db4o | La Base de Objetos de Código Abierto | Java y .NET

Bases de Objetos

por German Viscuso

German Viscuso es CIO de la empresa de software argentina Cyber Quality Control SRL (db4objects partner) y es Licenciado en Sistemas de Computación. Comenzó su carrera como desarrollador freelance y luego incursionó también en otras áreas informáticas como software QA, seguridad informática y técnicas de encriptación, simuladores 3d, técnicas de machine learning y sistemas inteligentes entre otras. Su pasatiempo consiste principalmente en el desarrollo de software de control inteligente para agentes y robots móviles.

A pesar de que las bases de datos relacionales lideran el mercado, las bases de objetos[1] ganan cada vez más aceptación. Las bases de datos relacionales son ideales para aplicaciones tradicionales que soportan tareas administrativas. Sin embargo, como resultado de avances recientes en hardware y software, han surgido aplicaciones más sofisticadas. Aplicaciones de ingeniería (como las de CAD/CAM), aplicaciones CASE, sistemas multimedia, sistemas de información geográfica (GIS) y médica, aplicaciones 3D y sistemas inteligentes, aplicaciones de bioinformática, telecomunicaciones y robótica, entre otras, pueden caracterizarse por estar compuestas de objetos complejos relacionados entre si también de forma compleja. La representación de tales objetos y relaciones en un modelo relacional implica que los mismos deben ser descompuestos en una gran cantidad de tuplas. Luego, como las tablas están anidadas profundamente, se requieren muchas juntas (*joins)* para recuperar un objeto lo que disminuye dramáticamente el desempeño. Las bases de objetos son ideales para almacenar y recuperar datos complejos permitiendo a los usuarios su navegación directa (sin un mapeo entre distintas representaciones).

Otro problema de importancia en los sistemas de bases de datos tradicionales es que normalmente existe un desfasaje entre las construcciones típicas provistas por los modelos de datos y las provistas por los ambientes de programación basados en objetos (denominado en ocasiones como un “desfasaje de impedancia” entre el modelo relacional y el de objetos). La evolución tecnológica en los ambientes de programación intenta subsanar este problema sin abandonar las bases de datos relacionales bien volcándose hacia el lado de los elementos del ambiente de programación con los *frameworks* de mapeo objeto-relacional o bien haciéndolo hacia el lado de los elementos de la base de datos proveyendo estructuras de datos con equivalencia directa en la base (ej. *datasets*) y permitiendo la ejecución de procesos de usuario en la base (ej. *stored procedures*). Sin embargo aun se intentan reconciliar dos paradigmas completamente distintos: el relacional y el de objetos. Cada vez que los objetos deben almacenarse en una base de datos relacional se requiere un mapeo desde las estructuras provistas por el ambiente de programación a las estructuras del modelo de datos ya que las bases de datos relacionales comprenden un solo ‘lenguaje’ primitivo (el de tablas). Para almacenar un objeto, todos sus datos deben ser normalizados previamente, es decir, deben ser reducidos a primitivas y ordenados por tipo. A continuación los elementos primitivos del objeto se almacenan en tablas distintas. Al final, para recuperar el objeto es necesario reensamblarlo.

[1] A una base de este tipo también se la llama base orientada a objetos y se la abrevia con siglas como OODBMS, ODBMS u ODB. Como todas las bases de datos cumplen con los criterios ‘ACID’

