cpi fp Los Enlaces

Acceso a Datos

Organización y persistencia de la Información

Contenido

1. Introducción
2. Acceso a la Información
3. ¿Qué método usar?
    - 3.1 Ficheros
    - 3.2 Bases de Datos (Relacionales, Orientadas a Objetos, Objeto – Relacional, No-SQL)

1. Introducción.

Podemos diferenciar, a grandes rasgos, las aplicaciones en dos partes:

- Por un lado, el **programa** propiamente dicho, que realiza las operaciones deseadas con los datos necesarios.

- Por otro lado, los **datos** con los que opera el programa. Esos datos pueden ser obtenidos por el programa mediante diversos métodos: leídos mediante teclado, escaneados, leídos de algún soporte de almacenamiento secundario, etc.

En la mayoría de los casos, cuando programamos, nos interesa que el programa guarde los datos que le hemos introducido, o los resultados que dicho programa haya obtenido, de manera que si el programa termina su ejecución, los datos no se pierdan y puedan ser recuperados posteriormente, es decir, persistan.

2. Acceso a la Información.

Hay diversas estrategias de acceso a datos para gestionar la persistencia de los datos:

- Mediante ficheros.

- Bases de datos, que pueden ser:
  > Relacionales,
  > Orientadas a objetos,
  > Objeto-relacionales.
  > Mapeo objeto relacional (ORM).
  > Bases de datos XML (eXtensible Markup Language).

- Componentes.

3. ¿Qué método usar?

- No se puede afirmar que haya un método que sea el mejor de manera absoluta. Más bien, la cuestión es tener claro qué tipo de aplicación hay que construir y, según eso, estudiar qué tipo de sistema de almacenamiento será mejor usar: si una base de datos orientada a objetos, o una base de datos XML, etc.

- Conociendo el funcionamiento de las diferentes alternativas podemos comparar sus prestaciones al problema de la persistencia concreto que se nos presente.

- Cada una de las tecnologías tiene su propio origen y filosofía para alcanzar el mismo fin y, por esta razón, no es fácil analizar sus ventajas y desventajas frente a las demás alternativas.

3.1. Ficheros (I)

En las antiguas aplicaciones informáticas, antes de que surgieran las bases de datos, la información se guardaba en ficheros. Así, por ejemplo, una aplicación que guardaba los datos de personas, almacenaba dichos datos en un fichero convencional cuyo contenido podía ser este:

un fichero de texto no tiene campos

```
Antonio Pérez Pérez 30 C/ Morales nº 11 Madrid Madrid

Feliciano Gómez Sander 25 C/ Terreros nº 121 Vitoria Vitoria

Arturo Bueno Hernández 46 C/ Cocoliso nº 43 Murcia Murcia
```

- El **programador** de las aplicaciones que usaran ese fichero, **tuviera que construir el programa conociendo detalladamente las posiciones de los datos**, para saber desde qué posición hasta qué otra posición, se guardaba el nombre y apellidos, etc.

3.1. Ficheros (II)

- El programador tendría que controlar si se guardan filas de datos duplicadas, y así un montón de inconvenientes.
- Por eso, cuando surgieron las bases de datos, se empezó a dejar de usar los ficheros convencionales.
- Pero bien es cierto, que aún en las más **modernas aplicaciones**, a veces **necesitamos un simple fichero para guardar información**, como por ejemplo un **fichero de configuración, o un fichero log**. Es decir, no siempre nos hace falta una base de datos para almacenar la información.
- Se usan ficheros que guardan datos siguiendo un patrón o **estructura bien definida**, en otros métodos de almacenamiento, como por ejemplo en ficheros y en bases de datos XML. Éstos son archivos de texto que por consiguiente no necesitan un software propietario para ser interpretados, como ocurre con la mayoría de los archivos binarios.
- También debemos tener en cuenta, que las bases de datos relativamente modernas, como son las bases de datos XML o algunas orientadas a objetos, guardan sus datos empleando ficheros XML o JSON.

