

Reglas de estilos SQL

Código del documento

10/03/2022

Dawid Mateusz

Departamento IT

Contenido

[1. Introducción 5](#_Toc98252141)

[2. Configuración de Management Studio 5](#_Toc98252142)

[2.1 Saltos de línea y retorno de carro 5](#_Toc98252143)

[2.2 Tabulaciones 6](#_Toc98252144)

[2.3 Opciones para diseñadores de tablas y autorecuperación 6](#_Toc98252145)

[2.4 Atajos para cambiar de conexión 8](#_Toc98252146)

[2.5 Mapa de desplazamiento 8](#_Toc98252147)

[2.6 Anclaje de pestañas 9](#_Toc98252148)

[2.7 Mensaje de hora de finalización 9](#_Toc98252149)

[3. Reglas de nomenclatura 10](#_Toc98252150)

[4. Reglas de sangrado y estructura de código 10](#_Toc98252151)

[4.1 Tabulaciones 10](#_Toc98252152)

[4.2 Palabras reservadas en consultas 10](#_Toc98252153)

[4.3 Palabras reservadas en bloques de código 10](#_Toc98252154)

[4.4 Bloque SELECT 11](#_Toc98252155)

[4.5 Bloque FROM 11](#_Toc98252156)

[4.6 Bloque WHERE 11](#_Toc98252157)

[4.7 Join 12](#_Toc98252158)

[4.8 Apply 12](#_Toc98252159)

[4.9 Union y union all 13](#_Toc98252160)

[4.10 Exists 13](#_Toc98252161)

[4.11 Instrucción INSERT INTO 14](#_Toc98252162)

[4.12 Instrucción UPDATE 15](#_Toc98252163)

[4.13 Invocación a otros procedimientos almacenados 16](#_Toc98252164)

[5. Procedimientos almacenados o stored procedures 17](#_Toc98252165)

[5.1 Base de datos 17](#_Toc98252166)

[5.2 Comentario de inicio de PA 17](#_Toc98252167)

[5.3 Parámetros 18](#_Toc98252168)

[5.4 Declaración de variables 18](#_Toc98252169)

[5.5 Validaciones 19](#_Toc98252170)

[5.6 Cuerpo del PA 19](#_Toc98252171)

[5.7 Retorno 19](#_Toc98252172)

[5.8 Gestión de errores 19](#_Toc98252173)

[6. Vistas 20](#_Toc98252174)

[6.1 Base de datos 21](#_Toc98252175)

[6.2 Comentario de inicio de vista 21](#_Toc98252176)

[6.3 Cuerpo de la vista 21](#_Toc98252177)

[7. Funciones 21](#_Toc98252178)

[7.1 Estructura de una función 22](#_Toc98252179)

[1.7.1. Base de datos 22](#_Toc98252180)

[2.7.1. Comentario de inicio de PA 22](#_Toc98252181)

[3.7.1. Parámetros 22](#_Toc98252182)

[4.7.1. Declaración de variables 22](#_Toc98252183)

[7.2 Tipos de funciones 23](#_Toc98252184)

[1.7.2. Función escalar 23](#_Toc98252185)

[2.7.2. Función de tabla 23](#_Toc98252186)

[3.7.2. Función de tabla tipada 23](#_Toc98252187)

Control Documental

|  |  |  |
| --- | --- | --- |
| Redactado por: |  |  |
| Revisado por: |  |  |
| Aprobado por: |  |  |

Histórico de modificaciones

|  |  |  |  |
| --- | --- | --- | --- |
| CÓDIGO | VERSIÓN | FECHA | COMENTARIO |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

# Introducción

En este documento especificaremos las reglas y configuración que debemos de respetar a la hora de trabajar con SQL.

Primero explicaremos las configuraciones necesarias del IDE y después pasaremos a las reglas de estilos de programación.