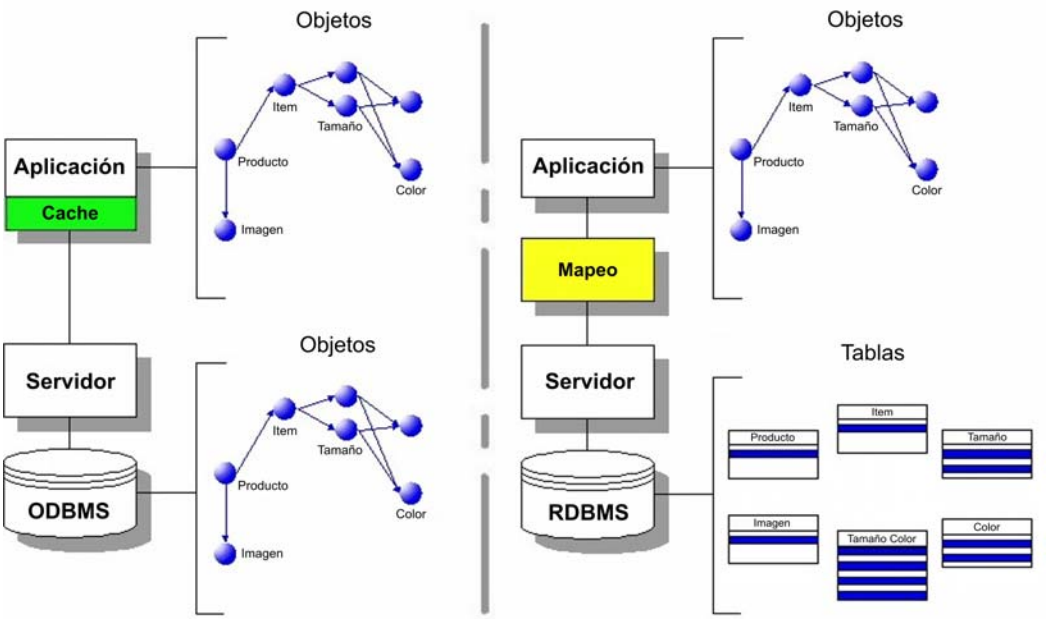

Figura 1 – Almacenamiento directo vs. mapeo de objetos

Las tecnologías objeto-relacionales existentes en la actualidad, aunque han ganado un lugar importante en el manejo de datos en aplicaciones OO, no resuelven el problema del desfasaje de ambos mundos completamente por lo que no constituyen la mejor alternativa (ofrecen transparencia en varios casos pero aun existe un mapeo de objetos subyacente que es costoso y poco flexible cuando los objetos y sus interacciones son complejos).

Las bases de objetos resuelven los problemas mencionados soportando de forma óptima la persistencia de objetos complejos e integrando la tecnología de bases de datos con el paradigma de objetos. Tanto las bases de objetos como los entornos de programación basados en objetos utilizan un mismo modelo eliminando el problema del "desfasaje de impedancia".

La Tecnología Relacional

A medida que los usuarios se expanden en la utilización de nuevos tipos de aplicaciones y amplían las existentes sus intentos por utilizar RDBMSs[2] encuentran una barrera donde no alcanzan los requerimientos de desempeño y funcionalidad de la base de datos. Esta barrera aparece cuando se extienden los modelos de información para soportar interrelaciones, tipos de datos extensibles, nuevos tipos de datos y soporte directo de objetos lo que excede claramente un procesamiento de información tabular (ver cuadro 2). De forma similar, la barrera aparece también en ambientes distribuidos con operaciones complejas. Todo intento de resolver el problema con tecnología relacional se traduce en una explosión de tablas, juntas, disminución del desempeño, baja escalabilidad y pérdida de integridad.

[2] Relational Database Management Systems: Sistemas de Administración de Bases de Datos Relacionales

No se trata de una falla en la tecnología relacional. De hecho, las bases de datos relacionales trajeron muchos beneficios a los usuarios como consultas por demanda, independencia de datos de la aplicación, estandarización en el manejo de datos, una muy buena variedad de *front-ends* de interfaz de usuario, y varios más. Este fenómeno es el resultado de aplicar a un problema la herramienta equivocada (en este caso los RDBMSs). No es aconsejable utilizar una base de datos basada en tablas para datos no tabulares, o un servidor central para aplicaciones distribuidas, o una base centralizada (y por ello poco escalable) para grandes aplicaciones que deben ser escalables.

Cuadro 2

Este cuadro indica las características principales de las aplicaciones para las que se construyeron los RDBMS (típicas de los 70's). Cualquier desvío sustancial de estas características presenta una serie de problemas para la tecnología.
• Datos relativamente estáticos: por ejemplo esquemas para productos y clientes no se crean en grandes cantidades cada día. Una RDBMS requiere de cierta estabilidad en los esquemas de datos para operar eficientemente. • Datos relativamente simples: los datos se acomodan bien en tablas y no hay muchas relaciones de N a N o recursivas como árboles o estructuras en red. Para datos binarios no estructurados se utilizan BLOBs. La base no comprende como se estructuran los datos binarios (solo comprende el lenguaje primitivo de tablas). • Cardinalidades manejables: "cientos" de productos, "miles" de clientes, en general que no superen los cientos de gigabytes en total. • Consulta por atributo: la recuperación de los datos se hace en base a valores en lugar de tratarse del seguimiento de relaciones directas de un objeto (o fila) a otro. Las relaciones se representan con valores especiales (claves primarias y foráneas), pero hay pocas consultas profundas que requieren atravesar muchas relaciones. • Transacciones pequeñas y cortas: Cada transacción altera pocos datos (id de clientes, pagos y fechas, etc.) y es llevada a cabo rápidamente. • Procesamiento centralizado: actualizaciones y consultas se hacen a través de un servidor central potente (ya sea *on-line* o *batch*). • Altamente administrado: Se requieren administradores de bases de datos detrás del RDBMS para asegurar su correcta operación.

