EDAF20: Laboratory Assignments

Department of Computer Science
Lund Institute of Technology

February 5, 2016

Notice:

- The course contains three compulsory laboratory exercises.
- The labs are mostly homework. Before each lab session, you must have done all the assignments in the lab, written and tested the programs, and so on. Contact your teacher if you have problems solving the assignments.
- Smaller problems with the assignments, e.g., details that do not function correctly, can be solved with the help of the teaching assistant during the lab session.

The labs are about:

1. Using SQL.
2. Designing and implementing a database.
3. Developing of a Java interface to the database in lab 2.

Also note:

- You are supposed to work in groups of two people. You should sign up for the labs online. Follow the instructions on the course home page.
- You need a MySQL account to do the labs (one account per group). These are handed out as mentioned during the first lecture.
- Exercise session 1 and 2 are devoted to preparations for lab 2, and the results will also be used during lab 3. You have to prepare for lab 1 on your own.
- One extra lab occasion is organized only for students who are ill during a lab. Please notify your teacher *before* the lab in case you are unable to attend the lab due to illness.

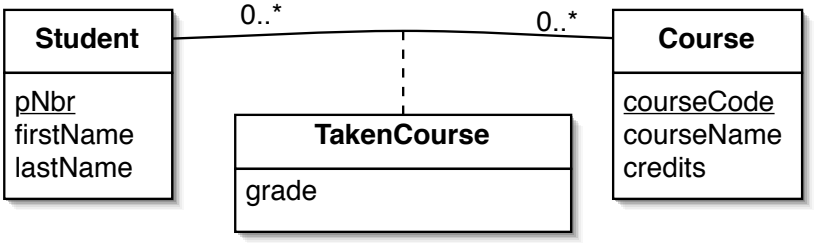
1 Lab Session 1

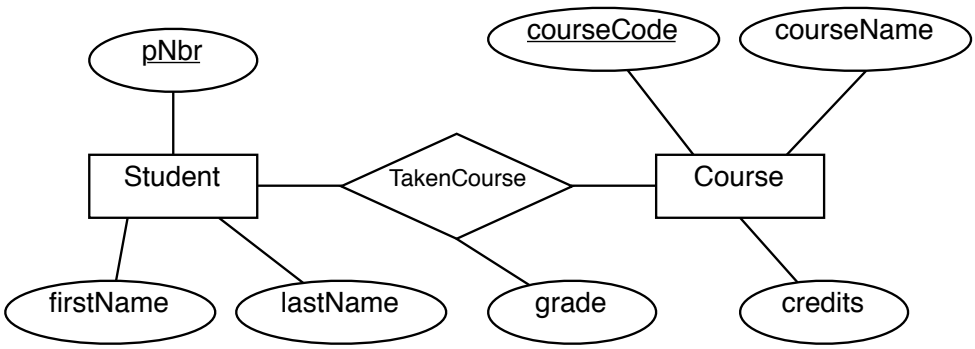
Objective: learn to write SQL queries and practice using the MySQL client.

1.1 Background

A database for registration of courses, students, and course results has been developed. The purpose was not a new LADOK database (Swedish university student database), so everything has been simplified as much as possible. A course has a course code (e.g., EDAF20), a name (e.g., Database Technology), and a number of credits (e.g., 5). Students have a person number (Swedish civic registration number) and a name. When a student has taken a course his/her grade (3, 4 or 5) is registered in the database.

We started by developing an E/R model of the system (E/R stands for Entity-Relationship). This model is developed in almost the same manner as the static model in object-oriented modeling, and you make use of the same kind of UML diagrams. You may instead use traditional E/R notation, as in the course book. However, diagrams in the traditional notation take more paper space, and the notation is not fully standardized, so we will only use the UML notation. The model looks like this in the different notations (the UML diagram is at the top):





The E/R model is then converted into a database schema in the relational model. We will later show how the conversion is performed; for now we just show the final results. The entity sets and the relationships have been converted into the following relations (the primary key of each relation is underlined):

```
Students(pNbr, firstName, lastName)
Courses(courseCode, courseName, credits)
TakenCourses(pNbr, courseCode, grade)
```

Examples of instances of the relations:

pNbr	*firstName*	*lastName*
790101–1234	Bo	Ek
770403–4321	Eva	Alm
791223–4443	Anna	Eriksson

courseCode	*courseName*	*credits*
EDA016	Programmeringsteknik	5
EDA027	Algoritmer och datastrukturer	5
EDA250	Helhetsbild av datatekniken	6

pNbr	*courseCode*	*grade*
770403–4321	EDA016	4
770403–4321	EDA027	3
760114–2952	EDA016	3

The tables have been created with the following SQL statements:

```
create table Students (
         pNbr char(11),
         firstName varchar(20) not null,
         lastName varchar(20) not null,
         primary key (pNbr)
);

create table Courses (
         courseCode char(6),
         courseName varchar(50) not null,
         credits integer not null check (credits > 0),
         primary key (courseCode)
);

create table TakenCourses (
         pNbr char(11),
         courseCode char(6),
         grade integer not null check (grade >= 3 and grade <= 5),
         primary key (pNbr, courseCode),
         foreign key (pNbr) references Students(pNbr),
         foreign key (courseCode) references Courses(courseCode)
);
```

All courses that were offered at the Computer Science and Engineering program at LTH during the academic year 2002/03 are in the table Courses (the course names have in some

cases been abbreviated to make them shorter than 50 characters). Also, the database has been filled with invented data regarding students and their taken courses. SQL statements like the following have been used to insert the data:

```
insert into Students values('790101-1234', 'Bo', 'Ek');
insert into Courses values('EDA016', 'Programmeringsteknik', 5);
insert into TakenCourses values('770403-4321', 'EDA016', 4);
```

1.2 Assignments

1. Study the sections on SQL in the textbook (sections 7.1–7.19 and chapter 8 in [PMR05], alternatively sections 6.1–6.4 and 6.7.1–6.7.3 in [UW02]). These sections contain more than you will use during this exercise, so it may be a good idea to study assignment 3 in parallel.

