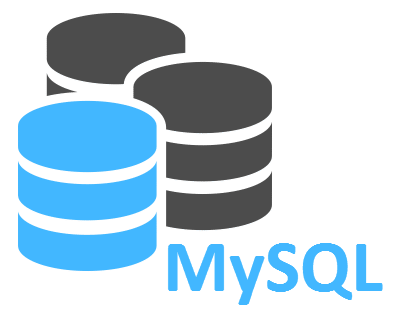
**PROYECTO BASES DE OBJETOS RELACIONALES**



Juan Sáez García

[juamber.com](https://juamber.com/)

**ÍNDICE**

[**1. ¿QUÉ SON LAS CLASES Y OBJETOS?**](#_9xwjf49gt4x0) **3**

[**2. CREACIÓN DE CLASES EN SQL**](#_298ytflzb2im) **4**

[CREATE OR REPLACE TYPE Animal AS OBJECT ();](#_ow8vwcvmcq8d) 4

[2.1 CREACIÓN DE ATRIBUTOS EN UNA CLASE](#_fnewjdte6pvy) 4

[CREATE OR REPLACE TYPE Animal AS OBJECT (](#_u9l5825oeqkq) 4

[2.2 CREACIÓN DE MÉTODOS EN UNA CLASE](#_fnewjdte6pvy) 5

[CREATE OR REPLACE TYPE Animal AS OBJECT (](#_oa3xi1ol5mys) 5

[**3. MODIFICAR Y BORRAR CLASES**](#_pr8amby48n0w) **5**

[**4. SELF**](#_g661pidcwc4c) **6**

[**5. CONSTRUCTOR**](#_kjz76wz3l0x7) **6**

[**6. OBJETOS**](#_f6mshsgrzpcw) **8**

[6.1 DECLARACIÓN E INICIALIZACIÓN DE OBJETOS](#_yxkt54ujev2g) 8

[6.2 ACCESO A ATRIBUTOS DE LOS OBJETOS](#_h2kfzu90am56) 8

[6.3 LLAMADA A LOS MÉTODOS DE LOS OBJETOS](#_l6wmt976d1mo) 8

[**7. HERENCIA**](#_cxu1ulzg5ljl) **9**

[**8. MÉTODOS MAP Y ORDER**](#_q5kem3e6wluz) **10**

[**10. TABLAS CON COLUMNAS TIPO OBJETO**](#_javp778ymj09) **12**

[**10. SENTENCIA SELECT CON OBJETOS**](#_z3jkti3apm93) **13**

[**11. INSERCIÓN, MODIFICACIÓN Y BORRADO DE OBJETOS**](#_kfchbhfdcdo6) **13**

[11.1 INSERCIÓN](#_8bclabr1zn6q) 13

[11.2 MODIFICACIÓN](#_70fafn9wivr8) 14

[11.2 BORRADO](#_if3r4o8ajl7c) 14

[**12. SENTENCIA VALUE**](#_6yoif2xuh1sl) **15**

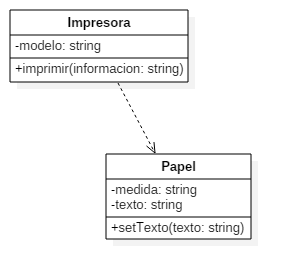
[**13. REFERENCIAS A OBJETOS**](#_bo3av9l9mr80) **16**

[13.1 NAVEGACIÓN ENTRE REFERENCIAS](#_hgyr0iw590px) 17

[**8. CONCLUSIONES**](#_2mrgcxl0pulu) **18**

[**9. BIBLIOGRAFÍA Y REFERENCIAS**](#_lk3hc7olz1xb) **19**

# 1. ¿QUÉ SON LAS CLASES Y OBJETOS?



Las clases son la abstracción de un concepto. Las clases están formadas por atributos y métodos. Los objetos son la materialización de la clase dándole valores a los atributos.

Pueden haber infinidad de objetos pero solo una clase.

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.

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 concurrencia y el rendimiento de las bases de datos relacionales.

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.

# 2. CREACIÓN DE CLASES EN SQL

La creación de una clase se hace con la sentencia CREATE TYPE “nombre” AS OBJECT para luego ser utilizada como tipo de datos en las columnas de nuestras tablas.

## CREATE OR REPLACE TYPE Animal AS OBJECT ();

## 2.1 CREACIÓN DE ATRIBUTOS EN UNA CLASE

Los atributos dentro de una clase no son más que variables de algún tipo de dato que van ligados a la clase, es decir que toman valor una vez creado un objeto o en el transcurso de su utilización. Para asignarle atributos se hace de igual manera que en la creación de columnas dentro de una tabla.

## CREATE OR REPLACE TYPE Animal AS OBJECT (

sexo ENUM(“M”,”F”,”N”),

tipo VARCHAR(40),

edad INT

);

## 2.2 CREACIÓN DE MÉTODOS EN UNA CLASE

Los métodos se crean con la sentencia MEMBER FUNCTION o PROCEDURE. Su funcionalidad se definirá fuera de la clase.

## CREATE OR REPLACE TYPE Animal AS OBJECT (

sexo ENUM(“M”,”F”,”N”),

tipo VARCHAR(40),

edad INT

MEMBER FUNCTION pull RETURN BOOLEAN

MEMBER PROCEDURE push ( n IN INTEGER )

);

CREATE TYPE BODY Animal AS

MEMBER PROCEDURE push ( n IN INTEGER ) IS

BEGIN

IF NOT full THEN

top:=top+1;

END push;

END

# 

# 3. MODIFICAR Y BORRAR CLASES

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 datos. Utilizando MODIFY ATTRIBUTE se podrán modificar los atributos existentes, y para eliminar atributos se dispone de manera similar de DROP ATTRIBUTE.