3.1. Ficheros (III)

Por eso, se recurre a utilizar este tipo de soluciones, el uso de ficheros en vez de bases de datos, y en particular de ficheros XML cuando se necesita intercambiar información a través de varias plataformas de hardware o de software, o de varias aplicaciones. A veces se exporta de una base de datos a ficheros XML para trasladar la información a otra base de datos que leerá esos ficheros XML.

3.2. Bases de Datos (I)

Al principio, en los primeros tiempos de la informática, los datos se guardaban en ficheros convencionales. Con el tiempo, y la experiencia de trabajar con dichos ficheros, se observaron los inconvenientes de los ficheros, y para intentar solucionar los inconvenientes que se observaron surgieron las bases de datos.

Un sistema de bases de datos es:

- Un **sistema de información** orientado hacia los datos, que pretende recuperar y almacenar la información de manera eficiente y cómoda.

Surge en un intento de resolver las dificultades del procesamiento tradicional de datos, teniendo en cuenta que los datos suelen ser independientes de las aplicaciones

3.2. Bases de Datos (II)

Ventajas

- **Independencia** de los datos respecto de los procedimientos. El usuario tiene una visión abstracta de los datos, sin necesidad de ningún conocimiento sobre la implementación de los ficheros de datos, índices, etc. Esto supone un gran ahorro en los costes de programación, de forma que la modificación de la estructura de los datos no suponga un cambio en los programas y viceversa. Sin ella, el mantenimiento de la base de datos ocuparía el 50% de los recursos humanos dedicados al desarrollo de cualquier aplicación.
- **Disminución de las redundancias** y en consecuencia, disminución de la posibilidad de que se produzca **inconsistencia** de datos.
- Mayor **integridad** de los datos.
- Mayor **disponibilidad** de los datos.
- Mayor **seguridad** de los datos.

3.2. Bases de Datos (III)

- Mayor **privacidad** de los datos.
- Mayor **eficiencia** en la recogida, codificación y entrada en el sistema.
- Lo que se suele denominar interfaz con el pasado y futuro: una base de datos debe estar abierta a reconocer información organizada físicamente por otro software.
- **Compartición** de los datos. Los datos deben poder ser accedidos por varios usuarios simultáneamente, teniendo previstos procedimientos para salvaguardar la integridad de los mismos.

Podemos afirmar generalizando, que se usa un sistema de ficheros convencional **cuando la cantidad de datos a guardar es tan reducida** que no justifica las desventajas del uso de los sistemas de bases de datos. Por ejemplo, para guardar los datos del resultado de la instalación de un programa, usamos un fichero de texto, no se guardan los datos en una base de datos.

3.2.1. BBDD Relacionales (I)

- **El propósito del modelo relacional es proporcionar un método declarativo para especificar datos y consultas**.

- Así, en el diseño de la base de datos establecemos qué información contendrá dicha base de datos, luego recuperaremos la información que queramos, y dejamos al software del sistema gestor de la base de datos que se ocupe de: describir las estructuras de datos para almacenarlos, y gestionar los procedimientos de recuperación para obtener las consultas deseadas.

- Las bases de datos relacionales son adecuadas para manejar grandes cantidades de datos, compartir datos entre programas, realizar búsquedas rápidas, etc. Pero tienen como desventaja fundamental que no presentan un buen modelo de las relaciones entre los datos, ya que todo se representa como tablas bidimensionales, o sea, en filas y columnas.

3.2.1. BBDD Relacionales (II)

Podemos decir de las bases de datos relacionales que:

- Están muy extendidas.
- Son muy robustas.
- Permiten interoperabilidad entre aplicaciones, es decir una forma de compartir datos entre aplicaciones.
- Son el común denominador de muchos sistemas y tecnologías. Una base de datos puede ser utilizada en programas realizados en Java, o en C++, etc.