2. Read the introduction to MySQL (see separate instructions).

3. Write SQL queries for the following tasks and store them in a text file. Format the SQL code according to the rules (`select` on one line, `from` on one line, `where` on one line, ...). Don't use tabs to indent the SQL code, MySQL uses tabs for auto-completion of table names and attribute names.

   The tables Students, Courses and TakenCourses already exist in your database. If you change the contents of the tables, you can always recreate them with the following command (the script `makeLab1.sql` is available on the course homepage):

   ```
   mysql> source makeLab1.sql
   ```

   After most of the questions, there is a number in brackets. This is the number of rows generated by the question. For instance, [72] after question a) means that there are 72 students in the database.

   (a) What are the names (first name, last name) of all the students? [72]

   (b) Same as question a) but produce a sorted listing. Sort first by last name and then by first name.

   (c) Which students were born in 1975? [2]

   (d) What are the names of the female students, and which are their person numbers? The next-to-last digit in the person number is even for females. The MySQL function `substr(str,m,n)` gives `n` characters from the string `str`, starting at character `m`, the function `mod(m,n)` gives the remainder when `m` is divided by `n`. [26]

   (e) How many students are registered in the database?

   (f) Which courses are offered by the department of Mathematics (course codes `FMAxxx`)? [24]

   (g) Which courses give more than five credits? [21]

(h) Which courses (course codes only) have been taken by the student with person number 790101-1234 ? [7]

(i) What are the names of these courses, and how many credits do they give?

(j) How many credits has the student taken?

(k) Which is the student's grade average on the courses that he has taken?

(l) Same questions as in questions h)-k), but for the student Eva Alm. [5]

(m) Which students have taken 0 credits? [16]

(n) Which student has the highest grade average? Advice: define and use a view that gives the person number and grade average for each student.

(o) List the person number and total number of credits for all students. Students with no credits should be included in the list! [72]

(p) Same question as in question o) but with names instead of person numbers.

(q) Is there more than one student with the same name? If so, who are these students? [7]

4. If you haven't picked up your MySQL account before the lab, you will be given your username and password at the lab, when you sign for it.

5. Log in to MySQL (see separate instructions). Change your MySQL password immediately.

6. Use `mysql` to execute the SQL queries that you wrote in assignment 3 and check that the queries give the expected results. Advice: open the file containing the queries in a text editor and copy and paste one query at a time, instead of writing the queries directly in `mysql`.

2 Lab Session 2

Objective: learn to design and to implement a database. This involves creating an E/R model for the application and converting the model into a relational model. You will also learn to create SQL tables and to insert data into the tables. During lab 3 you will develop a Java interface to the database.

Exercise sessions 1 and 2 (partially) are devoted to preparations for this lab.

2.1 Background

A database contains information regarding ticket reservations for movie shows. To make a reservation you must be registered as a user of the system. In order to register you choose a unique username and enter your name, address, and telephone number (the address is optional). When you use the system later, you just have to enter your username.

In the system, a number of theaters show movies. Each theater has a name and a number of (unnumbered) seats. A movie is described by its name only. (In a real system you would, naturally, store more information: actor biographies, poster images, video clips, etc.)

A movie may be shown several times, but then during different days. This means that each movie is shown at most once on any day.

A user can only reserve one ticket at a time to a movie show[1], and cannot reserve more tickets than are available at a performance. When a user makes a reservation for a ticket he receives a reservation number that he uses when he picks up the ticket.

2.2 Assignments

1. Study the sections on E/R modeling and conversion of the E/R model to relations in the textbook (chapters 2 and 6 in [PMR05], or alternatively chapter 2 and sections 3.1–3.2 in [UW02]).

2. Develop an E/R model for the database that is described above. Start by finding suitable entity sets in the system. For this, you may use any method that you wish, e.g., start by finding nouns in the requirements specification and after that determine which of the nouns that are suitable entity sets.

3. Find relationships between the entity sets. Indicate the multiplicities of the relationships.

4. Find attributes of the entity sets and (possibly) of the relationships. Consider which of the attributes that may be used as keys for the entity sets. Draw a UML diagram of your model.

5. Convert the E/R model to a relational model. Use the method that has been described during the lectures and chapter 6 in [PMR05] (or sections 3.1–3.2 in [UW02]). Describe your model on the form Relation1(attribute, ...), Relation2(attribute, ...). Identify primary keys and foreign keys. About primary keys: a movie name and a date together

[1] If the user wants several tickets for the same performance he must make several separate reservations.

suffice to identify a movie show, since each movie is shown at most once on one day. This means that {movie name, date} is a key of the relation that describes movie shows. If you convert the E/R model according to the rules, it may happen that the key also contains the name of the theater. (That the name of the theater should not be a part of the key is indicated by the *functional dependency*
movieName date → theaterName, which means that you can deduce the theater name if you know the movie name and the date. We will discuss functional dependencies later in the course.)

