RESUMEN PL/SQL

La unidad básica en PL/SQL es el *bloque*. Todos los programas de PL/SQL están ompuestos por bloques que pueden estar anidados.

Estructura de un Bloque:

```
DECLARE
/*Aquí se declaran las variables, tipos cursores y subprogramas locales*/
BEGIN
/* Aquí se incluyen las órdenes SQL y PL/SQL, es obligatoria y debe contener, al menos una orden ejecutable */
END;
EXCEPTION
/* Sección para el manejo de excepciones (errores)*/
END;
```

Podemos crear diferentes tipos de bloques:

Bloques anónimos: Se construyen de forma dinámica y se suelen ejecutar una sola vez.
Bloques nominados: Igual que los anónimos pero con una etiqueta que les da nombre.
Subprogramas: Procedimientos, paquetes y funciones, almacenados en la BD y que se ejecutan en múltiples ocasiones. Los subprogramas se ejecutarán mediante una llamada.
Disparadores ("Triggers"): Bloques nominados que se almacenan en la BD y se ejecutan ante algún suceso.

Tipos de datos

•Incluye los tipos de datos y operadores de SQL.

ESCALARES:
- **NUMBER**
- BINARY-INTEGER
- CHAR
- **VARCHAR2**
- LONG
- RAW
- **BOOLEAN**
- **DATE**
- ROWID

COMPUESTOS
- **REGISTROS**
- **ARRAYS**

Podemos declarar variables, cada una debe tener su tipo asociado.
Las variables pueden ser de los mismos tipos que las columnas de una base de datos:

```
DECLARE
v_NombreEstudiante VARCHAR2(20);
v_FechaActual DATE;
v_Puntos NUMBER(3);
```

_ También existen otros tipos adicionales (enteros binarios y lógicos):

```
DECLARE
v_ContadorBucle BINARY_INTEGER;
v_Registrado BOOLEAN;
```

- **BINARY_INTEGER:** Se usa para almacenar valores que sólo van a ser utilizados en cálculos y no se van a almacenar en la BBDD.
- **BOOLEAN:** Puenden contener los valores TRUE,FALSE o NULL.

  Sintaxis: *nombre_variable tipo* [CONSTANT] [NOT NULL] [:=*valor*]

_Además admite tipos definidos por el usuario (tablas y registros):

_También admite tipos de objetos

_ El atributo %TYPE permite declarar una variable del mismo tipo que otra ya existente, especialmente útil para declarar variables del mismo tipo que atributos de una tabla.

*NombreVariable variableReferencia***%TYPE;** *NombreVarRef RegReferencia***%ROWTYPE;**

Identificadores

Los identificadores válidos empiezan por una letra que puede ser seguida de una secuencia de caracteres que puede incluir letras, números,$,_ ,#. La longitud máxima de un identificador es de 30 caracteres.

Literales

Podremos utilizar literales numéricos (enteros y reales), booleanos (TRUE,FALSE Y NULL) y de carácter (uno o más caracteres delimitados por comillas simples).

Registros y Tablas

PL/SQL permite utilizar tipos compuestos definidos por el usuario (Registros y Tablas)

Registros

La sintaxis general para definir un tipo registro es:

TYPE *tipo_registro* **IS RECORD(**
campo1 tipo1 [**NOT NULL**][**:=** *expr1*],
campo2 tipo2 [**NOT NULL**][**:=** *expr2*],
.....
campoN tipoN [**NOT NULL**][**:=** *exprN*]);

Ejemplo**:**

DECLARE
TYPE *t_EjRegistro* **IS RECORD(**
Cont **NUMBER (4),**
Nombre **VARCHAR(10) NOT NULL**,
Fecha **DATE,**
Descripcion **VARCHAR (45)**);

v_Ejemplo1 t_EjRegistro;
v_Ejemplo2 t_Ejregistro;

_ Para hacer referencia a los campos de un registro se utiliza el punto (.)

nombre_registro.nombre_campo.

_ Para poder asignar un registro a otro ambos deben ser del mismo tipo.

_ También se pueden asignar valores a un registro completo mediante la orden SELECT que extraería datos de la BD y los almacenaría en el registro.

