BASES DE DATOS OBJETO-RELACIONALES

Hugo Pelayo Aseko

Bases de datos

4 de mayo de 2023

Índice

Introducción

Los sistemas de gestión de bases de datos objeto-relacional surgieron de investigaciones realizadas en la década de 1990. Estas investigaciones ampliaron los conceptos de las bases de datos relacionales añadiendo conceptos de objetos. Los investigadores intentaron mantener como componente central de la arquitectura un lenguaje de consulta declarativo basado en cálculo de predicados. El proyecto de investigación más destacado, Postgres (UC Berkeley), dio lugar a dos productos que se derivan de esa investigación: Illustra y PostgreSQL.

A mediados de la década de 1990 aparecieron los primeros productos comerciales, como Illustra, Omniscience y UniSQL. También se desarrolló la primera versión de la base de datos Valentina, creada por el desarrollador ucraniano Ruslan Zasukhin, fundador de Paradigma Software, como un SDK de C++. En la siguiente década, PostgreSQL se convirtió en una base de datos comercialmente viable y es la base de varios productos actuales que mantienen sus características de ORDBMS.

Los científicos de la computación comenzaron a referirse a estos productos como sistemas de gestión de bases de datos objeto-relacional o ORDBMS. Muchas de las ideas de los primeros esfuerzos de bases de datos objeto-relacional se han incorporado en gran medida en SQL:1999 a través de tipos estructurados. De hecho, cualquier producto que cumpla con los aspectos orientados a objetos de SQL:1999 podría describirse como un producto de gestión de bases de datos objeto-relacional. Por ejemplo, IBM Db2, Oracle Database y Microsoft SQL Server, afirman admitir esta tecnología.

Las bases de datos objeto-relacionales son bases de datos que siguen el modelo de datos relacional proporcionando características de la programación orientada a objetos.

En un DBMS relacional, los datos se organizan en tablas con filas y columnas, mientras que, en un modelo de datos orientado a objetos, los datos se representan como objetos que contienen propiedades y métodos. Un ORDBMS (del inglés *Object Relational Daatbase Management System*) combina estas dos formas de modelado de datos, permitiendo que los objetos se almacenen y se recuperen a través de una interfaz SQL, que es el lenguaje de consulta estructurado utilizado en los sistemas de gestión de bases de datos relacionales.

Las bases de datos objeto-relacionales permiten una mayor flexibilidad en el modelado de datos que los sistemas de bases de datos relacionales tradicionales. Al utilizar un modelo de datos orientado a objetos, se pueden representar relaciones complejas entre los datos, como objetos que contienen otros objetos, o incluso objetos que son heredados de otros objetos.

Además, los ORDBMS también permiten que se definan tipos de datos personalizados y que se definan funciones y procedimientos almacenados, lo que puede simplificar el proceso de programación de aplicaciones que utilizan la base de datos.

Los ORDBMS también pueden ser escalables y eficientes en términos de rendimiento. Algunos sistemas ORDBMS utilizan técnicas de optimización de consultas, como la indexación y el almacenamiento en caché, para acelerar la recuperación de datos y mejorar el rendimiento de la base de datos.

El modelo objeto-relacional es una combinación de las técnicas orientadas a objetos y las bases de datos relacionales, lo que proporciona una gran cantidad de ventajas. Con este modelo, los desarrolladores pueden acceder a los datos de manera más intuitiva, como si fueran entidades de la vida real, lo que facilita la comprensión y el manejo de los datos.

El uso de tipos de objetos permite organizar y acceder a los datos de una manera más eficiente, y al mismo tiempo se mantiene la alta capacidad de concurrencia y el rendimiento de las bases de datos relacionales. Los tipos de objetos permiten modelar objetos de la vida real, consiguiendo una implementación que se abstrae de los detalles innecesarios de los objetos que se modela, y almacenar los datos orientados a objetos de forma permanente en una base de datos.

La programación orientada a objetos es especialmente útil para la construcción de componentes reutilizables y aplicaciones complejas. En PL/SQL, la programación orientada a objetos se basa en tipos de objetos, lo que 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 de Oracle, ya que esta ofrece soporte nativo para la manipulación de datos utilizando el paradigma de la programación orientada a objetos.