Additionally, relations must be *normalized* to avoid redundancy and anomalies in the database. We omit normalization for now, since we haven't discussed it yet. (Also, relations usually are reasonably normalized if you start with a good E/R model.)

6. Study the sections in the textbook (7.21–7.23 in [PMR05] or 6.5–6.6 and 7.1–7.2 in [UW02]) on SQL statements to modify, create tables and key constraints.

7. Write SQL statements for the following tasks, and execute the statements in `mysql`:

   (a) Create the tables. Don't forget primary keys and foreign keys. Insert data into the tables. Invent you own data with real-world movie names and theater names. Use the data type DATE for dates, entered and displayed in the form '2006–12–24'. Advice: write the SQL statements in a text file with the extension `.sql`. Execute the statements with the `mysql` command
   `source filename`
   The file should have the following structure:

   ```
   -- Delete the tables if they exist. To be able to drop
   -- them in arbitrary order, disable foreign key checks.
   set foreign_key_checks = 0;
   drop table if exists Users;
   ...
   set foreign_key_checks = 1;
   -- create the tables
   create table Users ( ... );
   ...
   -- insert data into the tables
   insert into Users values(...);
   ```

   (b) List all movies that are shown, list dates when a movie is shown, list all data concerning a movie show.

   (c) Create a reservation. Advice: reservation numbers can be created automatically by specifying the number column as an auto-increment column, like this:

   ```
   create table Bookings (
        nbr integer auto_increment,
   ... );
   ```

When you insert rows into the table and don't give a value for the auto-increment column `nbr` (or specify it as 0 or null), it will be assigned the values 1, 2, ... When a ticket is reserved a new row must be inserted into the reservation table, and the number of available seats for the show must be updated. Before you do this you have to check that there are seats available for the show. It is not easy to check this in pure SQL, so we save this for lab 3, when you will code in Java and have access to `if`-statements.

8. Check that the key constraints that you have stated work as intended: try to insert two movie theaters with the same name, insert a movie show in a theater that is not in the database,...

9. Consider the following problem: when you make a ticket reservation you first check that seats are available for the show, then you create a reservation, then update the number of available seats. Which problems can arise if several users do this simultaneously? Draw a diagram which illustrates the problem.

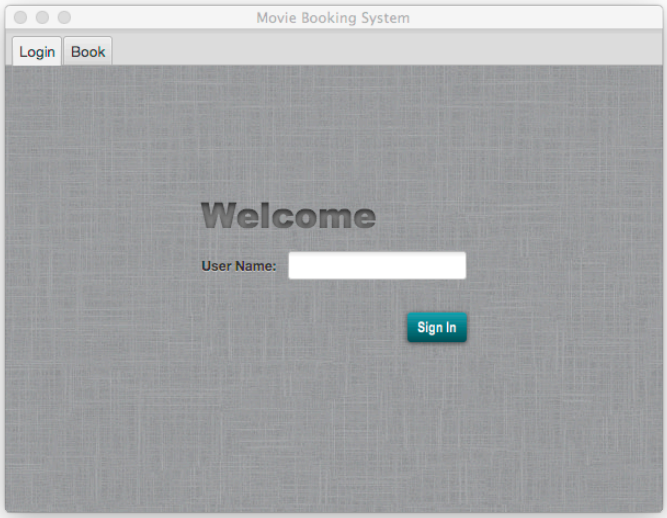
10. What facilities must be offered by a database manager to make it possible to solve problems of the kind described in task 9?

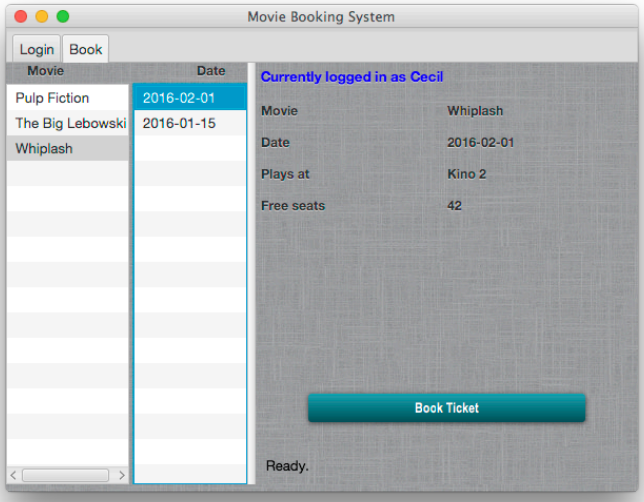
3 Lab Session 3

Objective: you will learn to use JDBC to communicate with a database from a Java program. You will also get to practice designing a graphical user interface with the JavaFX framework.

3.1 Background

A program which makes it possible to interactively make ticket reservations[2] for movie shows uses the database which you developed during lab 2. The program has a graphical user interface. The program is a stand-alone application, and the user must have the application installed on his/her own computer. The user interface for the program is shown below:





The window has two tabs: `Login` and `Book`. The first tab is used when a user[3] logs in to the system with his user name, the second tab is used to make ticket reservations.

The reservation tab has two lists. In the left list are the titles of the movies currently showing. When one selects a movie, the corresponding show dates are shown in the right list. When one selects a date, information regarding the show is displayed in the text fields to the right. When one clicks the `Book Ticket` button a reservation for the show is made, if there are available seats. An error message or dialog box should be displayed if there are no available seats.

