Data Structures, Task 2

Escuela Politécnica Superior, UAM, 2020–2021

Introduction

The main objective of this task was to implement an interface in c which allows us to search into the database 'classicmodels' using model queries and changing the data inside them (like a product number or a customer name). To do this, the first step was to link our database with our terminal in linux creating a datasource and using the odbc library in c (so the queries could be executed correctly).

Our work is divided in three sections, as are the different menus where you can make searches: products, orders and customers.

Program Implementation

After all the database configuration the next objective was to create a C program to be used as an interface to work with the database. The implemented menus were based on the given examples so they could test the program easily with the test files. After making the menu and the submenus for each section we started working in the implementation (with the different queries). To do this, in all cases we had to, firstly, connect to the database, and then, at the end, disconnect from it correctly.

Products Submenu

This menu shows three different options: stock, find and back (to exit to the main menu).

→ Stock:

Input: Productcode
Output: Quantity in stock of the asked productcode
Query: `select quantityinstock from products where productcode=?;`
Explanation: The function consists of having a productcode as an input and receiving the quantity in stock of that product as an output. The used query is really simple, it goes to the products table and find the result that corresponds with the given productcode, to show it's "quantityinstock" value as an output. To prevent injection security problems, we use the "SQLBindParameter" function instead of directly writing the input into the query. To implement the query we used only one statement.

→ Find:

Input: Productname
Output: List with all the products (whose name contains the supplied string) with the corresponding productcode and productname
Query:
```
select productcode, productname from products where
productname like '%' || ? || '%' order by productcode;
```
Explanation: To make this function work, it is needed a productname to complete the query down below (as the '?'). Once this is completed, the program searches

the code and the name of the products that contain the string (which we call productname) in their name. Finally, that list of products is ordered by the different product codes and printed.

• Orders Submenu

This menu shows four different options: open, range, detail and exit (to exit to the main menu).

→ Open:

Input: None

Output: Order numbers of orders without shipped date. (They haven't been sent yet)

Query: `select ordernumber from orders where shippeddate is null order by ordernumber;`

Explanation: This function is very simple: it gives as an output all the order numbers not sent yet. The query just gives as an output all the order numbers where the shipped date is NULL (it doesn't exist), so it doesn't need an input, and there are no injection problems. The output is written to the command line by the function with a loop that prints the results until there are none left. The implementation only needs one statement.

→ Range:

Input: Two dates in format 'YYYY-MM-DD - YYYY-MM-DD'

Output: List with all the orders that were placed between these dates, with the corresponding order number, order date, and shipped date

Query: `select ordernumber, orderdate, shippeddate from orders where (orderdate <= ? and orderdate >= ?) or (orderdate >= ? and orderdate <= ?) order by ordernumber;`

Explanation: As we did in last queries, to introduce the attributes needed (both order dates), we put them as an '?' in our query and then introduce their value with the format shown before. Once this is completed, the program searches in the database the order number, order date and shipped date of the orders placed between the two dates entered. The output orders are ordered by their order number.

→ Detail:

Input: Ordernumber

Output: List with all the details about the order which corresponds with that order number (order date and status, total cost and products in the order with their corresponding code, quantity ordered and price per each of them)

Queries: `select o.orderdate, o.status, sum(od.quantityordered*od.priceeach) as amount from orderdetails od, orders o where o.ordernumber =? and od.ordernumber = o.ordernumber group by o.orderdate, o.status;`

```
select productcode, quantityordered, priceeach from
orderdetails where ordernumber = ? order by
orderlinenumber;
```

Explanation: In this function we have two different queries, as we want to search for different types of data in different lines of the output. The first one searches for all the order dates and status of a given order number, also for the total amount of money of the specified order (with order number), making the product of the quantity of products ordered and the price per each product. The last one gives us the different products, the quantity ordered and price per each of them (for that specific order number), ordering them by their order line number. The first one could be divided in two, but as we want the most optimized queries as possible, we made one for both consults.

- ## Customers Submenu

This menu shows four different options: find, list of products, balance and exit (to exit to the main menu). This submenu also has a paging system implemented in the find and list of products functions that may make the test files to not work correctly. To exit this mode while using it use 1 as the input. More explanation is given in the "Find" function explanation.

→ Find:

Input: String with a part of a contact firstname or lastname, offset

Output: customernumber, customername, contactfirstname, contactlastname

Query:
```
select customernumber, customername, contactfirstname,
contactlastname from customers where (contactfirstname
like '%' || ? || '%') or (contactlastname like '%' || ? ||
'%') limit 10 offset ?;
```

Explanation: This function needs as an input part of a contact name as an input. The query is going to find all the results in table customers where the "contactfirstname" or "contactlastname" columns contain the given input. This is done by using the '%' before and after the input, that means it will find all the results even if they have something written before or after the string. The || is used to merge the '%' with the query. Then the query returns the customer number, customer name, and the contact firstname and lastname of the results. The query has one more input, the offset, because in this case the function will only show you the first ten results, and you can use the < and > keys to use the query again with a different offset to show you the next results (with an increment of 10). To end this loop instead of using < and > you should write '1' as the input. This paging system makes the test file for this function to not work correctly. The function also prevents injection like the others.

→ Products:

Input: Customer number

Output: List with all the products requested by that client (with customer number '?') counting on all of the orders placed. One product per line with its corresponding name, and units requested in total.

Query:
```
select p.productname, sum(od.quantityordered) from
products p, orderdetails od, orders o where
o.customernumber = ? and o.ordernumber = od.ordernumber
and od.productcode = p.productcode group by
p.productname, p.productcode order by p.productcode limit
10 offset ?;
```

Explanation: This function only needs one statement for it to work. The query searches for all the products that appear in any of the orders placed by the specified customer. Then, with that product codes, it searches for the name of each of them in products table, and grouping them by their name/code, makes the summation of how many times they appear on order details table (counting the total amount of products ordered). Finally, we use the offset as a limit for the number of results shown and the < and > keys to use the query again with a different offset to show you the next results (with an increment of 10). The products are printed ordered by their product code.

→ <u>Balance:</u>

Input: Customer number, offset

Output: balance of the customer (payments - price of orders)

Query:
```
select (select sum(pa.amount) from payments pa where
pa.customernumber = ?) - (select
sum(od.quantityordered*od.priceeach) from orders o,
orderdetails od where o.customernumber = ? and
o.ordernumber = od.ordernumber);
```

Explanation: The function asks for a customer number, that is written in the query by the safe from injection method explained before. The query is divided in two subqueries: The first one sums all the amounts of the table payments with the given customer number, and the second one sums all the quantity ordered times the price each of each of the orders where the customer number is the given one. Then the query subtracts the second result from the first one finally giving the balance of that customer. These function also uses the paging method explained in the find function

• Conclusion

In this lab we have learned how interaction with databases work through a user interface and how the odbc implementation works. We also learned about the injection security problem and about user interface creation. Also, an important part of this task was learning how to link a database with our terminal in linux.