Los problemas se hacen cada vez más evidentes a medida que la complejidad de la aplicación aumenta. Crece el modelo de datos (ya que comienzan a manejarse más relaciones) y se requiere un mayor acceso distribuido y escalable. Como resultado se obtiene una explosión de tablas (con juntas costosas entre ellas, ver figura 2), una complejidad cada vez más difícil de manejar, perdida de integridad y perdida de desempeño de varios ordenes de magnitud, todo en aumento. El usuario se esfuerza en lograr que su aplicación se acomode en tablas.

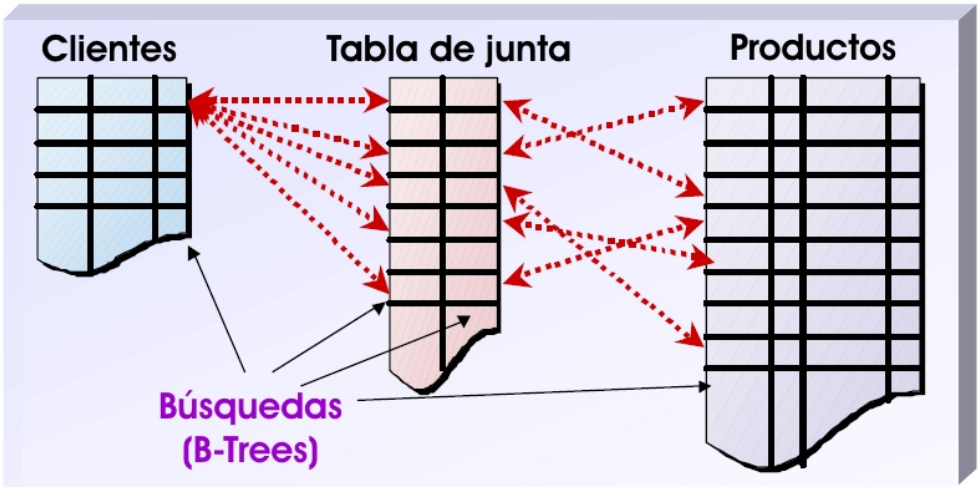

Figura 2 – Búsquedas sobre árboles B necesarias durante una junta en una base de datos relacional

Y no se trata solo de un aumento rápido del tiempo de ejecución a medida que aumenta la complejidad de la aplicación. También se incrementa la dificultad de cambio (perdida de flexibilidad) tanto de la base de datos como de la aplicación y, de forma similar, se sacrifica la extensibilidad.

Las Bases de Objetos

Cuando las capacidades de bases de datos se integran con la tecnología de objetos el resultado es una Base de Objetos (u ODBMS). A diferencia de los RDBMSs estas bases permiten que sus elementos se accedan como objetos propios de un entorno de programación basado en objetos. En la figura 3 se ilustra como se combinan los conceptos de tecnología de objetos y bases de datos para conformar una base de objetos.

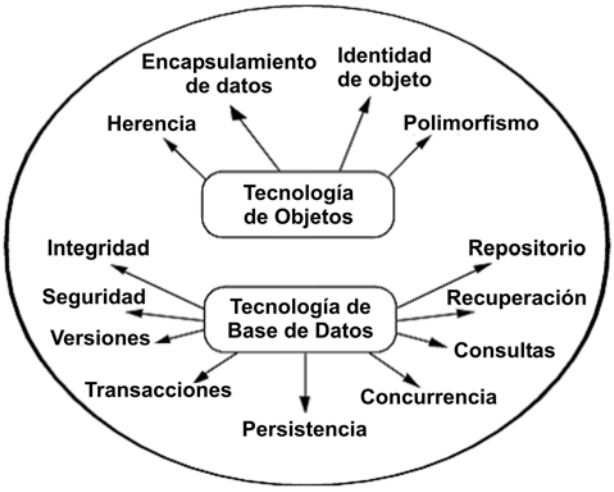

Figura 3 – Elementos de una base de objetos

Un ODBMS también extiende el lenguaje con persistencia transparente de datos, control de concurrencia, recuperación de datos, consultas asociativas y otras capacidades.