Es bastante habitual definir un registro en PL/SQL con los mismos tipos que una fila de una base de datos. Esto se puede realizar directamente con el operador **%ROWTYPE**

Ejemplo:
```
DECLARE
v_RegAlumno alumno%ROWTYPE;
…..
```

Tablas:

La sintaxis general para definir una tabla es:
***TYPE** tipotabla* **IS TABLE *OF** tipo* **INDEX BY BINARY_INTEGER**

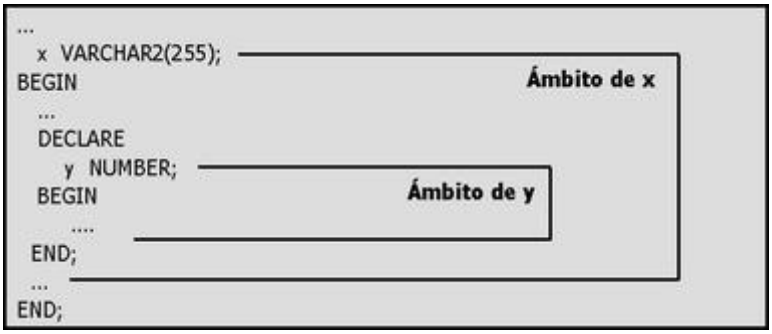
Ejemplos:
***TYPE** tipotabla* **IS TABLE *OF** tipo* **INDEX BY BINARY_INTEGER**

```
DELCARE
  TYPE t_tablaNombres IS TABLE OF students.first_name%TYPE
     INDEX BY BINARY INTEGER;
  TYPE t_tablaFechas IS TABLE OF DATE
    INDEX BY BINARY INTEGER;
 v_Nombres t_tablaNombres;
 v fechas t tablaFechas;
…..
```

Asignar valor a un elemento: MiTabla(8):=‘SMITH’;
Para referirnos a un elemento, ponemos el índice entre paréntesis tras el nombre de la tabla

- Métodos que permiten realizar todas las operaciones necesarias con tablas :
  - EXISTS(n): Devuelve TRUE si existe el elemento n.
  - COUNT: Devuelve el número de elementos de la tabla.
  - FIRST: Devuelve el índice del primer elemento.
  - LAST: Devuelve el índice del último elemento.
  - PRIOR(n): Devuelve el índice del elemento anterior a n. Si ya está en el primero devuelve NULL
  - NEXT(n): Devuelve el índice del elemento siguiente a n. Si ya está en el último devuelve NULL
  - DELETE: Borra todos los elementos.
  - DELETE(n): Borra el elemento de índice n.
- Llamada al método: NombreTabla.NombreMétodo(parámetros)

Bloques Anidados y Ámbito de la Variable:



Generalidades de PL/SQL

- Para hacer referencia a una variable de sustitución en PL/SQL debe anteponer a su nombre dos puntos (:) a modo de prefijo:

```
:return_code := 0;
IF credit_check_ok(acct_no) THEN
        :return_code := 1;
END IF;
```

- Directrices de Programación para facilitar el mantenimiento del código:
  - Documenta el código con comentarios.
  - Desarrollauna convención de mayúsculas/minúsculas para el código.
  - Desarrolla convenciones de nomenclatura para identificadores y otros objetos.
  - Sangra el código para facilitar la lectura.
  - Evita la ambigüedad entre variables locales, parámetros formales y nombres de columnas de las tablas de la B.D.
- Cuando trabaje con valores nulos puede evitar algunos de los errores más habituales si recuerda las siguientes reglas:
  - Las comparaciones en las que se incluyen valores nulos siempre resultan NULL.
  - Si se aplica el operador lógico NOT a un valor nulo resulta NULL.
  - En las sentencias de control condicionales, si la condición resulta NULL, no se ejecutarán las sentencias asociadas.

Sentencias SQL en PL/SQL

- SELECT … INTO .. recupera exactamente UNA fila.
- INSERT añade una fila.
- UPDATE modifica una o más filas existentes.
- DELETE suprime una o más filas existentes.
- COMMIT hace permanentes todas las modificaciones pendientes.
- ROLLBACK elimina todas las modificaciones pendientes.
- SAVEPOINT marca un punto intermedio en el procesamiento de las transacciones.

Sentencia SELECT

- Sintaxis:

```
SELECT lista_de_campos
INTO {nombre_variable [, nombre_variabe] ...
      | nombre_registro}
FROM tabla
WHERE condición;
```

- Recuerde, sólo se debe de recuperar una fila. Más de una fila provocará errores.