Además, los objetos pueden incluir acciones para realizar ciertas tareas sobre los datos, lo que facilita la obtención de información de manera más sencilla y eficiente. Por ejemplo, un objeto "Pedido" puede incluir un método para calcular el importe total de los artículos comprados, mientras que un objeto "Cliente" puede tener métodos que permitan obtener su historial de compras. Al utilizar estos métodos, las aplicaciones pueden obtener la información necesaria sin tener que realizar complejas consultas a la base de datos.

El objetivo principal de una base de datos objeto-relacional es implantar en las bases de datos relacionales las técnicas de modelado orientado a objetos que se utilizan en lenguajes de programación como Java, C++ o C#, por ejemplo, la abstracción de entidades sobre clases, la implementación de la herencia entre tipos, entre otras

características. Una alternativa popular para integrar el modelo de datos objeto-relacional en las bases de datos relacionales consiste en utilizar un sistema de bases de datos relacionales estándar con algún tipo de software de mapeo objeto-relacional (ORM). A diferencia de los productos tradicionales de RDBMS o SQL-DBMS que se enfocaban en la gestión eficiente de datos extraídos de un conjunto limitado de tipos de datos (definidos por los estándares del lenguaje relevante), un ORDBMS permite a los desarrolladores de software integrar sus propios tipos y los métodos que se aplican a ellos en el DBMS.

El ORDBMS se integra con un lenguaje de programación orientado a objetos. Las propiedades características de ORDBMS son:

1. datos complejos
2. herencia de tipos
3. comportamiento de objetos.

La creación de datos complejos en la mayoría de los ORDBMS de SQL se basa en una definición de esquema preliminar a través del tipo definido por el usuario (UDT). La jerarquía dentro de los datos complejos estructurados ofrece una propiedad adicional, la herencia de tipos, es decir, un tipo estructurado puede tener subtipos que reutilicen todos sus atributos y contengan atributos adicionales específicos del subtipo. Otra ventaja, el comportamiento de objeto, está relacionada con el acceso a los objetos del programa. Dichos objetos del programa deben ser almacenables y transportables para el procesamiento de la base de datos, por lo tanto, generalmente se les llama objetos persistentes. Dentro de una base de datos, todas las relaciones con un objeto de programa persistente son relaciones con su identificador de objeto. Todos estos puntos se pueden abordar en un sistema relacional adecuado, aunque el estándar SQL y sus implementaciones imponen restricciones arbitrarias y una complejidad adicional.

En la programación orientada a objetos (OOP), el comportamiento del objeto se describe a través de los métodos, que son operaciones que se pueden realizar sobre el objeto. Los métodos designados con un nombre se distinguen por el tipo de sus parámetros y el tipo de objetos a los que están asociados, conjunto de elementos que se conoce como firma del método. Los lenguajes de OOP llaman a esto el principio de polimorfismo, que se define brevemente como *one interface, many implementations*, en castellano "una interfaz, muchas implementaciones" [1]. Otros principios de OOP, como la herencia y la encapsulación, están relacionados tanto con los métodos como con los atributos. La herencia de métodos está incluida en la herencia de tipos.

Una base de datos objeto relacional es aquella que incorpora aquellos aspectos más destacados del paradigma de la programación orientada a objetos, como ejemplo

tenemos la encapsulación, propiedad que permite ocultar la información que contiene un objeto impidiendo la libre manipulación de los atributos de un objeto, permitiendo la consulta y modificación de estos atributos únicamente a través de métodos públicos que ofrece la clase que describe el objeto. Tenemos luego la herencia, que es una propiedad a través de la cual ciertos objetos pueden heredar los comportamientos y atributos de otra clase comúnmente llamada clase madre. Tenemos también el polimorfismo que es una propiedad que nos permite tener varios métodos con el mismo nombre, pero definiendo diferentes funcionalidades según convenga.

En una base de datos objeto relacional los datos no son atómicos ya que podemos tener estructuras que engloban otros tipos, por ende, no se puede garantizar que estén en 1FN, por tanto, el concepto de normalización no se aplica. Los objetos de una base de datos constan generalmente de atributos y métodos, ambos recogidos en estructuras que pueden llegar a ser de gran tamaño y que a la vez describen las diferentes relaciones que hay entre objetos de nuestra base de datos objeto-relacional.

Desde una perspectiva conceptual, un objeto en el contexto de bases de datos objeto relacionales puede ser equiparado a una entidad en el modelo E-R. No obstante, a nivel de implementación, un objeto es un contenedor que engloba tanto un conjunto de operaciones (es decir, código y datos) como una unidad única que no es visible externamente. Esta estructura protege la información contenida en el objeto, permitiendo realizar cambios sin afectar a otros objetos que interactúen con él.