When a user makes a reservation, one receives a reservation number, which should be displayed (preferably in the status text at under the booking button). The number of available seats should be updated on each reservation.

[2]The program only handles new reservations. All other tasks concerning the database, e.g., creation of new shows and creation of new users, are performed by other programs. In your case, you will use `mysql` to perform such tasks.

[3]Note: the "user" here is the user of the ticket reservation system, who chose a user name when he registered in the system. Do not confuse this user with the database user, who must log in to the database system with his MySQL username and password.

3.2 Assignments

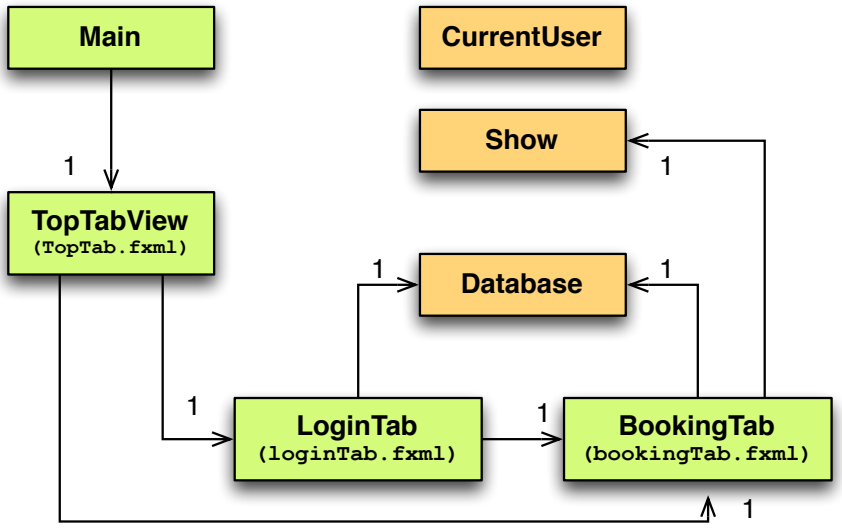
1. Read about JDBC in the textbook [PMR05] section 20.8 (better covered in section 8.6 in [UW02]) and in the overhead slides. Further information on JDBC is available through the course home-page links.

2. A large part of the program for this system is already written, implementing a minimal functionality which you will need to extend. The code is organized in two packages: `gui` package containing the classes and *fxml* implementing the user interface, and `datamodel` package dedicated to the database specific operations. See appendix A for details.

   The source is provided to you as a compressed ECLIPSE project, *lab3FilesFX.zip* available for download from the course home-page.

3. Your task is to complete the program by adding or replacing code at the points specified in the source by a `/* --- TODO: --- */` comment. This will mean filling the simplistic action-handling methods in the GUI and the `Database` class with meaningful code.

   The classes in the program are grouped into two packages, one implementing the GUI and one implementing the database connectivity. Shown below is a simplified UML diagram of the classes. The diagram is schematic and only shows the most important classes and associations. Study the pictures of the user interface (under Background, above) and the Java code (appendix A), and compare with the following description:



   In the `gui` package:

   `Main` is the main program, loading the top view specification.

   `TopTabView` is the controller for the top view associated with `TopTab.fxml`, which embeds the login and booking tabs.

   `LoginTab` is the controller for the tab used to log into the system, with the layout described by `loginTab.fxml`.

`BookingTab` is the controller for the other tab, which helps the user to make a reservation to a movie show. Its layout is described by `bookingTab.fxml`.

In the `datamodel` package:

`Database` should encapsulate all communication with the database system.

`CurrentUser` is a singleton class, which keeps track of the user logged into the system.

`Show` is models a single show, and is used to pass information between the `Database` and rest if the application.

Using `.fxml` GUI descriptions allows for an easy way to separated application specific functionality from layout descriptions. The Java code associated with each control focuses on handling the events rather than building the interface.

The login tab controller handles the submit button pushed by the user once the user name and password were introduced. Its functionality should be extended with checking the data introduced against the database.

The reservation tab contains two lists: one for movie names, `moviesList`, and one for performance dates, `datesList`. The area on the right contains strings and text fields used to display information regarding the selected movie show. The bottom area contains the reservation button and a message line. Listeners for handling GUI events related to the `moviesList`, `datesList` and `bookTicket` button are specified programmatically during the initialization of the `BookingTab`.

4. In the program, the following tasks shall be performed:

   (a) When you click the Login button: log in with the specified username.

   (b) When you enter the reservation panel: show movie names in the movie name list.

   (c) When you select a movie name: show performance dates for the movie in the date list.

   (d) When you select a performance date: show all data concerning the performance in the text fields.

   (e) When you click Book ticket: make a ticket reservation. Make sure you report the booking number to the user!

   Consider in detail the tasks to be performed in the database for each of the actions above.

   JDBC calls are used for the communication between the program and the database system. You should **not** have a tight coupling between the user interface and the database communication, so you must collect all JDBC calls in a class `Database`. Parts of this class are prewritten.

   Specify more methods in the class `Database` to perform the tasks in 4a–4e. When you need to show information concerning a performance (task 4d), you have to fetch the information from the database. The already provided `Showclass`, which has the same

attributes as the corresponding table in the database, helps you do this. Feel free to modify this class or add your own classes in the `datamodel` package.