3.2.2. BBDD Orientadas a Objetos (I)

El origen de las Bases de datos orientadas a objetos (en adelante: BDOO) se debe básicamente a las siguientes razones:

- La existencia de problemas al representar cierta información y modelar ciertos aspectos del mundo real. Los modelos clásicos permiten representar gran cantidad de datos, pero las operaciones y representaciones que se pueden realizar sobre ellos son bastante simples.

- Pasar del modelo de objetos al modelo relacional, para almacenar la información, genera dificultades que en el caso de las BDOO no surgen, ya que el modelo es el mismo. Es decir, los datos de los programas escritos en lenguaje orientado a objetos se pueden almacenar directamente, sin conversión alguna, en las BDOO.

Los sistemas de bases de datos orientadas a objetos soportan un modelo de objetos puro, ya que no están basados en extensiones de otros modelos más clásicos como el relacional.

3.2.2. BBDD Orientadas a Objetos (II)

Por ello, una característica general es que el **lenguaje de programación y el esquema de la base de datos utilizan las mismas definiciones de tipos**.

El acceso a los datos puede ser más rápido con las bases de datos orientadas a objetos que con las bases de datos tradicionales porque no hay necesidad de utilizar las uniones o joins, que si se necesitan en los esquemas relacionales tabulares. Esto se debe a que un objeto puede recuperarse directamente sin una búsqueda, simplemente siguiendo punteros.

Ventajas:

- **Transparencia** (no ensucia la construcción de una aplicación)
- El desarrollador así se debe preocupar de los objetos de su aplicación, en lugar de cómo los debe almacenar y recuperar de un medio físico.

3.2.2. BBDD Orientadas a Objetos (III)

- Permiten mayor capacidad de modelado. El modelado de datos orientado a objetos permite modelar el mundo real de una manera mucho más fiel. Esto se debe a:
  - un objeto permite encapsular tanto un estado como un comportamiento
  - un objeto puede almacenar todas las relaciones que tenga con otros objetos
  - los objetos pueden agruparse para formar objetos complejos (herencia).

- Extensibilidad, debido a que:
  - Se pueden construir nuevos tipos de datos a partir de los ya existentes.
  - Podemos agrupar propiedades comunes de diversas clases e incluirlas en una superclase, lo que reduce la redundancia.
  - Tenemos reusabilidad de clases, lo que repercute en una mayor facilidad de mantenimiento y un menor tiempo de desarrollo.

3.2.2. BBDD Orientadas a Objetos (IV)

- **Prestaciones**. Los sistemas gestores de BDOO proporcionan mejoras significativas de rendimiento con respecto a los SGBD relacionales. Aunque hay autores, que han argumentado que los bancos de prueba, usados en dichas pruebas, están dirigidos a aplicaciones de ingeniería donde los SGBDOO son más adecuados. También está demostrado, que los SGBDR tienen un rendimiento mejor que los SGBDOO en las aplicaciones tradicionales de bases de datos como el procesamiento de *transacciones*.
- **Reglas de acceso**. En las BDRELACIONALES, a los atributos se accede y se modifican a través de operadores relacionales predefinidos. En las orientadas a objetos se procede mediante las interfaces que se creen a tal efecto de las clases. Desde este punto de vista, los sistemas orientados a objetos dan independencia a cada objeto que el sistema relacional no permite.
- **Clave**. En el modelo relacional, las claves primarias generalmente tienen una forma representable en texto, sin embargo los objetos no necesitan una representación visible del identificador.

3.2.2. BBDD Orientadas a Objetos (V)

Desventajas:

- La reticencia del mercado, tanto para desarrolladores como usuarios, a este tipo de bases de datos.

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

- Carencia de experiencia. Al ser una tecnología relativamente nueva, todavía no se dispone del nivel de experiencia del que se dispone para los sistemas relacionales.

