Universidad Autónoma de Madrid

Departamento de Informática

Estructura de Datos
Práctica - 2

Roberto Marabini Ruiz

Índice

1. Objetivo

El usuario final nunca accede directamente a la base de datos (no lanza peticiones SQL), si no que interactúa con una interfaz de usuario que a su vez utiliza una capa lógica (software) que construye las consultas y las lanza. En esta práctica experimentaremos con el acceso a bases de datos desde un programa escrito en lenguaje C. En particular, usaremos la librería ODBC para crear un programa que actuará sobre la base de datos `classicmodels` que hemos proporcionado en la primera práctica. Todo lo aprendido en esta práctica es casi directamente aplicable a cualquier lenguaje de programación que disponga de una implementación de tipo ODBC, JDBC o similar.

2. Acceso a la base de datos desde C

Como ya se comentó, en esta práctica usaremos ODBC para crear un programa en C que ejecute ciertas operaciones en la base de datos. ODBC es una librería para enviar consultas a bases de datos y gestionar los resultados. Podéis encontrar documentación relacionada con ODBC en el URL `http://www.unixodbc.org/`. En particular, es interesante el "Programming Manual tutorial" (`http://www.unixodbc.org/doc/ProgrammerManual/Tutorial/`). Otro enlace a resaltar es "ODBC from C Tutorial Part 1" (`https://www.easysoft.com/developer/languages/c/odbc_tutorial.html`).

Además, se os proporcionan los ficheros `odbc.h` y `odbc.c` que contienen las funciones más comunes de conexión, desconexión e impresión de errores, así como una colección de ejemplos que usan estas funciones.

3. Programa a implementar

Se requiere escribir en lenguaje C un programa que:

- Muestre un menú con 4 opciones

    1. Products.

2. Orders.
3. Customers.
4. Exit.

- Pida por teclado la opción deseada (el número).
- Ejecute la opción del menú seleccionada mientras que el usuario no seleccione la opción 4 (Exit) del menú.

Cuando se elija cualquiera de las tres primeras opciones debe mostrarse un nuevo sub-menú cuyo contenido se describe en detalle a continuación. Nota, en la subsección 3.4 se dan algunas directrices sobre cómo implementar los menús.

3.1. Submenú Products

El submenú "Products" mostrará tres opciones:

1. Stock.
2. Find.
3. Back.

- Stock: Se solicita al usuario un identificador de producto (productcode) y se devuelve el número de unidades del producto en stock.
- Find: Se solicita al usuario el nombre del producto y se devuelve un listado con todos los productos cuyo nombre contenga la cadena suministrada. Cada línea de la salida contendrá dos cadenas `productcode` y `productname`. La salida estará ordenada por el atributo `productcode`. Nótese que la salida puede constar de más de un producto.
- Back: Permite volver al menú anterior.

3.2. Submenú Orders

El submenú "Orders" mostrará cuatro opciones

1. Open.
2. Range.
3. Detail.
4. Back.

- Open: Devuelve un listado con todos los pedidos que todavía no se hayan enviado (fecha no asignada a `shippeddate`). Se mostrará el valor del identificador `ordernumber`. Ordena la salida por `ordernumber`.
- Range: Se solicitan al usuario dos fechas siguiendo el formato YYYY-MM-DD - YYYY-MM-DD y se devuelve un listado con todos los pedidos solicitados entre dichas fechas (`orderdate`). La salida debe contener `ordernumber`, `orderdate` y `shippeddate`. Ordena la salida por `ordernumber`.
- Detail: Se solicita al usuario un identificador de pedido (`ordernumber`) y se devuelve un listado conteniendo los detalles del pedido. La primera línea mostrará la fecha en que se realizó el pedido (`orderdate`) así como si ya se ha enviado (`status`); la segunda línea mostrará el coste total del pedido. A continuación se mostrará, en una línea por producto, el código del producto (`productcode`), la cantidad de unidades solicitadas (`quantityordered`) y el precio por unidad (`priceeach`). La salida de las líneas donde se listen los productos se ordenará por `orderlinenumber`.
- Back: Permite volver al menú anterior.

3.3. Submenú Customers

El submenú "Customer" mostrará cuatro opciones

1. Find.

2. List Products
3. Balance.
4. Back.

- Find: Se solicita al usuario el **nombre de contacto** del cliente y se devuelve un listado con todos los clientes cuyo nombre (`contactfirstname`) o apellido (`contactlastname`) de contacto contenga la cadena suministrada. El listado debe incluir el nombre del cliente (`customername`) y el nombre (`contactfirstname`) y el apellido (`contactlastname`) de contacto asi como el identificador (`customernumber`) del cliente. Ordena la salida por `customernumber`.
- List Products: Se solicita el identificador de un cliente (`customernumber`) y se devuelve un listado con todos los productos solicitados por el cliente en cualquier pedido ("orders"). La salida costará de una linea por cada producto solicitado mostrando el nombre del producto `productname` y número TOTAL de unidades solicitadas. Esto es, si el mismo producto se ha solicitado en varios pedidos debe aparecer una única línea con la suma de todas las unidades solicitadas. Ordenar la salida por `productcode`.
- Balance: Se solicita el identificador de un cliente ("customer") y se devuelve el saldo del mismo, esto es la suma de los pagos realizados menos el valor de la suma de todos los productos comprados.
- Back: Permite volver al menú anterior.

