MEDAC
Instituto Oficial de Formación Profesional

BASES DE DATOS

Tipos de datos compuestos

16

ÍNDICE

OBJETIVOS

Comprender el concepto de registro y aprender a utilizarlos.

Conocer qué es un array, comprender sus posibles utilizaciones y su implementación en PL/SQL.

Analizar el concepto de tabla anidada en PL/SQL, interpretar sus posibles utilizaciones y de qué manera se implementan en PL/SQL.

Comprender los tipos de datos tipo cursor, y estudiar su utilización e implementación en PL/SQL.

/ 1. Introducción y contextualización práctica

Tanto en PL/SQL como en cualquier otro lenguaje de programación que tenga cierta complejidad existen una serie de elementos que resultan de especial utilidad para realizar determinadas acciones, como pueden ser los registros, *arrays,* tablas anidadas y cursores que se estudiarán en este tema, y a los que se les conoce como tipos de datos compuestos.

Estas estructuras de datos son muy utilizadas para la resolución de determinados problemas que pueden tener mayor complejidad, en los que no son suficientes el uso de tipos de datos simples estudiados en temas anteriores.

En este tema, analizaremos de manera pormenorizada en qué consisten estos conceptos, y de qué manera se implementan en SQL. Al mismo tiempo, desarrollaremos diversos ejemplos prácticos de los mismos que nos ayudarán a comprender de mejor manera su utilización y funcionamiento.

Escucha el siguiente audio en el que planteamos la contextualización práctica del tema. Encontrarás su resolución en el apartado final.

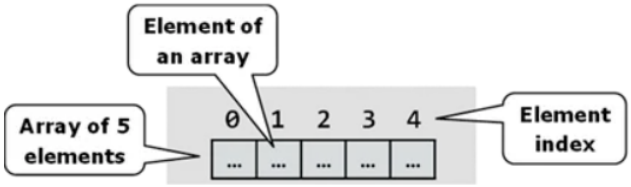

Fig. 1. Ejemplo de array de cinco elementos.
https://www.complexsql.com/varray-oracle-oracle-varray-examples/

Audio intro. "Utilidad de los tipos de datos compuestos"

https://bit.ly/3lvL36o

/ 2. Registros

Los registros en sí mismos pueden definirse como un conjunto de datos relacionados entre sí, que se encuentran agrupados en campos. Por ejemplo, puede existir el registro HABITANTE, con los campos ID, Nombre y Domicilio. Obviamente, cada uno de los campos que componen un registro puede tener distintos tipos de datos. Otro ejemplo de registro podría ser el que vemos en la figura 2.

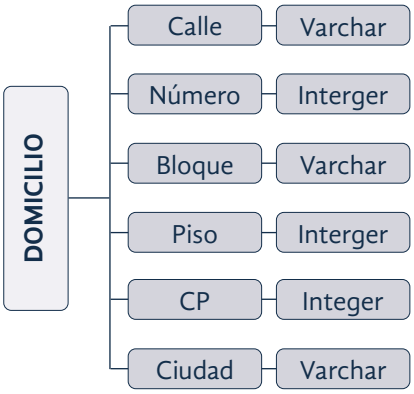

Fig. 2. Ejemplo de registro.

Se trata de estructuras muy habituales en lenguajes de programación. Como se puede comprobar en base a la definición anterior, se trata de un concepto muy parecido al de registro o fila de una tabla, lo que explica por qué son muy utilizados en PL/SQL.

La sintaxis de declaración de un registro en PL/SQL es la siguiente:

```
TYPE registro IS RECORD
(
   Campo1 TipoDatos1 [[NOT NULL] := Expresion1],
   Campo2 TipoDatos2 [[NOT NULL] := Expresion2],
   ...
   CampoN TipoDatosN [[NOT NULL] := ExpresionN],
);
```

Código 1. Sintaxis de un registro.

Para acceder a la información contenida en cualquiera de los campos del registro, se utiliza la siguiente sintaxis:

```
NombreRegistro.Campo
```

Código 2. Sintaxis de acceso a un campo de un registro.

2.1. Ejemplos de uso y asignación

En el siguiente ejemplo (implementado en Oracle SQL Developer), se realiza la creación de un registro. Además, se implementa parte de código en PL/SQL para dar de alta el registro en la tabla PRUEBA de la base de datos:

```
DECLARE
    TYPE Persona IS RECORD(
        CODIGO INTEGER(2),
        NOMBRE VARCHAR(20),
        EDAD NUMBER(3)
    );
    Juan Persona;
BEGIN
    Juan.CODIGO := 1;
    Juan.NOMBRE := 'Juan';
    Juan.EDAD := 33;
    INSERT INTO PRUEBA VALUES Juan;
END;
```

Código 3. Ejemplo de creación de un registro.

