UD 07 - USO DE BASES DE DATOS OBJETO-RELACIONALES

1. Características de las bases de datos objeto-relacionales

Las **bases de datos objeto-relacionales** son aquellas que han evolucionado del modelo relacional tradicional a un modelo híbrido que utiliza tecnología oprientadas a objetos.

Los esquemas de la base de datos soportan DDL, DML y, además, clases, objetos, herencia... Da soporte a una extensión del modelo de datos con creación personalizada de tipo de datos y métodos.

Oracle implementa el modelo orientado a objetos como **extensión del modelo relacional**, soportando la funcionalidad estandar de bases de datos relacionales.

- El modelo objeto-relacional ofrece ventajas de las técnicas orientadas a objetos (reutilización, uso intuitivo de objetos) a la par que mantiene alta capacidad de concurrencia y buen rendimiento de las bases de datos relacionales.
- Es un método de acceso **a alto nivel**. Por debajo de la capa de objetos, los datos seguirán estando almacenados en columnas y tablas.
- La reutilización de objetos permite desarrollar aplicaciones de bases de datos de forma más rápida y eficiente. Además, los desarrolladores puede acceder directamente a las mismas estructuras de datos desde sus aplicaciones orientadas a objetos.
- Los datos son solo datos. Pero los objetos pueden contener además **acciones sobre los datos**
- En PL/SQL la POO está basada en tipos de objetos.
- En las bases de datos objeto-relacionales los dominios de la base de datos ya no son solo atómicos luego no se cumple la 1FN, debido a que las tuplas también pueden ser una relación, que llevará a relaciones de relaciones.

2. Tipos de datos objeto

Tipo de dato objeto es un tipo de dato compuesto definido por el usuario. Representa a una **estructura de datos** y a **funciones y procedimientos** para manipular datos.
Las **variables** de un tipo de dato escalar pueden almacenar **un único valor** y las **colecciones varios siendo todos del mismo tipo**.
Los tipos de datos objetos permiten almacenar datos de distintos tipos, posibilitando asociar código a dichos datos

Los tipos de datos completos se componen de variables (atributos, propiedades) y métodos (funciones y procedimientos, acciones)

Al definir el tipo de objeto, se crea una **plantilla abstracta**.
Aunque los atributos son públicos (visibles desde otros programas cliente), la manipulación de datos deberá hacerse solo a través de los métodos declarados en el tipo objeto (ellos pueden hacer un chequeo de los datos para que se mantenga un estado apropiado en los mismos)

En la ejecución, la aplicación crea **instancias** de un tipo objeto, referencias a objetos reales con valores asignados en sus atributos.

3. Definición de tipos de objeto

La definición o declaración del tipo de objeto está dividida en una especificación y un cuerpo.
Especificación:

- Define el interfaz de programación donde se declaran los atributos y las operaciones para manipular los datos.
- Toda la información que un programa necesita para usar los métodos lo encuentra dentro de la especificación.

- En ella los atributos deben ser declarados **antes que** los métodos.
- Como mínimo debe tener un atributo declarado y un máximo de 1000.
- Los métodos son opcionales. Puede crearse sin método
- En ella pueden declararse atributos y métodos pero no constantes (CONSTANTS), excepciones (EXCEPTIONS), cursores (CURSORS) o tipos (TYPES)
  Cuerpo:
- Se implementa el código fuente de los métodos.
- Si la especificación solo declara atributos, no es necesario declarar el cuerpo
- No pueden declararse atributos en el cuerpo
- Todas las declaraciones realizadas en la especificación son públicas (visibles fuera del tipo del objeto)

Definición del tipo objeto

Para definir el tipo de objeto se usa la sentencia `CREATE TYPE`.... `AS OBJECT`

Eliminar el tipo de objeto

```
DROP TYPE nombre_tipo_objeto;
```

3.1. Declaración de atributos

La declaración de atributos es parecida a la declaración de variables. Como en otras ocasiones puede ser cualquiera de los tipos de Oracle excepto:

- LONG y LONG RAW
- ROWID y UROWID
- Tipos específicos de PL/SQL: BINARY_INTEGER (y subtipos), BOOLEAN, PLS_INTEGER, RECORD, REF CURSOR, %TYPE y %ROWTYPE
- Tipos definidos en un paquete PL/SQL

No se pueden inicializar con := , ni usar DEFAULT, ni NOT NULL

El tipo de dato declarado puede ser **otro tipo de objeto**.

