Tabla de Contenido

Objetivo del documento

El presente documento ha sido confeccionado con el propósito de establecer los criterios base para codificar de forma correcta, la programación orientada a las bases de datos, dentro del entorno de sistemas de Banco Bolivariano.

Contenido general del documento

- Errores más comunes
- Recomendaciones Generales
- Consideraciones en la programación COBIS.
- Consideraciones en la replicación de bases de datos
- Consideraciones en el uso de tablas temporales.

Errores comunes

Errores que se encuentran generalmente en el código durante las revisiones.

- If exists (Select *
  - If exists (Select 1
- No usan if anidados, lo que obliga a evaluar cada condición, ejemplo:
  If a= 1 then
  If a= 2 then
  If a= 3 then
- Uso de parámetros locales en las condiciones del WHERE
  - Se debe enviar el parámetro desde el Front End ó justificar en la solicitud de aprobación QA
- Uso de SPs con opciones
  - Dividir los SPs ó justificar en la solicitud
- While (1=1)
  - Cambiar a otra condición para salir del WHILE, no se permite ciclos infinitos en el código.
- Table Scan
  - Crear índice adecuado ó justificar en la solicitud
- Control de creación/eliminación de objetos de base de datos
  - Validar existencia
- Carga masiva sin dump
  - Hacer dump tran cada 5000 registros
- Ejecución de SELECT con alta cantidad de registros en horarios de oficina
  - Ejecutar fuera de horarios de oficina
  - Controlar la cantidad de registros consultados
- Actualizar estadísticas o recompile a una tabla diferente
- No cerrar correctamente las transacciones al validar errores.
- No cumplir con el instructivo de llenar la solicitud de aprobación
  - Se debe asegurar que la solicitud cumpla con todo lo mencionado en el instructivo de llenar la solicitud de aprobación, tales como:
    - ✓ Especificar la cantidad de registros y tiempos de ejecución
    - ✓ Justificación de opciones y parámetros
    - ✓ Justificación de TABLE SCAN
    - ✓ Adjuntar los Query Plan,
    - ✓ Especificar los cambios, motivos, etc.
- Olvidar adjuntar la solicitud de aprobación o alguno de los archivos requeridos.

Recomendaciones generales

Revisión de código

- En los scripts que crean objetos nuevos, se debe validar la existencia del objeto antes de crearlo. Se debe validar la existencia de todos objeto de base de datos antes de crearlo ó borrarlo, sea nuevo ó existente:
  - Tablas
  - Vistas
  - Store procedure
  - Indices
  - Primary key

  En caso de SQL Server anteponer el esquema al nombre del objeto, Ejemplo: dbo.tabla

  Para el caso de sybase que se elimina y vuelve a crear los SP, deben incluir al final el script con los permisos para los usuarios aplicativos, solo en los casos que los permisos ya existan en producción, de lo contrario el trámite de los mismos es con Seguridad Informática

  Para evitar la pérdida de los permisos en SQL Server, usar ALTER PROCEDURE

- Al crear índice sobre tablas existentes ó nuevas en producción, se debe ejecutar al final de la creación los comandos

  - UPDATE STATISTICS
  - SP_RECOMPILE.

  Nota:
  No es necesario ejecutar estos comandos para tablas nuevas, pero se solicita para establecer un estándar.
  De este modo se evitarán confusiones sobre si las tablas existen ó no.
  QA-DBA decidirá si considera en la revisión dependiendo del caso

- En la creación de índices de preferencia se considere campos numéricos.

Uso de Parámetros locales en las condiciones del WHERE,

- Se permite si:
  - Son necesarios en los procesos Batch. Esto aplica a SPs nuevos o ya existentes. (Justificar)
  - Al modificar SPs que ya existen en Producción. (Justificar)
  - Cuando el SP se ejecuta entre una y diez veces al día, desarrollo lo debe especificar en la solicitud de aprobación.
  - Si el SP genera un plan óptimo de ejecución y un IO menor a 300. Esto aplica a SPs nuevos y existentes.
  - Si el parámetro local no se aplica a la primera columna del índice, aplica a SP nuevos y existentes.
  - Si el parámetro local no se aplica al índice, aplica a SP nuevos o existentes

Uso de SPs con opciones.

