

Uso de bases de datos objeto-relacionales.



Caso práctico

Juan ha realizado un curso de perfeccionamiento sobre programación con bases de datos, y ha conocido las ventajas que pueden ofrecer el uso de las bases de datos objeto-relacionales. Hasta ahora siempre había usado las bases de datos relacionales, pero el conocimiento de las bases de datos orientadas a objetos le ha ofrecido nuevas perspectivas para aprovechar de manera más óptima la reutilización de código, el trabajo colaborativo, y la interconexión de las bases de datos con otros lenguajes de programación orientados a objetos.



Materiales formativos de F.P. Online propiedad del Ministerio de Educación, Cultura y Deporte.

[Aviso Legal](#)

1.- Características de las bases de datos objeto-relacionales.




Caso práctico

Ana ha sabido que Juan ha realizado el curso de formación de bases de datos objeto-relacionales, y éste se ha ofrecido a compartir los conocimientos que ha adquirido. Lo primero va a ser es que conozca las características que tienen estos tipos de bases de datos, para que compruebe las diferencias que tienen respecto a las bases de datos relaciones que ha usado hasta ahora. Usarán la base de datos de Oracle que han utilizado anteriormente, pero ahora van a darle el enfoque de este nuevo tipo de bases de datos.



Las **bases de datos objeto-relacionales** que vas a conocer en esta unidad son las referidas a aquellas que han evolucionado desde el modelo relacional tradicional a un modelo híbrido que utiliza además la tecnología orientada a objetos. Las **clases, objetos, y herencia** son directamente soportados en los esquemas de la base de datos y el lenguaje de consulta y manipulación de datos. Además da soporte a una extensión del modelo de datos con la creación personalizada de **tipos de datos y métodos**.

La base de datos de Oracle implementa el modelo orientado a objetos como una **extensión del modelo relacional**, siguiendo soportando la funcionalidad estándar de las bases de datos relacionales.

El modelo objeto-relacional ofrece las **ventajas** de las técnicas orientadas a objetos en cuanto a mejorar la reutilización y el uso intuitivo de los objetos, a la vez que se mantiene la alta capacidad de  **conurrencia** y el rendimiento de las bases de datos relacionales.

Los tipos de objetos que vas a conocer, así como las características orientadas a objeto, te van a proporcionar una mecanismo para **organizar los datos y acceder a ellos a alto nivel**. Por debajo de la capa de objetos, los datos seguirán estando almacenados en columnas y tablas, pero vas a poder trabajar con ellos de manera más parecida a las entidades que puedes encontrar en la vida real, dando así más significado a los datos. En vez de pensar en términos de columnas y tablas cuando realices consultas a la base de datos, simplemente deberás seleccionar entidades que habrás creado, por ejemplo clientes o pedidos.

Podrás utilizar las características orientadas a objetos que vas a aprender en esta unidad a la vez que puedes seguir trabajando con los datos relacionamente, o bien puedes usar de manera más exclusiva las técnicas orientadas a objetos.

En general, el modelo orientado a objetos es **similar al que puedes encontrar en lenguajes** como C++ y Java. La **reutilización** de objetos permite desarrollar aplicaciones de bases de datos más rápidamente y de manera más eficiente. Al ofrecer la base de datos de Oracle soporte nativo para los tipos de objetos, permite a los desarrolladores de aplicaciones con lenguajes orientados a objetos, acceder directamente a las **mismas estructuras de datos** creadas en la base de datos.

La programación orientada a objetos está especialmente enfocada a la construcción de componentes reutilizables y aplicaciones complejas. En **PL/SQL**, la programación orientada a objetos está basada en **tipos de objetos**. Estos tipos te permiten modelar objetos de la vida real, separar los detalles de los interfaces de usuarios de la implementación, y almacenar los datos orientados a objetos de forma permanente en una base de datos. Encontraras especialmente útil los tipos de objetos cuando realizas programas interconectados con Java u otros lenguajes de programación orientados a objetos.

Las tablas de bases de datos relacionales sólo contienen datos. En cambio, los objetos pueden incluir la posibilidad de realizar determinadas **acciones sobre los datos**. Por ejemplo, un objeto "Compra" puede incluir un método para calcular el importe de todos los elementos comprados. O un objeto "Cliente" puede tener métodos que permitan obtener su historial de compras. De esta manera, una aplicación tan sólo debe realizar una llamada a dichos métodos para obtener esa información.



Para saber más

Si deseas conocer más en detalle la definición de las bases de datos objeto-relacionales puedes visitar la web de wikipedia:

[Definición de base de datos objeto-relacional.](#)

2.- Tipos de datos objeto.



Caso práctico

Juan comienza a explicarle a **Ana** uno de los conceptos básicos en las bases de datos orientadas a objetos. Se trata de los tipos de datos objeto. Le hace ver que cualquier objeto que manejamos a diario se puede considerar un posible tipo de objeto para una base de datos.



Un **tipo de dato objeto** es un tipo de dato compuesto definido por el usuario. Representa una **estructura de datos** así como **funciones y procedimientos** para manipular datos. Como ya sabemos, las variables de un determinado tipo de dato escalar (**NUMBER**, **VARCHAR**, **BOOLEAN**, **DATE**, etc.) pueden almacenar un único valor. Las colecciones permiten, como ya veremos, almacenar varios datos en una misma variable siendo todos del mismo tipo. Los tipos de datos objetos nos permitirán almacenar datos de distintos tipos, posibilitando además asociar código a dichos datos.

En el mundo real solemos pensar en un objeto, entidad, persona, animal, etc. como un conjunto de propiedades y acciones. Por ejemplo, el alumnado tiene una serie de propiedades como el nombre, fecha de nacimiento, DNI, curso, grupo, calificaciones, nombre del padre y de la madre, sexo, etc., y tiene asociadas una serie de acciones como matricular, evaluar, asistir a clase, presentar tareas, etc. Los tipos de datos objeto nos permiten representar esos valores y estos comportamientos de la vida real en una aplicación informática.



Reflexiona

Piensa en un objeto y enumera algunas de sus propiedades y acciones que se pueden realizar sobre él.

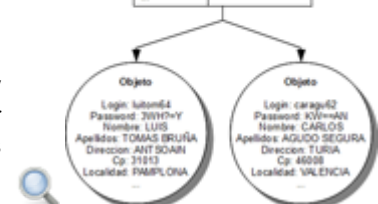
Las variables que formen la estructura de datos de un tipo de dato objeto reciben el nombre de **atributos** (que se corresponde con sus propiedades). Las funciones y procedimientos del tipo de dato objeto se denominan **métodos** (que se corresponde con sus acciones).

Cuando se define un tipo de objeto, se crea una **plantilla abstracta** de un objeto de la vida real. La plantilla especifica los atributos y comportamientos que el objeto necesita en el entorno de la aplicación. Dependiendo de la aplicación a desarrollar se utilizarán sólo determinados atributos y comportamiento del objeto. Por ejemplo, en la gestión de la evaluación del alumnado es muy probable que no se necesite conocer su altura, peso, etc. o utilizar comportamientos como desplazarse, comer, etc., aunque formen todos ellos parte de las características del alumnado.

Aunque los atributos son públicos, es decir, visibles desde otros programas cliente, los programas deberían **manipular los datos únicamente a través de los métodos** (funciones y procedimientos) que se hayan declarado en el tipo objeto, en vez de asignar u obtener sus valores directamente. Esto es debido a que los métodos pueden hacer un chequeo de los datos de manera que se mantenga un estado apropiado en los mismos. Por ejemplo, si se desea asignar un curso a un miembro del alumnado, sólo debe permitirse que se asigne un curso existente. Si se permitiera modificar directamente el curso, se podría asignar un valor incorrecto (curso inexistente).

Durante la ejecución, la aplicación creará **instancias** de un tipo objeto, es decir, referencias a objetos reales con valores asignados en sus atributos. Por ejemplo, una instancia será un determinado miembro del alumnado con sus

Tipo Objeto: Usuarios	
Atributos	Métodos
Login	cambiarDatos
Password	borrarUsuario
Nombre	cambiarCredito
Apellidos	comprobarCredito
Direccion	
Cp	
Localidad	



datos personales correspondientes.