3.2.2. BBDD Orientadas a Objetos (VI)

- Panorama actual. Tanto los sistemas gestores de bases de datos **como los sistemas gestores de bases de datos objeto-relacionales están muy extendidos** SQL es un estándar aprobado y ODBC y JDBC son estándares 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.

- **Dificultades en optimización**. La optimización de consultas necesita realizar 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.

3.2.3. BBDD Objeto-Relacional (I)

Entendemos **Base de Datos Objeto Relacional (BDOR)**, una base de datos que ha evolucionado desde el modelo relacional a otro extendido que incorpora conceptos del paradigma orientado a objetos. Por tanto, un Sistema de Gestión Objeto-Relacional (SGBDOR) contiene ambas tecnologías: relacional y de objetos.

- En una Base de Datos Objeto Relacional el diseñador puede crear sus propios tipos de datos y crear métodos para esos tipos de datos.
- Las estructuras de datos que se utilizan para almacenar la información siguen siendo tablas, los diseñadores pueden utilizar muchos de los mecanismos de orientación a objetos para definir y acceder a los datos.
- Se reconoce el concepto de objetos, de tal manera que un objeto tiene un tipo, se almacena en cierta fila de cierta tabla y tiene un identificador único (OID). Estos identificadores se pueden utilizar para referenciar a otros objetos y así representar relaciones de asociación y de agregación.

3.2.3. BBDD Objeto-Relacional (II)

Ventajas

- La ventaja de este tipo de base de datos es que los usuarios pueden pasar sus aplicaciones actuales sobre bases de datos relaciones este nuevo modelo sin tener que reescribirlas.

- Más tarde, se pueden ir adaptando las aplicaciones y bases de datos para que utilicen las funciones orientadas a objetos.

- Con las Bases de Datos Objeto-Relacional se amplía el modelo relacional destacando las siguientes aportaciones:
  > Se aumentan la variedad en los tipos de datos, se pueden crear nuevos tipos de datos que permitan construir aplicaciones complejas con una gran riqueza de dominios. Se soportan tipos complejos como: registros, conjuntos, referencias, listas, pilas, colas y vectores.

3.2.3. BBDD Objeto-Relacional (III)

- Hay extensiones en el control de la Semántica de datos Objeto-Relacionales:
  > Se pueden crear procedimientos almacenados y funciones que tengan un código en algún lenguaje de programación, como por ejemplo: SQL, Java, C, etc.

- Se pueden compartir varias librerías de clases ya existentes, esto es lo que conocemos como reusabilidad

- Como productos comerciales de bases de Datos Objeto-Relacional podemos destacar:
  - DB2 Universal Database de IBM (International Business Machines).
  - Universal Server de Informix (ahora de IBM),
  - INGRES II, de Computer Associates.
  - ORACLE de Oracle Corporation,
  - SyBASE

- Como productos de código abierto, destacamos PostGreSQL

3.2.4. Conectores de Bases de Datos (I)

- Para gestionar la información de las bases de datos hay unos estándares que facilitan a los programas informáticos manipular la información almacenada en ellas.

- En Java existe un API basado en estos estándares que permite al desarrollar aplicaciones, que se pueda acceder a bases de datos relacionales: JDBC (Java Database Connectivity, conectividad de bases de datos de Java).

- La mayoría de las aplicaciones importantes de una empresa están respaldadas por una arquitectura normalizada y optimizada de bases de datos relacionales. Tradicionalmente, dichas aplicaciones están basadas en sentencias SQL con las cuales se gestionan todos los datos que manejan.

3.2.4. Conectores de Bases de Datos (II)

- Este modelo continúa teniendo una gran importancia estratégica y es la base para el continuo crecimiento del mapeo Objeto-Relacional (O/R) y está asociado a los mecanismos de persistencia.

- La ventaja de usar conectores, por ejemplo JDBC, es que independiza de la base de datos que utilice.

