Estándar de Programación en Transact-SQL

Contenido

Introducción

El presente documento tiene como objetivo definir un estándar de estilo de programación para el lenguaje de consultas Transact-SQL para las aplicaciones realizadas por desarrolladores. La necesidad principal de definir un estándar, nace de la realidad de generar desarrollos de software modularizados, por grupos de personas que no se encuentran ubicados geográficamente en la misma oficina, ni en la misma ciudad.

Además, los estándares brindan ventajas alternativas que enriquecen la calidad del servicio brindado, mejoran la calidad del software y facilitan notablemente el posterior mantenimiento, principalmente cuando cambian los integrantes de los equipos de desarrollo.

El lenguaje Transact-SQL es una extensión del lenguaje de consultas SQL, para comunicarse con instancias de servidores Microsoft SQL Server.

Metodología

Para el lenguaje de consultas Transact-SQL no existen estándares bien definidos, por lo tanto la mayoría de las empresas crean un estilo propio. De todos modos los estilos de codificación se parecen bastante, ya que derivan del utilizado por Microsoft, el mayor cambio se presenta en la nomenclatura de los diferentes objetos que forman la base de datos (tablas, vistas, procedimientos, etc.) incluso la nomenclatura de la misma base de datos.

Herramientas

No se va a utilizar una herramienta particular para verificar el estilo de codificación de las consultas SQL, con lo cual el programador deberá leer el estándar y acostumbrarse a su utilización. Para ayudar en esta tarea, el entorno del SQL Management Studio trae Intellisense, que ayuda en la utilización de las mayúsculas y minúsculas correctas en los objetos de la base de datos.

Convenciones

Objetos de la Base de Datos

Se entiende por objetos de la base de datos a todas las entidades que se pueden crear por el programador, como ser tablas, campos, procedimientos, etc. Al darle nombre a estos objetos, se suelen utilizan palabras compuestas, es decir, dos o más palabras que deben unirse en una sola palabra. Para estos casos se utiliza la convención PascalCase en lo que respecta a letras mayúsculas y minúsculas.

Sentencias Transact-SQL

Son todas las palabras reservadas del lenguaje. Ej: SELECT, UPDATE, FROM, etc. La convención más común es utilizar UpperCase.

Pascal Case

La primera letra de cada palabra comienza con mayúscula y el resto de las letras en minúsculas. Ej: MiTabla. Se utiliza para todo tipo de declaraciones en la base de datos, incluso para darle nombre a la base de datos.

Upper Case

Todas las letras de la palabra se escriben en mayúsculas. Para palabras compuestas, se separan con un guión bajo ("_"). Ej: UPDATE. Se utiliza para todas las palabras reservadas del lenguaje Transact-SQL.

Recomendaciones

Se recomienda seguir todas las reglas a no ser que exista una buena razón para no hacerlo. En cuanto a la documentación, por lo menos se recomiendan comentarios básicos en los encabezados de los procedimientos. No se recomienda comentar cosas obvias ya que el exceso de comentarios muchas veces dificulta la comprensión del código.

Código Heredado

Es muy fácil implementar este estándar en un proyecto nuevo en el cual que se parte desde cero. Para proyectos avanzados o terminados, se podría correr un análisis que informe la cantidad de inconsistencias y así determinar el tiempo que demandaría la corrección del código para apegarse al estándar.

Reglas

Se dividieron las reglas a utilizan en tres categorías: de Nomenclatura, de Codificación y de Performance. Las de codificación tienen que ver con la utilización del estándar en cuanto a la escritura de un código normalizado y fácil de entender. Las de performance son recomendaciones o buenas prácticas de programación para evitar problemas de performance en la base de datos.

Reglas de Nomenclatura

Las reglas de nomenclatura fueron divididas en los distintos objetos que se pueden crear en una base de datos, y en las consultas posteriores. Para la nomenclatura de todos los objetos de base de datos se utiliza PascalCase y los nombres deben ser representativos y no incluir abreviaturas ni diminutivos, ya que los distintos programadores pueden abreviar la misma palabra de diferentes maneras.

