**Ejercicios UD05**

**Bases de Datos Orientadas a Objetos**

Ciclo Superior de Desarrollo de Aplicaciones Multiplataforma

(DAM2)

Curso 2022/2023

Autor/a: [Carlos Rodrigo Pla](mailto:210176@iessierradeguara.com)

Módulo: Acceso a Datos.



# SESIÓN 1

**Ejercicio 1.-** De la misma manera que Codd, creador del modelo relacional, definió en 1985 12 reglas que debían cumplir un sistema gestor de bases de datos para ser considerado relacional, en los años 90 se definió un conjunto de reglas equivalente para los sistemas orientados a objetos, denominado *Manifiesto de las BDOO*, en diferentes etapas y por diferentes expertos en bases de datos como el profesor Malcolm Atkinson. Dicho manifiesto estaba formado por 13 reglas obligatorias + 5 opciones de implementación no obligatorias.

**Averigua y explica cada una de estas 13 reglas obligatorias** (p.ej. en un fichero Word).

## Manifiesto Malcolm

En 1989 se hizo el Manifiesto de los sistemas de base de datos orientados a objetos el cual propuso trece características obligatorias para un SGBDOO y cuatro opcionales.

1. **Deben soportarse objetos complejos**

Debe ser posible construir objetos complejos aplicando constructores a objetos básicos

1. **Deben soportarse mecanismos de identidad de los objetos**

Todos los objetos deben tener un identificador, el cual es independiente de los valores de sus atributos.

1. **Debe soportarse la encapsulación**

Los programadores solo tienen acceso a la interfaz de los métodos, y los datos e implementación de estos métodos están en los objetos.

1. **Deben soportarse los tipos o clases**

El esquema de una base orientada a objetos contiene un conjunto de clases o tipos.

Deben ser capaces de heredar de sus súper-tipos o superclases los atributos y los métodos.

1. **Los tipos o clases deben ser capaces de heredar de sus ancestros**

deben ser capaces de heredar de sus súper-tipos o superclases los atributos y los métodos.

1. **Debe soportarse el enlace dinámico**

La sobrecarga debe ser soportada, los métodos deben poder aplicarse a diferentes tipos.

1. **El DML debe ser computacionalmente complejo**

Debe ser un lenguaje de programación de propósito general.

1. **El conjunto de todos los tipos de datos debe ser ampliable**

No habrá distinción entre los tipos definidos por el usuario y los tipos definidos por el sistema,

1. **Debe proporcionarse persistencia a los datos**

Los datos deben mantenerse después de que la aplicación que los creó haya finalizado, el usuario no tiene que hacer copia explícitamente.

1. **El SGBD debe ser capaz de gestionar bases de datos de muy gran tamaño**.

Debe ser capaz de manejar bases de datos grandes.

1. **El SGBD debe soportar a usuarios concurrentes**

debe soportar la concurrencia. Debe disponer del mecanismo para el control de la concurrencia.

1. **El SGBD debe ser capaz de recuperarse de fallos hardware y software**

El sistema gestor debe proveer mecanismos de recuperación de la información en caso de fallo de sistema.

1. **El SGBD debe proporcionar una forma simple de consultar los datos.**

Debe proveer de manera fácil de hacer consulta**s**

**Características opcionales**

* Herencia múltiple
* Comprobación de tipos e inferencia de tipos
* Sistema de base de datos distribuido
* Soporte de versiones

**Ejercicio 2.-** Anota las ventajas e inconvenientes (en el fichero Word) de la utilización de bases de datos orientadas a objetos.

El uso de una BDOO puede ser ventajoso frente a una Base de Datos Relacional si nuestra aplicación requiere alguno de estos elementos :

* Un gran número de tipos de datos diferentes.
* Un gran número de relaciones entre los objetos.
* Objetos con comportamientos complejos.

Una de las principales ventajas de los sistemas de bases de datos orientados a objetos es la **transparencia**, (manipulación directa de datos utilizando un entorno de programación basado en objetos), por lo que el programador, solo se debe preocupar de los objetos de su aplicación, en lugar de cómo los debe almacenar y recuperar de un medio físico.

