**EQUIPO DE PROYECTOS INFORMÁTICOS DE LA DGC**

**BUENAS PRÁCTICAS PARA LA PROGRAMACIÓN SQL**

**NOVIEMBRE 2021**

**Versión 1.0**

|  |  |  |  |
| --- | --- | --- | --- |
| ELABORADO POR: | REVISADO POR: | REVISADO POR: | APROBADO POR: |
| Equipo Proyectos DGC  Jonatan Silva Cachay | Rita Huambachano |  |  |
| Fecha: 10/12/2021 | Fecha: | Fecha: | Fecha: \_\_/\_\_/\_\_\_\_ |

HISTORIAL DE REVISIONES

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Versión** | **Autor** | **Descripción** | **Fecha de Elaboración** | **Revisado por** | **Fecha de Revisión** |
| 1.0 | Jonatan Silva | Creación del documento | 10/11/2021 |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

1. **OBJETIVO.**

Elaborar el documento de buenas prácticas para el desarrollo de scripts o procedimientos almacenados en SQL Server.

1. **ALCANCE.**

Este documento es aplicable al Equipo de Proyectos de la DGC al momento elaborar los scripts o procedimientos almacenados que serán enviados a ejecución al ambiente de Consulares, Test y Producción.

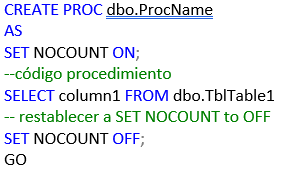
1. **PROBLEMATICA.**

Cada día se necesita procesar mayor cantidad de datos y obtener de manera más rápida y precisa la información. Muchos de los problemas de rendimiento se deben entre otras cosas al hardware, al software, al motor de base de datos y por sobre todo al diseño, índices y mala formulación de consultas SQL.

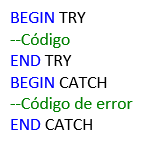
Para fines de este documento nos centraremos en estos últimos en donde siguiendo algunas recomendaciones veremos que se puede mejorar el tiempo de respuesta de nuestro motor de BD significativamente.

1. **CONSIDERACIONES.**
   1. **Diseño de Base de datos**

* Las tablas normalizadas permiten reducir al mínimo el espacio ocupado por nuestra base de datos y permiten asegurar la consistencia de la información al mismo tiempo que son muy rápidas para la realización de transacciones, pero generan un mayor tiempo de demora a la hora de consultarlas ya  que se deben realizar generalmente la unión de varias tablas, **por lo que en caso de necesidad de altas velocidades de respuesta con grandes volúmenes de datos un modelo desnormalizado es más que conveniente teniendo en cuenta todas las implicancias del caso.**
* Ajustar al máximo el tamaño de los campos ayuda a no desperdiciar espacio.
* Eliminar todo campo que no sea de utilidad ya que por más que no contenga datos genera retrasos.
  1. **Índices**
* Los índices son campos que permiten la búsqueda a partir de dicho campo a una velocidad notablemente superior. Sin embargo, cuentan con la desventaja que hacen más lenta la actualización, carga y eliminación de los registros ya que por cada modificación en la tabla se deberá modificar también el índice, además se debe tener en cuenta el hecho de que los índices también ocupan espacio en disco. Es por esto que no es factible indexar todos los campos de la base de datos y se hace necesario seleccionarlos cuidadosamente. Cabe destacar que por defecto las tablas no contienen índices por lo que la introducción de estos puede llegar a producir mejoras de más del 100% en algunos casos.
* Los campos que se recomiendan indexar son:
  + Claves Primarias
  + Claves Foráneas
  + Campos por los cuales se realizarán búsquedas
  + Campos por los cuales se va a ordenar
* Siempre conviene indexar tablas con gran cantidad de registros y que van a ser consultadas intensamente.
  1. **Consultas SQL**
* A la hora de ejecutar una consulta SQL la forma en que esta es expresada afecta directamente al motor de BD, pequeños cambios pueden significar la ganancia de muchos segundos o minutos que el usuario debe esperar al momento de ejecutar la consulta.
* Algunas **recomendaciones son**:
  + **No utilizar SELECT \*** por que el motor debe leer primero la estructura de la tabla antes de ejecutar la sentencia.
  + Seleccionar solo aquellos campos que se necesiten, cada campo extra genera tiempo extra.
  + Especificar el alias de la tabla delante de cada campo definido en el **select**, esto le ahorra tiempo al motor de tener que buscar a que tabla pertenece el campo especificado.
  + Evitar el uso de Cast. Y formulas dentro de las consultas, cada formula y casteo retrasan el motor considerablemente.
  + El orden de ubicación las tablas en el from deberían ir en lo preferible de menor a mayor según el número de registros, de esta manera reducimos la cantidad de revisiones de registros que realiza el motor al unir las tablas a medida que se agregan.
  + Incluir **SET NOCOUNT ON** Con cada sentencia SELECT y DML, el servidor SQL devuelve un mensaje que indica el número de filas afectadas por esa declaración. Esta información es sobre todo útil para depurar el código, pero es inútil después de eso. Mediante el establecimiento de **SET NOCOUNT ON**, se puede desactivar la función de devolución de esta información adicional. Para los procedimientos almacenados que contienen varias declaraciones o contener Transact-SQL lazos, establecer SET NOCOUNT en ON puede proporcionar un aumento de rendimiento significativo, porque el tráfico de red se reduce considerablemente.



