UD5: BBDD orientadas a objetos y objeto-relacionales

Índice

- Introducción
- BBDD Orientadas o objetos
  - Características
  - Ventajas e inconvenientes
  - El estándar ODMG
    - El lenguaje de definición de datos ODL
    - El lenguaje de consultas OQL
- BBDD objeto-relacionales

Introducción

Las Bases de Datos Orientadas a Objetos (BDOO) son aquellas cuyo modelo de datos está orientado a objetos. Las BDOO simplifican la programación orientada a objetos (POO) almacenando directamente los objetos en la BD y empleando las mismas estructuras y relaciones que los lenguajes de POO.

Podemos decir que un Sistema Gestor de Base de Datos Orientada a Objetos (SGBDOO) es un sistema gestor de base de datos (SGBD) que almacena objetos.

Características de las BBDD Orientadas a Objetos

Las características asociadas a las BDOO son las siguientes:

- Los datos se almacenan como objetos.
- Cada objeto se identifica mediante un identificador único u OID (Object Identifier), este identificador no es modificable por el usuario.
- Cada objeto define sus métodos y atributos y la interfaz mediante la cual se puede acceder a él, el usuario puede especificar qué atributos y métodos pueden ser usados desde fuera.
- En definitiva, un SGBDOO debe cumplir las características de un SGBD: **persistencia, concurrencia, recuperación ante fallos, gestión del almacenamiento secundario y facilidad de consultas**; y las características de un sistema orientado a objetos (OO): **encapsulación, identidad, herencia y polimorfismo**.

Ventajas

- **Mayor capacidad de modelado**. La utilización de objetos permite representar de una forma más natural los datos que se necesitan guardar.
- **Extensibilidad**. Se pueden construir nuevos tipos de datos a partir de tipos existentes.
- Existe una **única interfaz** entre el LMD (lenguaje de manipulación de datos) y el lenguaje de programación. Esto elimina el tener que incrustar un lenguaje declarativo como SQL en un lenguaje imperativo como Java o C.
- **Lenguaje de consultas más expresivo**. El lenguaje de consultas es navegacional de un objeto al siguiente, en contraste con el lenguaje declarativo SQL.
- **Soporte a transacciones largas**, necesario para muchas aplicaciones de bases de datos avanzadas.
- **Adecuación a aplicaciones avanzadas de bases de datos** (CASE, CAD, sistemas multimedia, etc.).

Inconvenientes

- **Falta de un modelo de datos universal**, la mayoría de los modelos carecen de una base teórica.
- **Falta de experiencia**, el uso de los SGBDOO es todavía relativamente limitado.
- **Falta de estándares**, no existe un lenguaje de consultas estándar como SQL, aunque está el lenguaje OQL (Object Query Language) de ODMG que se está convirtiendo en un estándar de facto.
- **Competencia con los SGBDR y los SGBDOR**, que tienen gran experiencia de uso.
- **La optimización de consultas compromete la encapsulación**: optimizar consultas requiere conocer la implementación para acceder a la BD de una manera eficiente.
- **Complejidad**: el incremento de funcionalidad provisto por un SGBDOO lo hace más complejo que un SGBDR. La complejidad conlleva productos más caros y difíciles de usar.
- **Falta de soporte a las vistas**: la mayoría de SGBDOO no proveen mecanismos de vistas.
- **Falta de soporte a la seguridad**.

El estándar ODMG

ODMG (Object Datábase Management Group) es un grupo formado por fabricantes de bases de datos con el objetivo de definir estándares para los SGBDOO. Uno de sus estándares, (ODMG) especifica los elementos que se definirán, y en qué manera se hará, para la consecución de persistencia en las BDOO que soporten el estándar.

La última versión del estándar, ODMG 3.0 propone los siguientes componentes:

- Modelo de objetos.
- Lenguaje de definición de objetos (ODL, Object Definition Language).
- Lenguaje de consulta de objetos (OQL, Object Query Language).
- Conexión con los lenguajes C++, Smalltalk y Java.

El modelo de objetos ODMG especifica las características de los objetos, cómo se relacionan, cómo se identifican, construcciones soportadas, etc. Las primitivas de modelado básicas son: los objetos, caracterizados por un identificador único (OID-Object IDentifier) y los literales que son objetos que no tienen identificador, no pueden aparecer como objetos, están embebidos en ellos.