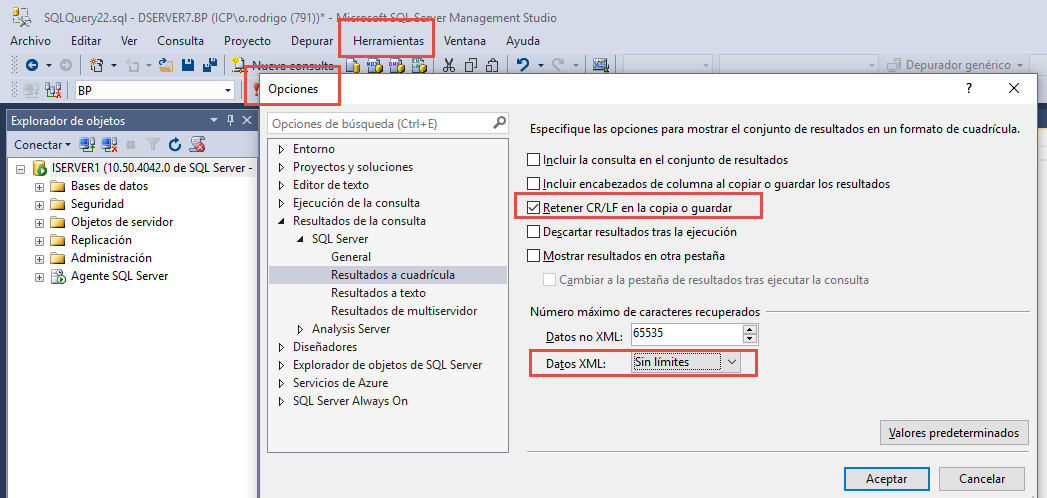
# Configuración de Management Studio

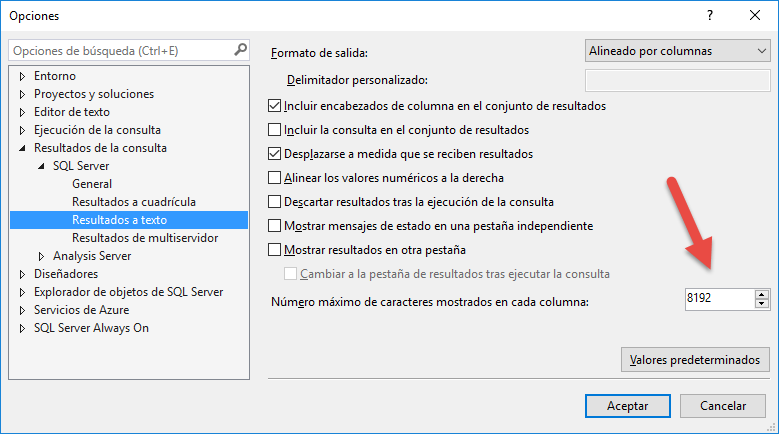
Para que todos podamos visualizar el código de una determinada forma, tenemos que configurar una serie de parámetros que, durante nuestro día a día, harán que el código esté ordenado y correctamente.

Desde el Management Studio, nos dirigiremos a la sección “Herramientas > Opciones”

## Saltos de línea y retorno de carro

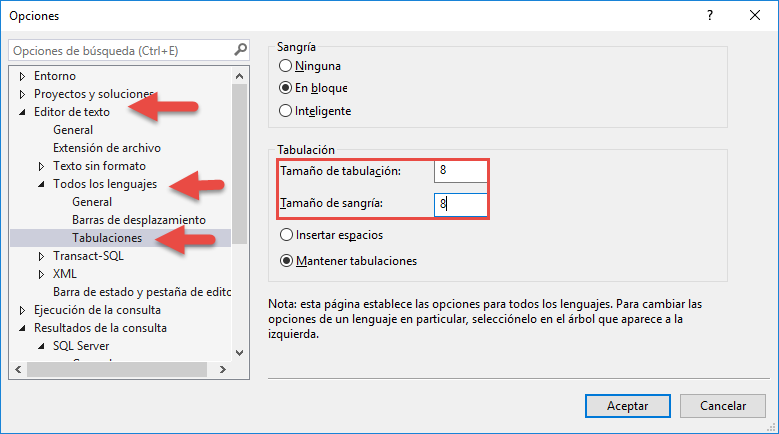
Primero configuraremos los saltos de línea y retornos de carro tal como se muestra en las imágenes.





