Arquitectura de sistemas

Abelardo Pardo

University of Sydney
School of Electrical and Information Engineering
NSW, 2006, Australia
Autor principal del curso de 2009 a 2012

Iria Estévez Ayres

Damaris Fuentes Lorenzo

Pablo Basanta Val

Pedro J. Muñoz Merino

Hugo A. Parada

Derick Leony

Universidad Carlos III de Madrid
Departamento de Ingeniería Telemática
Avenida Universidad 30, E28911 Leganés (Madrid), España

Capítulo 10. Modelado de estructura de datos

Tabla de contenidos

Este material proporciona una breve introducción al modelado de datos y cómo aplicarlo a diferentes aplicaciones software a desarrollar.

10.1. El modelado de datos

En toda aplicación software a desarrollar, antes de proceder a la implementación concreta del código de programa, se debe realizar un modelado de las diferentes partes que componen la aplicación, de manera que ayude a tener una visión clara y nítida de lo que posteriormente se va a implementar.

Entre los aspectos a modelar en una aplicación software, se encuentra los datos del programa. Esto engloba cualquier información que pueda ser relevante o haya que utilizar a lo largo del programa.

El modelado de datos en su primera fase, implica realizar una serie de notaciones gráficas, textuales, etc. que representan la información que la aplicación va a manejar, sus relaciones, etc. pero expresado de una manera que es independiente del lenguaje de programación utilizado. Luego, en posteriores fases, se decidirá como esa información se debe guardar para una aplicación o/y lenguaje de programación determinada, teniendo en cuenta las características particulares del lenguaje de programación a utilizar.

De hecho, existen lenguajes de modelado de alto nivel como UML (Unified Modeling Language) para ayudar en estas tareas, aunque no son objeto de aprendizaje de esta asignatura.

ayudar en estas tareas, aunque no son objeto de aprendizaje de esta asignatura.

10.1.1. Preguntas de autoevaluación

Para cada una de las afirmaciones seleccione si es verdadero o falso

1. El modelado de una aplicación es una actividad opcional durante el desarrollo de software.
   - Verdadero
   - Falso
2. El modelado de datos debe expresarse en una notación independiente del lenguaje de programación utilizado.
   - Verdadero
   - Falso
3. UML es una lenguaje de programación.
   - Verdadero
   - Falso

10.2. Aplicación de referencia

A continuación se presenta una aplicación de referencia para este material. Para la explicación y actividades sucesivas, nos basaremos en múltiples ocasiones en esta aplicación de referencia.

La aplicación de referencia para esta sesión, debe ser capaz de gestionar un conjunto de fotos indefinido, cada una de las cuales tiene un identificador único, un nombre de fichero, un tipo de fichero (indica el formato, que puede ser JPG, BMP, o GIF), el tamaño de la foto y el autor de dicha foto.

Adicionalmente, muchas de las fotos son realizadas por los mismos autores, y de cada autor se tiene que almacenar su nombre, apellidos, edad, y teléfono de contacto.

Esas fotos se deben poder listar según diferentes condiciones, y mostrar la información asociada a dichas fotos así como de sus autores. Así mismo, la aplicación debe ser capaz de mostrar la información de qué número total de fotos lleva realizadas un autor específico, y la hora que actualmente tiene el sistema.

10.3. Modelado de la información en una aplicación

En primer lugar, un diseñador podría preguntarse cuál es toda la información necesaria en la aplicación dada en una especificación. La información puede dividirse en los siguientes 2 grupos dependiendo de si se admiten instancias o no de la información:

1. Información que no puede tener múltiples instancias. Un ejemplo de este tipo de información en una aplicación podría ser la hora del sistema. La aplicación, no puede tener a la vez diferentes horas del sistema, sino sólamente una, que por supuesto irá cambiando a lo largo del tiempo, pero que en un instante de tiempo no tendrá varias instancias. Para este tipo de información, se puede tener a su vez que se tenga claro desde el inicio cual es la memoria necesaria para guardar dicha información (como es el caso de la hora) o bien que no se tenga claro desde el inicio cual es la cantidad de memoria necesaria para guardar dicha información ya que puede variar (por ejemplo podría ser una cadena de longitud indefinida).