Los tipos de objetos son:

- **Atómicos**: boolean, short, long, unsigned long, unsigned short, float, double, char, string, enum, octect.
- **Tipos estructurados**: date, time, timestamp, interval.
- **Colecciones** <interfaceCollection>:
  - **set<tipo>**: grupo desordenado de objetos del mismo tipo que no admite duplicados.
  - **bag<tipo>**: grupo desordenado de objetos del mismo tipo que permite duplicados.
  - **list<tipo>**: grupo ordenado de objetos del mismo tipo que permite duplicados.
  - **array<tipo>**: grupo ordenado de objetos del mismo tipo a los que se puede acceder por su posición. El tamaño es dinámico.
  - **dictionary<clave,valor>**: grupo de objetos del mismo tipo, cada valor está asociado a su clave.

Los literales pueden ser atómicos, colecciones, estructuras y NULL.

Mediante las Clases especificamos el estado y el comportamiento de un tipo de objeto, pueden incluir métodos. Son equivalentes a una clase concreta en los lenguajes de programación. Una clase es un tipo de objetos asociado a un "extent".

ODL (Object Definition Languaje)

El lenguaje ODL es el equivalente al lenguaje de definición de datos (DDL) de los SGBD tradicionales. La sintaxis de ODL extiende el lenguaje de definición de interfaces de CORBA (Common Object Request Broker Architecture). Algunas de las palabras reservadas para definir los objetos son:

- *class*: declaración del objeto, define el comportamiento y el estado de un tipo de objeto.
- *extent*: define la extensión, nombre para el actual conjunto de objetos de la clase. En las consultas se hace referencia al nombre definido en esta cláusula, no se hace referencia al nombre definido a la derecha de class.
- *key[s]*: declara la lista de claves para identificar las instancias.
- *attribute*: declara un atributo.
- *set* | *list* | *bag* | *array*: declara un tipo de colección.
- *struct*: declara un tipo estructurado.
- *enum*: declara un tipo enumerado.
- *relationship*: declara una relación.
- *inverse*: declara una relación inversa.
- *extends*: define la herencia simple.

Ejemplo

```
class Cliente (extent Clientes key NIF) {
/*Definición de atributos*/
    attribute struct Nombre_Persona {
        string apellidos,
        string nombrepern
    } nombre;
    attribute string NIF;
    attribute date fecha_nac;
    attribute enum Genero{H,M} sexo;
    attribute struct Dirección{
        string calle,
        string poblac
    } direc;
    attribute set<string> telefonos;
/*Definición de operaciones*/
    short edad(); }
```

```
class Producto (extent Productos key IDPRODUCTO)  {
/*Definición de atributos*/
    attribute short IDPRODUCTO;
    attribute string descripcion;
    attribute float pvp;
    attribute short stockactual;  }
```

```
class LineaVenta (extent Lineaventas)  {
/*Definición de atributos*/
    attribute short numerolinea;
    attribute Producto product;
    attribute short cantidad;
/*Definición de operaciones*/
    float importe (); }
```

Ejemplo

```
class Venta (extent Ventas key IDVENTA)  {
/*Definición de atributos*/
    attribute short IDVENTA;
    attribute date fecha_venta;
    attribute set <LineaVenta> lineas;

/*Definición de relaciones*/
    relationship Cliente pertenece_a_cliente inverse Cliente::tiene_venta;

/*Definición de operaciones*/
    float total_venta();
}
```

OQL (Object Query Languaje)

OQL es el lenguaje estándar de consultas de BDOO. Las características son las siguientes:

- Es orientado a objetos y está basado en el modelo de objetos de la ODMG.
- Es un lenguaje declarativo del tipo de SQL. Su sintaxis básica es similar a SQL.
- Acceso declarativo a los objetos de la base de datos (propiedades y métodos).
- Semántica formal bien definida.
- No incluye operaciones de actualización (solo de consulta). Las modificaciones se realizan mediante los métodos que los objetos poseen.
- Dispone de operadores sobre colecciones (max, min, count, etc.) y cuantificadores (for all, exists).