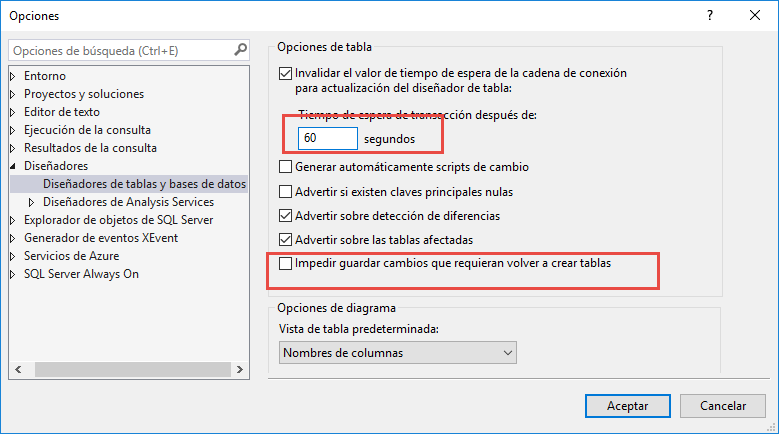
## Tabulaciones

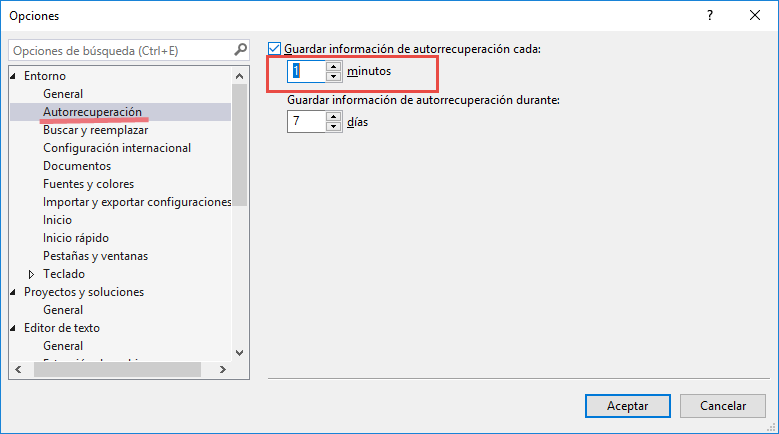
Estableceremos las tabulaciones con la configuración que se muestra a continuación:



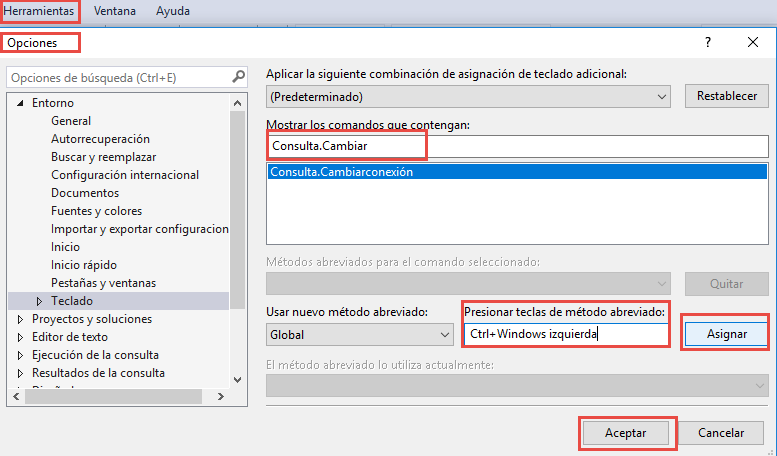
## Opciones para diseñadores de tablas y autorecuperación

Esta configuración nos facilitará el trabajo durante el mantenimiento de datos.



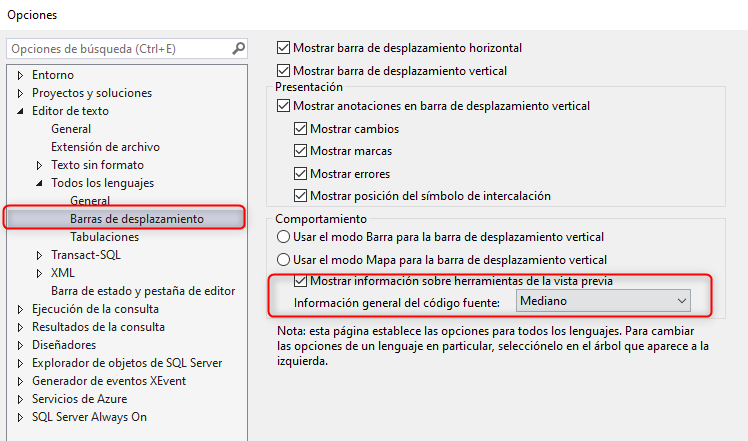


## Atajos para cambiar de conexión



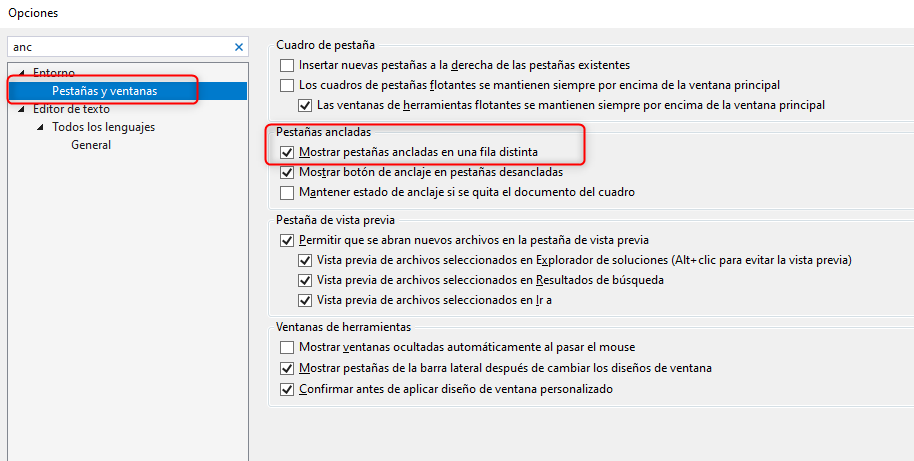
## Mapa de desplazamiento

Esto es espacialmente útil para poder situarte rápidamente en la porción de código que nos interese