La persistencia transparente en las bases de objetos se refiere a la habilidad de manipular directamente, sin traducción o conversión, datos almacenados en la base utilizando un entorno de programación basado en objetos como Smalltalk, Java o C#. Esto contrasta con los RDBMSs donde se utiliza un sublenguaje como SQL o una interfaz de operación como ODBC o JDBC, y luego se utiliza código adicional para hacer la conversión a objetos. El utilizar una base de objetos se traduce en menos código relacionado a la persistencia (de un 25% a un 70% menos) lo que disminuye dramáticamente los tiempos de desarrollo (aunque cabe reconocer que varios de los frameworks de mapeo objeto-relacional del mercado logran un buen nivel de ahorro en la codificación). Con persistencia transparente la manipulación y recorrido de objetos persistidos se realiza directamente con el entorno de programación basado en objetos de la misma manera en que se lo hace con objetos en memoria no persistidos gracias al uso de un *caching* inteligente y sistemas de consultas que cada vez son más nativos al lenguaje (como *native queries* de db4o).

Otro beneficio relacionado ocurre en producción. Si se trabaja con datos complejos un ODBMS puede ofrecer un desempeño que es de 10 a 1000 veces superior comparado a un RDBMS. Cuando los datos se leen de disco ya están en el formato que usa por ejemplo Java o C++ por lo que no se requiere adaptación alguna. El nivel de mejora en el desempeño depende de la complejidad de los datos utilizados.

Los ODBMSs están diseñados con el propósito de almacenar y compartir objetos; constituyen una solución para el manejo de la persistencia de objetos (donde datos persistentes son datos que permanecen luego de que un proceso ha finalizado). Esto facilita el versionamiento de objetos ya que se almacena un objeto nuevo cada vez que se realizan cambios.

En general las bases de objetos almacenan los datos junto con los métodos apropiados para accederlos (proveen encapsulamiento), y pueden reducir la necesidad de realizar *paging* habilitando solo los objetos que se necesitan en un momento dado para que sean cargados en memoria. Esto contrasta con las bases de datos relacionales que cargan tablas que contienen ambos los datos requeridos y los que no son necesarios.

Los esfuerzos de estandarización de los 90s que intentaron imitar a SQL no tuvieron éxito. Sin embargo hay puntos en común entre las arquitecturas de los diferentes ODBMS; son básicamente tres componentes los necesarios: administradores de objetos, servidores de objetos y repositorios de objetos. A grandes rasgos las aplicaciones interactúan con administradores de objetos que trabajan a través de servidores de objetos para lograr acceder a repositorios de objetos.

Características Principales de las Bases de Objetos

Hay una serie de características que las bases de objetos poseen y nos permiten conocer algunos detalles de la tecnología.

Las bases de objetos no solo permiten trabajar transparentemente con un entorno de programación basado en objetos sino que soportan todos los conceptos de la tecnología de objetos:

- Agregación: objetos que están compuestos por otros objetos
- Encapsulamiento: almacenamiento de atributos junto con métodos. No todas las bases soportan métodos pero se apoyan en las clases definidas en los esquemas para reconstruir el objeto con sus métodos.
- Herencia: los objetos heredan atributos y comportamiento de sus objetos padre.
- Polimorfismo: permite a los objetos responder de forma distinta a un mismo mensaje (por ejemplo implementando un método de otra manera). Se soportan distintas versiones de los objetos.

Las bases de objetos presentan en muchos casos una arquitectura distribuida. Los objetos se comparten en un ambiente distribuido y la base de datos puede replicarse en múltiples computadoras. También operan en ambientes heterogéneos proveyendo soporte multiplataforma. La base puede correr en diferentes configuraciones de hardware y sistema operativo.

La mayoría de estas bases tienen la capacidad de procesamiento transaccional que soporta concurrencia. El procesamiento de transacciones asegura que se realice la transacción completamente o, en caso contrario, ninguna (todo o nada). Las transacciones soportan concurrencia y recuperación de datos. Una falla causa un *roll-back* de los datos (como normalmente se ve en las bases de datos relacionales). En lo que respecta a la concurrencia, las bases de objetos verifican en que momento permiten el acceso en paralelo a los datos. Este acceso concurrente implica que más de una aplicación o hilo podrían estar leyendo o actualizando los mismos objetos a la vez. Esto suele denominarse *commit* de dos fases (dos procesos pueden trabajar sobre el mismo objeto concurrentemente). Para ello se utiliza normalmente bloqueo de datos para lecturas y escrituras. Algunos de los métodos utilizados para el control de concurrencia en las bases de objetos incluyen:

- Pesimista: cuando uno o más procesos están leyendo no pueden realizarse actualizaciones en los datos.
- Multilectura: las actualizaciones no se bloquean. Los datos deben ser consistentes cuando comienza la transacción. En otras palabras, si se realizó la lectura y los datos fueron modificados por otro proceso antes de que fueran guardados, la transacción no es válida hasta que los datos sean leídos nuevamente.