Definición del tipo objeto con atributos

```
CREATE (OR REPLACE) TYPE Usuario AS OBJECT(
        login VARCHAR2(10),
        nombre VARCHAR2(30),
        f_ingreso DATE,
        credito NUMBER
);
```

Modificación del tipo objeto con atributos

```
ALTER TYPE Usuario DROP ATTRIBUTE f_ingreso;
ALTER TYPE Usuario ADD ATTRIBUTE (apellidos VARCHAR2(40), localidad VARCHAR2(50));
ALTER TYPE Usuario
        ADD ATTRIBUTE cp VARCHAR2(5),
        MDOIFY ATTRIBUTE nombre VARCHAR2(35);
```

3.2. Definición de métodos

Con `MEMBER` o `STATIC`

Definición de especificación

```
CREATE (OR REPLACE) TYPE Usuario AS OBJECT(
        login VARCHAR2(10),
        nombre VARCHAR2(30),
        f_ingreso DATE,
        credito NUMBER,
        MEMBER PROCEDURE incrementoCredito(inc NUMBER)
);
```

Definición de cuerpo

```
CREATE  OR REPLACE TYPE BODY Usuario as
        MEMBER PROCEDURE incrementoCredito(inc NUMBER) IS
                BEGIN
                        credito := credito + inc
                END incrementoCredito;
END;
```

Debe existir el correspondiente cuerpo por cada especificación o el método debe declararse como `NOT INSTANTIABLE`, para indicar que el cuerpo del método se encontrará en un subtipo de ese tipo de objeto.

El código fuente de los métodos no solo puede escribirse en PL/SQL. También en otros lenguajes como Java o C.

Con `ALTER TYPE` puede añadirse, modificar o eliminar métodos de un tipo de objeto existente.

3.3. Parámetro SELF

`SELF` se refiere a una instancia (objeto) del mismo tipo de objeto. Siempre se declara automáticamente aunque no esté declarado específicamente.

- En funciones MEMBER: Si no se declara su modo por defecto se toma como INT
- En procedimientos MEMBER: Si no se declara su modo por defecto se toma como INT OUT
- Los métodos STATIC no puede usar este parámetro especial.
- No puede especificar el modo OUT para el parámetro SELF
- Al hacer referencia a SELF dentro del cuerpo de un método se está haciendo referencia al objeto que invoca ese método. `SELF.nombre_atributo`, `SELF.nombre_metodo`....(Equivalente a `this`, vamos)

3.4. Sobrecarga

Los métodos pueden ser sobrecargados: **Utilizar el mismo nombre para métodos diferentes**, siempre que sus parámetros sean diferentes en cantidad o en tipo de dato.
Se ejecutará aquel en el que haya una coincidencia entre los parámetros actuales y los formales del método declarado.

```
MEMBER PROCEDURE setNombre(Nombre VARCHAR2)
MEMBER PROCEDURE setNombre(Nombre VARCHAR2, Apellidos VARCHAR2)
```

3.5. Métodos constructores

Cada tipo de objeto tiene un método constructor.
Oracle crea uno por defecto.

Pueden declararse métodos propios constructores reescribiendo ese método o definiendo un nuevo con otros parámetros.

Se puede hacer en el nuevo método constructor una verificación de que los datos que se van a asignar a los atributos son correctos (por ejemplo)

Para reemplazar el constructor por defecto debe usarse la sentencia `CONSTRUCTOR FUNCTION` seguida del nobre del tipo de objeto en el que se encuentra. Después se indican los parámetros necesarios. Por último el valor de retorno de la función que es el propio objeto usando la cláusula `RETURN SELF AS RESULT`