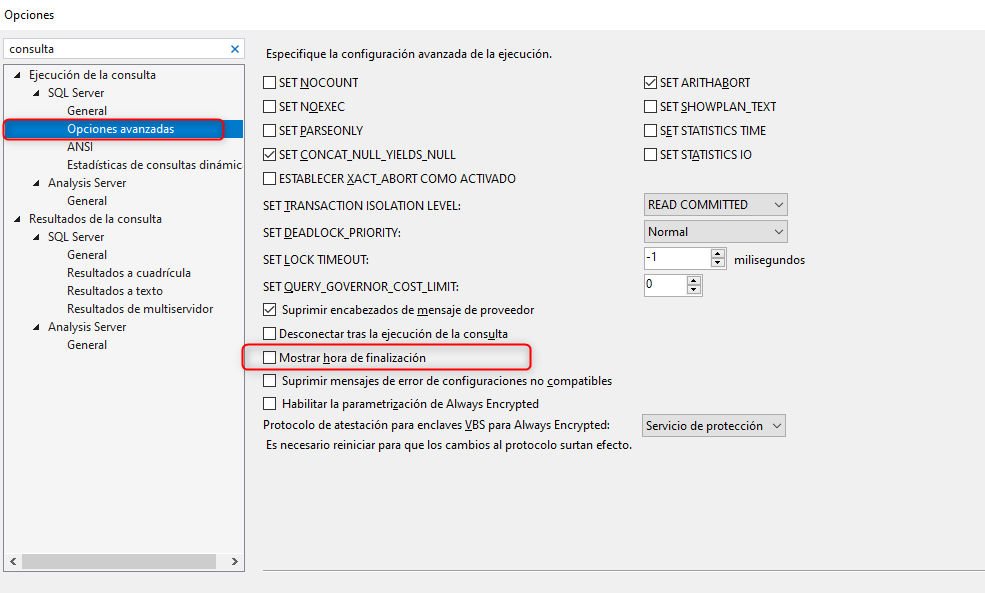
## Anclaje de pestañas

Esta opción nos permite dejar pestañas que anclamos en una línea diferente a las que no estan ancladas.



## Mensaje de hora de finalización

Es un mensaje que aparece cada vez que ejecutamos un script. Es realmente molesto y se puede quitar aquí:



# Reglas de nomenclatura

En este apartado definiremos las reglas de nomenclatura y nominación para los scripts de SQL.

1. Los scripts tienen que estar escritos en español a excepción de las palabras reservadas.
2. Todos los scripts tienen que ir escritos en mayúsculas. Salvo los mensajes de retorno.
3. Separaremos las nominaciones compuestas de varias palabras mediante el guion bajo “\_”.
4. Procuraremos utilizar palabras con significado.

# Reglas de sangrado y estructura de código

Una vez configurado correctamente nuestro Management Studio debemos de tener en cuanta las siguientes reglas a la hora de sangrar (tabular) nuestro código.

En os ejemplos veremos flechas naranjas que indicarán tabulación y puntos naranjas que indicarán espacios en blanco.

## Tabulaciones

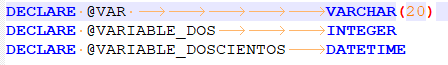
Utilizaremos las tabulaciones principalmente para alinear verticalmente las palabras que pertenezcan a la misma categoría o bloque dentro de un script de SQL.

Estas reglas se aplican a todo tipo de código SQL: consultas. procedimientos almacenados, funciones, disparadores, vistas, etc.

## Declaración de variables

Las variables las declararemos, en la medida de lo posible, en el comienzo de los procedimientos almacenados, funciones y triggers. Debemos de alinear la palabra DECLARE verticalmente y separaremos la declaración del nombre de la variable del DECLARE mediante un espacio. Después, los tipos tienen que estar alineados verticalmente también. Esto implica que tendremos sangrar los tipos de las variables que tengan un nombre corto para ajustarse a los nombres más largos.

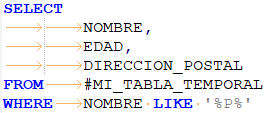
Ejemplo:



## Palabras reservadas en consultas

Intentaremos mantener las palabras reservadas alineadas verticalmente en una misma columna. Separaremos las palabras no reservadas de las reservadas mediante una tabulación.

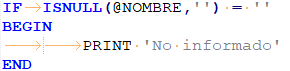
Ejemplo:



## Palabras reservadas en bloques de código

De un modo simular a las palabras reservadas en consultas, las palabras reservadas que se utilizan para indicar bloques de código.