Ejemplo de SELECT

- Recupere la suma de los salarios de todos los empleados de un departamento específico:

```
DECLARE
        v_sum_sal   emp.sal%TYPE;  /* Aconsejable */
        V_deptno    NUMBER NOT NULL := 10;
BEGIN
        SELECT SUM(sal)  -- función de grupo
        INTO    v_sum_sal
        FROM    emp
        WHERE deptno = v_deptno;
END;
```

Inserción de Datos

- Añada nueva información sobre los empleados en la tabla emp:

```
DECLARE
        v_empno   emp.empno%TYPE;
BEGIN
        SELECT empno_sequence.NEXTVAL
        INTO    v_empno
        FROM   dual;

        INSERT INTO emp (empno, ename, job, deptno)
                VALUES (v_empno, 'HARDING', 'CLERK', 10);
END;
```

Actualización de Datos

- Aumente el salario de todos los empleados de la tabla emp que son Analistas:

```
DECLARE
        v_sal_increase  emp.sal%TYPE := 2000;
BEGIN
        UPDATE  emp
        SET       sal = sal + v_sal_increase
        WHERE  job = 'ANALYST';
END;
```

Supresión de Datos

- Suprima filas pertenecientes al departamento 10 de la tabla emp:

```
DECLARE
        v_emp  deptno.emp%TYPE := 10;
BEGIN
        DELETE FROM emp
        WHERE deptno = v_deptno;
END;
```

Control de Transacciones

- COMMIT finaliza la transacción actual realizando todos los cambios pendientes en la B.D.

```
COMMIT [WORK];
```

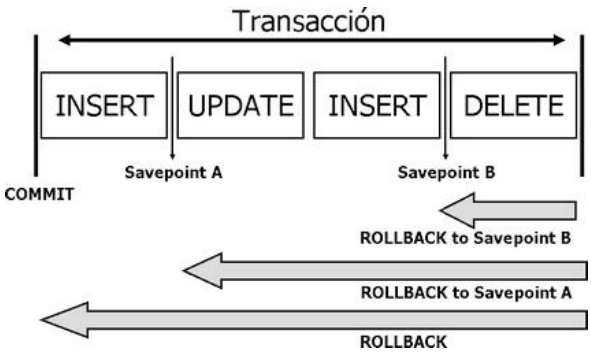
Donde WORK está para la conformidad con los estándares ANSI

- ROLLBACK finaliza la transacción actual desechando todos los cambios pendientes.

```
ROLLBACK [WORK];
```

Donde WORK está para la conformidad con los estándares ANSI

Control de Transacciones



Control Flujo de Ejecución

- Puede modificar el flujo lógico de sentencias utilizando sentencias IF condicionales y estructuras de control de bucles.
- Sentencias IF condicionales:
    - IF-THEN-END IF;
    - IF-THEN-ELSE-END IF;
    - IF-THEN-ELSIF-END IF;
    - IF-THEN-ELSIF-ELSE-END IF;

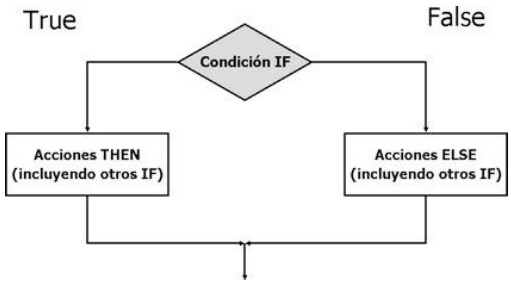
- Control de bucles:
    - Bucle básico LOOP
    - Bucle FOR
    - Bucle WHILE

Sentencia IF

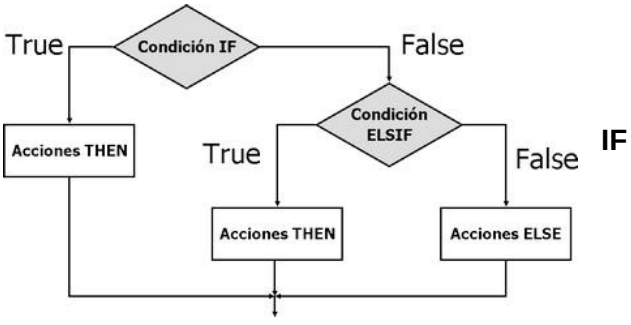
- Sintaxis:

```
IF condición THEN
    sentencias;
[ELSIF condición THEN
    sentencias;]
[ELSE
    sentencias;]
END IF;
```