La interacción entre un objeto y el resto del sistema se lleva a cabo a través de un conjunto de mensajes, mientras que el objeto en sí consta de un conjunto de propiedades o atributos que contienen los datos del objeto (equivalentes a los atributos en el modelo E-R), un conjunto de mensajes a los que el objeto responde y un conjunto de métodos que se encargan de implementar dichos mensajes, devolviendo un valor como respuesta

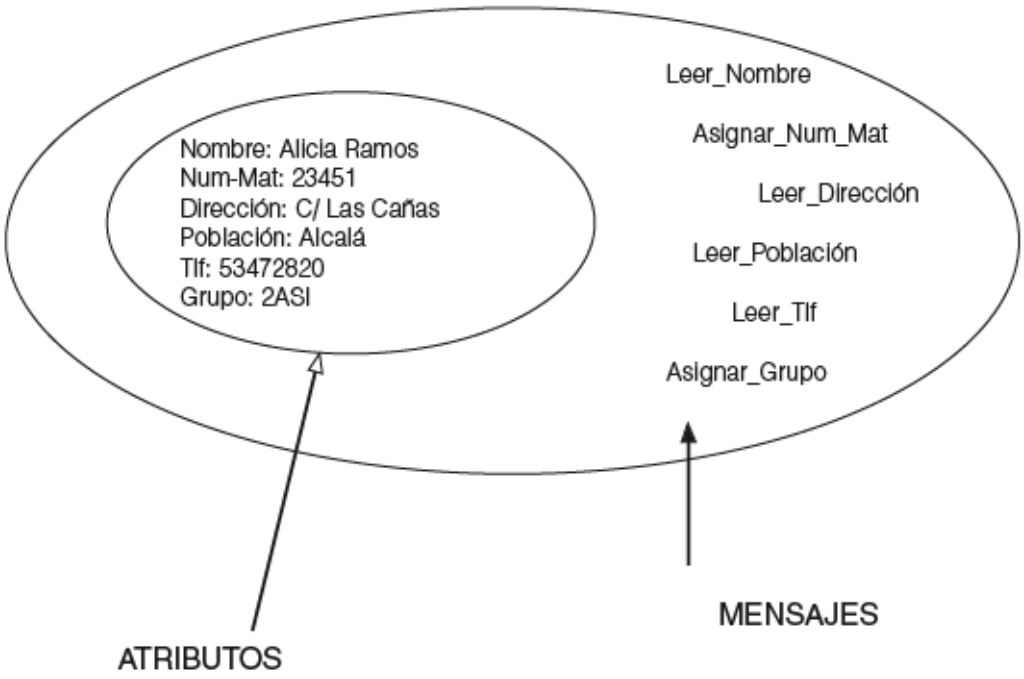

Figura 1. Representación objeto alumno. Fuente: Libro SGBD

Los objetos en sí siguen una misma estructura predefinida, la cual viene descrita por una clase. Todos los objetos de la misma clase comparten el comportamiento y los mismos atributos, aunque difieren en los valores que estos últimos puedan contener.

Tipos de datos

Un tipo de dato hace referencia a una estructura definida por el usuario que puede contener uno o más tipos de datos. Contiene atributos y generalmente funciones y procedimientos con los cuales podemos operar sobre los atributos. Las variables simples o *built-in* permiten guardar únicamente un tipo de dato, por ejemplo, `INT`, `VARCHAR`, `DATETIME`, entre otros. Sin embargo, las estructuras nos permiten englobar varios de los tipos mencionados anteriormente e incluso otras estructuras, de tal manera que se pueden anidar.

Normalmente pensamos en los objetos como un grupo de atributos y métodos con los cuales podemos operar sobre los atributos. Por ejemplo, si tenemos un objeto Persona, a este le podemos asignar los atributos documento nacional de identidad, edad, fecha de

nacimiento, estudios, progenitores, entre otros. Partiendo de estos atributos podemos generar métodos que nos permitan trabajar sobre estos datos, así como consultarlos o modificarlos. Podemos en estas situaciones hacer ciertas optimizaciones como, por ejemplo, no guardar la edad y tener un método que nos puede calcular la edad de cualquier objeto persona en base a la fecha de nacimiento.

