ÍNDICE

Contenido

Antolín·Muñoz·Chaparro¶
Jefe·de·Proyecto·Sistemas·Informáticos·¶
División·III·de·Aplicaciones·de·Costes·de·Personal·y·Pensiones·Públicas¶
Oficina·de·Informática·Presupuestaria·(DIR3:·EA0027952)¶
Intervención·General·de·la·Administración·del·Estado¶

Antolín·Muñoz·Chaparro¶
Jefe·de·Proyecto·Sistemas·Informáticos·¶
División·III·de·Aplicaciones·de·Costes·de·Personal·y·Pensiones·Públicas¶
Oficina·de·Informática·Presupuestaria·(DIR3:·EA0027952)¶
Intervención·General·de·la·Administración·del·Estado¶

SQL VS PL/SQL

SQL es un lenguaje estructurado de consulta que define como manipular los datos en Oracle. Esa potencia unida a los componentes procedimentales que se han empezado a ver anteriormente, y que se proporcionan a través de PL/SQL, hacen que la gestión de un sistema gestor de base de datos como Oracle sea completa.

Las órdenes SQL se pueden dividir en 6 categorías que se muestra a continuación:

- Lenguaje de manipulación de datos (DML – Data Manipulation Language).
- Lenguaje de definición de datos (DDL – Data Definition Language).
- Las órdenes de control de transacciones.
- Las órdenes de control de sesión.
- Las órdenes de control del sistema.

Lenguaje DML

El lenguaje DML (Data Manipulation Language) permite cambiar o consultar los datos contenidos en una tabla, pero no permite cambiar la estructura de la misma u otro objeto de la base de datos.

Las órdenes básicas que se utilizan en el lenguaje DML se indican en la siguiente tabla:

Comando	Descripción
SELECT	Se utiliza para consultar registros de las tablas de la base de datos que satisfagan un criterio determinado.
INSERT	Se utiliza para insertar información en las tablas de la base de datos en una única operación.
UPDATE	Se utiliza para modificar los valores de las columnas y registros de las tablas de la base de datos.
DELETE	Se utiliza para eliminar registros de una tabla de una base de datos.

Lenguaje DDL

El lenguaje DDL (Data Definition Language) permite crear, borrar o modificar la estructura de un objeto del esquema de la base de datos. Las órdenes para asignar y revocar permisos sobre estos objetos también pertenecen a este lenguaje.

Las órdenes básicas que se utilizan en el lenguaje DDL se indican en la siguiente tabla:

Comando	Descripción
CREATE	Se utiliza para crear nuevas estructuras: tablas, procedimientos, paquetes, índices, secuencias, vistas, etc.
DROP	Se utiliza para eliminar estructuras creadas.
ALTER	Se utiliza para modificar las estructuras creadas.
GRANT	Se utiliza para asignar derechos.
REVOKE	Se utiliza para revocar derechos asignados.

Órdenes para el control de transacciones

Este conjunto de instrucciones que se indican en la siguiente tabla, garantizan la consistencia de los datos.

Comando	Descripción
COMMIT	Esta orden finaliza la tarea procesada en una transacción haciendo efectivo los cambios sobre la base de datos.
ROLLBACK	Esta orden finaliza la tarea procesada en una transacción, pero deshaciendo los cambios que se hubiesen invocado, permaneciendo la base de datos en el estado original antes de comenzar la transacción.
SAVEPOINT	Esta orden marca un punto en la transacción desde el cual se puede invocar una operación de COMMIT o ROLLBACK.

Órdenes para el control de la sesión

Este conjunto de instrucciones cambian las opciones de una conexión determinada a la base de datos, por ejemplo habilitar la traza SQL.

Las órdenes básicas para el control de la sesión se indican en la siguiente tabla:

Comando	Descripción
ALTER SESSION	Permite alterar parámetros de la sesión actual.
SET ROLE	Permite asignar un role de permisos a un usuario.

Órdenes para el control del sistema

Este conjunto de instrucciones cambian las opciones que afectan a la base de datos completa, por ejemplo la activación o desactivación del archivado.

Las órdenes básicas para el control del sistema se indican en la siguiente tabla:

Comando	Descripción
ALTER SYSTEM	Permite alterar los parámetros del sistema.

Utilización de SQL en PL/SQL

El lenguaje PL/SQL admite la interpretación de sentencias SQL dentro de un bloque BEGIN..END, pero con una serie de restricciones que se indican a continuación:

- No se permiten la utilización de sentencias DDL directas dentro del código incluido en el bloque PL/SQL.
- Se pueden utilizar las sentencias DML: insert, update y delete, directamente en el código incluido en el bloque PL/SQL.
- Sólo puede utilizar la sentencia SELECT del lenguaje DML, directamente en el código incluido en el bloque PL/SQL, si va acompañada de la cláusula INTO.
- El uso de la sentencia SELECT directamente en el código incluido en el bloque PL/SQL, está limitado a que el resultado de la consulta devuelve un solo

registro. Si la consulta devuelve más de un registro o ninguno, generará un error de ejecución del bloque PL/SQL.

- Está permitido el uso de sentencias para el control de transacciones dentro del código incluido en el bloque PL/SQL.
- No se permite el uso de sentencias de control del sistema o de control de sesión, directamente en el código incluido en el bloque PL/SQL.

Para poder utilizar el resto de instrucciones del lenguaje SQL o aquellas que no están permitidas directamente en el código, se utiliza SQL dinámico.

La sintaxis de la instrucción SELECT para interpretarla directamente en el código del bloque PL/SQL, es la siguiente:

```
SELECT elemento_lista1[,elemento_lista2,…]
INTO registro_PLSQL/variable1[,variable2,…]
FROM tabla
[WHERE condicionantes];
```

Como se ha indicado anteriormente y como se puede apreciar en la sintaxis, la única diferencia con la sintaxis de un SELECT ejecutado en el lenguaje SQL, es que se incluye la cláusula INTO para devolver los valores que retorna la instrucción en una variable de tipo registro o en variables individuales del mismo tamaño y tipo que las columnas que se están consultado.

A continuación puede observar una serie de ejemplos del uso correcto e incorrecto de la instrucción SELECT dentro de un bloque PL/SQL, tomando como referencia la tabla EMPLEADOS con el siguiente contenido:

DNI	Nombre	Apellidos	Sexo	Edad
11111111A	JOSE ANTONIO	BLAZQUEZ SANCHEZ	M	31
22222222B	JACINTO	BERMUDEZ TAPIA	M	32
33333333C	ELISA	RIAÑO TORAL	F	33
44444444D	CLARA	YAÑEZ SANCHEZ	F	33

EJEMPLO DE USO CORRECTO DE LA INSTRUCCIÓN SELECT EN PL/SQL

```
DECLARE
      V_Edad     NUMBER(2);
      V_Conteo   NUMBER;
BEGIN
      SELECT COUNT(*) INTO V_Conteo FROM empleados
      WHERE edad = 31;

      IF V_Conteo = 1 THEN

         SELECT edad INTO V_Edad FROM empleados
         WHERE edad = 31;
      ELSIF V_Conteo = 0 THEN

         DBMS_OUTPUT.PUT_LINE('No hay personas con '||
                              '31 años');
      ELSE
         DBMS_OUTPUT.PUT_LINE('Hay más de 1 persona con'||
                              ' 31 años.');
      END IF;
END;
```

En este ejemplo se definen 2 variables (`V_Edad` y `V_Conteo`) para la recogida de la información que se va a recuperar posteriormente en las consultas del bloque BEGIN..END.

Dentro del bloque BEGIN..END, en primer lugar se cuenta el número de registros que hay para la condición de búsqueda del SELECT: que la edad sea igual a 31 años, llevando el resultado a la variable `V_Conteo`.

