Online Radiology Information System

CMPUT 391

Mar. 31, 2014

David Chau-Tran, Frank Yau

The project was completed with the following modules:

Login module

Completed using Login.java and EditInfo.java. In this class, the user is asked to enter their username and password to attempt to login. If the username or password is not correct, an error message is returned. To do so, this class checks the entered user exists in the database, and if so, checks if the entered password is the same as the password in the database. This check is done by using a cursor that grabs the entered user information by using the following SQL statement:

SELECT user\_name, password, person\_id, class FROM users WHERE user\_name = “{input username}”

If the user successfully logs in, their ID number and their class/permission level is stored in the session. This class also handles the clearing of the session when the user decides to log out.

The user is also able to edit their personal info as part of the login module. The EditInfo class will display a user’s first and last name, address, phone #, email and password. This class wil also allow the user to edit their personal information. The class will then ask the user to insert a new value. If the input is empty, then no personal information will be changed and the page will be refreshed. The following SQL statements are used:

To edit first name: UPDATE persons SET first\_name =? WHERE person\_id =?

To edit last name: UPDATE persons SET last\_name =? WHERE person\_id =?

To edit address: UPDATE persons SET address =? WHERE person\_id =?

To edit email: UPDATE persons SET email =? WHERE person\_id =?

To edit phone: UPDATE persons SET phone =? WHERE person\_id =?

To edit password: UPDATE users SET password =? WHERE person\_id =?

The first ? is what the user enters into the input field and the second one is that users person\_id obtained from the login module.

User management module

Completed using Update.java and Insert.java. This module is only accessible by the system administrator. The user can choose to either update to a table or insert to a table. When updating, the user can update the following tables given from a drop down menu: Users and Persons. After choosing the table, the user can choose the column they wish to update, and must give a condition to match it for. If any of the text boxes are empty, no entry will be updated in the database and the page will be refreshed. The following SQL statements are used for this class:

UPDATE {table name} SET {dropdown box 1} = {text box 1} WHERE {user\_name/person\_id} = {text box 2}

The user is also able to insert new rows into the table. This is done using Insert.java. The user can choose to insert the data into the same tables that updating allowed, and one more: Users, Persons and Family\_doctor. Inserting into each table requests different information, but all of them require all the fields to be filled to insert the data into the database. If any of the text boxes are empty, no entry will be added to the database and the page will be refreshed. The following SQL statements are used:

Users Table: INSERT INTO users (user\_name,password,class,person\_id,date\_registered) VALUES(…)

Persons Table: INSERT INTO persons (person\_id,first\_name,last\_name,address,email,phone) VALUES(….)

Family\_doctor: INSERT INTO family\_doctor (doctor\_id,patient\_id) VALUES(…)

SELECT person\_id FROM persons: Gets person\_ids from person to use in Users.

SELECT u.person\_id FROM persons p, users u WHERE p.person\_id = u.person\_id AND u.class = 'd': Gets a list of doctor ids to be used in family\_doctor table.

SELECT u.person\_id FROM persons p, users u WHERE p.person\_id = u.person\_id AND u.class = 'p': Gets a list of patient ids to be used in the family\_doctor table.

Report generating module

Completed using Generate.java. In this class, the user can generate a list of patients that match the inputs. The user must enter a diagnosis and two test dates. The module will generate a report which will return the first and last name, address and phone number of patients that have that diagnosis and if the test date is in between the two test dates that were entered. If any of the text boxes are empty, no report will be generated and the page will be refreshed. This is done using the following SQL statement:

SELECT DISTINCT first\_name, last\_name, address, phone FROM persons p, radiology\_record r WHERE p.person\_id = r.patient\_id AND r.diagnosis LIKE{diagnosis} AND r.test\_date BETWEEN {test date 1} AND{test date 2}

Uploading module

Completed using UploadImage.java and CreateRecord.java. Only admins and radiologists have the option to use this module. In this class, the user can choose to upload an image to a specified radiology record. Only one image may be uploaded at a time, and the user must choose an existing record by choosing the record ID from a drop down menu. If the user attempts to upload nothing or a non-valid image, an error is returned. To create the drop down menu of record IDs, the following SQL statement is used:

SELECT record\_id FROM radiology\_record

This allows a cursor to get all the record IDs in the database and output them as choices.

Uploading images requires a few more SQL statements. First, a unique record ID must be created for each individual image. This is done by using a pre-created SQL sequence. The SQL statement to access this sequence is the following:

SELECT pic\_id\_sequence.nextval from dual

Then, to insert the images into a record, they must first be prepared beforehand. This is done using the following SQL statement:

INSERT INTO pacs\_images VALUES("{record\_id}","{image\_id}",empty\_blob(),empty\_blob(),empty\_blob())

Then, to find the entry that was just inserted, this SQL statement is used:

SELECT \* FROM pacs\_images WHERE image\_id = "{image\_id}" FOR UPDATE

The uploaded image is then resized, and put into the corresponding column.