3.- Definición de tipos de objeto.



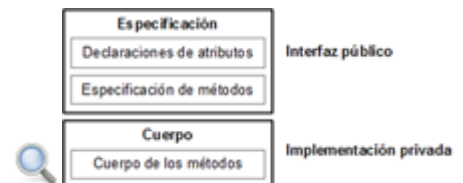
Caso práctico



Ya conoce **Ana** el concepto de las bases de datos orientadas a objetos y que los tipos de datos objeto son la base para ellas. Así que para asentar esos conocimientos se plantea modificar, con la ayuda de **Juan**, la base de datos de la plataforma de juegos on-line que hasta ahora era relacional para convertirla en una base de datos orientada a objetos.

Para ello, se plantea crear el tipo de objeto Usuario, pero necesita conocer cómo se declaran los tipos de datos objeto en Oracle.

La estructura de la definición o declaración de un tipo de objeto está dividida en una especificación y un cuerpo. La **especificación** define el interfaz de programación, donde se declaran los atributos así como las operaciones (métodos) para manipular los datos. En el **cuerpo** se implementa el código fuente de los métodos.



Toda la información que un programa necesita para usar los métodos lo encuentra en la especificación. Se puede modificar el cuerpo sin cambiar la especificación, sin que ello afecte a los programas cliente.

En la especificación de un tipo de objeto, todos los **atributos debes declararlos antes que los métodos**. Si la especificación de un tipo de objeto sólo declara atributos, no es necesario que declares el cuerpo. Debes tener en cuenta también que no puedes declarar atributos en el cuerpo. Además, todas las declaraciones realizadas en la especificación del tipo de objeto son públicas, es decir, visibles fuera del tipo de objeto.

Por tanto, un tipo de objeto contiene (encapsula) datos y operaciones. Puedes declarar atributos y métodos en la especificación, pero no constantes (**CONSTANTS**), excepciones (**EXCEPTIONS**), cursores (**CURSORS**) o tipos (**TYPES**). Al menos debe tener un atributo declarado, y un máximo de 1000. En cambio los métodos son opcionales, por lo que se puede crear un tipo de objeto sin métodos.

Para definir un objeto en Oracle debes utilizar la sentencia **CREATE TYPE** que tiene el siguiente formato:

```
CREATE TYPE nombre_tipo AS OBJECT (  
    Declaración_atributos  
    Declaración_métodos  
);
```

Siendo `nombre_tipo` el nombre deseado para el nuevo tipo de objeto. La forma de declarar los atributos y los métodos se verá en los siguientes apartados de esta unidad didáctica.

En caso de que el **nombre del tipo de objeto ya estuviera siendo usado** para otro tipo de objeto se obtendría un error. Si se desea reemplazar el tipo anteriormente creado, por el nuevo que se va a declarar, se puede añadir la **cláusula OR REPLACE** en la declaración del tipo de objeto:

```
CREATE OR REPLACE TYPE nombre_tipo AS OBJECT
```

Por ejemplo, para crear el tipo de objeto Usuario, **reemplazando la declaración** que tuviera anteriormente se podría hacer algo similar a los siguiente:

```
CREATE OR REPLACE TYPE Usuario AS OBJECT (  
  Declaración_atributos  
  Declaración_métodos  
);
```

Si en algún momento deseas **eliminar el tipo de objeto** que has creado puedes utilizar la sentencia **DROP TYPE**:

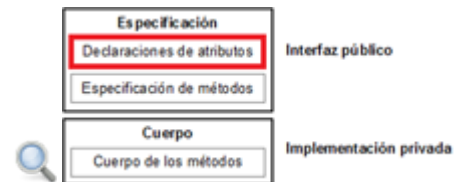
```
DROP TYPE nombre_tipo;
```

Donde nombre_tipo debe ser el nombre del tipo de dato objeto que deseas eliminar. Por ejemplo, para el tipo de objetos anterior, deberías indicar:

```
DROP TYPE Usuario;
```

3.1.- Declaración de atributos.

La declaración de los atributos la puedes realizar de forma muy similar a declaración de las variables, es decir, utilizando un nombre y un tipo de dato. Dicho nombre debe ser único dentro del tipo de objeto, aunque puede ser reutilizado en otros tipos de objeto. El tipo de dato que puede almacenar un determinado atributo puede ser **cualquiera de los tipos de Oracle excepto** los siguientes:



- ✓ LONG y LONG RAW.
- ✓ ROWID y UROWID.
- ✓ Los tipos específicos PL/SQL BINARY_INTEGER (y sus subtipos), BOOLEAN, PLS_INTEGER, RECORD, REF CURSOR, %TYPE, y %ROWTYPE.
- ✓ Los tipos definidos dentro de un paquete PL/SQL.

Debes tener en cuenta que no puedes inicializar los atributos usando el operador de asignación, ni la cláusula **DEFAULT**, ni asignar la restricción **NOT NULL**.

El **tipo de dato de un atributo puede ser otro tipo de objeto**, por lo que la estructura de datos puede ser tan complicada como sea necesario.

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

Después de haber sido creado el tipo de objeto, se pueden **modificar sus atributos** utilizando la sentencia **ALTER TYPE**. Si se desean añadir nuevos atributos se añadirá la cláusula **ADD ATTRIBUTE** seguida de la lista de nuevos atributos con sus correspondientes tipos de dato. Utilizando **MODIFY ATTRIBUTE** se podrán modificar los atributos existentes, y para eliminar atributos se dispone de manera similar de **DROP ATTRIBUTE**.

Aquí tienes varios ejemplos de modificación del tipo de objeto Usuario creado anteriormente:

```
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),  
    MODIFY ATTRIBUTE nombre VARCHAR2(35);
```



Para saber más

En la web de Oracle puedes encontrar la sintaxis completa de la instrucción **CREATE TYPE**. Ahí podrás conocer que hay más posibilidades de uso:

[Sintaxis completa de la instrucción CREATE TYPE.](#) (En inglés)

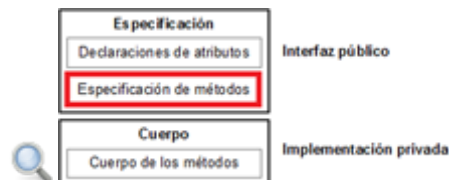
En la web de Oracle puedes encontrar la sintaxis completa de la instrucción **ALTER TYPE**. Ahí podrás conocer que hay más posibilidades de uso:

3.2.- Definición de métodos.

Un **método es un subprograma** que declaras en la especificación de un tipo de objeto usando las palabras clave **MEMBER** o **STATIC**. Debes tener en cuenta que el nombre de un determinado método no puede ser el mismo nombre que el tipo de objeto ni el de ninguno de sus atributos. Como se verá más adelante, se pueden crear métodos con el mismo nombre que el tipo de objeto, pero dichos métodos tendrán una función especial.

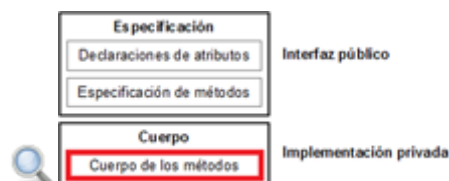
Al igual que los subprogramas empaquetados, los métodos tienen **dos partes: una especificación y un cuerpo**.

En la **especificación** o declaración se debe encontrar el nombre del método, una lista opcional de parámetros, y, en el caso de las funciones, un tipo de dato de retorno. Por ejemplo, observa la especificación del método `incrementoCredito` que se encuentra detrás de las declaraciones de los atributos:



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

En el **cuerpo** debes indicar el código que se debe ejecutar para realizar una determinada tarea cuando el método es invocado. En el siguiente ejemplo se desarrolla el cuerpo del método que se ha declarado antes:



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

Por cada especificación de método que se indique en el bloque de especificación del tipo de objeto, debe existir su correspondiente cuerpo del método, o bien, el método debe declararse como **NOT INSTANTIABLE**, para indicar que el cuerpo del método se encontrará en un subtipo de ese tipo de objeto. Además, debes tener en cuenta que las cabeceras de los métodos deben coincidir exactamente en la especificación y en el cuerpo.