A continuación se evalúa mediante un IF, el contenido de la variable `V_Conteo`, de forma que sólo si tiene valor 1 (es decir, la consulta va a devolver únicamente un registro), entonces se ejecuta otra sentencia SELECT que recupera el valor de la edad y la lleva a la variable `V_Edad`.

En caso de que la variable `V_Conteo` tenga valor 0 (no se recupera ningún empleado con 31 años) o mayor que 1 (recupera más de un empleado con 31 años),

se devuelve un mensaje de error al usuario, sin que el bloque aborte por errores de ejecución.

EJEMPLO DE USO INCORRECTO DE LA INSTRUCCIÓN SELECT EN PL/SQL

```
DECLARE
      V_Nombre   EMPLEADOS.NOMBRE%TYPE;
BEGIN
      SELECT Nombre INTO V_Nombre FROM empleados
      WHERE edad = 33;
END;
```

Este ejemplo provocaría un error en tiempo de ejecución del bloque (no en compilación), porque al consultar la tabla empleados para aquellos que tenga 33 años, la consulta devuelve dos registros, lo que impide que dichos valores se puedan almacenar a la vez en la variable `V_Nombre`.

Una posible solución a este problema habría sido insertar un conteo previo de registros para la misma condición (empleados con 33 años), para determinar cuántos devuelve la consulta, de forma que si sólo devuelve 1, entonces llevarla a cabo, como se muestra a continuación:

```
DECLARE
      V_Nombre   EMPLEADOS.NOMBRE%TYPE;
      V_Conteo   NUMBER;
BEGIN
      SELECT COUNT(*) INTO V_Conteo FROM empleados
      WHERE edad = 33;

      IF V_Conteo = 1 THEN
            SELECT Nombre INTO V_Nombre FROM empleados
            WHERE edad = 33;
      END IF;
END;
```

Otra alternativa sería la introducción de una sección para el control de errores que se denomina EXCEPTION para que se pudiese capturar el error en tiempo de ejecución sin que aborte el bloque y tratarlo adecuadamente en la sección EXCEPTION, como se muestra a continuación:

```
DECLARE
      V_Nombre   EMPLEADOS.NOMBRE%TYPE;
BEGIN
      SELECT Nombre INTO V_Nombre FROM empleados
      WHERE edad = 33;

EXCEPTION
      WHEN TOO_MANY_ROWS THEN
         DBMS_OUTPUT.PUT_LINE('Hay más de 1 persona con'||
                              ' 33 años.');
      WHEN NO_DATA_FOUND THEN
         DBMS_OUTPUT.PUT_LINE('No hay personas con'||
                              ' 33 años.');
      WHEN OTHERS THEN
         DBMS_OUTPUT.PUT_LINE('Se ha producido un error'||
                              ' al ejecutar el bloque.');

END;
```

EJEMPLO DE USO INCORRECTO DE LA INSTRUCCIÓN SELECT EN PL/SQL

```
DECLARE
      V_Nombre   EMPLEADOS.NOMBRE%TYPE;
BEGIN
      SELECT Nombre FROM empleados
      WHERE edad = 33;
END;
```

Este ejemplo provocaría un error en tiempo de compilación, porque no se permite el uso de una instrucción directa SELECT dentro de un bloque PL/SQL si no incluye la cláusula INTO.

SQL dinámico

El lenguaje PL/SQL ofrece la posibilidad de ejecutar sentencias SQL a partir de cadenas de caracteres (uso embebido del lenguaje SQL).

Para ello debemos emplear la instrucción EXECUTE IMMEDIATE. De esta forma podemos llegar a ejecutar no solo sentencias DML, sino que también podemos llegar a ejecutar cualquier otra sentencia SQL, como las instrucciones DDL.

Para controlar el número de filas afectadas por la ejecución de instrucciones DML del tipo insert, update o delete, ejecutadas mediante EXECUTE IMMEDIATE, se utiliza la instrucción SQL%ROWCOUNT.

El SQL DINÁMICO también permite ejecutar sentencias SELECT, pero no retorna filas salvo que se utilice un cursor.

EJEMPLO DE SQL DINÁMICO CON UNA INSTRUCCIÓN UPDATE

```
SET SERVEROUTPUT ON;

DECLARE
  filas_afectadas      NUMBER;
  Cadena_sql           VARCHAR2(1000);

BEGIN

Cadena_sql := 'UPDATE HOSPITAL SET TELEFONO = 910001122';
EXECUTE IMMEDIATE Cadena_sql;
filas_afectadas := SQL%ROWCOUNT;
dbms_output.put_line(TO_CHAR(filas_afectadas));

END;
```

Este ejemplo define una variable de tipo texto (varchar2) denomina `Cadena_sql`, que es sobre la que se asigna la instrucción de actualización (UPDATE). A continuación se ejecuta dicha cadena con la instrucción EXECUTE IMMEDIATE y por último, se obtiene el conteo de filas actualizadas con SQL%ROWCOUNT.

EJEMPLO DE SQL DINÁMICO CON UNA INSTRUCCIÓN SELECT

```
SET SERVEROUTPUT ON;
DECLARE
  filas_afectadas      NUMBER;
  TYPE tipocursor      IS REF CURSOR;
  V_cursor             tipocursor;
  v_registro           DEPARTAMENTO%ROWTYPE;
  Cadena_sql           VARCHAR2(1000);
BEGIN
Cadena_sql := 'SELECT COUNT(*) FROM DEPARTAMENTO';
EXECUTE IMMEDIATE Cadena_sql INTO filas_afectadas;
dbms_output.put_line(TO_CHAR(filas_afectadas));
Cadena_sql := 'SELECT * FROM DEPARTAMENTO';
OPEN v_cursor FOR Cadena_sql;
LOOP
     FETCH v_cursor INTO v_registro;
     EXIT WHEN v_cursor%NOTFOUND;
dbms_output.put_line('Departamento: '||
v_registro.NOMBRE);
END LOOP;
END;
```

Este ejemplo se define una variable de tipo REF CURSOR, que son las que se emplean para recuperar la información de una instrucción SELECT que se ejecuta en SQL dinámico (EXECUTE IMMEDIATE). Así mismo se utiliza la variable `Cadena_sql` para almacenar el texto correspondiente a la instrucción SELECT.

Dentro del bloque en primer lugar se define una consulta para recuperar el número de filas que tiene la tabla empleado (SELECT COUNT…). Tras asignar el resultado del conteo a la variable `filas_afectadas` se visualiza.

A continuación se define otra consulta (SELECT *…), que recupera todos los registros (filas) de la tabla empleados. Para poder manejar esta información se realiza un bucle (LOOP .. END LOOP) que recorre todos los resultados de la consulta, utilizando la variable `v_cursor` para recorrer los registros como un cursor.

Uso de parámetros en el SQL dinámico

El SQL dinámico permite la parametrización a través de variables host.

Una variable host es una variable que pertenece al programa que está ejecutando la sentencia SQL dinámica, y que podemos asignar en el interior de la sentencia SQL con la palabra clave USING.

Las variables host van precedidas de dos puntos ":".

El siguiente ejemplo muestra el uso de variables host para parametrizar una sentencia SQL dinámica.

```
DECLARE
  nuevo_telefono    hospital.telefono%type;
  sentencia         VARCHAR2(1000);

BEGIN
    nuevo_telefono := 912001010;
    sentencia   :=   'UPDATE   HOSPITAL   SET   TELEFONO   =
:nuevo_telefono';
    EXECUTE IMMEDIATE sentencia USING nuevo_telefono;
    dbms_output.put_line('Este   bloque   ha   actualizado
'||SQL%ROWCOUNT||' filas.');
END;
```

ESTRUCTURAS DE CONTROL

Las estructuras de control, permiten controlar el comportamiento del bloque a medida que éste se ejecuta. Estas estructuras incluyen las órdenes condicionales y los bucles.