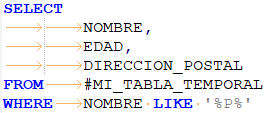
Ejemplo:



## Bloque SELECT

Después de la palabra SELECT escribiremos un salto de línea y tabularemos dos veces para separaros de la alineación vertical de las palabras reservadas. Todos los campos del SELECT deben de ir alineados verticalmente.

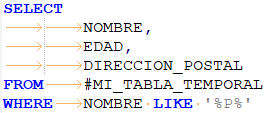
Ejemplo:



## Bloque FROM

Una vez escrito el bloque SELECT, alinearemos el FROM verticalmente con la palabra SELECT y tabularemos tantas veces como haga falta para alinear las los nombres de las tablas con los campos del SELECT.

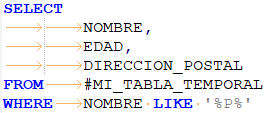
Ejemplo:



## Bloque WHERE

El bloque WHERE irá alineado verticalmente con las palabras reservadas SELECT y FROM y tabularemos los campos con sus condiciones para que estén alineados con las tablas del FROM.

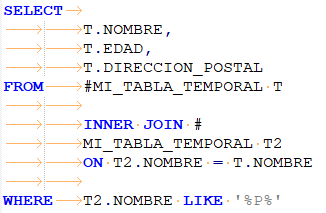
Ejemplo:



## Join

Las declaraciones de join deberán de estar situados en una línea independiente y los nombres de las tablas alineados verticalmente con la instrucción join en la siguiente línea después del retorno de carro. Escribiremos las condiciones en una línea nueva debajo de la instrucción join.

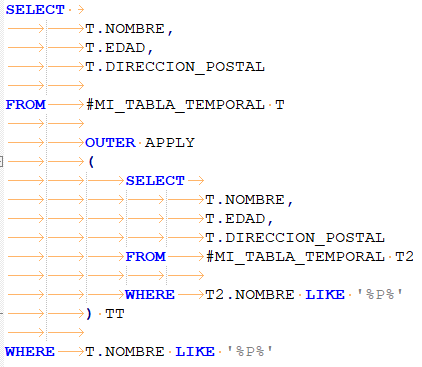
Ejemplo:



## Apply

En las declaraciones de apply alinearemos los paréntesis de apertura y cierre y respetaremos las reglas mencionadas anteriormente.

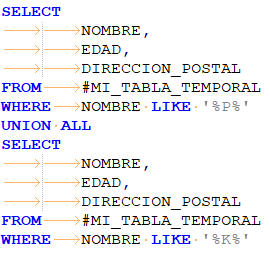
Ejemplo:



## Union y union all

Para las consultas en las que necesitemos unir varias sentencias SELECT, alinearemos las palabras reservadas UNION SELECT FROM y WHERE verticalmente de todas las consultas que conformen la unión de la consulta final. Después aplicaremos las reglas establecidas para los puntos anteriores.

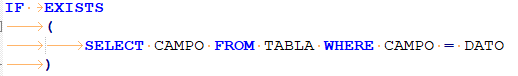
Ejemplo:



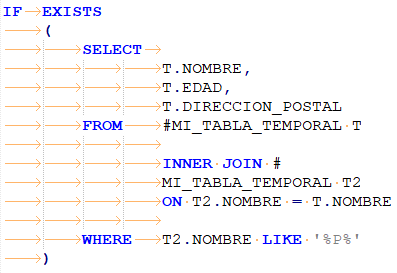
## Exists

Cuando utilicemos exists, debemos de alinear el paréntesis de apertura y cierre con la palabra reservada EXISTS

Ejemplo:



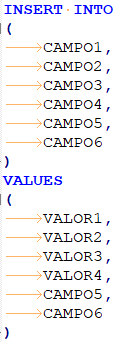
Ejemplo:



## Instrucción INSERT INTO

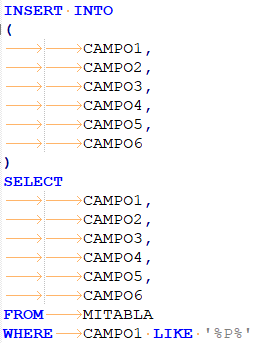
Para las instrucciones alinearemos verticalmente todos los campos y todos los valores sangrándolos una vez para separarlos del alineamiento de las aperturas y cierres de los paréntesis.

1. Ejemplo de un insert con valores declarados o estáticos:



1. Para inserciones que provengan de una sentencia SELECT, aplicaremos las normas establecidas para los puntos 3.2, 3.4 y 3.5. Alinearemos los campos de la declaración del insert con los campos de la proyección del SELECT.