Igualmente, se pueden utilizar los atributos:

- **%TYPE:** se usa para declarar una variable, que tendrá el mismo tipo que una columna de una tabla.
- **%ROWTYPE:** indica que un registro coincide con la fila de una tabla.

En PL/SQL, se permite la **asignación** de registros de igual tipo, pero no si son de distinto tipo. En este último caso, se pueden asignar por campos, uno a uno. Tampoco es posible comparar registros entre sí, aun siendo del mismo tipo.

Como se puede comprobar en el ejemplo, está permitido hacer consultas SELECT sobre los registros, pero no del tipo INSERT.

```
DECLARE
TYPE Persona IS RECORD(
    CODIGO INTEGER(2),NOMBRE VARCHAR(20),EDAD NUMBER(3)
    );

TYPE Persona_aux IS RECORD(
    CODIGO INTEGER(2),NOMBRE VARCHAR(20),EDAD NUMBER(3)
    );
Juan Persona;
Pepe Persona_aux;
SUBTYPE Persona_prueba IS PRUEBA%ROWTYPE;
Ana Persona_prueba;

BEGIN
Juan := Ana; --Sería permitido
Juan := Pepe; -- No estaría permitido
SELECT * INTO Juan From PRUEBA; -- Se podría realizar
END;
```

Código 4. Ejemplos de asignación de registros.

Vídeo 1. "Uso y asignación de registros en Oracle Developer"

https://bit.ly/3oauO0k

/ 3. Arrays: Definición y sintaxis

Los *arrays* también son elementos típicos de los lenguajes de programación. Son un grupo de elementos del mismo tipo que se encuentran en un orden determinado, por lo que es posible hacer referencia a cada uno de ellos a través de un índice.

En PL/SQL, se utiliza la expresión **VARRAY** para su creación y referencia. Se debe tener en cuenta que a la hora de crear un *array,* se debe establecer la capacidad máxima del mismo a partir de la cual no se admitirán más elementos en el *array.* Obviamente, el tamaño mínimo ha de ser de un elemento, sin que haya un límite máximo.

La sintaxis de declaración de un *array* es la siguiente:

```
TYPE nombreArray IS VARRAY (máximo_capacidad) OF
tipo_de_elementos [NOT NULL];
```

Código 5. Sintaxis para la declaración de un array.

El tipo de elementos puede ser cualquiera de los tipos de datos de PL/SQL, con algunas excepciones como BOOLEAN, LONG, NATURAL, BINARY INTERGER, etc.

Conviene tener en cuenta que en el momento de definir un *array,* se crea vacío, por lo que, posteriormente a su creación, es necesario introducir los elementos que contendrá.

En el siguiente ejemplo, se muestra el código necesario para crear e inicializar un *array:*

```
TYPE persona_prueba IS VARRAY(100) OF Persona;
Ana persona_prueba := persona_prueba(Persona(4,'Ana',45));
Prueba persona_prueba :=persona_prueba();
```

Código 6. Ejemplo de creación de un array y de su inicialización.

Como se puede comprobar, se ha creado un *array* que puede contener hasta cien elementos del tipo PERSONA. A continuación, se han creado varios elementos dentro del mismo.

De igual forma que en el caso de los registros, se puede referenciar cualquier elemento del interior de un *array* mediante la sentencia:

```
Nombre_Array(índice_elemento)
```

Código 7. Sintaxis de acceso a un elemento del array.

/ 4. Caso práctico 1: "Registros con %TYPE y %ROWTYPE"

Planteamiento: Apoyándonos en el código correspondiente a la aplicación de gestión de habitaciones del hotel visto en la primera autoevaluación, vamos a poner en práctica los conceptos teóricos en los que se fundamenta para crear registros utilizando los atributos %TYPE y %ROWTYPE.

Nudo: Para ello, vamos a crear dos registros equivalentes entre sí. Uno mediante el uso de %TYPE y otro mediante el uso de %ROWTYPE. Además, una vez creados, podemos implementar código PL/SQL para obtener información acerca de los mismos mediante el uso de SELECT.

Desenlace: Utilizando el ejemplo de código incluido en la autoevaluación como uso de %TYPE, podemos implementar un registro similar utilizando %ROWTYPE de la siguiente manera:

```
DECLARE
   Hotel99 Hotel%ROWTYPE;
```

Código 8. Declaración de registro con %ROWTYPE.