Estas estructuras junto con las variables son las que dotan al lenguaje PL/SQL de su poder y flexibilidad.

Tenemos las siguientes estructuras de control:

- IF – THEN – ELSE – END IF
- CASE …. WHEN …. END CASE
- LOOP .... END LOOP
- WHILE .... LOOP .... END LOOP
- FOR.... LOOP .... END LOOP
- GOTO
- NULL

IF-THEN-ELSE-END IF

La sintaxis de esta estructura de control es la siguiente:

```
IF <expresión booleana1> THEN
            <operaciones1>;
[ELSIF <expresión booleana2> THEN
            <operaciones2>;
.....
[ELSE
            <operaciones3>;]
END IF;
```

Permite evaluar expresiones booleanas y admite anidamiento.

EJEMPLO

```
IF var1 < 50 THEN
      <operaciones1>;
ELSIF var1 > 70 THEN
      <operaciones2>;
ELSE
      <operaciones3>;
END IF;
```

En este ejemplo si la variable *var1* tiene un valor inferior a 50 ejecutará el conjunto de operaciones denominado *operaciones1*. Si por el contrario, el valor es superior a 70 entonces ejecutará el conjunto de operaciones denominado *operaciones2*. En caso de que no se cumplan ninguna de las 2 condiciones anteriores, ejecutará el conjunto de operaciones denominado *operaciones3*.

Esta forma de plantear la estructura de control, sería equivalente al siguiente formato:

```
IF var1 < 50 THEN
      <operaciones1>;
ELSE
      IF var1 > 70 THEN
            <operaciones2>;
      ELSE
            <operaciones3>;
      END IF;
END IF;
```

CASE

Permite evaluar múltiples opciones booleanas equivalentes en cuanto al tipo de dato. Es una estructura que permite el anidamiento.

La sintaxis de esta estructura es la siguiente:

```
CASE <variable a comprobar>
    WHEN <valor de comparación 1> THEN <operaciones1>;
    WHEN <valor de comparación 2> THEN <operaciones2>;
    …..
[ELSE <operaciones si no se cumplen las otras>];
END CASE;
```

EJEMPLO (USO DE LA OPCIÓN ELSE DEL CASE)

```
DECLARE
  grado CHAR(1);
BEGIN
  grado := 'B';
  CASE grado
       WHEN 'A' THEN DBMS_OUTPUT.PUT_LINE('Excelente');
       WHEN 'B' THEN DBMS_OUTPUT.PUT_LINE('Bueno');
       WHEN 'C' THEN DBMS_OUTPUT.PUT_LINE('Regular');
  ELSE DBMS_OUTPUT.PUT_LINE('Malo');
  END CASE;
END;
/
```

Aunque la cláusula ELSE es opcional para esta estructura, PL/SQL añade internamente una condición ELSE cuando no se especifica implícitamente en el código. Esta condición interna sería la siguiente:

```
ELSE RAISE CASE_NOT_FOUND;
```

Esto significa que en una sentencia CASE que no se especifique ELSE, el SGBD (Sistema Gestor de Base de Datos) cuando ejecuta el código, si no encuentra ninguna condición WHEN que satisfaga el valor de la variable, ejecutará ELSE RAISE CASE_NOT_FOUND, lo que conlleva que el programa salga del bucle CASE y ejecute la excepción (RAISE) "CASE_NOT_FOUND", por tanto cuando se diseñe un código con CASE en el que no se quiera incluir el ELSE hay que tener en cuenta esta excepción en el control de errores del programa.

EJEMPLO (USO DEL CASE SIN ELSE Y CON CONTROL ERRORES)

```
DECLARE
    grado CHAR(1);
BEGIN
    grado := 'B';
    CASE grado
        WHEN 'A' THEN DBMS_OUTPUT.PUT_LINE('Excelente');
        WHEN 'B' THEN DBMS_OUTPUT.PUT_LINE('Bueno');
    END CASE;
EXCEPTION
    WHEN CASE_NOT_FOUND THEN
         DBMS_OUTPUT.PUT_LINE('Condición fuera del CASE');
END;
/
```

LOOP…END LOOP

Bucle repetitivo de ejecución indefinida con terminación mediante la evaluación de una condición, o mediante salida directa (EXIT).

La sintaxis de esta estructura es la siguiente:

```
LOOP
      <secuencia de órdenes>
      [EXIT [WHEN condicion]];
END LOOP;
```

EJEMPLO

```
LOOP
      INSERT INTO expediente
            .......
      IF var1 > 50 THEN
            EXIT;
      END IF;
END LOOP;
```

Este bucle ejecutara una inserción en la tabla *expediente* hasta que el valor de la variable *var1* sea superior a 50, en cuyo caso ejecutará la instrucción EXIT para salir del bucle.

Otra forma similar de haber implementado el código anterior es la siguiente:

```
LOOP
      INSERT INTO expediente
            .......
      EXIT WHEN var1 > 50;
      END IF;
END LOOP;
```

WHILE…LOOP…END LOOP

Bucle repetitivo donde la evaluación de la condición se produce antes de ejecutarse cada iteración del código a ejecutarse.

La sintaxis de esta estructura es la siguiente:

```
      WHILE <condición> LOOP
            <secuencia de órdenes>
      END LOOP;
```

EJEMPLO

```
WHILE var1 <= 10 LOOP
      INSERT INTO expediente
            .......
END LOOP;
```

Como en cualquier bucle LOOP…END LOOP, podemos forzar la salida del mismo con la instrucción `EXIT`, aunque no se cumpla la condición de evaluación indicada antes de la ejecución de la secuencia de instrucciones del bucle.

FOR…LOOP…END LOOP

Bucle repetitivo de ejecución con un número determinado de ejecuciones determinado por un contador.

La sintaxis de esta estructura es la siguiente:

```
FOR <contador> IN [REVERSE] <límite_inferior>..<límite_superior>
LOOP
      <secuencia de órdenes>
END LOOP;
```

EJEMPLO

```
FOR v_conta IN 1..50 LOOP
     INSERT INTO expediente
          .......;
END LOOP;
```

La variable que se utiliza como índice del bucle (contador) no es necesaria declararla previamente en la sección DECLARE del bloque, se declara implícitamente cuando se asocia a un bucle FOR y Oracle la gestiona con un tipo de dato BINARY_INTEGER.

En caso de utilizar la cláusula REVERSE el proceso de recorrido de los límites del bucle se realiza de forma inversa: del límite superior al límite inferior.

GOTO

Realiza un salto a una etiqueta del bloque.

La sintaxis de esta estructura es la siguiente:

```
GOTO etiqueta;
```

Las etiquetas se definen con la siguiente sintaxis:

```
<<  nombre de la etiqueta >>
```

EJEMPLO

```
BEGIN
      .......
      GOTO eti1;
<<eti1>>
      .......
END;
```

NULL

Permite introducirla en un bloque cuando no se quiere ejecutar operación alguna pero es necesario completar la sintaxis de una instrucción de control. Puede ser muy útil cuando se está diseñando un programa en el que se empiezan a escribir las estructuras de control que se van a utilizar pero no se conoce aún el contenido de lo que tendrá que evaluar cada fase de la misma.

EJEMPLO

```
IF valor > 1 THEN
      ......;
ELSE
      NULL;
END IF;
```

CONTROL DE TRANSACCIONES

Una transacción se considera aquella operación de manipulación de datos para su alteración (insert, update o delete), que permanece a la espera de ser validada (confirmada) o rechazada.

