called or executed.

### Structure

Stored procedures must follow this structure within the script:

**SQL:**

1. Object documentation section.
2. Script Initialization section.
3. Object and parameter declaration section.
4. Local variables declaration section.
5. Local variables and DDL initialization section.
6. Implementation Section.
7. Catch Section.
8. Termination section.

### Representation SQL

IF EXISTS (SELECT \* FROM sysobjects WHERE id = object\_id('[dbo].[usp\_VENTRANObjectName]')

AND OBJECTPROPERTY(id, 'IsProcedure') = 1)

Script initialization section

BEGIN

EXEC ('DROP PROCEDURE [dbo].[usp\_VENTRANObjectName] )

END

GO

CREATE PROCEDURE usp\_VENTRANObjectName

@campo1 VARCHAR(3)

Object and parameter declaration section.

@campo2 INT

AS

--\*--

Object documentation section

--\*-Object: usp\_VENTRANObjectName

--\*-Author: Author Name

--\*-Date:

--\*-Objective: Objective of the Object

--\*-Input: List of input parameters, with its description

--\*-Output: List of output parameters, with its description

--\*-Preconditions:

--\*-Return:

--\*-Revisions:

--\*- Movement - Modified by – Date of the Movement

--\*--

--\*-- Declaration of Local Variables

DECLARE @var1 NVARCHAR(10);

Local variables declaration section.

CREATE TABLE #TTable1( Field1 INT NOT NULL,

Field2 INT NOT NULL );

--\*-- Initialization of the object

Local variables and DDL initialization section.

SET NOCOUNT ON

BEGIN TRY

INSERT INTO #TTable1 ...

BEGIN TRANSACTION;

--\*-- Body object

Error handling section.

INSERT INTO Table1(campo1, campo2)

FROM #TTable1 …

INSERT INTO TABLE2(column1, column2)

VALUES (@campo1, @campo2);

...

...

...

COMMIT;

END TRY

Implementation section.

BEGIN CATCH

IF @@TRANCOUNT > 0 ROLLBACK;

THROW;

END CATCH;

--\*-- Ending object

Termination section.

SET NOCOUNT OFF;

RETURN

### [LN-SQL-STRC-029] Object documentation section

* Contains all the necessary information to identify the object created by the user.
* Is located at the beginning of the text file.
* Must fulfill the Standards of Object Documentation.
* This section is required.

### [LN-SQL-STRC-030] Script initialization section

* Contains the necessary code to validate the existence of the stored procedure. If the stored procedure exist then it will be droped.

### [LN-SQL-STRC-031] Object and parameter declaration section

* Contains all the declaration of the object indicating
  + Type of Object (Stored Procedure)
  + Descriptive Name
  + Parameters
    - Type of Reference (Entered and/or Exit)
    - descriptive Name of the parameter
    - Data type
* Using the LN-SQL-STRC-006, the object will exist by the time this section is executed, so an ALTER statement must be used instead of a CREATE statement.
* This section is required.

### [LN-SQL-STRC-032] Local variables declaration section

* Contains all the declarations of data structures that could be referenced only within the object.
* Is located after the object and parameter declaration.
* Must comply with the variable naming standards (shown later in this document).
* This section is optional.

### [LN-SQL-STRC-033] Local variables and DLL initialization section

* Contains the code to initialize local variables.
* Contains the code to create or initialize local objects.
* Contains the allocation of environment variables if used.
* It is located after the local variables declarations section.
* This section is required, if the stored procedure uses local objects, variables or returns a value.
* All objects must be created and initialized.

### [LN-SQL-STRC-034] Implementation Section

* Contains the code with the necessary logic for the implementation of the operation for which the object was created.
* It is delimited between the initialization section and the termination section.
* The code within this section must comply with the Control Structures Standards and the Data Manipulation Language Standards.
* The code within this section must be properly commented using the Block Comment Standards or Line Comment Standards.
* This section is required.
* [LN-SQL-STRC-035] Contains the BEGIN TRY and END TRY section that handles the exceptions inside the implementation

### [LN-SQL-STRC-036] Catch Section

* Contains the code with the necessary logic to return the error information.

### [LN-SQL-STRC-037] Termination Section

* Contains the code to assign the return value if needed.
* Contains the code for the destruction of all the used local objects.
* Contains the code to return environment variables to their original values.
* **[LN-SQL-STRC-038]** It is located at the end of the implementation section and must finish with the RETURN statement.
* **[LN-SQL-STRC-039]** This section is required, if the stored procedure modifies environment variables, uses local objects, local variables or returns a value.