- (a) Es permitido si:
  - Las tablas que maneja el SP tienen:
    - ✓ Una cantidad de registros (Menor a 10000)
    - ✓ Óptimo plan de ejecución
    - ✓ IO menor a 300
    - ✓ No genere TABLE SCAN.
  - Cuando las opciones no tienen código directo sino que hacen llamados a otros SPs y el Lider no pudo llevar esa lógica a la capa aplicativa (Front End).

- Cuando es un SP existente y a una opción existente se le está dando mantenimiento.
- Cuando las opciones no manejan tablas, ni ejecutan SPs, por ejemplo asignación de variables, operaciones escalares o similares.

NOTA: las opciones que se deben justificar deben tener nombres como: @i_oper ó @i_opcion,

- (b) No es permitido cuando:
  - Las tablas que maneja el SP tienen alta cantidad de registros (Mayor a 10000)
  - El SP tiene mal plan de ejecución
  - El SP Provoca IO Mayor a 300
  - El SP Provoca TABLE SCAN
  - No es permitido si a un SP existente se le agregan más opciones agravando más la situación inicial del SP, y no debe crecer en más opciones.

- Si el select tiene varias tablas se debe realizar el enlace de las mismas con inner join, la clausula where se debe utilizar solo para filtros

- No está permitido el uso del *= o =*, en su defecto se debe usar el left join o el right join

- Si se va a eliminar todos los registros de una tabla se debe usar la senctencia TRUNCATE TABLE

- Después de cada transacción se debe validar el @@ERRORLOG, o dependiendo de la lógica el @@ROWCOUNT

- Evitar el deadlock

  1. Las transacciones sean las más cortas posibles
  2. Que el orden de programación de acceso a las tablas sea el mismo, ejemplo. Si obtengo la secuencia de una tabla para crear un nuevo registro, la siguiente sentencia debería ser el insert

- Para evitar evaluaciones innecesarias, usar if anidados, ejemplo:

```
If a = 1 then
  Exec sp_1
Elseif a = 2 then
        Exec sp_2
Elseif a = 3 then
        Exec sp_3
```

- Evitar las conversiones de datos en las clausulas where, mucho más si el campo forma parte del índice, la variable con la cual se filtra debe ser del mismo tipo de dato del campo de la tabla. Caso contrario no se estaría usando el índice, ejemplo:

Código erróneo:

```
Declare  @campo char(10)
Select ….
Where convert(int, campo_indice) = @campo
```

Código correcto:

```
Declare  @campo int
Select ….
Where campo_indice = @campo
```

- Tener en cuenta que al tener un índices con los campos A, B y C (en ese orden), el select toma el índice solo en las siguientes combinaciones:

  Where a = dato
  Where a = dato and b = data2
  Where a = dato and b = data2 and c = data3

  Si se utiliza el siguiente filtro el índice no es usado:

  Where b = data2
  Where c = data2
  Where b = dato and c = data2

- Intentemos evitar en la clausula where lo siguiente:

Evitar uso	**Use:**
Value like 'abc'	value = 'abc'
Substring (value, 1, 3) = 'abc'	value like 'abc%'
Amount !=0	amount > 0
amount + 3000 < 5000	amount < 2000
Last_name + ", " + first_name = 'some, name"	last_name = 'some' and first_name = 'name'
<, >	<=, >=
Value / 100 = 100	value = 100 * 100
Or	UNION ALL or UNION

- Utilizara el bloque de transaccionalidad no es permitido si se está usando una sola sentencia

- Cerrar los cursores antes de la sentencia return

- Evitemos el uso del curso, usemos mejor un cliclo (while), esta técnica es conocida como cursores artificiales.

- Evitemos el uso del IN y usemos el EXISTS

- Evitemos el uso de NOT IN y usemos el NOT EXISTS

- Evitemos el uso de select * y pongamos la lista de campos select A, B, C

Programación en COBIS

Recomendaciones

- Verificar que el Máximo nivel de anidamiento y recursividad de store procedures debe ser 16.
- Recordar que el máximo número de parámetros para un store procedure es 255.
- Longitud máxima para nombre de parámetros: 30 caracteres.
- Aplicar el estándar de nombrado.
- Máxima cantidad de sesiones por cliente es 16.
- Incluir variable @w_return para registrar el estatus de retorno.
- No combinar CREATE PROC con otras sentencias SQL en un mismo batch.
- No mezclar la creación de un procedimiento con: CREATE VIEW, CREATE DEFAULT, CREATE RULE, CREATE TRIGGER.
- Procedimientos invocados por un tercero no debe referenciar tablas temporales del procedimiento que lo invoco.
- Gestionar permisos de uso y ejecución de procedures de otras bases que vayan a ser invocados.
- Validar que el código de transacción recibido como parámetro sea el que corresponde, antes de programar la lógica transaccional.