Definición de tipos de objeto

Un tipo de objeto se compone de dos partes: la especificación y el cuerpo. La especificación sirve como interfaz para las aplicaciones, ya que aquí se definen las estructuras de datos, que consisten en un conjunto de atributos, y las operaciones, que son los métodos necesarios para manipular los datos. El cuerpo, por otro lado, es donde se implementan los métodos definidos en la especificación, que define la forma en que los objetos de un tipo determinado se pueden utilizar y manipular, mientras que el cuerpo proporciona la funcionalidad real para llevar a cabo esas operaciones.

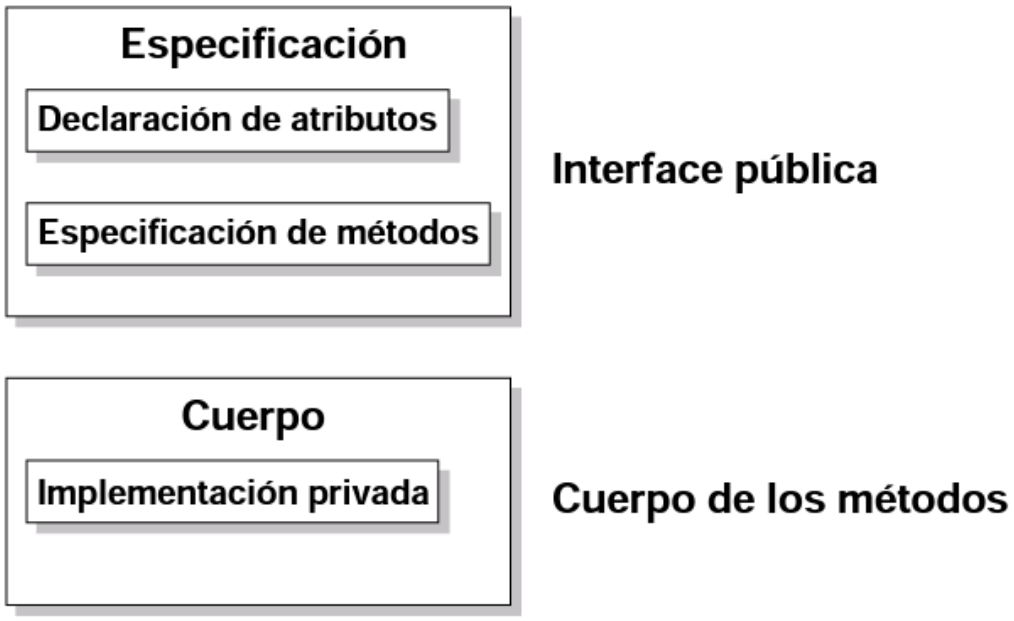

Figura 2. Definición objeto. Fuente: Recurso UV

La especificación y el cuerpo son dos partes de un tipo de objeto, donde la especificación se encarga de definir la interfaz para las aplicaciones y donde se declaran las estructuras

de datos y los métodos necesarios para manipular los datos. El cuerpo, por su parte, implementa los métodos definidos en la especificación.

Toda la información necesaria para que un usuario pueda utilizar los métodos se encuentra en la especificación, que puede ser vista como la interfaz operacional, mientras que el cuerpo se puede ver como una caja negra. Es posible depurar, mejorar o reemplazar el cuerpo sin necesidad de modificar la especificación y sin afectar a las aplicaciones usuario.

En una especificación de tipo de objeto, los atributos deben ser declarados antes que cualquier método y no es posible declarar atributos en el cuerpo del tipo de objeto. Todas las declaraciones hechas en al área de especificación del tipo son públicas y, por tanto, visibles fuera del tipo de objeto, mientras que el cuerpo puede contener declaraciones privadas, que definen métodos imprescindibles para el funcionamiento privados de nuestro tipo de objeto. El ámbito de estas declaraciones privadas es local al cuerpo del objeto, es decir, no es posible acceder a ellas de manera directa fuera de donde se define el tipo de objeto.

Cada objeto tiene un conjunto de atributos que definen su estructura y características, una de ellas son los atributos. Estos pueden ser de varios tipos, como números, cadenas de caracteres, fechas, objetos, entre otros. Al igual que en las bases de datos relacionales, los atributos en las bases de datos objeto relacionales pueden tener restricciones de integridad que se aplican a los valores que se almacenan en ellos.

En la programación orientada a objetos, los atributos se denominan **propiedades** o **variables miembro**, y se definen en las clases que definen la estructura de los objetos. En las bases de datos objeto relacionales, los atributos se definen en los tipos de objetos que se utilizan para crear las tablas.