Tablas

Las tablas representan entidades al igual que las clases en los lenguajes de programación. Por lo tanto se utiliza la misma sintaxis, sin prefijos. Por ejemplo para referenciar a los usuarios de un sistema, se diseña la entidad Usuario, que se representa en una clase llamada Usuario y se almacena en una tabla llamada Usuario. Si el sistema está dividido en módulos que pueden referirse a distintas partes del negocio, y que pueden llegar a necesitar tablas con el mismo nombre, se recomienda utilizar esquemas para agrupar las tablas, en lugar de prefijos personalizados. Continuando con el mismo ejemplo, si tenemos usuarios en el módulo de “Ventas” y otro tipo de usuarios en el módulo de “Seguridad”, se crean los esquemas respectivos, y a las tablas se accede anteponiendo siempre el nombre del esquema al cual pertenece: [Ventas].[Usuario] y [Seguridad].[Usuario].

Columnas, Campos y Variables

Las columnas de una tabla son los atributos de una entidad, por lo tanto deben ser descriptivos y representativos del dato que contienen. Por ejemplo para la tabla “Usuario”, las columnas pueden ser Nombre, Apellido, etc. No llevan prefijos, ya que están contenidos dentro de un objeto que los representa (la tabla).

Para el caso de las variables es exactamente lo mismo, la única diferencia es que las mismas deben llevar como prefijo el carácter @ (arroba). Por ejemplo: @Nombre o @PuntoDeMedicion.

Vistas

Las vistas no son ni más ni menos que representaciones de tablas, con lo cual se utilizan las mismas convenciones. En muchos casos las vistas están formadas por la unión (JOIN) de dos o más tablas, por lo tanto el nombre de la vista no siempre representa una única entidad. Por ejemplo: Si se tiene que unir la tabla "PuntoDeMedicion" con la tabla "InstrumentoDeMedicion", a la vista se le puede dar el nombre "InstrumentosPorPuntoDeMedicion".

Las vistas también suelen utilizarse para resumir o totalizar valores de tablas o generar reportes. En estos casos se le pueden anteponer prefijos que indiquen el resultado como "TotalVentasPorMes" o "ResumenPromediosPorAlumno". Siempre utilizando PascalCase.

Procedimientos

Los stored procedures siempre generan acciones, con lo cual su nombre debe describir lo que realizan. Las cuatro acciones más comunes son seleccionar, insertar, modificar y eliminar datos. Aunque también realizan procesos que no devuelven resultados, como ser tareas de mantenimiento o backups. A continuación se detallan los prefijos a utilizar para identificar cada acción. Los prefijos están en inglés por convención.

Acción	Prefijo	Ejemplo
Obtener datos (SELECT)	Get	GetUsuarios
Insertar registros (INSERT)	Ins	InsInstrumento
Actualizar registros (UPDATE)	Upd	UpdPersona
Eliminar registros (DELETE)	Del	DelPuntoDeMedicion
Procesos	Proc	ProcCalcularIndicadores

Para agrupar procedimientos se utilizan esquemas al igual que para las tablas. Incluso los procedimientos que realicen acciones sobre tablas de un esquema deben ir en el mismo esquema.

Funciones

Para las funciones se utiliza el mismo criterio que para los procedimientos. No hace falta anteponer prefijos que indiquen si es una función o un procedimiento ya que se sobreentiende con solo ver la forma en que se utilizan.

Triggers

Los triggers son básicamente stored procedures, pero asociados a una tabla en particular. Es por esto que lo mejor para su identificación es que lleven como prefijo el nombre de la tabla a la cual pertenecen. Las acciones que disparan de los triggers pueden ser de inserción, actualización o borrado de registros. Por lo tanto luego del nombre de la tabla se deberá identificar la acción del trigger utilizando los mismos diminutivos que se utilizan como prefijos de los procedimientos: Ins, Upd y Del. En caso de triggers que se disparen para más de una acción, se concatenan. Como sufijo se agrega la palabra Trigger. Por ejemplo un trigger de borrado sobre la tabla "PuntoDeMedicion" del esquema "Mediciones" se debe nombrar de la siguiente manera: "[Mediciones].[PuntoDeMedicionDelTrigger]". Un ejemplo de trigger combinado podría ser el de inserción o actualización de un usuario: "[Ventas].[UsuarioInsUpdTrigger]".

Reglas de Codificación

Las reglas de codificación contemplan el orden y legibilidad de una consulta o código fuente. Indican como indentar las sentencias, y la utilización de diferentes bloques de código.

