PL SQL - Estándares de Programación

Versión 3.9

16/07/18

Revisión

Fecha	Versión	Descripción	Autor
Mayo	1.0	Creación del documento	Juan Maya
06/08/2004	2.0	Promoción versión estable	Juan M. Tenorio Hernández
15/10/2004	2.1	Modificaciones a los estándares	Carlos Bernal Chávez
22/10/2004	2.2	Modificaciones a los estándares	Carlos Bernal Chávez
22/10/2004	3.0	Promoción a integración	Gerardo Villaseñor V.
09/11/2004	3.1	Se agregó sección Métodos de acceso y consultas	Juvenal Hernández Carrillo.
11/11/2004	3.2	Se agregaron reglas específicas para el desarrollo de la Regional de Crédito	Gilberto Canseco García
15/11/2004	3.3	Se estandarizó el formato y corrigieron algunos errores	Gilberto Canseco García
30/05/2005	3.4	Se corrigió la nomenclatura para el nombrado de las funciones	Abigail Salazar Torcuato.
01/06/2011	3.5	Modificación de Estándares	Dba Oracle
19/05/2014	3.6	Modificación deEstándares	Dba Oracle
23/03/2015	3.7	Modificación de Estándares	Dba Oracle
18/11/2017	3.8	Corrección de Estándares	Luis Iván Miranda C.
16/07/2018	3.9	Corrección de Estándares	Luis Iván Miranda C.

Aprobaciones

Contenido

1. Objetivo

Dar a conocer la definición de un conjunto de normas y estándares para el desarrollo de programas PL. Dentro de este marco se definirán mecanismos de trabajo así como normas de construcción y documentación de programas.

2. Creacion de Funciones, Paquetes y Store Procedure

Los nuevos objetos creados en la base de datos deberán seguir la siguiente nomenclatura de nombrado:

Objeto	Prefijo
Tabla	TA
Tabla Temporal	TT
Stored Procedure	SP
Trigger	TR
Función	FN
Paquete	PA
Vista	VI
Vista Materializada	VM
Índice	IX
Secuencia	SE
Tablespace	TS
Llave Primaria	PK
Llave Foránea	FK
Sinónimo	SI
Esquema	SC
DB Link	DL

NOTA: En el caso de las tablas previamente existentes, se conservará su nombre; la nomenclatura anterior aplicará para nuevas tablas y demás objetos.

3. Declaración de variables

El objetivo de esta sección es el de mostrar los puntos generales que deben tomarse en cuenta para tener programas estandarizados, de fácil lectura y de fácil mantenimiento. Para ello deben considerarse los siguientes puntos:

3.1 Longitud máxima de variables e identificadores

La longitud máxima de un identificador PL/SQL es de 20 caracteres.

3.2 Identificadores de PL/SQL

Se utilizará la siguiente nomenclatura (solo si el objeto es nuevo) para los identificadores en bloques PL/SQL:

FORMA DE DECLARACION	DESCRIPCION
vl_<nombre>	Variable local a una función o procedimiento cuyo tipo de datos es escalar
vg_<nombre>	Variable global declarada en un package cuyo tipo de datos es escalar
cur_<nombre>	Declaración de cursor explícito
rec_<nombre>	Declaración de un tipo de dato Record.
tab_<nombre>	Declaración de un tipo de dato PL/SQL Table.
obj_<nombre>	Referencia a tipo de objeto definido por el usuario
arr_<nombre>	Declaración de un tipo de dato PL/SQL Varray
typ_<nombre>	Tipo definido por el usuario
pa_<nombre>	Parámetro de una función o procedimiento cuyo tipo de datos es escalar
prec_<nombre>	Parámetro de una función o procedimiento cuyo tipo de datos es Record
vlrec_<nombre>	Variable local de una función o procedimiento cuyo tipo de datos es Record
vgrec_<nombre>	Variable global en un package cuyo tipo de datos es Record
ptab_<nombre>	Parámetro de una función o procedimiento cuyo tipo de datos es PL/SQL Table
vltab_<nombre>	Variable local de una función o procedimiento cuyo tipo de datos es PL/SQL Table
Vgtab_<nombre>	Variable global en un package cuyo tipo de datos es PL/SQL Table
parr_<nombre>	Parámetro de una función o procedimiento cuyo tipo de datos es PL/SQL Varray
vlva_<nombre>	Variable local de una función o procedimiento cuyo tipo de datos es PL/SQL Varray
viva_<nombre>	Variable global en un package cuyo tipo de datos es PL/SQL Varray
csl_<nombre>	Constantes local definidas en una función o procedimiento
csg_<nombre>	Constantes global definida en un package
rcl_<nombre>	Declaración de ref cursor local
rcg_<nombre>	Declaración de ref cursor global en un package
exc_<nombre>	Excepción definida por el usuario

3.3 Escritura de identificadores

Los nombres de variables se escriben con una mezcla de mayúsculas y minúsculas, poniendo en mayúsculas la primera letra de cada palabra y componente.