En los ODBMSs los objetos se encuentran interrelacionados por referencias entre ellos de manera similar a como los objetos se referencian entre si en memoria. Se soportan todas las cardinalidades en las relaciones (uno a uno, uno a N, N a uno, N a N). Las relaciones entre objetos definen asociaciones y la posibilidad de que los objetos puedan detectarse mutuamente en una o dos direcciones. En las bases de objetos se utilizan normalmente relaciones bidireccionales para implementar la recolección de basura (*garbage collection*). La recolección de basura opera sobre los objetos que ya no son referenciados por la base de datos eliminándolos y evita que los programas externos deban llevar un registro de los punteros a objetos.

Con respecto a la transparencia las bases de objetos buscan la persistencia transparente que consiste en la manipulación directa de datos utilizando un entorno de programación basado en objetos. En muchas ocasiones se utiliza una clase con capacidad de persistencia o una interfaz persistente para la implementación. Si se utiliza una interfaz se aíslan las particularidades de las llamadas a la base con una capa intermedia lo que permite al cliente cambiar más fácilmente de vendedor en el futuro si fuera necesario. Algunos vendedores consideran esto como transparencia, sin embargo, el objetivo final de la persistencia

transparente es hacer desaparecer completamente la base de objetos a los ojos del usuario. En esta dirección es muy prometedor el uso de “consultas nativas” (*native queries*) que expresa las consultas sobre la base en el propio lenguaje de programación y las políticas de cero administración. db4objects es pionero en el uso de consultas nativas a partir de su versión 5 siguiendo una tendencia de la industria en la que también participa Microsoft con su iniciativa LINQ planeada para .NET3.

Adicionalmente, las bases de objetos soportan las siguientes características (aunque depende de cada producto):

- Interfaces a la base – Las bases pueden soportar SQL, OQL, y alguna interfaz de programación de aplicaciones (API), ODBC, JDBC, etc.
- Integridad de datos – Existen dos tipos:
    - La integridad estructural de la base de datos asegura que los contenidos son consistentes con el esquema de la base. La integridad referencial requiere relaciones bidireccionales entre objetos para asegurar que no hay referencias a objetos eliminados.
    - Integridad lógica: asegura que las propiedades lógicas de los datos son correctas. El valor es correcto para los datos de forma consistente y el acceso concurrente no provoca que se registren valores inválidos.
- Versionamiento de objetos – Un único objeto está representado por múltiples versiones. Dos tipos de versionamiento normalmente utilizados son:
    - Lineal – Las versiones previas del objeto se guardan a medida que el objeto es modificado.
    - Por rama (*branch*) – Múltiples usuarios pueden actualizar el objeto de forma concurrente. Cambios simultáneos de distintos usuarios generan nuevas ramas en el árbol de modificaciones.
- Notificación – La notificación puede ser activa o pasiva. Un sistema pasivo puede determinar de forma mínima si un objeto ha cambiado de estado. Un sistema activo puede informar a una aplicación cuando un objeto ha sido modificado.
- Indexación – Sistemas de indexación adicionales pueden utilizarse para mejorar la eficiencia de recuperación de datos. Normalmente se utiliza *hashing* y *b-trees*.
- Seguridad – Puede soportarse encriptación de los repositorios de objetos y/o transmisiones. También existen normalmente diferentes métodos y niveles de autenticación para acceder a la base de acuerdo al producto.
- Tolerancia a fallos – Son características que proveen una tolerancia a fallos en el caso de que ocurra un problema con el hardware o software. En las bases de objetos el procesamiento transaccional protege contra fallas de software mientras que la replicación de datos a otros servidores en la red lo hace contra fallas de hardware.
- Acceso a datos – El acceso normalmente se realiza utilizando un iterador para recorrer los datos como si los objetos estuvieran en colecciones (o diccionarios). De esta manera no se requiere que los objetos estén cargados en memoria antes de que se obtenga el objeto deseado (los objetos son cargados ‘por demanda’).
- Ordenamiento – Pueden obtenerse todos los objetos de una clase dada o clase padre fácilmente.
- Almacenamiento de métodos – Los métodos de los objetos que les proveen comportamiento se almacenan también en la base de datos.
- Herramientas y GUIs para trabajar con la base de objetos.

