Estándares de Codificación

Vicepresidencia de Operaciones y Tecnología

2015

Estándares de Codificación

CONTROL DE CAMBIOS DEL DOCUMENTO

FECHA	VERSIÓN	MODIFICACIÓN	AUTOR(ES)
16/03/15	1.0	Creación Documento.	Marly Johana Piedrahita Giraldo Jorge Ivan Alvarez Garcia Andres Extuar Villamizar Diaz Jaime Felipe Leal Pardo

Tabla de Contenido

1. INTRODUCCIÓN

Un estándar de codificación completo comprende todos los aspectos de la generación de código. Si bien los programadores deben implementar un estándar de forma prudente, éste debe tender siempre a lo práctico. Un código fuente completo debe reflejar un estilo armonioso, como si un único programador hubiera escrito todo el código de una sola vez.

Al comenzar un proyecto de software, se debe establecer un estándar de codificación para asegurarse de que todos los programadores del proyecto trabajen de forma coordinada.

La **legibilidad** del código fuente repercute directamente en lo bien que un programador comprende un sistema de software. El **mantenimiento** del código es la facilidad con que el sistema de software puede modificarse para añadirle nuevas características, modificar las ya existentes, depurar errores, o mejorar el rendimiento.

Aunque la legibilidad y el mantenimiento son el resultado de muchos factores, una faceta del desarrollo de software en la que todos los programadores influyen especialmente es en la técnica de codificación. El mejor método para asegurarse de que un equipo de programadores mantenga un código de calidad es establecer un estándar de codificación sobre el que se efectuarán luego revisiones de rutina.

Usar técnicas de codificación sólidas y realizar buenas prácticas de programación con vistas a generar un código de alta calidad es de gran importancia para la calidad del software y para obtener un buen rendimiento. Además, si se aplica de forma continua un estándar de codificación bien definido, se utilizan técnicas de programación apropiadas, y, posteriormente, se efectúan revisiones de rutinas, caben muchas posibilidades de que un proyecto de software se convierta en un sistema de software fácil de comprender y de mantener.

Aunque el propósito principal para llevar a cabo revisiones del código a lo largo de todo el desarrollo

es localizar defectos en el mismo, las revisiones también pueden afianzar los estándares de codificación de manera uniforme. La adopción de un estándar de codificación sólo es viable si se sigue desde el principio hasta el final del proyecto de software. No es práctico, ni prudente, imponer un estándar de codificación una vez iniciado el trabajo.

2. OBJETIVO

Brindar los lineamientos necesarios para el estándar de codificación de las aplicaciones de Colpensiones.

3. ALCANCE

Se plantean los lineamientos como se deben crear los objetos en el motor de base de datos y de la programación de Transact-SQL de SYBASE y el nombramiento de objetos y variables de los programas desarrollados en JAVA. Para las otras bases de datos y lenguajes de programaciones se deberá tomar también como referencia estos estándares en los casos que aplique.

4. CONTROL

Todos los proveedores, terceros y personal de Colpensiones que realicen actividades de programación deberán seguir los lineamientos de este documento de estándares.

5. NOMENCLATURA

En general, se sugiere utilizar siempre nombres que sugieran lo que contiene o realiza el objeto. El nombre de las variables, atributos, métodos, clases y paquetes debe seguir el estándar Camel, el cual consiste en dejar unidas las palabras de los nombres, pero usar una letra mayúscula al inicio de cada palabra, por ejemplo: nombreDeUnaVariable, parametroCantidadDias, etc.

Existen dos casos para esta notación:

- UpperCamelCase, Es cuando la primera letra de cada una de las palabras esta en mayúscula. Ejemplo: **EjemploDeComoEs**
- lowerCamelCase, igual que la anterior con la excepción de que la primera letra es minúscula. Ejemplo: **ejemploDeComoEs**

5.1 Base De Datos

5.1.1 Bases de datos

- El nombre de la base de datos debe reflejar el contenido de la misma
- No debe tener espacios o prefijos innecesarios.

5.1.2 Tablas y Vistas

- Usar notación UpperCamelCase para el nombrado de tablas y vistas
- No usar prefijos para el nombrado de estos objetos

- No usar espacios
- No abreviar el nombre de las tablas o columnas
- Para las vistas, se debe utilizar el prefijo **vw_ + Nombre en Notación Camel**

  Ejemplo: **vw_CuentaAfiliadoPension**

5.1.3 Columnas

- Usar notación UpperCamelCase para el nombrado de las columnas
- No usar palabras reservadas para el nombrado de las columnas.
- No abreviar el nombre de las columnas

5.1.4 Llaves e Índices

- Nombrar los índices indicando la tabla o tablas a las cuales pertenece o campos
- Usar el prefijo que se acople al caso:

  Llaves primarias **pk_**

  Llaves foráneas **fk_TablaPadre_TablaHija**

  Índice: **idx_NombreTabla_Campo1xCampo2xCamponN**

  Ejemplo: Llave Primaria: **pk_Afiliado**

  Llave Foránea: **fk_Afiliado_AfiliadoSeguro**

  Índice: **idx_Afiliado_TipoDocumentoxNumeroDocumento**

5.1.5 Procedimientos Almacenados

- Usar el prefijo **pr_** para el nombrado de los procedimientos
- Utilizar para el nombre UpperCamelCase
- No abreviar los nombres de los procedimientos almacenados
- El nombre del procedimiento almacenado debe mostrar la funcionalidad el mismo.

5.1.6 Funciones

- Usar el prefijo **fn_** para el nombrado de las funciones
- Utilizar para el nombre UpperCamelCase
- No abreviar los nombres de las funciones
- El nombre del procedimiento almacenado debe mostrar la funcionalidad el mismo.

5.1.7 Triggers

- Utilizar para el nombre UpperCamelCase
- El nombre de los triggers deben identificar el trabajo que hacen
- Utilice el prefijo **tr_** para identificar que un Triggers

5.2 Aplicaciones en Software Orientado a Objetos

5.2.1 Packages

- El prefijo del nombre de un package se debe escribir siempre con letras ASCII en minúsculas, y debe ser uno de los nombres de dominio de alto nivel, actualmente com, edu, gov, mil, net, org, o uno de los códigos ingleses de dos letras que identifican cada país como se especifica en el ISO Standard 3166, 1981.
- Los subsecuentes componentes del nombre del paquete indicaran a que componente del negocio y su respectiva funcionalidad

Ejemplo:

co.gov.colpensiones.comun.mail

co.gov.colpensiones.interaccionasofondos.generadorarchivos

co.gov.colpensiones.fab0262.comun.consultasnativas

5.2.2 Clases

- Se utilizará el nombrado UpperCamelCase
- Los nombres de las clases deben ser simples y descriptivos.
- Usar palabras completas, evitar acrónimos y abreviaturas (a no ser que la abreviatura sea mucho más conocida que el nombre completo, como URL, FTP HTML).
- Los nombres de clases tipo bean deberán tener el sufijo “Bean”, como por ejemplo **UsuarioRolBean**
- Los nombres de clases tipo DAO deberán tener el sufijo “DAO”, como por ejemplo **ExamenDAO**
- Los nombres de Servlets deben incluir la funcionalidad que controlan y con el sufijo Servlet, por ejemplo **RecepcionExpedienteServlet**

Ejemplo: **public class Afiliado{**
……Contenido de la clase…..
}

5.2.3 Interfaces

- Se utilizará el nombrado UpperCamelCase.
- Los nombres de las interfaces deben ser simples y descriptivos.
- Utilizar palabras completas, evitar acrónimos y abreviaturas

Ejemplo: **public Interface Ciudadano {**
……Contenido de la Interface…..
}

5.2.4 Métodos

- Debe utilizarse el nombrado lowerCamelCase.
- Los nombres de los métodos deben ser simples y descriptivos.
- Los métodos deben iniciar por verbos.

Ejemplo: **obtenerCedula ();**

5.3 Variables y Constantes

5.3.1 Variables y atributos (clases)

- Deben ser cortas y descriptivas de si mismas.
- Para las variables creadas dentro de la función , procedimiento almacenado, método , etc., se utilizará el prefijo v_
- Para la variables recibidas por parámetro tendrá el prefijo p_
- Para los cursores de Bases de Datos se utilizará el prefijo c_
- Las variables después de los prefijos utilizarán la notación lowerCamelCase.
- Los nombres de variables cortos p.e. v_i, v_j, p_i se deben evitar, excepto para variables de índices temporales.

5.3.2 Constantes

- Los nombres de las variables declaradas como constantes o variables globales deben ir totalmente en mayúsculas separando las palabras con un subguión ("_").
- Las constantes ANSI se deben evitar, para facilitar su depuración.

5.3.3 Ejemplos

5.3.3.1 SYBASE

```
create procedure pr_CalcularEdad
@p_numeroDocumento numeric(15,0),
@p_tipoDocumento char(1)
as
BEGIN
DECLARE
        @mi_edad numeric(3,0),
        @mi_fecha date
        IF NOT EXISTS(select 1
                from  SABASS..Afiliado
                WHERE Af_Numerodocumento  = @p_numeroDocumento
```

```
                    and   Af_TipoDocumento    = @p_tipoDocumento)
          BEGIN
                    RAISERROR 20001 'El afiliado NO EXISTE'
                    RETURN
           END
          SELECT @mi_fecha=Af_FechaNacimiento
          FROM Sabass..Afiliado
          WHERE Af_NumeroDocumento  = @p_numeroDocumento
          and   Af_TipoDocumento    = @p_tipoDocumento

          select @mi_edad=Year(getDate())-Year(@mi_fecha)

           UPDATE Sabass..Afiliado
          set Edad=@mi_edad
           WHERE af_NumeroDocumento  = @p_NumeroDocumento
          and   af_TipoDocumento    = @p_TipoDocumento
END;
```

5.3.3.2 Java

```
package co.gov.colpensiones.comun.mail;
import java.io.File;
import javax.persistence.EntityManager;
import org.jboss.seam.Component;
import org.jboss.seam.contexts.Contexts;
import org.jboss.seam.contexts.Lifecycle;
import org.jboss.seam.faces.Renderer;
public class Mail {
   public void sendMail(String[] p_to, String p_from, String p_subject, String p_message, String p_facelet) {
      Lifecycle.beginCall();
      Renderer v_renderer = (Renderer) Component.getInstance("org.jboss.seam.faces.renderer", true);
```

```
Contexts.getConversationContext().set(ConstantesMediosMagneticos.ASUNTO_ERROR_REGISTROS_INCONS
ISTENTES, p_subject);

Contexts.getConversationContext().set(ConstantesMediosMagneticos.MENSAJE_ERROR_REGISTROS_INCON
SISTENTES, p_message);

Contexts.getConversationContext().set(ConstantesMediosMagneticos.FUENTE_ERROR_REGISTRO_INCONSIS
TENTES, p_from);

    for (int v_i = 0; v_i < p_to.length; v_i++) {

      p_to[v_i] = p_to[v_i].trim();

    }
Contexts.getConversationContext().set(ConstantesMediosMagneticos.DESTINATARIO_ERROR_REGISTRO_IN
CONSISTENTES,p_to);

    v_renderer.render(p_facelet);

  }
```