Al igual que los atributos, los parámetros formales se declaran con un nombre y un tipo de dato. Sin embargo, el tipo de dato de un parámetro no puede tener restricciones de tamaño. El tipo de datos puede ser cualquiera de los empleados por Oracle salvo los indicados anteriormente para los atributos. Las mismas restricciones se aplican para los tipos de retorno de las funciones.

El código fuente de los métodos no sólo puede escribirse en el lenguaje PL/SQL. También con otros lenguajes de programación como Java o C.

Puedes usar la sentencia `ALTER TYPE` para **añadir, modificar o eliminar métodos** de un tipo de objeto existente, de manera similar a la utilizada para modificar los atributos de un tipo de objeto.

3.3.- Parámetro SELF.

Un parámetro especial que puedes utilizar con los métodos **MEMBER** es el que recibe el nombre **SELF**. Este parámetro **hace referencia a una instancia (objeto) del mismo tipo de objeto**. Aunque no lo declares explícitamente, este parámetro siempre se declara automáticamente.

El **tipo de dato** correspondiente al parámetro **SELF** será el mismo que el del objeto original. En las funciones **MEMBER**, si no declaras el parámetro **SELF**, su modo por defecto se toma como **IN**. En cambio, en los procedimientos **MEMBER**, si no se declara, se toma como **IN OUT**. Ten en cuenta que no puedes especificar el modo **OUT** para este parámetro **SELF**, y que los métodos **STATIC** no pueden utilizar este parámetro especial.



Si se hace referencia al parámetro **SELF dentro del cuerpo de un método**, realmente se está haciendo referencia al objeto que ha invocado a dicho método. Por tanto, si utilizas **SELF.nombre_atributo** o **SELF.nombre_método**, estarás utilizando un atributo o un método del mismo objeto que ha llamado al método donde se encuentra utilizado el parámetro **SELF**.

```
MEMBER PROCEDURE setNombre(Nombre VARCHAR2) IS
```

```
BEGIN
```

```
    /* El primer elemento (SELF.Nombre) hace referencia al atributo del tipo de c  
    segundo (Nombre) hace referencia al parámetro del método */
```

```
    SELF.Nombre := Nombre;
```

```
END setNombre;
```



Autoevaluación

¿Cómo se denonima a los elementos que realizan determinadas acciones sobre los objetos?

- ☐ Atributos.
- ☐ Métodos.
- ☐ Tipos de datos objeto.
- ☐ Parámetros.

3.4.- Sobrecarga.

Al igual que ocurre con los subprogramas empaquetados, los métodos pueden ser sobrecargados, es decir, puedes **utilizar el mismo nombre para métodos diferentes** siempre que sus parámetros formales sean diferentes (en cantidad o tipo de dato).

Cuando se hace una llamada a un método, se comparan los parámetros actuales con los parámetros formales de los métodos que se han declarado, y se ejecutará aquél en el que haya una coincidencia entre ambos tipos de parámetros.

Ten en cuenta que no es válida la sobrecarga de dos métodos cuyos parámetros formales se diferencian únicamente en su modo, así como tampoco en funciones que se diferencien únicamente en el valor de retorno.

Estos dos ejemplos son correctos, ya que se diferencian en el número de parámetros:

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

No es válido crear un nuevo método en el que se utilicen los mismos tipos de parámetros formales aunque se diferencien los nombres de los parámetros:

```
MEMBER PROCEDURE setNombre(Name VARCHAR2, Surname VARCHAR2)
```



Autoevaluación

¿Son correctas las siguientes declaraciones de métodos?

```
MEMBER FUNCTION getResultado(Valor VARCHAR2)
MEMBER FUNCTION getResultado(Valor INTEGER)
```

- ☐ Verdadero.
- ☐ Falso.



3.5.- Métodos Constructores.

Cada tipo de objeto tiene un método constructor, que se trata de una **función con el mismo nombre que el tipo de objeto** y que se encarga de **inicializar los atributos y retornar una nueva instancia** de ese tipo de objeto.

Oracle crea un **método constructor por defecto** para cada tipo de objeto declarado, cuyos parámetros formales coinciden en orden, nombres y tipos de datos con los atributos del tipo de objeto.

También puedes declarar **tus propios métodos constructores**, reescribiendo ese método declarado por el sistema, o bien, definiendo un nuevo método con otros parámetros. Una de las ventajas de crear un nuevo método constructor personalizado es que se puede hacer una verificación de que los datos que se van a asignar a los atributos son correctos (por ejemplo, que cumplen una determinada restricción).



Si deseas reemplazar el método constructor por defecto, debes utilizar la sentencia **CONSTRUCTOR FUNCTION** seguida del nombre del tipo de objeto en el que se encuentra (recuerda que los métodos constructores tienen el mismo nombre que el tipo de objeto). A continuación debes indicar los parámetros que sean necesarios de la manera habitual. Por último, debes indicar que el valor de retorno de la función es el propio objeto utilizando la cláusula **RETURN SELF AS RESULT**.

Puedes crear **varios métodos constructores** siguiendo las restricciones indicadas para la sobrecarga de métodos.

En el siguiente ejemplo puedes ver la declaración y el cuerpo de un método constructor para el tipo de objeto Usuario. Como puedes comprobar, utiliza dos parámetros: login y crédito inicial. El cuerpo del método realiza un pequeño control para que en caso de que el crédito indicado sea negativo, se deje en cero:

```
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;
```

En la siguiente presentación puedes ver un ejemplo de declaración de un tipo de dato objeto, con un método constructor y sobrecarga de métodos.

[Resumen textual alternativo](#)

4.- Utilización de objetos.



Caso práctico

Ada ha observado que **Juan** y **Ana** están muy volcados en el desarrollo de algo. Ellos le cuentan que están realizando pruebas para modificar sus bases de datos relacionales para darles funcionalidades orientadas a objetos.

De momento han realizado la declaración de los tipos de datos objeto, pero no pueden enseñarle a **Ada** su funcionamiento práctico, puesto que todavía no han creado objetos de esos tipos que ya tienen creados.

Le piden un poco de paciencia, porque pronto podrán enseñarle los beneficios que puede reportar esta técnica de trabajo con bases de datos y para el desarrollo de aplicaciones.



Una vez que dispones de los conocimientos necesarios para tener declarado un tipo de dato objeto, vamos a conocer la forma de utilizar los objetos que vayas a crear de ese tipo.

En los siguientes apartados vas a ver cómo puedes **declarar variables** que permitan almacenar objetos, darles un **valores iniciales a sus atributos**, acceder al **contenido** de dichos atributos en cualquier momento, y **llamar a los métodos** que ofrece el tipo de objeto utilizado.

4.1.- Declaración de objetos.

Una vez que el tipo de objeto ha sido definido, éste puede ser utilizado para **declarar variables de objetos** de ese tipo en cualquier bloque PL/SQL, subprograma o paquete. Ese tipo de objeto lo puedes utilizar como tipo de dato para una variable, atributo, elemento de una tabla, parámetro formal, o resultado de una función, de igual manera que se utilizan los tipos de datos habituales como **VARCHAR** o **NUMBER**.

Por ejemplo, para declarar una variable denominada u1, que va a permitir almacenar un objeto del tipo Usuario, debes hacer la siguiente declaración:

```
u1 Usuario;
```



En la declaración de cualquier **procedimiento o función**, incluidos los métodos del mismo tipo de objeto o de otro, se puede utilizar el tipo de dato objeto definido para indicar que debe **pasarse como parámetro un objeto** de dicho tipo en la llamada. Por ejemplo pensemos en un procedimiento al que se le debe pasar como parámetro un objeto del tipo Usuario:

```
PROCEDURE setUsuario(u IN Usuario)
```

La llamada a este método se realizaría utilizando como parámetro un objeto, como el que podemos tener en la variable declarada anteriormente:

```
setUsuario(u1);
```

De manera semejante una función puede **retornar objetos**:

```
FUNCTION getUsuario(codigo INTEGER) RETURN Usuario
```

Los objetos se crean durante la ejecución del código como instancias del tipo de objeto, y cada uno de ellos pueden contener valores diferentes en sus atributos.