En el contexto de las bases de datos objeto relacionales, los métodos se definen como subprogramas declarados en la especificación de un tipo de objeto mediante la palabra clave `MEMBER`. Es importante destacar que el nombre del método no puede ser el mismo que el del tipo de objeto ni de ninguno de sus atributos.

En la mayoría de los casos, un método consta de dos partes: la especificación y el cuerpo. La especificación incluye el nombre del método, una lista opcional de parámetros y, en el caso de las funciones, un tipo de retorno. El cuerpo es el código que se ejecuta para llevar a cabo una operación específica.

Es necesario que exista un cuerpo correspondiente para cada especificación de método en la especificación de tipo. El PL/SQL realiza una comparación entre la especificación y

el cuerpo del método token a token, por lo que las cabeceras deben coincidir palabra por palabra.

```
CREATE TYPE Stack AS OBJECT (
    top INTEGER,
    MEMBER FUNCTION full RETURN BOOLEAN,
    MEMBER FUNCTION push (n IN INTEGER),
    ...
);
/

CREATE TYPE BODY Stack AS
    ...
    MEMBER FUNCTION push (n IN INTEGER) IS
    BEGIN
        IF NOT full THEN
            top := top + 1;
            ...
    END push;
END;
/
```

Figura 3. Declaración y definición tipo Stack. Fuente: Recurso BDOR UV

Al instanciar un método cualquiera de un objeto, todos reciben como primer parámetro una instancia predefina del mismo tipo, se le suele llamar parámetro implícito en otros lenguajes de programación, en PL/SQL este parámetro se llama `SELF`. Indistintamente de que se declare de manera explícita en la lista de parámetros al definir un método o no, este parámetro siempre ocupa la primera posición de la lista de parámetros de los métodos de un objeto.

Si no se declara el parámetro `SELF` de manera explícita el definir un método, tiene modificadores de acceso variados según se trate de un método o una función. En el caso de funciones es un parámetro de entrada (`SELF IN`) y en el caso de método es un parámetro de entrada y salida (`SELF IN OUT`).

En PL/SQL para bases de datos objeto relacionales, cada tipo de objeto tiene un constructor que es una función definida por el sistema con el mismo nombre que el objeto. El constructor se utiliza para inicializar una instancia de ese tipo de objeto. Se genera un constructor por defecto para cada tipo de objeto, cuyos parámetros coinciden

con los atributos del tipo de objeto, esto es, se declaran en el mismo orden y tienen el mismo nombre y tipo.

Es importante destacar que PL/SQL no invoca al constructor de manera implícita, por lo que el usuario debe invocarlo explícitamente para instanciar objetos de un tipo específico.

Es posible definir constructores personalizados para los tipos de objetos, lo que sería sobrecargar los constructores, cosa que permite tener más control sobre el proceso de inicialización y validación de las instancias. Al utilizar constructores personalizados, se pueden llevar a cabo operaciones adicionales ya estos son definidos por el usuario. Como ejemplo, podemos validar los valores de atributos o la asignación de valores predeterminados.

Herencia

PL/SQL es un lenguaje que soporta la herencia simple de tipos de objetos, lo cual permite definir subtipos de los tipos de objeto existentes. Estos subtipos, también conocidos como tipos heredados, incluyen todos los atributos y métodos del tipo padre, pero tienen la capacidad de añadir atributos y métodos adicionales o incluso sobrescribir los métodos del tipo padre. Para especificar que un tipo de objeto es heredado de otro, se utiliza la palabra clave `UNDER`. Cabe destacar que el tipo de objeto del que se hereda debe tener la propiedad `NOT FINAL`. Por defecto los tipos de objetos estás declarados como `FINAL`, lo que significa que no es posible crear un tipo de objeto que herede de ellos.

Es posible declarar tipos de objeto que no se puedan instanciar utilizando la opción `NOT INSTANTIABLE`. Estos tipos de objeto tienen como finalidad ser utilizados como tipos padre para otros tipos de objeto.

```
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;
/
```

Figura 4. Ejemplo herencia. Fuente: Recurso Tema 7

Tipos de datos colección

En las bases de datos objeto-relacionales, los tipos de datos colección son tipos de datos que permiten almacenar conjuntos de valores de un mismo tipo. Estos tipos de datos permiten modelar estructuras de datos más complejas que los tipos de datos simples, como números y cadenas de caracteres.

