educación IT
CODE YOUR FUTURE

Bootcamp Java Developer

Fase 3 - Java Architect

Módulo 30

educación IT
CODE YOUR FUTURE

Persistencia de Objetos

¿Qué es la persistencia?

Definición

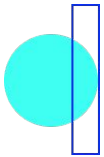
La **persistencia** es uno de los aspectos fundamentales en cualquier aplicación que trabaja con datos.

Representa el acto de persistir los datos en el tiempo.

Se utiliza normalmente para almacenar datos en una base de datos. En aplicaciones desarrolladas con el enfoque de Programación Orientada a Objetos, *persistencia* significa persistir el estado del objeto en el tiempo. Es decir, a la capacidad de un objeto de mantener su estado o datos incluso después de que la aplicación que lo creó haya finalizado.

Bases de datos relacionales

Es la forma por excelencia de **almacenar datos**. Son extremadamente flexibles y robustas para la administración de datos. Los DBMS (*DataBase Management Systems*) poseen interfaces basadas en SQL para interactuar con los datos.



SQL tiene como categorías a DDL y DML:

- **DDL (*Data Definition Language*):**

  Se utiliza para la creación de esquemas y tablas. Los comandos más conocidos son `CREATE`, `ALTER` y `DROP`.

- **DML (*Data Manipulation Language*):**

  Se utiliza para la manipulación de datos. Los comandos más conocidos son `SELECT`, `INSERT`, `UPDATE`, `DELETE`.

Archivos planos

Son **archivos de texto sin formato**, otra forma de persistencia. Si bien es posible guardar casi cualquier información, se hace relativamente difícil poder organizar y consultar de manera eficiente la información deseada.

En caso de objetos, se podrían persistir en archivos planos a través de serialización, y volver a utilizarlos a través de la deserialización.

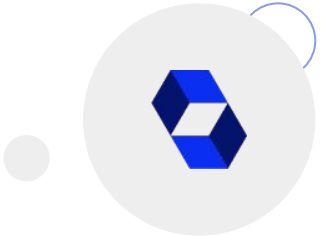
Modelo Relacional vs. Modelo Orientado a Objetos

Problemática

La problemática que se plantea es que los RDBMS (*Relational DBMS*) están basados en el modelo relacional, pero por su lado las aplicaciones orientadas a objetos están basadas en el paradigma de objetos.

Estos modelos tienen sus propias características, y como consecuencia se genera la problemática de tratar datos de una tabla como un objeto y viceversa, es decir, llevar datos de un objeto a una tabla.

Se produce lo que se conoce como un **"*gap*" entre ambos modelos, y se dificulta la integración y coordinación de ambos.**



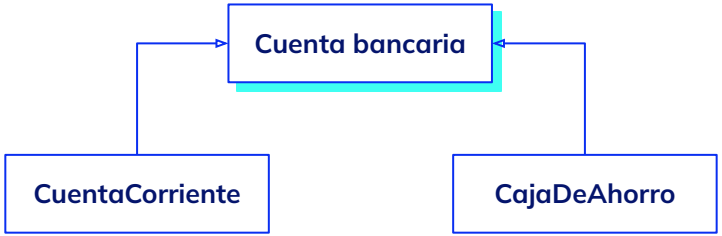
Una tabla, una clase

La forma más sencilla de relacionar ambos modelos es tratar a **cada clase en el modelo de objetos** como una tabla en el modelo relacional. Si bien es la forma más sencilla de trabajar, se presentan varios problemas a tratar debido a las diferencias entre el paradigma de objetos y el relacional. Veámoslos a continuación.

1. El problema de la herencia

Uno de los pilares del **paradigma OO** (orientado a objetos) es la herencia. **La herencia representa a la relación "*es un*".** Es muy común trabajar con diagramas de clases que poseen una gran jerarquía de clases a través de la herencia.

El modelo relacional está basado en la relación "*tiene un*", pero no cuenta con la relación "*es un*". La problemática se basa en que dentro del modelo relacional, no es posible armar una relación de herencia.



2. El problema de la identidad

Para realizar **chequeos de identidad en registros**, se utiliza la **PK (*Primary Key*)** como método de comparación.

Para realizar **chequeos de identidad en objetos**, se utilizan los métodos **`equals()`** y **`compareTo()`** según corresponda, como también el operador **`==`** dependiendo del caso.

La problemática surge cuando hay que realizar una actualización en cascada de la PK desde el modelo de objetos – el campo *nombre* - ya que el impacto es grande y genera procesamiento adicional que podría ser evitable.

Para evitar este tipo de problemática, la recomendación es utilizar una PK que nada tenga que ver con los datos, posiblemente un campo auto-numérico, y construir un atributo adicional ID en la clase correspondiente.

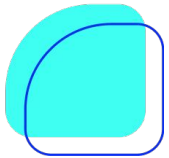
3. El problema de las asociaciones

En el paradigma de objetos, las asociaciones representan la forma de relacionar objetos.

Un objeto podría tener a otro objeto como asociado, por ejemplo un Auto tiene un Stereo, entonces es posible decir que la clase *Auto* tiene como atributo un objeto del tipo *Stereo* (obviamente, *Stereo* tiene sus propios atributos).

En el modelo relacional, dichas asociaciones se presentan a través de las claves foráneas.

Las asociaciones tienen dirección, es decir que cada clase que forma parte de la relación, tiene (o no) una referencia a la otra clase. Así, las relaciones pueden ser unidireccionales o bidireccionales. Veámoslo a continuación.



En una **relación unidireccional**, una de las dos clases tiene como referencia a la otra. Por ejemplo, el *Auto* tiene como referencia al *Stereo*, **o** podría ser al revés también, el Stereo tiene como referencia al Auto.

En una **relación bidireccional**, ambas clases se tienen como referencia. Por ejemplo, el *Auto* tiene como referencia al *Stereo* **y** el *Stereo* tiene como referencia al Auto.

Las claves foráneas no son en su concepción bidireccionales, lo que generaría inicialmente un desacople entre el modelo relacional y el modelo de objetos.

Un objeto podría tener un **conjunto** (una colección) de objetos asociados, por ejemplo un *Aula* tiene *Alumnos.* La clase *Aula* tiene como atributo un objeto del tipo `Set` que contiene objetos del tipo *Alumno* (obviamente, cada *Alumno* tiene sus propios atributos).

Es normal contar con relaciones del tipo ***muchos-a-muchos*****,** por ejemplo un *Aula* posee muchos alumnos, pero un *Alumno* puede tener asignada más de un *Aula.*

En el modelo de objetos, existen únicamente dos clases, pero en el modelo relacional surge una nueva tabla para manejar la relación *muchos-a-muchos*.

4. El problema de la navegación

La relación entre clases está dada a través de una **nueva clase**, o **colecciones**. Por ejemplo, la clase *Universidad* tiene *Facultades*, la *Facultad* tiene *Alumnos*, el *Alumno* tiene *Asignaturas*, y así sucesivamente.

El problema está en que al realizar la consulta hay que determinar hasta qué **nivel de información** (es decir hasta qué objetos) traer como datos.

El problema de la navegación es uno de los problemas que más impacta sobre la performance de la aplicación.

¡Sigamos trabajando!