5. The `BookingTab` uses a special function to figure out the titles of the movies to be displayed in the *Movies* list. This method, called `fillNamesList`, implements only a dummy behaviour, reporting a static list of names. You need to implement this method properly, by adding and calling the appropriate method in the `Database` class.

6. Similarly for the *Dates* list, using the `fillDatesList`. Implement and call the corresponding methods in the `Database` class.

   To fetch a date column from a JDBC result set, use the method `getString()` (all you want to do with the date is to display it).

7. To compile and test the program use either ECLIPSE[4] (recommended) or `javac` in a command shell[5]. Do not forget to test the case when you try to reserve a ticket for a fully-booked performance; in that case you should receive an error message. It is a good idea, for debug purposes, to print out all the queries you issue (e.g. using `System.err.prinln`).

   The program needs access to the MySQL JDBC driver Connector/J, namely the `com.mysql.jdbc.Driver` class, which is located in the *mysql-connector-java-5.1.38-bin.jar*. In the provided archived ECLIPSE project, this *.jar* is already present and added to the path (using project *Preferences/Java Build Path/Libraries Add External Jars...*). When running the program from the command line:

   ```
   java -cp .:downloadpath\mysql-connector-java-5.1.38-bin.jar Main
   ```

   Alternatively, you can set CLASSPATH as shown below. Then, the program may be executed with `java MovieBooking`:

   ```
   set CLASSPATH=.:downloadpath\mysql-connector-java-5.1.38-bin.jar
   ```

References

[PMR05] T. Padron-McCarthy and T. Risch. *Databasteknik*. Studentlitteratur, 2005.

[UW02] J. D. Ullman and J. Widom. *A first course in database systems*. Prentice Hall, 2002.

[4] More about ECLIPSE IDE (Integrated Development Environment) during the labs.
[5] Invoked in Windows by typing *cmd* in the START/RUN... dialog

Appendix A — Classes, Lab 3

gui/Main.java	page 13
gui/TopTabView.java	page 14
gui/LoginTab.java	page 14
gui/BookingTab.java	page 15
gui/TopTab.fxml	page 18
gui/loginTab.fxml	page 18
gui/bookingTab.fxml	page 19
datamodel/CurrentUser.java	page 20
datamodel/Database.java	page 21
datamodel/Show.java	page 22

Listing 1: gui/Main.java

```
package gui;

import datamodel.Database;
import javafx.application.Application;
import javafx.fxml.FXMLLoader;
import javafx.scene.Parent;
import javafx.scene.Scene;
import javafx.scene.control.Alert;
import javafx.scene.control.Alert.AlertType;
import javafx.stage.Stage;


public class Main extends Application {

        private Database db = new Database();

        @Override
        public void start(Stage primaryStage) {
                try {
                        // BorderPane root = new BorderPane();
                        FXMLLoader loader = new FXMLLoader(getClass().getResource("TopTab.fxml"));
                        Parent root = loader.load();

                        Scene scene = new Scene(root,600,440);
                        scene.getStylesheets().add(
                                getClass().getResource("application.css").toExternalForm());

                        // obtain main controller
                        TopTabView wc = (TopTabView) loader.getController();
                // make the database object visible to the controller
                wc.setDatabase(db);

                        primaryStage.setTitle("Movie Booking System");
                        primaryStage.setScene(scene);
                        primaryStage.show();

                        // opening database connection
                /* --- TODO: change xxx to your user name, yyy to your passowrd --- */
                if(!db.openConnection("xxx", "yyy")) {
                    Alert alert = new Alert(AlertType.ERROR);
                    alert.setTitle("Database error");
                    alert.setHeaderText(null);
                    alert.setContentText("Could not connect to the database!"++
                        " Check console for details.");
                    alert.showAndWait();
                }
                } catch(Exception e) {
                        e.printStackTrace();
```

```
                }
        }

        public static void main(String[] args) {
                launch(args);
        }

        public void stop() {
                // close the database here
                db.closeConnection();
                try {
                        super.stop();
                } catch (Exception e) {
                        e.printStackTrace();
                }
        }
}
```

Listing 2: gui/TopTabView.java

```
package gui;

import javafx.scene.Parent;
import datamodel.Database;
import javafx.fxml.FXML;

public class TopTabView {
        @FXML private Parent aLoginTab;
        @FXML private LoginTab aLoginTabController;

        @FXML private Parent aBookingTab;
        @FXML private BookingTab aBookingTabController;

        public void initialize() {
                System.out.println("Initializing TopTabView");

                // send the booking controller reference to the login controller
                // in order to pass data between the two
                aLoginTabController.setBookingTab(aBookingTabController);
        }

        public void setDatabase(Database db) {
                aLoginTabController.setDatabase(db);
                aBookingTabController.setDatabase(db);
        }
}
```