En ambos casos, podríamos acceder a la información contenida en una de las habitaciones del hotel (por ejemplo, en la 99) mediante la utilización del siguiente código PL/SQL:

```
BEGIN
  SELECT * INTO Hotel99 FROM HOTEL WHERE ID=99;
  DBMS_OUTPUT.PUT_LINE ('Cód. Hotel   : ' || Hotel99.ID);
  DBMS_OUTPUT.PUT_LINE ('Habitaciones: ' || Hotel99.NHABS);
END;
```

Código 9. Código de acceso a un registro.

/ 5. Asignación de arrays y palabras clave

De manera similar a los registros, es posible asignar un VARRAY a otro, siempre y cuando ambos sean del mismo tipo:

```
DECLARE
TYPE prueba1 IS VARRAY(20) OF NUMBER;
TYPE prueba2 IS VARRAY(20) OF NUMBER;
una_prueba1 prueba1 := prueba1(); --Elemento array prueba1
una_prueba2 prueba2 := prueba2(); --Elemento array prueba2
otra_prueba1 prueba1 := prueba1(); --Elemento array prueba1
BEGIN
otra_prueba1 := una_prueba1; --Se puede realizar
una_prueba1 := una_prueba2; --No se puede realizar
END;
```

Código 10. Ejemplo de asignación de un VARRAY a otro.

Del mismo modo, existen una serie de palabras clave que son muy útiles para trabajar con *arrays*, tanto en PL/SQL como en otros lenguajes de programación:

- **EXTEND(a,b):** Se utiliza para extender un *array,* es decir, añadir copias de algún elemento del mismo. Mediante su uso, se añadirían 'a' copias del elemento al que se accede mediante el índice 'b'.

```
una_prueba1.EXTEND(); -- Se añade un elemento nulo
una_prueba1.EXTEND(5); -- Se añaden 5 elementos nulos
una_prueba1.EXTEND(5,2); -- Se añaden 5 copias del elemento 2
```

Código 11. Ejemplo de uso de EXTEND sobre array una_prueba1.

- **COUNT:** Su uso posibilita conocer cuántos elementos contiene un *array* en un momento determinado.

```
una_prueba1.COUNT;
```

Código 12. Ejemplo de uso de COUNT sobre array una_prueba1.

- **LIMIT:** Indica cuál es el tamaño máximo de un *array*.

```
una_prueba1.LIMIT; --Indica la capacidad (20)
```

Código 13. Ejemplo de uso de LIMIT sobre array una_prueba1.

- **FIRST:** Hace referencia al primer elemento del *array*.
- **LAST:** Hace referencia al último elemento del *array*.
- **PRIOR:** Devuelve el elemento anterior al referenciado.
- **NEXT:** En este caso, se devuelve el elemento posterior al referenciado.

/ 6. Tablas anidadas

De forma similar al caso de los *arrays,* las tablas anidadas son grupos de elementos de un mismo tipo, pero, en este caso, no tienen un límite máximo en cuanto a número de elementos, por lo que se puede afirmar que tienen un tamaño dinámico.

La sintaxis que se sigue para su declaración es la siguiente:

```
TYPE nombreTabla IS TABLE OF  tipo_de_elementos [NOT NULL];
```

Código 14. Sintaxis para la declaración de una tabla anidada.

De igual manera que ocurre con los arrays, una tabla anidada se declara vacía, por lo que es necesario incluir elementos para su uso.

Asimismo, existen ciertas palabras clave que también se pueden utilizar para tablas anidadas:

- **EXTEND(a,b)**
- **COUNT**
- **FIRST y LAST**
- **PRIOR y NEXT**
- **TRIM(a):** Se utiliza para borrar los últimos elementos de la tabla anidada.
- **DELETE(a,b):** Se utilizan para borrar los elementos de la tabla entre el índice 'a' y el 'b'. Se puede utilizar con un único argumento, en cuyo caso se borra solo el elemento correspondiente a ese índice.

En el caso de LIMIT, no tiene sentido su utilización en tablas anidadas.

En el siguiente ejemplo, se muestra un caso de uso sobre una tabla anidada con diferentes operaciones sobre la misma, indicándose el resultado que se obtiene en cada caso:

```
DECLARE
TYPE numbers IS TABLE OF NUMBER;
table_numbers numbers := numbers();
num NUMBER;
BEGIN
num := table_numbers.COUNT; --Debería devolver cero
FOR a IN 1..100 LOOP
    table_numbers.EXTEND;
    table_numbers(a) := a;
END LOOP;
num := table_numbers.COUNT; --Ahora valdría 100
table_numbers.DELETE(100);
num := table_numbers.LAST; -- Su valor sería 99
num := table_numbers.FIRST;-- Su valor sería 1
table_numbers.DELETE(1);
num := table_numbers.FIRST;-- Su valor sería 2
END;
```

Código 15. Ejemplo de uso de tabla anidada.