Para el control de las transacciones se dispone de las siguientes instrucciones:

- COMMIT
- ROLLBACK
- SAVEPOINT <nombre_marcador>
- ROLLBACK TO <nombre_marcador_savepoint>

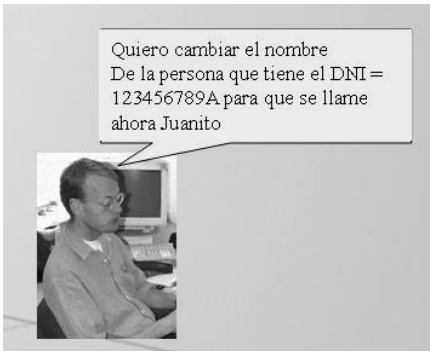
COMMIT

La instrucción COMMIT permite validar toda transacción ejecutada que se encuentre pendiente de confirmar o rechazar.

¿CÓMO SE REALIZA EL PROCESO DE VALIDACIÓN (COMMIT)?

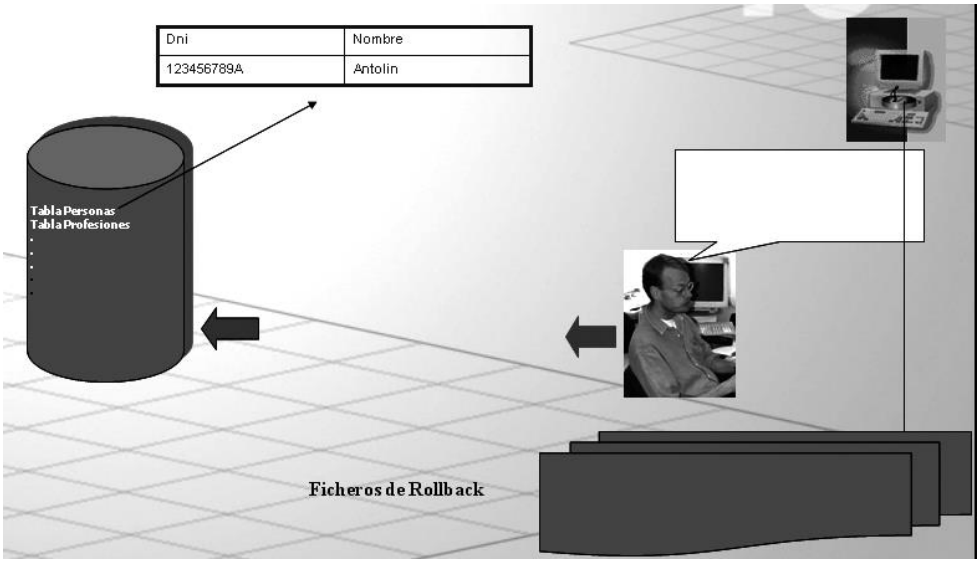
A continuación se muestra de forma esquemática las fases del proceso de validación de una transacción.

1. El proceso de validación de una instrucción comienza con el deseo por parte del usuario de realizar una operación contra la base de datos:



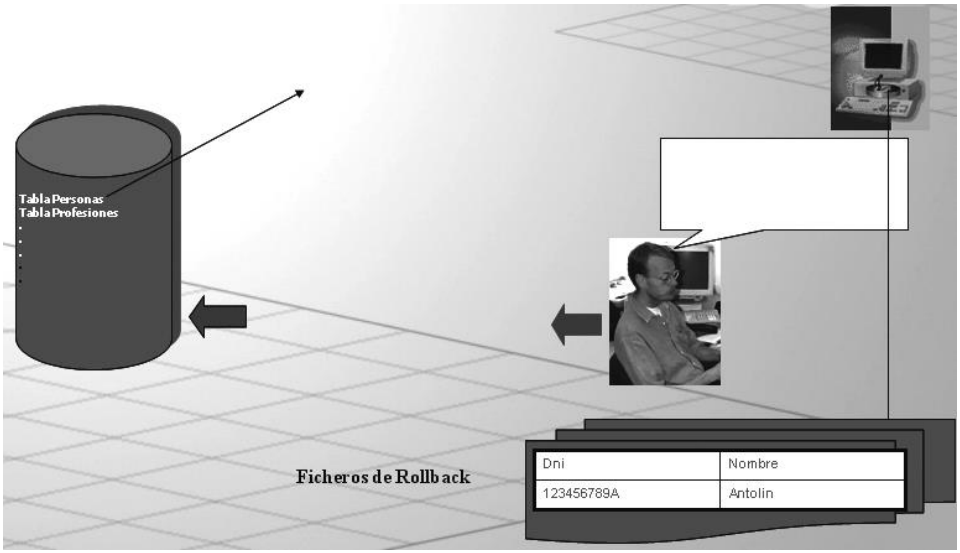
2. El siguiente paso supone la traducción de ese deseo en una instrucción de base de datos. En nuestro ejemplo, una instrucción SQL.

3. A continuación se procede a la recuperación física de los datos afectados por la instrucción. Para ello, el SGBD de Oracle accede hasta la ubicación donde se encuentran los datos y recupera aquellas filas que cumplen las condiciones impuestas en la instrucción enviada por el usuario. Además prepara los ficheros de rollback, que son aquellos ficheros que almacenan las transacciones efectuadas pendientes de confirmación, así como los datos originales antes del cambio.

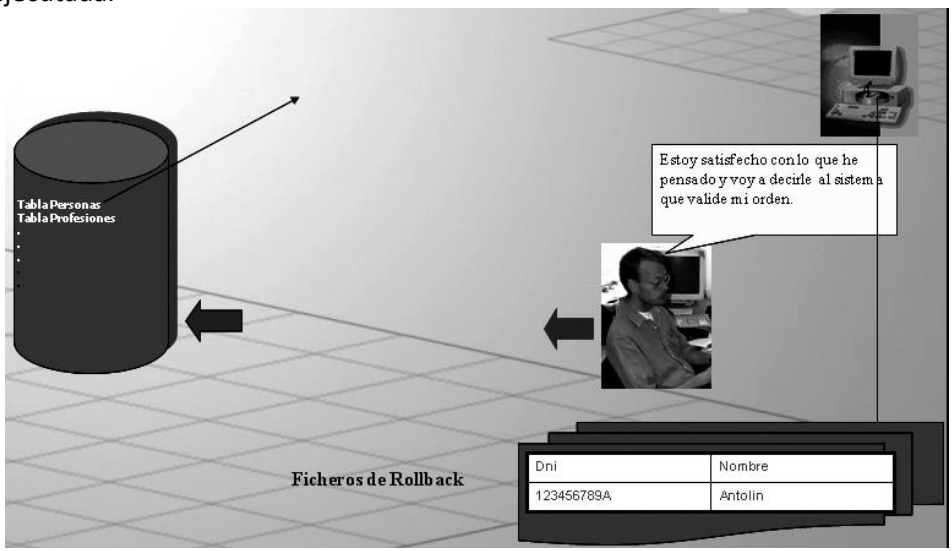


Antolín·Muñoz·Chaparro¶
Jefe·de·Proyecto·Sistemas·Informáticos·¶
División·III·de·Aplicaciones·de·Costes·de·Personal·y·Pensiones·Públicas¶
Oficina·de·Informática·Presupuestaria·(DIR3:·EA0027952)¶
Intervención·General·de·la·Administración·del·Estado¶

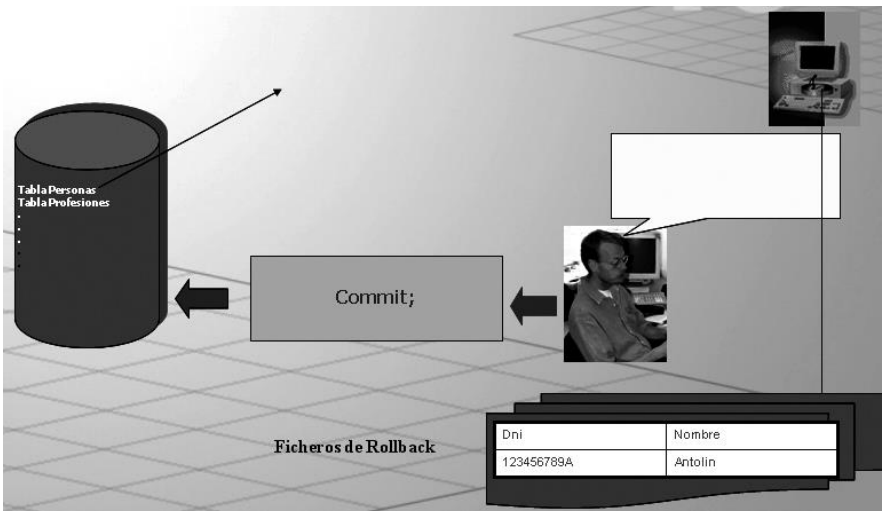
4. Seguidamente el SGBD traslada la información original recuperada de los ficheros físicos de información de la B.D. a los ficheros rollback.