El **ámbito de los objetos** sigue las mismas reglas habituales en PL/SQL, es decir, en un bloque o subprograma los objetos son creados (instanciados) cuando se entra en dicho bloque o subprograma y se destruyen automáticamente cuando se sale de ellos. En un paquete, los objetos son instanciados en el momento de hacer referencia al paquete y dejan de existir cuando se finaliza la sesión en la base de datos.




Autoevaluación

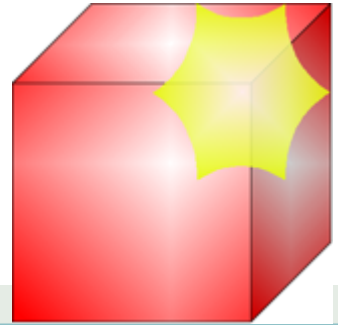
Suponiendo que tienes declarado el tipo de objeto Factura. ¿Cuál de las siguientes declaraciones de variable para guardar un objeto de ese tipo es correcta?

- ☐ Factura factura1;
- ☐ factura1 := Factura;
- ☐ factura1 Factura;

4.2.- Inicialización de objetos.

Para **crear o instanciar un objeto** de un determinado tipo de objeto, debes hacer una **llamada a su método constructor**. Esto lo puedes realizar empleando la  **instrucción NEW** seguido del nombre del tipo de objeto como una llamada a una función en la que se indican como parámetros los valores que se desean asignar a los atributos inicialmente. En una asignación también puedes optar por hacer eso mismo omitiendo la palabra NEW.

El **orden de los parámetros** debe coincidir con el orden en el que están declarados los atributos, así como los tipos de datos. El formato sería como el siguiente:



```
variable_objeto := NEW Nombre_Tipo_Objeto (valor_atributo1, valor_atributo2, ...);
```

Por ejemplo, en el caso del tipo de objeto Usuario:

```
u1 := NEW Usuario('luitom64', 'LUIS ', 'TOMAS BRUÑA', '24/10/07', 100);
```

En ese momento se **inicializa el objeto**. Hasta que no se inicializa el objeto llamando a su constructor, el objeto tiene el valor NULL.

Es habitual inicializar los objetos en su declaración.

```
u1 Usuario := NEW Usuario('luitom64', 'LUIS ', 'TOMAS BRUÑA', '24/10/07', 100);
```

La llamada al método constructor se puede realizar en cualquier lugar en el que se puede hacer una llamada a una función de la forma habitual. Por ejemplo, la llamada al método constructor puede ser utilizada **como parte de una expresión**.

Los **valores de los parámetros** que se pasan al constructor cuando se hace la llamada, son asignados a los atributos del objeto que está siendo creado. Si la llamada es al método constructor que incorpora Oracle por defecto, debes indicar un parámetro para cada atributo, en el mismo orden en que están declarados los atributos. Ten en cuenta que los atributos, en contra de lo que ocurre con variables y constantes, no pueden tener valores por defecto asignados en su declaración. Por tanto, los valores que se desee que tengan inicialmente los atributos de un objeto instanciado deben indicarse como parámetros en la llamada al método constructor.

Existe la posibilidad de **utilizar los nombres de los parámetros formales** en la llamada al método constructor, en lugar de utilizar el modelo posicional de los parámetros. De esta manera no es obligatorio respetar el orden en el que se encuentran los parámetros reales respecto a los parámetros formales, que como se ha comentado antes coincide con el orden de los atributos.

```
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;
/
```



Autoevaluación

¿Cuál de las siguientes inicializaciones de objetos es correcta para el tipo de objeto **Factura**, suponiendo que dispone de los atributos **número (INTEGER)**, **nombre (VARCHAR2)** e **importe (NUMBER)**?

- ☐ factura1 := NEW Factura(3, 'Juan Álvarez', 30.50);
- ☐ factura1 = Factura(3, 'Juan Álvarez', 30.50);
- ☐ factura1 := NEW Factura('Juan Álvarez', 3, 30.50);
- ☐ factura1 := NEW (3, 'Juan Álvarez', 30.50);

4.3.- Acceso a los atributos de objetos.

Para hacer referencia a un atributo de un objeto debes utilizar el **nombre de dicho atributo**, utilizando el **punto** para acceder al valor que contiene o bien para modificarlo. Antes debe ir **precedido del objeto** cuyo atributo deseas conocer o modificar.

```
nombre_objeto.nombre_atributo
```



Por ejemplo, la consulta del valor de un atributo puede utilizarse como parte de una asignación o como parámetro en la llamada a una función:


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

La **modificación del valor** contenido en el atributo puede ser similar a la siguiente:

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

Los nombres de los atributos pueden ser encadenados, lo que permite el acceso a atributos de tipos de objetos anidados. Por ejemplo, si el objeto sitio1 tiene un atributo del tipo de objeto Usuario, se accedería al atributo del nombre del usuario con:

```
sitio1.usuario1.nombre
```

Si se utiliza en una expresión el acceso a un atributo de un objeto que **no ha sido inicializado**, se evalúa como NULL. Por otro lado, si se intenta asignar valores a un objeto no inicializado, éste lanza una  **excepción ACCESS_INTO_NULL**.

Para comprobar si un objeto es NULL se puede utilizar el operador de comparación **IS NULL** con el que se obtiene el valor **TRUE** si es así.

De manera similar, al intentar hacer una llamada a un método de un objeto que no ha sido inicializado, se lanza una excepción **NULL_SELF_DISPATCH**. Si se pasa como parámetro de tipo **IN**, los atributos del objeto NULL se evalúan como NULL, y si el parámetro es de tipo **OUT** o **IN OUT** lanza una excepción al intentar modificar el valor de sus atributos.



Autoevaluación

¿Cuál de las siguientes expresiones es correcta para asignar el valor 50 al atributo importe del objeto factura1?

- ☐ factura1.importe := 50;
- ☐ importe.factura1 := 50;
- ☐ 50 := factura1.importe;



4.4.- Llamada a los métodos de los objetos.

Al igual que las llamadas a subprogramas, puedes invocar a los métodos de un tipo de objetos **utilizando un punto entre el nombre del objeto y el del método**. Los parámetros reales que se pasen al método se indicarán separados por comas, entre paréntesis, después del nombre del método.

```
usuario1.setNombreCompleto('Juan', 'García Fernández');
```

Si el método **no tiene parámetros**, se indicará la lista de parámetros reales vacía (sólo con los paréntesis), aunque se pueden omitir los paréntesis.

```
credito := usuario1.getCredito();
```

Las llamadas a métodos pueden **encadenarse**, en cuyo caso, el orden de ejecución de los métodos es de derecha a izquierda. Se debe tener en cuenta que el método de la izquierda **debe retornar un objeto** del tipo correspondiente al método de la derecha.

Por ejemplo, si dispones de un objeto sitio1 que tiene declarado un método `getUsuario` el cual retorna un objeto del tipo Usuario, puedes realizar con ese valor retornado una llamada a un método del tipo de objeto Usuario:

```
sitio1.getUsuario.setNombreCompleto('Juan', 'García Fernández');
```

Los **métodos MEMBER** son invocados utilizando una instancia del tipo de objeto:

```
nombre_objeto.metodo()
```

En cambio, los **métodos STATIC** se invocan usando el tipo de objeto, en lugar de una de sus instancias:

```
nombre_tipo_objeto.metodo()
```



Autoevaluación

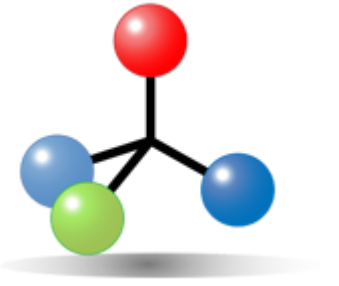
¿Cuál de las siguientes llamadas al método `getImporte` es correcto para el objeto `factura1`?

- ☐ valor := getImporte.factura1();
- ☐ valor := factura1.getImporte();
- ☐ valor := getImporte().factura1;

4.5.- Herencia.

El lenguaje PL/SQL admite la herencia simple de tipos de objetos, mediante la cual, puedes **definir subtipos** de los tipos de objeto. Estos subtipos, o tipos heredados, **contienen todos los atributos y métodos del tipo padre**, pero además pueden contener **atributos y métodos adicionales**, o incluso sobrescribir métodos del tipo padre.