Bloques BEGIN…END

Todas las sentencias de código que tengan que agruparse dentro de un bloque BEGIN...END, deben ir indentados con una tabulación. Las palabras reservadas BEGIN y END, no van indentadas, sino en la misma columna que el código que las precede.

```
IF (@Cantidad > 100)
BEGIN
	SET @Cantidad = 100
	PRINT @Cantidad
END
```

Así mismo los bloques anidados van incrementando la indentación a medida que se incrementando el nivel de anidamiento.

```
IF (@Cantidad < 1000)
BEGIN
	PRINT 'Rango 1'
END
ELSE
BEGIN
	IF (@Cantidad < 2000)
	BEGIN
		PRINT 'Rango 2'
	END
	ELSE
	BEGIN
		PRINT 'Rango 3'
	END
END
```

Consultas

Al tener que declararse explícitamente todos los campos de una tabla al hacer una consulta, en ocasiones pueden llegar a ser muy extensa la línea, con lo cual los campos deberán escribirse uno debajo del otro. Todos los campos deben ir indentados un nivel, debajo de la sentencia SELECT. La sentencia FROM va al mismo nivel que el SELECT, pero la tabla debe ir indentada un nivel.

Lo mismo para los filtros de una consulta deben ir indentados un nivel debajo de la sentencia WHERE, y en caso de ser más de un filtro, se colocarán uno debajo del otro.

```
SELECT
	Campo1,
	Campo2
FROM
	Tabla
WHERE
	Campo3 = Filtro1
	AND Campo4 = Filtro2
```

Declaración de Variables

Las variables a declarar irán una debajo de la otra con un nivel de indentación, debajo de la sentencia DECLARE.

```
DECLARE
	@Id AS INT,
	@Nombre AS VARCHAR(50)
```

Funciones

Las funciones, ya sean nativas de SQL o programadas, se utilizan de la misma manera que los campos, y todos los parámetros que lleven van en la misma línea.

```
SELECT
	Nombre AS NombreCompleto,
	SUBSTRING(Nombre, 1, 5) AS NombreAbreviado
FROM
	Usuario
```

Uniones (JOINS)

Al realizar uniones entre tablas, cada tabla deberá utilizar la misma indentación que la tabla principal, y cada sentencia JOIN, irá al mismo nivel que la sentencia FROM. La sentencia ON, junto con todas las condiciones de unión irán en la misma línea que la tabla. Todos los campos a seleccionarse deberán llevar siempre el prefijo de la tabla a la cual pertenecen. En estos casos se recomienda utilizar alias para que el código no quede tan extenso.

```
SELECT
      T1.Campo1,
      T1.Campo2
FROM
      Tabla1 AS T1
INNER JOIN
      Tabla2 AS T2 ON T1.Campo1 = T2.Campo1
WHERE
      T1.Campo3 = Filtro1
      AND T1.Campo4 = Filtro2
```

Concatenaciones (UNION)

Al concatenar dos consultas, cada una de ellas irá con la indentación normal vista hasta ahora. La sentencia de concatenación UNION irá al mismo nivel que las consultas, pero separada por una línea en blanco antes y después de la misma.

```
SELECT
      Campo1,
      Campo2
FROM
      Tabla1

UNION ALL

SELECT
      Campo1,
      Campo2
FROM
      Tabla2
```

Transacciones (TRAN)

Las sentencias que indican el comienzo y fin de las transacciones no indican un bloque con lo cual el código que sigue a continuación no va indentado, salvo que se encuentre dentro de un bloque BEGIN…END.

```
BEGIN TRAN MiTransaccion
INSERt INTO Usuario (Id, Nombre)
VALUES (@Id, @Nombre)
IF (@@ERROR = 0)
BEGIN
```

```
      COMMIT TRAN MiTransaccion
END
ELSE
BEGIN
      ROLLBACK TRAN MiTransaccion
      RAISERROR ('No se pudo insertar el registro.', 16, 1)
END
```

Orden (ORDER BY)

Los campos que forman parte del orden de la consulta, van indentados de la misma manera que los campos de la sentencia FROM.

```
SELECT
      Campo1,
      Campo2
FROM
      Tabla1
ORDER BY
      Campo1
```

Agrupamiento (GROUP BY)