2. Información que puede tener múltiples instancias. Se podría establecer una analogía entre instancia y objeto. Un ejemplo de este tipo de información en la aplicación de referencia es las fotos y otra los autores. Ya que pueden existir múltiples instancias de fotos y de autores presentes en el mismo instante de tiempo. Además, suele ocurrir que el número de instancias no está predefinido a priori y que puede variar a lo largo del tiempo, como ocurre en el caso de la aplicación de referencia para las fotos y los autores.

Para modelar cada uno de los elementos que pueden tener múltiples instancias, se utiliza una tabla, que incluirá todos los atributos de información que puede tener dicho elemento, junto con la indicación del tipo de datos que corresponde a cada atributo. Así mismo, también se indica que campo o combinación de campos hacen posible identificar a cualquier instancia de dicho elemento, diferenciándola de todas las demás instancias. A esto se denomina PRIMARY KEY (PK). Por ejemplo, para las fotos podría ser un identificador único, y para los autores la combinación de los campos nombre y apellidos, los que los identifique unívocamente.

La estructura de la tabla que almacena los elementos "foto" es la siguiente:

Foto	
Campo	**Tipo de datos**
ID(PK)	Entero
Nombre	String
Tipo fichero	Entero
Tamaño	Entero
Autor_ID	Entero

La estructura de la tabla que almacena los elementos "autor" es la siguiente:

Autor	
Campo	**Tipo de datos**
Autor_ID	Entero
Nombre (PK)	String
Apellidos (PK)	String
Edad	Entero
Teléfono	Entero

10.3.1. Preguntas de autoevaluación

Responde a las siguientes preguntas

1. El campo de una tabla que hace posible la identificación única de una instancia recibe el nombre de "primary key"

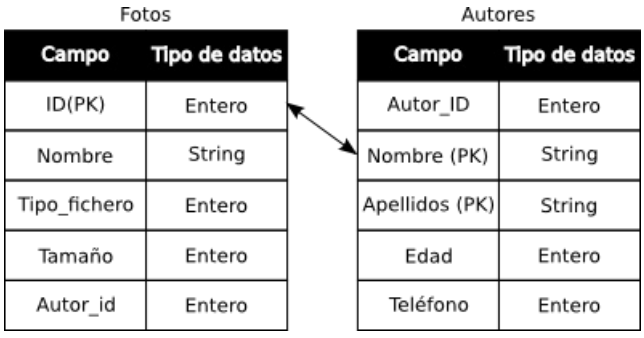
nombre de "primary key".

- Verdadero
- Falso

2. La "primary key" puede formarse por varios campos de la tabla
   - Verdadero
   - Falso
3. La "PRIMARY KEY" de la tabla que almacena los elementos “autor” está formada por los campos: Nombre y Apellidos. De acuerdo con esta restricción la tabla podrá almacenar?
   - Dos autores con el mismo nombre y apellido
   - Solo autores con diferente nombre

10.4. Establecimiento de relaciones entre diferentes tablas de la aplicación

En multitud de aplicaciones, existen relaciones entre diferentes tablas presentes en la aplicación y eso se debe reflejar en el modelo de datos. Por ejemplo, en la aplicación de referencia, existe una relación entre las tablas de fotos y autores, dado que cada foto tiene un autor. Una posible solución, diferente a la propuesta, habría sido tener una única tabla de FOTOS donde se incluyeran todos y cada uno de los campos de la tabla de autores, además de los propios de la tabla de fotos. Pero dicha solución sería muy ineficiente, dado que existen fotos que comparten el mismo autor, por lo que replicar la información de un mismo autor varias veces no es una buena idea. Al separar por otro lado la tabla de AUTORES, estamos utilizando sólo una instancia de autor para todas las fotos que comparten dicho autor. Para establecer esta relación entre tablas, establecemos una relación entre los campos de AUTOR_ID de ambas tablas, de tal manera que relacionamos ambos campos de las tablas, de forma que la información del autor de una foto vamos a buscarla a la tabla de autores que tenga el mismo identificador en dicho campo. Este es el motivo de que se haya añadido el campo AUTOR_ID en la tabla de autores. Gráficamente, esa relación hay que expresarla en el modelo de datos, mediante una flecha por ejemplo. A tal relación se la denomina una FOREIGN KEY (FK).