* + Utilice **try-catch** para el manejo de errores:   
    La sintaxis es la siguiente:



* + Utilice **NOLOCK** para evitar el bloqueo de la tabla a consultar:
  + Si se crea una **tabla temporal** finalizando el procedimiento proceder a eliminar dicha tabla
* Para tener un buen rendimiento en los procedimientos almacenados o sentencias sql hay que tener en cuenta entre otras cosas, los puntos **enumerados a continuación:**
  + Tratar de evitar en lo posible el uso de sentencias llamadas non-sargable en los argumentos de 'where'.

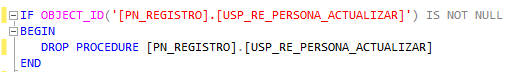
**Por ejemplo** "is null", "or" , "<>" , "!=", "!>" , "NOT EXIST", "NOT IN", " NOT LIKE" y "LIKE %500" que aunque no siempre, a menudo impiden que el optimizador use un índice para ejecutar la búsqueda.

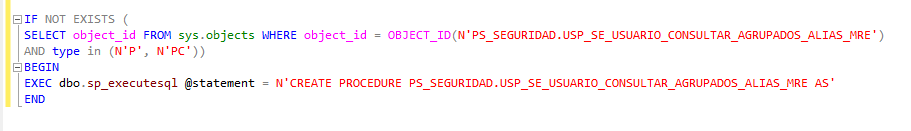
* + Si tienes un query que usa la sentencia 'NOT IN’, que ofrece un bajo desempeño debido a que el optimizador tiene que realizar un nested table scan, es mejor utilizar alguna de las siguientes alternativas:
    - * EXISTS ó NOT EXISTS
      * IN
      * Realizar un LEFT OUTER JOIN y checar con una condición nula.
  + Cuando uses LIKE en la sentencia where , trata de usar dos o más caracteres que antecedan al comodín (%), por ejemplo LIKE 'ma%' , así el optimizador tendrá la habilidad de usar un índice para ejecutar el query, en cambio si el primer carácter es % el optimizador no podrá tomar el índice y un table scan será ejecutado.
  + No utilizar funciones en las columnas de las tablas en el **where**, por ejemplo: Esta sentencia no tomará el índice:



- Aquí aplicamos la recomendación y el índice será utilizado:



* Para mantener el código del SQL lo más claro evitar código SQL comentado.
* Evitar el dropeo del procedimiento para que no se pierda los permisos en caso ya existiera.
  + EVITAR:
  + USAR:



* Por recomendación de OTI, se debe colocar los alias en el where de las consultas con JOINs.