Flujo de IF-THEN-ELSE –END IF



Flujo de IF-THEN-ELSIF – END IF



Condiciones Booleanas

AND	TRUE	FALSE	NULL	**OR**	TRUE	FALSE	NULL	**NOT**	
TRUE	TRUE	FALSE	NULL	TRUE	TRUE	TRUE	TRUE	TRUE	FALSE
FALSE	FALSE	FALSE	FALSE	FALSE	TRUE	FALSE	NULL	FALSE	TRUE
NULL	NULL	FALSE	NULL	NULL	TRUE	NULL	NULL	NULL	NULL

Bucle Básico: LOOP

- Sintaxis:

```
LOOP
  sentencia1;
  . . .
  EXIT [WHEN condición];
END LOOP;
```

- Donde condición es una expresión o variable booleana (TRUE, FALSE o NULL).

Bucle FOR

- Sintaxis:

```
FOR índice IN [REVERSE] desde..hasta LOOP
  sentencia1;
  . . .
END LOOP;
```

- No declare el índice; se declara implícitamente como un BINARY_INTEGER. Fuera del bucle el índice no está definido.
- Los límites desde..hasta deben de ser literales numéricos. Pueden ser expresiones que se convierten en valores numéricos.

Bucle WHILE

- Sintaxis:

```
WHILE condición LOOP
  sentencia1;
  . . .
END LOOP;
```

- La condición se evalúa al inicio de cada iteración

Etiquetas y Loops Anidados

- Puede anidar bucles a varios niveles.
- Utilice etiquetas para distinguir entre los bloques y los bucles.
- Salga al bucle externo con la sentencia EXIT que hace referencia a la etiqueta.
- Los nombres de etiquetas deben ir antes de la palabra LOOP y entre los delimitadores << >>.

Etiquetas y Loops Anidados

- Ejemplo:

```
BEGIN
<<Loop_Exterior>>
LOOP
        v_counter := v_counter + 1;
        EXIT WHEN v_counter > 10;
        <<Loop_Interior>>
        LOOP
                ...
                EXIT Loop_Exterior WHEN total_done = 'YES';
                EXIT WHEN inner_done = 'YES';
                ...
        END LOOP Loop_Interior;
END LOOP Loop_Exterior;
```

CURSORES

- Útiles para las consultas que devuelven más de una fila.
- Son declarados y nombrados por el programador, y manipulados por medio de sentencias específicas en las acciones ejecutables del bloque.

Control de Cursores

- 1º. Crear un área SQL específica ⟶ **DECLARE**
- 2º. Identificar el juego activo ⟶ **OPEN**
- 3º. Cargar la fila actual en variables ⟶ **FETCH**
- 4º. Si todavía existen filas sin leer, volver a 3º.
- 5º. Si no existen más filas a leer ⟶ **CLOSE**

Declaración del Cursor

- Sintaxis:

```
CURSOR nombre_cursor IS
    sentencia_select;
```

- No incluya la cláusula INTO en la declaración del cursor.
- Si es necesario procesar filas en algún orden, incluya la cláusula ORDER BY.

Ejemplo de Declaración

```
DECLARE
  CURSOR c1 IS
        SELECT empno, ename, job, sal
        FROM   emp
        WHERE sal > 2000;
BEGIN
...
END;
```

Apertura del Cursor

- Sintaxis:

```
OPEN  nombre_cursor;
```

- Si la consulta no devuelve ninguna fila, no se producirá ninguna excepción al abrir el cursor.
- Para los cursores declarados con la cláusula FOR UPDATE, la sentencia OPEN bloquea estas filas.

Recuperación de Datos

- Sintaxis:

```
FETCH nombre_cursor
        INTO [variable1, ...] | nombre_registro];
```

- Incluya, en el FETCH, el mismo número de variables que las definidas en el cursor, y en el mismo orden.
- Compruebe si el cursor tiene filas.

Cierre del Cursor

- Sintaxis:

```
CLOSE  nombre_cursor;
```

- Cierre el cursor una vez completado el procesamiento de las filas.
- Si es necesario, vuelva a abrir el cursor.
- No intente recuperar los datos de un cursor una vez que ha sido cerrado (INVALID_CURSOR).

