**Cumplir con el estándar de programación de Base de datos vigente.**

**Validación de existencia de objetos:**

|  |
| --- |
| if exists(select 1 from cobis..sysobjects where name = 'cl\_ente' and type = 'U')  begin  print 'Tabla Física cobis..cl\_ente existe'  end  else  begin  print 'Tabla Física cobis..cl\_ente NO existe'  end |
| if object\_id('cobis..cl\_ente') is null  begin  print 'Tabla Física cobis..cl\_ente NO existe'  end  else  begin  print 'Tabla Física cobis..cl\_ente existe'  end |
| if exists(select 1 from cobis..sysobjects where name = 'sp\_ruteo' and type = 'P')  begin  print 'SP Existe'  end  else  begin  print 'sp NO Existe'  end |
| if object\_id('tempdb..#prueba') is null  begin  print 'Tabla Temporal #prueba NO Existe'  end  else  begin  print 'Tabla Temporal #prueba Existe'  end |
| if exists(select 1  from cobis..syscolumns  where id = (select id from cobis..sysobjects where name = 'nombre\_tabla' and type = 'U')  and name = 'nombre\_campo')  begin  print 'Existe campo  end  else  begin  print 'No existe campo'  end |

**Usemos ANSI SQL, evitemos el uso de TransacSQL**

* Las tablas deben tener alias para que sea más legible la sentencia.
* Es necesario describir los campos, no se permite el uso de select \*
* No se permite \*= ni =\*.

|  |
| --- |
| select a.te\_valor, b.en\_nombre, c.campoY  from cl\_telefono a  inner join **cl\_ente b**  on **b.en\_ente** = a.te\_ente  left join <tabla> c  on c.id = b.campo  where a.te\_ente = @e\_ente |

**Creación de tablas**

* A excepción de los campos opcionales (como fecha de modificación, usuario de modificación), los campos deben ser definidos como obligatorios.
* Se debe anteponer al nombre de la table el usuario dbo.

|  |
| --- |
| Create table dbo.<tabla>(  Campo1 not null,  Campo2 not null,  ….) |

* Toda table debe contener clave primaria, para lo cual se usará la instrucción:

alter table dbo.<tabla> add constraint pk\_tabla primary key (id)

**Creación de campos**

* A excepción que la lógica lo requiera, evitemos crear un campo opcional.
* Es necesario validar la existencia del campo antes de crearlo:

|  |
| --- |
| if exists(select 1  from syscolumns  where id = (select id from sysobjects where name = 'nombre\_tabla\_a' and type = 'U')  and name = 'campo\_opcional')  begin  print 'Campo ya existe'  end  else  begin  print 'Creando campo'  exec ( 'alter table dbo.nombre\_tabla\_a add campo\_opcional datetime null')  end |
| if exists(select 1  from syscolumns  where id = (select id from sysobjects where name = 'nombre\_tabla\_b' and type = 'U')  and name = 'campo\_obligatorio')  begin  print 'Campo ya existe'  end  else  begin  print 'Creando campo'  exec ( 'alter table dbo.nombre\_tabla\_b add campo\_obligatorio int default 0 not null')  end |

* Se permite crear campos con valores por defecto:

|  |
| --- |
| if exists(select 1  from syscolumns  where id = (select id from sysobjects where name = 'nombre\_tabla\_c' and type = 'U')  and name = ' campo\_opcional')  begin  print 'Campo ya existe'  end  else  begin  print 'Creando campo'  exec ( 'alter table dbo.nombre\_tabla\_c add campo\_opcional int default 0 null')  end |
| if exists(select 1  from syscolumns  where id = (select id from sysobjects where name = 'nombre\_tabla\_d' and type = 'U')  and name = 'campo\_obligatorio')  begin  print 'Campo ya existe'  end  else  begin  print 'Creando campo'  exec ( 'alter table dbo.nombre\_tabla\_d add campo\_obligatorio int default 0 not null')  end |

* Cuando se requiera crear más de un campo a una misma tabla se debe ejecutar en una sola instrucción:

|  |
| --- |
| if exists(select 1  from syscolumns  where id = (select id from sysobjects where name = 'nombre\_tabla' and type = 'U')  and name = 'campo3') and  exists(select 1  from syscolumns  where id = (select id from sysobjects where name = 'nombre\_tabla' and type = 'U')  and name = 'campo4')  begin  print 'Campos ya existen'  end  else  begin  print 'Creando campos'  exec ( 'alter table dbo.nombre\_tabla add campo3 int default 0 not null, campo4 int null')  end  go |