**Otras ventajas** de un sistema de bases de datos orientado a objetos son las siguientes:

* **Gran capacidad de modelado**. El modelado de datos orientado a objetos permite modelar el 'mundo real' de una manera óptima gracias al encapsulamiento y la herencia.
* **Flexibilidad**. Permiten una estructura cambiante con solo añadir subclases.
* **Soporte para el manejo de objetos complejos**. Manipula de forma rápida y ágil objetos complejos, ya que la estructura de la base de datos está dada por referencias (apuntadores lógicos) entre objetos.
* **Alta velocidad de procesamiento**. Como el resultado de las consultas son objetos, no hay que reensamblar los objetos cada vez que se accede a la base de objetos
* **Extensibilidad**. Se pueden construir nuevos tipos de datos a partir de los ya existentes, agrupar propiedades comunes de diversas clases e incluirlas en una superclase, lo que reduce la redundancia.
* **Mejora los costes de desarroll**o, ya que es posible la reutilización de código, una de las características de los lenguajes de programación orientados a objetos.
* **Facilitar el control de acceso y concurrencia**, puesto que se puede bloquear a ciertos objetos, incluso en una jerarquía completa de objetos.
* Funcionan de forma **eficiente en entornos cliente/servidor y arquitecturas distribuidas.**

Pero aunque los sistemas de bases de datos orientados a objetos pueden proporcionar soluciones apropiadas para muchos tipos de aplicaciones avanzadas de bases de datos, también tienen sus **desventajas**. Éstas son las siguientes:

* **Carencia de un modelo de datos universal**. No hay ningún modelo de datos aceptado universalmente, y la mayor parte de los modelos carecen de una base teórica.
* **Falta de estándares**. Existe una carencia de estándares general para los sistemas de BDOO.
* **Complejidad**. La estructura de una BDOO es más compleja y difícil de entender que la de una BDR.
* **Competencia de otros modelos**. Las bases de datos relacionales y objeto-relacionales están muy asentadas y extendidas, siendo un duro competidor.
* **Difícil optimización de consultas**. La optimización de consultas requiere una compresión de la implementación de los objetos, para poder acceder a la base de datos de manera eficiente. Sin embargo, esto compromete el concepto de encapsulación.

**Ejercicio 3.-** ¿Qué es el ODMG? ¿ Cuáles son los principales componentes de ODMG 3.0? (puedes responder en el fichero Word anterior).

**ODMG** es el acrónimo de ***Object Data Management Group***. Se usa tanto para definir el grupo de personas y empresas encargadas de desarrollar el modelo de objetos para [persistencia](https://es.wikipedia.org/wiki/Persistencia), así como para la definición de [dicho estándar](https://es.wikipedia.org/wiki/Object_Query_Language).

Este modelo especifica los elementos que se definirán, y en qué manera se hará, para la consecución de persistencia en las [bases de datos orientadas a objetos](https://es.wikipedia.org/wiki/Base_de_datos_orientada_a_objetos) que soporten el estándar.

Consta de un lenguaje de definición de objetos, ODL, que especifica los elementos de este modelo.

**Principales componentes de ODMG 3.0**

* ***Modelo de objeto*.** Esto se basó en el modelo de objetos del Grupo de gestión de objetos. El modelo central de OMG fue diseñado para ser un denominador común para los intermediarios de solicitudes de objetos, sistemas de bases de datos de objetos, lenguajes de programación de objetos, etc. El ODMG diseñó un perfil agregando componentes al modelo de objetos central de OMG.
* ***Idiomas de especificación de objetos.*** El lenguaje de definición de objetos ODMG (ODL) se utilizó para definir los tipos de objetos que se ajustan al modelo de objetos ODMG. El formato de intercambio de objetos ODMG (OIF) se utilizó para volcar y cargar el estado actual hacia o desde un archivo o conjunto de archivos.
* ***Lenguaje de consulta de objetos (OQL).*** El ODMG OQL era un lenguaje declarativo (sin procedimientos) para consultas y actualizaciones. Utilizaba SQL como base, cuando era posible, aunque OQL admite capacidades orientadas a objetos más potentes.
* ***Enlace de lenguaje C ++.*** Esto definió un enlace C ++ de ODMG ODL y un lenguaje de manipulación de objetos C ++ (OML). El ODL de C ++ se expresó como una biblioteca que proporciona clases y funciones para implementar los conceptos definidos en el Modelo de Objetos ODMG. La sintaxis y semántica de C ++ OML son las de C ++ estándar en el contexto de la biblioteca de clases estándar. El enlace de C ++ también proporcionó un mecanismo para invocar OQL.
* ***Enlace de lenguaje Smalltalk.*** Esto definió el mapeo entre ODMG ODL y Smalltalk, que se basó en el enlace OMG Smalltalk para el lenguaje de definición de interfaz OMG (IDL). El enlace Smalltalk también proporcionó un mecanismo para invocar OQL.
* ***Enlace de lenguaje Java***. Esto definió el enlace entre ODMG ODL y el lenguaje de programación Java como lo define la plataforma Java 2. El enlace de Java también proporcionó un mecanismo para invocar OQL.