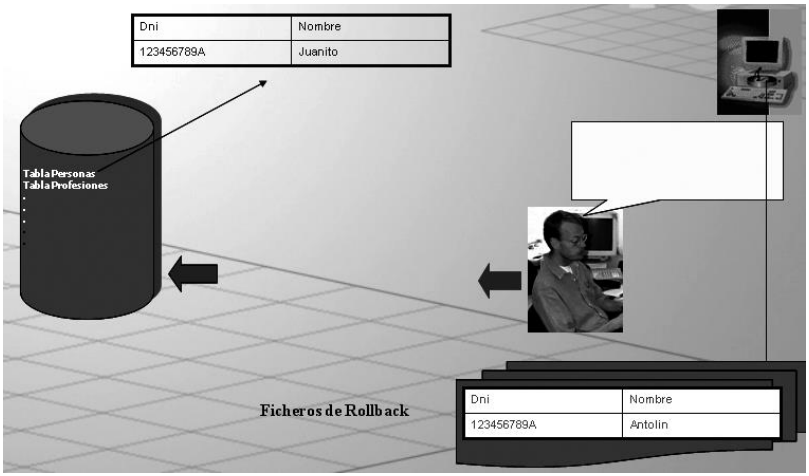
5. A continuación el usuario se plantea la intencionalidad de validar la información que se ha modificado (en nuestro ejemplo), como consecuencia de la instrucción ejecutada.



6. Seguidamente el usuario ejecuta la instrucción, que implica una ordenación al SGBD de Oracle para que valide la información modificada con su primera instrucción y que se encuentra pendiente en los ficheros rollback, para lo que envía un comando COMMIT.



7. Como consecuencia de ejecutar la instrucción COMMIT el SGBD modifica físicamente la información contenida en la base de datos, de acuerdo a la instrucción primitiva que envió el usuario.



8. Por último el SGBD libera los ficheros rollback. Sólo libera aquella información original recuperada con la instrucción primitiva del usuario antes de ejecutar el COMMIT.

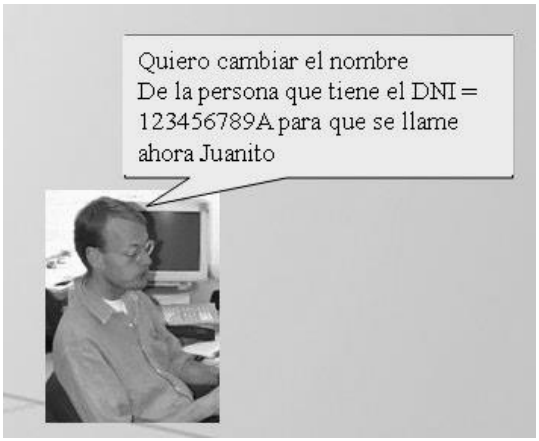
ROLLBACK

La instrucción ROLLBACK permite rechazar (deshacer y dejar a su estado original) toda transacción ejecutada que se encuentre pendiente de confirmar o rechazar.

¿CÓMO SE REALIZA EL PROCESO DE RECHAZO (ROLLBACK)?

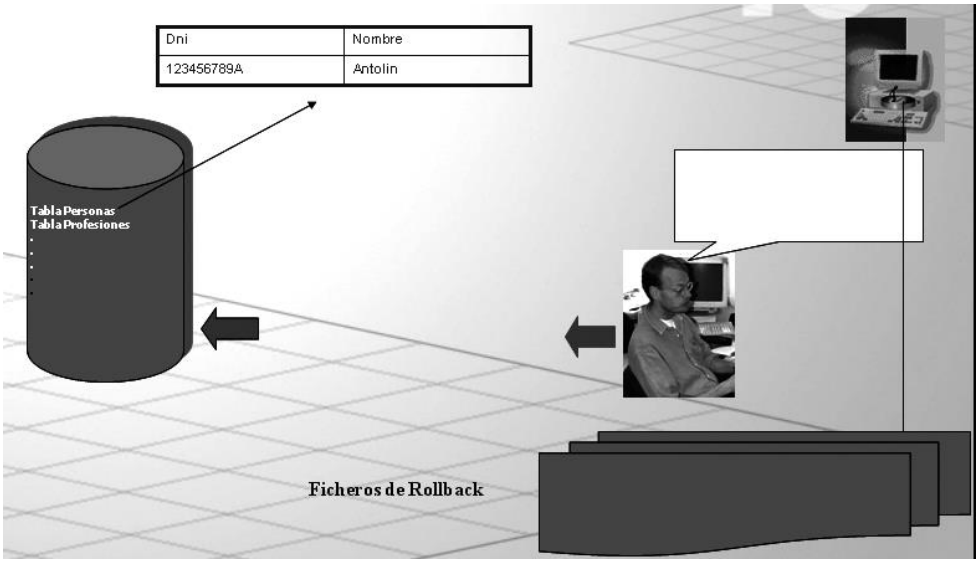
A continuación se muestra de forma esquemática las fases del proceso de rechazo de una transacción.

1. El proceso de rechazo de una instrucción comienza con el deseo por parte del usuario de realizar una operación contra la base de datos:



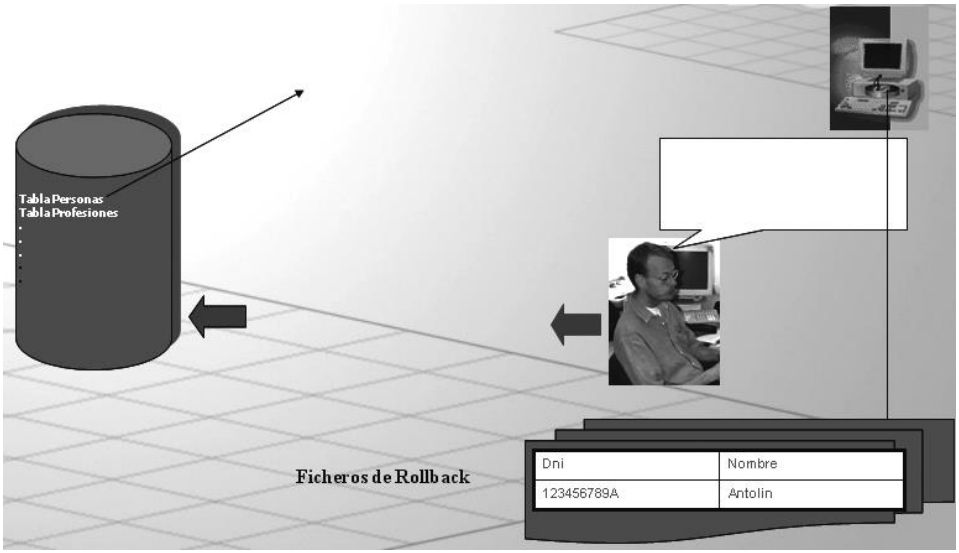
2. El siguiente paso supone la traducción de ese deseo en una instrucción de base de datos. En nuestro ejemplo, una instrucción SQL.