ALTER TYPE Usuario

ADD ATTRIBUTE cp VARCHAR(5),

MODIFY ATTRIBUTE nombre VARCHAR(35);

ALTER TYPE Usuario DROP ATTRIBUTE ingreso;

# 4. SELF

SELF es siempre el primer parámetro pasado a un método. Por ejemplo, el método transform declara SELF como un parámetro IN OUT:

CREATE TYPE Complex AS OBJECT (

MEMBER FUNCTION transform (SELF IN OUT Complex)

SELF se usa dentro de los métodos de una clase para hacer referencia a los atributos de la clase usando esta sintaxis:

SELF.variable

# 5. CONSTRUCTOR

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).

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;

# 

# 6. OBJETOS

## 6.1 DECLARACIÓN E INICIALIZACIÓN DE OBJETOS

Para declarar un objeto se le pasan los parámetros del constructor en su orden y tipo correcto con la sentencia NEW.

variable\_objeto := NEW Nombre\_Tipo\_Objeto (valor\_atributo1, valor\_atributo2);

## 6.2 ACCESO A ATRIBUTOS DE LOS OBJETOS

Acceder a los atributos de un objeto es tan sencillo como poner el nombre del objeto declarado anteriormente más “.” y luego el nombre del atributo.

unNombre := usuario1.nombre;

## 6.3 LLAMADA A LOS MÉTODOS DE LOS OBJETOS

Llamar a los métodos es tan sencillo como poner el nombre del objeto declarado anteriormente más “.” y luego el nombre del la función más “()” y si necesita parámetros para su función se le pasarán entre las llaves.

credito := usuario1.getCredito();

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

# 7. HERENCIA

La herencia se da cuando una clase hereda de una clase, se dice que es su clase padre. Contienen todos los atributos y métodos del tipo padre, pero además pueden contener atributos y métodos adicionales, o incluso sobreescribir métodos del tipo padre.

CREATE TYPE Persona AS OBJECT (

nombre VARCHAR(20),

apellidos VARCHAR(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;

# 8. MÉTODOS MAP Y ORDER

Los valores de un tipo escalar, como CHAR o REAL, tienen un orden predefinido que permite compararlos. Sin embargo, las instancias de un objeto carecen de un orden predefinido. Para ordenarlas, el PL/SQL invoca a un método de MAP definido por el usuario. En el siguiente ejemplo, la palabra clave MAP indica que el método convert ordena los objetos Rational proyectándose como números reales:

CREATE TYPE Rational AS OBJECT (

num INTEGER ,

den INTEGER ,

MAP MEMBER FUNCTION convert RETURN REAL

) ;

CREATE TYPE BODY Rational AS

MAP MEMBER FUNCTION convert RETURN REAL IS

BEGIN RETURN num / den;

END convert ;

END;

El PL/SQL usa esta función para evaluar expresiones booleanas como x > y y para las comparaciones implícitas que requieren las cláusulas DISTINCT, GROUP BY y ORDER BY. Un tipo de objeto puede contener sólo una función de MAP, que necesariamente debe carecer de parámetros y debe devolver uno de los siguientes tipos escalares: DATE, NUMBER, VARCHAR2 y cualquiera de los tipos ANSI SQL (como CHARACTER o REAL). Alternativamente, es posible definir un método de ordenación (ORDER). Un método ORDER utiliza dos parámetros: el parámetro predefinido SELF y otro objeto del mismo tipo. En el siguiente ejemplo, la palabra clave ORDER indica que el método match compara dos objetos. Supongamos que c1 y c2 son objetos del tipo Customer. Una comparación del tipo c1 > c2 invoca al método match automáticamente. El método devuelve un número negativo, cero o positivo que indica que SELF es respectivamente menor, igual o mayor que el otro parámetro:

CREATE TYPE Customer AS OBJECT (

id NUMBER,

name VARCHAR(20) ,

addr VARCHAR(30) ,

ORDER MEMBER FUNCTION match ( c Customer ) RETURN INTEGER

) ;

CREATE TYPE BODY Customer AS

ORDER MEMBER FUNCTION match ( c Customer ) RETURN INTEGER IS

BEGIN IF i d < c . i d THEN

RETURN 1 ;

ELSEIF i d > c . i d THEN

RETURN 1 ;

ELSE

RETURN 0 ;

END IF ;

END;

END;

# 10. 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 alguna 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 VARCHAR(10),

unUsuario Usuario,

partidasJugadas SMALLINT

);

# 10. SENTENCIA SELECT CON OBJETOS

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;

# 11. INSERCIÓN, MODIFICACIÓN Y BORRADO DE OBJETOS

## 11.1 INSERCIÓN

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));

## 11.2 MODIFICACIÓN

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) WHERE u.login = 'caragu72';

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;

## 11.2 BORRADO

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;

# 12. SENTENCIA 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);

# 13. 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 VARCHAR(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);

CREATE OR REPLACE TYPE tipo2 AS OBJECT ( tipo1\_ref REF tipo1);

## 13.1 NAVEGACIÓN ENTRE 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 variables 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 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.

# 8. CONCLUSIONES

Usar objetos es otra manera para almacenar información en bases de datos. Su punto fuerte bajo mi juicio son los métodos de las clases que le dan un aspecto funcional a los registros. Para bases de datos que por su estructura requieren de mucha información enlazada directamente desde una misma tabla me parece una solución limpia e interesante.

# 9. BIBLIOGRAFÍA Y REFERENCIAS

<https://www.sitiolibre.com/curso/pdf/BD07.pdf>

<http://www.xtec.cat/~iguixa/materialsGenerics/DAMDAW_M02_UF4_UV_BDOR.pdf>