Listing 3: gui/LoginTab.java

```
package gui;

import datamodel.CurrentUser;
import datamodel.Database;
import javafx.event.ActionEvent;
import javafx.fxml.FXML;
import javafx.scene.text.Text;
import javafx.scene.control.TextField;

import javafx.scene.control.Alert;
import javafx.scene.control.Alert.*;

// controller for both the top tabs and login tab!

public class LoginTab {
    @FXML private Text actiontarget;
    @FXML private TextField username;
```

```
    private BookingTab bookingTabCtrl;
    private Database db;

    @FXML protected void handleSubmitButtonAction(ActionEvent event) {

        if(!db.isConnected()) {
                // inform the user that there is no check against the database
                Alert alert = new Alert(AlertType.ERROR);
                alert.setTitle("Login fail");
                alert.setHeaderText(null);
                alert.setContentText("No database connection! Cannot check user credentials.");
                alert.showAndWait();
        } else {
            String uname = username.getText();

            /* --- TODO: add code to query the database credentials --- */
                // could be if(!db.login(uname)) alert...

                // inform the user that there is no check against the database
                Alert alert = new Alert(AlertType.INFORMATION);
                alert.setTitle("Login fail");
                alert.setHeaderText(null);
                alert.setContentText("No user check implemented yet!");
                alert.showAndWait();
                /* --- END TODO --- */

            // setting the user name
                CurrentUser.instance().loginAs(uname);

                // inform the user about logging in
                actiontarget.setText("Sign in user "+uname);

                // inform booking tab of user change
                bookingTabCtrl.userChanged();
        }
    }

    public void initialize() {
        System.out.println("Initializing LoginTab.");
    }

    // helpers
    // use this pattern to send data down to controllers at initialization
    public void setBookingTab(BookingTab bookingTabCtrl) {
        System.out.println("LoginTab sets bookingTab:"+bookingTabCtrl);
        this.bookingTabCtrl = bookingTabCtrl;
    }

    public void setDatabase(Database db) {
        this.db = db;
    }

}
```

Listing 4: gui/BookingTab.java

```
package gui;

import javafx.collections.FXCollections;
import javafx.fxml.FXML;
import javafx.scene.control.Label;
import javafx.scene.control.ListView;
import javafx.scene.control.Button;
import javafx.scene.text.Text;

import java.util.List;
```

```
import datamodel.CurrentUser;
import datamodel.Database;
import datamodel.Show;

import java.util.ArrayList;


public class BookingTab {
        // top context message
        @FXML private Text topContext;
        // bottom message
        @FXML private Text bookMsg;

        // table references
        @FXML private ListView<String> moviesList;
        @FXML private ListView<String> datesList;

        // show info references
        @FXML private Label showTitle;
        @FXML private Label showDate;
        @FXML private Label showVenue;
        @FXML private Label showFreeSeats;

        // booking button
        @FXML private Button bookTicket;

        private Database db;
        private Show crtShow = new Show();

        public void initialize() {
                System.out.println("Initializing BookingTab");

                fillNamesList();
                fillDatesList(null);
                fillShow(null,null);

                // set up listeners for the movie list selection
                moviesList.getSelectionModel().selectedItemProperty().addListener(
                                (obs, oldV, newV) -> {
                                        // need to update the date list according to the selected movie
                                        // update also the details on the right panel
                                        String movie = newV;
                                        fillDatesList(newV);
                                        fillShow(movie,null);
                                });

                // set up listeners for the date list selection
                datesList.getSelectionModel().selectedItemProperty().addListener(
                                (obs, oldV, newV) -> {
                                        // need to update the details according to the selected date
                                        String movie = moviesList.getSelectionModel().getSelectedItem();
                                        String date = newV;
                                    fillShow(movie, date);
                                });

                // set up booking button listener
                // one can either use this method (setup a handler in initialize)
                // or directly give a handler name in the fxml, as in the LoginTab class
                bookTicket.setOnAction(
                                (event) -> {
                                        String movie = moviesList.getSelectionModel().getSelectedItem();
                                        String date = datesList.getSelectionModel().getSelectedItem();
                                        /* --- TODO: should book a ticket via the database --- */
                                        /* --- do not forget to report booking number! --- */
                                        /* --- update the displayed details (free seats) --- */
                                        report("Booked one ticket to "+movie+" on "+date);
                                });
```

```
        report("Ready.");
    }

    // helpers
    // updates user display
    private void fillStatus(String usr) {
        if(usr.isEmpty()) topContext.setText("You must log in as a known user!");
        else topContext.setText("Currently logged in as " + usr);
    }

    private void report(String msg) {
        bookMsg.setText(msg);
    }

    public void setDatabase(Database db) {
        this.db = db;
    }
    // fill the movies list with available titles
    private void fillNamesList() {
        List<String> allmovies = new ArrayList<String>();

        // query the database via db
        /* --- TODO: replace with own code --- */
        allmovies.add("Pulp Fiction");
        allmovies.add("The Big Lebowski");
        allmovies.add("Whiplash");
        /* --- END TODO --- */

        moviesList.setItems(FXCollections.observableList(allmovies));
        // remove any selection
        moviesList.getSelectionModel().clearSelection();
    }
    // fill the date list for the given movie
    private void fillDatesList(String m) {
        List<String> alldates = new ArrayList<String>();
        if(m != null) {
            // query the database via db
            /* --- TODO: replace with own code --- */
            alldates.add("2016-02-01");
            alldates.add("2016-01-15");
            /* --- END TODO --- */
        }
        datesList.setItems(FXCollections.observableList(alldates));
        // remove any selection
        datesList.getSelectionModel().clearSelection();
    }
    // fill the details on the right
    private void fillShow(String movie, String date) {
        if(movie == null) // no movie selected
            crtShow = new Show();
        else if(date == null) // no date selected yet
            crtShow = new Show(movie);
        else // query the database via db
            crtShow = db.getShowData(movie, date);

        showTitle.setText(crtShow.getTitle());
        showDate.setText(crtShow.getDate());
        showVenue.setText(crtShow.getVenue());
        if(crtShow.getSeats() >= 0) showFreeSeats.setText(crtShow.getSeats().toString());
        else showFreeSeats.setText("-");
    }

    // called in case the user logged in changed
    public void userChanged() {
        fillStatus(CurrentUser.instance().getCurrentUserId());
        fillNamesList();
        fillDatesList(null);
```

```
                fillShow(null,null);
        }
}
```