Atributos de Cursores

- Proporcionan información de estado del cursor:

Atributo	Tipo	Descripción
%ISOPEN	Booleano	TRUE si el cursor está abierto.
%NOTFOUND	Booleano	TRUE si la recuperación más reciente no devuelve ninguna fila.
%FOUND	Booleano	TRUE si la recuperación más reciente devuelve una fila.
%ROWCOUNT	Número	Proporciona el número total de filas devueltas hasta ese momento.

El Atributo %ISOPEN

- Utilice el atributo de cursor %ISOPEN antes de ejecutar una recuperación para comprobar si el cursor está abierto.

- Ejemplo:

```
IF NOT cursor_nombres%ISOPEN THEN
        OPEN cursor_nombres;
END IF;
LOOP
        FETCH cursor_nombres INTO ...
END LOOP;
```

Atributos %NOTFOUND, %ROWCOUNT

- Utilice %ROWCOUNT para recuperar un número exacto de filas.
- Utilice %NOTFOUND para determinar cuándo salir del bucle de lectura del cursor.
- Antes de la primera recuperación, %NOTFOUND es NULL, así que si FETCH no se ejecuta nunca satisfactoriamente, no se saldría nunca del bucle de lectura.

Ejemplo %NOTFOUND, %ROWCOUNT

- Ejemplo:

```
LOOP
        FETCH cursor_nombres INTO v_ename, v_deptno;
        IF cursor_nombres%ROWCOUNT > 20 THEN
                ...
        END IF;
        EXIT WHEN cursor_nombres%NOTFOUND
              OR cursor_nombres%NOTFOUND IS NULL;
        ...
END LOOP;
```

Bucles FOR de Cursor

- Sintaxis:

```
FOR nombre_registro IN nombre_cursor LOOP
      setencia1;
      . . .
END LOOP;
```

- Apertura, recuperación y cierre implícitos.
- No declare el registro, está declarado implícitamente.

Cursores con Parámetros

- Sintaxis:

```
CURSOR nombre_cursor
      [(nombre_parámetro tipo_de_dato, ...)]
 IS sentencia_select;
```

- Permite abrir un cursor varias veces con un juego activo distinto cada vez.
- Cada parámetro formal de la declaración del cursor debe tener un parámetro real correspondiente en la sentencia OPEN.
- La sintaxis de los parámetros es:

```
nombre_parámetro [IN] tipo_de_dato
                  [{:= | DEFAULT} expresión]
```

- Transfiera el número de departamento y el cargo a la cláusula WHERE:

```
DECLARE
     CURSOR  cursor_empleados (v_deptno NUMBER, v_job VARCHAR2) IS
        SELECT ename, sal, hiredate
        FROM    emp
        WHERE deptno = v_deptno
            AND title = v_job;
  BEGIN
     . . .
```

Cláusula FOR UPDATE

- Sintaxis:

```
SELECT ...   FROM ...
FOR UPDATE [OF nombre_columna] [NOWAIT]
```

- El bloqueo explícito le permite denegar el acceso mientras dura una transacción.
- Bloquee las filas antes de la actualización o supresión.
- La cláusula FOR UPDATE es la última cláusula de una sentencia SELECT, incluso después del ORDER BY.
- NOWAIT devuelve un error de Oracle si las filas han sido bloqueadas por otra sesión, de lo contrario se espera.

Cláusula WHERE CURRENT OF

- Sintaxis:

```
{UPDATE | DELETE} ....
WHERE CURRENT OF nombre_cursor;
```

- Incluya la cláusula FOR UPDATE en la definición del cursor para bloquear las filas.
- Especifique WHERE CURRENT OF en la sentencia UPDATE o DELETE para referirse a la fila actual del cursor.

EXCEPCIONES

- ¿Qué es una excepción? Es un identificador de PL/SQL que surge durante la ejecución.
- ¿Cómo surge? Se produce por un error Oracle o bien puede ser provocada explícitamente.
- ¿Cómo se gestiona? Interrumpiéndola con un manejador de excepciones o propagándola al entorno de llamadas.