Ejemplo:

```
Vl_NombreEmpleado VARCHAR2(50); -- Comentario
Vl_ApellidoPaternoEmpleado VARCHAR2(50); -- Comentario
```

3.4 Declaración de identificadores usando %TYPE y %ROWTYPE

- **%TYPE**

Si la variable que utiliza en un bloque PL/SQL es asociado a un tipo de dato correspondiente a una tabla, utilizar siempre Tabla.Columna%TYPE;

Ejemplo:

```
Vl_SeatsSold GROSS_RECEIPT.SEATSSOLD%TYPE;
```

- **%ROWTYPE**

Por otro lado, cuando se defina una variable tipo registro igual a la estructura de una tabla, utilizar siempre NombreTabla%ROWTYPE;

Ejemplo:

```
recSeatsSold GROSS_RECEIPT%ROWTYPE;
```

4. Lineamientos de programación

Todo programa desarrollado en PL/SQL debe de estar debidamente documentado siguiendo además las indicaciones que se describen a continuación:

4.1 Generales

4.1.1 Front End

1. Se emplearán funciones para la generación de los datos de salida que despliegan las páginas de la Regional de Crédito y siempre deberán ir calificadas al esquema que le corresponde.

   Ejemplo:

```
CREATE OR REPLACE FUNCTION RCREDITO.FN000454
   (pa_Pais IN NUMBER
   ,pa_Canal IN NUMBER
   ,pa_Sucursal IN NUMBER)
```

2. El nombrado de las funciones tendrá 25 caracteres y la siguiente forma:

   Ejemplo: FNCLCMLS0001

```
FN (Indica si es un función)
¿? (Las letras de la primera clasificación)
¿? (Segunda Clasificación)
¿  (Línea – Es una consulta operativa
    Batch – Se utilizan en un proceso batch)
¿  (Tipo de función que realiza DML, DDL
      G diversas (consulta, inserta, borrar, actualiza,
      cualquier combinación)
      S consulta
      I inserción
      D borrar
      U actualización)
9999  (Consecutivo que se toma desde el primer nivel de la
      clasificación a la que pertenece la función)

Con base en la siguiente estructura:
Colocación
CL
      Solicitudes             SL
      Cambaceo                CM
      Guardadito Credito      GC
```

```
      Productos                PR

Cartera
CR
      Estadisticos             ES
      BalanceoCuentas          BC
      Zonificación             ZN
      Al día                   AD


Cobranza
CB
      Ruta                     RT
      Cargo Automatico         CA
      RMD                      RM
      Gestiones                GS
      Estadisticos             ES


Clientes
CN
      Buro                     BR
      Digitalización           DG
      Clientes                 CL

Administrar
AD
      Activos                  AC
      RecursosHumanos          RH
      Utilerias                UT
      Alertas                  AL
      Catalogos                CT
      Sistema                  SS
      CatalogosPaginas         CP

SIE
SI
      Colocacion               CL
      Cartera                  CR
      Cobranza                 CB
      Caja                     CJ
      Captacion                CP
      Clientes                 CN
      Mapas                    MP
      Administrar              AD
```


```
Autorizador
AT
      Conciliación            CN


Compensaciones
CM
      Colocacion              CL
      Cobranza                CB
      Captación               CP
      Caja                    CJ
      Cartera                 CR
      Tarjeta                 TR

Autorizador
TR
      Prosa                   PR
```

Ejemplo:

FNCLCMLS0001

3. Las funciones regresarán un tipo de dato REF CURSOR, es decir, un cursor con la información a desplegar en la página. El tipo de dato se encuentra especificado en el paquete PATYPES.

Ejemplo:

```
      CREATE OR REPLACE FUNCTION RCREDITO. FNCLCMLS0001
   (pa_Pais IN NUMBER
   ,pa_Canal IN NUMBER
   ,pa_Sucursal IN NUMBER)
RETURN PATYPES.rcgCursor;
IS
...
   rclCursorSalida PATYPES.rcgCursor; --Cursor de Salida
BEGIN
...
...
   RETURN rclCursorSalida;
...
EXCEPTION
...
END FN00454;
```

4. El cursor de salida será llenado con la consulta necesaria según el requerimiento de la función.

   Ejemplo:

```
OPEN rclCursorSalida FOR
   SELECT FCCOLONIA
     FROM CODIGO_POSTAL
    WHERE FIPAISID = pa_Pais
      AND FICANALID = pa_Canal
      AND FISUCURSALID = pa_Sucursal
 ORDER BY FCCOLONIA;

RETURN rclCursorSalida;
```

5. Se empleará la sección de Excepciones en todas las funciones. En la sección se incluirá la acción ROLLBACK en caso de que se hayan insertado, borrado o actualizado registros; la llamada al stored procedure SPREGISTRAERROR, el cual registrará el mensaje de la excepción en una bitácora; y se deberá regresar un cursor de una columna con valor cero como código de error en la ejecución.

