TRABAJO PRÁCTICO GRUPAL
BASES DE DATOS ORIENTADAS A OBJETOS

Mateo Holzer
Matías Meneghetti

13 de Abril de 2023
Facultad de Ingeniería

Descripción de la arquitectura, las características y sus funcionalidades

Una base de datos orientada a objetos (OODB, por sus siglas en inglés) es un tipo de sistema de gestión de bases de datos que utiliza el paradigma de la programación orientada a objetos (POO) para modelar y almacenar datos. Los datos se modelan como objetos. Los objetos a su vez se dividen en clases, creando así una jerarquía de clases y subclases, en la que las subclases heredan las propiedades las clases superiores y las complementan con sus propios atributos

El sistema de gestión de base de datos orientadas a objetos asigna de forma automática un código de identificación único a cada registro (que será inmutable), permitiendo así recuperar los objetos cuando estos se han guardado y realizar consultas.

La estructura de una base de datos orientada a objetos se basa en encapsular los datos y el código relacionado con cada objeto en una sola unidad. Las interacciones entre los objetos y el resto del sistema se realizan mediante una interfaz.

La interfaz de usuario es la forma en que los usuarios interactúan con la base de datos. Los usuarios pueden crear, modificar y eliminar objetos a través de una interfaz de programación de aplicaciones (API) o una interfaz gráfica de usuario (GUI).

La estructura de estas BDOO se diseña a partir de una serie de diagramas con los que se establecen las clases y sus relaciones, las interacciones entre los objetos y su comportamiento:

Así, a través de un diagrama de clases se presentan las clases con sus respectivas relaciones estructurales y de herencia, que se puede acompañar de un diagrama de objetos cuando no está muy claro cómo serán las instancias de las clases.

Se emplea un diagrama de secuencias para presentar las interacciones entre los objetos organizados en una secuencia temporal y describir cómo colaboran.

A continuación se presentan algunas de las características más importantes de las OODB:

1. Modelado de datos basado en objetos: En una OODB, los datos se modelan y almacenan en forma de objetos, que pueden tener atributos y métodos.

2. Herencia: Las OODB admiten la herencia de clases, lo que permite que las clases hereden atributos y métodos de clases superiores. Esto puede simplificar el diseño de la base de datos y reducir la cantidad de código necesario para implementar ciertas funcionalidades.
3. Capacidad de almacenar objetos complejos: Las OODB pueden almacenar objetos complejos, como imágenes, archivos de audio y video, y otros tipos de datos multimedia. En las bases de datos relacionales, estos datos a menudo se almacenan como referencias a archivos externos.
4. Polimorfismo: admiten polimorfismo, lo que significa que los objetos pueden tener diferentes formas o comportamientos, dependiendo del contexto en el que se usen.
5. Encapsulamiento: admiten el encapsulamiento, lo que significa que los datos y los métodos de un objeto están ocultos del mundo exterior. Esto puede mejorar la seguridad e integridad de los datos almacenados en la base de datos.
6. Triggers y eventos: admiten triggers y eventos, lo que significa que se pueden programar acciones que se desencadenan automáticamente cuando se produce un evento específico, como la inserción o eliminación de datos.
7. Concurrencia: pueden manejar múltiples usuarios que acceden y modifican la base de datos al mismo tiempo, lo que las hace ideales para aplicaciones de alto tráfico y sistemas de tiempo real.

Ventajas y desventajas

Ventajas

1. - Mayor capacidad de modelado:
2. Un objeto permite encapsular tanto un estado como un comportamiento.
3. Un objeto puede almacenar todas las relaciones que tenga con otros objetos. Los objetos pueden agruparse para formar objetos complejos (herencia).
4.
5. - Ampliabilidad: Se pueden construir nuevos tipos de datos a partir de los ya existentes
6. Agrupar propiedades comunes de diversas clases e incluirlas en una superclase, lo que reduce la redundancia.
7.
8. - Reusabilidad de clases, lo que repercute en una mayor facilidad de mantenimiento y un menor tiempo de desarrollo.
9.