Interrupción de Excepciones

- Sintaxis:

```
EXCEPTION
     WHEN excepción1 [OR excepción2 ...] THEN
          sentencias;
     [WHEN excepción3 [OR excepción4 ...] THEN
          sentencias; ]
     [WHEN OTHERS THEN
          sentencias; ]
```

Excepciones de Oracle

Nombre de la excepción	Número de error Oracle	Descripción
NO_DATA_FOUND	ORA-01403	La sentencia SELECT no devolvió datos.
TOO_MANY_ROWS	ORA-01422	La sentencia SELECT devolvió más de una fila.
INVALID_CURSOR	ORA-01001	Se produjo una operación de cursor ilegal.
ZERO_DIVIDE	ORA-01476	Se intentó dividir entre cero.
DUP_VAL_ON_INDEX	ORA-00001	Se intentó insertar un valor duplicado.
INVALID_NUMBER	ORA-01722	Falla la conversión de una cadena de caracteres a números.

```
BEGIN
   SELECT ……
EXCEPTION
   WHEN NO_DATA_FOUND THEN
        sentencias;
        DBMS_OUTPUT.PUT_LINE('Excepción NO_DATA_FOUND');
   WHEN TOO_MANY_ROWS THEN
        sentencias;
   WHEN OTHERS THEN
        sentencias para las demás excepciones no tratadas;
END;
```

SQLPredefinidas de ORACLE, sin nombre. Método 1.Ejemplo

DECLARE
// declaración y asociación de la excepción
e_restri_FK_empEXCEPTION;
PRAGMA EXCEPTION_INIT(e_restri_FK_emp, -2292);
v_dptodepartamentos.dpto%type:=10;
BEGIN
DELETE FROM departamentos WHERE dpto= v_dpto;
COMMIT;
EXCEPTION
WHEN e_restri_FK_empTHEN
dbms_output.put_line('no se puede borrar porque existen empleados en ese departamento');
END;

SQLPredefinidas de ORACLE, sin nombre. Método 2

Cuando se produce una excepción sin nombre, hay dos pseudovariables que almacenan información del error:
SQLCODE**:**Nº del código del error.
SQLERRM**:**Mensaje asociado al nº del error.
Así, se pueden gestionar en el manejador WHEN OTHERS:
WHEN OTHERS THEN
IF SQLCODE= -2292 THEN
dbms_output.put_line ('Error de integridad referencial');

END IF;

Excepciones de Usuario

- Se declaran en la sección declarativa DECLARE.
- Se provocan explícitamente en la sección ejecutable utilizando la sentencia **RAISE**.
- Se gestiona la excepción dentro del bloque de excepciones EXCEPTION.

Ejemplo:

```
DECLARE
        mi_excepción  EXCEPTION;
BEGIN
        ...
        RAISE mi_excepción;
        ...
EXCEPTION
        WHEN mi_excepción THEN
                sentencias;
END;
```

Funciones para Identificar Excepciones

- **SQLCODE** ⟶ Devuelve el valor numérico del código de error SQL. No se puede referenciar directamente, hay que asignarlo a una variable PL/SQL de tipo NUMBER.

Valor de SQLCODE	Descripción
0	No se encontró ninguna excepción
1	Excepción definida por el usuario
+100	Excepción NO_DATA_FOUND
Negativo	Otro número de error del Servidor Oracle8

- **SQLERRM** ⟶ Devuelve el mensaje asociado con el número de error. Tipo VARCHAR2.

RAISE_APPLICATION_ERROR

- Sintaxis:

```
RAISE_APPLICATION_ERROR (número_de_error,
        mensaje [, {TRUE | FALSE}]);
```

- Utilice el procedimiento RAISE_APPLICATION_ERROR para comunicar de forma interactiva una excepción predefinida, devolviendo un código y un mensaje de error no estándar.
- Se utiliza en dos lugares distintos:
    - SECCIÓN EJECUTABLE
    - SECCIÓN DE EXCEPCIONES
- Ejemplos:

```
EXCEPTION
  WHEN NO_DATA_FOUND THEN
      RAISE_APPLICATION_ERROR (-20201, 'No existen registros');
END;

BEGIN
  IF SQL%NOTFOUND THEN
      RAISE_APPLICATION_ERROR (-20201, "No existen registros');
  END IF;
END;
```

PROCEDIMIENTOS PL/SQL

CREATE OR REPLACE PROCEDURE [esquema].nomproc

(nombre-parámetro {IN, OUT, IN OUT} tipo de dato, ..) {IS, AS}

Declaración de variables;
Declaración de constantes;
Declaración de cursores;

BEGIN
Cuerpo del subprograma PL/SQL;

EXCEPTION
Bloque de excepciones PL/SQL;

END**;**

FUNCIONES PL/SQL

- Una función es un bloque nombrado PL/SQL que devuelve un valor.
- Una función puede estar almacenada en la B.D., como objeto de la B.D., para repetidas ejecuciones.
- Una función puede ser llamada como parte de una expresión.