   Ejemplo:

```
CREATE OR REPLACE FUNCTION RCREDITO.FNCLCMLS0001
...
EXCEPTION
WHEN OTHERS THEN
     ROLLBACK;
     SPREGISTRAERROR(SYSDATE --Fecha en que se genera el error
                     ,SQLCODE       --Código del error
                     ,SQLERRM       --Mensaje del error
                     ,'FNCLCMLS0001');    --Nombre   de   la
                   función
     RAISE_APPLICATION_ERROR(-20001,SUBSTR(SQLERRM,1,512));
     RETURN rclCursorSalida;
END FNCLCMLS0001;
```

4.2 Documentación en el encabezado de objetos

Al principio de cada objeto, debe incluirse un encabezado que contenga los siguientes datos.

```
/*************************************************************
Proyecto: (Obligatorio. Nombre del proyecto)
Descripción: (Obligatorio. Descripción de lo que hace el programa)
Parámetros de entrada: (Obligatorio cuando aplique. ¿Para que se utiliza?)
Parámetros de salida: (Obligatorio cuando aplique. ¿Para que se utiliza?)
Valor de retorno: (Obligatorio en funciones. Documentar resultado)
Parámetros de entrada-salida: (Obligatorio cuando aplique.¿Para que se
utiliza?)
Precondiciones: (Obligatorio. Reglas de negocio y restricciones)
Creador: (Obligatorio. Nombre del programador que crea el programa)
Fecha de creación: (Obligatorio. Fecha en que se crea el programa)
*************************************************************/
```

Se deben llenar los apartados que apliquen al objeto, si alguno no aplicara, dejarlo en blanco.

Consideraciones:

- El texto debe colocarse al principio de la definición de la lógica del objeto, después de la palabra reservada IS ó AS.
- La palabra reservada IS ó AS debe ir en una sola línea después de la declaración de parámetros.

4.3 Escribir programas PL/SQL con pocas líneas de código

Los programas escritos en PL deberán ser en la medida de lo posible de pocas líneas de instrucciones, esto con la finalidad de que el código sea de fácil mantenimiento y lectura.

4.4 Alineación en la codificación de parámetros

Los parámetros en los procedimientos, funciones y paquetes deberán ser codificados uno en cada línea incluyendo la coma, además de que deberán colocarse debajo del nombre del objeto y estar identados.

Ejemplo:

```
CREATE OR REPLACE PROCEDURE SPNom_Proc
   (pa_Pais IN NUMBER
   ,pa_Canal IN NUMBER
   ,pa_SUCURSAL IN NUMBER)
IS
   <Declaraciones>
BEGIN
   <Sentencias>
END SPNom_Proc;
```

4.5 INSERT y UPDATES

Siempre que se haga un INSERT ó UPDATE, nombrar explícitamente las columnas en las cuales se inserta o actualiza.

Los campos deberán ser codificados uno en cada línea incluyendo la coma.

Ejemplo:

- ***INSERT***

```
INSERT INTO NombreTabla (campo1
                        ,campo2
                        ,campo3)
                 VALUES (vl_Campo1
                        ,vl_Campo2
                        ,vl_Campo3);
```

- ***UPDATE***

```
UPDATE EMPLEADO
   SET NOMBRE = vl_NuevoNombreEmpleado
      ,APELLIDOPAT = vl_NuevoApellidoPaterno
 WHERE EMPID = vl_EmpId;
```

Siempre que se realice una función DML, se deberá regresar en el cursor los elementos

Código	Descripción
1	Exitoso
2	No exitoso

5. Convenciones de escritura

Todo programa desarrollado en PL/SQL debe de estar completamente documentado siguiendo detalladamente cada uno de los siguientes puntos:

5.1 Identación

La identación dentro de un bloque PL/SQL es de 3 caracteres, esto es, tres espacios en blanco, por lo tanto queda prohibida la tabulación.

5.2 Alineación de instrucciones SQL

Cuando se codifique una sentencia SQL, las palabras reservadas deben estar alineadas en la última letra de la primera instrucción.