Para indicar que un tipo de objeto es **heredado** de otro hay que usar la **palabra reservada UNDER**, y además hay que tener en cuenta que el tipo de objeto del que **hereda** debe tener la **propiedad NOT FINAL**. Por defecto, los tipos de objeto se declaran como **FINAL**, es decir, que no se puede crear un tipo de objeto que herede de él.



Si no se indica lo contrario, siempre se pueden crear objetos (instancias) de los tipos de objeto declarados. Indicando la opción **NOT INSTANTIABLE** puedes declarar tipos de objeto de los que **no se pueden crear objetos**. Estos tipos tendrán la función de ser padres de otros tipos de objeto.

En el siguiente ejemplo puedes ver cómo se crea el tipo de objeto Persona, que se utilizará heredado en el tipo de objeto UsuarioPersona. De esta manera, este último tendrá los atributos de Persona más los atributos declarados en UsuarioPersona. En la creación del objeto puedes observar que se deben asignar los 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;  
/
```



Autoevaluación

Un tipo de objeto que se ha declarado como hijo de otro, ¿hereda los atributos y métodos del tipo de objeto padre?

- ☐ Verdadero.
- ☐ Falso.

5.- Métodos MAP y ORDER.



Caso práctico



Juan se ha dado cuenta de que va a tener un problema cuando necesite realizar comparaciones y órdenes entre objetos del mismo tipo.

Cuando tenga una serie de objetos de tipo Usuario, y necesite realizar una operación de ordenación entre ellos, ¿cómo debe hacerlo? Podría indicar que se realizaran las consultas de forma ordenada en función de alguno de los atributos del tipo de objeto. Ha estudiado el funcionamiento de los métodos **MAP** y **ORDER**, y ha comprobado que son los que le van a ofrecer un mecanismo sencillo para ello.

Las instancias de un tipo de objeto no tienen un orden predefinido. Si deseas establecer un orden en ellos, con el fin de **hacer una ordenación o una comparación, debes crear un método MAP**.

Por ejemplo, si haces una comparación entre dos objetos Usuario, y deseas saber si uno es mayor que otro, ¿en base a qué criterio se hace esa comparación? ¿Por el orden alfabético de apellidos y nombre? ¿Por la fecha de alta? ¿Por el crédito? Hay que establecer con un método **MAP** cuál va a ser el valor que se va a utilizar en las comparaciones.



Para crear un método **MAP** debes declarar un método que **retorne el valor que se va a utilizar para hacer las comparaciones**. El método que declares para ello debe empezar su declaración con la palabra **MAP**:

```
CREATE OR REPLACE TYPE Usuario AS OBJECT (  
    login VARCHAR2(30),  
    nombre VARCHAR2(30),  
    apellidos VARCHAR2(40),  
    f_ingreso DATE,  
    credito NUMBER,  
    MAP MEMBER FUNCTION ordenarUsuario RETURN VARCHAR2  
);  
/
```

En el cuerpo del método se debe retornar el valor que se utilizará para realizar las comparaciones entre las instancias del tipo de objeto. Por ejemplo, si se quiere establecer que las comparaciones entre objetos del tipo Usuario se realice considerando el orden alfabético habitual de apellidos y nombre:

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

El lenguaje PL/SQL utiliza los métodos **MAP** para evaluar expresiones lógicas que resultan valores booleanos como **objeto1 > objeto2**, y para realizar las comparaciones implícitas en las cláusulas **DISTINCT**, **GROUP BY** y **ORDER BY**.

A continuación puedes encontrar el código completo de un ejemplo de declaración y uso de un método MAP para establecer el orden en las comparaciones entre dos instancias de objetos del tipo Usuario. Como puedes comprobar, las comparaciones se realizan un función del orden alfabético de los apellidos seguidos del nombre.

[Resumen textual alternativo](#)

Cada tipo de objeto **sólo puede tener un método MAP declarado**, y sólo puede retornar alguno de los siguientes tipos: **DATE**, **NUMBER**, **VARCHAR2**, **CHARACTER** o **REAL**.

5.1.- Métodos ORDER.

De forma **similar al método MAP**, puedes declarar en cualquier tipo de objeto un método **ORDER** que te permitirá **establecer un orden** entre los objetos instanciados de dicho tipo.

Cada tipo de objeto **sólo puede contener un método ORDER**, el cual debe **retornar un valor numérico que permita establecer el orden entre los objetos**. Si deseas que un objeto sea menor que otro puedes retornar, por ejemplo, el valor -1. Si vas a determinar que sean iguales, devuelve 0, y si va a ser mayor, retorna 1. De esta manera, considerando el valor retornado por el método **ORDER**, se puede establecer si un objeto es menor, igual o mayor que otro en el momento de hacer una ordenación entre una serie de objetos del mismo tipo.



Para declarar qué método va a realizar esta operación, debes indicar la palabra **ORDER** delante de su declaración. Debes tener en cuenta que va a **retornar un valor numérico (INTEGER)**, y que necesita que se indique un **parámetro que será del mismo tipo de objeto**. El objeto que se indique en ese parámetro será el que se compare con el objeto que utilice este método.

En el siguiente ejemplo, el sistema que se va a establecer para ordenar a los usuarios se realiza en función de los dígitos que se encuentran a partir de la posición 7 del atributo login.

```
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 e
        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;
```

Con ese ejemplo, el usuario con login 'luitom64' se considera mayor que el usuario 'caragu62', ya que '64' es mayor que '62'.

El método debe retornar un número negativo, cero, o un número positivo que significará que el objeto que utiliza el método (**SELF**) es menor, igual, o mayor que el objeto pasado por parámetro.

Debes tener en cuenta que puedes declarar **un método MAP o un método ORDER, pero no los dos**.

Cuando se vaya a ordenar o mezclar un alto número de objetos, es preferible usar un método **MAP**, ya que en esos casos un método **ORDER** es menos eficiente.

En el siguiente enlace puedes encontrar el código completo de un ejemplo de declaración y uso de un método **ORDER** para establecer el orden entre objetos del tipo Usuario. Como puedes comprobar, las comparaciones se realizan un función de los dos últimos dígitos del atributo login.



Autoevaluación

¿Los métodos MAP sólo sirven para evaluar expresiones lógicas que resultan valores booleanos?

- ☐ Verdadero.
- ☐ Falso.

6.- Tipos de datos colección.



Caso práctico

Ana ha estudiado los vectores y matrices que se usan en varios lenguajes de programación. Le pregunta a Juan si con las bases de datos objeto-relacionales se puede utilizar algo parecido para almacenar los objetos instanciados. Él le dice que para eso existen las colecciones.



Para que puedas almacenar en memoria un conjunto de datos de un determinado tipo, la base de datos de Oracle te ofrece los tipos de datos **colección**.

Una **colección** es un conjunto de elementos del mismo tipo. Puede compararse con los **vectores y matrices** que se utilizan en muchos lenguajes de programación. En este caso, las colecciones sólo pueden tener **una dimensión** y los elementos se indexan mediante un valor de tipo numérico o cadena de caracteres.

La base de datos de Oracle proporciona los tipos **VARRAY** y **NESTED TABLE** (tabla anidada) como tipos de datos colección.

- ✓ Un **VARRAY** es una colección de elementos a la que se le establece una dimensión máxima que debe indicarse al declararla. Al tener una longitud fija, la eliminación de elementos no ahorra espacio en la memoria del ordenador.
- ✓ Una **NESTED TABLE** (**tabla anidada**) puede almacenar cualquier número de elementos. Tienen, por tanto, un tamaño dinámico, y no tienen que existir forzosamente valores para todas las posiciones de la colección.
- ✓ Una variación de las tablas anidadas son los **arrays asociativos**, que utilizan valores arbitrarios para sus índices. En este caso, los índices no tienen que ser necesariamente consecutivos.

Cuando necesites almacenar un número fijo de elementos, o hacer un recorrido entre los elementos de forma ordenada, o si necesitas obtener y manipular toda la colección como un valor, deberías utilizar el tipo **VARRAY**.

En cambio, si necesitas ejecutar consultas sobre una colección de manera eficiente, o manipular un número arbitrario de elementos, o bien realizar operaciones de inserción, actualización o borrado de forma masiva, deberías usar una **NESTED TABLE**.

Las colecciones pueden ser declaradas como una instrucción SQL o en el bloque de declaraciones de un programa PL/SQL. El tipo de dato de los elementos que puede contener una colección declarada en PL/SQL es cualquier tipo de dato PL/SQL, excepto **REF CURSOR**. Los elementos de las colecciones declaradas en SQL, además **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 objetos declarado previamente puede ser utilizado como tipo de elemento para una colección.

Una tabla de la base de datos puede contener **columnas que sean colecciones**. Sobre una tabla que contiene colecciones se podrán realizar operaciones de consulta y manipulación de datos de la misma manera que se realiza con tablas con los tipos de datos habituales.



Para saber más

En este documento puedes encontrar más información sobre los tipos de datos colección entre otros conceptos de las bases de datos objetos-relacionales:



Autoevaluación

¿Qué tipo de colección tiene un tamaño fijo?

- ☐ VARRAY.
- ☐ NESTED TABLE.
- ☐ Array Asociativo.



6.1.- Declaración y uso de colecciones.

La **declaración** de estos tipos de 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;
```