**Ejercicio 4.-** A lo largo de esta unidad utilizaremos el SGBDOO (ODBMS) *ObjectDB* (<https://www.objectdb.com/>). Investiga otros 3 SGBDOO distintos indicando si siguen vigentes en la actualidad o corresponden a proyectos que están en desuso (puedes responder en el fichero Word anterior).

* **Neodatis:** Parece estar en desuso ya que no existen actualizaciones ni nuevas versiones a partir de 2010
* **ObjectStore:** Sistema gestor actualmente en uso y mantenido y comercializado por Ignitetech
* **GemStone:** Uno de los más antiguos y longevos que parece estar obsoleto desde 2012.

# SESIÓN 2

**Ejercicio 5 .-** Dedica un tiempo a probar la herramienta **ObjectDB Explorer** anterior.

**Ejercicio 6 .-** ¿Con qué otras dos interfaces trabajadas con la herramienta Hibernate relacionarías las interfaces EntityManagerFactory y EntityManager? (anota la respuesta en el fichero Word).

**SessionFactoryUtil y SessionFactory**

**Ejercicio 7.-** En los ejemplos anteriores se han utilizado en las consultas dos interfaces distintas, Query y TypedQuery. ¿En qué se diferencian? (averigualo anota la respuesta en el fichero Word).

La principal diferencia entre una query y un typedquery en JPA (Java Persistence API) es el tipo de resultado que se espera obtener.

Una query es una consulta genérica que se puede utilizar para recuperar cualquier tipo de resultado, ya sea una entidad, una lista de entidades, un valor escalar o una lista de valores escalares. Una query se crea utilizando el método createQuery() del EntityManager.

Por otro lado, un typedquery es una consulta específica que sólo se puede utilizar para recuperar un tipo de entidad específico. Un typedquery se crea utilizando el método createQuery() del EntityManager y especificando el tipo de entidad que se espera recuperar.

En términos de funcionalidad, ambos tipos de consultas son similares, pero el typedquery proporciona una mayor seguridad en tiempo de compilación ya que el compilador verifica que el tipo de entidad especificado en la consulta coincide con el tipo de entidad especificado en el método createQuery().

# SESIÓN 4

**Ejercicio 10.-** Investiga y averigua las cuestionas que se plantean a continuación. Puedes anotar las respuestas en un fichero Word. En la anotación de la relación de la ilustración siguiente (líneas 19 a 20):

* 1. ¿Qué significa el parámetro *cascade*?

El parámetro "cascade" es una anotación de JPA (Java Persistence API) que se utiliza para especificar cómo se deben propagar las operaciones de persistencia (guardar, actualizar, eliminar) desde una entidad principal a las entidades relacionadas.

La anotación @OneToMany, @ManyToOne, @ManyToMany y @OneToOne tienen un parámetro cascade que se utiliza para especificar qué operaciones deben propagarse. El valor predeterminado es CascadeType.PERSIST.

* 1. ¿Qué significa y qué otras posibilidades hay además de la opción *CascadeType.ALL*?

CascadeType.ALL es una anotación de JPA (Java Persistence API) que indica que todas las operaciones de persistencia (guardar, actualizar, eliminar) deben propagarse desde la entidad principal a las entidades relacionadas. Esto significa que si realizas un cambio en una entidad principal, automáticamente se realizarán los mismos cambios en las entidades relacionadas.