Ejemplo:

```
SELECT EMP.ID
  INTO vl_EmpIDTemp
  FROM EMPLEADO EMP
 WHERE EMP.CIUDAD = Vl_Ciud
   AND EMP.ID = vl_EmpId;
```

5.3 Declaración de cursores

La declaración de un cursor debe hacerse de la manera siguiente:

Ejemplo:

```
CURSOR NombreCursor IS
   SELECT EMP.ID
     FROM EMPLEADO EMP
    WHERE EMP.CIUDAD = Vl_Ciud
      AND EMP.ID = vl_EmpId
 ORDER BY 1;
```

5.4 Condiciones del WHERE

En la cláusula WHERE de una sentencia SQL cada condición se escribe en una línea y el operador lógico al comienzo de la misma. Las palabras reservadas deben estar alineadas en la última letra de la primera instrucción.

Ejemplo:

```
    SELECT EMP.ID
          ,CATP.Puesto
      INTO vl_EmpID
          ,vl_Puesto
      FROM EMPLEADO EMP
          ,CAT_PUESTO CATP
     WHERE EMP.CIUDAD = Vl_Ciud
       AND EMP.ID = vl_EmpId
       AND CATP.Puesto = EMP.Impuesto
ORDER BY 2, 1;
```

No se acepta código duro

5.5 Una línea por cada campo y/o instrucción SQL

Cuando se codifique FROM, SELECT, UPDATE, INTO, ORDER BY, GROUP BY y todos aquellos casos en los que se listen más de un campo, variable ó tabla, se deberá tener uno por línea y al principio de este la coma.

Ejemplo:

```
    SELECT FISUCURSAL
          ,FITRANNO
          ,FITRANTIPO
      INTO vl_fiSucursal
          ,vl_fiTranno
          ,vl_fiTranTipo
      FROM TRANCRECAB
     WHERE FIPAISID = vl_Pais
       AND FICANAL = vl_Canal
       AND FICSUCURSAL = vl_Sucursal
ORDER BY FICSUCURSAL
          ,FITRANNO;
```

5.6 Un parámetro por línea en llamados de procedimientos

Al mandar llamar un procedimiento, función ó paquete, los parámetros deberán codificarse uno por renglón incluyendo la coma.

Ejemplo:

```
VALIDANOMBRE(vl_Pais
            ,vl_Canal
            ,vl_Sucursal);
```

5.7 Ayuda de formato

Se recomienda usar TOAD para dar formato a los scripts, en caso de no contar con dicho software se puede usar SQL Developer.

Convenciones de formato

6.1 Fechas

El formato a utilizar en fechas para los parámetros de entrada y salida deberá ser el siguiente:

```
Día / Mes / Año
```

Ejemplos:

```
TO_DATE('10/11/2004', 'dd/mm/yyyy');
TO_CHAR('10/11/2004', 'dd/mm/yyyy');
```

6.2 Numéricos

1. En el caso de utilizar valores numéricos con decimales, se deberá especificar la precisión necesaria.

   Ejemplos:

   ```
   Vl_Total NUMBER(10,2);
   ```

2. Las cantidades monetarias serán retornadas en formato numérico (con decimales), pero sin formato de moneda. El formato de moneda será aplicado en los JSP's.

   Ejemplo:

   ```
   Incorrecto:    SELECT TO_CHAR(campo, '$999,999.99')
   Correcto:      SELECT campo
   ```

6. Mejores prácticas

7.1 Casos especiales

- Queda estrictamente prohibido la utilización de:
  1. La letra ñ y de todo tipo de caracteres especiales.
  2. El uso de código duro en cualquier parte de las sentencias
     a. Dependiendo el caso deberán usar variables bind o constantes
  2. El Uso de Querys Dinámicos
     Ejemplo 1:
     VL_QUERY:= ' SELECT /INSERT/UPDATE/DELETE campos
     FRM TABLA
     WHERE CAMPO =VlValor;'
     Ejemplo 2
     VL_QUERY:= ' SELECT /INSERT/UPDATE/DELETE campos
     FROM TABLA Partition ||VlPart||
     WHERE CAMPO1 =||VlValor||
     AND campo2 =||vSmena;'
  3. Uso de Vistas Dinamicas

Excepción para CODIGO DURO

Lo que venga dentro de las siguientes funciones SI puede llevar código duro:

- LPAD
- NVL
- RPAD
- REGEXP_REPLACE
- SUBSTR
- TRANSLATE

- TO_CHAR -- MASCARAS
- TO_DATE -- MASCARAS

7.2 Comentarizar declaración de variables

Delante de cada declaración de variable, se deberá poner un comentario simple describiendo el uso que se le va a dar a esta.

Ejemplo:

```
Vl_NomEmp VARCHAR2(50); -- Nombre empleado
Vl_ApPat VARCHAR2(50); -- Apellido paterno
```

7.3 Comentarizar código

No es necesario poner comentarios a cada instrucción, pero un comentario que explique el propósito del siguiente conjunto de instrucciones resultará útil. El

algoritmo utilizado puede resultar obvio a partir del propio código, así que es mejor describir el propósito del algoritmo, es decir, para qué se utilizarán los resultados. Los comentarios deben ser significativos y no volver a expresar lo que el propio código PL/SQL ya expresa.