/ 7. Cursores

Escucha el siguiente audio para comprender el concepto de cursor:

Audio 1. "Cursores en PL/SQL"
https://bit.ly/3lqAVMh

Los cursores cuentan con diversos atributos a través de los cuáles se puede obtener información sobre el mismo:

ATRIBUTO	USO
%FOUND	Devuelve TRUE si el último FETCH ha devuelto una fila.
%NOTFOUND	Devuelve TRUE si el último FETCH no ha devuelto ningún valor.
%ISOPEN	Devuelve TRUE si el cursor está abierto.
%ROWCOUNT	Devuelve el número de filas procesadas en el cursor.

Tabla 1. Atributos de los cursores.

Los cursores pueden dividirse en dos tipos: implícitos y explícitos.

a. **Cursores implícitos**

Estos cursores se utilizan cuando la consulta devuelve un único registro. En este caso, no es posible utilizar la instrucción FETCH para su control.

b. **Cursores explícitos**

Este tipo de cursores se utilizan cuando la consulta devuelve varios registros. La sintaxis que se sigue para la declaración de un cursor de manera explícita es la siguiente:

```
CURSOR nombre IS SELECT...
```

Código 16. Sintaxis de declaración de un cursor.

Un ejemplo de uso podría ser:

```
CURSOR cursorEmpleados IS SELECT * FROM EMPLEADOS;
```

Código 17. Uso del cursor.

Mediante la instrucción **"OPEN nombreCursor",** se abre el mismo, ejecutándose la sentencia SELECT a la que se refiere y colocando el puntero en la primera fila. De igual manera, mediante la instrucción **"CLOSE nombreCursor",** se libera la memoria ocupada por el cursor, permitiendo abrir uno nuevo o, incluso, reabrir el mismo. En el siguiente vídeo, se muestra un ejemplo de cómo se recorre un cursor de manera completa:

Vídeo 2. "Ejemplo de uso de cursores implícitos y explícitos"
https://bit.ly/37qCvsP

7.1. Operar con cursores

a. **Cursores explícitos con parámetros**

Los cursores explícitos pueden aceptar parámetros, algo que es de especial utilidad cuando se pretende que el resultado de un cursor tenga dependencia de una variable. Para ello, se debe indicar en la declaración del cursor entre paréntesis, tal y como se muestra en el siguiente ejemplo:

```
DECLARE
CURSOR cursor_empleado(id NUMBER, salario NUMBER) IS
    SELECT * FROM EMPLEADOS
    WHERE ID_EMPLEADO = id
    AND SALARIO = salario;
BEGIN
    OPEN cursor_empleado(1,1100);
    --CÓDIGO
    CLOSE cursor_empleado;
END;
```

Código 18. Sintaxis de declaración de un cursor con parámetros.

En este caso, en el código PL/SQL, se hace referencia a través del cursor al departamento número 1 y salario 1100.

El valor de estos parámetros debe indicarse en cualquier caso en el momento en el que se abre el cursor, ya sea a través de la sentencia OPEN o a través de un bucle FOR.

b. **Actualizar datos al recorrer cursores**

Es muy común que al mismo tiempo que se está recorriendo un cursor, se lleven a cabo actualizaciones de registros sobre el mismo, lo que puede producir problemas de bloqueo.

Para evitar estos problemas, se incluye la cláusula **"FOR UPDATE"** en la declaración del cursor, al final del SELECT. De manera opcional, puede incluirse la cláusula **"NOWAIT"** para que el programa no se pare si la tabla está siendo utilizada por otro usuario.

```
CURSOR nombre IS SELECT...
FOR UPDATE OF campo [NOWAIT]
```

Código 19. Sintaxis de declaración de un cursor con FOR UPDATE.

A continuación se indica un ejemplo de utilización de esta sentencia:

```
DECLARE
CURSOR cursor_empleado IS
    SELECT id_empleado,nombre,apellidos,salario
    FROM EMPLEADOS
    WHERE departamentos.id_empleado = empleados.id_empleado
    FOR UPDATE OF salario NOWAIT;
BEGIN

--CÓDIGO

END;
```

Código 20. Código ejemplo para actualizar datos al recorrer cursores.

/ 8. Caso práctico 2: "Prácticas con cursores en PL/SQL"

Planteamiento: En este caso práctico, vamos a repasar los conceptos de cursores estudiados anteriormente.