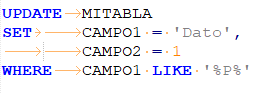
Ejemplo:



## Instrucción UPDATE

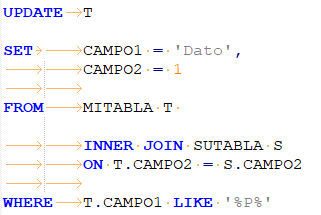
Para las instrucciones UPDATE, alinearemos las palabras reservadas y separaremos las no reservadas de las reservadas mediante quedando alineados los campos y nombres tablas. Respetaremos las normas establecidas anteriormente.

Ejemplo:



En el caso de que los valores de los campos a actualizar provengan de una SELECT alinearemos las palabras reservadas de las instrucciones verticalmente y separaremos los nombres de los campos y tablas mediante tabulaciones para que queden alineados verticalmente.

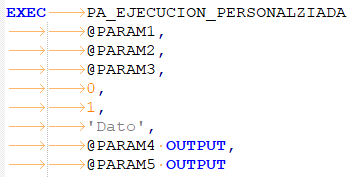
Ejemplo:



## Invocación a otros procedimientos almacenados

Cuando invoquemos otros PA separaremos la palabra EXEC del nombre del PA mediante una tabulación y alinearemos los parámetros enviados con el nombre del PA verticalmente.

Ejemplo:



# Procedimientos almacenados o stored procedures

En esta sección intentaremos detallar como tiene que estructurarse un Procedimiento Almacenado (PA).

Lo primero y más importante es que todos los procedimientos almacenados empezaran por las siglas PA\_.

Nos basaremos en la siguiente plantilla:



El PA se divide en las siguientes secciones empezando desde arriba: base de datos, comentario inicial, parámetros, declaración de variables, validaciones, cuerpo del PA, retorno y gestión de errores.

## Base de datos

Al comienzo de cualquier script de un PA debemos indicar la base de datos:

Por ejemplo:

USE INCIDENCIAS

GO

Esto nos conectara directamente a la base de datos que debe de ejecutarse el script antes de la acción que queramos hacer sobre el PA.

## Comentario de inicio de PA

Continuaremos con el comentario de inicio de PA, este indicará datos del autor del PA, fecha de creación, y una breve explicación de lo que hace. Dejaremos preparada una sección para posibles comentarios de futuras modificaciones.

Por ejemplo:

/\*

#######################################################################

NOMBRE DEL PROCEDIMIENTO: [NOMBRE PA]

FECHA DE CREACIÓN: [FECHA]

AUTOR: [AUTOR]

RUTA REPOSITORIO: [RUTA]

USADO POR: [WEB o APP]

FUNCIONAMIENTO: [EXPLICACIÓN BREVE]

#######################################################################

FECHA DE MODIFICACIÓN:

AUTOR:

EXPLICACIÓN:

#######################################################################

\*/

## Parámetros

Existen dos tipos de parámetros: parámetros de entrada y parámetros de entrada y salida.

En todo procedimiento debemos de mantener los siguientes parámetros que pasaremos a explicar:

ALTER PROCEDURE PA\_PLANTILLA

/\*

MAS PARAMETROS AQUI

\*/

@INVOKER INTEGER,

@COD\_IDIOMA VARCHAR(5),

@RETCODE INT OUTPUT,

@MENSAJE VARCHAR(8000) OUTPUT

AS

* INVOKER: este parámetro nos permitirá indicar un valor al podremos asociar alguna lógica en base al invocador del parámetro. Y, dependiendo de este, tomar un camino diferente durante el flujo de la ejecución del PA
* COD\_IDIOMA: este parámetro nos indicará la cultura o idioma en la que el PA deberá de retornarnos los mensajes hacia el invocador. Para saber que idiomas tenemos disponibles consultaremos la tabla IDIOMAS.
* RETCODE: es un parámetro de entrada y salida, lo usaremos principalmente para recibir un código, hacia el invocador. Retornaremos un 0 para cuando la ejecución del PA no tenga errores, retornaremos un > 0 para indicar que hemos tenido un error controlado y < 0 para indicar error no controlado (principalmente el catch).
* MENSAJE: al igual que RETCODE, es un parámetro de entrada y salida, a través del cual retornaremos al invocador el mensaje en el lenguaje que se haya enviado a través de COD\_IDIOMA.

## Declaración de variables

En esta sección debemos de declarar todas las variables que vayamos a utilizar durante todo el script. No es recomendable declarar variables en el cuerpo del PA debido a puede llevar a errores de contexto y también por una mayor legibilidad. Aprovecharemos para inicializar las variables en la misma sección.