Ejemplo:

El siguiente bloque es inválido porque el mismo código indica cual será la acción a realizar

```
vlValor := vlValor + 1;  -- Suma uno a vlvalor
```

7.4 Inicialización de variables

Inicializar siempre todas las variables que se ocupen en el programa antes de ser utilizadas.

Ejemplo:

```
CREATE OR REPLACE PROCEDURE SPVALIDA
AS
   Vl_Contador NUMBER(10); --Contador de registros válidos
BEGIN –Inicio del proceso de validación PROCVALIDA
   Vl_contador := 0;
   <Código>
END SPVALIDA; --Fin del proceso de validación SPVALIDA
```

7.5 Nombrado de alias

El nombrado de alias para tablas puede ponerse siguiendo el orden del abecedario (A, B, C... etc.) o puede tener una abreviatura significativa como EMP, DEPT, etc.

7.6 ORDER BY ó GROUP BY

En las instrucciones ORDER BY es válido utilizar la posición de la columna o el nombre explícito del campo.

Ejemplo:

```
  SELECT FICANALID
       ,FCTDADESC
    FROM CANAL
ORDER BY 1;

Ó

  SELECT FICANALID
       ,FCTDADESC
    FROM CANAL
ORDER BY FICANALID;
```

En las instrucciones GROUP BY siempre se debe emplear el nombre explícito del campo, o bien los campos calculados. Por sintaxis, no se puede emplear la posición de la columna.

Ejemplo:

```
Incorrecto:
  SELECT TA.FICANALID
        ,TA.FICANALID||' '||CA.FCCANALDESC
        ,COUNT(TA.FISUCURSAL)
    FROM TIENDA_CR TA
        ,CANAL CA
   WHERE TA.FICANALID = CA.FICANALID
GROUP BY 1
        ,2;

Correcto:
  SELECT TA.FICANALID
        ,TA.FICANALID||' '||CA.FCCANALDESC
        ,COUNT(TA.FISUCURSAL)
    FROM TIENDA_CR TA
        ,CANAL CA
   WHERE TA.FICANALID = CA.FICANALID
GROUP BY TA.FICANALID
        ,TA.FICANALID||' '||CA.FCCANALDESC;
```

7.7 Comentarización de instrucciones de control

A continuación de cada IF, WHILE ó instrucción de control o que involucre un bloque de código debe ir un comentario acerca de la condición que describa la acción a realizar. Por otra parte los operadores (AND, OR, etc) deben ir al comienzo de la línea y el THEN al comienzo de una nueva línea.

Ejemplo:

```
IF --el documento no esta verificado
   Vl_Verificado = FALSE
   AND <cond_1>
   AND <cond_2>
THEN  .....
   <Código>
ELSE  --el documento esta verificado
   <Código>
END IF; --fin de la validación
```

7.8 Inclusión del nombre del SP al final del mismo

El END final de cada unidad de programa debe ir seguido del nombre de la unidad de programa.

Ejemplo:

```
CREATE OR REPLACE PROCEDURE SPVALIDA
AS
BEGIN
   <Código del procedimiento>
END SPVALIDA;
```

7.9 Codificación de instrucciones largas

Las asignaciones de variable en caso que tengan que continuar en el renglón siguiente, deberán terminar el renglón previo con un operador (+,-,*,etc).

Ejemplo:

```
Vl_x := a + b + c +
        d + e.....;
```

Si se tratara de una concatenación, se haría de la siguiente manera:

Ejemplo:

```
Vl_NombreCompleto := EMP.NOMBRE ||
                     EMP.APELLIDOPATERNO ||
                     EMP.APELLIDOMATERNO;
```

7.10 Piezas de código estándares y reutilizables

En la medida de lo posibles, cada función, procedimiento, etc, deberá ser escrita de manera que su código pueda ser reutilizable en cualquier parte del sistema.

8. Métodos de acceso y consultas.

8.1 SELECT * en las consultas.

No hacer uso de `SELECT *` en las consultas. Siempre utilizar los nombres de las columnas requeridas después del enunciado SELECT. Esta técnica reduce IO en disco y menor tráfico en la red y por lo tanto mejor rendimiento. Además identifica cambios inesperados en la estructura de las tablas.

Excepción: Cuando se use la función PIVOT se debe usar select *

8.2 Utilización de ROWNUM.

Auxiliarse de `WHERE ROWNUM < n`, donde n es el límite de registros a extraer de la tabla. No debe llevar `ORDER BY`, de lo contrario se hará un `FULL TABLE SCAN`.

8.3 Palabras reservadas.

Todas las palabras reservadas de PL/SQL y SQL deberán escribirse en mayúscula, esto incluye funciones y procedimientos, paquetes, tipos predefinidos, nombres de tablas y de campos.