Declaración y cuerpo

```
CONSTRUCTOR FUNCTION Usuario(login VARCHAR2, credito NUMBER)
        RETURN SELF AS RESULT

CREATE OR REPLACE TYPE BODY Usuario AS
        CONSTRUCTOR FUNCTION Usuario(login VARCHAR2, credito NUMBER)
                RETURN SELF AS RESULT
        IS
                BEGIN
                        IF (credito >= 0) THEN
                                SELF.credito := credito;
                        ELSE
                                SELF.credito := 0;
                        END IF;
                        RETURN;
                END;
END;
```

Métodos MAP / ORDER

Las instancias de tipo de objeto no tienen orden predefinido.
Para establecerlo debe crearse método `MAP` o método `ORDER`

MAP

Para comparar. Debe empezar la declaración con la palabra MAP. Solo puede haber un método declarado.
`MAP MEMBER FUNCTION ordenarUsuario RETURN VARCHAR2`

El cuerpo del método debe retornar el valor que se usará para realizar las comparaciones.

```
CREATE OR REPLACE TYPE BODY Usuario AS
        MAP MEMBER FUNCTION ordenarUsuario RETURN VARCHAR2 IS
        BEGIN
                RETURN (apellidos || ' ' || nombre);
        END ordenarUsuario;
END;
/
```

ORDER

- Permitirá establecer un orden. Debe empezar la declaración con la palabra ORDER
- Solo puede haber un método ORDER.
- Debe retornar un valor numérico que permita establecer el orden entre los objetos. (ejemplo -1, 0, 1)
  ``

```
CREATE OR REPLACE TYPE BODY Usuario AS
        ORDER MEMBER FUNCTION ordenUsuario(u Usuario) RETURN INTEGER IS
        BEGIN
                /* La función substr obtiene una subcadena desde la posición indicada hasta
el final*/
                IF substr(SELF.login, 7) < substr(u.login, 7) THEN
                        RETURN -1;
                ELSIF substr(SELF.login, 7) > substr(u.login, 7) THEN
                        RETURN 1;
                ELSE
                        RETURN 0;
                END IF;
```

```
        END;
END;
```

Se puede declarar un método MAP o un método ORDER pero no los dos. Cuando se vaya a ordenar o a mezclar un alto número de objetos es preferible usar el método MAP porque ORDER sería menos eficiente

4. Utilización de objetos

Veamos cómo se declaran variables para almacenar objetos, se dan valores iniciales a los atributos, se accede a su contenido y se llama a los métodos que ofrece el tipo de objeto.

4.1. Declaración de objetos

Cuando el tipo de objeto ha sido creado, puede usarse para declarar variables de objetos de ese tipo en cualquier bloque PL/SQL, subprograma o paquete. Puede usarse como tipo de dato para variable, atributo, elemento de tabla, parámetro formal o resultado de una función igual que se usaban los tipos habituales.

Si hemos creado un objeto usuario, podemos declarar una variable:
`u1 Usuario;`

Puede ponerse en la declaración de un procedimiento para que se pase un objeto por atributo: `PROCEDURE setUsuario(u IN Usuario)` y pasarlo sin más `setUsuario(u1)`

La función puede retornar objetos: `FUCTION getUsuario(codigo INTEGER) RETURN Usuario;`

Los objetos se crean durante la ejecución del código como instancias y cada uno puede contener valores diferentes en sus atributos.

Los **ámbitos** siguen las reglas habituales de PL/SQL. En el bloque o subprograma los objetos se crean cuando se entra en ese bloque o subprograma y se destruyen automáticamente cuando se sale de ellos. En el paquete, se instancian cuando se hace referencia al paquete y dejan de existir cuando finaliza la sesión en la BBDD.

4.2. Inicialización de objetos

Para instanciar el objeto se debe llamar al método constructor usando `NEW` seguido del nombre del objeto (lo de siempre, vamos).
El orden de los parámetros debe coincidir con cómo están declarados los atributos, así como sus tipos de datos (obvio).

```
Usuario := NEW Usuario('luisitom','LUIS','TOMAS UREÑA', '01/02/03', 100);
```

Lo habitual sería incluso inicializarlos en la declaración (y eso que nos quitamos)
En estos casos puede suprimirse el NEW

```
Usuario := NEW Usuario('luisitom','LUIS','TOMAS UREÑA', '01/02/03', 100);
```

El método constructor puede llamarse en cualquier lugar en dónde se llame a una función habitualmente. Podría por ejemplo usarse como parte de uan expresión.