Por ejemplo:

AS

DECLARE @NTRANS INT = @@TRANCOUNT

DECLARE @T\_JSON\_PARAMETROS NVARCHAR(MAX)

DECLARE @ID\_IDIOMA INT

SET @MENSAJE = ''

SET @RETCODE = 0

## Validaciones

En este fragmento debemos hacer todas las validaciones necesarias para intentar minimizar los posibles errores en las siguientes líneas de código. Podremos verificar:

* Posibles valores nulos en caso de que puedan llevar a error
* Posibles referencias a datos que no existen en nuestro sistema.
* Posibles desbordamientos durante castings

En el caso que el resultado de nuestra validación sea negativo, debemos de controlar el error y retornar el mensaje y el código de error.

Por ejemplo:

SELECT @ID\_IDIOMA = ID\_IDIOMA FROM IDIOMAS WHERE COD\_IDIOMA = @COD\_IDIOMA

IF ISNULL(@ID\_IDIOMA, 0) = 0

BEGIN

EXEC PA\_PARAMETROS\_MENSAJE\_FORMATEAR @COD\_IDIOMA, NULL, NULL, NULL, NULL, @T\_JSON\_PARAMETROS OUTPUT

;THROW 50013, 'Idioma no encontrado', 1

END

## Cuerpo del PA

En el cuerpo del PA escribiremos toda la ejecución de nuestro PA. Evidentemente, en esta sección podemos llamar a otros PA.

Tendremos que evaluar previamente, pero en la mayoría de los casos, el cuerpo del PA tiene que estar transaccionado. Esto quiere decir que debemos de abrir y cerrar una transacción.

Ejemplo de código para abrir transacción

IF @NTRANS = 0 BEGIN TRANSACTION [TX\_PLANTILLA]

Ejemplo de código para cerrar una transacción.

IF @NTRANS = 0 AND @@TRANCOUNT > 0 COMMIT TRANSACTION [TX\_PLANTILLA]

Es importante controlar que no exista una transacción abierta antes de abandonar el PA.

## Retorno

En la mayoría de los casos, el retorno será **RETURN @RETCODE,** pero este no será la forma con la que nos comunicaremos con el invocador. Para ello, @RETCODE, @MENSAJE y otros parámetros se declararán como OUTPUT, de tal forma que nos permita emitir estos valores hacia el invocador.

## Gestión de errores

Por lo general, la los PA deben de tener un bloque **TRY CATCH** que englobe desde la validación de los campos hasta la última línea del cuerpo del PA.

No obstante, podemos hacer uso de mas bloques TRY CATCH para control de errores dentro del cuerpo del PA.

Para lanzar las excepciones utilizaremos THROW y este recibirá 3 parámetros en este orden:

* **error\_number**: siempre será superior a 50000 debido a que los inferiores estan reservados para el sistema.
* **mensaje**: con un máximo de 2048 caracteres. Este, por lo general, lo tendremos que recuperar de con la función FE\_MENSAJE\_ICP y así recibirlo en el idioma adecuado.
* **state**: recibirá un 1 o un 255. El 1 indicará que el error **no** proviene de un PA llamado con anterioridad, es decir, que es un error que hemos controlado en nuestro PA. En cambio, pondremos 255 si el error lo controlamos tras haber invocado a otro PA.

En un error controlado el propio PA, el **error\_number** se guardará en **@RETCODE** en el bloque **CATCH** y será enviado al invocador de vuelta que controlará el error y volverá a lanzar **THROW** usando el **@RETCODE** como parámetro **error\_number** y el **@MENSAJE** como mensaje y así sucesivamente hasta llegar al invocador final.

Por ejemplo:

Para cuando no proviene de un error controlado en el PA que lo detecta

THROW 50013, 'El idioma indicado no existe', 1

Para cuando proviene de un error de haber invocado otro PA que nos ha devuelto un RETCODE superior a 50000

THROW @RETCODE, @MENSAJE, 255

En el bloque **CATCH**, controlaremos el **state** recibido para o bien buscar el mensaje con la función **FE\_MENSAJE\_ICP** si recibimos state = 1 o bien simplemente retornamos el mensaje recibido si recibimos state = 255.

Ejemplo de bloque CATCH

BEGIN CATCH

DECLARE @ID\_MENSAJE\_ICP INT = ERROR\_NUMBER()

IF @ID\_MENSAJE\_ICP < 50000 SET @ID\_MENSAJE\_ICP = @ID\_MENSAJE\_ICP + 10000000

IF ERROR\_STATE () <> 255

SET @MENSAJE = dbo.FE\_MENSAJE\_ICP(@ID\_MENSAJE\_ICP, @ID\_IDIOMA, @T\_JSON\_PARAMETROS)

IF @NTRANS = 0 AND @@TRANCOUNT > 0 ROLLBACK TRANSACTION [TX\_PLANTILLA]

SET @RETCODE = @ID\_MENSAJE\_ICP

END CATCH

# Vistas

En esta sección expondremos como se tienen que formar los scripts para las vistas de SQL.