- Verificación de claves foráneas recibidas como parámetros - validar la existencia del valor recibido en la tabla a la que pertenece.
- Incluir la Subsección ERRORES en los siguientes escenarios:
  - Después de cada invocación a store procedure
  - Después de operaciones DML (Inserción, actualización o eliminación)
  - Después de una integridad referencial

Que no debo hacer

- Usar expresiones matemáticas en SARGS:
  - SELECT * FROM cl_tabla WHERE **ta_codigo +10** = 520.
- Usar tipos de datos incompatibles entre columnas, SARGS o parámetros en store procedures. Tipos de datos incompatibles: Float – Int, Char – Varchar, Binary – Varbinary.
- Uso de condiciones OR, especialmente en columnas diferentes de las mismas tablas.
- Usar demasiados subqueries.
- No usar la llave completa del índice en el WHERE, o la expresión ¡=.
- No usar variables locales en el WHERE siempre usar parámetros del SP.
- Evite usar barrido de tablas (TABLE-SCAN).

Prevención Bloqueos

- Revisar el código asegurar que las transacciones de UPDATE y DELETE no bloqueen la tabla.
- Utilizar Transacciones simples y cortas.
- Uso de cursores para afectaciones masivas (versión 10).
- No incluir iteración con usuario en medio de una transacción.
- Para transacciones encadenadas, validar que los programas invocados en una transacción tengan manejo de bloqueo y liberación.
- Uso moderado de HOLDLOCK o ISOLATION LEVEL 3 ya que retienen los bloqueos compartidos hasta que se de el commit o rollback.
  - **Nivel 3** - SQL Server **garantiza que los datos leídos por una transacción sean válidos hasta el final de dicha transacción**. SQL Server da soporte a este nivel mediante la palabra clave **holdlock** de la instrucción **select** que aplica un bloqueo de lectura en los datos especificados.
- Declaración de cursores (read only / update) para implicaciones de concurrencia, selección de índice o prevención de deadlock.

Batch

- Evitar overhead minimizando la interacción Cliente / Servidor.
- Dividir un proceso en varios que se ejecuten concurrentemente.
- Usar cursores de SQL en triggers y store procedures.
- En caso de depuraciones masivas realizarlo por bloques de 5000, ejemplo:

```
Set @w_total = 0
set rowcount 5000
while @w_cantidad_registros=5000
begin
  Delete from  er_cliente Where cl_fecha_corte = @i_fechacorte
  Select @w_cantidad_registros = @@rowcount, @error = @@error,
@w_total = @w_total + @@rowcount
  if @error != 0
    begin
```

```
            set rowcount 0
            return 1
        end
      dump transaction <bd> with truncate_only
    end
    set rowcount 0
```

Secuenciales

- Llevar una tabla de llaves primarias.
- Manejar su actualización separada de la transacción donde se generó el secuencial.
- Para evitar contención al obtener la secuencia en las tablas de parámetros de cobis usar la siguiente lógica:

```
update tabla_parametro
        set @secuencial = secuencial + 1,
            secuencial = secuencial + 1
where tabla = 'tabla'
```

Cobis

- Reducir las escrituras en el log, ya que la tabla syslogs no tiene índice y puede ocasionar contención y y problemas en la asignación de espacio
- Evitar el uso de select anidado para cargar tablas temporales. Es preferible el uso de SELECT INTO ya que la transacción no se graba completamente en el log.
- Preferir el uso de RPC en ambiente cliente – servidor; y usar el paso de parámetros por posición y no por nombre.
- Preferir el diseño e implementación de base de datos normalizada, excepto para aplicaciones de apoyo a la toma de decisiones.

Índices

- Definir índice clustered para agilizar procesos de insert y update sin generar contención. De preferencia columnas accesadas por un rango y que no sea la clave primaria.
- Definir como índice NO CLUSTERED sobre columnas utilizadas para accesar a menos del 20% de datos. Tratar de cubrir los queries con este tipo de índices.

