Lección 4: Guías de Código de SQL

Casi puedo escucharlo…. Noooooo, mas lineamientos de codificación! SI, mas lineamientos de código. Usted no solo escribe y organiza código fuente, Ud. también tiene que escribir y organizar cosas en la base de datos. Es por eso que empecemos directamente y sin perder mucho tiempo. Los beneficios de porque hacer esto están en la lección anterior.

**1) Tablas de la base de datos.**

* Normalizar los datos basados siempre en la tercera forma normal. [http://en.wikipedia.org/wiki/Third\_normal\_form](http://web.archive.org/web/20090516001811/http:/en.wikipedia.org/wiki/Third_normal_form)). No intente desnormalizar información hasta que pueda demostrar que solucionará un problema haciendo esto.
* Los nombres de las Tablas tienen que ser Pascal y en Ingles
* Los nombres de las tablas tienen que ser Palabras reales, de preferencia sustantivos. Por ejemplo Auto, Proyecto, Persona, Empleado, etc. No usar abreviaciones a menos que sea una que cualquier nuevo programador que ingrese al equipo pueda conocer.
* Las tablas que se crean para representar relaciones muchos a muchos entre dos tablas y que no almacenan otra información se deben llamar “NombreTabla1” + “NombreTabla2” + “Map” por Ejemplo ProjectCategoryMap
* No usar “\_” o espacios en los nombres de tablas
* No ponga prefijos a las tablas
* Los nombres de las tablas no pueden ser palabras reservadas del SQL. Si no está seguro, escriba la palabra en la ventana de consultas del administrador del SQL, si cambia de color no puede ser usada.
* Los nombres de las tablas tienen que ser singular. Si el nombre de la tabla en singular está reservado pero en plural no, busca otro nombre de cualquier manera. Por ejemplo User, lo puedes cambiar por AppUser o SystemUser. Si bien viola la regla de las abreviaciones, es bastante común como para aceptarlo.
* Todas las tablas tienen que tener un dueño. Explícitamente crear con el nombre dbo. Ejemplo Create table dbo.Test
* A menos que seas un DBA, no uses Triggers. Nunca, por ninguna razón. Mantén las cosas sencillas.

**2) Columnas**

* Los nombres de las columnas tiene que ser Pascal
* No uses “\_” o espacios en los nombres de columna.
* La primera columna de cualquier tabla tiene que ser la columna de identidad. Se debe llamar “NombreTabla” + Id. Por ejemplo si tienes una tabla Proyecto, la primer columna se llamará ProyectoId y no ProyectoID
* El siguiente conjunto de columnas son las columnas que referencian a otras tablas. Por Ejemplo Si un proyecto tiene un dueño. La estructura de la tabla Proyecto, deberá tener una columna ClienteId que referencie a la tabla Cliente.
* Si la tabla representa algo que tenga un nombre o etiqueta, la columna debera ser llamada “Nombre de Tabla” + Nombre. Por ejemplo si el proyecto tuviera un nombre, la columna se debería llamar ProyectoNombre.
* A menos que la base de datos valla a tener un montón de información (y realmente me refiero a una cantidad gigante de datos), limita la cantidad de tipos de datos que usas. Por ejemplo haz todos los valores decimales “decimal” en vez de Money, flota y todos los otros tipos de datos. Esto va a evitar a otros desarrolladores pensar que tipo de datos has usado.
* No uses los tipos “n” (nvarchar, nchar, etc) a menos que tengas una buena razón para hacerlo.
* No uses tipos texto o imagen. Usa varchar (Max) para textos largos. Las imágenes almacénalas en un filesystem y almacena en la base de datos la dirección relativa a la imagen. No almacenes la dirección absoluta al archivo en la base de datos, porque si el día de mañana tienes que migrar el fileSystem tendrás que correr un script que actualice la ubicación de todas las imágenes.
* Las Constraints deben ser usadas donde sean necesarias.