Hay posibilidad de usar los **nombres de los parámetros formales** al llamar al constructor, en lugar de usar el modelo posicional de los parámetros. Así no es necesario respetar el orden:

```
DECLARE
u1 Usuario;
BEGIN
u1 := NEW Usuario('user1', -10);
/* Se mostrará el crédito como cero, al intentar asignar un crédito negativo */
dbms_output.put_line(u1.credito);
```

```
END;
/
```

4.3. Acceso a los atributos de objetos

Se accede con la notación de punto. `nombre_objeto.nombre_atributo`

```
unNombre := usuario1.nombre;
dbms_output.put_line(usuario1.nombre);
```

Se modifica:

```
usuario1.nombre:= 'Nuevo Nombre';
```

Los objetos pueden ser encadenados para acceder a tipos de objetos anidados.

Si se intenta **acceder a un objeto no inicializado** da excepción `ACCESS_INTO_NULL`
Puede comprobarse si un objeto es NULL con `IS NULL`
Si se intenta **llamar a un método de un objeto no inicializado** da excepción `NULL_SELF_DISPATCH`
Si se pasa como parámetro de IN un NULL, se evalúa como NULL
Si el parámetro es de tipo OUT o IN OUT, lanza excepción al intentar modificar sus atributos.

4.4. Llamada a los métodos de objetos

Utilizando un punto entre el nombre del objeto y el método.
`credito := usuario1.getCredito();`
Las llamadas a métodos pueden encadenarse.
`sitio1.getUsuario.setNombreCompleto('Juan', 'García Fernández');`
Los métodos `MEMBER` son invocados usando instancia del objeto
`nombre_objeto.metodo()`
Los métodos `STATIC` son invocados usando el tipo del objeto
`nombre_tipo_objeto.metodo()`

4.5. Herencia

PL/SQL admite herencia simple de objetos, mediante la cual se pueden definir subtipos de los tipos de objeto. Estos subtipos **contienen todos los atributos y métodos del tipo padre** y además atributos y métodos adicionales o incluso sobreescribir métodos del tipo padre.

Para que pueda heredarse un objeto debemos poner al final de su declaración `NOT FINAL` (porque por defecto se declaran como `FINAL`).

Para indicar que un tipo de objeto es heredado se usa la palabra reservada `UNDER`:

```
CREATE OR REPLACE TYPE Envio AS OBJECT(
        referencia VARCHAR2(5),
        fecha DATE
);
/
CREATE OR REPLACE TYPE Paquete UNDER Envio (
        peso NUMBER(3)
);
/
CREATE OR REPLACE TYPE Sobre UNDER Envio (
        peso NUMBER(3)
);
```

```
DECLARE
        vEnvioMaria Paquete;
        vEnvioPepe Sobre;
BEGIN
        vEnvioMaria:= NEW Paquete('20001','P3','Normal');
```

```
        vEnvioPep := NEW Sobre (('20002','P3','Urgente',5,10,5,5,250);
```

Si no se indica lo contrario, siempre se pueden crear instancias de los tipos declarados.
Con la opción `NOT INSTANTIABLE` se puede declarar tipos de objetos de los que no se pueden crear objetos. Estos tipos tendrán como función ser padres de otros objetos.

En la creación de un objeto se puede observar que se asignan valores para todos los atributos incluyendo los heredados.

```
CREATE TYPE Persona AS OBJECT (
        nombre VARCHAR2(20),
        apellidos VARCHAR2(30)
) NOT FINAL;
/

CREATE TYPE UsuarioPersona UNDER Persona (
        login VARCHAR(30),
        f_ingreso DATE,
        credito NUMBER
);
/

DECLARE
        u1 UsuarioPersona;
BEGIN
        u1 := NEW UsuarioPersona('nombre1', 'apellidos1', 'user1', '01/01/2001', 100);
        dbms_output.put_line(u1.nombre);
END;
/
```

5. Tipos de datos colección

Los **tipos de datos colección** son ofrecidos por Oracle para almacenar en memoria un conjunto de datos de un tipo determinado. (Parecidos a lenguajes y matrices en otros lenguajes).
Las colecciones solo pueden tener una dimensión y los elementos se indexan mediante valor numérico o cadena de caracteres.