Listing 5: gui/TopTab.fxml

```
<?xml version="1.0" encoding="UTF-8"?>

<?import javafx.scene.layout.*?>
<?import javafx.scene.control.*?>

<BorderPane fx:controller="gui.TopTabView"  xmlns:fx="http://javafx.com/fxml/1"
        styleClass="root">
<top>
        <TabPane tabClosingPolicy="UNAVAILABLE">
        <tabs>
        <Tab text="Login">
                <fx:include fx:id="aLoginTab" source="loginTab.fxml"/>
        </Tab>
        <Tab text="Book">
                <fx:include fx:id="aBookingTab" source="bookingTab.fxml"/>
        </Tab>
    </tabs>
    </TabPane>
</top>
</BorderPane>
```

Listing 6: gui/loginTab.fxml

```
<?xml version="1.0" encoding="UTF-8"?>

<?import java.net.*?>
<?import javafx.geometry.*?>
<?import javafx.scene.control.*?>
<?import javafx.scene.layout.*?>
<?import javafx.scene.text.*?>

<GridPane fx:controller="gui.LoginTab"
    xmlns:fx="http://javafx.com/fxml" alignment="center" hgap="10" vgap="10"
    styleClass="root">
    <padding><Insets top="25" right="25" bottom="25" left="25"/></padding>

    <Text id="welcome-text" text="Welcome"
          GridPane.columnIndex="0" GridPane.rowIndex="0"
          GridPane.columnSpan="2"/>

    <Label text="User Name:"
          GridPane.columnIndex="0" GridPane.rowIndex="1"/>

    <TextField fx:id="username"
          GridPane.columnIndex="1" GridPane.rowIndex="1"/>

    <HBox spacing="10" alignment="bottom_right"
          GridPane.columnIndex="1" GridPane.rowIndex="4">
          <Button text="Sign In"
          onAction="#handleSubmitButtonAction"/>
    </HBox>

    <Text fx:id="actiontarget"
          GridPane.columnIndex="1" GridPane.rowIndex="6"/>

    <stylesheets>
      <URL value="@login.css" />
    </stylesheets>

</GridPane>
```

Listing 7: gui/bookingTab.fxml

```
<?xml version="1.0" encoding="UTF-8"?>

<?import javafx.scene.control.Button?>
<?import javafx.scene.control.Label?>
<?import javafx.scene.control.ListView?>
<?import javafx.scene.control.SplitPane?>
<?import javafx.scene.layout.AnchorPane?>
<?import javafx.scene.layout.ColumnConstraints?>
<?import javafx.scene.layout.GridPane?>
<?import javafx.scene.layout.RowConstraints?>
<?import javafx.scene.text.*?>

<SplitPane dividerPositions="0.4" maxHeight="-Infinity" maxWidth="-Infinity"
        minHeight="-Infinity" minWidth="-Infinity" prefHeight="400.0" prefWidth="600.0"
        xmlns="http://javafx.com/javafx/8.0.65" xmlns:fx="http://javafx.com/fxml"
        fx:controller="gui.BookingTab" styleClass="root">
  <items>
    <AnchorPane minHeight="0.0" minWidth="0.0" prefHeight="400.0" prefWidth="240.0">
         <children>
            <ListView fx:id="moviesList" prefHeight="400.0" prefWidth="120.0"
                AnchorPane.bottomAnchor="0.0" AnchorPane.leftAnchor="0.0"
                AnchorPane.topAnchor="20.0" />
            <ListView fx:id="datesList" layoutX="120.0" prefHeight="380.0"
                prefWidth="110.0" AnchorPane.bottomAnchor="0.0"
                AnchorPane.rightAnchor="0.0" AnchorPane.topAnchor="20.0" />
            <Label layoutX="14.0" layoutY="6.0" text="Movie" AnchorPane.leftAnchor="20.0"
                AnchorPane.topAnchor="0.0" />
            <Label layoutX="158.0" layoutY="6.0" text="Date" AnchorPane.rightAnchor="20.0"
                AnchorPane.topAnchor="0.0" />
         </children></AnchorPane>
    <AnchorPane minHeight="0.0" minWidth="0.0" prefHeight="400.0" prefWidth="360.0">
         <children>
            <Button fx:id="bookTicket" mnemonicParsing="false" text="Book Ticket"
                AnchorPane.bottomAnchor="60.0" AnchorPane.leftAnchor="50.0"
                AnchorPane.rightAnchor="50.0" />
            <Text fx:id="topContext" text="Not logged in!" AnchorPane.leftAnchor="5.0"
                AnchorPane.topAnchor="5.0" />
            <GridPane AnchorPane.leftAnchor="5.0" AnchorPane.rightAnchor="5.0"
                AnchorPane.topAnchor="30.0">
              <columnConstraints>
                <ColumnConstraints hgrow="SOMETIMES" minWidth="10.0" prefWidth="100.0" />
                <ColumnConstraints hgrow="SOMETIMES" minWidth="10.0" prefWidth="100.0" />
              </columnConstraints>
              <rowConstraints>
                <RowConstraints minHeight="10.0" prefHeight="30.0" vgrow="SOMETIMES" />
                <RowConstraints minHeight="10.0" prefHeight="30.0" vgrow="SOMETIMES" />
                <RowConstraints minHeight="10.0" prefHeight="30.0" vgrow="SOMETIMES" />
                <RowConstraints minHeight="10.0" prefHeight="30.0" vgrow="SOMETIMES" />
              </rowConstraints>
               <children>
                  <Label text="Movie" />
                  <Label text="Date" GridPane.rowIndex="1" />
                  <Label text="Plays at" GridPane.rowIndex="2" />
                  <Label text="Free seats" GridPane.rowIndex="3" />
                  <Label fx:id="showTitle" text="Label" GridPane.columnIndex="1" />
                  <Label fx:id="showDate" text="Label" GridPane.columnIndex="1"
                         GridPane.rowIndex="1" />
                  <Label fx:id="showVenue" text="Label" GridPane.columnIndex="1"
                         GridPane.rowIndex="2" />
                  <Label fx:id="showFreeSeats" text="Label" GridPane.columnIndex="1"
                         GridPane.rowIndex="3" />
               </children>
            </GridPane>
            <Text fx:id="bookMsg" layoutX="83.0" layoutY="368.0" text="Label"
                textAlignment="CENTER" AnchorPane.bottomAnchor="10.0"
                AnchorPane.leftAnchor="10.0" AnchorPane.rightAnchor="10.0" />
```

```
        </children></AnchorPane>
  </items>

</SplitPane>
```