Ejemplo:

```
SELECT EMP.ID
  INTO vl_EmpID
  FROM EMPLEADO EMP
 WHERE EMP.CIUDAD = 'DF'
   AND EMP.ID = vl_EmpId;
```

8.4 Cálculos en columnas indexadas.

No hacer cálculos sobre una columna indexada, porque no utilizará el índice.

Ejemplo:

```
WHERE salary*5 > myvalue
```

8.5 Valores numéricos y caracter.

Siempre especificar valores numéricos en forma numérica y valores caracter en forma de caracter. Nunca mezclar tipos de datos en consultas Oracle, porque invalidan el

índice.
Si la columna es numérica, recordar no utilizar comillas.
Ejemplo:

```
Salary = 5000
```

Para columnas indexadas de tipo char, siempre utilizar comillas.
Ejemplo:

```
NumEmp = '912856'
```

8.6 Valores NULL en índice.

Evitar valores NULL en una columna indexada.

8.7 Subqueries y JOIN.

Evitar utilizar subqueries cuando se puede resolver con un JOIN.

8.8 Consideración sobre DECODE.

Utilizar la función DECODE para minimizar el número de veces que una tabla tiene que ser seleccionada.

8.9 FULL TABLE SCAN vs INDEX SCAN.

Si el query regresa más de un 20% de la los renglones de la tabla, utilizar un `FULL TABLE SCAN` en lugar de un `INDEX SCAN`.

8.10 Alias de tablas.

Siempre utilizar alias de tablas cundo se haga referencia a columnas, la llamada a una tabla siempre deberá ser calificada por el esquema en la que se encuentra dicha tabla. Ejemplo: Select fcempnum **from RCREDITO.**empleado_cr;

8.11 Consideraciones en índices.

Asumir un índice en la tabla `ADRESS(city, state)`.

La referencia a una columna índice no-principal, no utiliza el índice

```
WHERE STATE = 'TX'                      [Índice no utilizado]
WHERE CITY = 'DALLAS'                   [Índice utilizado]
WHERE STATE = 'TX' AND CITY = 'DALLAS'  [Índice utilizado]
```

NOT, != y <> deshabilita el uso de índice

```
WHERE STATE NOT IN ('TX', 'FL','OH')    [Índice no utilizado]
WHERE STATE != 'TX'                     [Índice no utilizado]
```

Referencia a valores NULL nunca utilizan índices

```
WHERE STATE IS NULL                     [Índice no utilizado]
WHERE STATE IS NOT NULL                 [Índice no utilizado]
```

Considere las siguientes expresiones:

```
WHERE substr(city,1,3) = 'DAL'          [Índice no utilizado]
```

```
WHERE CITY LIKE 'DAL%'                    [Índice utilizado]
WHERE CITY || state = 'DALLASTX'          [Índice no utilizado]
WHERE CITY = 'DALLAS' AND STATE = 'TX'    [Índice utilizado]
WHERE SALARY * 12 >= 24000                [Índice no utilizado]
WHERE SALARY >= 2000                      [Índex utilizado]
```

8.12 Evaluación de condiciones AND y OR.

Oracle evalúa condiciones no-indexadas ligadas por `AND` de abajo hacia arriba

```
Incorrecto: SELECT *
              FROM ADDRESS
             WHERE AREACODE = 972
               AND TYPE_NR = (SELECT SEQ_NR
                                FROM CODE_TABLE
                               WHERE TYPE = 'HOME')


Correcto:   SELECT *
              FROM ADDRESS
             WHERE TYPE_NR = (SELECT SEQ_NR
                                FROM CODE_TABLE
                               WHERE TYPE = 'HOME')
               AND AREACODE = 972
```

Oracle evalúa condiciones no indexadas ligadas por OR de arriba hacia abajo

```
Incorrecto: SELECT *
              FROM ADDRESS
             WHERE TYPE_NR = (SELECT SEQ_NR
                                FROM CODE_TABLE
                               WHERE TYPE = 'HOME')
                OR  AREACODE = 972
Correcto:   SELECT *
              FROM ADDRESS
             WHERE AREACODE = 972
                OR TYPE_NR = (SELECT SEQ_NR
                                FROM CODE_TABLE
                               WHERE TYPE = 'HOME')
```

8.13 Orden de tablas en cláusula FROM.

Recordar que el optimizador basado en reglas de Oracle, revisa el orden de los nombres de tabla en la cláusula `FROM` para determinar la tabla que manda. Asegurarse siempre que la última tabla especificada en la cláusula `FROM` es la tabla que regresa el menor número de renglones. En otras palabras, especificar múltiples tablas, poniendo la tabla con el conjunto de datos más grande a regresar al principio de la cláusula `FROM`. No afecta bajo el optimizador basado en costos.

