# TEMA-9-ACCESO-A-DATOS.pdf

user_4383848

Acceso a datos

1º Desarrollo de Aplicaciones Multiplataforma

Estudios España

# TEMA 9 - ACCESO A DATOS
## BBDD ORIENTADAS A OBJETOS

### 1. DEFINICIÓN DE LAS BBDD ORIENTADAS A OBJETOS:

El concepto de bases de datos orientadas a objetos es cuando las técnicas de bases de datos se combinan con objetos. El resultado de esto es un sistema gestor orientado a objetos: **ODBMS (Object Data Base Management System)**.
**El modelo de datos orientado a objetos (OODM)** son modelos lógicos que capturan la semántica de los objetos de la aplicación que va unida o relacionada. Realmente, son modelos que van en consonancia con el modelo de la aplicación. Los OODMs implementan modelos conceptuales directamente y pueden representar complejidades que van más allá de los sistemas de bases de datos relacionales.
Han heredado muchos de los conceptos que fueron pensados e implementados en los lenguajes de programación orientados a objetos. Realmente, una base de datos orientada a objetos no es más que una colección de objetos definidos por un modelo orientado a objetos. Este tipo de bases de datos pueden extender la existencia de los objetos para que se almacenen indefinidamente. Por lo tanto, los objetos son almacenados más allá de la finalización de la aplicación con la que estemos trabajando. Pueden ser recuperados más tarde y compartidos por otras aplicaciones.

### 2. CARACTERÍSTICAS DE LAS BD ORIENTADAS A OBJETOS:

Características:
• Mantiene una relación directa entre el mundo real y los objetos de la base de datos. Los objetos no pierden ni su identidad ni su integridad.
• Proporciona un **identificador de objeto**, generado por el sistema para cada objeto para que un objeto pueda identificarse fácilmente.
• Son fáciles de extender y de añadir nuevos tipos de datos y operaciones que se realizarán en ellos.
• Proporciona encapsulación. La representación de la información y las implementaciones de los métodos ocultan entidades externas.
• También proporciona propiedades de herencia, mediante la cual un objeto hereda las propiedades de otros objetos.

Los objetos disponen de una serie de conceptos asociados, que los podríamos resumir en:
• **Atributos:** Son las características que suelen describir los objetos. También conocidos como variables de instancia. Cuando los atributos son asignados a valores en un momento dado, se asume que el objeto está en un estado determinado en ese momento.
• **Objeto:** Un objeto es una representación abstracta de una entidad del mundo real, la cual tiene un identificador único, propiedades embebidas y la capacidad de interactuar con otros objetos por sí mismo.
• **Identidad:** La identidad es un identificador externo (el ID del objeto) que se mantiene por cada objeto. El ID del objeto es asignado por el sistema cuando el objeto es creado y no puede ser cambiado. Es distinto a las bases de datos relacionales, ya que, por ejemplo, este ID está almacenado en el interior del objeto y, además, se usa para identificarlo.

## 3. SISTEMAS GESTORES DE BBDD ORIENTADOS A OBJETOS:

Se constituye con un sistema gestor de almacenamiento de datos que soporta el modelado y la creación de los datos como objetos. Permite la concurrencia y la recuperación. Para los consumidores de bases de datos relacionales, significa olvidarse de la traducción de filas y columnas, y, por lo tanto, manipular directamente con objetos.
Posee una serie de datos relacionados entre sí y una aplicación o aplicaciones rodeando dicha base de datos que tendrá acceso a ella.
Un **gestor de bases de datos genérico** debe poseer las características de:
• Concurrencia
• Persistencia
• Recuperación de errores
• Gestión de almacenamiento
• Consultas
Si hablamos de un sistema gestor de **base de datos orientado a objetos**, tendríamos alguna característica más:
• Abstracción
• Modularidad
• Jerarquía
• Encapsulación
• Tipología de objetos

## 4. LENGUAJE DE CONSULTAS PARA OBJETOS:

El lenguaje de consulta de objetos u **Object Query Language (OQL)** es un lenguaje declarativo de tipología SQL que nos facilitará realizar consultas de modo efectivo en bases de datos orientadas a objetos y a estructuras de los mismos.
Este lenguaje no contiene primitivas que se ocupen de modificar el estado de dichos objetos, ya que este tipo de modificaciones se realizarán a través de métodos que poseen los objetos.

### Agregación y asociación

La agregación es un tipo específico de asociación.
La diferencia entre asociación y agregación es que, cuando borramos un objeto que forma parte de una asociación,el resto de objetos relacionados continúan existiendo. Por el contrario, en la agregación, la introducción o borrado del objeto es igual a insertar o eliminar el resto de sus componentes relacionados. De esta forma, un objeto que pertenece a otro no puede ser introducido o borrado de forma aislada en la base de datos.

### Especialización, generalización y herencia

Definimos una clase para organizar una serie de objetos parecidos. En algunos casos, los objetos de una clase pueden ser organizados de nuevo formando ciertas agrupaciones que pueden ser relevantes para la base de datos.