5.4 Convenciones de Programación Java

5.4.1 Con respecto a la indentación

- Usar cuatro espacios como unidad de indentación.
- Cuando una expresión no entre en una sola línea, se debe romper de acuerdo a estos principios generales:
- Romper después de una coma
- Romper antes de un operador binario, por ejemplo.

```
numeroEntregado
= 50;
```

- Alinear la nueva línea al principio de la expresión al mismo nivel de la línea anterior.
- Si las reglas anteriores conducen a la confusión del código o el código se alinea contra el margen derecho, indentamos menos espacios
  Ejemplo:

```
someMethod(longExpression1, longExpression2, longExpression3,

longExpression4, longExpression5);

var

= someMethod1(longExpression1,

someMethod2(longExpression2,longExpression3));
```

5.4.2 Con respecto a las Declaraciones

- Se debe declarar cada variable en su propia línea.
- Es válido inicializar variables al momento de su declaración.
  Ejemplos válidos:

```
int inumero = 9;

int icantidad;
```

NO utilizar:

```
int inumero, icantidad;
```

5.4.3 Sentencias

- Buscar usar sentencias simples, cada sentencia debe ser escrita en su propia línea, es decir, no utilizar líneas como:

```
inumero++; a = b + 2;
```

- No utilizar el operador coma para agrupar declaraciones múltiples. Evitar:

```
if(err) {
        Format.print(System.out, "error"), exit(1);
}
```

5.4.4 Sentencias de retorno

- Una sentencia return con un valor no debe usar paréntesis a menos que hagan el valor de retorno más obvio de alguna manera. Ejemplo:

```
return;

return miDiscoDuro.size();

return (tamanyo ? tamanyo : tamanyoPorDefecto);
```

- Intentar hacer que la estructura del programa se ajuste a su intención. Ejemplo:

```
if (expresionBooleana) {
    return true;
} else {
    return false;
}
```

en su lugar se debe escribir

```
return expressionBooleana;
```

5.4.5 Sentencias if

- Deben tener la siguiente forma:

```
if (condition) {
    statements;
}
if (condition) {
    statements;
} else {
    statements;
}
```

- Las sentencias if siempre utilizan paréntesis, evitar lo siguiente:

```
if (condition)
 statement;
```

5.4.6 Sentencias for

- Deben tener la siguiente forma:

```
for (initialization; condition; update) {
      statements;
}
```

- Una sentencia for vacía (una en la cual todo el trabajo es hecho en la cláusula de inicialización, condición y actualización) debe tener la siguiente forma:

```
for (initialization; condition; update);
```

5.4.7 Sentencias while

- Deben tener la siguiente forma:

```
while (condition) {
      statements;
}
```

- Una sentencia while vacia deberia tener la siguiente forma:

```
while (condition);
```

- Una sentencia do – While debe tener la siguiente forma:

```
do {
      statements;
} while (condition);
```

- Se debe tener en cuenta que este tipo de sentencias se ejecutan por lo menos una vez antes de evaluar la condición.

5.4.8 Sentencias switch

- Deben tener la siguiente forma:

```
switch (condition) {
case ABC:
      statements;
      /* no se concreta */
case DEF:
      statements;
      break;
case XYZ:
      statements;
      break;
default:
       statements;
}
```

- Cada vez que un caso no se concreta (no incluir la sentencia break), agregue un comentario donde la sentencia break debería ir normalmente. Esto es mostrado en el ejemplo precedente con el comentario /* no se concreta */.

- Cada sentencia switch debe incluir un case por default.

5.4.9 Con respecto a las Sentencias Try – Catch

- Deben tener el siguiente formato:

```
try {
      statements;
} catch (ExceptionClass e) {
      statements;
}
```

- Se recomienda usar la sentencia finally, cuando como parte de los statements se maneja conexiones a BD, para asegurar que la conexión a BD sea cerrada.

```
try {
    statements;
} catch (ExceptionClass e) {
    statements;
} finally {
    statements;
}
```

5.4.10 Con respecto a las concatenaciones

- La concatenación de cadenas String no debe realizarse con la unión de varias cadenas, pues esto consume recursos de memoria innecesariamente. Se debe utilizar la clase StringBuffer para realizar la concatenación.
- Este es un modo incorrecto de concatenar cadenas:

```
String cadenaFinal = "Esta es " + "una " + "concatenación " +
"incorrecta";
```

- Este es un modo correcto de concatenar cadenas:

```
StringBuffer sb = new StringBuffer();
```

```
sb.append("Esta es ");

sb.append("una ");

sb.append("concatenación ");

sb.append("correcta");

String cadenaFinal = sb.toString();
```

5.4.11 Con respecto a las colecciones

- Se deben evitar usar clases que van a ser deprecadas, en lugar de Hashtable usar HashMap, en lugar de usar Vector usar ArrayList, en lugar de Enumeration usar Iterator.

5.5 Convenciones para Fuentes Htmls Y Jsps

- Los includes jsp pueden contener scriptlets jsp o código HTML. No deben incluir funciones o rutinas. Todo el contenido del include formará parte del JSP generado. Se recomienda que los includes no sean muy extensos, y normalmente usarlos para cabecera, pie de página o frames. Ejemplo de uso:

  ```
  <%@include file="../include/rsControl.jsp"%>
  ```

- Evitar asignar el mismo valor a varias variables en un misma sentencia. En lugar de usar:

  ```
  fooBar.fChar = barFoo.lchar = 'c' // Evitar
  ```

  Usar:

  ```
  fooBar.fChar = barFoo.lchar;

  barFoo.lchar = ' c ';
  ```

- Utilizar Cascade Style Sheets en vez de asignar colores directamente a los elementos de la página. Por ejemplo:

  ```
  <input id="txtNombre" name="txtNombre" class="clsControlObligatorio">
  ```

  Ejemplo de lo que no se debe hacer:

  ```
  <input id="text1" name="text1" style="BACKGROUND-COLOR: aqua; FONT-FAMILY: sans-serif; FONT-SIZE: x-small; FONT-STYLE: normal">
  ```

- En vez de leer de un objeto una propiedad varias veces, es mejor asignar la propiedad a una variable y usarla
- ```
  UserVO user =
  ```
  ```

```
UserVO)session.getAttribute(GlobalConstant.SESSION_USER);

String sCodUsr = String.valueOf(user.getCodigoUsuario());
```

- Mover el código a Clases en vez de tenerlo en un JSP. Dentro de lo posible no poner en los componentes la visualización de la data.
- En lo posible No intercalar los <% %> con el código HTML.
- Guardar la data más usada y estática en variables Application, usar el mismo criterio que para las Session.
- Copiar la data más usada de los objetos en variables sencillas.
- El código fuente debe tener un formato que refleje la estructura lógica y los niveles de anidamiento. Para reflejar los niveles de anidamiento el desarrollador debe usar un espaciado de un tamaño estándar de 4 posiciones.

5.6 Convenciones de Documentación

Los programas Java pueden tener dos tipos de comentarios: comentarios de implementación y comentarios de documentación. Los comentarios de implementación están delimitados por /* … */ , y //.

Los comentarios de documentación (conocidos como “comentarios doc”) son únicos en Java y están delimitados por /** … */ estos pueden ser extraídos para convertirlos a archivos HTML usando la herramienta Javadoc. Todo el código Java debe estar documentado con el estándar Javadoc, para permitir exportar el documento correspondiente.

Los comentarios de implementación son utilizados para comentar código o para comentarios sobre la implementación particular. Los comentarios de documentación son usados para definir la especificación del código, desde una perspectiva libre de implementación para ser leído por los desarrolladores quienes no necesariamente tienen el código fuente a disposición.

Los comentarios deben ser usados para brindar una vista general del código y proveer información adicional que no está disponible en el propio código. Los comentarios deberían contener sólo información que es relevante para leer y entender el programa. Por ejemplo, información acerca de cómo el paquete correspondiente ha sido desarrollado o en que directorio reside no deben ser incluidos en los comentarios.

Los comentarios sobre las decisiones de diseño importantes o no obvias son apropiados. Pero evitar hacer una copia de la información que esta presente (y en forma clara) en el código. Es muy fácil que los comentarios queden desactualizados.

Consideramos que los programas tienen tres estilos de comentarios de implementación: Bloque de Comentarios, Comentarios de una línea, comentarios finales.

5.6.1 Bloque de Comentarios

Los bloques de comentarios se usan para proporcionar descripciones de métodos, estructuras de datos y algoritmos. También pueden usarse en otros lugares, como, dentro de los métodos. Los bloques de comentarios dentro de una función o método deben estar indentados al mismo nivel que el código que describen. Un bloque de comentario debe ir precedido por una línea en blanco para configurar un apartado del resto del código.

```
/*

* Aquí un bloque de comentarios

*/
```

5.6.2 Comentarios de una línea

Los comentarios cortos pueden aparecer como una sola línea indentada al nivel del código que la sigue. Si un comentario no se puede escribir en una sola línea, debería seguir el formato de los bloques de comentario. Un comentario de una sola línea debería ir precedido de una línea en blanco. Aquí tenemos un ejemplo:

```
int i = 0;

/* Manejando la Condición. */

if (condition) {

     ...

}
```

5.6.3 Comentarios finales

El delimitador de comentario // puede comentar una línea completa o una línea parcial. No debería usarse en líneas consecutivas para comentar texto; sin embargo, si puede usarse en líneas consecutivas para comentar secciones de código. Ejemplo:

```
if (cliente.equals(empleado)) {

      ...

} else{

       return false;    //no es un cliente.