10. - Lenguaje de consulta más expresivo.
11. El acceso navegacional desde un objeto al siguiente es la forma más común de acceso a datos en un SGBDOO. Mientras que SQL utiliza el acceso asociativo.
12. El acceso navegacional es más adecuado para gestionar operaciones como los despieces, consultas recursivas, etc.
13.
14. - Adecuación a las aplicaciones avanzadas de base de datos. Hay muchas áreas en las que los SGBD tradicionales no han tenido excesivo éxito como el CAD, CASE, OIS, sistemas multimedia, etc. en los que las capacidades de modelado de los SGBDOO han hecho que esos sistemas sí resulten efectivos para este tipo de aplicaciones.
15.
16. - Mayores prestaciones. Los SGBDOO proporcionan mejoras significativas de rendimiento con respecto a los SGBD relacionales.

Desventajas

- Carencia de un modelo de datos universal. No hay ningún modelo de datos que esté universalmente aceptado para los SGBDOO y la mayoría de los modelos carecen de una base teórica.

- Carencia de experiencia. Todavía no se dispone del nivel de experiencia del que se dispone para los sistemas tradicionales.

- Carencia de estándares. Existe una carencia de estándares generales para los SGBDOO.

- Competencia. Con respecto a los SGBDR, estos productos tienen una experiencia de uso considerable. SQL es un estándar aprobado y ODBC es un estándar de facto. Además, el modelo relacional tiene una sólida base teórica y los productos relacionales disponen de muchas herramientas de soporte que sirven tanto para desarrolladores como para usuarios finales.

- La optimización de consultas compromete la encapsulación. 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.

- El modelo de objetos aún no tiene una teoría matemática coherente que le sirva de base.

Cuándo conviene usar la base de datos

Conviene usar OODB cuando:

1. Datos complejos: Si la aplicación requiere el manejo de datos complejos como objetos, imágenes, sonidos y otros tipos de datos multimedia, entonces OODB puede ser una mejor opción que las bases de datos relacionales.
2. Manipulación directa de objetos de software: Si la aplicación está escrita en un lenguaje de programación orientado a objetos, entonces es posible que sea más fácil y natural manejar los objetos de software directamente usando una OODB.

No conviene usar OODB cuando:

1. Requerimientos simples: Si los requerimientos de la aplicación son simples y no requieren el manejo de datos complejos, una base de datos relacional puede ser más adecuada.
2. Experiencia en OODB: Si el equipo de desarrollo no tiene experiencia en el uso de OODB, puede resultar difícil y costoso aprender y aplicar correctamente las técnicas de OODB.
3. Herencia de clases: Si la aplicación no requiere la herencia de clases, una base de datos relacional puede ser una opción más adecuada, ya que no se está aprovechando una de las principales ventajas de OODB.

Comparación con base de datos relacional

Las bases de datos orientadas a objetos (OODB) y las bases de datos relacionales son dos enfoques diferentes para almacenar y gestionar datos. A continuación se presentan algunas diferencias clave entre ambas:

1. Modelo de datos: Las bases de datos relacionales utilizan un modelo de datos basado en tablas, donde cada tabla tiene filas y columnas que representan los datos almacenados. Las OODB utilizan un modelo de datos basado en objetos, donde los datos se representan como objetos con atributos y métodos.
2. Escalabilidad y rendimiento: Las OODB pueden ofrecer una mayor escalabilidad y rendimiento que las bases de datos relacionales, especialmente en aplicaciones de alta demanda y tiempo real

3. Consultas: Las bases de datos relacionales utilizan el lenguaje SQL para realizar consultas, mientras que las OODB utilizan lenguajes de programación orientados a objetos, como Java o C++. Esto puede hacer que las consultas en las OODB sean más naturales y fáciles de entender para los programadores.
4. Integridad de los datos: Las bases de datos relacionales tienen un sistema de integridad de datos incorporado que garantiza que los datos sean coherentes y precisos. Las OODB no tienen un sistema de integridad de datos tan completo, lo que puede requerir más trabajo por parte del desarrollador para garantizar la integridad de los datos.
5. Soporte para tipos de datos complejos: Las OODB pueden manejar tipos de datos complejos, como objetos geométricos, imágenes, sonidos y otros tipos de datos multimedia. Las bases de datos relacionales pueden manejar estos tipos de datos, pero a menudo requieren una estructura de tabla compleja o la utilización de varios tipos de datos diferentes.