**Eliminación de campos**

* Es necesario validar la existencia del campo antes de eliminarlo.

|  |
| --- |
| **SYBASE**  if exists(select 1  from syscolumns  where id = (select id from sysobjects where name = 'nombre\_tabla' and type = 'U')  and name = 'campo3')  begin  print 'Eliminando campo'  exec ( 'alter table dbo.nombre\_tabla drop campo3')  end  else  begin  print 'No existe campo'  end |
| **SQLSERVER**  if exists(select 1  from syscolumns  where id = (select id from sysobjects where name = 'nombre\_tabla' and type = 'U')  and name = 'campo3')  begin  print 'Eliminando campo'  exec ( 'alter table dbo.nombre\_tabla drop column campo3')  end  else  begin  print 'No existe campo'  end |

* Cuando se requiera eliminar más de un campo a una misma tabla se debe ejecutar en una sola instrucción:

|  |
| --- |
| **SYBASE**  if exists(select 1  from syscolumns  where id = (select id from sysobjects where name = 'nombre\_tabla' and type = 'U')  and name = 'campo2') and  exists(select 1  from syscolumns  where id = (select id from sysobjects where name = 'nombre\_tabla' and type = 'U')  and name = 'campo3')  begin  print 'Eliminando campo'  exec ( 'alter table dbo.nombre\_tabla drop campo2, campo3')  end  else  begin  print 'No existe campo'  end |
| **SQLSERVER**  if exists(select 1  from syscolumns  where id = (select id from sysobjects where name = 'nombre\_tabla' and type = 'U')  and name = 'campo2') and  exists(select 1  from syscolumns  where id = (select id from sysobjects where name = 'nombre\_tabla' and type = 'U')  and name = 'campo3')  begin  print 'Eliminando campo'  exec ( 'alter table dbo.nombre\_tabla drop column campo2, campo3')  end  else  begin  print 'No existe campo'  end |

**Validación de control de error**

* En Sybase y en SQL server hasta la versión 2000 se valida el control de error con la variable @@error.

|  |
| --- |
| insert into <tabla> (col1, col2) values(@e\_dat1, @e\_dat2)  if @@error<>0  begin  -- validaciones de error  end  delete <tabla>…  if @@error<>0  begin  -- validaciones de error  end  update <tabla>  if @@error<>0  begin  -- validaciones de error  end  exec <procedimiento>  if @@error<>0  begin  -- validaciones de error  End |

* Es posible capturar el código de error de la instrucción ejecutada y devolverá al cliente:

|  |
| --- |
| insert into <tabla> ...  set @v\_error = @@error  if @v\_error <> 0  begin  -- validaciones de error  return @v\_error  end |

* Es posible capturar el código de error de la instrucción ejecutada y la cantidad de registros afectados.

|  |
| --- |
| insert into <tabla> ...  set @v\_error = @@error, @v\_cant = @@rowcount  if @v\_cant = 0  begin  -- Validaciones de no existencia de registro  end  if @v\_error <> 0  begin  -- validaciones de error  return @v\_error  end |

* En versiones SQL Server 2005 en adelante la validación de error puede ser realizado a través del siguiente bloque:

|  |
| --- |
| Begin try  select 1/0  end try  begin catch  print 'error de procesamiento'  end catch |
| Advertencia: Desde cobis no es posible invocar a un procedimiento que tenga la instrucción begin try a pesar de estar en una versión de SqlServer 2005 o superior. |

**Procedimientos**

* En la sección de propósito se debe indicar si se trata de procesamiento OLTP ó BATCH.
* Para evitar que un procedimiento este asignado al usuario que realiza la compilación, al nombre del procedimiento se debe antepone **dbo.**