- Tamaño de la llave debe ser el menor posible. De preferencia numéricos.
- Uso de FILLFACTOR para minimizar el particionamiento de paginas, para reducir el efecto de bloqueos.

sybase

- Preferir el uso de UPDATE-IN-PLACE para evitar contención de bloqueos, probabilidad de deadlock, tiempo de respuesta y consumo de recursos.
- Preferir el uso de segmentos de SYBASE y varios dispositivos lógicos y físicos para balancear la carga transaccional y mejorar el rendimiento.
- Uso de CustomControl, entre ellos el TXTINPUT.VBX (el no uso de esta función puede provocar comportamientos anómalos del mapeador).
- Validar tipos de datos definidos en store procedures a nivel de Front-End y a nivel de paso de parámetros.

- Verificar consistencia entre número de resultados que envía el store procedure y como llegan, con la recepción de variables y vaciado del buffer de resultados para evitar errores de recepción y pérdida de información.

Sybase – In-place update

Adaptive Server realiza las actualizaciones In-place siempre que sea posible.

Cuando Adaptive Server realiza una actualización In-place, las filas siguientes de la página no se mueven; los identificadores de fila siguen siendo los mismos y los punteros en la row offset table no se cambian.
Para obtener una actualización In-place, deberán cumplirse los siguientes requisitos:
La fila a ser modificada no puede cambiar su longitud.

La columna a actualizar no puede ser la clave, o parte de la clave, de un índice agrupado en una tabla con un bloqueo Allpages. Debido a las filas de un índice agrupado en una tabla Allpages se almacenan en orden de las claves, un cambio en la clave casi siempre significa que la ubicación de la fila se cambia.

Uno o más índices deben ser únicos o debe permitir duplicados.

La instrucción de actualización cumple las condiciones enumeradas en "Restrictions on Update modes through joins".

Las columnas afectadas no se utilizan para la integridad referencial.

No puede haber un trigger en la columna.

La tabla no debe ser replicada (a través de Replication Server).

Una actualización In-place, es el más rápido tipo de actualización, ya que hace que un solo cambio en la página de datos. esto cambia todas las entradas del índice afectados por la eliminación de la fila del índice anterior e inserta la nueva fila del índice.
La actualización In-place afecta solo los índice cuya clave ha cambiado por el update, En lugar de cambios afectan sólo índices cuyas claves se cambian por la actualización, la página y fila no se cambian.

Consideraciones por la replicación de BD

- No objetos (SPs, tablas, índices, etc.), ni temporales, ni fijos dentro de las bases tempdb, master, sybsystemprocs o sybsystemdb.
  - No se pueden crear objetos en la base de sistemas, por espacio (data y log).
  - No crear objetos fijos en la tempdb
- No eliminar objetos sin validar su preexistencia.
  - Desarrollo debe agregar en los scripts esta validación.
  - Durante la replicación si se elimina un objeto que no existe se genera una excepción y la replicación se detiene.
- No se debe utilizar uso excesivo de tablas temporales en los procesos batch.
  - Cuando la frecuencia de insert y delete es tan alta, se generan colas en el replicador de varios gigabytes, solamente con información temporal que se inserta para procesar una transacción y se borra al final de la misma. (se tratará mas adelante)
- No se debe actualizar claves únicas
  - En el origen no genera problemas pero en los ambientes replicados el update podría resultar en registros duplicados o errores en la base de datos réplicas.
- Crear lo objetos del orden "objetos hijos hacia objetos padres“
  - El SP se compila en la base inicial, pero no en la de réplica.

- El administrador de replicación genera una excepción de la transacción de creación del objeto creado fuera de un orden adecuado.
- Adicionalmente puede generar excepciones por querer insertar en un objeto que no existe.

- Ejecutar los programas SQR con la opción –XP
  - Sin usar esa opción se crean procedures en las base de datos que en su mayoría no se replican.
  - Esto genera una excepción al tratar de ejecutar un sp que no existe.
- BCP rápido
  - Se debe cargar la tabla con los índices creados. para que las operaciones de inserción se registren en el LOG y se puedan replicar.
  - Si se necesita borrar los índices para cargar la data con mayor rapidez, y esta información se requiere en los ambientes replicados el bcp también debe realizarse en los servidores remotos.
- DDL en SP's
  - Las sentencias DDL no replican cuando se encuentran dentro de stored procedures.
  - Se debe ejecutar las sentencias DDL en scripts directamente sobre la base
- Llenado de los Log
  - El proceso de replicación esta basado en lo que se encuentra en los LOGS de la base de datos. Por lo que los procesos generados por las aplicaciones no deben borrar el LOG a menos que las transacciones estén confirmadas (commit o rollback)
  - Se debe tomar precaución al momento de desarrollar SPs de carga ó actualización masiva de usar el comando:

    "dump tran with truncate_only"

    Cada cierto número de registros, por ejemplo procesando de 5000 en 5000 registros.
- Llenado de los Log
  - El comando anterior permite vaciar el log de las transacciones que ya han sido procesadas y por ende replicadas.