Sintaxis Creación Funciones

```
CREATE [OR REPLACE] FUNCTION nombre_func
      (argumento1 [modo] tipo_de_dato,
       argumento2 [modo] tipo_de_dato,
       . . .)
      RETURN tipo_de_dato
IS | AS
      bloque_PL/SQL;
```

Creación de una Función

```
SQL> CREATE OR REPLACE FUNCTION obtener_salaric
          (v_id  IN  emp.empno%TYPE)
           RETURN NUMBER
           IS
                 v_salario  emp.sal%TYPE := 0;
           BEGIN
                 SELECT sal  INTO v_salario
                 FROM    emp
                 WHERE empno = v_id;
                 RETURN (v_salario);
          END obtener_salario;
```

Desde dónde llamarlas

- Como columna de un SELECT.
- Condiciones en cláusulas WHERE y HAVING.
- Cláusulas ORDER BY y GROUP BY.
- Cláusula VALUES de un comando INSERT.
- Cláusula SET de un comando UPDATE.

Borrado de Funciones

- Para quitar una función de la parte del servidor, utilizando SQL*Plus, ejecutar el comando DROP FUNCTION.
- Sintaxis:

```
DROP FUNCTION  nombre_de_función;
```

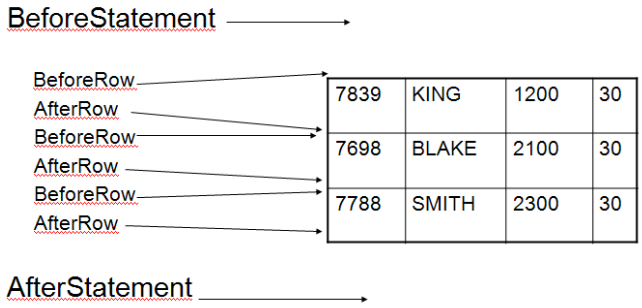
Triggers

Bloques de código PL-SQL que se ejecutan automáticamente cuando se produce un evento.

- Momento: BEFORE o AFTER o INSTEAD OF.
- Evento: INSERT ó UPDATE ó DELETE
- Nombre Tabla: ON NombreTabla
- Tipo de Disparo: Por fila o por sentencia.
- Condición de disparo: Claúsula WHEN. (sólo en triggers por fila).
- Cuerpo del trigger:
  DECLARE
  BEGIN
  EXCEPTION
  END;

  Puede contener variables, cursores, excepciones…
  Si el trigger es de fila se permite acceder al valor de los datos afectados por la orden DML empleando **:old** y **:new**.

Orden de disparo:



Predicados condicionales:

Cuando un trigger puede ser disparado por varias operaciones distintas, hay una forma de saber dentro del código del trigger cuál de ellas lo disparó, se llaman predicados condicionales y se usan así:

- IF INSERTING THEN…
- IF UPDATING [('nombreColumna')] THEN…
- IF DELETING THEN…

Ejemplo:

```
CREATE OR REPLACE TRIGGER SeguridadEmp
BEFORE INSERT ON emp
BEGIN
   IF (TO_CHAR(sysdate, 'DY') IN ('SAT', 'SUN') OR
      TO_CHAR(sysdate,'HH24) NOT BETWEEN '08' AND '15' THEN
         RAISE_APPLICATION_ERROR( -20100, 'No puedes insertar
         registros fuera del horario normal de oficina');
   END IF;
END;
/
```

- Nota: **Cuando un trigger termina con excepciones no tratadas, la sentencia que lo disparó hace un ROLLBACK automáticamente.**

Restricciones en el uso de triggers:

- No pueden ejecutarse instrucciones DDL.
- No pueden ejecutarse instrucciones de control de transacciones (COMMIT, ROLLBACK, …)
- Por sentencia: No tiene sentido el uso de :old y :new.
- Por fila: No se pueden consultar los datos de la tabla que ha disparado el trigger, esto es, no puedo hace una SELECT de esa tabla (problema de tablas mutantes).

