LINEAMIENTOS DE CODIFICACION DE SOFTWARE

Gestión del Ciclo de Vida de Software
Gestión de la Implementación y Entrega del Servicio TI

1. OBJETIVO

Establecer los lineamientos necesarios para el estándar de codificación de las aplicaciones implementadas por las fábricas de desarrollo contratadas por Colpensiones y desarrollos Internos.

2. ALCANCE

Este documento se realiza con el fin de que el lector tenga en conocimiento los lineamientos de cómo 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.

3. DESCRIPCIÓN

- Base de datos: Programa capaz de almacenar gran cantidad de datos, relacionados y estructurados, que pueden ser consultados rápidamente de acuerdo con las características selectivas que se deseen.
- Software: Conjunto de programas y rutinas que permiten a la computadora realizar determinadas tareas.
- Sentencia: elementos básicos en los que se divide el código en un lenguaje de programación.

4. DESCRIPCIÓN DEL DOCUMENTO

4.1 INTRODUCCIÓN AL ESTANDAR DE CODIFICACIÓN.

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.

4.2 CONTROL.

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

4.3 NOMENCLATURA.

En general, se recomeinda 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 está en mayúscula. Ejemplo: **EjemploDeComoEs**
- lowerCamelCase, igual que la anterior con la excepción de que la primera letra es minúscula. Ejemplo: **ejemploDeComoEs**

4.3.1 Base De Datos.

a. Bases de datos

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

b. 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**

c. 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.

d. 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**

e. 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.

f. 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.

g. 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

4.3.2 Aplicaciones de Software Orientado a Objetos.

a. 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

b. 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…..
}

c. 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…..
}

d. Métodos

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

Ejemplo: **obtenerCedula ();**

4.3.3 Variables y Constantes.

a. 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.

b. 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.

Ejemplos:

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;
```

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_INCONSISTENTES, p_subject);

Contexts.getConversationContext().set(ConstantesMediosMagneticos.MENSAJE_
ERROR_REGISTROS_INCONSISTENTES, p_message);
```

```
Contexts.getConversationContext().set(ConstantesMediosMagneticos.FUENTE_E
RROR_REGISTRO_INCONSISTENTES, 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.DESTINAT
ARIO_ERROR_REGISTRO_INCONSISTENTES,p_to);
      v_renderer.render(p_facelet);
   }
```

4.3.4 Convenciones de Programación Java

a. 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));
```

b. 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;

c. 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);
}
```

d. 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;
```

e. 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;
```

f. 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);
```

g. 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.

h. 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.

i. 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;
}
```

j. 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();
```

k. 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.

4.4 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.

4.4.1 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 está 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.

a. 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
```

```
*/
```

b. 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){
    ...
}
```

c. 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 está 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
*/
```

4.4.2 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.".

4.5 ESTRUCTURA DIRECTORIOS DESARROLLOS.

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

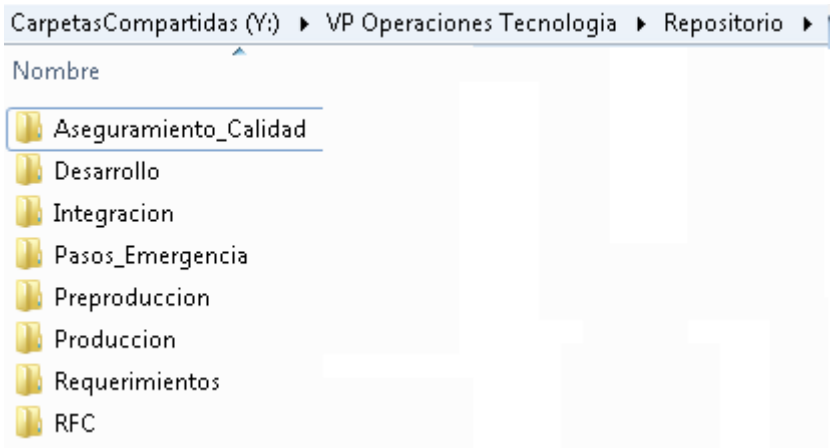


Imagen 1 Carpetas de Repositorio.

La ruta de las carpetas de repositorio es:
Y:\VP Operaciones Tecnologia\Repositorio; Allí se encuentran las carpetas para desarrollo, integración, aseguramiento de calidad, Preproducción, Producción 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, Preproducción

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

Integración, Preproducción, Producción y Pasos de emergencia.

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

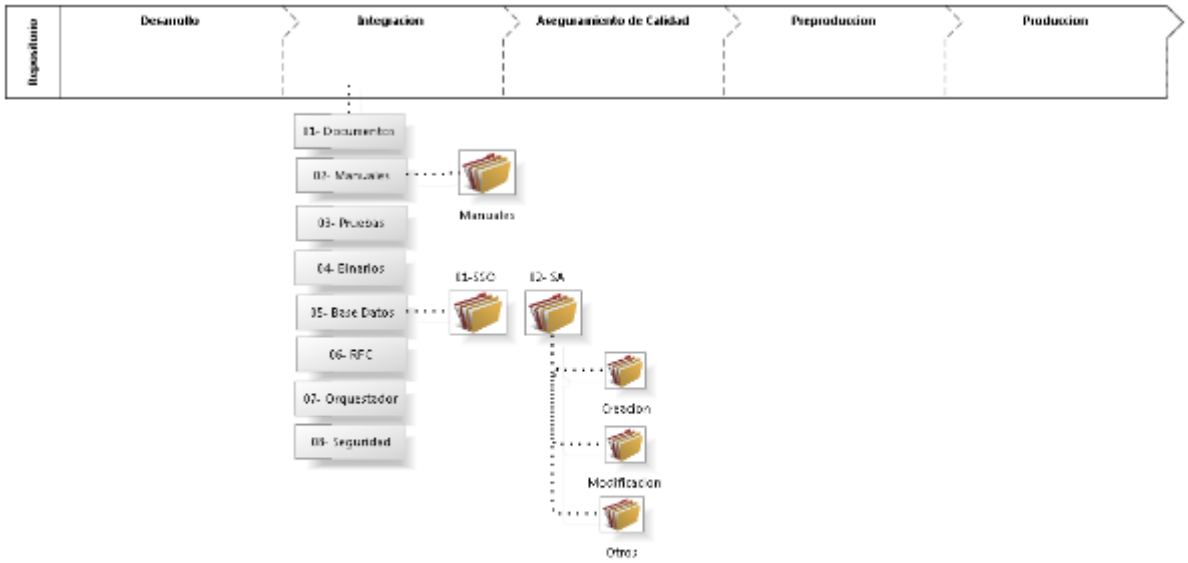


Imagen 2. Estructuras de carpetas.

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

4.5.1 - 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 más de uno se deberá mantener el orden de numeración e indexación.

4.5.2 - Manuales:

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

1-Manual_Instalacion.PDF, si existe más 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 más 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 más de uno se deberá mantener el orden de numeración, indexación (ejemplo; 3-Manual_Usuario1.PDF,3-Manual_Usuario2.PDF)

4.5.3- 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 deben estar en PDF y debe mantener la numeración y codificación que tiene el proceso Gestión de SQA en el SIG.

4.5.4 binarios:

Se debe dejar los archivos binarios que se requiere desplegar.

4.5.5 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:

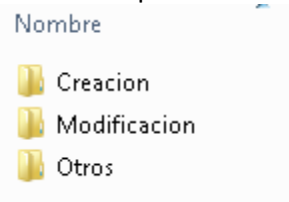


Imagen 3. Sub carpetas de Carpeta 02.SA.

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

4.6 MEJORES PRACTICAS DESARROLLO BASES DE DATOS.

4.6.1 Mejores prácticas desarrollo Sybase.

A. Tablas.

- Asigne nombres a las tablas que puedan identificar cada objeto de manera única dentro de cada dataserver.
- 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.
- 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.
- 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.
- Utilice particionamiento en tablas donde se conjuguen las siguientes situaciones: a. altos niveles de inserción de datos y b. concurrencia de sesiones de usuario.

