DD1334 Databasteknik

Laboration 2: Application Programing and Functional Dependancy

Andreas Gustafsson, Hedvig Kjellström and Michael Minock and John Folkesson

The purpose of Laboration 2 is to learn simple application programing with databases and how to model relational data structures based on properties of the data. You will learn 1)How to write code to interface with a database 2) how to create relational schemas using E/R diagrams;
The recommended reading for Laboration 2 is that of Lab 1 plus Lectures 1, 8 and 9.

Laboration 2: ***Come prepared to the review session!*** *Only one try is allowed – if you fail, you are not allowed to try again for a higher grade that session. The review will take 20 minutes or less, so have all papers in order. Passing requires:*

Completed Task 1 with an interface function that does take correct input from the user and makes insertions and deletions to the database, Task 2 with a customer interface that allows customers to see their own shipments but not any more (ie no injections). The program should not crash due to incorrect inputs by the user. Task 3 with a shipment interface that does not allow shipments if the book is not in stock and which keeps track of the stock by decrementing it. If the shipment is not inserted the stock should not me decremented and visa versa. The insertion of shipments using the interface should be made safe from concurrent insertions in different processes by using transactions. Task 4 with an E/R diagram with at least 3 relations, that can store all data as described in Modeling Problem, and found at least 3 correct dependencies.

The grades are based on lateness from the date due. See the Lab Grading page in bilda contents for the due dates for the labs and the grading of late assignments.

Laboration 2 is intended to take 30h to complete.

Task 1: Basic Application Program

Create your own private copy of Booktown as described in lab 1.

In the course directory, /info/DD1334/dbtek14/labbar/lab2/ in the NADA tree, there are two code skeletons, customerInterface.py, shipmentInterface.py and interface.py. Start with interface.py and complete the two unfinished methods: *insert*, and *remove*. The program should handle correct user input and you need not worry about other cases here. You only need to show that you can insert or remove tuples from the tables. You will find helpful comments in the skeletons and are encouraged to seek help as needed from the web or you classmates.

Before the review: Comment your code so that you easily can explain what it does. Use descriptive variable names.

At the review: Show the code to the assistant and prepare an execution so that the assistant easily can test the interface. Both group members should be prepared to answer questions about the code.

The code skeletons provided are designed to run in Eclipse IDE on a NADA Ubuntu machine. You are welcome to work on your own machine, but no guarantees are made that they will function out of the box on any computer. The skeletons require a database connector.

For Python on Ubuntu, python-pygresql should work. The simplest way to run the code is to log on to u-shell.csc.kth.se as described above. The python skeleton is extensively commented to explain the code at a tutorial level as well as giving links to web resources that may help in extending it to the other 2 methods.

Task 2: Simple Application Program

Now you should complete the customerInterface.py skeleton. Here we pretend to give the interface to customers to do queries themselves. As 'security' we demand they provide a customer_id and name that matches our database. They can then see that customer's shipment data. You task is to get the input of customer_id and name from the user, check it against the customers table and then print a listing of (shipment_id, ship_date, isbn, title) tuples. Now the program should not crash for incorrect input and should not allow SQL injection attacks. There are many hints in the comments.

Injections are attacks where unanticipated inputs given by the crafty user allow unauthorized access to the database. Protection against these is thus focused on the user input. One protection is to strongly type cast the input, so if you expect an integer you should put the input into an integer variable. Then trying to input text will raise an exception which can be caught. As in

```
try:
            variblename = int(input("promt: "))
except (NameError,ValueError, TypeError,SyntaxError):
            print("That was not a number.... :(")
            return
```

Notice that python cares about indents. Another way to protect the input is using the built in function to remove escape characters such as ' or \:

```
variablename= pgdb.escape_string(raw_input("promt: ").strip())
```

Task 3: Not as Simple Application Program

Now you should complete the shipmentInterface.py skeleton. Here we pretend to give the interface to employees to enter shipments. We want to be sure that simultaneous insertions by different employees at different computers will not put the stock into an inconsistent state. Most of the code is in place but you need to finish the makeShipments function. You should do the same sort of type casting, escape removal and exception raising as on task 2. Here the main new issue is transactions. A transaction should be started at the right point in the code, rolled back if there is any problem and committed if all goes well.

`self.conn.commit()` will both commit the current transaction and start a new one.
`self.conn.rollback()` will do the rollback to the last commit.

The code should ask for the shipment information then test if there is a book to ship then if so insert the shipment and decrement the database. Any failure should not change anything. It should not be possible for one user to check if there is a book then another remove the book from stock before the first user has done all the changes to stock and shipments.

Be prepared to explain why the transactions are started, rolled back, and committed where they are. Also to answer questions like what if we started a new transaction here, (pointing to some line in your code).

Task 4: Modeling with E/R Diagrams

Now for something else altogether.

A historical museum would like to build a database over their collections. Up until now, all information has been kept in paper binders with one section per collection item. For each item, the following details are stored:

Name	Description
item_id	A number uniquely identifying the item within the museum.
discovery	One paper for each place where the item has been (re)discovered over the years, with information about names of the town, region, country and date of discovery. Although most items are only found once and the placed in the museum, som items have over time, disappeared (e.g. during a war), and then been rediscovered before they end up in the museum. The catalogue must therefore allow storage of several discoveries.
finder	The name of the person that discovered the item, could be different persons for different discoveries (see above).
document	Generic name for any sketch, photo, notes, etc, of the object, that is *not* the main document (first page in the section about this object in the binder).
department	The museum department where the item is currently located (could also be the inventory if the object is not currently on display).
location	The specific place (museum stand) within the department where the item is currently located (or the location within the inventory if the object is in the inventory).
date	The date when the object was moved to its current location in the museum.

The museum would like to computerize their data. They first hire a programmer that did not take a course in relational databases, who creates a computer registry consisting of one large table with the following columns:

item_id, item_name, item_text, discovery_town, discovery_region, discovery_country, discovery_date, discovery_finder, document_id, document_text, dept_name, dept_location, dept_date

It is soon discovered that this is not a good idea, and therefore you are hired to clean the mess up!

Use E/R diagrams to construct a database schema of at least three relations, suitable for storing the museum's data. You can start off with the column names in the single large table above and group the different attributes according to the description above, using the design process described in Garcia-Molina, Sections 4.1-4.5.

From the description above, identify at least three functional dependencies among the attributes (column names). Are the relations in the database schema able to represent the dependencies you found? (Violations to this could be that there are more than one relational schema per dependency, or more than one dependency per relational schema.)

Before the review: Write down three reasons why one big table is not a good idea. Then, document all steps in your design process. Write down the functional dependencies and note in your database schema if there are violations of the dependencies.

At the review: Show all documentation of the design process, including the final E/R diagram and database schema, to the assistant. Both group members should be prepared to answer questions about the design process, and why it is not good with one big table. You should then present the set of functional dependencies, and describe what dependencies are violated by the database schema.