¿Cuando utilizar un ODBMS?

Hay situaciones donde las RDBMSs trabajan bien. Una gran cantidad aplicaciones sólidas han corrido sobre bases de datos relacionales por años. De forma similar hay motores relacionales disponibles de muy buena calidad. Sin embargo, hay casos donde el utilizar una

base de objetos resulta más efectivo. A continuación se brinda una lista (no exhaustiva) de estos casos.

Aplicaciones con DBMS Embebido

Las aplicaciones con DBMS embebido incluyen una base de datos en dispositivos móviles parcialmente conectados o un *caché* de objetos en una aplicación que requiere tiempos de respuesta muy rápidos. Si se utiliza Java o .NET se requiere una solución de persistencia auto contenida, no intrusiva y fácil de desplegar (ya sea *client-side* o en el *middleware*).

Si se almacenan objetos Java o .NET "tal cual como están en memoria" se logra implementar una solución de persistencia de forma elegante y poco intrusiva. El utilizar un RDBMS presenta la sobrecarga del mapeo objeto-relacional que resulta en una mayor demanda de recursos. Adicionalmente un RDBMS requiere un mayor compromiso en la administración de las bases de datos (especialmente cuando se deben desplegar esquemas de clases actualizados en la aplicación).

Relaciones de Datos Complejas

O más precisamente relaciones de objetos complejas. En aplicaciones con esta característica las clases definen múltiples referencias cruzadas mutuas. Las estructuras de datos en red presentan esta particularidad.

Se conforman así datos complejos que se representan comúnmente con un grafo (figura 4) y poseen las siguientes características:

- una falta de identificación única y natural,
- una gran cantidad de relaciones N a N
- normalmente accedidos utilizando recorridos (navegación de nodos)
- relaciones entre datos que determinan redes, árboles, estructuras recursivas, etc.
- uso frecuente de tipos de datos

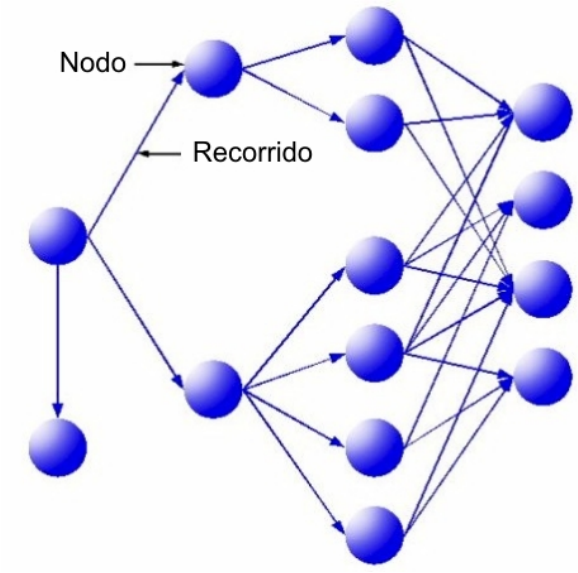

Figura 4 – Ejemplo de dato complejo (los nodos son objetos y los arcos sus interrelaciones)

En una base de datos relacional las referencias cruzadas complejas entre objetos son difíciles de modelar y tienden a presentar errores. En un RDBMS las relaciones entre objetos normalmente se manejan utilizando claves foráneas (*foreign keys*). Por ello el recuperar un objeto y luego los objetos que referencia y así sucesivamente puede resultar en un código complicado y difícil de mantener.

En cambio, la mayoría de los ODBMSs implementan un esquema de persistencia por capacidad de alcance. Ello significa que cualquier objeto referenciado por un objeto persistente también es persistido. Normalmente el programador puede especificar la profundidad de operación de este esquema en un árbol de objetos. De esta manera, conjuntos completos de objetos pueden ser almacenados y recuperados con una sola llamada; el ODBMS maneja automáticamente los detalles de mantenimiento de las referencias cuando se guardan y recuperan objetos.

Estructuras de Objetos 'profundas'

Este tema está relacionado al punto anterior. No todos los datos puede organizarse fácilmente en forma tabular (con filas y columnas) como se necesita en las bases relacionales. Algunos datos se organizan mejor en estructuras de grafos con forma 'inusual' o en estructuras de árbol. Una estructura de árbol muy profunda, por ejemplo, presenta al programador de la base de datos una cadena de referencias extensa que asocia padres, hijos, nietos, etc. lo que resulta difícil de mantener en un RDBMS.