Oracle ofrece:

- `VARRAY`. Se le establece dimensión máxima. Al tener longitud fija, la eliminación de elementos no ahorra espacio en memoria.
- `NESTED TABLE`. Tabla anidada. Puede almacenar cualquier número de elementos. Es de tamaño dinámico. No tienen que existir forzosamente valores para todas las posiciones de la colección.
- **Arrays asociativos**. Usan valores arbitrarios para sus índices. No tienen que ser necesariamente consecutivos.

Para almacenar número fijo de elementos, recorrerlos de forma ordenada u obtener y manipular toda la colección como un valor -> VARARRAY
Para ejecutar consultas de forma eficiente, manipular número arbitrario, realizar operaciones de inserción, actualización, borrado de forma masiva -> NESTED TABLE.

Las colecciones pueden ser declaradas como una instrucción SQL o en el bloque de declaraciones del programa PL/SQL. El tipo de dato de los elementos que puede contener una colección puede ser cualquiera excepto REF_CURSOR. No pueden ser de los tipos BINARY_INTEGER, `PLS_INTEGER`, BOOLEAN, `LONG`, LONG RAW, `NATURAL`, NATURALN, `POSITIVE`, POSITIVEN, `REF CURSOR`, SIGNTYPE, `STRING`.

Cualquier tipo de objeto previamente declarado puede ser usado como tipo de elemento para una colección.

La tabla de la base de datos puede contener columnas que sean colecciones y sobre ellas hacer operaciones de consulta y manipulación de datos igual que se hace con tablas con los tipos de datos habituales.

5.1. Declaración y uso de colecciones

La declaración de las colecciones sigue el formato siguiente:

```
TYPE nombre_tipo IS VARRAY (tamaño_max) OF tipo_elemento;
TYPE nombre_tipo IS TABLE OF tipo_elemento;
TYPE nombre_tipo IS TABLE OF tipo_elemento INDEX BY tipo_índice;
```

indicando el nombre de la colección, el tamaño máximo en el caso del VARRAY y el tipo de elemento que la compone. El tipo_indice representa el tipo de dato que se usará para el índice. Puede ser PLS_INTEGER, BINARY_INTEGER, VARCHAR2.

Si se hace en SQL, fuera del subprograma se debe declarar como `CREATE (OR REPLACE) TYPE nombre_tipo IS...`

Hasta que la colección no se inicialice, esta es NULL. Para inicializarla debe usarse el constructor pasando los valores iniciales de la colección. Ejemplo:

```
DECLARE
        TYPE Colores IS TABLE OF VARCHAR(10);
        misColores Colores;
BEGIN
        misColores := Colores('Rojo', 'Naranja', 'Amarillo', 'Verde', 'Azul');
END;
```

No solo en el bloque, sino en la zona de declaraciones también puede ser inicializada:

```
DECLARE
        TYPE Colores IS TABLE OF VARCHAR(10);
        misColores Colores := Colores('Rojo', 'Naranja', 'Amarillo', 'Verde', 'Azul');
```

Para obtener uno de los elementos se debe indicar el nombre de la colección y su indice `misColores(2)`
El contenido de la posición puede modificarse `misColores(3) := 'Gris'`

```
CREATE TYPE ListaColores AS TABLE OF VARCHAR2(20);
/
CREATE TABLE flores (nombre VARCHAR2(20), coloresFlor ListaColores)
        NESTED TABLE coloresFlor STORE AS colores_tab;

DECLARE
        colores ListaColores;
BEGIN
        INSERT INTO flores VALUES('Rosa', ListaColores('Rojo','Amarillo','Blanco'));
        colores := ListaColores('Rojo','Amarillo','Blanco','Rosa Claro');
        UPDATE flores SET coloresFlor = colores WHERE nombre = 'Rosa';
        SELECT coloresFlor INTO colores FROM flores WHERE nombre = 'Rosa';
END;/
```

6. Tablas de objetos

Los objetos pueden almacenarse en tablas de igual manera que los tipos de datos habituales.
Los tipos de datos objetos pueden estar en una tabla exclusivamente formada por elementos de ese tipo o como un tipo de columna más entre otros tipos de datos.