Aunque no las veremos en este material, se pueden establecer otros tipos de relaciones entre las tablas de una aplicación.

10.5. Persistencia de la información

Una pregunta que debe plantearse un desarrollador de aplicaciones software acerca de los datos de su aplicación, es si estos deben ser persistentes o no ante diferentes ejecuciones del programa.

Habitualmente, un programa software se ejecutará varias veces. Cada vez que se ponga en ejecución, se hará después de que el programa se lance a través de un comando. Dicho programa en ejecución

finalizará al recibir una señal, porque llega al final de su código (quizás existe incluso una opción de finalizar que el propio usuario tiene disponible en el menú) o porque se ha realizado una llamada a una función que termina el programa. Ese programa se puede volver a ejecutar varias veces, repitiendo el ciclo en el que el programa se ejecuta y termina.

Hay ocasiones en las que la aplicación requiere que los datos que hubiera resultado de la historia de previas ejecuciones, se mantengan, es decir que sean persistentes (por ejemplo que fotos añadidas en anteriores ejecuciones se vean en los listados de futuras ejecuciones del programa), mientras que hay otras ocasiones en las que la aplicación requiere que los datos de anteriores ejecuciones no se mantengan y se empiece desde cero tras cada nueva ejecución, es decir los datos no sean persistentes (por ejemplo que al principio el listado de fotos esté vacío y se muestren sólo las fotos que se han tomado en la última sesión).

Para el almacenamiento de información persistente ante diferentes ejecuciones, no cabe la posibilidad de almacenarla en memoria del programa ya sea estática o dinámica, porque tras la terminación de ejecución del programa, dicha información desaparece o aún peor, queda inaccesible consumiendo memoria (en el caso de las fugas de memoria). Por ello, para el almacenamiento de información persistente, se deben utilizar otros métodos, tales como los siguientes:

1. Almacenamiento en una Base de datos. No se verán en la asignatura. Son utilidades que permiten el almacenamiento persistente a través de una serie tablas. Permiten definir tablas, sus primary keys, las relaciones entre tablas, etc. y están optimizadas para realizar operaciones de datos a gran velocidad.
2. Almacenamiento en ficheros. Se puede utilizar uno o varios ficheros para almacenar información de manera persistente entre diferentes ejecuciones de programa. Los ficheros en sí, sobrevivirán a diferentes ejecuciones del programa, por lo que la información será persistente, y se podrá leer y escribir en diferentes ejecuciones. Si se utiliza esta opción, se debe seguir un criterio a la hora de almacenar los datos en un fichero, de forma que se puedan recuperar posteriormente siguiendo el mismo criterio.

Con respecto a la información no persistente se puede utilizar tanto memoria estática como dinámica. Para el caso concreto de cada una de las tablas y sus relaciones, se darán unas posibilidades de almacenamiento más adelante en este material.

10.5.1. Preguntas de autoevaluación

Responde a las siguientes preguntas

1. La información persistente es aquella que se almacena después de cada ejecución del programa y puede recuperarse en las ejecuciones siguientes
   - Verdadero
   - Falso
2. Donde se almacena la información persistente?
   - La memoria principal
   - Bases de datos y ficheros
3. Para almacenar los datos en un fichero debe seguirse un criterio, de tal manera que estos puedan recuperarse posteriormente siguiendo el mismo criterio
   - Verdadero

- Falso

10.6. Almacenamiento de información procesada

Es muy habitual en multitud de aplicaciones, que la aplicación deba manipular a menudo información que se obtiene fruto del procesamiento según diferentes operaciones de otra información inicial.