### 4.1. Consultas de un objeto:

Continuando con la sintaxis del lenguaje OQL, hay distintas formas de consultar un objeto. Las variables llamadas de tipo «iterador» se pueden nombrar de 3 formas distintas:
• C in Customer
• Customer C
• Customer as C
El resultado de este tipo de query podría ser válido si forma parte de la tipología definida previamente en el modelo.
No es obligatorio que posea la cláusula SELECT, ya que, simplemente nombrando cualquier objeto, se devolverán todas sus existencias. Es decir:

• **Customer:** Devolverá una colección de todos los objetos de tipo Customer que existan en la base de datos. De la misma forma, si existiera un objeto concreto PremiumCustomer, se realizaría la siguiente consulta:
• **PremiumCustomer:** Obtendremos el resultado de ese tipo específico de Customer.
Por lo general, una consulta OQL comienza con el nombre del objeto persistente y, a continuación, se le añade uno, varios atributos o ninguno, mediante un punto. Veamos algún ejemplo:

customer.empresa
customer.empresa.nombre
customer.empresas_asociadas

En el **primer ejemplo,** devolvería un objeto de tipo Empresa, que estaría vinculado a ese Customer.
En el **segundo caso**, se accede al atributo de la empresa llamado nombre. Por lo tanto, nos devolvería un objeto de tipo String con el nombre de la empresa.
Y, en el **último ejemplo**, nos devolvería un Set<Empresa>, una colección de tipo Empresa donde podríamos ver todas las empresas que ese cliente tuviera asociadas.

## 5. VENTAJAS EN LA UTILIZACIÓN DE LAS BD ORIENTADAS A OBJETOS:

• Dispondremos de una **capacidad mayor de realizar el modelado**. Esto podemos considerarlo debido a:
    • Dentro de los objetos podremos encapsular tanto comportamientos como estados.
    • Las relaciones de un objeto pueden ser almacenadas en su interior.
    • Al agruparse, los objetos forman objetos complejos. Es el concepto que denominamos como herencia.
• También **dispondremos de una flexibilidad importante**. Esto se debe a:
    • Podremos construir nuevos objetos con tipologías nuevas, partiendo de los que ya tenemos.
    • Se reduce la redundancia, ya que podremos aunar características o propiedades de distintas clases y agruparlas en superclases.
    • Las clases existentes u objetos son reusables. Lo cual influye directamente en el tiempo de desarrollo.

• Por medio de la intuición, **podremos realizar de forma práctica algunas de las consultas,** ya que es un lenguaje expresivo. Es realmente fácil navegar entre objetos y sus herencias, ya que es un acceso navegacional.
• **Rendimiento muy competitivo.** Algunos autores han comparado rendimientos de bases de datos orientadas a objetos y bases de datos relacionales. En aplicaciones orientadas a objetos, el rendimiento de la base de datos orientada a objetos es superior.

## 6. DESVENTAJAS EN LA UTILIZACIÓN DE LAS BD ORIENTADAS A OBJETOS:

• **La falta de un modelo de datos universal:** No existe dicho modelo de datos universalmente aceptado y a la mayoría de los sistemas gestores de bases de datos orientados a objetos les falta una buena base teórica.
• **Falta de experiencia:** Es evidente que no es comparable con los sistemas de base de datos relacionales.
• **Falta de estándares:** Realmente, no existen estándares definidos como tal.
• **La competencia:** Tal y como hemos comentado anteriormente respecto a los sistemas relacionales, incluso los objeto-relacionales poseen una gran competencia, ya que dichos sistemas tienen un gran estándar aprobado, como SQL, y una base teórica sólida.
• **La encapsulación es casi una forma obligada de realizar consultas:** El desarrollo de los objetos de forma encapsulada es prácticamente una obligación, ya que es la forma que accederemos a futuro mediante el sistema de consultas.
• **En relación a la teoría matemática:** Podemos decir que el modelo de objetos aún no posee una aprobación.

## 7. PROGRAMACIÓN DE APLICACIONES CON ACCESO A BD ORIENTADAS A OBJETOS: DISEÑO DE UNA API:

**API (Application Programming Interface)** viene a definir una serie de especificaciones, de reglas a cumplir, para consumir ciertas funcionalidades de un sistema externo determinado. El concepto API está muy orientado a las aplicaciones web con patrón de diseño REST. En esta arquitectura, se definen una serie de EndPoints o puertas de entrada al código back de la aplicación con la que hay que conectar y de la que hay que obtener las funcionalidades o servicios.
Estas funcionalidades han debido ser previamente definidas y analizadas, y deben cubrir la totalidad de la lógica que un sistema de back-end puede devolver a un front-end.
**Los EndPoints** son, básicamente, métodos definidos como entrada para realizar dichas funcionalidades vinculados por un path que los denomina. Este sería el caso de una API en tecnología o patrón de diseño REST.

En nuestro caso, disponemos de una aplicación que en su tramo final accede a una base de datos orientada a objetos. En ese punto, **sería de gran interés analizar y diseñar una API en la capa DAO (Data Access Object)**, capa de la aplicación donde accedemos a datos. Sería recomendable (y signo de robustez y orden) diseñar una interfaz con una serie de métodos cuyo objetivo fuera poblar de todas las respuestas necesarias en cuanto a datos se refiere. Es decir, **definir los llamados EndPoints** cuyo nombre dan respuesta al objeto u objetos que se van a devolver.