Nudo: Para ello, vamos a declarar un cursor implícito y otro cursor explícito, de manera que podamos poner en práctica los conocimientos teóricos estudiados.

Vamos a indicar el código que sería necesario implementar para desarrollar:

- Un cursor implícito cualquiera. Utiliza el comando %ROWTYPE.
- Un cursor explícito que muestre el nombre y salario de los cinco primeros empleados.

Desenlace: En el caso del cursor implícito, al no ser necesario declararlo como cursor, sería válido el siguiente código PL/SQL:

```
DECLARE
    Domiciliol Domicilio%ROWTYPE;
BEGIN
....
END;
```

Código 21. Código PL/SQL.

En el caso del cursor explícito, podría ser válido el siguiente código PL/SQL:

```
DECLARE
    CURSOR prueba IS
    SELECT nombre, salario FROM EMPLEADOS;

    nombre VARCHAR(20);
    salario NUMBER(7,0);
BEGIN
    OPEN prueba;
    LOOP
        FETCH prueba INTO nombre, salario;
        --Se muestran los 5 primeros
        EXIT WHEN (prueba%ROWCOUNT > 5);
        DBMS_OUTPUT.PUT_LINE(prueba%ROWCOUNT || salario || nombre);
    END LOOP;
    CLOSE prueba;
END;
```

Código 22. Código PL/SQL.

/ 9. Resumen y resolución del caso práctico inicial

En este tema, hemos estudiado **los tipos de datos compuestos,** estructuras que es necesario conocer y dominar para poder implementar programas robustos y completos en PL/SQL. En primer lugar, analizamos el concepto de **registro,** cómo se declaran y cómo se deben utilizar, haciendo uso de diversos ejemplos para profundizar con mayor detalle en su estudio.

Posteriormente, hemos estudiado el concepto de ***array,*** estructura básica y comúnmente utilizada en cualquier lenguaje de programación de cierta complejidad. Además, hemos verificado diversos ejemplos prácticos de uso. De igual manera, hemos estudiado, a través de ejemplos prácticos, las **tablas anidadas,** así como sus opciones de uso.

Para finalizar el tema, se ha analizado el concepto de **cursor,** que es necesario comprender y manejar para poder aprovechar la potencia de PL/SQL. Se han estudiado los tipos de cursores existentes (implícitos y explícitos) y las diferencias entre los mismos, realizándose distintos ejemplos prácticos para ayudar a ello.

Resolución del caso práctico inicial

- En los lenguajes de programación de cualquier tipo, existen estructuras que ayudan a tratar la información y que funcionan de manera similar a una memoria a la que se puede acceder en tiempo real para leer y modificar su contenido. **¿A qué tipo de estructuras nos estamos refiriendo?**

- En este caso, nos estaríamos refiriendo a estructuras tipo registro, tabla anidada o tipo *array,* que son comunes a la mayor parte de los lenguajes de programación existentes. Su uso es común y básico para realizar operaciones en las que se trabaja con un conjunto de elementos de un mismo tipo, ya sean números, caracteres, objetos, etc.

- En el caso de programación de bases de datos, existen estructuras similares, pero con particularidades propias, de manera que son muy útiles en este entorno de trabajo. **¿Cuáles son estas estructuras? ¿Para qué pueden ser necesarias?**

- Como hemos estudiado, los cursores son estructuras características de la programación en bases de datos, ya que se utilizan para almacenar los diferentes registros (filas) obtenidos a partir de una consulta sobre la base de datos. Permiten trabajar con esta información y procesarla para alcanzar el objetivo que se desee, por lo que su uso es muy habitual en este entorno. El concepto de cursor podría ser similar al de un *array* formado por filas de una determinada tabla que cumplen, todas ellas, las especificaciones indicadas en la consulta de la que son resultado.

/ 10. Bibliografía

Oppel, A. (2009). *Databases: A Beginner's Guide.* Madrid, España: McGraw-Hill.

Elmasri, R. y Navathe, S. (2007). *Fundamentos de Bases de Datos* (5.a. ed.). Madrid, España: Pearson Addison-Wesley

López, I.; Castellano, M.J., y Ospino, J. (2011). *Bases de datos.* Madrid, España: Garceta

Cabrera, G. (2011). *Sistemas gestores de bases de datos.* Madrid, España: Paraninfo.

Pérez Marqués, M. (2016). *Administración básica de bases de datos con Oracle 12c SQL*. Madrid, España: Alfaomega

Urman S.; Hardman, R., y McLaughlin, M. (2004). *Oracle database 10g. PL/SQL Programming.* Oracle Press.