}
```

Los comentarios de documentación describen clases Java, interfaces, constructores, métodos y

campos. Cada comentario de documentación va dentro de los delimitadores de comentario /** … */, con un comentario por clase , interface o miembro. Este comentario debe aparecer justo antes de la declaración.

Toda clase debe empezar con un comentario de cabecera que indique el autor, la fecha de creación y una descripción breve del objetivo de la clase:

```
/**
 * @author Colpensiones
 * @version 1.0
 * 11/09/2006
 * La clase Rol.java ha sido creada con el fin...
 *
 */

public class Rol { ...
```

La primera línea del comentario del documento (/**) para las clases e interfaces no esta indentada; las subsecuentes líneas del comentario de documentación tendrán un espacio de indentación (para alinear verticalmente los asteriscos). Los métodos, incluyendo constructores, tienen 4 espacios para la primera línea de comentario de documentación y 5 espacios después.

Si necesitamos dar información sobre una clase, una interface, una variable o un método que no es apropiada para documentación, usamos una implementación de bloque de comentario, o una declaración de una línea inmediatamente después de la declaración. Por ejemplo, los detalles sobre la implementación de una clase deberían ir en un bloque de comentario seguido por la sentencia class, no en el comentario de documentación de la clase.

Los comentarios de documentación no deben colocarse dentro de un bloque de definición de un método o constructor porque Java asocia los comentarios de documentación con la primera declaración que hay después del comentario.

El bloque de comentarios al inicio de cada método debe considerar lo siguiente:

Nombre Nombre del método
Propósito QUE es lo que hace el método (NO COMO).
Inputs Cada parámetro no obvio en líneas separadas con comentarios en

línea.
Retorno Explicación del valor retornado por el método.

En los casos de modificación del código fuente existente, se aplica el siguiente formato, para documentar el cambio:

```
/* Fecha de Modificacion: dd/mm/aaaa
 * Modificado por: Nombre Apellido
 * Requerimiento No: XXX
 * Descripción de la modificación:
 *    Incluir la validación de periodos cotizados
*/
```

5.7 Otras Consideraciones en la Construcción

Se debe manejar una página de error única por aplicación. Esta debe incluir la posibilidad de que el usuario envíe un e-mail informando sobre el problema.

Una vez autenticado un usuario, se debe considerar guardar como variables de sesión los datos de uso continuo como por ejemplo: el código de usuario, los apellidos y nombre, el documento de identidad, y otros similares o los necesarios para la aplicación en especial.

En java se usará el tipo de dato BigDecimal para el manejo de montos monetarios en pesos. Para estos casos no se usará el tipo de dato double. Se debe manejar una precisión de 2 decimales.

Si la aplicación va a ser accedida por terceros se debe considerar tener un log donde se registre información relativa a la sesión:

- Código de usuario.
- Dirección IP desde la que realizo el request.
- Fecha y hora.
- Tipo de usuario.
- URL de la opción ejecutada.
- Fecha y hora de inicio y finalización de su sesión.

El manejo de los errores en tiempo de ejecución en los códigos java debe considerar el registro de un log de errores, de la información relativa al error ocurrido. Esta información debe incluir la fecha y hora de ocurrencia del error, el usuario al que le ocurrió el error, el tipo de usuario y la clase en la que ocurrió el error.

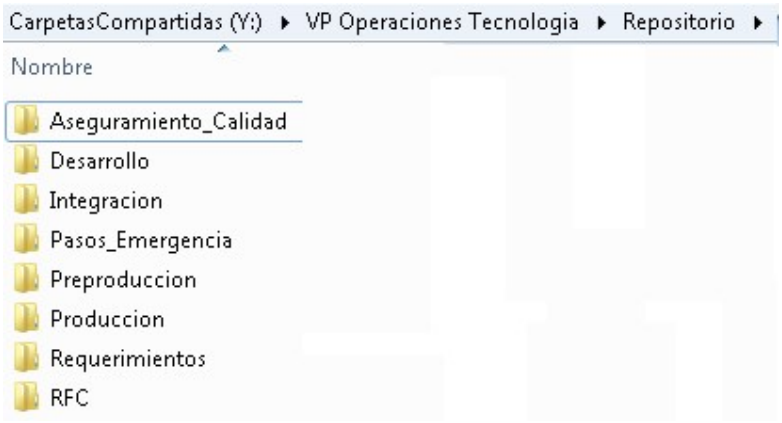
En general, los mensajes deben ser lanzados al llamador, de tal manera de centralizar el manejo de

los errores. Esto aplica para el manejo de los errores “inesperados” del sistema. Los errores inesperados o Runtime son los errores de operación de la aplicación, como por ejemplo “NullPointerException”, “Divide by zero”, “ArrayIndex out of Bounds”.

También se debe usar una forma estándar para el manejo de los errores “esperados”, estos son los errores provocados por alguna regla o validación de negocio. Estos errores son normales y se espera que aparezcan durante el funcionamiento del sistema, por ejemplo: "No existe stock suficiente.".

6. ESTRUCTURA DIRECTORIOS DESARROLLOS

Para el paso al ambiente de integración de las soluciones se establece el uso de la carpeta de repositorio como esta la estructura actual:



La ruta de las carpetas de repositorio es:

Y:\VP Operaciones Tecnologia\Repositorio

Allí se encuentran las carpetas para desarrollo, integración, aseguramiento de calidad, Preproduccion, Produccion y Pasos de emergencia.

Las capetas asignadas al grupo de desarrollo son:

Desarrollo, integración y Pasos de emergencia.

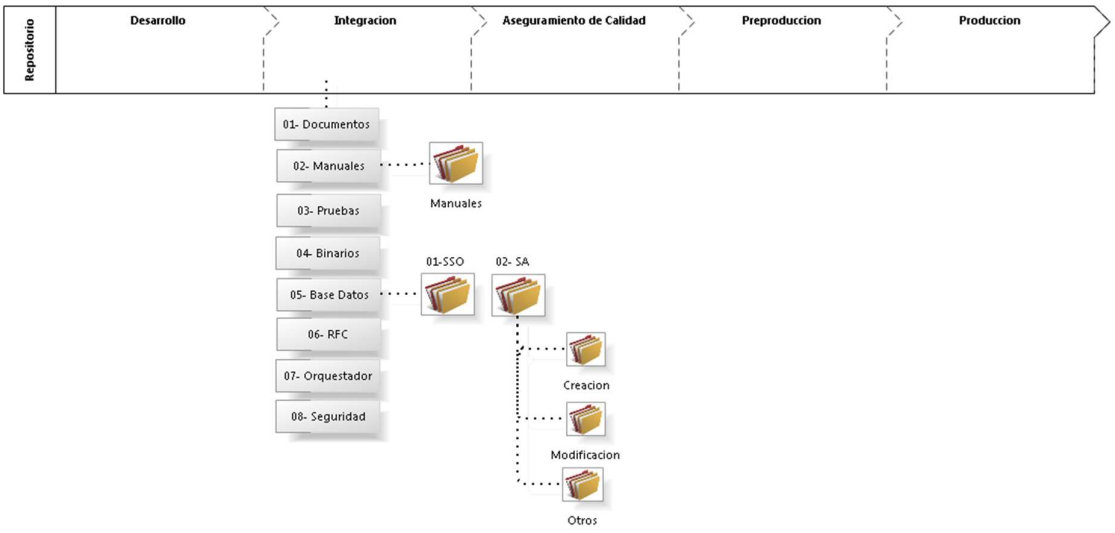
Las capetas asignadas al grupo de SQA son:

Integración, Aseguramiento de calidad, Preproduccion

Las capetas asignadas al grupo de Produccion (BD, Capa Media, SSO, Orquestador, Seguridad) son:

Integración, Preproduccion, Produccion y Pasos de emergencia.

Dentro de cada carpeta según el ambiente de catalogación se encuentra la siguiente estructura de carpetas:



Así en las carpetas se deberá dejar cada entregable según aplique:

01- Documentos:

Contiene los documentos relacionados a la lista de entregables (casos de uso, especificaciones, etc.) los documentos deben venir numerados como 01-Nombre_Documento.ext, si existe mas de uno se deberá mantener el orden de numeración e indexación.

02- Manuales:

Contiene los manuales entregables (Manual de usuario, técnico, de instalación) los documentos deben venir numerados así:

1-Manual_Instalacion.PDF, si existe mas de uno se deberá mantener el orden de numeración e indexación (ejemplo; 1-Manual_Instalacion1.PDF, 1-Manual_Instalacion2.PDF)

2-Manual_Tecnico.PDF, si existe mas de uno se deberá mantener el orden de numeración e indexación (ejemplo; 2-Manual_Tecnico1.PDF, 1-Manual_Tecnico2.PDF)

3-Manual_Usuario.PDF, si existe mas de uno se deberá mantener el orden de numeración, indexación (ejemplo:3-Manual_Usuario1.PDF,3-Manual_Usuario2.PDF)

03- Pruebas:

Contiene los entregables de prueba (casos de prueba, script de prueba, plan de pruebas, evidencias de prueba, certificación de pruebas técnicas realizadas por la fábrica, evidencias de pruebas de Integración realizadas. Los documentos debe estar en PDF y debe mantener la numeración y codificación que tiene el proceso Gestion de SQA en el SIG.

04- binarios:

Se debe dejar los archivos binarios que se requiere desplegar.

05- Base de Datos:

En la raíz de la carpeta se encuentra el archivo de catalogación el cual es un documento en Excel y define el orden de catalogación requerido, se define de acuerdo a las políticas del protocolo de base de datos.

Nota: Cada objeto (tabla, vistas, triggers y procedimientos almacenados) debe estar en scripts independientes

Al interior de la carpeta debe existir 2 carpetas: 01-SSO y 02-SA.

Carpeta 01. SSO: Se debe dejar los script a ser ejecutados por el grupo SSO para la creación y asignación de seguridad. Estos scripts se deben ajustar a las políticas del presente documento en cuanto a codificación de nombres, y debe venir numerados según el orden de ejecución requerido como figuran en el archivo de catalogación sección SSO.

Carpeta 02.SA: Se debe dejar los script a ser ejecutados por el grupo SA para la creación, modificación o eliminación de objetos en base de datos. Estos scripts se deben ajustar a las políticas del presente documento en cuanto a codificación de nombres, y debe venir numerados según el orden de ejecución requerido como figuran en el archivo de catalogación sección SA.

Dentro de la carpeta se encuentran 3 carpetas más así:

En ella se deberá ubicar los scripts de catalogación según las siguientes reglas

En la carpeta **Creación** lo pertinente a creación de objetos en BD. El orden lógico para creación de script, de acuerdo a los tipos de objetos es:

01. Creación de Tablas, llaves primarias, sus índices y llaves foráneas.

02. Creación de Vistas.

03. Creación de Triggers.

04. Creación de Procedimientos almacenados

En la carpeta **Modificacion** lo pertinente a la alteración objetos en BD. El orden lógico para esta tarea de acuerdo a los tipos de objetos es:

01. Alter de Tablas, llaves primarias, sus índices y llaves foráneas.

02. Alter de Vistas.

03. Alter de Triggers.

04. Alter de Procedimientos almacenados

En la carpeta **Otros** se dejara los scripts de operaciones en la base datos así:

01-Asignación de permisos al usuario a los objetos creados

02-Scripts de Cargues de información (Si aplica)

03-Scripts de Ejecuciones

04-Scripts de Extracción de información (si aplica)

05-Scripts de Borrado de objetos creados

7. MEJORES PRACTICAS DESARROLLO BASES DE DATOS

7.1 Mejores prácticas desarrollo Sybase

7.1.1 Tablas

1. Asigne nombres a las tablas que puedan identificar cada objeto de manera única dentro de cada dataserver.

2. Utilice tablas de tipo DOL (Datarows o Datapages) únicamente en las tablas que puedan tener altos niveles de contención. Con el fin de optimizar uso de candados.

3. En lo posible, definir tipos de datos de ancho fijo y no variable (varchar), con el fin de evitar el movimiento de registros a otras páginas por actualización en estos campos, lo disminuye las posibilidades de fragmentación.

4. Asegure que los tipos de datos utilizados, corresponden con los valores que serán almacenados. Ejemplos:

- Para campos de valor entero defina el tipo (tinyint, smallint, int, bigint) de acuerdo al mayor valor posible a almacenar.
- Campos de tipo fecha – hora (date, time, smalldatetime, datetime, etc) c. Ancho máximo de cadenas de texto en campos de tipo char.

5. Utilice particionamiento en tablas donde se conjuguen las siguientes situaciones: a. altos niveles de inserción de datos y b. concurrencia de sesiones de usuario.

7.1.2 Indices

1. Evite duplicar índices (mismos campos, en el mismo orden) en la misma tabla.

2. No construya índices en tablas, que puedan estar contenidos en otros índices o que puedan tener la misma funcionalidad: a. El índice A,B,C no debe ser construido, si existe un índice A,B,C,D. ya que el segundo cubre la misma funcionalidad del primero.

El índice X,Y,Z es diferente al índice Z,Y,X, por tanto su funcionalidad puede ser diferente.

3. En el caso de llaves foráneas: los campos utilizados como referenciados (tabla foránea) deben estar vinculados a un índice de llave única. Los campos utilizados como referencia (tabla local) deben estar vinculados a un índice.

4. En lo posible defina campos básicos (o combinaciones de estos), para los accesos a datos y sobre estos construya índices (llaves primarias, índices secundarios), con el fin de tener la mínima cantidad de índices en cada tabla, esto favorecerá el tiempo que se requiere para la ejecución de transacciones.

5. No incluya demasiadas columnas en un índice. En lo posible no incluya columnas que puedan ser modificadas de manera frecuente.

6. Construya índices donde el nivel de selectividad sea pequeño, con el fin de garantizar que pequeños bloques de datos serán accedidos por cada llave del índice en cada recorrido (Ejemplo: un índice que permite acceder hasta el 5% de los registros por cada llave, tendrá mejor selectividad y por ende, mejor desempeño, que uno que acceda al 10% de los registros por cada llave). Tenga en

cuenta que dependiendo de la selectividad de la llave de búsqueda se definirá el método de acceso a los datos (Table Scan, Positioning by key, Positioning at index start o Positioning at index end).

7. Si la tabla va a ser replicada mediante Replication Server, se requiere que cada tabla cuente con una llave primaria o mínimo un índice único.

8. Evite la construcción de índices clustered en tablas de tipo DOL, ya que se degradará más rápido y podría no ser funcional (salvo que se garanticen rutinas periódicas de mantenimiento que reconstruyan el índice).

9. Campos donde se vayan a realizar operaciones MIN o MAX es recomendable que estén indexados junto con los campos que puedan ser utilizados en las búsquedas. select min(campo3) from t1 where campo1='A' and campo2=10 create index indx1 on t1(campo1,campo2,campo3) select max(campo3) from t1 create index indx2 on t1(campo3)

10. Contemple los siguientes requerimientos para definir si un índice debe ser clustered:

a. Lo fundamental es que sea una llave que es ingresada siempre de manera incremental de acuerdo al índice (nunca hay inserciones en lugares intermedios).

b. Preferiblemente, llave que forma la clave primaria de la tabla, aunque no es obligatorio. c. Llave que podría ser accedida por rangos d. Posible uso en clausulas order by, group by y joins.

11. En caso de poseer sentencias de filtrado, donde se requiera el uso de funciones para la validación, se pueden construir índices sobre dichas funciones (Indexing with function-based indexes), o sobre combinaciones campos + funciones.