Cuando esto sucede, se puede tomar la decisión de almacenar sólo la información básica a partir de la cual se puede deducir cualquier otra información necesaria, fruto de la manipulación y procesamiento de esta. Si esta es la decisión entonces cada vez que necesitemos información fruto del procesamiento de otra, deberemos realizar el cálculo de esa información procesada a partir de la información básica. En ocasiones, tener que calcular cada vez el procesamiento de la información para obtener otra, puede resultar muy costoso computacionalmente y puede merecer la pena sólo calcularlo una vez e ir actualizándolo sucesivamente y dejar almacenado en otra porción de memoria dicha información procesada.

Otra decisión posible es almacenar no sólo la información básica a partir de la cual se puede deducir la demás, sino almacenar también información procesada que se va a utilizar muchas veces en la aplicación y reservar memoria para ella. De esta forma, ocuparemos más memoria, pero a cambio podremos reducir el tiempo de ejecución destinado al cálculo de la información procesada.

Comparando las dos soluciones, hay que tomar una decisión de compromiso teniendo en cuenta diferentes factores como: cuáles son las limitaciones de nuestra aplicación (en memoria, en tiempo de ejecución, etc.), cuánta más memoria implica tener la información extra procesada, cuánto más tiempo de cálculo implica tener que procesar la información cada vez en lugar de sólo procesarlo una vez y actualizarlo cada vez que hay un cambio (dependiendo de la aplicación esa actualización puede ser más o menos costosa en tiempo, al igual que la operación de procesado en sí puede ser más o menos costosa).

En el caso de nuestra aplicación de referencia, un ejemplo de información que se puede obtener como procesamiento de otra es el número de fotos que ha sacado un determinado autor. Dicha información se puede extraer a partir de las tablas de fotos y autores, recorriendo todas las instancias de fotos y viendo cuales se corresponden con un determinado autor, en cuyo caso se sumará 1. Por lo tanto, se puede tomar la decisión de o bien procesar la información básica para obtener el número de fotos de un autor, o bien almacenar en otra área de memoria el número de fotos que tiene un determinado autor, calcularlo sólo una vez al inicio e implementar en el programa para que cada vez que ese autor añada una foto, se incremente su número de fotos en 1. Esta área de memoria extra, podría ser una expansión de la tabla de autores para incluir este campo de número de fotos. Si bien es información redundante (se puede obtener del procesamiento de otra información), puede resultar beneficioso según la funcionalidad de la aplicación tener esa réplica de la información.

10.6.1. Preguntas de autoevaluación

Responde a las siguientes preguntas

1. Se considera una buena práctica almacenar siempre la información básica a partir de la cual podremos otbener la información procesada
   - Verdadero
   - Falso
2. Qué factores deberían tenerse en cuenta para decidir que información almacenar y cual obtener como resultado del proceso del programa
   - Memoria principal y tiempo de ejecución

>

- Tiempo de ejecución y espacio libre en disco

3. Qué otro nombre recibe la información que se almacena y que se puede calcular a partir de la información básica?
   - Información redundante
   - Información básica

10.7. Realizaciones de tablas de datos en lenguaje C

Tras realizar el modelado completo de una aplicación, finalmente para los datos no persistentes, usualmente se debe traducir a un código concreto en el lenguaje de programación que corresponda, que sea capaz de definir las estructuras de datos necesarias para soportar dichos datos. Así mismo, se deben implementar en el lenguaje de programación concreto, aquellas funciones encargadas de realizar las manipulaciones de datos.