Por lo tanto, las estructuras de objetos altamente conectadas no se traducen fácilmente a tablas de una base de datos relacional. El código de conversión puede ser confuso y difícil de mantener (en definitiva se pierde la estructura original en la traducción). Este esquema también puede presentar problemas de integridad (ej. solo se almacena una parte del árbol) y requiere que grandes porciones de código sean encapsuladas en transacciones para salvaguardar las relaciones de datos impidiendo así alcanzar un buen desempeño en aplicaciones multiusuario.

Un ODBMS no requiere una traducción de la estructura original a un modelo apto para la base de datos. El ODBMS le provee al programador control sobre la profundidad del alcance en la persistencia con lo que este puede controlar si se almacena o recupera todo el árbol, sólo una rama o algunas hojas. Y, nuevamente, la integridad de la estructura la conserva el propio motor del ODBMS.

Estructuras de Objetos cambiantes

En ocasiones puede anticiparse que las estructuras de las clases de una aplicación cambiarán con el tiempo. Quizá se reconoce que hay una alta probabilidad de que se agreguen nuevos miembros o nuevas relaciones (la mayoría de las aplicaciones cambian a medida que envejecen así como sus estructuras de datos subyacentes).

Un ODBMS típico soportará más fácilmente cambios a las estructuras de datos que un RDBMS. Si se utiliza este último, probablemente se necesite cambiar el esquema de la base (para acomodar la nueva estructura de los objetos) y luego alterar el código de consulta para manejar los cambios. Hasta en ocasiones puede ser necesario escribir una aplicación de conversión descartable para actualizar las tablas al nuevo formato.

Algunas ODBMSs permiten cambiar la estructura de los objetos "*on the fly*". Pueden mezclarse versiones nuevas y viejas de objetos en la misma base de datos. Si la nueva estructura del objeto tiene campos adicionales, el leer un objeto viejo en la aplicación carga los campos extra con valores por omisión (ej. null o cero). Si en cambio la nueva estructura tiene menos campos, al leer un objeto viejo se descartan los campos inexistentes (db4o, por ejemplo, provee incluso un mecanismo mediante el cual objetos viejos pueden actualizarse a objetos nuevos de forma transparente a medida que estos son recuperados desde la base de datos).

Uso de Técnicas Agiles en el Equipo de Desarrollo

Las técnicas ágiles de programación han crecido en popularidad demostrando beneficios en la reducción de errores en el desarrollo (entre otras ventajas). Un ODBMS se acopla de forma más conveniente a un entorno ágil de desarrollo comparado a un RDBMS. El desarrollo moderno de software es evolutivo por naturaleza, incluye técnicas de refactorización, modelado ágil, pruebas de regresión continuas, gestión de la configuración, etc. El uso de tecnología relacional complica la adopción de estas técnicas debido al desfasaje técnico-cultural y a la falta de herramientas adecuadas. Las bases de objetos hacen más fácil el desarrollo ágil.

Programación en un Entorno de Objetos

Parece un punto obvio pero es necesario mencionarlo. El utilizar un ODBMS en vez de un RDBMS significa que no es necesario escribir código de transacciones que pasan datos entre filas obtenidas de la base y los objetos concretos de la aplicación. Tampoco se necesita escribir código de mapeo entre los objetos y el esquema de la base (si se utiliza un framework de mapeo objeto-relacional). Esto se vuelve muy problemático cuando, por ejemplo, se deben mantener varias aplicaciones que acceden la misma base de datos pero de formas ligeramente distintas. En tal situación es necesario asegurarse que todo el código de traducción en las distintas aplicaciones esté sincronizado. Y si luego hay algún cambio en la estructura de la base, es necesario recorrer todas las aplicaciones y cerciorarse que el cambio se maneja correctamente.

Con un ODBMS el acceso es el mismo para todas las aplicaciones ya que los objetos recuperados y almacenados son manipulados de la misma forma. Y si se ha factorizado correctamente la aplicación un cambio en una estructura de clase es un cambio aislado en una biblioteca. No es necesario recorrer las aplicaciones para arreglar el código de traducción.

Objetos que incluyen Colecciones

En algunos casos las aplicaciones incluyen una o más clases que definen miembros que son colecciones (List, Set, etc).

Una colección en un objeto representa una relación uno a muchos. Tales relaciones, modeladas por un RDBMS, requieren una tabla intermedia que sirve como nexo entre el objeto padre (mantenido en una tabla) y los objetos de la colección (mantenidos en otra tabla). Esto se vuelve aun más dificultoso si la colección puede almacenar objetos de diferentes clases.