B. Indices.

- Evite duplicar índices (mismos campos, en el mismo orden) en la misma tabla.
- 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.
- 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.
- 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.
- No incluya demasiadas columnas en un índice. En lo posible no incluya columnas que puedan ser modificadas de manera frecuente.
- 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).
- 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.
- 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).
- 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)
- Contemple los siguientes requerimientos para definir si un índice debe ser clustered:
- 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. c. Llave que podría ser accedida por rangos d. Posible uso en clausulas order by, group by y joins.
- 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.

C. Tablas temporales.

- 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:
  - Poseen menores capacidades de almacenamiento que las bases de datos de usuario (por lo general son bases de datos más pequeñas).
  - Son volátiles, por tanto grandes esfuerzos de I/O, escribiendo grandes cantidades de registros podrían perderse en caso de una falla.

- 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.
- 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.
- 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.

D. Sentencias.

a. Construcción:

- Límite las sentencias SELECT a los campos requeridos. Evite seleccionar columnas que no serán utilizadas.
- Asegure que en los filtros, los campos son del mismo tipo y precisión que las variables utilizadas (@)

- Asegure que en consultas que vinculen N tablas (donde N ≥ 2), existan N-1 joins que conjuguen todas las tablas, como mínimo1.
- Asegure que en los joins, los campos de intersección son del mismo tipo y precisión.
- Evite el forzamiento de índices.
- 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)
- 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.
- 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.
- Optimice el uso de funciones. Ejemplos:
    - No utilice la función isnull() sobre campos que no permiten valores nulos.
    - 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)
    - En general, el uso adecuado de funciones, reducirá el tiempo requerido en la resolución de una consulta.
- No realice validaciones de existencia de registros mediante el uso de count(), en dichos casos utilice la validación EXISTS.
- Utilizar todos los argumentos de búsqueda que pueda dar el optimizador tanto como sea posible para trabajar.
- 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.
- 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.

E. 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.

- Utilice set option show_missing_stats, para determinar posibles campos (o conjunción de campos), candidatos para construcción de índices.
- 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.
- 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:
    - Incrementa las posibilidades de que se presenten bloqueos y que estos tomen mayor tiempo.
    - Incrementa el log requerido y por ende el tamaño de los backups transaccionales.
    - Las operaciones de rollback pueden tomar tanto tiempo, como tomo la operación de modificación.

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

- 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.
- En particular, evitar las transacciones que deben esperar para la interacción con el usuario mientras mantiene bloqueos.
- 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 perdida de conexión puedan dejar transacciones abiertas (bloqueando los objetos implicados en la transacción), para esto:
- 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.
- 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.
- Actualice todos los campos necesarios en un registro, en un único acceso.
- Procure que las actualizaciones se realicen bajo el método direct updates, ya que utilizan menor número de candados y son más eficientes.
- 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.

F. Objetos procedimentales.

a. Cursores:

- En la declaración del cursor defina su finalidad:
  - Para modificaciones FOR UPDATE (de manera explícita señalar las columnas a modificar).
  - Para solo lectura FOR READ ONLY 2. Use UNION o UNION ALL en vez de cláusulas OR o listas IN.
- 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.

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

b. Procedimientos.

- Procure realizar la menor cantidad de lecturas del mismo registro dentro de una misma operación.
- 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.
- 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
```

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

- Indente el código de manera adecuada para facilitar su lectura.
- Defina prefijos diferentes, que permitan identificar la finalidad de cada variable:
- parámetros de entrada: variables utilizadas para lanzar el proceso.
    - parámetros de salida: variables que son retornadas con datos.
    - variables de trabajo: variables internas al procedimiento
- Añada comentarios que permitan documentar el flujo del proceso dentro del procedimiento.

G. Triggers.

- Haga uso de estos únicamente para:
    - Operaciones de validación de datos en registros (validaciones que no incluya largos procesos).
    - Validación de integridades referenciales o borradas en cascada.
    - Para registrar en auditorias datos sobre la operación realizada.
- 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:
    - Cree procedimientos almacenados que reciban como parámetros los valores de los registros implicados en la operación.
    - 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.
    - En un bloque transaccional, aplique todas las modificaciones requeridas en todas las tablas implicadas.
- 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.

H. Ejecución de procesos.

a. Aspectos Básicos

- Evite permitir que las aplicaciones de usuario realicen procesamiento batch. Defina para tal fin mecanismos asociados a dicho tipo de procesamiento.
- Defina mecanismos de control, para llevar el registro de:
    - Tiempo de ejecución, inicio y fin de cada proceso.
    - Cantidad de registros involucrados en el ciclo principal del proceso.
    - 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.

- 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.
- 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).
- Establezca métodos que permitan la detención de los procesos, en caso de requerirse.
- 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.
- Establezca límites de ejecución:
    - Horarios en los cuales se puede ejecutar
    - Dependencias de otros procesos.
    - Cantidad de hilos de ejecución permitidos (si se establece ejecución en paralelo).
    - Ejecución en paralelo con otros procesos (cuáles son permitidos y cuáles no).
- 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.
- Evite incorporar operaciones no logeadas3.
- En caso de requerir mover grandes cantidades de registros de una tabla a otra utilice BCP para facilitar la tarea.

I. Procesamiento Paralelo (en Hilos).

- Asegure que cada hilo de proceso será único en los datos a procesar.
- Establezca métodos que permitan la configuración de parámetros que determinen el comportamiento del proceso:
    - Cantidad de hilos que pueden estar en ejecución para un mismo proceso.
    - Tiempo máximo de ejecución, mediante este se podría indicar si se tiene una ventana máxima de tiempo definida para la actividad de un hilo.
    - 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.
- Establezca una tabla de control donde:
    - Cada proceso tome de manera única su bloque de proceso.

- Cada hilo registre, hora inicio, hora fin, registros, estado, etc. En general valores de control de cada proceso.

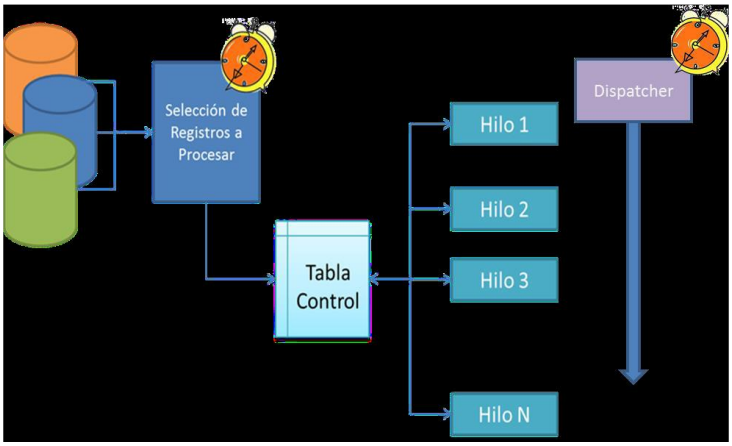


Imagen 3. Procesamiento Hilos.

- 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.
- Establezca un proceso despachador (dispatcher), que lance a ejecución los hilos del proceso y mantenga control sobre la cantidad y estado de estos.
- El tiempo de vida de un hilo puede ser definido por:
  - El tiempo de ejecución determinado para cada proceso.
  - El procesamiento de todos los datos en la tabla de control, en cuyo caso el proceso padre no asigna nuevos datos para procesamiento.
  - 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.

J. Aspectos del cliente.

a. Drivers

- Utilice en las aplicaciones drivers que estén certificados para la versión SAP ASE con la cual se establecerá conexión.
- En lo posible no utilice drivers anteriores a la versión de su dataserver.
- Si desea utilizar drivers de otros proveedores, garantice el correcto funcionamiento de su aplicación.
- 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).

K. Aplicación.

- Determine una cota máxima de registros que puedan pasar a un cliente, en especial en funcionalidades de usuario que están asociadas a reportes.
- Se recomienda que todo proceso asociado con los datos almacenados en el dataserver, se mantengan del lado del motor de base de datos, permitiendo:
  - Disminuir el flujo de control entre aplicación y dataserver.
  - Independizar las funciones propias de los datos, de la aplicación.
  - Optimiza el uso de recursos en el servidor del dataserver.
  - Aplicaciones más livianas. e. Disminuye tiempos de modificación y/o optimización.
- 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).
- Asegure que toda sesión de usuario sea cerrada:
  - Al cerrar la sesión del usuario en la aplicación.
  - Al terminar la operación realizada y no requerir nuevamente la conexión.
  - Al detener el servicio de la aplicación.

L. Red.

- Establezca el valor de Keep Alive de la red en una cota no inferior a 2 horas.
- 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.
- Amplié el tamaño de paquete en la configuración de la conexión (PacketSize).

4.7 Mejores prácticas desarrollo SQL.

"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"[1].

4.7.1 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.

[1] https://msdn.microsoft.com/en-us/library/office/bb219479(v=office.12).aspx

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.

4.7.2 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.

a. 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.

b. 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.

c. Especificar a qué 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.

4.7.3 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 número 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 cuántos 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 matrículas 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.

4.7.4 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 más 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 está utilizando.

No se recomienda el uso del **INSERT INTO** ya que esto origina que la tabla se bloquee mientras se está llevando a cabo 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.

4.8 GESTIÓN DE TRANSACCIONES EN SAP ASE.

Los comandos "begin transaction" y "commit transaction" delimitan en SYBASE ASE una transa cion. 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 punto donde se ha salvado la transacción llamado "save point".

4.8.1 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.

4.8.2 "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.

4.8.3 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.

4.8.4 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.

4.8.5 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*".

4.8.6 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*.

4.9 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.

4.9.1 ¿Cuándo 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

4.10 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.

4.10.1 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.

4.10.2 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
```