A menudo, la misma información se puede almacenar de diferentes formas utilizando un mismo lenguaje de programación. Por ejemplo, en el lenguaje de programación C, para almacenar información no persistente de una tabla modelada (por ejemplo la de fotos o la de autores), se pueden utilizar entre otros los siguientes métodos:

1. Con una tabla estática de N elementos. El problema de esta solución es que normalmente el número de elementos de la tabla es indefinido, y al poner memoria estática estamos poniendo un límite máximo de N elementos, si hubiera más de esos elementos la aplicación no funcionaría correctamente. Por otro lado, si N es muy grande y el número de elementos actuales pequeños, se estaría utilizando mucha más memoria de la necesaria. En contrapartida, en cuanto a ventajas de esta solución tendremos que las operaciones de búsqueda son mucho más rápidas que por ejemplo una lista enlazada.

2. Con una zona de memoria contigua que puede cambiar de tamaño para ajustarse al número de elementos que realmente hay. En este caso se tendrá un puntero que apunte al principio de esa área de memoria. Todos los elementos de la tabla están en posiciones continuas y no se tiene más memoria de la necesaria reservada, ya que se ajusta la memoria al número de elementos que realmente hay. Para ello, la primera reserva de memoria se hará con un malloc, y seguidamente cuando haya que añadir o eliminar un elemento se hará un realloc para ajustar el tamaño del área de memoria reservada para ajustarla aumentando o reduciendo memoria respectivamente.

3. Con una lista enlazada, en la que cada elemento de la tabla es un elemento de la lista enlazada. Según se van incluyendo nuevos elementos, se insertan a la lista enlazada, y según se van eliminando elementos, se liberan dichas porciones de memoria de la lista enlazada y se enlaza adecuadamente.

4. Con una tabla de hash, teniendo varias listas enlazadas. Cada elemento de la tabla de datos modelada, ocupará una posición de la tabla de hash, según el resultado de realizar una función de hash sobre uno de los campos del elemento. Así se obtendrá el índice de la tabla de hash donde se ubicará dicho elemento. Luego, cada una de las posiciones de la tabla de hash, tendrá su propia lista enlazada. Este método de almacenamiento, hace que en lugar de tener una sola lista enlazada se tengan M (tamaño de la tabla de hash), de forma que los tiempos de búsqueda en la lista enlazada se reducen, a costa de ocupar más memoria. Tal como se vio en una sesión anterior, dicha longitud M se puede aumentar o decrementar según el concepto de densidad.

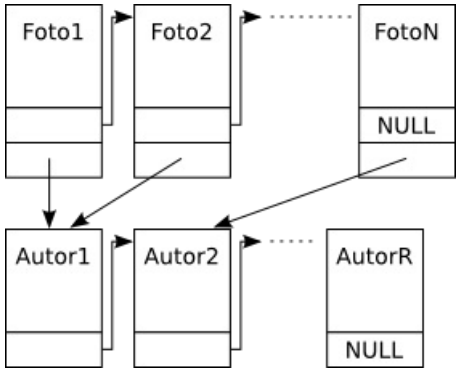
Hasta ahora, se ha comentado en esta sección sobre implementaciones posibles en C de diferentes

Hasta ahora, se ha comentado en esta sección sobre implementaciones posibles en C de diferentes tablas de datos. Por otro lado, es interesante pensar también en cómo implementar la relación entre esas tablas, como por ejemplo las tablas de fotos y autores. Entre los métodos posibles para implementar dicha relación, están los siguientes, que se pueden combinar con cada uno de los 4 métodos que acabamos de ver:

1. Relacionar un elemento foto con su elemento de autor a través de un campo de la tabla que sea un entero, que será el mismo en los dos elementos.

2. Relacionar un elemento foto con su elemento autor a través de un puntero, tal que un campo del elemento foto tendrá un puntero al inicio del área de memoria donde se encuentra la información del autor correspondiente. Así, varias fotos apuntarán a la misma área de memoria correspondiente a su respectivo autor. En este caso, hay que notar que el campo AUTOR_ID no existiría en la tabla de autores, sino que su función se reemplazaría por dicho puntero. Es de notar, que no siempre es necesario que un determinado modelado de datos en genérico mapee luego exactamente todos sus campos en un lenguaje de programación específico, como es este el caso.