Los campos que se utilizan para agrupar y totalizar, van indentados de la misma manera que los campos de la sentencia FROM. Lo mismo para los filtros aplicados sobre los grupos a través de la sentencia HAVING. Las funciones de agregado (SUM, AVG, MAX, etc) se utilizan igual que las columnas de una tabla.

```
SELECT
      MAX(Valor) AS ValorMaximo,
      COUNT(Valor) AS Cantidad
FROM
      Tabla1
GROUP  BY
      Campo1
HAVING
      COUNT(Valor) > 2
```

Actualizaciones (UPDATE)

La tabla a actualizar a través de la sentencia UPDATE va debajo de la misma y con un nivel de indentación. Los campos de la sentencia SET van uno debajo del otro con un nivel de indentación. El resto de las sentencias que forman parte de la actualización, siguen las mismas reglas que para las consultas.

```
UPDATE
      Usuario
SET
      Nombre = @Nombre,
```

```
      Apellido = @Apellido
WHERE
      Id = @Id
```

Inserciones (INSERT)

La sentencia INSERT debe incluir siempre todos los campos a insertar y van en la misma línea. La sentencia VALUES irá en la línea siguiente con todos los valores a insertar.

```
INSERT INTO Usuario (Id, Nombre, Apellido)
VALUES (@Id, @Nombre, @Apellido)
```

Consultas anidadas

Al utilizar consultas anidadas, ya sean sub-consultas o tablas derivadas, se utilizarán los bloques de anidamiento como si se tratase de un bloque BEGIN…END, con la diferencia de que las sentencias BEGIN y END son reemplazados por los paréntesis que abren y cierran el bloque.

```
SELECT MIN(Promedio)
FROM
(
    SELECT TOP 2 Promedio
    FROM Alumnos
    ORDER BY Promedio DESC
) AS A
```

Reglas de Performance

Las siguientes recomendaciones están orientadas a hacer un buen uso del motor de base de datos para no perder performance debido a malas prácticas de programación.

- Tener siempre en mente la performance al diseñar base de datos y las consultas. Visualizar siempre el plan de ejecución que trae la herramienta “Management Studio” para comprobar que siempre se realicen búsquedas en índices (index seek) en lugar de escaneos de tablas (table scan) o de índices (index scan).

- Nunca utilizar “SELECT *” en las consultas, detallar siempre los nombres de los campos.

- Lo mismo para las inserciones en datos en las tablas, siempre enumerar los campos a insertar y el orden de inserción de los mismos, por ejemplo INSERT INTO Tabla1 (Campo1, CampoN) VALUES (Valor1, ValorN).

- Utilizar siempre el nombre del esquema al que pertenece una tabla al consultarla, ya que es mucho más fácil encontrarla para el motor de base de datos. Por ejemplo SELECT Campo1 FROM [Esquema1].[Tabla1].

- Evitar el uso de cursores siempre que sea posible. Muchas veces los cursores pueden evitarse utilizando consultas un tanto más complejas, pero ofrecen mejor performance.

- Evitar la creación de tablas temporales ya que requieren de mucha escritura en disco. Alternativamente se pueden utilizar variables de tipo TABLE, tablas derivadas o vistas.

- Evitar el uso del carácter comodín "%" al principio de las cadenas, al filtrar utilizando la sentencia LIKE. Esto fuerza un escaneo del índice (index scan), lo cual va en contra de la utilización de índices. En cambio la utilización de "%" entre la cadena o al final de la misma, utiliza el índice (index seek).

- Intentar evitar la utilización de los operadores "NOT" y "<>" ya que realizan escaneos completos sobre las tablas e índices.

- Utilizar tablas derivadas en lugar de subconsultas, ya que mejoran muchísimo la performance. Por ejemplo la siguiente consulta busca el menor promedio de los dos mejores promedios:

Utilizando IN	Utilizando Tablas Derivadas

`SELECT MIN(Promedio)`
`FROM Alumnos`
`WHERE IdAlumno IN`
`(`
`SELECT TOP 2 IdAlumno`
`FROM Alumnos`
`ORDER BY Promedio DESC`
`)`
	`SELECT MIN(Promedio)`
	`FROM`
	`(`
	`SELECT TOP 2 Promedio`
	`FROM Alumnos`
	`ORDER BY Promedio DESC`
	`) AS A`

- En entornos productivos, utilizar siempre “SET NOCOUNT ON” al comienzo de las consultas, ya sean procedimientos, triggers, etc. Esto evita mensajes innecesarios que genera el motor de base de datos luego de las ejecuciones de sentencias como SELECT, INSERT, UPDATE y DELETE.