Donde `nombre_tipo` representa el nombre de la colección, `tamaño_max` es el número máximo de elementos que podrá contener la colección, y `tipo_elemento` es el tipo de dato de los elementos que forman la colección. El tipo de elemento utilizado puede ser también cualquier tipo de objetos declarado previamente.

En caso de que la declaración se haga en SQL, fuera de un subprograma PL/SQL, se debe declarar con el formato **CREATE [OR REPLACE] TYPE**:

```
CREATE TYPE nombre_tipo IS ...
```

El `tipo_índice` representa el tipo de dato que se va a utilizar para el índice. Puede ser **PLS_INTEGER**, **BINARY_INTEGER** o **VARCHAR2**. En este último tipo se debe indicar entre paréntesis el tamaño que se va a utilizar para el índice, por ejemplo, **VARCHAR2(5)**.

Hasta que no sea inicializada una colección, ésta es **NULL**. Para **inicializar** una colección debes utilizar el constructor, que es una función con el mismo nombre que la colección. A esta función se le pasa como parámetros los valores iniciales de la colección. Por ejemplo:

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

La inicialización se puede realizar en el bloque de código del programa, o bien, directamente en el bloque de declaraciones como puedes ver en este ejemplo:

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

Para **obtener uno de los elementos** de la colección o modificar su contenido debes indicar el nombre de la colección seguido, entre paréntesis, del índice que ocupa el elemento deseado. Tanto en los **VARRAY** como en **NESTED TABLE**, el primer elemento tiene el índice 1.

Por ejemplo, para mostrar en pantalla el segundo elemento ('Naranja') de la colección `Colores`:

```
dbms_output.put_line(misColores(2));
```

En el siguiente ejemplo se modifica el contenido de la posición 3:

```
misColores(3) := 'Gris';
```

En el siguiente ejemplo puedes comprobar cómo pueden utilizarse las colecciones para almacenar sus datos en una tabla de la base de datos, así como la utilización con sentencias de consulta y manipulación de los datos de la colección que se encuentra en la tabla.

```
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;/
```

7.- Tablas de objetos.



Caso práctico



Aunque **Ana** ya ha aprendido a almacenar objetos en memoria gracias a las colecciones de objetos, necesita almacenarlos de manera persistente en una base de datos. **Juan** va a enseñarle que puede realizarlo de manera muy parecida a la gestión de tablas que ya conoce, en la que se usan los tipos de datos habituales de las bases de datos.

Después de haber visto que un grupo de objetos se puede almacenar en memoria mediante colecciones, vas a ver en este apartado que también se pueden **almacenar los objetos en tablas** de igual manera que los tipos de datos habituales de las bases de datos.

Los tipos de datos objetos se pueden usar para formar una tabla exclusivamente formado por elementos de ese tipo, o bien, para usarse como un tipo de columna más entre otras columnas de otros tipos de datos.

En caso de que desees crear una **tabla formada exclusivamente por un determinado tipo de dato objeto**, (tabla de objetos) debes utilizar la sentencia **CREATE TABLE** junto con el tipo de objeto de la siguiente manera:



```
CREATE TABLE NombreTabla OF TipoObjeto;
```

Siendo NombreTabla el nombre que deseas dar a la tabla que va a almacenar los objetos del tipo TipoObjeto. Por ejemplo, para crear la tabla UsuariosObj, que almacene objetos del tipo Usuario:

```
CREATE TABLE UsuariosObj OF Usuario;
```

Debes tener en cuenta que si una tabla hace uso de un tipo de objeto, **no podrás eliminar ni modificar la estructura de dicho tipo de objeto**. Por tanto, desde el momento en que el tipo de objeto sea utilizado en una tabla, no podrás volver a definirlo.

Para poder crear este ejemplo previamente debes tener declarado el tipo de objeto Usuario, que se ha utilizado en apartados anteriores.

Al crear una tabla de esta manera, estamos consiguiendo que podamos almacenar objetos del tipo Usuario en una tabla de la base de datos, quedando así sus datos **persistentes** mientras no sean eliminados de la tabla. Anteriormente hemos instanciado objetos que se han guardado en variables, por lo que al terminar la ejecución, los objetos, y la información que contienen, desaparecen. Si esos objetos se almacenan en tablas no desaparecen hasta que se eliminen de la tabla en la que se encuentren.

Cuando se instancia un objeto con el fin de almacenarlo en una tabla, dicho objeto no tiene identidad fuera de la tabla de la base de datos. Sin embargo, el tipo de objeto existe independientemente de cualquier tabla, y puede ser usado para crear objetos en cualquier modo.

Las tablas que sólo contienen filas con objetos, reciben el nombre de **tablas de objetos**.

En la siguiente imagen se muestra el contenido de la tabla que incluye dos filas de objetos de tipo Usuario. Observa que los atributos del tipo de objeto se muestran como si fueran las columnas de la tabla:



```
SQL> select * from usuarios34;
```

LOGIN	NOMBRE	APellidos	F_INGRES	CREDITO
belind4	LILIE	SOMME BEHME	24/10/87	58
caraga72	CARLOS	RODRO GEDINA	06/07/87	100



Para saber más

En este documento puedes encontrar información general sobre las bases de datos objeto-relacionales, con algunos ejemplos de creación de tablas de objetos:



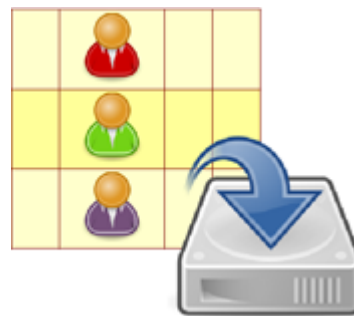
[Bases de datos orientadas a objetos.](#) (0.19 MB)

7.1.- Tablas con columnas tipo objeto.

Puedes usar cualquier tipo de objeto, que hayas declarado previamente, para utilizarlo como un **tipo de dato de una columna más en una tabla** de la base de datos. Así, una vez creada la tabla, puedes utilizar cualquiera de las sentencias SQL para insertar un objeto, seleccionar sus atributos y actualizar sus datos.

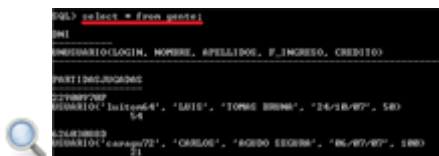
Para crear una tabla en el que alguno de sus columnas sea un tipo de objeto, simplemente debes hacerlo como si fuera una columna como las que has utilizado hasta ahora, pero en el tipo de dato debes especificar el tipo de objeto.

Por ejemplo, podemos crear una tabla que contenga, entre otras columnas, una columna de objetos del tipo Usuario que hemos utilizado anteriormente.



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

Como puedes comprobar en la siguiente imagen, los datos del campo unUsuario se muestran como integrantes de cada objeto Usuario, a diferencia de la tabla de objetos que has visto en el apartado anterior. Ahora todos los atributos del tipo de objeto Usuario no se muestran como si fueran varias columnas de la tabla, sino que forman parte de una única columna.



