Unidad 8

Las bases de datos objeto - relacionales

Bases de datos

Índice

Introducción

La programación orientada a objetos se crea en los años noventa como una alternativa a la metodología que se usaba hasta aquel entonces. Esto se usa también en las bases de datos, donde empiezan a surgir los primeros SGBDOO, sistemas gestores de bases de datos orientados a objetos. En estos sistemas la información se almacena en la base de datos de manera que luego sea sencillo usarse en el lenguaje de programación orientado a objetos. Estas bases de datos además presentaban menos complejidades que las creadas en el modelo relacional.

Hoy en día todavía no se usan mucho estas bases de datos, pero las vamos a ver porque sí que creemos que es un sistema en auge y que puede que sean las principales en un futuro no muy lejano.

Al finalizar esta unidad

- Sabremos cuales son los antecedentes históricos sobre la incidencia del paradigma de la orientación a objetos en la evolución del desarrollo de software.
- Conoceremos los elementos básicos de la orientación a objetos.
- Estaremos familiarizados con las características de las bases de datos objeto-relacionales.
- Seremos capaces de usar de manera adecuada los arrays y tipos complejos.
- Podremos trabajar con referencias.
- Conoceremos como aplicar el concepto de herencia correctamente.
- Seremos capaces de declarar y definir métodos

8.1. Las bases de datos objeto-relacionales

8.1.1. El paradigma de la orientación a objetos

El paradigma de POO supuso una revolución en el mundo de la programación al incluir novedosos conceptos que permitían generar código de más calidad, facilitar el desarrollo y reutilizar de una forma más práctica el código desarrollado.

De la misma forma que la POO partió de paradigmas de programación existentes, otros paradigmas han tomado la POO como punto de partida para ser desarrollados. Con la creciente demanda de aplicaciones que ofreciesen una interfaz gráfica cada vez más sofisticada, surgió la necesidad de adaptar la POO a un tipo de programación que permitiese controlar los eventos que se producían sobre las interfaces gráficas con las que el usuario interactuaba.

A continuación, serán descritos cada uno de los elementos y características de este extendido paradigma.

8.1.2. Características de las bases de datos objeto-relacionales

El repertorio objeto-relacional que nos ofrecen los distintos sistemas gestores de bases de datos ha cambiado mucho a lo largo de la historia, pero todos se siguen basando en el estándar SQL:1999. La teoría de la orientación a objetos nos dice que los modelos objeto-relacionales toman cada registro de una tabla como un objeto, y la definición de la tabla será la clase de dicho objeto.

La principal característica de las bases de datos objeto-relacionales se basa en la capacidad de gestionar tipos de datos complejos que se hayan definido previamente por el usuario, pero esto acarreará una serie de consecuencias graves para el modelo relacional subyacente:

- Los campos aceptan varios tipos de valores. Esto choca directamente con la primera forma normal ya que no podemos garantizar que todos los atributos vayan a ser atómicos.
- Al sufrir los datos cambios hasta poder representar una estructura, ya no tenemos a las tablas como elementos bidimensionales, pues se pueden incluir varias dimensiones para dotar de profundidad a la información.

La forma de almacenar la información que tienen las bases de datos objeto-relacionales se conoce como modelo relacional anidado ya que los valores de un campo pueden ser registro de otra tabla, y estos de otra, y así sucesivamente. Esta característica nos da la posibilidad de eliminar las tablas dedicadas a las relaciones y así poder simplificar las sentencias **SELECT**.

8.2. Tipos de datos objeto

8.2.1. Colecciones (arrays)

Datos de BD como arrays = atributos multivaluados

Los arrays en programación son un conjunto de elementos listados a los que accedemos indicando su posición en dicha lista. Como un array nos permite asignar varios valores a un mismo campo, pasamos a conocer a estos campos con arrays como atributos multivaluados.

Si quisiéramos implementarlo en SQL, dependiendo del SGBD que usemos sería de un modo o de otro. En MySQL se usan las concatenaciones, no existiendo un tipo array como tal.

8.2.2. Tipos complejos

La creación de un elemento tipo array requerirá muchas veces que se creen tipos nuevos como vimos en temas anteriores. Estos tipos pueden ser usados o bien para definir algunos campos de la tabla o bien para definir una tabla completa.

Un array también puede ser definido como tipo.

8.3. Referencias e identificadores

Cuando usamos un modelo objeto-relacional de la base de datos, nuestro fin no es únicamente la definición de tipos estructurados. Sino que también queremos que los valores de dichos tipos se almacenen como registros distintos en tablas independiente. Con esto conseguimos tres cosas:

- El valor de un campo no reside físicamente en la tabla a la que pertenezca.
- Al tener correspondencia con otro registro en otra tabla distinta, pueden ser usados los valores por varias tablas como si fuesen valores propios. Con esto eliminamos las redundancias.
- Alineado con este último punto, podemos dictaminar que se puede trabajar con valores de los campos de este registro directamente en su tabla de origen.