4.10.3 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:
```

4.10.4 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.

4.10.5 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
```

4.10.6 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
```

4.10.7 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.

4.10.8 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.

4.10.9 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-querie

4.11 LENGUAJE ABAP Y JAVA PARA SAP.

Contar con un marco referencial para uniformizar la nomenclatura a utilizar en los desarrollos ABAP, permitiendo una identificación rápida, precisa y oportuna durante la etapa de desarrollo y mantenimiento, además de contar con pautas y recomendaciones para conseguir programas óptimos

4.11.1 Módulos o componentes de SAP en Colpensiones.

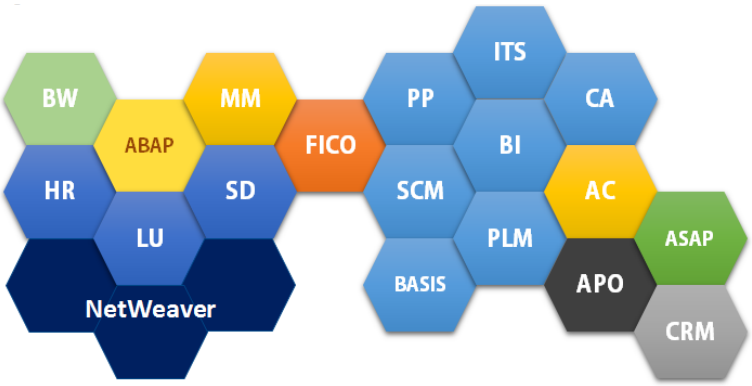


Imagen 4. Módulos o Componentes SAP Colpensiones.

Tabla de módulos y sub módulos SAP.

Módulo	Sub módulo	Sigla SAP
Finanzas	Contabilidad financiera - General	AC
	Contabilidad financiera	FI
	Ledger de libro mayor (FI-)	GL
	Consolidación empresarial (FI-)	LC
	Contabilidad de acreedores (FI-)	AP
	Contabilidad de deudores (FI-)	AR
	Gestión de cuentas bancarias (FI-)	BL

	Contabilidad de activos fijos (FI-)	AA
	Gestión presupuestaria (FI-)	FM
	Gestión de viajes (FI-)	TV
	Tesorería	TR
Control	Controlling	CO
	Gestión de inversiones	IM
	Sistema de proyectos	PS
	Controlling Empresarial	EC
	Administración de bienes inmuebles	RE
Logística	Logística - General	LO
	Gestión de materiales	MM
	Ejecución de logística	LE
	Gestión de calidad	QM
	Mantenimiento	PM
Comercial	Ventas y distribución	SD
	Servicio al cliente	CS
Operación	Control y planificación de la producción	PP
	Sistema de proyectos	PS
	Medio ambiente, Salud y Seguridad	EHS
Capital Humano	Capital humano	HCM
	Administración de personal	PM
	Desarrollo de personal	PD
Workbench	Basis Components	BC

4.11.2 Ordenes de Transporte OT´s.

Las órdenes de transporte deben ser nombradas de acuerdo a la siguiente nomenclatura:

- Sistema SAP R3: CPDK###### (# Consecutivo del sistema)
- Sistema SAP PI: DPIK###### (# Consecutivo del sistema)

Si se generan varias órdenes de transporte como solución a un requerimiento funcional, deben ser consolidadas en una sola orden de transporte, con el fin de no generar problemas de dependencia entre ellas.

4.11.3 Tipos de Programa.

Código	Denominación	Descripción
Válido para programa ó transacción:		
B	Batch	En ellos se encuentran los programas que manejan Batch Input, Call Transaction, etc

Código	Denominación	Descripción
P	Procesos y otros desarrollos	Para el caso de un programa que presente una serie de procesos o realice actividades particulares. Estos programas se caracterizan por realizar más de una actividad de las mencionadas en un mismo programa.
R	Reporte	Para todos aquellos programas que impliquen una selección de datos y un listado en pantalla.
M	Mantenimiento de tablas	Procesos exclusivos al mantenimiento o carga inicial de una tabla
Válido sólo para transacción:		
A	Actualización	Transacción de actualización de tabla.
Q	Query	Transacción creada para query

4.11.4 Tipo De Formato.

Código	Descripción	ID
CHAR	String	C
CURR	Campo moneda, almacenado como DEC	
DATS	Campo para fecha (AAAAMMDD), almacenado como CHAR (8)	D
DEC	Campo cálculo o de importe con coma y signo +/-	P
FLTP	Cifra coma flotante con 8 byte de exactitud	F
INT	Entero	I
NUMC	String sólo con cifras	N
QUAN	Campo cantidad, apunta a campo unidades con formato UNIT	Q
TIMS	Campo hora (HHMMSS), almacenado como CHAR (6)	T
UNIT	Clave de unidades para campos QUAN	U

4.11.5 Marca Para Modificaciones.

Las marcas de modificación deben ser nombradas de acuerdo a la siguiente nomenclatura:

<Acción> @<NNN>

Donde:

Acción : - INSERT : Insertar
- REPLACE : Modificar
- DELETE : Comentar

NNN : Número correlativo de 3 dígitos

Modelo ejemplo:

CASO1: Agregar varias líneas de código.

```
*{ INSERT @001
*---------------------------------------------------------------------*
* Form TEXTO_EXP
*---------------------------------------------------------------------*
FORM TEXTO_EXP USING PI_ASNUM.
DATA: LS_ASNUM TYPE ASMD-ASNUM,
LS_TDNAME TYPE THEAD-TDNAME.
REFRESH GDT_LINES.
CLEAR GDT_LINES.
CALL FUNCTION 'CONVERSION_EXIT_ALPHA_INPUT'
EXPORTING
INPUT = PI_ASNUM
IMPORTING
OUTPUT = LS_ASNUM.
LS_TDNAME = LS_ASNUM.
CALL FUNCTION 'READ_TEXT'
EXPORTING
ID = GC_ID
LANGUAGE = SY-LANGU
NAME = LS_TDNAME
OBJECT = GC_OBJECT
TABLES
LINES = GDT_LINES
EXCEPTIONS
NOT_FOUND = 4.
IF SY-SUBRC <> 0.
MESSAGE S006 WITH TEXT-003 PI_ASNUM TEXT-004.
ENDIF.
ENDFORM. "TEXTO_EXP
*} INSERT @001
```

CASO2: Modificar varias líneas de código.