**dbo.**<nombre\_procedimiento>

* Si el procedimiento existe y si se compila sobre el motor SQL Server, para no perder permisos previamente asignados se debe compilar con la instrucción **alter procedure**.
* Evitar definir parámetros de entrada como opcionales 🡪 @e\_campo varchar(10) = null.
* El parámetro de entrada debe de ser del mismo tipo de dato y longitud del campo de la tabla que accede. Con esto evitamos posibles table scan.
* La primera línea de codificación del procedimiento almacenado debe ser **set nocount on**.
* Siempre revisar los planes de ejecución.
* Cuando creamos una tabla dentro de un procedimiento y por el volumen de los datos debe ser indexado, aplicar la siguiente lógica:

|  |  |
| --- | --- |
| Procedimiento principal | Procedimiento secundario |
| drop procedure dbo.pa\_demo\_principal  go  create procedure dbo.pa\_demo\_principal  as  set nocount on  if not object\_id('tempdb..#demo\_tmp\_data') is null  drop table #demo\_tmp\_data  select top 1000000  en\_ente, en\_nombre, en\_ced\_ruc  into #demo\_tmp\_data  from cobis..cl\_ente  if @@error<>0  begin  print 'Error en procesamiento'  return  end  create index i\_tmp on #demo\_tmp\_data(en\_ente)  update index statistics #demo\_tmp\_data  exec pa\_demo\_secuncario 1, 1000  if not object\_id('tempdb..#demo\_tmp\_data') is null  drop table #demo\_tmp\_data  go | if not object\_id('tempdb..#demo\_tmp\_data') is null  drop table #demo\_tmp\_data  go  select top 0  en\_ente, en\_nombre, en\_ced\_ruc  into #demo\_tmp\_data  from cobis..cl\_ente  go  drop procedure dbo.pa\_demo\_secuncario  go  create procedure dbo.pa\_demo\_secuncario  @e\_entini int,  @e\_entfin int  as  select  en\_ente, en\_nombre, en\_ced\_ruc  from #demo\_tmp\_data  where en\_ente between @e\_entini and @e\_entfin  go  if not object\_id('tempdb..#demo\_tmp\_data') is null  drop table #demo\_tmp\_data  go |

* Evitemos usar variables locales para acceder a tablas que contiene índice en especial aquellas tablas históricas en procesos de inserción, eliminación ó actualización masivas. Dividirlo en procedimientos almacenados:

|  |  |
| --- | --- |
| Procedimiento principal | Procedimiento secundario |
| drop procedure dbo.pa\_demo\_principal  go  create procedure dbo.pa\_demo\_principal  as  set nocount on  declare  @v\_ini int,  @v\_fin int,  @v\_tope int  --Ejemplo de recorrido de índice único  Select @v\_ini=min(id)  From <tabla>  Select @v\_fin=max(id)  From <tabla>  -- Ejemplo de procesamiento de 5000 registros  While @v\_ini <= @v\_fin  Begin  Set @v\_tope = @v\_ini + 5000  exec pa\_demo\_secuncario @v\_ini, @v\_tope  set @v\_ini = @v\_tope + 1  end  go | drop procedure dbo.pa\_demo\_secuncario  go  create procedure dbo.pa\_demo\_secuncario  @e\_idini int,  @e\_idfin int  as  update <tabla>  set <campo> = ‘dataFija’  where id between @e\_idini and @e\_idfin  if @@error <> 0  begin  --Control de error  End  -- Solo si la actualización es de 5000 registros o superior  Dump transaction <base> with truncate\_only  go |

* Cuando se invoca a un procedimiento almacenado y el valor de retorno se almacena en una variable local, la variable debe ser evaluada:

|  |
| --- |
| declare @v\_retorno int  exec @v\_retorno = pa\_demo\_consulta  if @v\_retorno = 0  begin  -- validación de retorno de ejecución del procedimiento  end |

* Para un nuevo procedimiento no se permite incluir permisos de ejecución de usuario, **grant execute.** Se debe remitir la malla de permisos a Seguridad Informática quién ejecutará los permisos respectivos luego de la compilación de los procedimientos.
* En sybase, para un procedimiento existente solo se permitirá incluir en el script el permiso que esté asignado en ese momento en el ambiente de producción, el cual es validado cuando se realiza la revisión QA.
* De requerir asignar un nuevo permiso de ejecución de usuario a un procedimiento existente, se debe remitir la malla de permisos a Seguridad Informática quién ejecutará los permisos respectivos luego de la compilación del procedimiento. ***De ejecutarlo antes de la compilación se perderá la asignación realizada por S.I.***