La sintaxis básica de OQL es una estructura SELECT como en SQL:

SELECT <lista de valores>
FROM <lista de colecciones y miembros tipicos>
    [WHERE <condición>]

Donde las colecciones en FROM pueden ser extensiones (los nombres que aparecen a la derecha de extent) o expresiones que evalúan una colección. Se suele utilizar una variable iterador que vaya tomando valores de los objetos de la colección. Las variables iterador se pueden especificar de varias formas utilizando las cláusulas IN o AS:

```
FROM Clientes x
FROM x IN Clientes
FROM Clientes AS x
```

Para acceder a los atributos y objetos relacionados se utilizan expresiones de camino. Una expresión de camino empieza normalmente con un nombre de objeto o una variable iterador seguida de atributos conectados mediante un punto o nombres de relaciones. Por ejemplo, para obtener el nombre de los clientes que son mujeres, podemos escribir:

```
SELECT x.nombre.nombreper FROM x IN Clientes WHERE x.sexo="M"
SELECT x.nombre.nombreper FROM Clientes x WHERE x.sexo="M"
SELECT x.nombre.nombreper FROM Clientes AS x WHERE x.sexo="M"
```

En general, supongamos que v es una variable cuyo tipo es Venta:

- v.IDVENTA es el identificador de venta del objeto v.
- v.fecha_venta es la fecha de venta del objeto v.
- v. total_venta() obtiene el total venta del objeto v.
- v.pertenece_a_cliente es un puntero al cliente mencionado en v.
- v.pertenece_a_cliente.direc es la dirección del cliente mencionado en v.
- v.lineas es una colección de objetos del tipo LineaVenta.

- El uso de v.lineas.numerolinea NO es correcto porque v.lineas es una colección de objetos y no un objeto simple.
- Cuando tenemos una colección como en v.lineas, para poder acceder a los atributos de la colección podemos usar la orden FROM.

Ejemplos

Obtener los datos del cliente cuyo IDVENTA = 1:

```
SELECT v.pertenece_a_cliente.nombre, v.pertenece_a_cliente.direc, v.fecha_venta, v.total_venta()
FROM Ventas v
WHERE v.IDVENTA=1;
```

Obtenemos las líneas de venta del IDVENTA = 1, en este ejemplo el objeto v es usado para definir la segunda colección v.lineas:

```
SELECT lin.numerolinea, lin.product.descripcion, lin.cantidad, lin.importe()
FROM Ventas v, v.lineas lin
WHERE v.IDVENTA=1;
```

El resultado de una consulta OQL puede ser de cualquier tipo soportado por el modelo. Por ejemplo, la consulta anterior devuelve un conjunto de estructuras del tipo short, string, short y float; el resultado es del tipo colección: bag (struct(numerolinea:short, descripcion:string, cantidad: short, importe: float)).

En cambio la consulta: **SELECT x.nombre.nombreper FROM x IN Clientes WHERE x.sexo="M"**, devuelve un conjunto de nombres; el tipo devuelto es: bag(string).

Ejemplos de BBDDOO

Existen diferentes productos de BBDD Orientadas a Objetos

Lenguaje Java

- DB4O: BBDD embebida
- NeoDatis

Lenguaje Python

- ZODB (Zope.org)

BBDD Objeto-relacionales (BDOR)

Las Bases de Datos Objeto-Relaciónales (BDOR) **son una extensión de las bases de datos relaciónales tradicionales** a las que se les ha **añadido conceptos del modelo orientado a objetos**, por tanto un Sistema de Gestión de Base de Datos Objeto-Relacional (SGBDOR) **contiene características del modelo relacional y del orientado a objetos**; es decir, es un sistema relacional que **permite almacenar objetos en las tablas**.

Los fabricantes de SGBD relaciónales han ido incorporando en las nuevas versiones muchas de las propuestas para las bases de datos orientadas a objetos, un ejemplo son **Informix**, **Oracle** o **PostgreSQL**.

Características

Las características más importantes de los SGBDOR son las siguientes:

- Soporte de tipos de datos básicos y complejos. El usuario puede crear sus propios tipos de datos.
- Soporte para crear métodos para esos tipos de datos.
- Gestión de tipos de datos complejos con un esfuerzo mínimo.
- Herencia.
- Se pueden almacenar múltiples valores en una columna de una misma fila.
- Relaciones (tablas) anidadas.
- Compatibilidad con las bases de datos relaciónales tradicionales.
- El **inconveniente** de las BDOR es que **aumenta la complejidad del sistema**, esto ocasiona un aumento del coste asociado.