Gestión de triggers:

Desactivar/Activar un trigger:

```
ALTER TRIGGER NombreTrigger DISABLE;

ALTER TRIGGER NombreTrigger ENABLE;
```

Para activar/desactivar todos los triggers asociados a una tabla:

```
ALTER TABLE NombreTabla ENABLE ALL TRIGGERS;

ALTER TABLE NombreTabla DISABLE ALL TRIGGERS;
```

Para borrar un trigger definitivamente:

```
DROP TRIGGER NombreTrigger;
```

Paquetes:

Consta de los siguientes elementos:

- Especificación o cabecera: contiene las declaraciones públicas (es decir, accesibles desde cualquier parte de la aplicación) de sus programas, tipos, constantes, variables, cursores, excepciones, etc.
- Cuerpo: contiene los detalles de implementación y declaraciones privadas, es decir, accesibles solamente desde los objetos del paquete.
- La cabecera se compila independientemente del cuerpo.

La sintaxis de la cabecera es la siguiente:

```
CREATE [OR REPLACE] PACKAGE nombre_paquete AS
   <declaraciones públicas>
   <especificaciones de subprogramas>
END NOMBRE_PAQUETE;
```

La sintaxis del cuerpo sería la siguiente:

```
CREATE [OR REPLACE] PACKAGE BODY nombre_paquete AS
   <declaraciones privadas>
   <código de subprogramas>
   [begin
      <instrucciones iniciales>]
END NOMBRE_PAQUETE;
```

Ejemplo: paquete que nos permite buscar un empleado de tres formas distintas y visualizar sus datos.

```
/* Cabecera */
CREATE OR REPLACE PACKAGE BUSCAR_EMPLEADOS AS
   TYPE t_reg_empleados is RECORD
      (num_empleados empleados.num%TYPE,
      apellido empleados.apellido%TYPE,
      salario empleados.salario%TYPE,
      departamento empleados.dpto%TYPE);
   PROCEDURE ver_por_numero(v_cod empleados.num%TYPE);
   PROCEDURE ver_por_apellido(v_ape empleados.apellido%TYPE);
   FUNCTION datos (v_cod empleados.num%TYPE)
      return t_reg_empleados;
end buscar_empleados;

/* Cuerpo */
CREATE OR REPLACE PACKAGE BODY buscar_empleados AS
   vg_empleados t_reg_empleados;
   PROCEDURE ver_empleados; /* procedimiento privado*/
   PROCEDURE ver_por_numero (v_cod empleados.num%TYPE)
      is
        begin
          select num, apellido, salario, dpto into vg_empleados
            from empleados
            where num=v_cod;
          ver_empleados;
   end ver_por_numero;
   PROCEDURE ver_por_apellido (v_ape empleados.apellido%TYPE)
     is
       begin
          select num,apellido,salario,dpto into vg_empleados
            from empleados
            where apellido=v_ape;
         ver_empleados;
```

```
   end ver_por_apellido;
   FUNCTION datos (v_cod empleados.num%TYPE)
      return t_reg_empleados
     is
       begin
          select num,apellido,salario,dpto into vg_empleados
            from empleados
            where num=v_cod;
        ver_empleados;
        return vg_empleados;
        end datos;
```

```
PROCEDURE ver_empleados
      is
      begin
         DBMS_OUTPUT.PUT_LINE(vg_empleados.num_empleados || '*'
             || vg_empleados.apellido || '*' || vg_empleados.salario || '*'
            || vg_empleados.departamento);
   end ver_empleados;
end buscar_empleados;
```

Utilización de los objetos definidos en los paquetes

Podemos utilizar los objetos definidos en los paquetes básicamente de dos maneras distintas:

- Desde el mismo paquete: esto quiere decir que cualquier objeto puede ser utilizado dentro del paquete por otro objeto declarado en el mismo.

  Para utilizar un objeto dentro del paquete tan sólo tendríamos que llamarlo. La llamada sería algo así:

  v_empleados :=buscar_empleados.datos(v_n_ape);
- Desde fuera del paquete: Podemos utilizar los objetos de un paquete siempre y cuando haya sido declarado en la especificación del mismo. Para llamar a un objeto o procedimiento desde fuera del paquete utilizaríamos la siguiente notación: nombre_paquete.nombre_procedimiento(lista de parametros);

  Ejemplo llamadas:

```
declare
r eg   buscar_emple.t_reg_emple;
begin
reg:=buscar_emple.datos(7521);
end;
…..
begin
buscar_emple.ver_por_numero(7521);
end;
```