3. A continuación se procede a la recuperación física de los datos afectados por la instrucción. Para ello, el SGBD de Oracle accede hasta la ubicación donde se encuentran los datos y recupera aquellas filas que cumplen las condiciones impuestas en la instrucción enviada por el usuario. Además prepara los ficheros de rollback, que son aquellos ficheros que almacenan las transacciones efectuadas pendientes de confirmación, así como los datos originales antes del cambio.

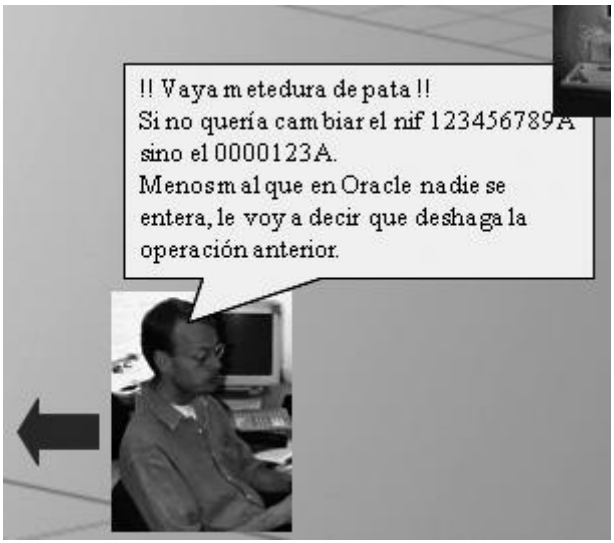


Antolín Muñoz Chaparro
Jefe de Proyecto Sistemas Informáticos
División III de Aplicaciones de Costes de Personal y Pensiones Públicas
Oficina de Informática Presupuestaria (DIR3: EA0027952)
Intervención General de la Administración del Estado
GOBIERNO DE ESPAÑA MINISTERIO DE HACIENDA

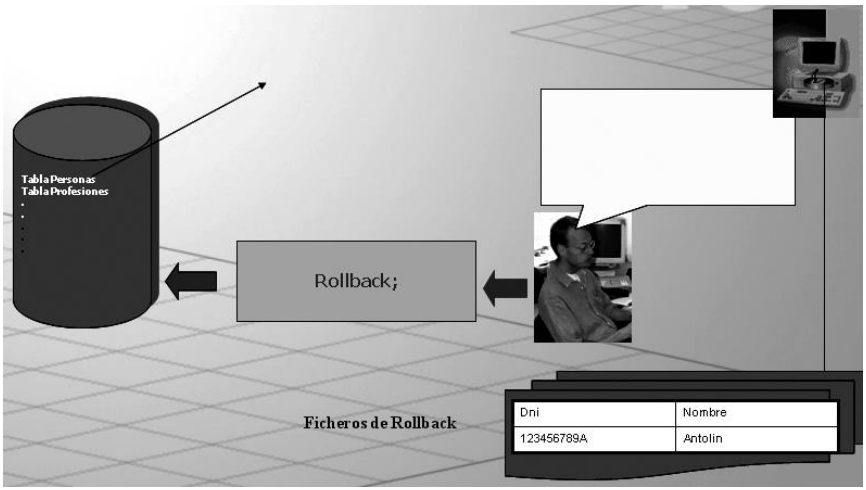
4. Seguidamente el SGBD traslada la información original recuperada de los ficheros físicos de información de la B.D. a los ficheros rollback.



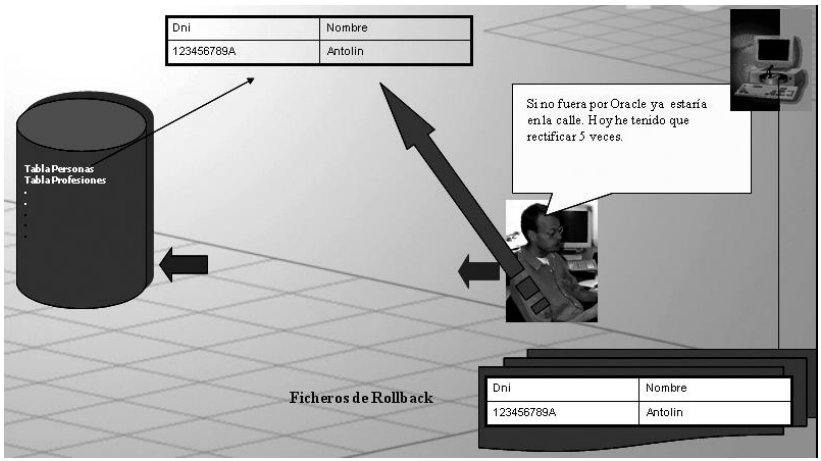
5. A continuación el usuario se plantea la intencionalidad de rechazar los cambios que se generan tras la ejecución de la instrucción.



6. Seguidamente, el usuario ejecuta la instrucción que le ordena al SGBD de Oracle que rechace la información modificada con su primera instrucción y que se encuentra pendiente en los ficheros rollback, para lo que envía un comando ROLLBACK.



7. Como consecuencia de ejecutar la instrucción ROLLBACK el SGBD recupera la información original contenida en los ficheros rollback, correspondiente a la instrucción primitiva que ejecutó el usuario, y la mueve a su posición original dentro de los ficheros de la base de datos.



8. Por último el SGBD dentro del mismo punto anterior, libera los ficheros rollback al realizar el movimiento de la información original recuperada con la instrucción primitiva del usuario.

SAVEPOINT

La instrucción SAVEPOINT permite introducir un marcador de posición dentro de una transacción del código PL/SQL, a fin de ser utilizado como referenciación para la instrucción ROLLBACK TO.

La sintaxis es la siguiente:

```
SAVEPOINT <nombre_marcardor>;
```

Por tanto permite delimitar las transacciones a ejecutar en el programa con objeto de rechazar unas u otras dependiendo de cada situación de ejecución del programa.

ROLLBACK TO

La instrucción ROLLBACK TO permite rechazar las transacciones que existan desde el punto de ruptura (savepoint) indicado, hasta la siguiente instrucción ROLLBAK, COMMIT o END que se encuentre siempre en dirección secuencial hacia abajo en el código PL/SQL.

EJEMPLO DE USO DEL CONTROL DE TRANSACCIONES

```
DECLARE
      V_temp      temporal%ROWTYPE;
      cuenta      number;
BEGIN
      INSERT INTO temporal(col1) VALUES ('Inserto 1');
      SAVEPOINT A;
      INSERT INTO temporal(col1)
      VALUES ('Realizo consulta');
      SELECT COL_NUMERICA INTO CUENTA FROM tab_misteriosa;
      SAVEPOINT B;
      INSERT INTO temporal(col1) VALUES (TO_CHAR(cuenta));
      COMMIT;
EXCEPTION
      -- Si no encuentra datos realiza rollback
      -- desde punto B
      WHEN NO_DATA_FOUND THEN
            ROLLBACK TO B;
            COMMIT;
      /* Si se devuelven múltiples filas realiza rollback
   desde el punto B */
      WHEN TOO_MANY_ROWS THEN
            ROLLBACK TO A;
            COMMIT;
      /* Si se produce cualquier otro error realiza
         rollback Total */
      WHEN OTHERS THEN
            ROLLBACK;
END;
```

TIPOS ESTRUCTURADOS

En PL/SQL a parte de los tipos estándar que ofrece Oracle tanto en SQL como en PL/SQL, también es posible definir otros tipos de usuario, de acuerdo a las siguientes estructuras:

- RECORD (Tipos registro)
- TABLE (Tipos table)
- VARRAY (Tipos array de elementos)

CREACIÓN DE UN TIPO RECORD (REGISTRO)

Es un tipo que define el usuario con la combinación de los tipos ya definidos anteriormente o que pertenecen al lenguaje PL/SQL estandar.