Ejemplo:

```
SELECT * FROM larger table, smaller table
```

```
SELECT * FROM larger table, smaller table, smallest table
SELECT * FROM larger table, smaller table, associative table
```

8.14 Clausula WHERE

Si hay una cláusula WHERE que incluye expresiones conectadas por dos o más operadores AND, se evalúan de izquierda a derecha en el orden en el que están escritos. Como resultado, toma en cuenta las siguientes consideraciones:

- Ubica primero la instrucción AND menos probable. De esta forma si la expresión AND es falsa, la cláusula terminará inmediatamente reduciendo el tiempo de ejecución.
- Si ambas partes de la expresión AND son igualmente probables de ser falsas, pon la expresión AND menos compleja primero, si es falsa, menos trabajo tomará evaluar toda la expresión.

8.15 IN/UNION vs. OR.

Si las columnas no están indexadas, unir con OR.

Si las columnas están indexadas, utilizar IN ó UNION en lugar de OR

Ejemplo:

```
Incorrecto: SELECT *
              FROM ADDRESS
             WHERE STATE = 'TX'
                OR STATE = 'FL'
                OR STATE = 'OH'


Correcto:   SELECT *
              FROM ADDRESS
             WHERE STATE IN ('TX','FL','OH')
```

Ejemplo UNION:

```
Incorrecto: SELECT *
              FROM ADDRESS
             WHERE STATE = 'TX'
                OR AREACODE = 972
Correcto:   SELECT *
              FROM ADDRESS
             WHERE STATE = 'TX'
             UNION
            SELECT *
              FROM ADDRESS
             WHERE AREACODE = 972
```

Si la instrucción SELECT incluye un operador IN con una lista de valores a ser probadas en el query, ordena la lista de valores de tal forma que los valores más frecuentes se encuentren al principio de la lista, y los valores menos frecuentes estén ubicados al final de la lista. Esto puede mejorar el desempeño porque la opción IN

regresa TRUE tan pronto como cualquiera de los valores en la lista produce un match. Entre más rápido el match es realizado, el query se completa más rápidamente.

8.16 EXIST vs. Table Join.

Utilizar table join en vez de `EXIST` sub-query.

Cuando el porcentaje de los renglones regresados del outer sub-query es alto.

Utilizar `EXIST` sub-query en vez de table join

Cuando el procentaje de los renglones regresados por el outer sub-query es bajo

Cuando exista la opción de usar IN o EXIST en una sentencia, usar la cláusula EXIST, ésta es usualmente más eficiente y se realiza más rápidamente.

8.17 Joins que utilizan DISTINCT.

Evitar joins que utilicen `DISTINCT`, en su lugar utilizar `EXIST` sub-query

```
Incorrecto:  SELECT DISTINCT DEPTNO
                  , DEPTNAME
               FROM EMP
                  , DEPT
              WHERE EMP.DEPTNO = DEPT.DEPTNO

Correcto:    SELECT DEPTNO
                  , DEPTNAME
               FROM DEPT
              WHERE EXISTS (SELECT 'X'
                              FROM EMP
                             WHERE EMP.DEPTNO = DEPT.DEPTNO)
```

8.18 NOT IN vs. NOT EXIST.

Evitar sub-queries que utilizan `NOT IN`, utilizar en su lugar `NOT EXIST`

```
Incorrecto: SELECT DEPTNO
              FROM EMP
             WHERE DEPTNO NOT IN (SELECT DEPTNO
                                    FROM DEPT
                                   WHERE DEPTSTATUS = 'A')

Correcto:   SELECT DEPTNO
              FROM EMP
             WHERE NOT EXISTS (SELECT 'X'
                                 FROM DEPT
                                WHERE DEPTSTATUS = 'A'
                                  AND DEPT.DEPTNO = EMP.DEPTNO)
```

8.19 Contando renglones con COUNT.

Cuando se cuenten renglones, utilizar `COUNT` en la columna indexada o asterisco

```
SELECT COUNT(INDEXED_COLUMN) FROM TABLE     [Más eficiente]
SELECT COUNT(ROWID) FROM TABLE              [Eficiente]
SELECT COUNT(1) FROM TABLE                  [Eficiente]
SELECT COUNT(*) FROM TABLE                  [Evitar]
```

8.20 Sentencia HAVING

Si la instrucción SELECT contiene la cláusula WHERE, escribir el query de tal forma que la cláusula WHERE haga la mayoría de trabajo (removiendo registros indeseados) en lugar de que la cláusula HAVING haga el trabajo de remover los renglones indeseados.

Usar la cláusula WHERE apropiadamente puede eliminar registros innecesarios antes de realizar el GROUP BY el HAVING, reduciendo el trabajo necesario y mejorando el performance.