Como ejemplo, la siguiente figura muestra el caso en el que fotos y autores se almacenan en una lista encadenadad, y como la relación entre los dos elementos se implementa mediante punteros.



10.7.1. Preguntas de autoevaluación

Para cada afirmación seleccione verdadero o falso

1. En lenguaje C la información no persistente se puede almacenar en la memoria principal de diferentes formas.
   - Verdadero
   - Falso

2. Las tablas estáticas permiten almacenar un número indefinido de elementos.
   - Verdadero

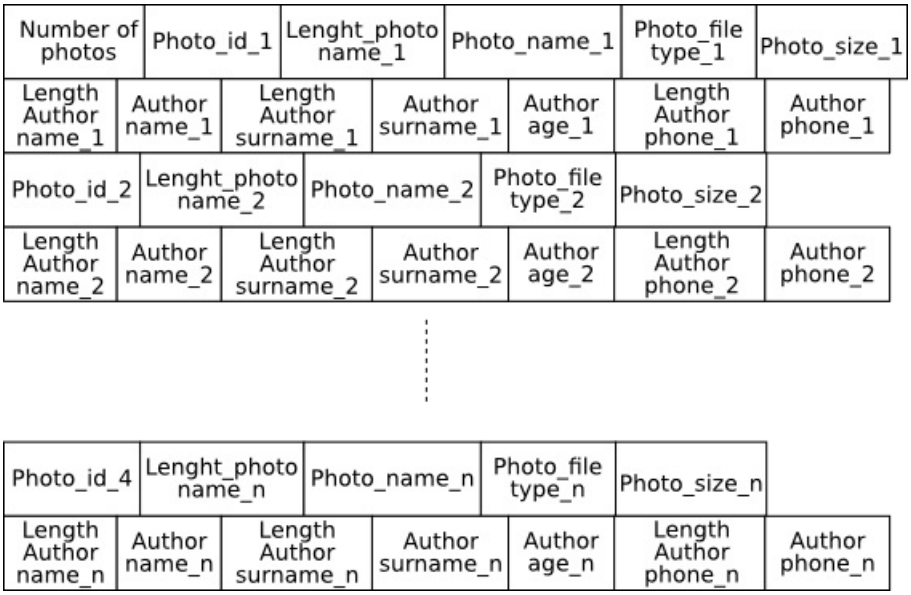
     >
   - Falso

3. Si almacenamos la información en listas enlazadas estas no permiten establecer relaciones entre los elementos de las listas.
   - Verdadero
   - Falso

10.8. Almacenamiento de datos en ficheros

Para almacenar los datos en ficheros de manera persistente, una de las cosas que hay que tener en cuenta es que hay que utilizar ciertos artilugios para delimitar cuando termina un elemento y empieza uno nuevo, cuando termina y empieza un nuevo campo de datos, o cuando ya no hay más elementos en el fichero. Así por ejemplo, si se va a guardar un string the longitud indefinida, una posible solución será añadir un campo adicional que indique la longitud de dicho string en bytes en el fichero.

La siguiente figura, muestra una manera de guardar la información de fotos y autores en un único fichero. Podemos observar que inicialmente se reservan 4 bytes para indicar en el fichero el número total de fotos que tendremos en el fichero, y seguidamente tendremos la información de cada una de las fotos, incluida la información de su autor. Por cada entero, tendremos 4 bytes en el fichero para almacenarlo, pero por cada string tendremos un campo adicional de 4 bytes que indicará la longitud en bytes del string, para delimitarlo.



Existen diferentes maneras de guardar la misma información. Otra manera de tener la información disponible en el fichero, se muestra en la siguiente imagen. Aquí, la información se almacena dividida en autores. De esta forma, tenemos un campo inicial de 4 bytes, indicando el número total de autores almacenados, seguidamente, se tiene la información por cada autor. Para cada autor, tendremos los datos propios de cada autor, y al finalizar los datos asociados a un autor, tendremos un número almacenado en 4 bytes, indicando el número total de fotos que ha realizado dicho autor, para finalmente indicar la información asociada a cada una de esas fotos.