```
*{ REPLACE @001
*&---------------------------------------------------------------------*
*& Form GET_FILE
*&---------------------------------------------------------------------*
FORM GET_FILE CHANGING PO_FILE TYPE STRING.
DATA: LDT_FILE_TABLE TYPE STANDARD TABLE OF FILE_TABLE,
LS_FILE_FILTER TYPE STRING,
LI_RC TYPE I.
LS_FILE_FILTER = CL_GUI_FRONTEND_SERVICES=>FILETYPE_EXCEL.
CALL METHOD CL_GUI_FRONTEND_SERVICES=>FILE_OPEN_DIALOG
EXPORTING
WINDOW_TITLE = 'Abrir en ...'
FILE_FILTER = LS_FILE_FILTER
DEFAULT_FILENAME = PO_FILE
INITIAL_DIRECTORY = '/'
CHANGING
FILE_TABLE = LDT_FILE_TABLE
RC = LI_RC.
IF SY-SUBRC EQ 0 AND LI_RC EQ 1.
READ TABLE LDT_FILE_TABLE INDEX 1 INTO PO_FILE.
ENDIF.
ENDFORM. " GET_FILE
*} REPLACE @001
```

CASO3: Eliminar/Comentar varias líneas de código.

```
*{ DELETE @001
**&---------------------------------------------------------------------*
**& Module STATUS_0100 OUTPUT
**&---------------------------------------------------------------------*
** text
**----------------------------------------------------------------------*
*MODULE status_0100 OUTPUT.
* SET PF-STATUS '0100'.
* SET TITLEBAR '0100'.
*ENDMODULE. " STATUS_0100 OUTPUT
*} DELETE @001
```

CASO4: Agregar una línea de código

```
DATA: GS_ID TYPE ICON-ID. "INSERT @001
```

CASO5: Modificar una línea de código

```
REFRESH GT_LINES. "REPLACE @001
```

CASO6: Eliminar/Comentar una línea de código

```
* CHECK NOT GHT_BSEG[] IS INITIAL. "DELETE @001
```

4.11.6 Objeto de Autorización.

Todo desarrollo debe contar con la validación de uno o varios objetos de autorización según lo solicitado en el requerimiento respectivo. Esto con el objetivo de restringir la información a mostrar o procesar.
Asimismo la transacción con su respectivo objeto de autorización debe registrarse en la tx SU24 a fin de llevar un mejor control.

El uso de autorizaciones depende en su totalidad del programa que se esté desarrollando. La autorización más común es la que esta inherente a la ejecución del programa. Por ejemplo un reporte que muestra información no necesita autorizaciones internas en el programa sino que se controla con el perfil asignado a nivel de transacción.

Frecuentemente los programas que necesitan manejo interno de autorizaciones son las mejoras que pueden tener funcionalidades diferentes como Visualizar y Modificar, o para restringir visualización de información. Para estos casos "Authorization Check" deberá ser implementado según la necesidad. El desarrollador deberá ser el responsable de crear el objeto de autorización correspondiente, con su definición y campos de autorización si fueran requeridos; y realizar el chequeo respectivo en el programa utilizando para ello la sentencia AUTHORITY-CHECK.

- El desarrollador deberá informar el nuevo objeto de autorización para que los administradores del sistema y seguridad, en este caso a Informar a Gestión de Accesos o Seguridad TI, los roles, Grupos de roles creen las autorizaciones respectivas y las asignen a los perfiles.

Pasos a tener en cuenta para definir las autorizaciones y grupos de Autorizacíon:

- Analizar las tareas que debe realizar cada usuario.
- Agrupar usuarios según tareas comunes.
- Crear roles que se ajusten a cada tarea. Es recomendable que sean lo más específicos posible y en algunos casos incluso puede ser necesario desarrollar un nuevo rol.
- Asociar las autorizaciones específicas a cada rol. Es importante fijarnos en las transacciones que se incorporan en cada rol.
- Asignar roles según las tareas que realiza cada usuario.
- Dentro de cada usuario deberemos hacer un análisis de incompatibilidades de transacciones

Ejemplo:

```
*---------------------------------------------------------------------*
* VALIDACION DE PARAMETROS DE PANTALLA
*---------------------------------------------------------------------*
AT SELECTION-SCREEN.
AUTHORITY-CHECK OBJECT 'ZMM_WERKS'
ID 'TCODE' FIELD SY-TCODE
ID 'WERKS' FIELD P_WERKS
ID 'ACTVT' FIELD '03'. "visualizar
IF SY-SUBRC NE 0.
MESSAGE E000 WITH TEXT-E01 P_WERKS.
ENDIF.
```

Nota: Considerar el campo actividad en la validación, la actividad a validar debe ir acorde a la función principal del programa (crear, modificar, visualizar, borrar, imprimir, contabilizar, etc).

4.11.7 Grupo de Autorización.

Toda tabla Z creada debe ser asociada a un grupo de autorización, según lo solicitado en el requerimiento respectivo. Esto con el objetivo de restringir el acceso a la data de dicha tabla. Adicionalmente toda tx de mantenimiento creada para una tabla Z debe validar un objeto de autorización. Existirá para ello un programa único por módulo para la validación de objeto de autorización a tablas Z.

4.11.8 Paquete - Clase de desarrollo.

Todo objeto de repositorio Z válido para PRD, debe crearse haciendo referencia a un paquete de desarrollo. La asignación estará relacionada con el módulo SAP al que pertenece.

Ejemplo:

Paquete Descripción

- ZBC Uso genérico
- ZBW Business Warehouse
- ZCO Controlling
- ZCS Servicio al Cliente
- ZFI Finanzas
- ZHR Recursos Humanos
- ZMM Materiales
- ZPM Mantenimiento
- ZPP Producción
- ZPS Gestión de Proyectos
- ZQM Calidad
- ZSD Ventas y Distribución

- ZWM Gestión de Almacenes

4.11.9 Objetos de Prueba.

Los objetos creados para pruebas que no sean necesarios para el ambiente PRD, deben ser creados como objetos locales y el nombre debe empezar con Y.

4.11.10 Estructura de programa y encabezado.

Todo programa desarrollado debe incluir comentarios, con el propósito de facilitar a futuros programadores una mejor comprensión del código desarrollado y disminuir el impacto que representa para esta persona la modificación de un código no propio. Todo comentario debe estar en letra minúscula, además debe ser claro y conciso, dando una idea general de la función que realiza esa sección de código en el programa.

El encabezado y la estructura del programa deben ser de la siguiente manera, siempre se deben actualizar las modificaciones.
Nota: esta estructura solo aplica a programación de eventos y no en objetos.

```
************************************************************************************
* Programa           : ZXXXXXXX
*
* Programador ABAP   : NOMBRE APELLIDO - EMPRESA
* Consul. Funcional  : NOMBRE APELLIDO - EMPRESA
* Area/Proyecto      : Mantenimiento SAP
*
* Fecha creación     : DD/MM/2020 (DD/MM/AAAA)
*
* Tipo Programa      : Reporte/Actualizador/Batch Input/Screen Painter/etc
*
* Modo Ejecución     : online/batch
*
* Objetivo           :
*
* Descripción        :
*
*
*
************************************************************************************
* Includes :  ZXXXXXXXXX  - Descripción
*             ZXXXXXXXXX -  Descripción
************************************************************************************
*
*                          MODIFICACIONES
*
*-----------------------------------------------------------------------
*  Fecha               Programador                      Orden
*-----------------------------------------------------------------------
*  DD.MM.AAAA          Nombre y Apellido (ID)
*
*  Descripción:
*
*-----------------------------------------------------------------------
*  DD.MM.AAAA          Nombre y Apellido (ID)
*
*  Descripción:
*
************************************************************************************

REPORT ZVPCRR01 MESSAGE-ID Z0
                LINE-SIZE  121
                LINE-COUNT 65
                NO STANDARD PAGE HEADING.
```

Imagen 5. Ejemplos de Comentario..

Las primeras líneas del programa deben ser destinadas al nombre del programa, clase de mensaje, tamaño del reporte de salida, etc.

El título MODIFICACIONES, sólo se coloca la primera vez que se va a hacer una modificación al programa, los encabezados se colocan todas las veces que se hagan modificaciones.