- No obstante hay un trabajo de traducción para mapear los campos devueltos por cada consulta a la colección de objetos correspondiente. Y hay que trabajar las sentencias de actualización, inserción y eliminación para cada uno de los campos. Esto constituye una razón para tratar de buscar alternativas menos costosas en tiempo de desarrollo.

3.2.4. Conectores de Bases de Datos (III)

JDBC.

Un driver JDBC es un componente software que posibilita a una aplicación Java interaccionar con una base de datos.

- El API JDBC define interfaces y clases para escribir aplicaciones de bases de datos en Java realizando conexiones de base de datos.
- Mediante JDBC el programador puede enviar sentencias SQL, y PL/SQL a una base de datos relacional. JDBC permite embeber SQL dentro de código Java.
- La ventaja de usar conectores JDBC es que independiza de la base de datos que utilice.

3.2.5. Mapeo Objeto Relacional (I)

Con la expansión de la programación orientada a objetos, se da la circunstancia de que las bases de datos relacionales se siguen utilizando hoy en día de manera masiva, por lo que muchas aplicaciones se desarrollan con POO pero los datos, no se almacenan en bases de datos orientadas a objetos, sino en las bases de datos relacionales.

El sistema más extendido en las empresas hoy en día para guardar la información de sus aplicaciones es el uso de una base de datos relacional. Tradicionalmente, dichas aplicaciones están basadas en sentencias SQL con las cuales se gestionan todos los datos que manejan. Este sistema es la base para el continuo crecimiento del **mapeo Objeto-Relacional (O/R)** y está asociado a los mecanismos de persistencia.

3.2.5. Mapeo Objeto Relacional (II)

Cuando se programan sistemas orientados a objetos, utilizando una base de datos relacional, los programadores invierten gran cantidad de tiempo en desarrollar los objetos persistentes, o sea, convertir los objetos del lenguaje de programación a registros de la base de datos. Igualmente, también pasan bastante tiempo implementando la operación inversa, es decir, convirtiendo los registros en objetos.

El mapeo objeto-relacional (Object-Relational Mapping, o ORM) consisten en una técnica de programación para convertir datos entre el sistema de tipos utilizado en un lenguaje de programación orientado a objetos y el sistema utilizado en una base de datos relacional.

Cuando se trabajan con programación orientada a objetos y con bases de datos relacionales, es fácil observar que estos son dos paradigmas diferentes. El modelo relacional trata con relaciones y conjuntos, es de naturaleza matemática.
Por el contrario, el paradigma orientado a objetos trata con objetos, atributos y asociaciones de unos con otros.

3.2.5. Mapeo Objeto Relacional (III)

Cuando se requiere almacenar la información de los objetos utilizando una base de datos relacional se comprueba que hay un problema de compatibilidad entre estos dos paradigmas, el llamado ***desfase objeto relacional***.

Por ello, para ahorrar trabajo al programador, se puede utilizar un framework que se encargue de realizar estas tareas de modo transparente, de modo que el programador no tenga por qué usar JDBC ni SQL y la gestión del acceso a base de datos esté centralizada en un componente único permitiendo su reutilización.

3.2.6. Bases de Datos No-SQL (I)

Pero llegó la web, el software como servicio, los servicios en la nube, las startups de éxito con millones de usuarios, la explosión de las redes sociales, el volumen y la variedad de información que se genera ha puesto de manifiesto unas necesidades de almacenamiento y procesamiento de la información que no se habían tenido hasta ahora De esta manera llegaron los problemas de alta escalabilidad.

Esta generación masiva de información si bien los modelos relaciones se pueden adaptar para hacerlos escalar incluso en los entornos más difíciles, han sacado a relucir ciertos problemas que experimentan los SGBD tradicionales cuando se les somete a ciertas situaciones: lentitud de accesos, problemas de bloqueos, dificultad para mantener bases de datos distribuidas geográficamente, necesidades avanzadas de particionado, etc.