```
select * from t1

where campo1='A' and

campo2 % 2 = 0

create index ind_t1 on t1(campo1,campo2 % 2)
```

Nota: Al respecto de los índices sobre funciones, se debe tener en cuenta que para poder construir el índice, SAP ASE materializa el valor del cálculo de la función para cada registro en un nuevo campo de la tabla y que luego dicho valor es el utilizado para la construcción del índice, actualizándose cada vez que se actualice el campo origen (por tanto estos índices demandaran más espacio que un índice normal). Dichos campos para materializar índices no son accesibles por los usuarios, por tanto debe asegurar que si se utilizan drivers diferentes a los de SAP estos soportan de manera adecuada el manejo de estas tablas.

7.1.3 Tablas temporales

1. Regule la cantidad de datos que serán escritos en tablas temporales (de tipo # o worktables). Con el fin de no sobrepasar los límites en data o log, de las bases de datos temporales, recuerde que en general estas bases:

a. Poseen menores capacidades de almacenamiento que las bases de datos de usuario (por lo general son bases de datos más pequeñas).

b. Son volátiles, por tanto grandes esfuerzos de I/O, escribiendo grandes cantidades de registros podrían perderse en caso de una falla.

2. Escriba en las tablas temporales solo los campos que requiera, en lo posible no realice creaciones de tablas temporales a partir de SELECT * INTO, si en realidad no requiere todos los campos.

3. Bajo toda circunstancia de manejo de tablas temporales, se recomienda una vez llenada con datos, realizar la creación de índices sobre dichas tablas (no antes de la inserción), con el fin de que el optimizador pueda calcular planes de ejecución más exactos. En el caso de procedimientos almacenados, si se llega a realizar esta operación dentro del cuerpo de ejecución del procedimiento, una vez se construya el índice se recompilara el plan de ejecución del procedimiento para ajustar los planes referentes a dichas tablas temporales.

4. Para casos donde se requieran acceder y manipular datos en tablas con grandes volúmenes de registros, una posible opción de optimización, es extraer (en bloques más pequeños) los datos a tablas temporales y luego allí hacer las operaciones que se requieran. En caso de requerir llevar datos a las tablas fijas de nuevo, hacerlo al final de toda la operación.

7.1.4 Sentencias

7.1.4.1 Construcción

1. Límite las sentencias SELECT a los campos requeridos. Evite seleccionar columnas que no serán utilizadas.

2. Asegure que en los filtros, los campos son del mismo tipo y precisión que las variables utilizadas (@)

3. Asegure que en consultas que vinculen N tablas (donde N ≥ 2), existan N-1 joins que conjuguen todas las tablas, como mínimo1.

4. Asegure que en los joins, los campos de intersección son del mismo tipo y precisión.

5. Evite el forzamiento de índices.

6. Operaciones donde se realicen agrupamientos/ordenamientos por campos A,B,C se beneficiaran de la existencia de índices que tengan los mismos campos en el mismo orden (ejemplo: índices A,B,C,…., pueden existir campos adicionales en el índice)

7. Evite el uso de validaciones NOT EXISTS o NOT IN, ya que en la mayoría de ocasiones obligan a realizar mayores recorridos sobre los objetos, incrementando la cantidad de I/O utilizado.

8. Evite el uso de % al comienzo de cadenas de comparación LIKE (LIKE '%ABCD%') ya que se presta para que se obligue el recorrido de los datos por medio de Table Scan.

9. Optimice el uso de funciones. Ejemplos:

a. No utilice la función isnull() sobre campos que no permiten valores nulos.

b. Optimice el uso de substring(), para extractar cadenas desde el primer campo, en conversiones de datos de tipo fecha: substring(convert(char(10),getdate(),112),1,6) puede ser optimizado así convert(char(6),getdate(),112)

c. En general, el uso adecuado de funciones, reducirá el tiempo requerido en la resolución de una consulta.

10. No realice validaciones de existencia de registros mediante el uso de count(), en dichos casos utilice la validación EXISTS.

11. Utilizar todos los argumentos de búsqueda que pueda dar el optimizador tanto como sea posible para trabajar.

12. Analice si los datos del conjunto solución involucrados en las sentencias, crecerán en la medida en que crezca la base de datos, ya que esto indicaría que el costo de 1 Un join es la relación de comparación (=,!=,<=,,>=,etc) entre campos de dos tablas. La ejecución de la sentencia también se incrementará con el tiempo, lo que puede llegar a incurrir en futuras optimizaciones.

13. Sea prudente con la anidación de validaciones de los condicionales AND y OR, tenga en cuenta que validaciones OR pueden ser reescritas como IN:

```
select t1.* from t1, t2

where t1.campo1=t2.campo1 --(1) and

t1.campo2='A' --(2) and

t2.campo3='B' --(3) or

 t2.campo3='C' --(4)
```

Se traerán todos los registros que cumplan al unísono las condiciones 1, 2 y 3 o la 4 por separado.