```
************************************************************************************
* INCLUDES
************************************************************************************


************************************************************************************
* DATA DECLARATION
************************************************************************************

*-TABLES------------------------------------------------------------------

*-CONSTANTS---------------------------------------------------------------

*-VARIABLES---------------------------------------------------------------

*-STRUCTURES/TYPES--------------------------------------------------------

*-INTERNAL TABLES---------------------------------------------------------

*-FIELD SYMBOLS-----------------------------------------------------------



************************************************************************************
* SELECT OPTIONS / PARAMETERS
************************************************************************************


************************************************************************************
* INITIALIZATION
************************************************************************************

INITIALIZATION.

*        ...program code ...
```

Imagen 6. Ejemplo de Comentarios.

```
************************************************************************************
* AT SELECTION-SCREEN.
************************************************************************************

AT SELECTION-SCREEN.

*        ... program code ...


************************************************************************************
* MAIN PROCESSING SECTION
************************************************************************************


START-OF-SELECTION.

*        ... program code ...

END-OF-SELECTION.

*        ... program code ...


AT USER-COMMAND.

*        ... program code ...

TOP-OF-PAGE.

*        ... program code ...
```

Imagen 7. Ejemplo de Comentarios.

```
*---------------------------------------------------------------------*
* FORMS
*---------------------------------------------------------------------*

*---------------------------------------------------------------------*
* FORM F_FORM_NAME
*---------------------------------------------------------------------*
* DESCRIPTION
*---------------------------------------------------------------------*
* INPUT:
*  -->  p1      text
* OUTPUT:
*  <--  p1      text
*---------------------------------------------------------------------*
*form f_XXXXXXXXXXXXX.

*-DATA DECLARATION----------------------------------------------------*
*-MAIN PROCESSING SECTION---------------------------------------------*

* Inicializa las variables de control de error.
*  VP_RETURN      = '0'.
*  VP_RETURN_DESC = ''.

*endform.
```

Imagen 8. Ejemplo de Comentarios.

4.11.11 Declaraciones

Se debe utilizar la siguiente nomenclatura para declarar los nuevos objetos de desarrollo ABAP

A. Declaraciones dentro de un Programa ABAP

Esta sección se debe utilizar para la declaración de todas las constantes y variables globales utilizadas en el programa.
La declaración de datos debe respetar el siguiente formato:

Declaración	Se debe escribir de la siguiente forma
Variables Locales	VL_XXXXXXXX
Variables Globales	V_XXXXXXXX
Checkbox	CH_XXXXXXXX
Radio Button	RB11_XXXXXXX, RB12_XXXXX, RB1N_XXXX
Rangos	R_XXXXXXX
Rangos locales	RL_XXXXXX
Constantes globales	C_XXXXXXX
Constantes locales	CL_XXXXXXX
Estructura	WA_XXXXX
Estructura local	WAL_XXXXX
Tabla interna	TI_XXXXX
Tabla interna local	TIL_XXXXX
Tipos	TY_XXXXXXXXX
Parámetros	P_XXXXXX

Select Option	SO_XXXXXXX
Forms	F_XXXXXXXXXXXXX
Parámetros de un Form	P_XXXXX

Modelo ejemplo:

```
*---------------------------------------------------------------------*
* DECLARACION DE TABLAS *
*---------------------------------------------------------------------*
TABLES: T001W, "Centros/Sucursales
MBEW, "Valoración de material
MSEG. "Segmento doc.material
*---------------------------------------------------------------------*
* CONSTANTES *
*---------------------------------------------------------------------*
CONSTANTS: GC_SOBKZ TYPE QBEW-SOBKZ VALUE 'Q',
GC_XXXXX TYPE C LENGTH 10 VALUE 'OTROS',
GC_WERKS TYPE T001W-WERKS VALUE '105A'.
*---------------------------------------------------------------------*
* DECLARACION DE VARIABLES *
*---------------------------------------------------------------------*
TYPE-POOLS: SLIS. "Agregar comentario
* Campos globales
DATA: GS_BUKRS TYPE T001-BUKRS, "Adicionar comentario
GS_CAMPO2 TYPE C LENGTH 3, "Adicionar comentario
GN_CAMPO3 TYPE N, "Adicionar comentario
GD_BUDAT TYPE BKPF-BUDAT. "Adicionar comentario
* Tipos
TYPES: GTY_MARC TYPE MARC.
TYPES: BEGIN OF GTY_MAKT,
MATNR TYPE MAKT-MATNR,
MAKTX TYPE MAKT-MAKTX,
END OF GTY_MAKT.
*Tipo tabla
TYPES: GTYT_MARC TYPE TABLE OF MARC,
"Standard
GTYD_MARC TYPE STANDARD TABLE OF GTY_MARC,
"Hashed
GTYH_MARC TYPE HASHED TABLE OF GTY_MARC WITH UNIQUE KEY MATNR WERKS,
"Sorted
GTYS_MARC TYPE SORTED TABLE OF GTY_MARC WITH NON-UNIQUE KEY MATNR.
*Tablas internas
DATA: GT_MARC TYPE TABLE OF GTY_MARC,
"Standard
GDT_MARC TYPE GTYD_MARC,
"Hashed
```

```
GHT_MARC TYPE GTYH_MARC,
"Sorted
GST_MARC TYPE GTYS_MARC.
* Estructuras
DATA: BEGIN OF GWA_XXXX,
BUKRS TYPE BKPF-BUKRS, "Sociedad
BELNR TYPE BSEG-BELNR, "Fecha de contabilización
END OF GWA_XXXX.
DATA: GWA_MARC LIKE LINE OF GT_MARC. "Agregar comentario
* Rangos
DATA: GR_BUKRS TYPE RANGE OF BKPF-BUKRS, "rango de sociedades
GWA_BUKRS LIKE LINE OF GR_BUKRS.
* Field symbols
FIELD-SYMBOLS: <GWA_MARC> LIKE LINE OF GDT_MARC,
<GS_MATNR> TYPE MARA-MATNR.
* Fields groups
FIELD-GROUPS: FG_HEADER, "Agregar comentario
 FG_DETALLE. "Agregar comentario
```

Donde:
Las primeras líneas de este bloque deben utilizarse para la declaración de las tablas y estructura de datos utilizada por el programa. Cada tabla declarada debe tener a su derecha el comentario sobre la descripción breve de la misma.

La segunda sección del bloque se utilizará para la declaración de constantes.

La tercera sección del bloque se utilizará para la declaración de variables globales. Esto incluye campos, tablas internas, estructuras, etc. en la forma y orden en que se muestra en el ejemplo de arriba. Cada objeto adicionado debe comentarse.

Nota: Debe tratarse en lo posible, definir las variables haciendo referencia a campos definidos en el diccionario de datos, mediante la utilización de TYPE.
La sentencia OCCURS todavía se permite por razones de compatibilidad con versiones más antiguas. Sin embargo, se recomienda que solamente se utilice las nuevas versiones de las definiciones de tablas. Esto facilita reutilizar el código en un contexto de Objetos ABAP.

Se recomienda (en la medida de lo posible) el uso de FIELD-SYMBOLS para modificar una tabla interna, en vez de usar líneas de cabeceras (_WA_XXXX), por motivos de performance.

4.11.12 Declaración de campos de pantalla.

Esta sección se debe utilizar para la declaración de todos los campos que se mostrarán en la pantalla de inicio del programa y que permiten la selección de la información (PARAMETERS, SELECT-OPTIONS, ETC).
No debe crearse un programa sin al menos un parámetro de selección. Esto posibilitará que al ejecutar el programa se muestre primero una pantalla de selección, con el título del programa y los campos de selección, evitando que el usuario ejecute el programa por error. La declaración de parámetros de pantalla debe respetar el siguiente formato:

Modelo ejemplo:

```
*---------------------------------------------------------------------*
* DISEÑO PANTALLA DE SELECCION
*---------------------------------------------------------------------*
SELECTION-SCREEN BEGIN OF BLOCK B_01 WITH FRAME TITLE TEXT-B01.
PARAMETERS: P_BUKRS TYPE T001-BUKRS OBLIGATORY DEFAULT '10',
P_WERKS TYPE T001W-WERKS MEMORY ID WRK OBLIGATORY.
SELECT-OPTIONS: S_BKLAS FOR MBEW-BKLAS NO INTERVALS NO-EXTENSION,
S_MATNR FOR MSEG-MATNR.
SELECTION-SCREEN END OF BLOCK B_01.
SELECTION-SCREEN BEGIN OF BLOCK B_02 WITH FRAME TITLE TEXT-B02.
PARAMETERS: P_LFMON TYPE MBEW-LFMON OBLIGATORY,
P_LFGJA TYPE MBEW-LFGJA OBLIGATORY,
P_WAERS TYPE T001-WAERS OBLIGATORY.
SELECTION-SCREEN END OF BLOCK B_02.
PARAMETERS: P_ALV RADIOBUTTON GROUP G1 DEFAULT 'X',
P_REL RADIOBUTTON GROUP G1,
P_ALM_R AS CHECKBOX,
P_STCK AS CHECKBOX DEFAULT 'X'.
```

Donde:
Deben posicionarse los distintos PARAMETERS y SELECT-OPTIONS, de acuerdo a la posición en la que se desea aparezcan en la pantalla.
Debe comentarse cada parámetro declarado. Este tipo de variables deben ser utilizadas como alternativa para evitar los “HARD_CODES”

4.11.13 Validación de campos de pantalla e inicialización.

En esta sección del programa se deben efectuar las validaciones de todos los campos de la pantalla de selección y realizar las inicializaciones de variables si corresponde. El formato es el que se muestra a continuación:

```
*---------------------------------------------------------------------*
* INICIALIZACION
*---------------------------------------------------------------------*
INITIALIZATION.
PERFORM INICIALIZACION.
*---------------------------------------------------------------------*
* VALIDACION DE PARAMETROS DE PANTALLA
*---------------------------------------------------------------------*
AT SELECTION-SCREEN OUTPUT.
AT SELECTION-SCREEN ON VALUE-REQUEST FOR P_WAERS.
PERFORM HELP_WAERS USING P_WAERS.
AT SELECTION-SCREEN ON BLOCK B_01.
AT SELECTION-SCREEN ON P_BUKRS.
AT SELECTION-SCREEN.
```

Toda validación que se realice sobre los parámetros de entrada debe efectuarse utilizando estos eventos.
De esta manera, se enviarán los mensajes de error o información sobre la pantalla de selección, posibilitando de esa manera que el usuario corrija el error según corresponda.

Pueden crearse subrutinas del tipo PERFORM para agrupar las validaciones correspondientes a un evento de este tipo y de esa manera mejorar la modularización del programa, facilitando los probables cambios posteriores.

El evento INITIALIZATION debe utilizarse para cargar previamente a su utilización las variables deseadas.
Podrá crearse una subrutina para agrupar todas las inicializaciones y modular el programa.

4.11.14 Paquetes o clases de desarrollo Z.

Al momento de realizar un nuevo desarrollo se debe asociar a un paquete para poder transportarlo entre los diferentes ambientes, en Colpensiones existen los siguientes paquetes activos:

- Z001: Configuraciones técnicas o tipo BASIS.
- ZFI: Desarrollos para el área Financiera.
- ZHCM: Desarrollos para Recursos Humanos.
- ZCO: Desarrollos para Controlling.
- ZFM: Desarrollos para Gestión Presupuestaria.
- ZTR: Desarrollos para Tesorería.
- ZBCM: Desarrollos para Bank Communication Management.
- ZMM: Desarrollos para Gestión de Materiales.
- ZEAPS: Desarrollos para Contabilidad Presupuestaria.
- ZWKFE: Desarrollos de eDOC.

4.11.15 Dynpros, Status y Títulos.

Para la creación de cada uno de estos objetos, la nomenclatura a seguir será números correlativos de 4 dígitos con intervalos de 100 en 100, es decir, 0100, 0200, etc., guardando asociación entre los 3 objetos mediante el mismo número de codificación.

Por ejemplo a la dynpro 0100 está asociada el status 0100 y el título 0100.

De darse el caso que dos dynpros estén reutilizando algún objeto en particular, se debe, mantener la integración en código de los tres objetos, es decir,

- Dynpro: 0100 -> Status: 0100 -> Título: 0100.
- Dynpro: 0200 -> Status: 0100 -> Título: 0200.
- Dynpro: 0300 -> Status: 0300 -> Título: 0300.

Como se observa el Status 0200 no se ha tomado, en caso, a futuro la Dynpro 0200 necesite poseer su propio status.

- **Elementos de Dynpros:** Para el diseño de una dynpro es necesario el uso de elementos de dynpros, para los cuáles usaremos la siguiente nomenclatura para los nombres de campo, dependiendo del elemento a utilizar:

Elemento	Nomenclatura
Campo de Texto	TF_<nombre campo texto>
Campo Entrada/Salida	PF_<nombre campo entrada/salida>
CheckBox	CB_<nombre CheckBox>
Radio Button	RB_<nombre Radio Button>
PushButton	PB_<nombre Push Button>
Marco	GB_<nombre Marco>
Control de TabStrip	SC_<nombre TabStrip>
Área SubScreen	SA_<nombre Subscreen Area>
Table Control	TC_<nombre Table Control>
Custom Control	CC_<nombre Custom Control>
Status Icon	SI_<nombre Status Icon>

Nota: Como es conocido, estos elementos pueden ser obtenidos también vía referencia ya sea del Diccionario de datos o también de las variables que posee el programa, en dicho caso el nombre del objeto cargado en la dynpro se mantendrá con el nombre que se coloca automáticamente al jalar el objeto con la referencia determinada. Además es posible indicar un nombre de campo precedido por un '*' cuando se trata de la misma referencia al Diccionario de datos ABAP.

4.11.16 Formatos de Datos del Diccionario ABAP.

Declaración	Se debe escribir de la siguiente forma
Campo nuevo en una tabla base de datos	XXXXXXX Donde XXXXXXX es el nombre del campo nuevo
Elemento de datos	ZZ_XXXXXXX (Donde XXXXXXX es el nombre del campo al que se va a asociar)
Dominio	ZDXXXXXXX (Donde XXXXXXX deberá referenciar el tipo de datos)
Nombre tabla	ZTXXXXXXX (Donde XXXXXXX se utiliza para describir el nombre de la tabla)
Estructuras	ZWXXXXXX (Donde XXXXXXX se utiliza para describir el nombre de la estructura)

A. Tabla.

Los nombres de estos objetos tendrán la siguiente forma Z[MS]T_[X…X]
Donde:

Z : Por definición SAP
MS : Módulo SAP (ver tabla 1)
T : Constante
X…X : Descripción literal
Ejm: ZCOT_MATCOSTO

Nota: Se debe definir la Categoría de Ampliación antes de activar la tabla.
Para acceder: ingresar a la tabla en la Tx. SE11, ir al menú Detalles -> Categoría de Ampliación.

Por seguridad se debe asociar un grupo de autorización para toda tabla Z (Ver punto 3.2 del presente Manual).

B. Campo De Tabla.

El nombre que llevarán los diferentes campos que conforman una tabla de base de datos, estará asignado de la siguiente forma, dependiendo el caso en que se encuentre:

- ✓ De ser un campo que hace referencia a un tipo de dato existente en SAP, se mantendrá el mismo nombre de campo usado por la nomenclatura estándar de SAP.

  Ejemplo: Sociedad -> BUKRS.

- ✓ De ser un campo creado según la situación del negocio o cliente, el nombre del campo estará asignado por un nombre adecuado seleccionado bajo el criterio del consultor responsable, tomando un tamaño recomendado de 5 caracteres.

Ejemplo:
Alimentos -> ALMTS.
Tienda de Textiles -> TDATL.

C. Indice De Tabla.

Los índices que sean necesarios crear, para la optimización en el acceso de lectura a las tablas de bases de datos tendrán la siguiente forma Z[NN]

Donde:

Z : Por definición SAP
NN : Secuencia numérica de 2 caracteres.

Ejm: Z01.

Nota: Al crear el índice Z por la tx SE11, se debe elegir la opción “Crear índice extensión”.

D. Vista.

Los nombres de estos objetos tendrán la siguiente forma Z[MS]V_[X…X].
Donde:

Z : Por definición SAP
MS : Módulo SAP (ver tabla 1)
V : Constante
X…X : Descripción literal

Ejm: ZMMV_MATCENTRO

E. Vista Append.

Los nombres de estos objetos tendrán la siguiente forma ZZ[MS]V_[X…X].

F. Vista Ayuda.

Los nombres de estos objetos tendrán la siguiente forma Z[MS]VH_[X…X].

G. Cluster De Vista.

Los nombres de estos objetos tendrán la siguiente forma Z[MS]VC_[X…X]

H. Estructura.

Los nombres de estos objetos tendrán la siguiente forma Z[MS]S_[X…X]
Donde:

Z : Por definición SAP
MS : Módulo SAP (ver tabla 1)

S : Constante
X...X : Descripción literal
Ejm: ZWMS_STOCKMAT

Nota: Se debe definir la Categoría de Ampliación antes de activar la estructura.

I. Estructura Append.

Los nombres de estos objetos tendrán la siguiente forma ZZ[MS]S_[X...X]
Los nombres de los campos deberán empezar con ZZ.

J. Tipo Tabla.

Los nombres de estos objetos tendrán la siguiente forma Z[MS]TT_[X...X]
Donde:

Z : Por definición SAP
MS : Módulo SAP (ver tabla 1)
TT : Constante
X...X : Descripción literal
Ejm: ZMMTT_RESERVAS.

4.11.17 Objetos que podemos crear en SAP ABAP.

Declaración	Se debe escribir de la siguiente forma
Reporte o Programa	ZXXXXXX Donde XXXXXX es el nombre descriptivo
Formularios	ZXXX_XXXXX Ej: ZHCM_RESUMEN_NOMINA. Primero se coloca el nombre del módulo y después el nombre descriptivo
Transacciones	ZXXX_XXXXX Ej: ZHCM_RESUMEN_NOMINA. Primero se coloca el nombre del módulo y después el nombre descriptivo
Módulo de Función	ZMFXXXXXX (Donde XXXXXX se utiliza para describir el nombre del módulo de función)
Grupo de Función	ZGFXXXXXXX (Donde XXXXXX se utiliza para describir el nombre del grupo de función)
Module Pool o Programa de Diálogo	ZMPXXXXXXX (Donde XXXXXX se utiliza para describir el objetivo del programa)
Vistas de Actualización	ZVAXXXXXXX (Donde XXXXXX se utiliza para describir el nombre de la vista)
Clase de Mensajes	ZCMXXXXXXX (Donde XXXXXX se utiliza para describir el nombre de la clase de mensajes)
Clase de Objetos ABAP	ZCLXXXXXXX (Donde XXXXXX se utiliza para describir el nombre de la clase)
Clase de Desarrollo o Paquete	ZXX (Donde XX se utiliza para describir el nombre módulo)
Ayuda de Búsqueda	ZABXXXXXXX (Donde XXXXXX se utiliza para describir el nombre de la ayuda de búsqueda)

Declaración	Se debe escribir de la siguiente forma
Objeto de Bloqueo	ZOBXXXXXXX (Donde XXXXXX se utiliza para describir el objetivo del objeto de bloqueo)

4.11.18 Includes.

Siempre se debe modularizar los programas de la siguiente manera:
Include ZXXXX_TOP. (Include de declaración de datos y selección, donde XXXX es el nombre del programa)
Include ZXXXX_FORMS. (Include de declaración de subrutinas, donde XXXX es el nombre del programa)

4.11.19 Orden de Transporte.

"Cuando un aplicativo es desarrollado, todos los objetos y características se asocian a un referente denominado orden de transporte" (Ehret Stefan, 2012, Taw10_1); este agrupa todos los elementos implicados en el desarrollo para que puedan ser transportados de un ambiente a otro.



Imagen 9. Orden de transporte.

4.11.20 Recomendaciones.

A. Código Quemado (HARD-CODE).

Bajo ninguna circunstancia está permitido el uso de Código quemado en los desarrollos.
En su reemplazo, se debe crear una tabla de Parámetros tipo "Z", una pantalla inicial de captura de parámetros que le permita al usuario realizar mantenimiento al desarrollo sin necesidad de efectuar modificaciones al código o un set de datos.

B. Unidades lógicas.

Siempre estructure la solución de tal forma que se identifique con facilidad la secuencia lógica de su operación. Defina rutinas o funciones lógicas reutilizables, para aquellos procesos complejos, de

forma que puedan ser aplicados en diferentes componentes, aplicando la lógica consistente de su concepción inicial.

C. Código Muerto.

Evite la existencia de código muerto a lo largo de los programas. Remueva definiciones de campos que nunca serán usados y códigos que nunca serán ejecutados, a menos que sean modificaciones del estándar SAP, donde hay que dejarlos como comentarios.

D. Indentación.

Para mantener el diseño del código uniforme, todos los desarrollos deberán utilizar la funcionalidad "Pretty Printer" del editor para auxiliar en la justificación del código fuente.