    No se debe usar el comando dump con la opción with no log porque esto generaría que todas las operaciones del log se eliminen y que la sincronización de la base de datos se pierda.

Uso de tablas temporales

- Evitar la creación de tablas temporales.muy grandes en la tempdb.
  - Genera contención a los demás procesos que utilizan la base de datos temporal, siempre definir conjuntos de resultados pequeños para almacenar en las tablas temporales.

- Minimizar la creación de tablas temporales en la base de usuario
  - Las bases de usuarios que se replican, también generan que las operaciones a las tablas temporales sean replicadas. Por lo que el uso de operaciones muy grandes a este tipo de tablas, genera un fujo de operaciones replicadas muy alto, esto genera que las demás operaciones se encolen hasta terminar de procesarse.
  - Se debe evitar creando las tablas temporales en una base que no sea replicada y realizar las operaciones sobre estas tablas.

- Si en un procedimiento almacenado definimos una tabla temporal en la cual se cargan datos y creamos índices, es necesario llevar la lógica del procesamiento a otro procedimiento para que el optimizador de acuerdo a la distribución de datos seleccione el correcto plan de ejecución.

Consideraciones para MySQL

Considerar los siguientes puntos

a) Verificar que siempre tenga asignado en la creación de procedures y functions la clausula DEFINER de la siguiente forma
CREATE **DEFINER=`root`@`localhost`** PROCEDURE `bb_SPMainMenuPublish`(
Cuando se compila con otro user el portal no funciona porque no tiene permisos para ejecutar el SP

b) Verificar que los procedures y funciones tengan al final $$; porque si no toma la siguiente instrucción como parte del SP.

c) Los nombres de objetos y usuarios deben estar asignadas con esta comilla ` ; si se coloca la comilla simple ‘ no reconoce el objeto y el portal no trabaja correctamente.

d) Texto siempre debe ir entre comillas simples **‘** no entre comillas dobles **”**

```
/*!50003 SET @TEMP_SQL_MODE=@@SQL_MODE, SQL_MODE='' */ $$
CREATE DEFINER=`root`@`localhost` PROCEDURE `bb_SPMainMenuPublish`(
    IN param_publish tinyint(1),
    IN param_ids TEXT
)
    COMMENT 'Publica un Main menu'
BEGIN
    DECLARE comma int DEFAULT 0;
    DECLARE mylist text DEFAULT param_ids;
    DECLARE temp text DEFAULT '';
    DECLARE strlen int DEFAULT LENGTH(param_ids);

    CREATE TEMPORARY TABLE temp_table (num int) TYPE=INNODB;
    SET comma = LOCATE(',', mylist);
    WHILE strlen > 0 DO
        IF comma = 0 THEN
            SET temp = TRIM(mylist);
            SET mylist = '';
            SET strlen = 0;
        ELSE
            SET temp = TRIM(SUBSTRING(mylist, 1, comma - 1));
            SET mylist = TRIM(SUBSTRING(mylist FROM comma + 1));
            SET strlen = LENGTH(mylist);
        END IF;

        IF CAST(temp as UNSIGNED) != 0 THEN
            INSERT INTO temp_table
            VALUES(CAST(temp as UNSIGNED));
        END IF;

        SET comma = LOCATE(',',mylist);
    END WHILE;

    UPDATE  bb_main_menu
    SET     bb_main_menu.is_active = param_publish
    WHERE   bb_main_menu.id IN (SELECT * FROM temp_table);

    DROP TEMPORARY TABLE IF EXISTS temp_table;

END $$
/*!50003 SET SESSION SQL_MODE=@TEMP_SQL_MODE */  $$
DELIMITER ;
```

CONTROL DE MODIFICACIONES DEL DOCUMENTO

Versión	Descripción del cambio	Fecha del cambio	Elaborado por	Revisado
1.0	Elaboración	23/jun/2010	Miguel Salazar	Miguel Salazar
1.1	Revisión y cambio de formato	24/ene/2011	Silvia Castillo	Miguel Salazar
1.2	Revisión y cambio de formato	13/may/2013	Marjorie Espinoza	Marjorie Espinoza y Marcelo Duran