Comparación con base de datos NoSQL

Las bases de datos orientadas a objetos (OODB) y las bases de datos NoSQL son dos enfoques diferentes para almacenar y gestionar datos. A continuación se presentan algunas diferencias clave entre ambas:

1. Modelo de datos:Las bases de datos NoSQL, por otro lado, pueden tener varios modelos de datos, como documentos, grafos, clave-valor y columnas.
2. Escalabilidad: Las bases de datos NoSQL están diseñadas para escalar horizontalmente, lo que significa que se pueden agregar más nodos para manejar un mayor volumen de datos y tráfico. Las bases de datos orientadas a objetos no tienen esta capacidad de escalabilidad horizontal nativa.
3. Consultas: Las bases de datos NoSQL utilizan lenguajes específicos para el modelo de datos que se esté utilizando, como MongoDB Query Language para bases de datos de documentos.
4. Integridad de los datos: Las bases de datos orientadas a objetos pueden tener una integridad de los datos más sólida debido a la naturaleza estricta del modelo de programación orientado a objetos. Las bases de datos NoSQL pueden tener menos integridad de los datos, pero pueden compensarlo con la escalabilidad y la flexibilidad.
5. Flexibilidad: Las bases de datos NoSQL son más flexibles que las OODB, ya que pueden manejar una amplia variedad de modelos de datos y pueden adaptarse a las necesidades de la aplicación. Las OODB, por otro lado, están diseñadas específicamente para el modelo de programación orientado a objetos y son menos flexibles en cuanto a la estructura de los datos.

Comparación con base de datos embebidas

1. Modelo de datos: Las bases de datos embebidas suelen tener un modelo de datos más simple, como tablas y columnas en una estructura de base de datos relacional.
2. Escalabilidad: Las bases de datos embebidas son limitadas en escalabilidad debido a su naturaleza embebida. Están diseñadas para aplicaciones que no requieren grandes cantidades de datos y no necesitan un alto nivel de escalabilidad.
3. Consultas: Las bases de datos embebidas son más adecuadas para aplicaciones que tienen requisitos de consulta más simples y predecibles. Debido a su modelo de datos más simple, estas bases de datos pueden manejar mejor las consultas simples y directas que se realizan en aplicaciones de baja complejidad.
4. Distribución de datos: Las bases de datos embebidas suelen ser monolíticas, lo que significa que todos los datos se almacenan en una sola ubicación física, generalmente dentro de la aplicación que la utiliza. En las OODB los datos se almacenan en múltiples servidores en diferentes ubicaciones geográficas.

Gestores y motores

- GemStone/S
  Como la mayoría de otras bases, tiene su versión gratuita, por cierta cantidad de días, y luego cuenta con versiones pagas y con sus respectivos beneficios
- ObjectDB
  Base de datos para Java, se puede utilizar de forma gratuita con la prueba de 30 días, y luego tiene dos versiones pagas, una para el uso más personal, con un valor de 500 euros, y una versión más utilizada para empresas, con un valor de 2500 euros
- DB4O
  Es una BDOO para Java y .NET, se puede utilizar, modificar y distribuir libremente, siempre y cuando se respeten los términos de la licencia.
- Velocity DB
  Al igual que objectDB, se puede utilizar la prueba gratuita por 30 días, y luego cuenta con 3 planes distintos, el más básico tiene un valor de 400 dólares por mes, el intermedio 600 y el más completo, 3000
- ZODB
  Base de datos orientada a objetos para Python, con licencia de software libre.