Un registro de Oracle es similar a las estructuras del lenguaje C. Proporciona un mecanismo para tratar con variables diferentes, pero relacionadas como si fueran una unidad.

La sintaxis para definir y crear un registro en un bloque PL/SQL es la siguiente:

```
TYPE tipo_registro IS RECORD (
     Campo1     tipo1 [NOT NULL] [:= valor],
     .......
);
```

EJEMPLO

```
TYPE  estudiante IS RECORD (
     Cod_matricula    NUMBER(5),
     Nombre           VARCHAR2(50),
     Apellidos        VARCHAR2(40),
     Media            NUMBER(6,2) NOT NULL := 0);

V_estudiate           estudiante;
```

Asignación de valores a un registro de forma directa

La sintaxis es la siguiente:

<nombre_var_registro>.<campo> := <valor>;

EJEMPLO

```
v_estudiante.media := 10;
```

Asignación de valores a un registro desde una consulta

La consulta se deberá diseñar con tantas columnas como elementos tenga el registro, debiendo existir equivalencia de tipos entre las columnas y los elementos del registro.

EJEMPLO

```
DECLARE
TYPE  estudiante IS RECORD (
      Cod_matricula    NUMBER(5),
      Nombre           VARCHAR2(50),
      Apellidos        VARCHAR2(40),
      Media            NUMBER(6,2) NOT NULL := 0);
V_estudiate            estudiante;

BEGIN
      SELECT cod_matrícula, nom, ape, media
      INTO v_estudiante
      FROM tab_estudiantes
      WHERE cod_matrícula = 50;
END;
/
```

%Rowtype

Permite definir un tipo registro con el número de campos, nombre, tamaño y tipo de una tabla.

La sintaxis es la siguiente:

<variable> <tabla>%ROWTYPE;

EJEMPLO

```
DECLARE
V_estudiante          tab_estudiantes%ROWTYPE;
BEGIN
      SELECT *
      INTO v_estudiante
      FROM tab_estudiantes
      WHERE cod_matrícula = 50;
END;
```

También con esta definición de tipo, podemos realizar actualizaciones de tablas de la base de datos.

Actualización de una tabla a partir de un tipo RECORD

Es posible actualizar una/varias/todas las columnas de una fila de una tabla a partir de un tipo RECORD.

Cuando no se actualizan todas las columnas de la tabla, se utiliza la sintaxis vista anteriormente: <nombre_var_registro>.<campo> := <valor>;

Pero para actualizar el contenido completo de una fila de una tabla se utiliza la pseudocolumna **ROW**, que nos permite indicarle al SGBD que lo que se va a actualizar es toda la fila de la tabla.

EJEMPLO DE ACTUALIZACIÓN DE COLUMNAS CONCRETAS

```
DECLARE
V_estudiante          tab_estudiantes%ROWTYPE;
BEGIN
      V_estudiante.notafinal := 9.95;
      V_estudiante.nombre := 'Antolin';

      UPDATE tab_estudiantes
      SET nota = v_estudiante.notafinal
      ,nombre = v_estudiante.nombre
      WHERE matricula = 1234567;
END;
/
```

EJEMPLO DE ACTUALIZACIÓN DEL CONTENIDO DE UNA FILA ENTERA

```
DECLARE
V_estudiante          tab_estudiantes%ROWTYPE;
BEGIN
      V_estudiante.notafinal := 9.95;
      V_estudiante.nombre := 'Antolin';

      UPDATE tab_estudiantes
      SET ROW = v_estudiante
      WHERE matricula = 1234567;
END;
/
```

CREACIÓN DE UN TIPO TABLE (PILA DE ELEMENTOS)

Este tipo de dato nos permite crear una matriz o secuencia de elementos en memoria. Todos ellos comparten el mismo tipo de dato y se aglutinan en un único objeto.

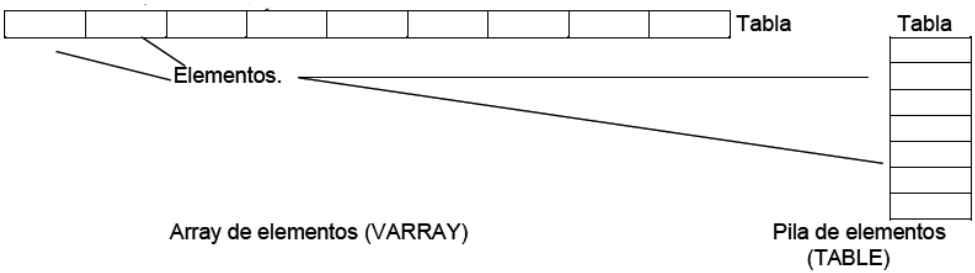


Fig. 4-1 Comparativa de un Varray y un tipo table.

Aunque sintácticamente un tipo TABLE (o tipo tabla) es una matriz de elementos, lo que la convertiría en un VARRAY, en cuanto al almacenamiento interno de Oracle, un tipo TABLE es una pila de elementos, por lo que se diferencia bastante de un array de elementos.

La sintaxis de la definición de un tipo table es la siguiente:

```
TYPE nombre_tipotabla IS TABLE OF tipo_de_dato
INDEX BY BINARY_INTEGER;
```

EJEMPLO

```
DECLARE
      TYPE t_nombres IS TABLE OF estudiante.nombre%TYPE
      INDEX BY BINARY_INTEGER;
      TYPE t_estudiantes IS TABLA OF estudiantes%ROWTYPE
      INDEX BY BINARY_INTEGER;
      TYPE t_matrículas IS TABLE OF NUMBER(5)
      INDEX BY BINARY_INTEGER;
```

```
      V_nombres        t_nombres;
      V_estudiantes    t_estudiantes;
      V_matrículas     t_matrículas;
```

Asignación de valores a un tipo table

La sintaxis para la asignación de valores a un tipo tabla (TABLE) es la siguiente:

<variable_tipo_tabla>(numero) := <valor>

EJEMPLO

```
DECLARE
TYPE t_nombres IS TABLE OF estudiante.nombre%TYPE
      INDEX BY BINARY_INTEGER;
TYPE t_estudiantes IS TABLE OF estudiantes%ROWTYPE
      INDEX BY BINARY_INTEGER;
TYPE t_matrículas IS TABLE OF NUMBER(5)
      INDEX BY BINARY_INTEGER;

V_nombres        t_nombres;
V_estudiantes    t_estudiantes;
V_matrículas     t_matrículas;
BEGIN
      V_matriculas(1) := 10;

      SELECT * INTO v_estudiantes(1) FROM estudiantes
      WHERE matricula = 1000;

      V_nombre(1) := 'ANTOLIN';
      V_estudiantes(1).nombre := 'PEPE';
END;
```

Atributos de un tipo table

Los atributos que se pueden asociar a un tipo tabla (TABLE) son los siguientes:

- COUNT
- DELETE
- EXISTS (valor)
- FIRST
- LAST
- NEXT (valor)
- PRIOR (valor)

ATRIBUTO COUNT

Permite contar el número de elementos o filas de un tipo tabla.

ATRIBUTO DELETE

Permite borrar filas o elementos de un tipo tabla. Se puede utilizar de 3 formas diferentes:

- DELETE: borra todas las filas del tipo.
- DELETE(valor): borra el elemento indicado en *valor.*
- DELETE(valor1,valor2): borra los elementos entre el *valor1* y el *valor2* incluyendo a ambos.