La elección para decidir una de entre varias maneras de almacenar la información en un fichero, depende de un análisis en el que hay que tener en cuenta las ventajas e inconvenientes, y tomar una decisión de compromiso. Así, hay que tener en cuenta qué solución ocupará más espacio en el fichero (en el caso presentado la primera opción ocupa más espacio, ya que hay que replicar la información de un mismo autor varias veces), o qué será más efectivo en cuanto a tiempo de ejecución para las operaciones que requiere la aplicación. Por ejemplo, una aplicación que requiera en multitud de ocasiones listar las fotos de un autor determinado, entonces para dicha operación resultará más efectiva la segunda opción ya que las fotos ya estarían ordenadas por autor, por lo que no tendría que buscar foto a foto, sino que una vez localizado el autor, todo estaría en posiciones contiguas en el fichero. Sin embargo, una aplicación que sólo requiera listar todas las fotos en el sistema con su información, sin importar que estén ordenadas por autor, entonces le puede resultar más efectiva la primera opción, ya que al insertar una nueva foto, la pondrá al final del fichero, sin tener que reubicar la información en el fichero para agruparla con su autor.

10.8.1. Preguntas de autoevaluación

Responde a las siguientes preguntas

Responde a las siguientes preguntas

1. Qué deberías tener en cuenta para guardar los datos en un fichero?
   - El inicio y el final del fichero
   - El inicio y el fin de cada campo, y cuando no hay más elementos

2. Cuando se almacena un string en un fichero es necesario guardar en un campo adicional la longitud del string en bytes en el fichero, para poder recuperar el string guardado
   - Falso
   - Verdadero

3. Cuando se almacena en un fichero un entero y una cadena de cuatro caracteres, se usan:
   - 3 campos
   - 4 bytes

10.9. Independencia del módulo de datos y reutilización

En aplicaciones software, suele ser habitual dividir la aplicación en módulos funcionales cada uno de los cuales son independientes, de forma que se puedan obtener múltiples ventajas tales como:

1. Facilidad de mantenimiento. Si hay por ejemplo un error en un módulo, la resolución se focaliza sólo en dicho módulo y no hay que mirar otras partes del programa, lo cual facilita el mantenimiento del código.
2. Reutilización. Si tenemos encapsulado un determinado módulo y bien definido, dicho módulo puede ser utilizado para otras aplicaciones.

Una de las divisiones funcionales típicas de aplicaciones software consiste en dividir todo código de programa relacionado con la interacción de datos, todo código relacionado con la lógica del programa, y todo código relacionado con la visualización e interacción con el usuario. De esta forma la aplicación final se divide en tres grandes módulos.

Es por lo tanto interesante pensar en cómo dividir dichos módulos en una aplicación, y en primer lugar delimitar la funcionalidad asignada al módulo de datos.

10.9.1. Preguntas de autoevaluación

Responde a las siguientes preguntas

1. Cuales son las principales ventajas de dividir la aplicación en módulos funcionales?
   - Faciliad de almacenamiento, reutilización de los datos
   - Facilidad de mantenimiento, reutilización de los módulos

2. Una división funcional típica de una aplicación incluye tres grandes módulos así: interacción de los datos, lógica del programa y presentación de los datos al usuario.
   - Verdadero
   - Falso

3. La primera actividad para dividir la aplicación en módulos sería delimitar la funcionalidad del módulo de interacción con el usuario
    - Verdadero
    - Falso

10.10. Bibliografía de apoyo

- S. Ceri, P. Fraternali, A. Bongio, M. Brambilla, S. Comai, M. Matera, "Designing Data-Intensive Web Applications", Capítulo 2: Data Model páginas 61-77.