```
select t1.*

from t1, t2

 where t1.campo1=t2.campo1 --(1) and
```

t1.campo2='A' --(2) and

(t2.campo3='B' --(3) or t2.campo3='C') --(4)

Se traerán todos los registros que cumplan las condiciones 1, 2 y que de las validaciones 3 y 4, una de las dos sea verdadera.

select t1.*

from t1, t2 where t1.campo1=t2.campo1 --(1) and t1.campo2='A' --(2) and

t2.campo3 in ('B','C') --(3)

Se traerán todos los registros que cumplan las condiciones 1, 2 y 3. 14. Evite las funciones, operaciones aritméticas, y otras expresiones en el lado de la columna de cláusulas de búsqueda, en caso de requerirlo tenga en cuenta la recomendación sobre construcción de índices sobre funciones.

Incorrecto

select * from t1

where campo1 = 'A' and campo2 * 2 = 20

Correcto

select * from t1

where campo1 = 'A' and campo2 = 10 15.

Si los joins involucran varias tablas, es recomendable hacer uso de tablas temporales para dividir las consultas y armar conjuntos de trabajo menores que luego puedan ser conjugados.

7.1.4.2 Evaluación de Sentencias

Evalué cada sentencia por medio de sp_showplan, con el fin de determinar:

a. Recorridos definidos en el plan de ejecución, para evitar situaciones de Table Scan, Positioning at index start o Positioning at index end.

b. Costo de I/O estimado para la ejecución de la sentencia (a mayor valor, posiblemente tome más tiempo su resolución). Un buen valor de Costo de I/O puede estar por debajo de un factor de 100.000 páginas2 . c. Disminuir situaciones de Deferred Updates.

2. Utilice set option show_missing_stats, para determinar posibles campos (o conjunción de campos), candidatos para construcción de índices.

3. Por medio de set statistics io, evalué si se presentan situaciones de Prefech. En lo posible, se debe procurar disminuir estas mediante la optimización de los recorridos en objetos.

4. Por medio de set statistics time, evalué el tiempo de ejecución requerido para cada sentencia. 3 TRANSACCIONES 1. Procure que los bloques transaccionales sean lo más cortos posibles, no involucre grandes cantidades de registros en una única operación:

a. Incrementa las posibilidades de que se presenten bloqueos y que estos tomen mayor tiempo.

b. Incrementa el log requerido y por ende el tamaño de los backups transaccionales.

c. Las operaciones de rollback pueden tomar tanto tiempo, como tomo la operación de modificación.

d. Incrementa los tiempos de recuperación de las bases de datos, en caso de una falla que provoque la detención del dataserver.

e. En caso de que las transacciones sean aplicadas sobre tablas particionadas, aumenta la posibilidad de que estas se desbalanceen más rápido, disminuyendo su desempeño. Cualquier transacción que adquiere bloqueos debe mantenerse lo más corta posible. Siempre será mejor, realizar 100 transacciones, donde cada una bloquee 1000 registros, a una sola que bloquee 100.000 registros.

5. En particular, evitar las transacciones que deben esperar para la interacción con el usuario mientras mantiene bloqueos.

6. El uso de bloques BEGIN TRAN – COMMIT – ROLLBACK desde la aplicación, la hace susceptible a mantener por mayor tiempo bloqueos sobre las tablas o a que 2 Dicho valor puede depender de muchos factores: sentencias OLTP vs DSS, cantidad de registros en las tablas involucradas, tamaño de los caches, algoritmo de asignación en estos, entre otros. Entre más pequeño sea este valor, mejor desempeño de la sentencia perdidas de conexión puedan dejar transacciones abiertas (bloqueando los objetos implicados en la transacción), para esto:

a. Traslade dichos bloques a procedimientos almacenados en el dataserver, que reciban los datos como parámetros y ejecuten la transacción de manera integral en el motor.

b. Minimice el uso de bloques BEGIN TRAN – COMMIT – ROLLBACK, desde la aplicación, de ser posible remita únicamente la operación de modificación (insert, update, delete) que requiera, en caso de alguna falla el motor mismo hará el rollback de la operación y retornará el error hacia la sesión cliente.

7. Actualice todos los campos necesarios en un registro, en un único acceso.

8. Procure que las actualizaciones se realicen bajo el método direct updates, ya que utilizan menor número de candados y son más eficientes.

9. Evite modificar el ISOLATION LEVEL de la sesión, con el fin de garantizar la integridad de los datos. 10. Evite el uso de encadenamiento de transacciones activo (CHAINED ON) en las sesiones de usuario.

7.1.5 Objetos procedimentales

7.1.5.1 Cursores

1. En la declaración del cursor defina su finalidad:

a. Para modificaciones FOR UPDATE (de manera explícita señalar las columnas a modificar).

b. Para solo lectura FOR READ ONLY 2. Use UNION o UNION ALL en vez de cláusulas OR o listas IN.

2. Evalué el plan de ejecución de la sentencia de definición del cursor por aparte, dado que no se retornará detalle alguno de dicho plan durante la ejecución del cursor.

3. No involucre grandes conjuntos de datos en los recorridos de los cursores, en preferencia, límite el uso de este mecanismo a conjuntos con miles de registros. En caso de requerirse el recorrido de conjuntos que puedan contener más de decenas de miles de registros se recomienda optar por mecanismos como recorridos con validación WHILE.

4. Establezca el orden de recorrido de los datos mediante la declaración explicita de ORDER BY.

7.1.5.2 Procedimientos

1. Procure realizar la menor cantidad de lecturas del mismo registro dentro de una misma operación. 2. Evite el uso de código dinámico. 3. Disminuya al mínimo necesario, la interacción entre el cliente y el procedimiento durante su ejecución, como la impresión de registros o labels que no sean necesarios.

3. No incluya en bloques transaccionales, código de validación de datos u otras operaciones. Estas sentencias deberían ser realizadas de manera previa y la transacción solo debería abarcar las modificaciones que sean requeridas.

```
begin tran

select .... if ....

begin

end

select .... update ... delete ... insert ...

if .... commit

  else rollback

select .... if ....