Tabla exclusivamente formada por un tipo de dato objeto (Tabla de objetos) debe usarse:

```
CREATE TABLE NombreTabla OF TipoObjeto
```

Si una tabla hace uso de un tipo de objeto **no puede eliminarse ni modificar la estructura de dicho tipo de objeto**. (No puede volver a definirse).
Además previamente debe estar declarado ese tipo de objeto.

Los atributos del tipo de objeto se muestran como si fueran las columnas de la tabla.

6.1. Tablas con columnas tipo objeto

Cuando es una columna más en una tabla simplemente debe hacerse como si fuera una columna más:

```
CREATE TABLE Gente (
        dni VARCHAR2(10),
        unUsuario Usuario,
        partidasJugadas SMALLINT
);
```

Los datos del campo `unUsuario` se muestran como integrantes de cada objeto `Usuario`.

6.2. Uso de sentencia SELECT

Las sentencia SELECT puede usarse para obtener datos de las filas almacenadas en tablas de objetos o en tablas de columnas de tipos de objetos.
Si se trata de tablas con columnas de tipo objeto el acceso a los atributos debe hacerse indicando previamente el nombre asignado a la columna que contiene los objetos.

```
SELECT g.unUsuario.nombre, g.unUsuario.apellidos FROM Gente g;
```

6.3. Inserción de objetos

Para insertar el objeto, debe suministrarse a la sentencia insert un objeto instanciado de su tipo de objeto correspondiente.

```
DECLARE
        u1 Usuario;
        u2 Usuario;
BEGIN
        u1 := NEW Usuario('luitom64', 'LUIS', 'TOMAS BRUNA', '24/10/2007', 50);
        u2 := NEW Usuario('caragu72', 'CARLOS', 'AGUDO SEGURA', '06/07/2007', 100);
        INSERT INTO UsuariosObj VALUES (u1);
        INSERT INTO UsuariosObj VALUES (u2);
END;
```

También se podría crear el objeto dentro de la sentencia INSERT directamente
`INSERT INTO UsuariosObj VALUES (Usuario('luitom64', 'LUIS', 'TOMAS BRUNA', '24/10/2007', 50));`

Igual en tablas con columnas de tipo objeto

```
INSERT INTO Gente VALUES ('22900970P', Usuario('luitom64', 'LUIS', 'TOMAS BRUNA',
'24/10/2007', 50), 54);
INSERT INTO Gente VALUES ('62603088D', u2, 21);
```

6.4. Modificación de objetos

Para modificar un objeto almacenado es igual que siempre también.

Si se trata de tabla de objeto se hace referencia a los atributos justo detrás del nombre de la tabla. Si es con columnas de tipo objeto se debe referenciar al nombre de la columna que contiene los objetos.

```
UPDATE UsuariosObj u
        SET u.credito = 0
        WHERE u.login = 'luitom64';
```

```
UPDATE Gente g
        SET g.unUsuario.credito = 0
        WHERE g.unUsuario.login = 'luitom64';
```

También puede cambiarse todo un objeto por otro.

```
UPDATE Gente g
        SET g.unUsuario = Usuario('juaesc82', 'JUAN', 'ESCUDERO LARRASA', '10/04/2011', 0)
        WHERE g.unUsuario.login = 'caragu72';
```

6.5. Borrado de objetos

Similar a como se hacía.

```
DELETE FROM UsuariosObj u WHERE u.credito = 0;
DELETE FROM Gente g WHERE g.unUsuario.credito = 0;
```

6.6. Consultas con función VALUE

Para hacer referencia a un objeto en lugar de a alguno de sus atributos puede usarse la función `VALUE` junto con el nombre de la tabla de objetos o su alias.

```
INSERT INTO Favoritos SELECT VALUE(u) FROM UsuariosObj u WHERE u.credito >= 100;

SELECT u.login FROM UsuariosObj u JOIN Favoritos f ON VALUE(u)=VALUE(f);
```