9. Limitaciones

1. No se validara los Stored Procedures, Packages, Function hasta no tener creados todos los objetos a los que hacen referencia (incluyendo llamadas a tablas, SPs, funciones, paquetes y TYPES).
2. Cada modificación a los Stored Procedures, Packages, Function se validara nuevamente los costos, en caso de tener costos altos se deberá de realizar la optimización correspondiente.
3. Está prohibido usar palabras reservadas de oracle para nombrar objetos o campos de una tabla.
4. El número máximo de índices por tabla es de 5.
5. Se debe evitar indexar más de una vez los campos de la tabla para evitar se generen varios planes de ejecución.
6. No se permiten los índices BITMAP para bases de datos transaccionales.
7. El número máximo de campos por tabla es de 40.
8. Se revisara que todas las variables usadas estén declaradas.
9. Longitud máxima en secuencias, sinónimos, triggers, tablas e índices es de 15 Caracteres.
10. La longitud máxima del nombre en procedimientos, paquetes, funciones será de 25 Caracteres.
11. No se otorgan permisos DML(Insert, Delete, Update) a usuarios, solo EXCECUTE.
12. No se permiten los grants de create session o connect a esquemas.
13. Después de cada Sentencia DML en los Store deberá llevar Commit y en la Excepción Rollback.
14. Todos los SPs, Funciones o Paquetes deben llevar manejo de excepciones.
15. Siempre deberán estar calificados todos los objetos.
16. Todo cambio realizado en objetos de producción deberán aplicar los estándares mencionados.
17. No se permiten realizar más de 10 joins en los stored Procedures.
18. No se permite realizar más de 10 subquerys.

19. El número máximo de líneas es de 400 en stored procedures y funciones con formato de TOAD.
20. El número máximo de líneas es de 800 en Packages con formato de TOAD.
21. El numero máximo de líneas en modificaciones de procedimientos, funciones y paquetes ya existentes es de 1000 líneas con formato de TOAD.
22. En caso de que el desarrollador no cuente con TOAD para dar formato a sus scripts se puede usar SQL Developer.
23. El execute immediate solo puede ser utilizado en Truncate a las tablas y para alterar a nivel sesión el parámetro nls_date_format.
24. Esta permitido que en las bases de datos de DWBDPROD y PFBDPROD ejecuten estadísticas sobre tablas únicamente con nuestro formato a través de un SP o Paquete.
25. Las secuencias deben llevar cache 20 o default en bases de datos en RAC, a menos de que la aplicación requiera de forma forzosa mantener la numeración sin gaps.
26. Evitar el uso de los tipos de datos char o varchar, usar siempre que sea posible varchar2.
27. En caso de que el usuario solicite más de 5 scripts por correo, tendrá que ser a través de folio de simbanet.
28. Toda información resultado de un query debe ser solicitada por folio.
29. El estándar de los password para un usuario nuevo o modificación es de 15 caracteres (mayúsculas, minúsculas, números y caracteres especiales).
30. Al revisar querys que hacen referencia a diferentes esquemas revisar si se tienen los permisos necesarios entre esquemas, en caso de que no se tengan indicarle al usuario que debe solicitarlos al inicio de su script.
31. La creación de FK entre diferentes esquemas revisar si se tienen los permisos necesarios entre esquemas, en caso de que no se tengan indicarle al usuario que debe solicitarlos al inicio de su script.
32. En caso de requerir información productiva por folio de simbanet solo se pueden enviar 50 registros, en caso de requerir mas información se tendrá que contar con la autorización del gerente de base de datos.
33. Los DBA's revisaran los planes de ejecución y costos.
34. La lógica y funcionalidad de los Store son responsabilidad del desarrollador.
35. En creación de usuarios nuevos se pedirán las ips desde donde se va a realizar la conexión, y el número de empleado del responsable, para agregar esta información a la tabla de seguridad.
36. No se permite el uso de HINTs en el código productivo de Sps, Funciones, Paquetes, etc. Salvo casos excepcionales.
37. NO está permitido el uso de triggers, vistas, ni vistas materializadas.

10. Excepciones

LA BD BANKIUSA SE LE PERMITEN EN FUNCIONES HASTA 700 LINEAS.
LA BD DE REGIONAL SE LE PERMITEN EN FUNCIONES Y PROCEDIMIENTOS HASTA 1000 LINEAS Y EN PAQUETES 1500.
LA BD HTBDPROD SE LE PERMITEN FUNCIONES DE HASTA 500 LINEAS Y SE DEBE VALIDAR EL FORMATO Y NUMERO DE LINEAS CON SQL DEVELOPER.

El numero de líneas permitido está basado en Toad 10, si se usa el TOAD 12 se aceptan 100 líneas mas de las que indica el documento.