Un ODBMS, en cambio, no tendrá dificultad en manejar tal escenario. La colección se maneja como cualquier otro objeto (aunque potencialmente ‘profundo’) y normalmente se podrá recuperar y almacenar el objeto padre junto con la colección asociada y su contenido en una sola llamada.

Los Datos se acceden por Navegación más que por Búsqueda

Esta característica surge de forma natural cuando se manejan relaciones de datos complejas y estructuras de objetos profundas.

Si los datos se almacenan en una estructura de red densa y el acceso a datos se hace principalmente navegando la estructura de objetos, utilizar un ODBMS es ciertamente superior en comparación a una consulta según los valores de los datos. La navegación por el árbol utilizando un RDBMS se traduce en una serie de operaciones de consulta y recuperación (típicamente sentencias SELECT de SQL). En un ODBMS la navegación por el árbol se expresa de forma natural utilizando las construcciones nativas del lenguaje. El código resultante es más fácil de entender y mantener.

Conclusión

Las bases de objetos están aquí para quedarse y tienen la capacidad de cubrir las necesidades de datos de aplicaciones donde la tecnología relacional comienza a tener problemas de desempeño, escalabilidad, flexibilidad y/o complejidad de mantenimiento. Por otra parte, las bases de objetos pasivas (o esquemas de persistencia) constituyen una alternativa potente a los RDBMSs (y esquemas de mapeo objeto-relacional) en aplicaciones más tradicionales (ej. en aplicaciones web) y especialmente en sistemas embebidos (donde hoy lideran la relación costo/performance). Esta es una tecnología con desempeño equiparable (a aun superior en los casos descriptos) al de las bases de datos relacionales y presenta un nivel de transparencia inmejorable en lo que respecta a persistencia de objetos.

Terminología

Base de Objetos Activa o Pasiva: los ODBMS pueden distinguirse por poseer bases pasivas o activas. Las bases pasivas (en ocasiones denominadas esquemas de persistencia) proveen capacidades de almacenamiento y distribución de objetos dejando que todo el procesamiento de la aplicación se realice en la aplicación cliente. Las bases pasivas pueden almacenar descripciones de interfaces de los métodos definidos para las clases en el esquema, pero normalmente no almacenan la implementación de esos métodos y no ejecutan consultas o actualizaciones de objetos en los procesos de la base de datos. Por ello, en una base de objetos pasiva, todos los objetos sobre los que se necesita trabajar deben ser movidos desde la base de datos a los procesos de la aplicación. Por otra parte las bases de objetos activas almacenan la implementación de los métodos de una clase y tienen la capacidad de ejecutarlos en el espacio de procesos de la base de datos (conforman un ambiente de objetos que opera sobre disco). Las bases de objetos activas pueden por lo tanto realizar tareas de cómputo significativas y búsquedas asociativas sin mover los datos sobre los que debe trabajar al espacio de procesos de la aplicación. Normalmente las bases activas también proveen una o más interfaces que permiten que los programas no basados en objetos puedan acceder a los objetos y métodos almacenados.

Base de Objetos Nativa o No Nativa: las bases de objetos nativas están desarrolladas en el mismo lenguaje de programación con el que se trabaja y almacenan los objetos con una representación directa en el lenguaje. Por otra parte las no nativas almacenan los objetos con una representación propia y pueden estar desarrolladas en cualquier lenguaje de programación. Ambos acercamientos tienen sus ventajas y desventajas. Por ejemplo las no nativas se prestan más fácilmente a almacenar objetos de distintos lenguajes de programación mientras que las nativas ofrecen ligeramente mayor rapidez y simplicidad. Un buen ejemplo de base nativa para Java es *db4objects*.

Base de Objetos Embebida (o empotrada): una base de datos embebida es una base que es invisible para el usuario final. Se utilizan normalmente en tres áreas: en dispositivos móviles (ej. celulares), en sistemas de tiempo real (ej. sistemas de control de trenes o robots industriales) y en aplicaciones de Internet donde el usuario no se relaciona con la base de datos subyacente de forma directa. En general son bibliotecas que tienen un tamaño (*footprint*) reducido. Un ejemplo de base de objetos embebida es *db4objects*.

Esquemas de Prevalencia: son *frameworks* que mantienen todos los objetos en memoria volátil y periódicamente utilizan esquemas de serialización de objetos para guardar una imagen (*snapshot*) de los objetos de la aplicación. Es un acercamiento muy simple y ultrarrápido a la persistencia aunque presenta algunos problemas de escalabilidad. Un ejemplo conocido para Java es *Prevlayer* (www.prevayler.org).