Cuando es una columna de tipo objeto la referencia que se hace a la columna permite obtener directamente un objeto sin necesidad de VALUE

```
SELECT g.dni FROM Gente g JOIN Favoritos f ON g.unUsuario=VALUE(f);
```

Con la cláusula `INTO` puede guardarse en variables el objeto obtenido con VALUE

```
DECLARE
        u1 Usuario;
        u2 Usuario;
BEGIN
        SELECT VALUE(u) INTO u1 FROM UsuariosObj u WHERE u.login = 'luitom64';
        dbms_output.put_line(u1.nombre);
        u2 := u1;
        dbms_output.put_line(u2.nombre);
END;
```

6.7. Referencias a objetos

El paso de objetos es ineficiente si son de gran tamaño. Es más conveniente pasar un puntero a dicho objeto. Eso se puede hacer como una referencia (REF).

Se crea usando el modificador REF delante del tipo de objeto y se puede usar con variables, parámetros, capos, atributos, variables de entrada o salida...

Solo pueden hacerse referencia a tipos de objetos que han sido declarados previamente.

```
CREATE OR REPLACE TYPE Partida AS OBJECT (
        codigo INTEGER,
        nombre VARCHAR2(20),
        usuarioCreador REF Usuario
);
/
```

```
DECLARE
        u_ref REF Usuario;
        p1 Partida;
BEGIN
        SELECT REF(u) INTO u_ref FROM UsuariosObj u WHERE u.login = 'luitom64';
        p1 := NEW Partida(1, 'partida1', u_ref);
END;
/
```

El orden de declaración no puede invertirse salvo en caso de hacer una **declaración de tipo anticipada**. Se indica el nombre del objeto que se detallará más adelante.

```
CREATE OR REPLACE TYPE tipo2;
/
CREATE OR REPLACE TYPE tipo1 AS OBJECT (
        tipo2_ref REF tipo2
        /*Declaración del resto de atributos del tipo1*/
);
/
CREATE OR REPLACE TYPE tipo2 AS OBJECT (
        tipo1_ref REF tipo1
        /*Declaración del resto de atributos del tipo2*/
);
```

6.8. Navegación a través de referencias

No se puede acceder directamente a los atributos de un objeto referenciado que se encuentre almacenado en una tabla.
Para eso debe usarse la función `DEREF` que toma referencia al objeto y retorna el valor del objeto.

Suponiendo que disponemos de las siguientes variable declaradas:

```
u_ref REF Usuario;
u1 Usuario;
```

Si u_ref hace referencia a un objeto de tipo Usuario que se encuentra en la tabla UsuariosObj, para obtener información sobre alguno de los atributos de dicho objeto referenciado, hay que utilizar la función DEREF.

Esta función se utiliza como parte de una consulta SELECT por lo que hay que utilizar una tabla tras la cláusula FROM. Esto puede resultar algo confuso, ya que las referencias a objetos apuntan directamente a un objeto concreto que se encuentra almacenado en una determinada tabla. Por tanto, no debería ser necesario indicar de nuevo en qué tabla se encuentra. Realmente es así. Podemos hacer referencia a cualquier tabla en la consulta, y la función DEREF nos devolverá el objeto referenciado que se encuentra en su tabla correspondiente.

La base de datos de Oracle ofrece la **tabla** DUAL para este tipo de operaciones. Esta tabla es creada de forma automática por la base de datos, es accesible por todos los usuarios, y **tiene un solo campo y un solo registro**. Por tanto, es como una tabla comodín que devuelve una única fila con una única columna.

```
SELECT DEREF(u_ref) INTO u1 FROM Dual;
dbms_output.put_line(u1.nombre);
```

Por tanto, para obtener el objeto referenciado por una variable REF, debes **utilizar una consulta sobre cualquier tabla**, independientemente de la tabla en la que se encuentre el objeto referenciado. Sólo existe la condición de que siempre se obtenga una solo fila como resultado. Lo más **cómodo es utilizar esa tabla** DUAL. Aunque se use esa tabla comodín, **el resultado será un objeto** almacenado en la tabla UsuariosObj.
PostgreSQL también es un sistema de gestión de bases de datos objeto-relacional.