Algunos ejemplos de tipos de datos colección incluyen:

Tablas: Las tablas son colecciones de registros que tienen el mismo conjunto de columnas. Cada columna de la tabla representa un atributo o campo distinto. Las tablas son uno de los tipos de datos más comunes en las bases de datos objeto-relacionales.

- Arrays: son colecciones de elementos del mismo tipo, a los que se accede mediante un índice. Pueden ser de tamaño fijo o de tamaño variable.

- Listas: son colecciones de elementos del mismo tipo, a los que se accede mediante un índice o mediante una referencia al elemento anterior o siguiente. Las listas pueden ser de tamaño variable y pueden crecer o disminuir a medida que se añaden o eliminan elementos.

- Conjuntos: Los conjuntos son colecciones de elementos únicos del mismo tipo. Los conjuntos no permiten elementos duplicados y no están ordenados.

- Mapas: Los mapas son colecciones de pares clave-valor, donde cada clave se corresponde con un valor asociado. Los mapas permiten buscar y acceder a valores mediante su clave.

La declaración de las colecciones tiene el siguiente formato:

```
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;
```

En estas declaraciones `nombre_tipo` representa el nombre de nuestra colección, `tamaño_max` es el máximo número de elementos que podrá contener nuestra colección y `tipo_elemento` representa el tipo de objeto que contendrá nuestra colección. A continuación, se adjunta unos ejemplos de declaración:

En el momento de declarar una colección, esta tiene valor nulo. Para inicializar una colección debemos utilizar su correspondiente constructora. La constructora es un método especial del mismo nombre de la colección a la cual podemos pasar parámetros que requiere la función constructora.

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

Figura 5. Declaración e inicialización colección. Fuente: Recurso Tema 7

La inicialización también se puede hacer en el bloque de declaración:

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

Figura 6. Inicialización en bloque DECLARE. Fuente: Recurso Tema 7

Los objetos de una base de datos objeto-relacional también se pueden almacenar en tablas, al igual que los tipos de datos convencionales de las bases de datos. Es posible utilizar tipos de datos objeto para formar una tabla exclusivamente compuesta por elementos de ese tipo, o bien, para utilizarlos como un tipo de columna más junto con

```
CREATE TABLE nombre_tabla OF tipo_de_objeto;
```

otros tipos de datos. Si se desea crear una tabla formada exclusivamente por un determinado tipo de dato objeto (conocida como tabla de objetos), se debe utilizar la sentencia `CREATE TABLE` junto con el tipo de objeto, de la siguiente manera:

Donde `nombre_tabla` es el nombre de nuestra tabla y `tipo_de_objeto` es el tipo de objeto que se almacena en nuestra tabla.

Se debe considerar que, si una tabla utiliza un tipo de objeto, no se podrá modificar ni eliminar la estructura de dicho tipo de objeto. Por lo tanto, una vez que el tipo de objeto sea utilizado en una tabla, es imposible redefinirlo.

Al instanciar 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 utilizado para crear objetos en cualquier momento.

Bibliografía

[1] Wikipedia, «Object-relational database,» 31 enero 2023. [En línea]. Available: https://en.wikipedia.org/wiki/Object%E2%80%93relational_database. [Último acceso: 7 mayo 2023].

[2] Wikipedia, «Bases de datos objeto-relacional,» 11 junio 2021. [En línea]. Available: https://es.wikipedia.org/wiki/Base_de_datos_objeto-relacional. [Último acceso: 7 mayo 2023].

[3] «BBD Objeto relacional - Bases De Datos Avanzada,» [En línea]. Available: https://sites.google.com/a/espe.edu.ec/bases-de-datos-ii/introduccion/bdd-objeto-relacional. [Último acceso: 2023 mayo 2023].

[4] A. R. F. M. María Jesus Ramos, Sistemas gestores de bases de datos, McGrawHill.

[5] J. L. Comesaña, «Recurso - Tema 7,» [En línea]. Available: https://www.sitiolibre.com/curso/pdf/BD07.pdf. [Último acceso: 2023 mayo 2023].

[6] W. D. Villanueva, «BASES DE DATOS OBJETO-RELACIONALES,» 2002. [En línea]. Available: chrome-extension://efaidnbmnnnibpcajpcglclefindmkaj/http://www.xtec.cat/~iguixa/materialsGenerics/DAMDAW_M02_UF4_UV_BDOR.pdf. [Último acceso: 13 mayo 2023].