**3) Claves e Índices**

* Una o más columnas tienen que construir la clave primaria. Por defecto, debería ser la columna de identidad de la tabla. En aquellas tablas que almacenen relaciones muchos a muchos (Tabla1+Tabla2+”Map”) la combinación de los identificadores. Luego cuando se hagan pruebas de carga se puede revisar y ajustar las Claves primarias
* No crees índices durante el desarrollo. Luego cuando se hagan pruebas de carga se puede revisar y crear índices.
* Crea claves foráneas para cualquier columna que refiera a columnas en otras tablas
* Si la tabla tiene un clave foránea a otra tabla, el nombre de la columna tiene que ser igual que la clave primaria de la otra tabla. Ejemplo si la tabla Proyecto tiene una clave foránea a la tabla Cliente, la columna con la clave primaria de la tabla cliente y la columna de la clave foránea de la tabla de proyecto se tienen que llamar ClienteId

**4) Procedimientos Almacenados**

* Toda interacción entre la aplicación y la base de datos se tiene que hacer mediante procedimientos almacenados. Sin excepción. Nunca embeba código SQL en la aplicación.
* Los procedimientos almacenados siguen la siguiente convención “app” + Tabla + “\_” + “Verbo” + “Condiciones” siempre en Ingles. (Ya hablaremos del porque el tema de ingles más adelante en el libro). Donde
  + App: es un prefijo que usaremos. Esto se vuelve útil a medida que pasa el tiempo y más de una aplicación o equipo comparten la misma base de datos. Va a ser más fácil para aislar los procedimientos almacenados usando un solo prefijo.
  + Tabla: es el nombre de la tabla de la base que se está usando
  + Verbo: ayuda al desarrollador a entender que es lo que el procedimiento almacenado está haciendo. Debería ser algo como “Get”, “GetAll”, “Save”, “Delete”
  + Condiciones: pueden representar parámetros u otra información útil para que el desarrollador pueda lograr entender el procedimiento almacenado sin tener que abrirlo. Por Ejemplo ncProject\_GetForUser va a retornar registros de la tabla proyectos para un usuario específico. ncProyect\_Get puede esperar un ProyectId y retornar un registro que corresponda con el Id. Podrías adivinar entonces que hace ncClient\_Get o ncClient\_GetAll?
* Todos los parámetros de entrada tienen que machear un nombre de columna. Si tu estas escribiendo un procedimiento almacenado para retornar proyectos para un cliente, yo esperaría que se llame ncProject\_GetForClient y tener un parámetro @ClientId int. Si en cambio llamas al parámetro @Client int, estarás agregando un paso más a otro programador para darse cuenta como se llama el parámetro
* No usar “sp” o “sp\_” como prefijo para procedimientos almacenados
* La primer línea de todo procedimiento almacenado debe ser “SET NOCOUNT ON”
* En General no use parámetros de salida o valores de retorno. Para retornar un valor en el procedimiento haga Select @var as Name
* No capturar errores en los procedimientos almacenados a menos que se provea un camino alternativo que asegure no generar el error nuevamente. Los errores se deben manejar a nivel framework y exponer los errores de base de datos como cualquier otro error de aplicación. Debes enfocarte en prevenir errores testeando los datos de entrada para aquellas condiciones que producen errores, y no capturarlos luego de que ocurran.
* Los procedimientos almacenados deberían ser sencillos y pequeños en la mayoría de los casos. Enfócate en que resuelvan lo que su nombre indica solamente.
* No uses transacciones dentro de los procedimientos almacenados.
* Los procedimientos almacenados deberían devolver un solo set de resultados. Si crees que necesitas múltiples, probablemente necesitas un único set de resultados pero con un Select mas complejo o simplemente muchos procedimientos almacenados.

**5) Sintaxis SQL**

* No uses Select \*. Siempre explicita la lista de columnas.
* La lista de columnas se debe escribir de a una por línea.
* Todas las consultas que devuelvan más de un registro tienen que tener un ordenamiento por defecto. Si es necesario algún tipo de ordenamiento adicional, deberá ser manejado por la aplicación.
* En tu editor, configura que los tabs sean reemplazados por 4 espacios así el formateo será retenido al margen del editor que sea utilizado.
* Cuando uses comas para separar los nombres de columnas, pon la coma al comienzo de la línea y no al final. Esto hace más fácil para comentar líneas cuando estés desarrollando y probando cosas.
* Usa NOLOCK y ROWLOCK lo mas que puedas
* Maneja los parámetros opcionales aceptando el parámetro como nulo y prueba recibiendo nulo y un valor válido en la cláusula Where. La definición del parámetro debería ser @ProjectId=null y la clausula where debería ser (@ProjectId is null or ProjectId = @ProjectId). Es importante que @ProjectId is null sea escrito primero en la condición para que el motor no evalué la segunda entrando en la tabla si el valor no está siendo provisto.
* Asignar alias a las columnas SOLAMENTE si entra en conflicto con otras columnas en la misma salida. Una columna llamada ProjectName debería siempre ser ProjectName en todas las consultas de la aplicación. No uses un alias, como por ejemplo, Select ProjectName as Project from Project. Los alias hacen más difícil para otros programadores el rehusar los procedimientos almacenados sin tener que mirar en detalle que hacen. Esto también hace difícil popular resultados en una propiedad de una clase del dominio.