# Templates

## Template Types

|  |  |
| --- | --- |
|  | if object\_id('dbo.ty\_TypeName') is not null begin  print 'dropping functon dbo.ty\_ TypeName';  drop TYPE dbo.ty\_TypeName;  end  go  CREATE TYPE ty\_TypeName AS TABLE  ( OrderId INT NULL );  GO |

## Template Functions

|  |  |
| --- | --- |
|  | if object\_id('dbo.fn\_FunctionName') is not null begin  print 'dropping procedure 'dbo.fn\_FunctionName ';  drop function 'dbo.fn\_FunctionName;  end  go  create function dbo.fn\_FunctionName  (  @pClientId int=null,  @pOrderId int=null  )  RETURNS TABLE  AS  /\*  //////////////////////////////////////////////////////////////////////  //   Description:  //         Input:  //        Output:  //   Return code:  //          Note:  //////////////////////////////////////////////////////////////////////  //          History: Date   Author          Comment  //          -----------     --------        ----------------------  //          dd/MMM/YYYY     ABC             Free comment  \*/  RETURN(SELECT ..  ) |

## Template Procedures

|  |  |
| --- | --- |
|  | if object\_id('dbo.usp\_ProcedureName') is not null begin  print 'dropping procedure dbo.usp\_ProcedureName';  drop procedure dbo.usp\_ProcedureName;  end  go  create procedure dbo.usp\_ProcedureName  (  @pClientId int=null,  @pOrderId int=null  )  as  begin  /\*  //////////////////////////////////////////////////////////////////////  //   Description:  //         Input:  //        Output:  //   Return code:  //          Note:  //////////////////////////////////////////////////////////////////////  //          History: Date   Author          Comment  //          -----------     --------        ----------------------  //          dd/MMM/YYYY     ABC             Free comment  \*/    print '';  end;  go    if object\_id('dbo.usp\_ProcedureName') is not null begin  print 'granting permissions on dbo.usp\_ProcedureName';  GRANT EXECUTE ON dbo.usp\_ProcedureName TO cnxuser;  end  go |

# Comments

## Considerations

* **[LN-SQL-COMM-043]** Comments should being with the ”--“ character (dash dash).
* **[LN-SQL-COMM-044]** With large comment blocks, /\* should be used to begin the comment and \*/ should be used to finish the comment.
* If a comment occupies more than one line the subsequent lines must begin in the same column as the first line.
* All scripts of code must be documented.
* **[LN-SQL-COMM-045]** All the control structures must be commented.
* **[LN-SQL-COMM-046]** All the data manipulation must be commented.
* **[LN-SQL-COMM-047]** Any code section than can be confusing or difficult to understand must be commented.

## Classification

Object Documentation

Section Comment

Block Comment

Line Comment

### Object Documentation

It contains all the necessary information to identify the object created by the user

* It is located at the beginning of the text file
* **[LN-SQL-DOC-048]** It must be delimited by the characters (--\*-) to begin and finish the section
* Begins on the first column
* **[LN-SQL-DOC-049]** The documentation must include the following information:
  + Object: Object Key- Object Name
  + Author:
  + Date:
  + Objective: Description of the purpose of the object
  + Entrance: List of input parameters
  + Exit: List of output parameters
  + Preconditions: List of necessary preconditions for the operation of the object
  + Return: Value of return expected by the execution of the object
  + Log: Movement Done - /XXX - Date Movement

# Data Structures

It contains the declaration of all the data structures required by the stored procedure, a data structure is any variable, object or constant that stores some value.

## Considerations

* Any declared data structure must comply with the Naming Standards (located later on this document).

## Classification

Variables

Control Structures

## Variables

### [LN-SQL-VAR-055] Declaration

* Declare the variables using DECLARE in each line or separating variables with commas, variables must be aligned in the same column, variable data types in different lines should be aligned in the same column.

### Representation

--\*Local Variables

DECLARE @variableName DataType

*Examples:*

*DECLARE @counter smallint*

*DECLARE @error bit,*

*@Quantity int*

# Using Cursors

## Considerations

* **[LN-SQL-CUR-085]** The use of cursors **is not allowed.**
* In case of requiring this kind of operations, this should be validated with the project’s technical leader, architect or Database Manager.
* Cursors consume large amounts of resources on the server, the use of nested cursors may even block the data base.
* An alternate way to avoid cursors is to use a temporary table with a sequential column (identity).

# Data Paging

## Considerations

* **[LN-SQL-PAG-089]** The following parameters should have default values:
  + @Page : Number of requested page
  + @Fetch : Size of the page in rows

### Example:

create procedure [dbo].[spVENSELGetstoresPaging]

@**Page** int = 1,

@**Fetch** int = 30

as

SET NOCOUNT ON;

-- Calculating the offset of rows

DECLARE @Offset AS INTEGER = (@Page - 1) \* @Fetch;

-- Query that returns the requested page

SELECT

stor\_id ,

stor\_name,

stor\_address,

city,

state,

zip

FROM Table

ORDER BY Zip

OFFSET @Offset ROWS

FETCH NEXT @Fetch ROWS ONLY;

SET NOCOUNT OFF;

Basic paging parameters:

@Page, what is the required page

@Fetch, what is the size in rows of the page

The query will just return the required page