 begin

 end
```

select .... begin tran

update ... delete ... insert ...

if .... commit

else rollback

4. Si en el cuerpo del procedimiento se invocan tablas temporales que no son creadas dentro del mismo procedimiento, coloque el código de creación de manera comentada al inicio del procedimiento. Esto facilita las tareas de revisión y recompilación, en caso de ser necesario.

5. Para mayor precisión en cálculos numéricos que puedan incurrir en valores de punto flotante, utilice datos de tipo flotante (float, real, double) para todos los valores implicados en el cálculo, con el fin de no afectar la precisión de los resultados.

6. Si tablas temporales son llenadas en el procedimiento, cree índices de manera posterior al llenado con el fin de que el procedimiento ajuste su plan de ejecución.

7. Controle la cantidad de registros que pueden ser retornados por un procedimiento.

8. Indente el código de manera adecuada para facilitar su lectura.

9. Defina prefijos diferentes, que permitan identificar la finalidad de cada variable:

a. parámetros de entrada: variables utilizadas para lanzar el proceso.

b. parámetros de salida: variables que son retornadas con datos.

c. variables de trabajo: variables internas al procedimiento

10. Añada comentarios que permitan documentar el flujo del proceso dentro del procedimiento.

7.1.5.3 Triggers

1. Haga uso de estos únicamente para:

a. Operaciones de validación de datos en registros (validaciones que no incluya largos procesos).

b. Validación de integridades referenciales o borrados en cascada.

c. Para registrar en auditorias datos sobre la operación realizada.

2. Evite añadir al código del trigger, intrínsecas operaciones sobre otras tablas o realización de cálculos sobre otros bloques de registros a partir de los datos implicados en el trigger; para tal fin:

a. Cree procedimientos almacenados que reciban como parámetros los valores de los registros implicados en la operación.

b. Dentro del código del procedimiento realice las operaciones de validación y cálculo que requiera con los registros de manera previa a la ejecución de la transacción.

c. En un bloque transaccional, aplique todas las modificaciones requeridas en todas las tablas implicadas.

3. Si se requiere realizar joins entre tablas de usuario, con las tablas inserted y/o deleted, para realizar inserción/actualización de campos, recorra uno a uno los registros de las tablas inserted y/o deleted para la realización de dichas operaciones en el trigger.

7.1.5.4 Ejecución de procesos

7.1.5.4.1 Aspectos Básicos

1. Evite permitir que las aplicaciones de usuario realicen procesamiento batch. Defina para tal fin mecanismos asociados a dicho tipo de procesamiento.

2. Defina mecanismos de control, para llevar el registro de:

a. Tiempo de ejecución, inicio y fin de cada proceso.

b. Cantidad de registros involucrados en el ciclo principal del proceso.

c. Estado de ejecución (En ejecución, Finalizado Exitosamente, Finalizado con Error, Abortado, otros)
d. En términos generales, la mayor cantidad de métricas que se puedan almacenar de cada proceso, permitirá un mejor conocimiento y evaluación en el futuro del mismo.

3. Evite la creación de tablas fijas de usuario (o de vistas) dentro de procesos, utilice tablas temporales para los mismos fines. Si la base de datos es Replicada no se debe permitir la creación de objetos fuera de los procedimientos formales de catalogación.

4. Estructure procesos para ser ejecutados en la base de datos, por el dataserver con disparadores por medio de procedimientos almacenados. Evite que el control de flujo del proceso esté controlado por una aplicación externa, ya que se aumentara el tiempo de ejecución, debido a la interacción entre el dataserver y la aplicación cliente que controla el proceso (pudiendo incurrir en afectaciones adicionales al motor, bloqueos, tiempos de espera, etc).

5. Establezca métodos que permitan la detención de los procesos, en caso de requerirse.

6. Es recomendable estructurar los procesos, de manera que en caso de falla no se pierda el procesamiento realizado hasta el punto de falla. De manera que al retomarse el proceso, pueda retornar al mismo punto de procesamiento.

7. Establezca límites de ejecución:

a. Horarios en los cuales se puede ejecutar

b. Dependencias de otros procesos.

c. Cantidad de hilos de ejecución permitidos (si se establece ejecución en paralelo).

d. Ejecución en paralelo con otros procesos (cuáles son permitidos y cuáles no).

8. No incorpore sentencias de perfil administrativo, como mantenimientos (UPDATE STATISTICS o REORGS) o ejecución de DBCCs, ya que dichas sentencias deben ser ejecutadas en el ámbito adecuado, con la supervisión adecuada de un DBA.

9. Evite incorporar operaciones no logeadas3 .

10. En caso de requerir mover grandes cantidades de registros de una tabla a otra utilice BCP para facilitar la tarea.

7.1.5.4.2 Procesamiento Paralelo (en Hilos)

1. Asegure que cada hilo de proceso será único en los datos a procesar.

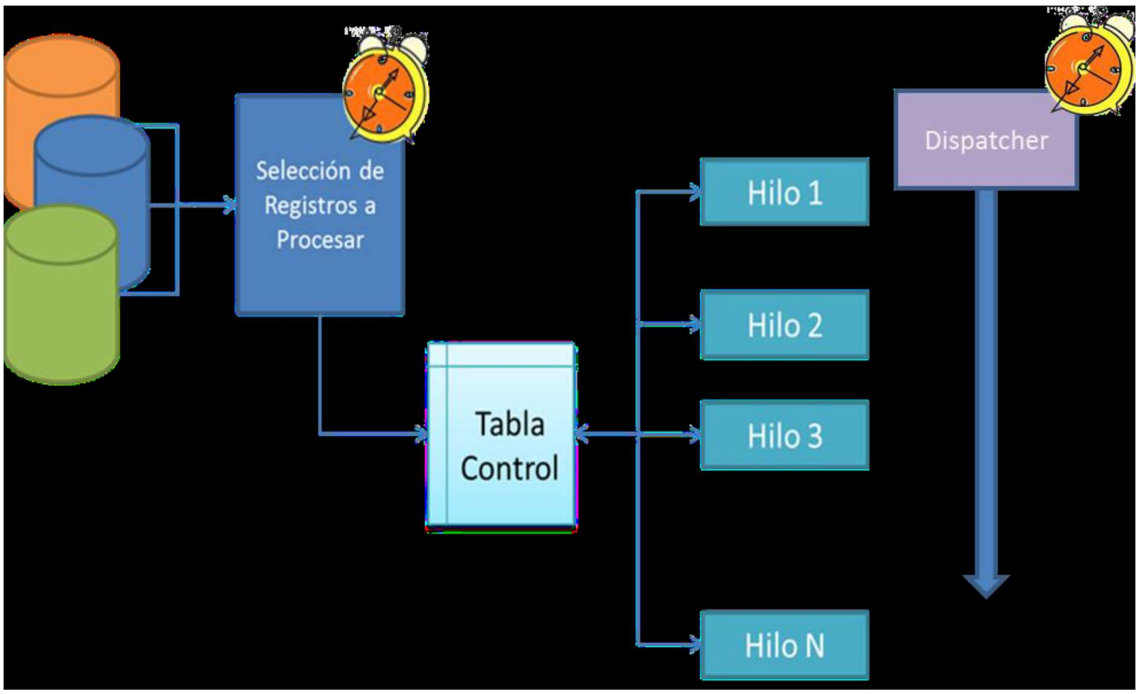
2. Establezca métodos que permitan la configuración de parámetros que determinen el comportamiento del proceso:

a. Cantidad de hilos que pueden estar en ejecución para un mismo proceso.

b. Tiempo máximo de ejecución, mediante este se podría indicar si se tiene una ventana máxima de tiempo definida para la actividad de un hilo.

c. Conclusión del proceso, este parámetro podría indicar al conjunto de hilos, si deben detenerse. Esto con el fin de que los hilos no sean cancelados de manera abrupta para detener el proceso.

3. Establezca una tabla de control donde: a. Cada proceso tome de manera única su bloque de proceso. b. Cada hilo registre, hora inicio, hora fin, registros, estado, etc. En general valores de control de cada proceso.



4. Establezca un único proceso padre, el cual se encargará de revisar si existen datos para procesar y alimentar la tabla de control, para indicar a los hilos de ejecución qué deben procesar.

5. Establezca un proceso despachador (dispatcher), que lance a ejecución los hilos del proceso y mantenga control sobre la cantidad y estado de estos.

6. El tiempo de vida de un hilo puede ser definido por:

a. El tiempo de ejecución determinado para cada proceso.

b. El procesamiento de todos los datos en la tabla de control, en cuyo caso el proceso padre no asigna nuevos datos para procesamiento.

c. La conclusión del proceso asignado en la tabla de control al hilo específico, en esta situación, el dispatcher debe lanzar un nuevo hilo tan pronto como observe que se cierra la sesión del anterior, con el fin de mantener la misma cantidad de hilos en ejecución.

7.1.6 Aspectos del cliente

7.1.6.1 Drivers

1. Utilice en las aplicaciones drivers que estén certificados para la versión SAP ASE con la cual se establecerá conexión.

2. En lo posible no utilice drivers anteriores a la versión de su dataserver.

3. Si desea utilizar drivers de otros proveedores, garantice el correcto funcionamiento de su aplicación.

4. En las cadenas de conexión indique los siguientes parámetros: a. El nombre de la aplicación (ApplicationName). b. Tamaño del paquete a utilizar (EnableServerPacketSize y PacketSize) c. Encadenamiento de transacciones inactivo (ServerInitiated Transactions en 0)

7.1.6.2 Aplicación

1. Determine una cota máxima de registros que puedan pasar a un cliente, en especial en funcionalidades de usuario que están asociadas a reportes.

2. Se recomienda que todo proceso asociado con los datos almacenados en el dataserver, se mantengan del lado del motor de base de datos, permitiendo:

a. Disminuir el flujo de control entre aplicación y dataserver.

b. Independizar las funciones propias de los datos, de la aplicación.

c. Optimiza el uso de recursos en el servidor del dataserver.

d. Aplicaciones más livianas. e. Disminuye tiempos de modificación y/o optimización.

3. No incluya ejecución de procesos o extracción de datos sobre grandes bloques de registros desde la aplicación cliente, para tal fin utilice otros mecanismos (orientados a ejecución de procesos en Batch).

4. Asegure que toda sesión de usuario sea cerrada:

a. Al cerrar la sesión del usuario en la aplicación.

b. Al terminar la operación realizada y no requerir nuevamente la conexión.

c. Al detener el servicio de la aplicación.

7.1.6.3 Red

1. Establezca el valor de Keep Alive de la red en una cota no inferior a 2 horas.

2. Evite tanto como sea posible, operaciones de tráfico de red durante la ejecución de procesos, ya que esto genera constantes intercambios de contexto del proceso, lo cual puede afectar negativamente su desempeño.

3. Amplié el tamaño de paquete en la configuración de la conexión (PacketSize).

8. 7.2 Mejores prácticas desarrollo SQL

(Tomado de https://msdn.microsoft.com/en-us/library/office/bb219479(v=office.12).aspx)

“El tema más importante en la elaboración de reportes o extracción de información de nuestra base de datos es el tiempo de procesamiento de las consultas que realizamos. Muchas veces, esto se debe al hardware, el software utilizado, el mal diseño de la base de datos, la mala formulación de índices y consultas. Por tal motivo, para mejorar este último punto desarrollaremos algunas consideraciones que nos permitirán minimizar el impacto que tiene las consultas en el tiempo de procesamiento de nuestra aplicación”.

T-SQL Crude vs Stored Procedure:

Lo recomendado es siempre utilizar las consultas dentro de motor de base de datos y no dentro de la aplicación.

Siempre realizar la consulta dentro de un procedimiento almacenado ya que éste se ejecuta más rápido que cualquier consulta externa fuera del motor de base de datos.

La primera vez que se ejecuta el procedimiento almacenado es compilado lo que produce un plan de ejecución que es un paso a paso de como el motor ejecutará las sentencias SQL. El plan es colocado en memoria (Cached) para su re-uso.

Otra ventajas:

Seguridad: no se le da acceso a objetos internos de BD.

Administración cualquier cambio se hace en el procedimiento no en la aplicación.

Tráfico de Red se reduce el tráfico porque se trabaja sobre el motor de BD.

Nunca se debe de utilizar el SELECT *:

No se debe usar **Select *** puesto que con esto el motor lee primero toda la estructura de la tabla antes de ejecutar la sentencia. Otra desventaja es que el resultado variará si se agrega o quitan campos.

Uso de JOIN, RIGTH JOIN y LEFT JOIN:

Usar JOIN, RIGTH JOIN y LEFT JOIN ya que mientras el motor de base de datos va leyendo las tablas va verificando que relación se necesita entre ellas y de este modo, éste lee menos registros y hace mas eficiente la consulta, a diferencia del WHERE que hace que las tablas se lean en su totalidad y después hace las relaciones.

Especificar a que tabla corresponde cada campo:

Si utilizas varias tablas en la consulta especifica siempre a que tabla pertenece cada campo, le ahorras al gestor el tiempo de localizar a que tabla pertenece el campo.

En lugar de:

SELECT Nombre, Factura FROM Clientes, Facturacion WHERE IdCliente=IdClienteFacturado,

Usar:

SELECT Clientes.Nombre, Facturacion.Factura WHERE Clientes.IdCliente = Facturacion.IdClienteFacturado.

Orden de las Tablas en las Consultas.

Cuando se realiza una consulta entre varias tablas el orden de estas debe ir de menor a mayor número de registros; dependiendo del numero de columnas y registros de la tabla se genera un resultado que puede llevar mucho tiempo para ser completado por parte del motor de base de datos.

Ej: Si deseamos saber cuantos alumnos se matricularon en el año 1996 y escribimos: FROM Alumnos, Matriculas WHERE Alumno.IdAlumno = Matriculas.IdAlumno AND Matriculas.Año = 1996 el gestor recorrerá todos los alumnos para buscar sus matriculas y devolver las correspondientes. Si escribimos FROM Matriculas, Alumnos WHERE Matriculas.Año = 1996 AND Matriculas.IdAlumno =

Alumnos.IdAlumnos, el gestor filtra las matrículas y después selecciona los alumnos, de esta forma tiene que recorrer menos registros.

Operadores en el filtro:

Si utilizamos WHERE, los operadores a utilizar deben ser los de mejor rendimiento. El orden de rendimiento de los operadores de mayor a menor es:

=, >,>=, <=, <, LIKE, IN, <>, NOT IN, NOT LIKE

Evitar el uso del comando **LIKE** con el siguiente wildcard '%R' ya que no permite el uso del indice si el campo lo tuviera, **LIKE** con el siguiente wildcard 'R%' permite el escanéo parcial y el uso del índice

El comando **BETWEEN** es más eficiente que el **IN** porque el primero busca un rango de valores y el segundo varios valores puntuales provocando que la sentencia sea menos efectiva.

Cuando utilicemos los operadores **AND** es preferible empezar por los campos que sean parte de algún índice para que la información seleccionada sea mas selectiva y la cantidad de registros leídos sea mínimo.

Cuando se tenga la necesidad de ordenar los datos en una consulta se recomienda que los campos a ordenar sea el menor número posible, y también se sugiere crear un índice de tipo **clustered index** sobre el campo que se esta utilizando.

No se recomienda el uso del **INSERT INTO** ya que esto origina que la tabla se bloquee mientras se esta llevando acabo el insert y de este modo se impide el uso del resto de datos a los usuarios.

Si dentro de una consulta se debe utilizar el comando **HAVING** se recomienda hacer todos los filtros posibles con el comando WHERE, de esta forma el trabajo que tiene que realizar el comando HAVING será el menor posible.

Se procurará elegir en la cláusula WHERE aquellos campos que formen parte del índice de la tabla

Además se especificarán en el mismo orden en el que estén definidos en la clave.

Filtrar siempre por campos que tengan índices.

Si deseamos preguntar por campos pertenecientes a índices compuestos es mejor utilizar todos los campos de todos los índices.

ANEXO 1: GESTION DE TRANSACCIONES EN SAP ASE

Los comandos "begin transaction" y "commit transaction" delimitan en SYBASE ASE una transacion. Por lo que definiríamos una transacción como una unidad de trabajo. El comando "rollback transaction" es utilizado para deshacer una transacción, bien sea hasta su comienzo bien sea hasta un pundo donde se ha salvado la transacción llamado "save point".

TRANSACCIONES ANIDADAS EN SAP ASE

Es posible anidar transacciones, sin embargo, este anidamiento es solo algo de sintaxis y de orden, pues solo la transacción más externa es la que controla el inicio y la finalización de toda la transacción.
Existe una variable global llamada *@@trancount* la cual se encarga de gestionar el nivel de anidamiento de las transacciones. Cada sentencia "*BEGIN TRANSACTION*" incrementa en 1 el valor de la variable, mientras que cada sentencia "*COMMIT TRANSACTION*" la decrementa en 1. Una transacción anidada no finaliza hasta que la variable @@trancount es igual a 0.
Ejemplo:

```
begin tran
select @@trancount -- @@trancount = 1
. . .
begin tran
select @@trancount -- @@trancount = 2
. . .
commit tran
select @@trancount -- @@trancount = 1
. . .
commit tran
select @@trancount -- @@trancount = 0
-- La transacción realmente finaliza con este commit
```

Cuando una transacción se deshace utilizando el comando *ROLLBACK TRANSACTION* sin incluir el nombre de un "save point", la transacción se deshace hasta el *BEGIN TRANSACTION* inicial. Sin embargo utilizando "*save points*" se puede deshacer solo parte de una transacción.

"SAVE POINTS" EN TRANSACCIONES

Se definen utilizando el comando "*SAVE TRANSACTION*".

Una vez definido un "*save point*" una transacción podrá deshacerse utilizando:

```
ROLLBACK TRAN <NOMBRE DE TRANSACCION>
ROLLBACK TRAN <NOMBRE DE SAVE POINT>
```

En el siguiente ejemplo se ilustra el uso de "*save points*"

```
begin tran A
select @@trancount /* @@trancount = 1 */
begin tran B
select @@trancount /* @@trancount = 2 */
....
save tran B1
....
if @@error != 0
rollback tran B1
select @@trancount /* @@trancount = 2 */
....
commit tran B
```

```
select @@trancount /* @@trancount = 1 */
....
rollback tran A
....
commit tran A
select @@trancount /* @@trancount = 0 */
```

El primer *rollback tran* deshace la transacción hasta el "*save point*" B1.
Si una transacción se deshace hasta un "*save point*", esta debe continuar y completarse como *COMMIT TRAN* o deshacerse en su totalidad con *ROLLBACK TRAN*.
El segundo *rollback* deshace la transacción completa.
Si se utilizan nombres de transacciones y "*save points*" estos deberán ser diferentes.

SELECT USANDO HOLDLOCKS Y TRANSACCIONES

Las sentencias *SELECT* utilizadas para recuperar datos establecen un bloqueo compartido sobre las páginas que están siendo recuperadas.

```
begin tran
select org_Id, short_Name
from Org_Table
where short_Name like "A%"

/* En este punto se liberan los shared locks sobre la tabla
Org_Table */
…..
commit tran
go
```

Aunque esta sentencia *SELECT* se encuentre dentro de una transacción, estos bloqueos compartidos son liberados inmediatamente cuando el dato es accedido, con lo que otros procesos pueden acceder a la información de dichas páginas, tanto para consulta como para actualización (siempre que se esté utilizando un *isolation level 1*).
Sin embargo si en la sentencia *SELECT* se utiliza la cláusula "*HOLDLOCK*", los bloqueos compartidos no son liberados hasta que la transacción se completa bien sea con "*COMMIT TRANSACTION*" o con "*ROLLBACK TRAN*".

```
begin tran
select Org_Id, Short_Name
from Org Holdlock
where short_Name like "A%"
…..
commit tran /* en este punto son liberados los shared locks sobre
la tabla */
```

Esto hace que mientras se mantienen estos bloqueos ningún otro proceso pueda cambiar las páginas que están siendo recuperadas por la sentencia *SELECT*, aunque si pueda accederlas en las búsquedas. Son muy pocas las ocasiones en las que nos interesa mantener este bloqueo sobre los datos recuperados, demostrándose estrictamente necesario solo en casos en los que se realiza un análisis exhaustivo de la información recuperada y es absolutamente necesario garantizar que ningún otro proceso ha cambiado la información accedida.
Tal es el caso conocido como "*Inconsistent Analysis Problem*" en el que los datos recuperados se utilizan para realizar labores de análisis tales como medias, desviaciones, etc, y en cuyos casos, los resultados pueden verse desvirtuados si no se garantiza la fiabilidad de los datos utilizados.

TAMAÑO DE LAS TRANSACCIONES

Es importante mantener todas las instrucciones T-SQL de una transacción dentro del menor número posible de paquetes de red, para evitar que los problemas de red puedan afectar a la duración de la misma.

Como todos sabemos, la comunicación entre los programas Cliente y el motor de Base de Datos se realiza utilizando siempre una red.

Los comandos SQL construidos en el cliente viajan por la red utilizando paquetes que deben llegar al motor de Base de Datos, ser leídos, interpretados y ejecutados. Cuanto mayor sea el texto SQL que debe viajar por la red desde el cliente hasta el motor de Base de Datos, mayor será el impacto que un mal funcionamiento de la red pueda tener en el tiempo de respuesta percibido por el programa cliente, desde que este realiza la petición al motor, hasta que se recibe la respuesta. Además, si la transacción es abierta por el cliente, todos los recursos que sea necesario bloquear dentro de esta transacción, se mantendrán bloqueados por más tiempo.
Por ejemplo, si tenemos un programa cliente que realiza la siguiente operación:

```
<<< CLIENTE SAP ASE>>>
Begin Tran
—>>> Red
/* Inicio de la transacción*/
update tabla
set campo = valor
where campo_id = valor_id
->>> Red
/* Bloqueo sobre página de "tabla" */
Commit Tran
->>> Red
/ * Fin de la Transacción */
```

Vemos que en el tiempo en que permanece abierta la transacción influye el tiempo de comunicación entre el cliente y el motor de Base de Datos a través de la red.
Este tiempo puede minimizarse, e incluso eliminarse si el protocolo de comunicación entre el Cliente y el motor de Base de Datos se utiliza un procedimiento almacenado.
El ejemplo anterior puede reescribirse de la siguiente manera:

```
<<< CLIENTE SQL SERVER>>>
Exec myproc
->>> RED
myproc

begin tran
update tabla
set campo = valor
where campo_id = valor_id
commit tran
return
```

También se debe eliminar completamente la interacción con el usuario dentro de una transacción, de tal manera que el mantenimiento de un bloqueo no dependa del comportamiento de un cliente

```
begin Tran
->>> RED
/* Inicio de la transacción*/
update tabla
set campo = valor
where campo_id = valor_id
/* Bloqueo sobre página de "tabla" "read input"
Espera por un valor que introduzca el usuario */
commit tran
->>> RED
/ * Fin de la Transacción */
```

Se observa que durante la duración de la transacción y en el tiempo que permanecen bloqueados los recursos, tales como páginas de tablas, influye directamente la acción de un usuario que puede decidir, por ejemplo, realizar cualquier otra actividad antes de introducir el valor que se requiere.

SELECT INTO VS INSERT INTO SELECT

Las instrucciones *SELECT INTO* e *INSERT INTO SELECT* sirven para cargar datos en una tabla, procedente del resultado de una consulta. Sin embargo, existen diferencias fundamentales entre las mismas:

- La instrucción *SELECT INTO* requiere que no exista la tabla mencionada en la cláusula *INTO* ya que esta será creada cuando se ejecute la consulta, mientras que la instrucción *INSERT INTO SELECT* se utiliza para guardar valores en una tabla ya existente.
- Las operaciones de inserción llevadas a cabo utilizando la instrucción *SELECT INTO* no quedan registradas en el log de transacciones de la base de datos, por lo que se ejecutan con mayor rapidez que si utilizásemos la instrucción *INSERT INTO SELECT*, cuyas inserciones si quedan registradas en el log de transacciones. Por esta razón la instrucción *SELECT INTO* no puede ser utilizada dentro de una transacción. Para poder utilizar la instrucción *SELECT INTO* es necesario que la base de datos tenga habilitada la opción "*select into/bulkcopy*".

TRUNCATE VS DELETE

Siempre que sea necesario borrar todos los registros de una tabla tenemos el uso de dos mecanismos para hacerlo.

Las instrucciones *TRUNCATE TABLE <TABLA>* y *DELETE <TABLA>*.

Las diferencias en la utilización de ambas instrucciones se resumen en:

- La instrucción *TRUNCATE TABLE <TABLA>* es una operación no registrada en el log de transacciones y por tanto se ejecuta con mayor rapidez que la operación *DELETE*, cuyos borrados si quedan registrados en el log de transacciones. Debido a que es una operación no registrada en el log, la instrucción *TRUNCATE TABLE* no puede ser utilizada dentro de una transacción.
- Solo el propietario de la tabla puede ejecutar la instrucción *TRUNCATE <TABLA>* y el permiso para realizar esta operación no puede ser transferido a otro usuario, mientras que para poder borrar datos de una tabla utilizando la instrucción *DELETE <TABLA>* solo es necesario que el usuario que ejecuta la instrucción haya recibido permiso de borrado sobre dicha tabla.

Como recomendación general, siempre que tengamos que borrar todas las filas de una tabla, y siempre que sea posible, desde el punto de vista de rendimiento será mejor utilizar la instrucción *TRUNCATE TABLE* frente a *DELETE*.

ANEXO 2: ÍNDICES CLUSTERED Y NON-CLUSTERED

La creación de índices en una base de datos relacional es una tarea fundamental que permite agilizar la recuperación de los datos y mejorar el rendimiento de las aplicaciones que ejecutan estos accesos.

En SAP ASE, una tabla es físicamente una cadena de páginas que contienen las filas de datos. A su vez un índice es físicamente una estructura *B-Tree* de páginas en las que aparecen ordenados los valores de la clave definida. Además de la clave del índice estas páginas contienen punteros a las páginas de datos. Por lo tanto el acceso a través de un *B-Tree* permite recuperar más rápidamente los datos.

En caso de no utilizar un índice, el acceso a cualquier fila de una tabla significara acceder a todas las páginas de esa tabla (conocido como *Table Scan*).

Los tipos de índices existentes son:

- Clustered Index
- Non-Clustered Index

El índice Clustered de una tabla es el que establece un orden físico de las filas en las páginas de los datos, es decir, en las páginas ocupadas por esta tabla. Por lo tanto cada vez que cambiamos (borramos y creamos) un índice clustered, estamos cambiando físicamente la ubicación de las filas en las páginas de datos. Por definición, solo puede crearse un único índice Clustered para la tabla, porque evidentemente solo puede haber un orden físico de las filas. Típicamente el índice Clustered de una tabla es su clave primaria, pero no necesariamente tiene que ser así.
La sentencia SQL de creación del índice debe indicar explícitamente esta característica si se desea que se clustered:

```
create <unique> clustered index <nombre_indice>
on <nombre_tabla>(campos…)
```

Un índice Non-Clustered no establece ningún cambio en el orden físico de las filas en las páginas de datos. El nivel "*leaf*" del *B-Tree* de un índice Non-Clustered está formado por páginas que contienen las claves ordenadas y un puntero a la página de datos donde se encuentra su fila. En otras palabras, un índice Non-Clustered necesita un nivel más en su *B-Tree* para acceder a las filas de las páginas de datos. Si no se indica nada en la sentencia SQL de creación del índice, el valor por defecto es Non-Clustered:

```
create <unique> index <nombre_indice>
on <nombre_tabla>(campos…)

create <unique> non-clustered index <nombre_indice>
on <nombre_tabla>(campos…)
```

El número máximo de indicies Non-Clustered que se pueden crear sobre una tabla es 249.

CUANDO USAR CLUSTERED O NON-CLUSTERED?

Los candidatos a ser índices Clustered serán:

- Lo fundamental es que sea una llave que es ingresada siempre de manera incremental de acuerdo al índice (nunca hay inserciones en lugares intermedios).
- Preferiblemente, llave que forma la clave primaria de la tabla, aunque no es obligatorio.
- Llave que podría ser accedida por rangos
- Posible uso en cláusulas order by, group by y joins.

En caso de que el índice no cumpla con las anteriores condiciones debe ser construido como Non-Clustered

ANEXO 3: OPTIMIZACIÓN DE UNA SENTENCIA

El optimizador es un componente del sistema gestor de base de datos que se encarga de determinar cómo se va a resolver una consulta en base a una serie de algoritmos. Para ver qué plan va a utilizar el optimizador en la resolución de una consulta es necesario activar las siguientes opciones:

```
use <base de datos>
go
set showplan on
go
```

```
set noexec on
go
<sentencia select>
```

***set showplan*:** Muestra como el optimizador va a resolver la consulta, que índices va a utilizar, si se trata de un Table Scan etc..
***set noexec*:** No devuelve resultados de la ejecución del select.
La combinación de ambas opciones, *set showplan on* y *set noexec on* es una buena forma de optimizar una sentencia sin tener que ejecutarla y cargar la máquina. Una vez que consideramos haber optimizado la sentencia o que tengamos varias sentencias candidatas se podría activar la opción *set statistics io on* y *set statistics time*, es necesario apagar la opción *set noexec off* y verificar el número de páginas lógicas, físicas leídas y el tiempo que tarda en ejecutarse.

UPDATES IN PLACE

Siempre que sea posible y nuestro modelo de datos lo permita, debemos utilizar la funcionalidad existente en SAP ASE conocida como "*updates in place*". Para poder realizar este tipo de actualizaciones deben cumplirse las siguientes condiciones:

- La tabla no debe tener un *trigger* de actualización
- Las columnas que van a ser actualizadas no pueden ser de longitud variable ni admitir nulos
- Las columnas que van a ser actualizadas no pueden formar parte del índice que el optimizador vaya a utilizar en el *query plan*.
- El *update* debe afectar solo una fila .

Si todas estas condiciones se cumplen, la actualización es realizada físicamente en el mismo lugar donde están los datos originales y no realizando primero un borrado y luego una inserción como en un proceso normal de actualización.

CONDICIÓN MAYOR QUE

Asumiendo que la tabla "*table*" tiene un índice por el campo "*int_col*" la consulta:

```
select * from table where int_col > 3
```

El optimizador utiliza el índice para buscar el primer valor que cumpla con la condición *int_col = 3* y después realiza un scan buscando el primer valor mayor que 3. Si hay muchos registros que cumplan con la condición *int_col = 3*, el motor de base de datos deberá realizar un scan de todas estas páginas hasta encontrar el primer registro en el que *int_col* es mayor que 3.
Probablemente la siguiente consulta es más eficiente si se escribe como:

```
select * from table where int_col >= 4
```

NOT EXISTS Y NOT IN

En subqueries y sentencias if, las búsquedas por criterios "*IN*" y "*EXISTS*" dan un mayor rendimiento que "*NOT EXISTS*" y "*NOT IN*" cuando los valores en la cláusula *WHERE* no están indexados.
Para búsquedas por "*IN*" y "*EXISTS*" SAP ASE puede retornar tan pronto como encuentra una fila que cumple el criterio especificado, sin embargo para el caso de "*NOT EXISTS*" o "*NOT IN*" debe examinar todos los valores para determinar si existen o no datos. Por ejemplo:

```
if not exists(select * from table where …)
begin
/* Sentencias A */
end
else
begin
/* Sentencias B */
```

```
end
```
se obtendrá un mejor rendimiento de la siguiente manera:
```
if exists(select * from table where …)
begin
/* Sentencias B */
end
else
begin
/* Sentencias A */
end
```

Incluso si no se tiene tratamiento para la cláusula else de la sentencia if, sentencias como la siguiente:
```
if not exists(select * from table where …)
begin
/* Sentencias A*/
end
```
Pueden reescribirse apoyándonos en la sentencia *GOTO* así:
```
if exists(select * from table where …)
begin
goto etiqueta
end
/* Sentencias A */
etiqueta:
```

COUNT Vs EXISTS

No debemos utilizar la función *COUNT* para chequear una existencia.
Por ejemplo:
```
select * from tabla
where 0 < ( select count(*) from tabla2 where …)
```
Puede reescribirse como:
```
select * from tabla
where exists( select * from tabla2 where …)
```
Cuando se utiliza la función *COUNT()* el optimizador de SAP ASE no es capaz de determinar que se está chequeando una existencia, por lo que realiza un conteo de todos los valores que cumplan el criterio de búsqueda, bien realizando un "*Table Scan*" o utilizando un índice si este existe.
Si se utiliza la función *EXISTS()*, el optimizador sabe que se está chequeando un existencia, por lo que detiene la búsqueda cuando encuentra el primer valor que cumpla con el criterio.

CLAUSULAS OR Vs UNIONS EN JOINS

En general "joins" unidos mediante el operador "*OR*" pueden presentar inconvenientes de rendimiento, sin embargo, se puede solventar mediante el uso de la cláusula "*UNION*", ya que optimiza separadamente cada una de las sentencias *SELECT* de los querys. Por ejemplo la consulta:
```
select * from tabla1, tabla2
where tabla1.a = tabla2.b
or tabla1.x = tabla2.y
go
```

Puede reescribirse en la mayoría de los casos como:

```
select * from tabla1, tabla2
where tabla1.a = tabla2.b
unión
select * from tabla1, tabla2
where tabla1.x = tabla2.y
go
```

FUNCIONES AGREGADAS MAX Y MIN

SAP ASE utiliza una optimización especial para las funciones agregadas *MAX()* y *MIN()*, cuando existe un índice por la columna agregada.

Para la función *MIN()* se lee el primer valor en la página raíz del índice.

Para la función *MAX()* va directamente al final del índice para buscar la última fila.

Esta optimización es siempre aplicable excepto si:

 La expresión utilizada en la función *MAX()* o *MIN()* no es una columna: Supongamos que la columna "*col*" tiene un índice Non-Clustered: Por ejemplo:

```
select max(col * 2) from tabla
```

No utilizara la optimización por índice mientras que

```
select max(col) * 2
```

Si lo utilizara

 La columna utilizada en la función agregada *MAX()* o *MIN()* no es la primera columna de un índice. En este caso si el índice es non-clustered podrá encontrar el valor máximo o mínimo en las últimas páginas del índice, mientras que si el índice es un índice Clustered deberá realizar un "*Table Scan*".

 El uso de funciones agregadas de manera concurrente en la misma sentencia. Un ejemplo de este tipo de optimización seria reescribir la consulta:

```
Select max(precio), min(precio) from Tabla
```

Como:

```
select max(precio) from Tabla
select min(precio) from Tabla
```

JOINS Y TIPOS DE DATOS

Cuando se realiza un "join" entre dos columnas con diferentes tipos de datos, el tipo de datos de una de las columnas deberá ser convertido al tipo de datos de la otra columna. El criterio seguido por el optimizador para realizar esta conversión es convertir el tipo de menor jerarquía al tipo de mayor jerarquía.

Si se realiza el "join" entre tablas con tipos de datos incompatibles, el optimizador de SAP ASE utilizara el índice por el campo que no realiza la conversión, pero no por el que tiene que ser convertido. Por ejemplo:

```
select * from tabla_pequena, tabla_grande
where tabla_pequena.columna_float = tabla_grande.columna_int
```

En este caso, el optimizador convierte la columna de tipo entero *int* a *float*, ya que *int* es de menor jerarquía que *float*. Esto ocasiona que no se utilice el índice por el campo *tabla_grande.columna_int*.

Lo mismo ocurriría para el caso:

```
select * from tabla_pequena, tabla_grande
where tabla_pequena.columna_char = tabla_grande.columna_varchar
```

La columna *varchar* va a ser convertida a tipo *char*, por lo que no se utilizara el índice por esta columna. En las columnas de tipo carácter esto puede ser evitado sencillamente reescribiendo la consulta como:

```
Select * from tabla_pequena, tabla_grande
Where convert(varchar(#), tabla_pequena.columna_char) =
tabla_grande.columna_varchar
```

Donde # es el tamaño de la columna *tabla_grande.columna_varchar*.
La mejor recomendación a este respecto es procurar en tiempo de diseño que las columnas comunes a varias tablas tengan los mismos tipos de datos. Recordar que a todos los efectos un campo *char* o *binary* que admite nulos se comporta como *varchar* o *varbinary* respectivamente.

PARÁMETROS Y TIPOS DE DATOS

Si un parámetro de un procedimiento almacenado no es del mismo tipo de datos que la columna con la que es comparado en una clausula *WHERE*, el optimizador obliga a la conversión, con el impacto en la utilización de índices descrito en el punto anterior.
Por ejemplo:

```
create proc proc1( @parametro1 varchar(30) )
as
select * from tabla
where columna_char = @parametro1
```

Puede reescribirse como:

```
create proc proc1( @parametro1 char(30) )
as
select * from tabla
where columna_char = @parametro1
```

Para obtener un mejor rendimiento.

USO DE CURSORES

Un cursor es un nombre simbólico asociado a una sentencia *SELECT* que permite el acceso fila a fila al resultado de la consulta. Aunque el lenguaje T-SQL está concebido para un procesamiento orientado a conjunto de resultados, SAP ASE a partir de su versión 10.0 implementa el mecanismo de cursores para proporcionar tratamiento a nivel de fila, por compatibilidad con el estándar SQL ANSI-89.
Sin embargo, para el uso de cursores en procedimientos se debe ser extremadamente cuidadosos ya que se puede afectar negativamente el rendimiento de un sistema totalmente orientado al proceso de conjunto de resultados.
Los recursos necesarios en cada una de las etapas de creación y utilización de un cursor son los siguientes:
DECLARE: Cuando se declara un cursor se reserva memoria de SAP ASE para el cursor y su plan de ejecución
OPEN: Cuando se abre el cursor se procesa la consulta asociada al mismo y si bien no se accede todavía a las filas resultantes de la consulta, si se establecen los "*intent-locks*" a nivel de tabla y si en la consulta existen sub-queries, suceden los bloqueos a nivel de página.