ATRIBUTO EXISTS (VALOR)

Permite evaluar si existe el elemento cuyo índice sea equivalente a *valor* dentro del tipo.

ATRIBUTO FIRST

Nos devuelve el valor del índice del primer elemento del tipo.

ATRIBUTO LAST

Nos devuelve el valor del índice del último elemento del tipo.

ATRIBUTO NEXT (VALOR)

Nos devuelve el valor del índice del siguiente elemento al indicado.

ATRIBUTO PRIOR (VALOR)

Nos devuelve el valor del índice del elemento anterior al indicado.

Ejemplo

```
DECLARE
TYPE t_matriculas IS TABLE OF NUMBER(5)
     INDEX BY BINARY_INTEGER;
V_matricula     t_matriculas;
V_total         NUMBER;
V_indice        NUMBER;
Contador        NUMBER;

BEGIN
     V_matricula(1) := 10;
     V_matricula(2) := 20;
     V_matricula(3) := 30;

     -- Devuelve 3 filas.
     V_total := v_matricula.COUNT;

     -- Borra todas las filas.
     V_matricula.DELETE;

     FOR contador IN 1..5 LOOP
          V_matricula(contador) := 10;
     END LOOP;
```

```
      -- Borrar elementos 1 a 3 inclusive, sin rehacer
      -- la numeración.
      V_matricula.DELETE(1,3);

      -- Borra el elemento 4, ya sólo existe la Fila 5.
      V_matricula.DELETE(4);

      -- Comprueba si existe la fila 1
      IF v_matricula.EXISTS(1) THEN
           NULL;
      END IF;

      -- Crea las filas 1 a 4 y a la 5 que existe, le
      -- reasigna al valor 10.
      FOR contador IN 1..5 LOOP
           V_matricula(contador) := 10;
      END LOOP;

      -- Ahora preguntamos si existe un valor anterior al
      -- primer elemento de la tabla
      IF V_matricula.PRIOR(V_matricula.FIRST) IS NULL THEN
         DBMS_OUTPUT.PUT_LINE('No existen valores
anteriores al 1');
      END IF;

      -- Ahora preguntamos si existe un valor posterior al
      -- último elemento de la tabla
      IF V_matricula.NEXT(V_matricula.LAST) IS NULL THEN
         DBMS_OUTPUT.PUT_LINE('No existen valores
posteriores al '||V_matricula.COUNT);
      END IF;

      -- El valor devuelto es 1
      V_indice := v_matricula.FIRST;

      --El valor devuelto es 5
      V_indice := v_matricula.LAST;

      V_indice := v_matricula.FIRST;
```

```
      --El valor devuelto es 2
      V_indice := v_matricula.NEXT(1);

      --El valor devuelto es 1
      V_indice := v_matricula.PRIOR(2);
END;
```

Antolín·Muñoz·Chaparro¶
Jefe·de·Proyecto·Sistemas·Informáticos·¶
División·III·de·Aplicaciones·de·Costes·de·Personal·y·Pensiones·Públicas¶
Oficina·de·Informática·Presupuestaria·(DIR3:·EA0027952)¶
Intervención·General·de·la·Administración·del·Estado¶

VARRAYS

Un VARRAY se manipula de forma muy similar a las tablas de PL pero se implementa de forma diferente.

Los elementos en el VARRAY se almacenan comenzando en el índice 1 hasta la longitud máxima declarada en el tipo VARRAY.

La sintaxis es la siguiente:

```
TYPE <nombre_tipo> IS VARRAY (tamaño_maximo>) OF <tipo_dato>;
```

Una consideración a tener en cuenta, es que en la declaración de un VARRAY el <tipo_dato> no puede ser uno de los siguientes tipos de datos:

- BOOLEAN
- NCHAR
- NCLOB
- NVARCHAR(n)
- REF CURSOR
- TABLE
- VARRAY

Sin embargo se puede especificar el tipo utilizando los atributos %TYPE y %ROWTYPE.

Inicialización de un VARRAY

Los VARRAY deben estar inicializados antes de poderse utilizar. Para inicializarlos se utiliza un constructor (podemos inicializar el VARRAY en la sección DECLARE o bien dentro del cuerpo del bloque):

EJEMPLO:

```
DECLARE
    /* Declaramos el tipo VARRAY de cinco elementos
       y de tipo VARCHAR2*/
    TYPE t_cadena IS VARRAY(5) OF VARCHAR2(50);

    /* Asignamos los valores con un constructor */
    v_lista t_cadena:= t_cadena('Aitor', 'Alicia',
                                'Pedro',null,null);
BEGIN
    v_lista(4) := 'Tita';
    v_lista(5) := 'Ainhoa';
    dbms_output.put_line(v_lista(4));
END;
```

El tamaño de un VARRAY se establece mediante el número de parámetros utilizados en el constructor, si declaramos un VARRAY de cinco elementos pero al inicializarlo pasamos sólo tres parámetros al constructor, el tamaño del VARRAY será tres. Si se hacen asignaciones a elementos que queden fuera del rango se producirá un error.

El tamaño de un VARRAY podrá aumentarse utilizando la función EXTEND, pero nunca con mayor dimensión que la definida en la declaración del tipo. Por ejemplo la variable *v_lista* que sólo tiene 3 valores definidos, por lo que se podría ampliar hasta cinco elementos, pero no más allá.

Un VARRAY comparte con las tablas de PL todas las funciones válidas para ellas, pero además añade las siguientes:

- LIMIT
- EXTEND
- EXTEND(n,i): Añade (n) copias del elemento (i) al final del VARRAY.

ATRIBUTO LIMIT

Devuelve el número máximo de elementos que admite el VARRAY.

ATRIBUTO EXTEND

Añade uno o varios elementos NULL al final del VARRAY.

Si utilizamos EXTEND a secas, se añade un único elemento al final del VARRAY.

Si utilizamos EXTEND(n), añadimos *n* elementos al final del VARRAY

ATRIBUTO EXTEND(N,I)

Añade *N* copias del elemento *I* al final del VARRAY.

Ejemplo

```
DECLARE
-- Declaramos el tipo VARRAY de cinco elementos VARCHAR2
TYPE t_cadena IS VARRAY(5) OF VARCHAR2(50);

-- Asignamos los valores con un constructor
v_lista t_cadena:= t_cadena('Aitor', 'Alicia', 'Pedro');

v_lista2 t_cadena;



BEGIN
      -- Devuelve: 1
      dbms_output.put_line('El elemento más pequeño está en
la posición: '||v_lista.first);

      -- Devuelve: 3
      dbms_output.put_line('El        varray        tiene:
'||v_lista.count||' elementos');

      -- Devuelve: 5
      dbms_output.put_line('El  nº  máximo  de  elementos
admitidos en el varray es: '||v_lista.limit);
```

```
      -- Se añade 1 elemento
      v_lista.extend;

      -- Devuelve vacio
      dbms_output.put_line(nvl(v_lista(4),'vacio'));

      v_lista(4) := 'Tita';
      -- Devuelve: Tita
      dbms_output.put_line('Ahora el elemento 4 tiene el
valor: '||nvl(v_lista(4),'vacio'));

      v_lista2 := t_cadena();
      -- Devuelve: 0
      dbms_output.put_line('El varray2 tiene:
'||v_lista2.count||' elementos');

      -- Añade 3 elementos
      v_lista2.extend(3);

      -- Devuelve: 3
      dbms_output.put_line('El varray2 tiene ahora:
'||v_lista2.count||' elementos');

      v_lista2(1) := 'Antolin';
      -- Añade 2 elementos copia del elem. 1
      v_lista2.extend(2,1);

      -- Devuelve: antolin.
      dbms_output.put_line('Ahora el elemento 5 del varray
2 tiene el valor: '||nvl(v_lista2(5),'vacio'));
END;
```