**Acceso de Índices**

* Evitemos el uso de la función isnull como evaluación de condiciones de aquellos campos que forman parte de un índice:

|  |
| --- |
| **Genera Table Scan**  select a.en\_ente, a.en\_nombre, a.en\_ced\_ruc, a.p\_pasaporte  from cobis..cl\_ente a  where a.en\_ced\_ruc = isnull(@e\_identifica, a.en\_ced\_ruc ) |
| **Solución:**  **Dividir en dos consultas:**  **Consulta por cédula específica**  select a.en\_ente, a.en\_nombre, a.en\_ced\_ruc, a.p\_pasaporte  from cobis..cl\_ente a  where a.en\_ced\_ruc = @e\_identifica and a.en\_subtipo = 'C'  **Consulta por todas las cédula**  select a.en\_ente, a.en\_nombre, a.en\_ced\_ruc, a.p\_pasaporte  from cobis..cl\_ente a  where a.en\_subtipo = 'C' |

* En la condición, definir primero los campos que son parte del índice y luego demás campos:

|  |
| --- |
| select a.en\_ente, a.en\_nombre  from cobis..cl\_ente a  where a.en\_ced\_ruc = @e\_cedruc and a.en\_pais = 1 |

* En la condición, evitemos evaluar variables/parámetro con un valor fijo:

|  |
| --- |
| select a.en\_ente, a.en\_nombre  from cobis..cl\_ente a  where @e\_tipo='C' and a.en\_ced\_ruc = @e\_cedruc and a.en\_pais = 1 |
| **Solución:**  If @e\_tipo='C'  Begin  select a.en\_ente, a.en\_nombre  from cobis..cl\_ente a  where a.en\_ced\_ruc = @e\_cedruc and a.en\_pais = 1  end |

* Evitemos el uso de funciones a los campos que son parte del índice:

|  |
| --- |
| select a.en\_ente, a.en\_nombre  from cobis..cl\_ente a  where convert(int, a.en\_ced\_ruc) = @e\_cedula and a.en\_pais = 10 |

**Depuración/actualización/inserción masiva de datos.**

* Luego de una inserción, actualización o eliminación masiva de datos, proceder a actualizar las estadísticas y ejecutar el comando sp\_recompile.

|  |
| --- |
| **Sybase:**  Update index statistics <tabla>  Exec sp\_recompile <tabla> |
| **SqlServer:**  Update statistics <tabla>  Exec sp\_recompile <tabla> |

**Consultas a tablas históricas**

* Cuando se requiere acceder a tablas con un gran volumen de información y estas a su vez realizan join a otras tablas, se debe dividir en tablas temporales para evitar consumo de memoria.

|  |
| --- |
| **Genera alto IO**  select c1, c2  from base\_his..tabla\_historica1 a  inner join tabla\_historica2 b  on b,id = a.codigo and b.estado = 'X'  inner join tabla\_historica3 c  on c,idhist3 = b.idhist2 and b.estado = 'Y'  where a.fecha between @e\_fecini and @e\_fecfin  and a.sucursal between @e\_sucini and @e\_sucfin  and a.entebetween @e\_entini and @e\_entfin |
| **Mejora de IO**  -- Creamos la tabla temporal, validando previamente la existencia de la tabla  select c1, c2  into #xyz\_tmp\_data  from base\_his..tabla\_historica1 a  where a.fecha between @e\_fecini and @e\_fecfin  and a.sucursal between @e\_sucini and @e\_sucfin  if @@error <> 0  begin  -- Validación de error  end  -- Accedemos a las otras tablas con el segmento de datos filtrado  select c1, c2  from #xyz\_tmp\_data  inner join base\_his..tabla\_historica2 b  on b,id = a.codigo  inner join base\_his..tabla\_historica3 c  on c,idhist3 = b.idhist2 |

**Cursores**

* Un cursor es usado cuando es necesario manipular datos a nivel de registros por una consulta que retorna muchos registros.
* Realizar actualizaciones, eliminación y / o inserción en una tabla usando una lógica orientada a conjuntos es casi siempre más rápido que usar un cursor.
* Se debe especificar el modo del cursor:
* Cuando el cursor no es actualizable usar **read only**
* Cuando el cursor se va a actualizar o eliminar usar: **update**
* Si dentro de un cursor es necesario salir del procesamiento con la instrucción return, es necesario cerrar y liberar los recursos del cursor.

|  |
| --- |
| while @@fetch\_status = 0  begin  ...  if @v\_data = 'X'  begin  close cursor c\_datos  deallocate cursor c\_datos  return  end  fetch c\_datos cursor @v\_dato1, @v\_dato2  end |

* Si dentro de un cursor es necesario salir del procesamiento con la instrucción return, es necesario cerrar y liberar los recursos del cursor.