Autoevaluación

En las tablas con columnas de tipo objeto, ¿los atributos se muestran como columnas de la tabla?

- ☐ Falso.
- ☐ Verdadero.

7.2.- Uso de la sentencia Select.


De manera similar a las consultas que has realizado sobre tablas sin tipos de objetos, **puedes utilizar la sentencia SELECT** para obtener datos de las filas almacenadas en tablas de objetos o tablas con columnas de tipos de objetos.

El uso más sencillo sería para mostrar todas las filas contenidas en la tabla:

```
SELECT * FROM NombreTabla;
```

Como puedes apreciar en la imagen, la tabla que forme parte de la consulta puede ser una tabla de objetos (como la tabla UsuariosObj), o una tabla que contiene columnas de tipos de objetos (como la tabla Gente).

En las sentencias **SELECT** que utilices con objetos, puedes incluir cualquiera de las **cláusulas y funciones de agrupamiento** que has aprendido para la sentencia **SELECT** que has usado anteriormente con las tablas que contienen columnas de tipos básicos. Por ejemplo, puedes utilizar: **SUM**, **MAX**, **WHERE**, **ORDER**, **JOIN**, etc.

Es habitual utilizar  **alias** para hacer referencia al nombre de la tabla. Observa, por ejemplo, la siguiente consulta, en la que se desea obtener el nombre y los apellidos de los usuarios que tienen algo de crédito:

```
SELECT u.nombre, u.apellidos FROM UsuariosObj u WHERE u.credito > 0
```

Si se trata de una **tabla con columnas de tipo objeto**, el acceso a los atributos del objeto se debe realizar indicando previamente el nombre asignado a la columna que contiene los objetos:

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

LOGIN	NOMBRE	APELLIDOS	F_INGRESO	CREDITO
usuario4	LEIS	TORRE BONA	24-10-97	10
usuario72	CARLOS	AGUDO SEGURA	04-07-97	100



Para saber más

En este documento puedes encontrar información general sobre las bases de datos objeto-relacionales, con algunos ejemplos de uso de la sentencia **SELECT** con tablas de objetos:



[Tipos de Objeto en PL/SQL](#). (1.28 MB)

7.3.- Inserción de objetos.

Evidentemente, no te servirá de nada una tabla que pueda contener objetos sino conocemos la manera de insertar objetos en la tabla.

La manera que tienes para ello **es la misma** que has utilizado para introducir datos de cualquier tipo habitual en las tablas de la base de datos: usando la sentencia **INSERT** de SQL.

En las tablas habituales, cuando querías añadir una fila a una tabla que tenía un campo **VARCHAR** le suministrabas a la sentencia **INSERT** un dato de ese tipo. Pues si la tabla es de un determinado tipo de objetos, o si posee un campo de un determinado tipo de objeto, tendrás que suministrar a la sentencia **INSERT** un objeto instanciado de su tipo de objeto correspondiente.



Por tanto, si queremos insertar un Usuario en la tabla Gente que hemos creado en el apartado anterior, previamente debemos crear el objeto o los objetos que se deseen insertar. A continuación podremos utilizarlos dentro de la sentencia **INSERT** como si fueran valores simplemente.

```
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;
```

De una manera más directa, puedes crear el objeto dentro de la sentencia **INSERT** directamente, sin necesidad de guardar el objeto previamente en una variable:

```
INSERT INTO UsuariosObj VALUES (Usuario('luitom64', 'LUIS', 'TOMAS BRUNA', '24/10/2007', 50))
```

Podrás comprobar los resultados haciendo una consulta **SELECT** sobre la tabla de la manera habitual:

```
SELECT * FROM UsuariosObj;
```

De manera similar puedes realizar la **inserción de filas en tablas con columnas de tipo objeto**. Por ejemplo, para la tabla Gente que posee entre sus columnas, una de tipo objeto Usuario, podríamos usar el formato de instanciar el objeto directamente en la sentencia **INSERT**, o bien, indicar una variable que almacena un objeto que se ha instanciado anteriormente:

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

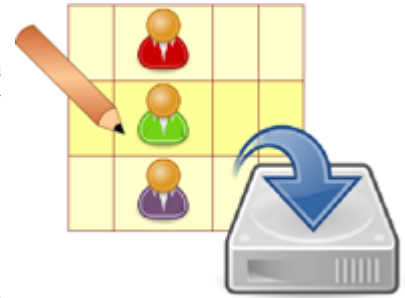
```
INSERT INTO Gente VALUES ('62603088D', u2, 21);
```

7.4.- Modificación de objetos.

Si deseas modificar un objeto almacenado en una tabla tan sólo tienes que utilizar las **mismas sentencias SQL** que disponías para modificar registros de una tabla. ¿Recuerdas la sentencia **UPDATE**? Ahora puedes volver a utilizarla para modificar también los objetos de la tabla, de igual manera que cualquier otro tipo de dato.

Hay una pequeña diferencia en la forma de especificar los nombre de los campos afectados, en función del tipo de tabla: según sea una tabla de objetos, o bien una tabla con alguna columna de tipo objeto.

Si se trata de una tabla de objetos, se hará referencia a los atributos de los objetos justo detrás del nombre asignado a la tabla. Sería algo similar al formato siguiente:



```
UPDATE NombreTabla
    SET NombreTabla.atributoModificado = nuevoValor
    WHERE NombreTabla.atributoBusqueda = valorBusqueda;
```

Continuando con el ejemplo empleado anteriormente, vamos a suponer que deseas modificar los datos de un determinado usuario. Por ejemplo, modifiquemos el crédito del usuario identificado por el login 'luitom64', asignándole el valor 0.

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

Es muy habitual abreviar el nombre de la tabla con un **alias**:

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

Pero no sólo puedes cambiar el valor de un determinado atributo del objeto. Puedes **cambiar un objeto por otro** como puedes ver en el siguiente ejemplo, en el que se sustituye el usuario con login 'caragu72' por otro usuario nuevo.

```
UPDATE UsuariosObj u SET u = Usuario('juaesc82', 'JUAN', 'ESCUDERO LARRASA', '10/04/2011', 0)
```

Si se trata de una tabla con columnas de tipo objeto, se debe hacer referencia al nombre de la columna que contiene los objetos:

```
UPDATE NombreTabla
    SET NombreTabla.colObjeto.atributoModificado = nuevoValor
    WHERE NombreTabla.colObjeto.atributoBusqueda = valorBusqueda;
```

A continuación puedes ver un ejemplo de actualización de datos de la tabla que se había creado con una columna del tipo de objeto Usuario. Recuerda que a la columna en la que se almacenaban los objetos de tipo Usuario se le había asignado el nombre unUsuario:

```
UPDATE Gente g
```

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

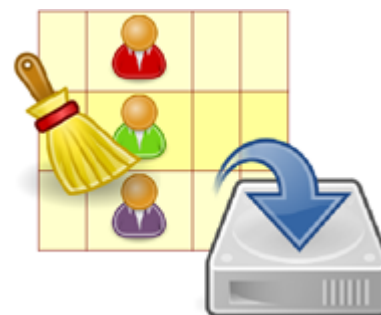
O bien, puedes cambiar todo un objeto por otro, manteniendo el resto de los datos de la fila sin modificar, como en el siguiente ejemplo, donde los datos de DNI y partidasJugadas no se cambia, sólo se cambia un usuario por otro.

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

7.5.- Borrado de objetos.

Por supuesto, no nos puede faltar una sentencia que nos permita eliminar determinados objetos almacenados en tablas. Al igual que has podido comprobar en las operaciones anteriores, tienes a tu disposición la misma sentencia que has podido utilizar en las operaciones habituales sobre tablas. En este caso de borrado de objetos deberás utilizar la sentencia **DELETE**.