Importante: Para obtener la calificación de 10 deberéis implementar paginación en esta consulta. Esto es, en caso de que el número de resultados sea superior a 10, se mostraran solo 10 resultados por pantalla. Utilizando las teclas "<" y "<", el usuario debe poder moverse a la página siguiente o a la página anterior siempre que estén disponibles.

3.4. Algunas ideas sobre como implementar los menús

En *Moodle* podéis encontrar el fichero `menu.c` que muestra una forma posible de implementar un sistema de menús y submenús. `menu.c` permite al usuario elegir entre una selección de cuentos de hadas ("fairy tales") o de canciones infantiles ("nursery rhymes"). Una vez seleccionado el tópico ofrece un listado de cuentos o canciones entre el cual se puede elegir una en concreto.

Importante: por motivos de seguridad cualquier consulta que incorpore un parámetro introducido por el usuario se realizará usando "prepared statements" (vease odbc-ejemplo4.c)

3.5. Testing

En moodle encontraréis una colección de ficheros de test con extensión `sh` y un contenido similar a:

```
#!/usr/bin/expect -f

set timeout -1
spawn ./menu

expect "Enter␣a␣number␣that␣corresponds␣to␣your␣choice␣>␣"
send -- "1\r"

expect "Enter␣a␣number␣that␣corresponds␣to␣your␣choice␣>␣"
send -- "2\r"

expect "Enter␣a␣number␣that␣corresponds␣to␣your␣choice␣>␣"
send -- "4\r"

expect "Enter␣a␣number␣that␣corresponds␣to␣your␣choice␣>␣"
send -- "3\r"
```

```
expect eof
```

Este fichero lanza el programa llamado `menu`, espera a que aparezca la cadena ``‘‘Enter a number that corresponds to your choice >’’`` por pantalla y envía un número seguido de un salto de línea en respuesta. El proceso se repite 4 veces enviando cada vez los valores 1, 2, 4 y 3 respectivamente.

Estos ficheros de test se usarán para corregir vuestras prácticas, así que es importante que comprobéis que funcionan. En particular, aseguraros de que las cadenas buscadas (`Enter a number that corresponds to your choice >` en el ejemplo) sean impresas por pantalla por el programa y que el formato de salida de las consultas de vuestro programa coincida con el que espera el fichero.

4. Material a entregar

Junto con el programa se debe entregar un fichero makefile que permita:

1. Compilar el código (make compile).
2. Borrar, crear y poblar la base de datos (make all).

Igualmente, se solicita que entreguéis el resultado de ejecutar el comando:

```
splint -nullpass *.c  *.h
```

y esperamos que todos los problemas reportados por esta utilidad los hayáis subsanado. Guardad la salida de splint en un fichero llamado `splint.log` y adjuntar este fichero al material entregado usando moodle.

Si bien un programa escrito en C se podría hacer en un único fichero de texto, cualquier proyecto serio requiere que el código fuente de un programa se divida en

varios ficheros para que sea manejable. Igualmente, se espera que el tamaño de las funciones creadas sea moderado (en número de líneas), que estas funciones estén comentadas y que incluyan el nombre de su creador.

4.1. Resumen del material a entregar tras la segunda semana

Subid a *Moodle* un fichero único en formato zip que contenga:

1. Fichero `Makefile` y todos los ficheros necesarios para crear un programa que que implemente el submenú "products" y satisfaga los tests relacionados con ese submenú. IMPORTANTE: **NO incluyáis el fichero** `classicmodels.sql`.

4.2. Resumen del material a entregar al final de la práctica

Subid a *Moodle* un fichero único en formato zip que contenga:

1. Fichero `Makefile` y todos los ficheros necesarios para crear el programa solicitado y ejecutar los tests. IMPORTANTE: **NO incluyáis el fichero** `classicmodels.sql`. Incluid el fichero `splint.log`
2. Memoria en formato pdf que, por cada opción implementada en el programa, incluya una breve explicación de la lógica implementada, así como del comando SQL utilizado.

5. Criterios de corrección

Para aprobar es necesario:

- Que el código subido a *Moodle* se pueda compilar con el fichero makefile suministrado.
- Tener 5 consultas totalmente correctas (se entiende por consulta las opciones no triviales del menú, esto es, aquéllas que requieren acceder a la base de datos). Estas consultas deben funcionar para cualquier instancia de la base de datos.
- Los ficheros de test correspondientes a las 5 consultas deben finalizar su ejecución.