Otras opciones para CascadeType son:

* CascadeType.PERSIST: Propaga solo la operación de persistencia (guardar) a las entidades relacionadas.
* CascadeType.MERGE: Propaga solo la operación de actualización (merge) a las entidades relacionadas.
* CascadeType.REMOVE: Propaga solo la operación de eliminación a las entidades relacionadas.
* CascadeType.REFRESH: Propaga solo la operación de actualización (refresh) a las entidades relacionadas.
* CascadeType.DETACH: Propaga solo la operación de desanexión (detach) a las entidades relacionadas.
* CascadeType.NONE: No se propaga ninguna operación a las entidades relacionadas.

La opción CascadeType.ALL es la más utilizada ya que permite una mayor flexibilidad en las operaciones de persistencia, pero es importante tener en cuenta que su uso puede dar lugar a problemas de consistencia de datos si no se realiza adecuadamente.

* 1. ¿Qué significa la opción *orphanRemoval=true*?

OrphanRemoval=true es una anotación de JPA (Java Persistence API) que indica que cuando se elimina una entidad relacionada, esta entidad debe ser eliminada automáticamente de la base de datos. Esto se conoce como "eliminación de huérfanos".

Por ejemplo, si tienes una entidad "Padre" con una relación One-To-Many con la entidad "Hijo", y la anotación orphanRemoval=true está presente en la relación, cuando se elimina una instancia de "Padre" de la base de datos, todas las instancias de "Hijo" relacionadas con esa instancia de "Padre" serán eliminadas automáticamente.

Es importante tener en cuenta que esta anotación solo se aplica a relaciones One-To-Many y Many-To-Many.

* 1. ¿Qué significa y qué otras posibilidades hay además de la opción *fetch = FetchType.EAGER*?

FetchType.EAGER es una anotación de JPA (Java Persistence API) que indica que una relación de entidad debe cargarse automáticamente cuando se recupera la entidad principal. En otras palabras, cuando se recupera una entidad de la base de datos, también se recuperan automáticamente todas las entidades relacionadas con ella. Esto se conoce como "carga ansiosa" (eager loading).

Investiga y averigua las cuestionas que se plantean a continuación referidas a la siguiente ilustración. Puedes anotar las respuestas en un fichero Word.

1. ¿Qué significa la anotación *@JoinColumn* de la línea 34? ¿Es obligatoria?

La anotación @JoinColumn es una anotación de JPA (Java Persistence API) que se utiliza para especificar la columna en la tabla de la entidad principal que se utiliza para unirse a la tabla de la entidad relacionada. Esta anotación se utiliza junto con las anotaciones @OneToOne, @OneToMany y @ManyToOne para indicar la columna de clave foránea que se utiliza para establecer la relación entre las dos entidades.

@JoinColumn es obligatoria cuando se quiere especificar una columna de unión en la tabla de la entidad principal, si no se especifica, JPA utilizará una convención de nombres para generar el nombre de la columna de unión.

En caso de @ManyToMany, no es obligatoria ya que se crea una tabla intermedia para almacenar las relaciones, pero se puede utilizar para especificar columnas adicionales en esa tabla intermedia.

En resumen, es obligatoria solo cuando se quiere especificar una columna de unión en la tabla de la entidad principal en las relaciones OneToOne, OneToMany y ManyToOne, si no se quiere especificar una columna de unión no es obligatoria.

1. ¿Qué se ha creado en las líneas 16 a 19 anteriores? ¿Con qué finalidad? (pista: clase *MainNamedQueries.java*

del proyecto.

Las named queries son consultas pre-definidas en JPA (Java Persistence API) que se utilizan para recuperar entidades de la base de datos. Una named query se define mediante la anotación @NamedQuery o @NamedQueries y se puede ejecutar en cualquier momento utilizando el EntityManager.

La anotación @NamedQuery se utiliza para definir una sola consulta, mientras que @NamedQueries se utiliza para definir varias consultas. Ambas anotaciones se colocan en la entidad y tienen dos atributos: name y query. El atributo name es el nombre de la consulta, y el atributo query es la consulta en sí.

Las named queries son útiles porque permiten separar las consultas de la lógica de negocio, lo que hace que el código sea más legible y fácil de mantener. Además, al estar pre-definidas, son más seguras y tienen un mejor rendimiento ya que JPA las almacena en caché.