Historia y antigüedad

A inicios de la década de los años 60 se gesta el origen de este tipo de bases de datos. Ocurre en Noruega, donde el doctor Nygaard, especialista en la elaboración de sistemas informáticos que llevaban a cabo simulaciones de sistemas de naturaleza física, puso a punto esta nueva tendencia. Acostumbrado a buscar la representación informática del rendimiento de elementos físicos como el motor de un coche, Nygaard tenía ciertas necesidades que hasta ese momento nadie podía satisfacer. En su época, todavía los sesenta, había ciertos obstáculos que impedían que su trabajo pudiera llegar a buen puerto. Se encontraba con que el software del momento era excesivamente complicado y debía recurrir a modificaciones en cada una de sus operaciones. El proceso de trabajo superaba los límites de la comodidad, dado que por cada iniciativa que llevaba a cabo tenía que contemplar varias alteraciones y posibilidades que cubrían las alternativas a sus planteamientos. Todo ello hacía que el trabajo fuera demasiado absorbente y que se saliera de los límites de lo tolerable.
Esta situación hizo que el doctor Nygaard y su equipo planteasen alternativas. Así es como trabajaron en la elaboración de un software que se diseñase de forma paralela al objeto físico en cuestión. Esto suponía cambiar la forma de trabajar, pero con el objetivo de alcanzar un resultado mucho más adecuado y beneficioso. El equipo del doctor planteó un requisito clave: que el programa fuera un reflejo del objeto. Si el objeto físico en cuestión estaba formado por 50 componentes principales, el software también debería tener 50 módulos. Esto garantizaría que por cada pieza del objeto habría una contraposición en el software, todo ello visible y analizable. Y si el objeto en cuestión funcionaba bien gracias a la comunicación de sus 50 piezas, el software también lo haría a través de la comunicación de los 50 módulos y de sus componentes.
¿Qué ganó Nygaard? Se encontró con que su solución le resolvía todos los problemas. Disponía de un mantenimiento absolutamente dominado, sin sorpresas, pero al mismo tiempo también veía que ahora sus programas se podían dividir de forma lógica. De esta forma, con los posteriores análisis y pruebas el doctor podría cambiar las piezas de forma independiente y cómoda, así como hacerlo con facilidad. Así es como comenzó a realizar pruebas más fiables, específicas y de resultados precisos.

Y entonces se descubrió su potencial. Uno de los principales problemas a los que se enfrentaba Nygaard en su época se encontraba en que debía realizar procesos de trabajo largos para una tarea posterior que no simbolizaba tanta dificultad. No era tirar horas de trabajo a la basura, pero sí sentía que su tiempo se estaba desperdiciando dentro del contexto. Por eso planteó soluciones y descubrió que finalmente la orientación a objetos con la que había trabajado le resolvía este aspecto. El motivo se encontró en que pudo introducir la reusabilidad en la creación

del software. A partir de este momento cada programa no representaba un inicio y final para el software, sino que en el proceso de creación muchos elementos podían recuperarse y mantenerse en activo para futuras creaciones.

Pero con este descubrimiento Nygaard necesitaba un lenguaje que le sirviera de apoyo. Así fue como nació Simula-67, lenguaje que hoy día aún se usa y que está considerado como uno de los inventos más importantes en el mundo de la informática aunque no sea algo conocido en términos populares.
Con el paso del tiempo la industria continuó el trabajo en las bases de datos orientada a objetos. Una década después el equipo de Alan Kay y Xerox tomó de referencia el trabajo de Simula-67 para crear otro lenguaje similar: Smalltalk. En los años 80 este lenguaje fue utilizado en combinación con algunas de las ideas que habían sido más características de Simula-67 para dar forma a otro lenguaje que tiene gran importancia hoy día: C++, que se ocupó de que este tipo de trabajo alcanzará sus máximos niveles de rendimiento.

Principales empresas donde funciona:

- IBM
- Microsoft
- Oracle
- Informix
- SAP
- Progress software

Proveedores que facilitan el producto:

- Objectivity
- InterSystems
- Versant
- Starcounter
- ObjectBox