**Consideración especial de los cursores**

* En producción, se ha detectado que un cursor puede generar bloqueos de las tablas que intervienen a otros procesos, para resolver esta novedad existen dos formas de solventarlos:

|  |
| --- |
| 1. Se reemplaza el algoritmo a procesamiento de registro a registro por ciclos. Esta práctica ha sido implementada y pesar de estar en el manual de mejores prácticas de programación de SYBASE, **no se recomienda debido a que se ha demostrado que consume recursos de cpu**. |
| 1. Crear una tabla temporal con el set de datos y luego recorrer la tabla temporal a través del cursor. |

* Se observa un alto IO al ejecutar el cursor, lentitud al ejecutar fetch cursor, esto se debe a que el plan de ejecución de la sentencia que forma parte del cursor no es la óptima, para resolver esta novedad existen dos formas de solventarlos:

|  |
| --- |
| 1. Forzar índices a las tablas que accede. Practica no recomendable. |
| 1. Crear una tabla temporal con el set de datos y luego recorrer la tabla temporal a través del cursor, de esta manera se podrá observar que el plan de ejecución es óptimo. |

* Cuando se declara un cursor como insensitive, el motor internamente al abrir el cursor crea una tabla temporal y copia los datos. Esta tabla temporal es la que recorre el cursor fila a fila.

**Uso de Transacciones**

* No se permite el uso de transacción para una sola sentencia DML.

|  |
| --- |
| **No es correcto:**  begin tran  insert into cobis..cl\_ente  (en\_ente)  values(  @e\_ente)  if @@error <> 0  begin  rollback  return  end  commit tran |
| **Es correcto:**  insert into cobis..cl\_ente  (en\_ente)  values(  @e\_ente)  if @@error <> 0  begin  set @s\_mensaje = 'Error al insertar datos'  return  end |

* La transacción debe ser lo más corta posible.
* Evitemos dentro de una transacción crear una tabla temporal.

|  |
| --- |
| **No es correcto:**  begin tran  create table #tmp (id int)  **Mensaje:**  Msg 2762, Level 16, State 3  The 'CREATE TABLE' command is not allowed within a multi-statement transaction in the 'tempdb5' database. |
| **No es correcto:**  begin tran  select id into #tmp from md\_tab01  **Mensaje:**  Msg 226, Level 16, State 1  SELECT INTO command not allowed within multi-statement transaction. |

* Cuando se inicia una transacción en un procedimiento almacenado obligatoriamente se la debe liberar o confirmar la transacción y luego asignar datos a variables locales o invocar a procedimientos almacenados de control de error.

|  |
| --- |
| begin tran  ...  update <tabla> ...  if @@error <> 0  begin  rollback  set @s\_mensaje = 'Error al actualizar datos'  exec sp\_error ...  return  end  ...  commit tran |

* Se permite evaluar la existencia de una transacción con el siguiente algoritmo:

|  |
| --- |
| if @@trancount > 0  begin  print 'Existe transacción activa'  end |

* Para liberar todas las transacciones abiertas solo se debe ejecutar la instrucción rollback

|  |
| --- |
| If @@trancount > 0  begin  rollback  return  end |

* Para confirmar todas las transacciones abiertas, evitando dejar transacciones abiertas, podemos ejecutar el siguiente algoritmo:

|  |
| --- |
| While @@trancount > 0  begin  commit tran  end |

**Procesos Batch**

* No se permite el uso del comando: dump tran <base> with no\_log.
* No se permite el uso del comando: DBCC SHRINKFILE.

Al usar los comandos anteriores corrompen procesos de replicación.

**Comparación de campos NULOS**

* Para evaluar si un campo es nulo se usará la sentencia is null.

|  |
| --- |
| **No se permite**  campo1 = null  campo2 <> null |
| **Se permite**  campo is null  campo is not null |