3.2.6. Bases de Datos No-SQL (II)

Los sistemas NoSQL intentan atacar este problema proponiendo una estructura de almacenamiento más versátil, aunque sea a costa de perder ciertas funcionalidades como las transacciones que engloban operaciones en más de una colección de datos, o la incapacidad de ejecutar el producto cartesiano de dos tablas (también llamado JOIN) teniendo que recurrir a la desnormalización de datos.

En estos sistemas importa más la flexibilidad, la velocidad y la capacidad de escalado horizontal que otras cuestiones tradicionalmente cruciales como la consistencia o disponer de una estructura perfectamente definida para los datos.

Por lo tanto hablar de bases de datos NoSQL es hablar de ***estructuras que nos permiten almacenar información*** en aquellas situaciones en las que las bases de datos relacionales generan ciertos problemas debido principalmente a problemas de escalabilidad y rendimiento de las bases de datos relacionales donde se dan cita miles de usuarios concurrentes y con millones de consultas diarias.

3.2.6. Bases de Datos No-SQL (III)

Los sistemas NoSQL ("no sólo SQL") difieren del modelo clásico de SGBDR (Sistema de Gestión de Bases de Datos Relacionales, RDBMS) en aspectos importantes, siendo el más destacado que no usan SQL como lenguaje principal de consultas.

Los datos almacenados no requieren estructuras fijas como tablas, normalmente no soportan operaciones JOIN, ni garantizan completamente ACID (atomicidad, consistencia, aislamiento y durabilidad), y habitualmente escalan bien horizontalmente.

3.2.6. Bases de Datos No-SQL (IV)

Algunas de las razones que nos pueden llevar a decantarnos por el uso de las bases de datos NoSQL en lugar de las clásicas SQL son:

- Cuando el volumen de los datos crece muy rápidamente en momentos puntuales, pudiendo llegar a superar el Terabyte de información.
- Cuando la escalabilidad de la solución relacional no es viable tanto a nivel de costes como a nivel técnico.
- Cuando tenemos elevados picos de uso del sistema por parte de los usuarios en múltiples ocasiones.
- Cuando el esquema de la base de datos no es homogéneo, es decir, cuando en cada inserción de datos la información que se almacena puede tener campos distintos.

3.2.6. Bases de Datos No-SQL (V)

Tipos de BBDD No-SQL:

- Bases de datos Clave-Valor
- Bases de datos Orientadas a objetos
- Bases de datos Orientadas a columnas
- Bases de datos Documentales (BD XML)
- Bases de datos en Grafo
- Bases de datos Orientadas a Objetos

3.2.7. Desarrollo de Componentes (I)

Un componente es una unidad de software que realiza una función bien definida y posee una interfaz bien definida.

Un componente software posee interfaces especificadas contractualmente, pudiendo ser desplegado independientemente y puede interaccionar con otros componentes para formar un sistema.

Un interfaz es un punto de acceso a los componentes: permite a los clientes acceder a los servicios proporcionados por un componente.

La idea de dividir u organizar en trozos el software, o sea, dividirlo en componentes surge para reducir la complejidad del software. Así se permite la reutilización y se acelera el proceso de ensamblaje del software.

3.2.7. Desarrollo de Componentes (II)

Los creadores de componentes pueden especializarse creando objetos cada vez más complejos y de mayor calidad.
La interoperabilidad entre componentes de distintos fabricantes:

> Aumenta la competencia,
> Reduce los costes y,
> Facilita la construcción de estándares.

Por tanto, así el software se hace cada vez más rápido, de mejor calidad y a menor coste incluso de mantenimiento.

Un componente software también debe especificar sus necesidades para funcionar correctamente, lo que se denominan las dependencias de contexto:

> Interfaces requeridas
> Entorno de ejecución: máquina necesaria, sistema operativo a utilizar, etc.

Dudas y preguntas