Listing 8: datamodel/CurrentUser.java

```
package datamodel;

/**
 * CurrentUser represents the current user that has logged on to
 * the movie booking system. It is a singleton class.
 */
public class CurrentUser {
    /**
     * The single instance of this class
     */
    private static CurrentUser instance;

    /**
     * The name of the current user.
     */
    private String currentUserId;

    /**
     * Create a CurrentUser object.
     */
    private CurrentUser() {
        currentUserId = null;
    }

    /**
     * Returns the single instance of this class.
     *
     * @return The single instance of the class.
     */
    public static CurrentUser instance() {
        if (instance == null)
            instance = new CurrentUser();
        return instance;
    }

    /**
     * Check if a user has logged in.
     *
     * @return true if a user has logged in, false otherwise.
     */
    public boolean isLoggedIn() {
        return currentUserId != null;
    }

    /**
     * Get the user id of the current user. Should only be called if
     * a user has logged in.
     *
     * @return The user id of the current user.
     */
    public String getCurrentUserId() {
        return currentUserId == null ? "<none>" : currentUserId;
    }

    /**
     * A new user logs in.
     *
     * @param userId The user id of the new user.
     */
    public void loginAs(String userId) {
```

```
        currentUserId = userId;
    }
}
```

Listing 9: datamodel/Database.java

```
package datamodel;

import java.sql.*;
/**
 * Database is a class that specifies the interface to the
 * movie database. Uses JDBC and the MySQL Connector/J driver.
 */
public class Database {
    /**
     * The database connection.
     */
    private Connection conn;

    /**
     * Create the database interface object. Connection to the database
     * is performed later.
     */
    public Database() {
        conn = null;
    }

    /**
     * Open a connection to the database, using the specified user name
     * and password.
     *
     * @param userName The user name.
     * @param password The user's password.
     * @return true if the connection succeeded, false if the supplied
     * user name and password were not recognized. Returns false also
     * if the JDBC driver isn't found.
     */
    public boolean openConnection(String userName, String password) {
        try {
            Class.forName("com.mysql.jdbc.Driver");
            conn = DriverManager.getConnection
                ("jdbc:mysql://puccini.cs.lth.se/" + userName,
                 userName, password);
        }
        catch (SQLException e) {
            System.err.println(e);
            e.printStackTrace();
            return false;
        }
        catch (ClassNotFoundException e) {
            System.err.println(e);
            e.printStackTrace();
            return false;
        }
        return true;
    }

    /**
     * Close the connection to the database.
     */
    public void closeConnection() {
        try {
            if (conn != null)
                conn.close();
        }
        catch (SQLException e) {
                e.printStackTrace();
```

```
        }
        conn = null;

        System.err.println("Database connection closed.");
    }

    /**
     * Check if the connection to the database has been established
     *
     * @return true if the connection has been established
     */
    public boolean isConnected() {
        return conn != null;
    }

        public Show getShowData(String mTitle, String mDate) {
                Integer mFreeSeats = 42;
                String mVenue = "Kino 2";

                /* --- TODO: add code for database query --- */

                return new Show(mTitle, mDate, mVenue, mFreeSeats);
        }

    /* --- TODO: insert more own code here --- */
}
```

Listing 10: datamodel/Show.java

```
package datamodel;

// Container for the database data
/* --- TODO: Modify as needed --- */

public class Show {
                // attributes associated with database columns
                private String title;
                private String date;
                private String venue;
                private Integer freeSeats;

                // constructor for "no show"
                public Show() { init("","","",-1); }

                // constructor defining all content
                public Show(String t, String d, String v, Integer fs) { init(t,d,v,fs); }

                // constructor defining only the title
                public Show(String t) { init(t,"","",-1); }

                // all constructors use this
                private void init(String t, String d, String v, Integer fs) {
                        title = t; date = d; venue = v; freeSeats = fs;
                }

                // getters
                public String getTitle() { return title; }
                public String getDate() { return date; }
                public String getVenue() { return venue; }
                public Integer getSeats() { return freeSeats; }
}
```