Para obtener una nota en el rango 5-7 es necesario:

- Satisfacer todos los requerimientos del apartado anterior.
- Las consultas deben ser robustas. Por ejemplo, el programa debe responder adecuadamente cuando se le proporcionan identificadores inexistentes o el resultado de la consulta sea el conjunto vacío.
- Tener 6 consultas no triviales totalmente correctas. Estas consultas deben funcionar para cualquier instancia de la base de datos.
- Los ficheros de test correspondientes a las 6 consultas deben finalizar su ejecución.

Para obtener una nota en el rango 7-8 es necesario:

- Satisfacer todos los requerimientos del apartado anterior.
- Presentar una memoria que esté correctamente redactada, demuestre vuestra comprensión de la práctica y se ajuste a lo solicitado en la práctica.
- Tener 7 consultas no triviales totalmente correctas. Estas consultas deben funcionar para cualquier instancia de la base de datos.
- Los ficheros de test correspondientes a las 7 consultas deben finalizar su ejecución.
- El código, tanto en C como en SQL, debe ser legible y estar comentado.

- El programa debe estar bien estructurado y estar distribuido en varios ficheros.

Para obtener una nota en el rango 8-9 es necesario:

- Satisfacer todos los requerimientos del apartado anterior.
- Que todas las consultas sean correctas.
- Que todos los ficheros de test finalicen su ejecución.
- El código, tanto en C como en SQL debe estar optimizado. Por ejemplo: (1) no se deben realizar dos consultas cuando el mismo resultado puede obtenerse con una única consulta y (2) las consultas deben ser resueltas por la base de datos, esto es, no os traigáis todas las entradas de una tabla y luego implementéis la búsqueda en el programa en C.

Para obtener una nota en el rango 9-10 es necesario:

- Satisfacer todos los requerimientos del apartado anterior.
- Haber implementado la paginación

NOTA: Cada día (o fracción) de retraso en la entrega de la práctica se penalizará con un punto.

NOTA: La ausencia o retraso en la entrega parcial (segunda semana) se penalizará con un punto.

NOTA: La falta de uso de git como sistema de control de versiones se penalizará con -0.5 puntos.

A. *Git*

Os aconsejamos que uséis *Git* como herramienta de control de versiones, concretamente, la implementación de la plataforma *GitHub*. Para ello, lo primero que debéis hacer es conectaros a `https://github.com/` y registraros de forma gratuita en la plataforma. A continuación se muestra una descripción de los principales comandos de *Git*.

Este apéndice muestra una breve descripción del funcionamiento de *Git*. En concreto, aquí no se va explicar la forma de trabajar con ramas, lo que puede ser un mecanismo interesante de conocer por el alumno. Para más detalles sobre el uso de *Git*, se recomienda revisar la documentación disponible en `https://git-scm.com/docs`.

Cuando trabajas con *Git*, lo haces sobre un repositorio en el que se almacena el histórico de versiones de tus ficheros. Existe un repositorio remoto alojado en el servidor de *Git*; y un repositorio local, que es una "copia" del repositorio remoto en un directorio local. Normalmente se empieza con un repositorio remoto clonado en la máquina de trabajo. Una vez descargado, todo el historial, todas las ramas y todas las versiones están disponibles sin conexión. Esto implica una separación entre guardar una versión de uno o varios archivos y subirla al servidor original para que otros puedan obtener esos cambios. La primera acción se llama *commit*, mientras que al acto de enviar esos datos al servidor se le llama *push*.

A continuación se muestran los principales comandos que se le pueden pasar a *Git* para interactuar e intercambiar información entre el repositorio local y el remoto:

- `git clone url_repositorio_remoto`
  Descarga y crea un enlace con el repositorio remoto en el directorio en el que nos encontramos.

- `git add <recurso>...`
  Modifica/crea uno o varios ficheros/directorios en el repositorio. Notar que *Git* obliga a hacer `add` tanto para nuevos elementos del repositorio como para archivos cambiados. Solo los elementos "añadidos" formarán parte del *commit*.

- `git rm <recurso>...`
  Elimina uno o varios recursos del repositorio.

- `git commit -m "Mensaje del commit"`
  Actualiza el repositorio local con todos los cambios realizados.

- `git status`
  Comprueba el estado del repositorio, mostrando tres listas de archivos: archivos modificados, archivos nuevos y archivos a incluir en el commit.

- `git checkout -- <recurso>`
  Sustituye un recurso local por la versión del repositorio remoto.

- `git push`
  Sube los *commits* locales al repositorio remoto. Para tratar los posibles conflictos con los cambios realizados por otros usuarios del repositorio, lo recomendable es actualizar primero el repositorio local.

- `git pull`
  Actualiza el repositorio local con las últimas versiones del repositorio remoto. Al actualizar el repositorio pueden producirse conflictos porque los ficheros incluidos en la actualización han cambiado localmente. Dependiendo del tipo del conflicto es posible que haya editar y corregir el conflicto manualmente, dejando el archivo como debería estar después de aplicar todas modificaciones locales y remotas. Una vez resueltos todos los conflictos los ficheros en conflicto se deben volver a añadir (`git add`) y hacer *commit*.