Si se usa este tipo de almacenamiento, las tablas guardan una referencia a la información almacenada en un campo. Esta información se almacena físicamente en otra tabla distinta.

Si pretendemos insertar datos, se haría de manera similar a con los tipos complejos. De otro modo, cuando vamos a realizar consultas, se tienen que usar un operador nuevo, llamado puntero (->. Con este lo que pretendemos es referenciar al valor que buscamos en la tabla principal.

Hay ocasiones en las que nos confundimos con la terminología normal de SQL y usamos un punto en vez de un puntero para especificar la información referenciada. Esto es un error que no dará ninguna información relevante.

8.4. Herencia

La herencia es una propiedad que solo poseen las clases que permite declarar nuevas clases a partir de otras clases declaradas con anterioridad. La clase original desde la que se hereda, se llama clase base, clase padre, clase ancestra o superclase, y la clase que hereda se llama clase extendida, clase hija, clase derivada o subclase.

Cuando una clase llamada "A" hereda de otra clase llamada "B", se transfieren desde la clase "A" a la "B" todos los miembros de esa clase, excepto destructores y constructores. La clase "B" puede añadir miembros de su propia clase (campos, métodos y propiedades) o puede sobrescribir los miembros que ha heredado.

El paradigma de la POO indica que la herencia puede ser múltiple, esto quiere decir que puede haber más de una clase padre de la cuál derive una clase. Pero son pocos los lenguajes de programación que utilizan herencia múltiple, como son C++ y Eiffel.

El concepto de visibilidad va unido al de herencia. Si bien, exceptuando constructores y destructores, todos los miembros son heredados, solo podrá accederse desde la clase hija a aquellos declarados como miembros públicos (+) o protegidos (#). Volveremos a este concepto más adelante.

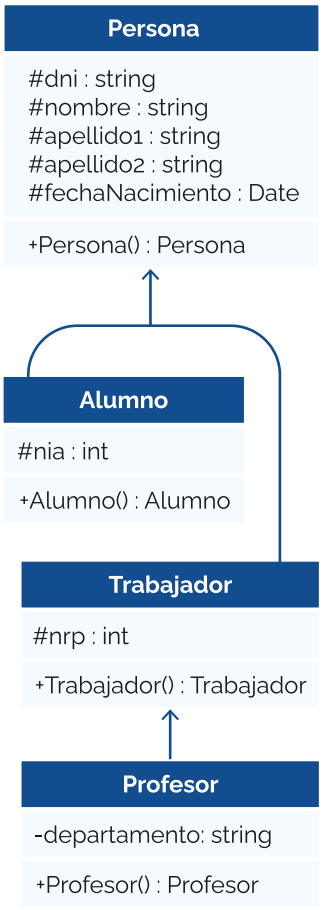

Imagen 1. Ejemplo de representación de la herencia.

8.4.1. De tipos

Si queremos que un subtipo herede todos los atributos y métodos de un supertipo debemos de usar la cláusula UNDER. Podemos al final de toda la declaración especificar la orden NOT FINAL para que se creen otros subtipos derivados de estos. Si por otra parte ponemos FINAL, no se pueden crear más subtipos.

8.4.2. De tablas

Cuando en las primeras unidades tratamos sobre conceptos de jerarquías en el modelo entidad relación, existía tanto la especialización como la generalización. Esto se puede implementar en la base de datos usando la herencia de tablas. En este caso, se heredan atributos y registros. La tabla padre posee tanto registros propios, como registros de sus tablas hijas, que son las que heredan.

8.5. Métodos

Cuando se realizar la programación orientada a objetos, cada bloque de código se ejecuta como un método asociado a un objeto de cada clase mientras que, en otros paradigmas de programación, se invocaban funciones y procedimientos independientes unos de otros. Cada uno de estos métodos, junto con los atributos, forman parte de la definición de una clase. Si, además, nos ponemos en el contexto objeto-relacional, los métodos también se asocian con la definición de los tipos de datos.

En la POO, los métodos trabajan sobre los atributos de cada uno de los registros. En ese tipo de programación, si nos queremos referir al registro sobre el que el método se encuentra trabajando, usaremos la palabra reservada **SELF**.

En los tipos estructurados, arrays o ROW, existen una función especial que sirve para que los atributos de un registro se inicialicen y así devolver ya creado, es la llamada función constructora. Esta función debe de nombrarse igual que el tipo, y como es implícita, no necesitamos codificarla salvo en caso de que se añadan restricciones o comprobaciones de carácter especial. Esta función se ejecuta de manera automática cada vez que creemos un nuevo registro.

Siguiendo con la idea de la encapsulación, las interfaces de todos los registros deben de ser manejadas y administradas únicamente con el uso de métodos. En estos métodos deberíamos de albergar las instrucciones de programación concretas para poder desarrollar las funciones necesarias.

En las bases de datos objeto-relacionales, el polimorfismo está admitido, lo que hace que un mismo método pueda ser usado para distintos tipos.

www.universae.com