Lo primero y mas importante es que todas las vistas tienen que empezar por V\_.

Nos basaremos en la siguiente plantilla:



La vista se divide en las siguientes secciones empezando desde arriba: base de datos, comentario inicial, cuerpo de la vista.

## Base de datos

Al comienzo de cualquier script de una vista debemos indicar la base de datos:

Por ejemplo:

USE INCIDENCIAS

GO

Esto nos conectara directamente a la base de datos que debe de ejecutarse el script antes de la acción que queramos hacer sobre la vista.

## Comentario de inicio de vista

Continuaremos con el comentario de inicio de vista, este indicará datos del autor de la vista, fecha de creación, y una breve explicación de lo que hace. Dejaremos preparada una sección para posibles comentarios de futuras modificaciones.

Por ejemplo:

/\*

#######################################################################

NOMBRE LA VISTA: [NOMBRE VISTA]

FECHA DE CREACIÓN: [FECHA]

AUTOR: [AUTOR]

RUTA REPOSITORIO: [RUTA]

USADO POR: [WEB o APP]

DESCRIPCION DE LA VISTA: [EXPLICACIÓN BREVE]

#######################################################################

FECHA DE MODIFICACIÓN:

AUTOR:

EXPLICACIÓN:

#######################################################################

\*/

## Cuerpo de la vista

Finalmente, escribiremos el script de la consulta respetando las reglas de nomenclatura, estructuración y tabulación.

# Funciones

En esta sección intentaremos detallar como tiene que estructurarse una función (FN).

Podemos clasificar las funciones en tres tipos: escalar, de tabla y de tabla tipada. Cada una se debe de utilizar en situaciones diferentes teniendo en cuenta el rendimiento.

Todas las funciones pueden tener dos prefijos diferentes:

* FE\_ para cuando se trate de una función escalar.
* FT\_ para cuando se trate de una función de tabla.

Una FN se divide en las siguientes secciones empezando desde arriba: base de datos, comentario inicial, parámetros, declaración de variables, cuerpo de la FN y retorno.

## Estructura de una función

### Base de datos

Al comienzo de cualquier script de un PA debemos indicar la base de datos:

Por ejemplo:

USE INCIDENCIAS

GO

Esto nos conectara directamente a la base de datos que debe de ejecutarse el script antes de la acción que queramos hacer sobre el PA.

### Comentario de inicio de PA

Continuaremos con el comentario de inicio de PA, este indicará datos del autor del PA, fecha de creación, y una breve explicación de lo que hace. Dejaremos preparada una sección para posibles comentarios de futuras modificaciones.

Por ejemplo:

/\*

#######################################################################

NOMBRE DEL FUNCIÓN: [NOMBRE FN]

FECHA DE CREACIÓN: [FECHA]

AUTOR: [AUTOR]

RUTA REPOSITORIO: [RUTA]

USADO POR: [WEB o APP]

FUNCIONAMIENTO: [EXPLICACIÓN BREVE]

#######################################################################

FECHA DE MODIFICACIÓN:

AUTOR:

EXPLICACIÓN:

#######################################################################

\*/

### Parámetros

En las funciones solo utilizaremos parámetros de entrada.

### Declaración de variables

En esta sección debemos de declarar todas las variables que vayamos a utilizar durante todo el script. No es recomendable declarar variables en el cuerpo de la FN debido a puede llevar a errores de contexto y también por una mayor legibilidad. Aprovecharemos para inicializar las variables en la misma sección.

## Tipos de funciones

### Función escalar

Utilizaremos la función escalar cuando requiramos un resultado sobre un sólo valor. Hay que intentar no utilizarlas si vamos a cruzarlas en otras consultas, ya sea en el bloque WHERE o en la proyección del SELECT, suelen ralentizar debido a que se lanzan una vez por cada registro que aún NO ha sido filtrado siquiera, lo que puede provocar grandes problemas de rendimiento.

Si nos vemos en la OBLIGACIÓN de utilizarla, recomiendo hacerlo dentro de un OUTER APPLY.



### Función de tabla

Actúa similar a una vista, pero puedes pasar parámetros para filtrar los datos. Es la que debemos de utilizar si vamos a cruzar el resultado en otras consultas, vistas, WHERE, etc. Suelen tener un mejor rendimiento general que cualquier otro tipo de función.



### Función de tabla tipada

Función de tabla recomendada sólo si NO la vamos a cruzar con otras consultas, ni a usarla en el WHERE. Sólo la utilizaremos si queremos obtener datos de algo en concreto una sola vez, que podemos volcar a una tabla para su posterior utilización, o devolver directamente este resultado.