Las siguientes líneas ilustran un poco lo que vimos más arriba

Select

c.ClientId

, c.ClientName

, ...

From

Client c with (nolock)

Inner join AppState s with (nolock) on s.AppStateId = c.AppStateId

Where

(@AppStateId is null or c. AppStateId = @AppStateId)

Order by

s.StateName asc

, c.CityName

, c.ClientName

No hacer:

Select \*

From Client, AppState

Where AppState.AppStateId = Client.AppStateId

And AppState.AppStateId = IsNull(@AppStateId, AppState.AppStateId)

Ni tampoco hacer

Select ClientId as Id, ClientName as Name, ...

From Client, AppState on

Where AppState.AppStateId = Client.AppStateId

And AppState.AppStateId = IsNull (@AppStateId, AppState.AppStateId)

Problemas que ocasionan las dos consultas de arriba

1. El formateo es pobremente manejado y es difícil de leer.
2. El Select del primer ejemplo no especifica nombres de columna y en el segundo constituye un bloque grande y difícil de leer de forma rápida.
3. Ambos usan estilos de unión sin alias para las tablas
4. El segundo usa alias para los nombre de columnas
5. “Name” es una palabra reservada de SQL y no debería ser usada en Procedimientos almacenados.
6. Usar IsNull() para un parámetro opcional tiene peor performance comparado con (@var is null or column = @var)
7. No tiene un ordenamiento por defecto.

**Algunos otros Ejemplos/Reglas de sintaxis de SQL**

* Usa Joins en la parte del *from* en vez de usar subconsultas en el *Select* o en el *Where*. Evita también usar where column in (select…) o Where column not in (select…).
* Usa joins para consultas de eliminación de información en vez de declaraciones con IN()
  + Hacer

Delete p

From

Client c

join Project p on p.ClientId = c.ClientId

where

c.ExpirationDate < @today

* + En vez de

delete

Project

Where

ClientId in (select ClientId

From Client

Where ExpirationDate < @today)

* Siempre lista explícitamente los nombres de las columnas en las declaraciones insert
* No uses cursores. Si consideras realmente la necesidad de hacerlo consulta primero con el líder técnico o arquitecto para ver la posibilidad de otras alternativas. Si no hay alternativa posible, se podría usar cursores.
* Por defecto, no usar paginado del lado de la base de datos. Esto en general debiera manejarse en la capa de aplicación. La paginación en la base de datos debería dejarse solamente si alguna consulta devuelve grandes cantidades de registros (más de 50 mil) y la implementación debería consultarse con el arquitecto/líder técnico.
* Si una declaración va a requerir muchas tablas y muchos criterios de filtros, la primera tabla debería ser aquella cuyo filtro siempre sea provisto resultando en un conjunto más chico de registros para hacer join.
  + Hacer

Delete

Project

From

Client c

Join Project p on p.ClientId = c.ClientId

Where

c.ExpirationDate < @today

* + En vez de

Delete

Project

From

Project p

Join Client c on c.ClientId = p.ClientId

Where

c.ExpirationDate < @today

La primera declaración va a reducir el número de clientes que macheen con la fecha de expiración antes de intentar leer y filtrar la tabla proyecto.

Si usamos la segunda declaración, SQL tendrá que hacer Join con todas las filas de la tabla proyecto con la de cliente, resultando en un número mayor de resultados a ser filtrados. Por ende la performance se verá afectada.

No he probado esta solución desde hace un tiempo, pero creo que todavía es un acercamiento valido a la solución y creo que es una práctica de diseño a la que deberías adherir.

**Clave del capítulo:**

Usa convenciones para escribir cuestiones de base de datos.

1) Tablas de la base de datos.

2) Columnas

3) Claves e Índices

4) Procedimientos Almacenados

5) Sintaxis SQL

**Eso es todo.**