El modo de uso de **DELETE** sobre objetos almacenados en tablas es muy similar al utilizado hasta ahora:



```
DELETE FROM NombreTablaObjetos;
```

Recuerda que si no se indica ninguna condición, se **eliminarán todos** los objetos de la tabla, por lo que suele ser habitual utilizar la sentencia **DELETE** con una condición detrás de la cláusula **WHERE**. Los objetos o filas de la tabla que **cumplan con la condición** indicada serán los que se eliminen.

```
DELETE FROM NombreTablaObjetos WHERE condición;
```

Observa el siguiente ejemplo en el que se borrarán de la tabla UsuariosObj, que es una tabla de objetos, los usuarios cuyo crédito sea 0. Observa que se utiliza un alias para el nombre de la tabla:

```
DELETE FROM UsuariosObj u WHERE u.credito = 0;
```

De manera similar se puede realizar el borrado de filas en tablas en las que alguna de sus columnas son objetos. Puedes comprobarlo con el siguiente ejemplo, donde se utiliza la tabla Gente, en la que una de sus columnas (unUsuario) es del tipo de objeto Usuario que hemos utilizado en otros apartados anteriores.

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

Esta sentencia, al igual que las anteriores, se puede **combinar con otras consultas SELECT**, de manera que en vez de realizar el borrado sobre una determinada tabla, se haga sobre el resultado de una consulta, o bien que la condición que determina las filas que deben ser eliminadas sea también el resultado de una consulta. Es decir, todo lo aprendido sobre las operaciones de manipulación de datos sobre las tablas habituales, se puede aplicar sobre tablas de tipos de objetos, o tablas con columnas de tipos de objetos.

7.6.- Consultas con la función VALUE.

Cuando tengas la necesidad de **hacer referencia a un objeto en lugar de alguno de sus atributos**, puedes utilizar la función **VALUE** junto con el nombre de la tabla de objetos o su alias, **dentro de una sentencia SELECT**. Puedes ver a continuación un ejemplo de uso de dicha función para hacer inserciones en otra tabla (Favoritos) del mismo tipo de objetos:



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

Esa misma función **VALUE** puedes utilizarla para hacer comparaciones de igualdad entre objetos, por ejemplo, si deseamos obtener datos de los usuarios que se encuentren en las tablas Favoritos y UsuariosObj.

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

Observa la diferencia en el uso cuando se hace la comparación con una columna de tipo de objetos. En ese caso la referencia que se hace a la columna (g.unUsuario) permite obtener directamente un objeto, sin necesidad de utilizar la función **VALUE**.

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

Usando la **cláusula INTO** podrás **guardar en variables el objeto obtenido** en las consultas usando la función **VALUE**. Una vez que tengas asignado el objeto a la variable podrás hacer uso de ella de cualquiera de las formas que has visto anteriormente en la manipulación de objetos. Por ejemplo, puedes acceder a sus atributos, formar parte de asignaciones, etc.

En el siguiente ejemplo se realiza una consulta de la tabla UsuariosObj para obtener un determinado objeto de tipo Usuario. El objeto resultante de la consulta se guarda en la variable u1. Esa variable se utiliza para mostrar en pantalla el nombre del usuario, y para ser asignada a una segunda variable, que contendrá los mismos datos que la primera.

```
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;
```



Para saber más

En este documento puedes encontrar información general sobre las bases de datos objeto-relacionales, incluyendo información y ejemplos de la función **VALUE**:



[Bases de datos objeto-relacionales.](#) (0.20 MB)

7.7.- Referencias a objetos.

El paso de objetos a un método resulta ineficiente cuando se trata de objeto de gran tamaño, por lo que es más conveniente **pasar un puntero a dicho objeto**, lo que permite que el método que lo recibe pueda hacer referencia a dicho objeto sin que sea necesario que se pase por completo. Ese puntero es lo que se conoce en Oracle como una **referencia (REF)**.



Al compartir un objeto mediante su referencia, **los datos no son duplicados**, por lo que cuando se hace cualquier cambio en los atributos del objeto, se producen en un único lugar.

Cada objeto almacenado en una tabla tiene un **identificador de objeto** que identifica de forma única al objeto guardado en una determinada fila y sirve como una referencia a dicho objeto.

Las referencias se crean utilizando el modificador **REF** delante del tipo de objeto, y se puede usar con variables, parámetros, campos, atributos, e incluso como variables de entrada o salida para sentencias de manipulación de datos en SQL.

```
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;  
/
```

Hay que tener en cuenta que sólo se pueden usar **referencias a tipos de objetos que han sido declarados previamente**. Siguiendo el ejemplo anterior, no se podría declarar el tipo Partida antes que el tipo Usuario, ya que dentro del tipo Partida se utiliza una referencia al tipo Usuario. Por tanto, primero debe estar declarado el tipo Usuario y luego el tipo Partida.

El problema surge cuando tengamos dos tipos que utilizan referencias mutuas. Es decir, un atributo del primer tipo hace referencia a un objeto del segundo tipo, y viceversa. Esto se puede solucionar haciendo una **declaración de tipo anticipada**. Se realiza indicando únicamente el nombre del tipo de 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*/  
);  
/
```


7.8.- Navegación a través de referencias.

Debes tener en cuenta que **no se puede acceder directamente a los atributos de un objeto referenciado que se encuentre almacenado en una tabla**. Para ello, puedes utilizar la función **DEREF**.

Esta función **toma una referencia a un objeto y retorna el valor** de ese objeto.

Vamos a verlo en un ejemplo 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.

```
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 sola 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`.



Para saber más

En este documento puedes encontrar información general sobre todo lo tratado en esta unidad, incluyendo ejemplos de tablas de objetos con uso de referencias:








[Bases de Datos Objeto-Relacionales en Oracle 8.](#) (0.06 MB)

Anexo.- Licencias de recursos.

Licencias de recursos utilizados en la Unidad

Recurso (1)	Datos del recurso (1)	R
	<p>Autoría: Ministerio de Educación. Licencia: Uso Educativo no comercial. Procedencia: Elaboración propia con imagen de http://commons.wikimedia.org/wiki/File:View-refresh.svg</p>	
	<p>Autoría: Manco Capac. Licencia: Creative Commons Attribution-Share Alike. Procedencia: http://commons.wikimedia.org/wiki/File:Megaphone.svg</p>	
	<p>Autoría: Gauthier Tellier. Licencia: GNU/GPL. Procedencia: http://commons.wikimedia.org/wiki/File:Applications-mplayer.svg</p>	
	<p>Autoría: Ministerio de Educación. Licencia: Uso Educativo no comercial. Procedencia: Elaboración propia a partir de imágenes de dominio público:</p> <ul style="list-style-type: none"> ✓ http://commons.wikimedia.org/wiki/File:Emblem-person-purple.svg ✓ http://commons.wikimedia.org/wiki/File:Emblem-person-yellow.svg 	
	<p>Autoría: Everaldo Coelho and Yellowlcon. Licencia: GNU/GPL. Procedencia: http://commons.wikimedia.org/wiki/File:Crystal_Clear_device_blockdevice.png</p>	
	<p>Autoría: Oracle. Licencia: Copyright (cita). Procedencia: Elaboración Propia. Captura de pantalla de la línea de comandos SQL de Oracle Database.</p>	

	<p>Autoría: Oracle. Licencia: Copyright (cita). Procedencia: Elaboración Propia. Captura de pantalla de la línea de comandos SQL de Oracle Database.</p>	
	<p>Autoría: Ministerio de Educación. Licencia: Uso Educativo no comercial. Procedencia: Elaboración propia a partir de imágenes de dominio público:</p> <ul style="list-style-type: none"> ✓ http://commons.wikimedia.org/wiki/File:Emblem-person-purple.svg ✓ http://commons.wikimedia.org/wiki/File:Emblem-person-green.svg ✓ http://commons.wikimedia.org/wiki/File:Emblem-person-red.svg ✓ http://commons.wikimedia.org/wiki/File:Document-save.svg ✓ http://commons.wikimedia.org/wiki/File:Emblem-person-yellow.svg 	
	<p>Autoría: Ministerio de Educación. Licencia: Uso Educativo no comercial. Procedencia: Elaboración propia a partir de imágenes de dominio público:</p> <ul style="list-style-type: none"> ✓ http://commons.wikimedia.org/wiki/File:Emblem-person-purple.svg ✓ http://commons.wikimedia.org/wiki/File:Emblem-person-green.svg ✓ http://commons.wikimedia.org/wiki/File:Emblem-person-red.svg ✓ http://commons.wikimedia.org/wiki/File:Document-save.svg ✓ http://commons.wikimedia.org/wiki/File:Edit-clear.svg 	