E. Puntos a revisar en una OT WB previo al pase a PRD.

El objetivo de este punto es asegurar que los desarrollos a transportar a PRD: cumplan los estándares, sean programas seguros, se eviten errores técnicos en PRD y sean programas ágiles.

- ✓ Verificar que las OTs no presenten cruce de versiones en la comparación de la OT de DEV versus QA y PRD(no debemos pasar cambios u objetos que no estén relacionados con el requerimiento de la OT – cambios no aprobados).
- ✓ Verificar que las OTs estén acumulando todas las versiones anteriores.
- ✓ Verificar que la OT a pasar sea la más actualizada del requerimiento (versión final).
- ✓ Revisar que se cumpla la nomenclatura de los objetos nuevos.
- ✓ Revisar validaciones críticas: FOR ALL ENTRIES con tabla interna llena, READ TABLE con verificación de SY-SUBRC EQ 0, FIELD SYMBOL con verificación de asignación previo a su invocación, etc.
- ✓ Revisar consultas críticas que puedan afectar performance: SELECT SINGLES dentro de LOOPs, LOOPs anidados, SELECT sin llave o índice, etc.
- ✓ Ejecutar la tx SLIN o Verificación ampliada de los programas, revisar sólo los puntos que puedan resultar críticos.
- ✓ Revisar que no se pase código nuevo comentado (evitar que "código nuevo que hemos desechado" pase a PRD).
- ✓ Verificar que todos los cambios estén encerrados en marcas y tengan un encabezado de creación ó modificación.
- ✓ Verificar que si son programas nuevos deben validar un objeto de autorización y deben estar registrados en la SU24.
- ✓ Verificar que si son tablas Z deben estar asociados a un grupo de autorización.
- ✓ Verificar que datos organizacionales ó datos críticos no pasen por código duro sino por tabla de constantes.

- ✓ Revisar posibles dependencias (por ejemplo, pasar una función cuyo grupo de funciones aún no está en PRD)
- ✓ Todos los transportes deben hacerse por la herramienta de transportes https://dynamicot.mybluemix.net/#/login.

4.12 INTEGRACIONES EN SAP PI.

Asegurar la Conectividad a los ambientes de SAP PI con Gestión de Accesos, y luego debe asegurar de contar con la siguiente información:

- Estructuras de origen y destino que vamos utilizar, es lo equivalente a la definición de la estructura de una tabla (nombre de campos, tipos, valores por defecto, restricciones) quizás un poco más flexible considerando que son estructuras XML y que estas pueden aceptar varios niveles de anidamiento y ocurrencia.

- Determinar las reglas que se van a utilizar para convertir la estructura de origen a la estructura de destino, para nuestro caso esto será casi intuitivo, pero para ejemplos posteriores veremos situaciones más complicadas.

- Determinar el medio por del cual se va obtener y dejar la información que se va procesar en PI. Para nuestro caso en particular nos referimos al folder y nombre del archivo de origen y el nombre o nombres de los archivos de destino. En ciertos casos en particular vamos a necesitar de reglas para obtener y dejar el mensaje esto más adelante será explicado.

Estructuras de origen y destino

Origen
<origen>
<nombre>Alejandra</nombre>
</origen>

Destino
<destino>
<mensaje>Hola Alejandra</mensaje>
</destino>

4.12.1 System Landscape Directory , Lista de actividades

a. Crear el Software Component Version en el System Landscape Directory (SLD)

4.12.2 Integration Repository

Importar el Software Component Version en el Integration Reposity (IR)
Crear los Namespaces
Crear los Data Types de las estructuras de Origen y de Destino
Crear los Message Types
Crear los Message Interfaces
Crear el Message Mapping
Crear la Interface Mapping

4.12.3 Integration Directory

Crear Scenario
Crear el Party de Origen y el Party de Destino
Crear el Service de Origen y El Service de destino
Crear el Communication Channel de origen y el Communication Channel de destino
Crear el Sender Agreement y Receiver Agreement
Crear el Receiver Determination y Interface Determination

4.12.4 Message Monitoring y Runtime Workbench

Verificar que se hayan ejecutado correctamente los pasos del Pipeline
Verificar que la información ha pasado correctamente por los adaptadores.

4.13 CALIDAD DEL CÓDIGO INSPECTOR Y TRANSACCIÓN SAT.

Razones por las que el sistema SAP ERP se ve afectado a nivel de rendimiento, a continuación, las posibles causas más frecuentes:

- La mayoría de los casos ocurre porque el programa zeta construido para satisfacer una necesidad no está leyendo los índices creados.
- En algunos casos los programas zeta no están construidos cumpliendo las buenas prácticas, por nombrar alguno de ellos, el select *, crear más de tres join en una sentencia, entre otros.
- Inconvenientes con el tamaño de los índices, estos tienen igual o mayor tamaño que la tabla. Solo se deben reconstruir a través de la se14, por lo general un índice no debe ocupar más del 10% del tamaño de la tabla, máximo un 20% exagerando.
- Problemas con los programas estándar. Para ello solo basta con revisar si existen mejoras que puedan ser aplicadas a través de notas OSS.
- La tabla que el programa zeta está leyendo no tiene índices creados.

- Otra causa es que el programa estándar a pesar de estar leyendo los índices aun toma mucho tiempo de ejecución, para ellos se debe consultar notas oss que la mayoría de las veces recomienda crear un índice adicional o en su defecto actualizar el programa estándar.
- También sucede que algunas sentencias ya son obsoletas y deben ser actualizadas.
- La tabla que el programa zeta o estándar no tiene las estadísticas actualizadas.
- Dentro del proceso de análisis, uno de los mensajes que se identifican y te ponen en alerta, es cuando en el programa zeta del cual se está realizando el análisis, no está utilizando los índices, se refleja con el texto conocido como "FULL TABLE SCAN".

Ahora, con relación a los índices para una tabla, existen algunas reglas que debemos tener claras a saber:

- No crear 2 índices con los mismos campos.
- El índice se debe crear con campos selectivos que van ser utilizados por la consulta y/o programa zeta y que son parte de la tabla.
- Como regla de oro un índice no debe contener más de cuatro campos
- Otra regla de oro a la hora de crear índices es no crear más de 5 índices por tabla.
- No cambie un índice creado por SAP, a menos que este soportado en una nota OSS.
- Nunca cambie un índice de más de 10 MB de tamaño durante horas de operación, ya que cuando se modifica el índice, este se bloquea y por ende toda operación que sobre el índice se esté realizando y que genera como consecuencia un congelamiento en algunas transacciones relacionadas o tienen relación con la tabla.

Transacciones propuestas para hacer seguimiento al programa y su buena ejecución:

- Code Inspector
- Transacción SAT (Abap runtme Analysis)
- Transacción ST05 (SQL trace)
- Transacción STAD (Statistics)
- Transacción ST03 (Work load analysis)
- SQLM /SDF/ZQLM abap SQL monitor

4.13.1 Code Inspector

El inspector de código es una herramienta genérica que se utiliza para comprobar los objetos del repositorio de SAP. Se utiliza para definir inspecciones que examinan conjuntos de objetos con la ayuda de variantes de verificación. Comprueba el rendimiento, la seguridad, la sintaxis y la adherencia a las convenciones de nomenclatura de objetos de repositorio individuales o conjuntos. También lo usa para recuperar información estadística o para buscar ciertos tokens ABAP. Como

resultado de una inspección, recibe información, advertencias y mensajes de error sobre las diferentes propiedades de los objetos examinados.

Nota para iniciar Code Inspector, realice una de las siguientes acciones:

- Elija Menú SAP → Herramientas → Mesa de trabajo ABAP → Prueba → Inspector de código.
- Desde SAP Easy Access, ingrese el código de transacción SCI .

Características

El inspector de código comprende los siguientes elementos:

- Inspección, donde puede realizar comprobaciones en un objeto.

Puede visualizar, crear, cambiar, copiar, crear una nueva versión de, eliminar y ver los resultados de una inspección.

- Conjunto de objetos, donde puede determinar los objetos a comprobar.

Puede visualizar, crear, cambiar, copiar y eliminar un conjunto de objetos.

- Variante de verificación, donde puede definir las verificaciones individuales que se realizan en los objetos.

Puede visualizar, crear, modificar, copiar y eliminar una variante de cheque.

Restricciones

El rango de funciones del Inspector de código se limita a verificar definiciones de objetos estáticos y solo puede señalar ciertos problemas. Por ejemplo, no puede dar una predicción precisa del desempeño general del programa. Para hacer esto, necesita analizar la ejecución del programa en tiempo de ejecución, utilizando el Análisis en tiempo de ejecución (código de transacción SAT), el Análisis de rendimiento (código de transacción ST05), el Análisis de rendimiento global (código de transacción ST30) u otra herramienta relevante.

Es requerido que el desarrollador ejecute el Code Inspector y entregue el resultado del mismo en conjunto con las la pruebas Unitarias o técnicas, en los ambientes de DEV y QA o SANDBOX.

4.13.2 Análisis de tiempo de ejecución SAT ABAP

La herramienta de análisis en tiempo de ejecución SAT ABAP se puede utilizar para identificar problemas de rendimiento en programas ABAP. Se debe ejecutar esta transacción y entregar su resultado como parte de las pruebas unitarias y técnicas.

Utilice una estrategia de reducción de tamaño en cada medición de rendimiento, al principio solo para declaraciones particulares (funciones, formularios, métodos, etc.) para buscar los principales consumidores. Luego, puede restringir la medición en una segunda ejecución a los módulos que están matando el rendimiento, que detectó como puntos de revisión durante la primera ejecución. Además, recomendamos medir solo la parte relevante de su aplicación ABAP activando y desactivando el rastreo de forma interactiva, de modo que rastree solo las secciones críticas.

CONTROL DE CAMBIOS DEL DOCUMENTO

FECHA	VERSIÓN	MODIFICACIÓN	ELABORÓ	REVISÓ.	APROBÓ.
02/09/2020	1	Creación	Nombre: Nicolas Salamanca Laverde. Cargo: Profesional Master 320-8	Nombre: Donald Pimienta Cargo: Director de sistemas de información.	Nombre: Javier Fúneme Cargo: Gerente de tecnologías de la información.