This module also allows the user to create a new record by selecting the option from the home menu. The user fills in all text boxes in this page, and clicks the create button. As long as the input patient ID exists in the database, the record is inserted into the database. The following SQL statement is used to insert the radiology record:

INSERT INTO radiology\_record (record\_id, patient\_id, doctor\_id, radiologist\_id, test\_type, prescribing\_date, test\_date, diagnosis, description) VALUES (‘{id}’, '{patient id}', '{doctor id}', '{radiologist id}', '{test type}', '{prescribing date}', '{test date}', '{diagnosis}', '{description}')

Search module

Completed using Search.java, GetOnePic.java, and View.java. This module allows the user to make a search of the radiology records. The user can search using keywords and/or a time period. The search can find keywords in a radiology record that match the patient’s first name, last name, record diagnosis, or record description. If a time period is specified, then the search makes sure all found records have a prescribing date or test date within the time period. The returned results are the patient’s ID, first and last name, record ID, doctor ID, radiologist ID, test type, prescribing date, test date, diagnosis, description and images related to any matching records. To get this information that satisfies all these conditions, this SQL statement is used:

SELECT p.first\_name, p.last\_name, r.\*, pi.image\_id [,6\*score("3 ")+6\*score("4")+3\*score("1")+score("2") AS rank ] FROM radiology\_record r FULL JOIN persons p ON r.patient\_id = p.person\_id FULL JOIN pacs\_images pi ON pi.record\_id = r.record\_id FULL JOIN users u ON p.person\_id = u.person\_id FULL JOIN family\_doctor f ON f.patient\_id = r.patient\_id WHERE [contains(r.diagnosis, ?, “1") > 0 OR contains(r.description, ?, "2") > 0 OR contains(p.first\_name, ?, "3") > 0 OR contains(p.last\_name, ?, "4") > 0] AND [r.patient\_id/r.doctor\_id/r.radiologist\_id = {user id}] AND [(r.prescribing\_date BETWEEN '{time period start}' AND '{time period end}' OR r.test\_date BETWEEN '{time period start}' AND '{time period end}') ORDER BY rank desc, r.record\_id

A problem that we encountered is when multiple keywords are entered. The solution that we used to solve this is to create a string array split by spaces from the entered input. With these split keywords, we then insert additional points into the rank, by doing the following for the SELECT clause:

SELECT p.first\_name, p.last\_name, r.\*, pi.image\_id, [6\*score(“3”)+6\*score(“4”)+3\*score(“1”)+score(“2”)] + [6\*score(“7”)+6\*score(“8”)+3\*score(“5”)+score(“4”)] + …

And the following for the WHERE clause:

WHERE [contains(r.diagnosis, ?, “1”) > 0 OR contains(r.description, ?, “2”) > 0 OR contains(p.first\_name, ?, “3”) > 0 OR contains(p.last\_name, ?, “4”) > 0] OR [contains(r.diagnosis, ?, “5”) > 0 OR contains(r.description, ?, “6”) > 0 OR contains(p.first\_name, ?, “7”) > 0 OR contains(p.last\_name, ?, “8”) > 0] OR …

The security module of the search is controlled using the class and ID stored in the session. Depending on the currently logged in user’s class, one of the following conditions are added to the WHERE clause:

AND r.patient\_id = ‘{user id}’/AND r.doctor\_id = ‘{user id}’ AND f.doctor\_id = r.doctor\_id/AND r.radiologist\_id = ‘{user id}

This module also allows the user to sort the results by either the prescribing date or the test date, and either in ascending order or descending order. This is done by modifying the ORDER BY clause in the SQL statement with one of the following:

ORDER BY r.prescribing\_date ASC/ORDER BY r.prescribing\_date DESC/ORDER BY r.test\_date ASC/ORDER BY r.test\_date DESC

The results of the query are then printed out as a table in HTML, and if there are multiple images associated with a record, they are all listed in the same cell.

All the images are linked to a page generated by the View class. This page simply links to a page with a larger version of the image, and the user can click that image again to view the original image. This page gets its images from the GetOnePic class.

The GetOnePic class simply gets the requested image from the database, given the image ID. The following SQL statements are used in this class:

SELECT thumbnail FROM pacs\_images WHERE image\_id={picid.substring(5)}

SELECT regular\_size FROM pacs\_images WHERE image\_id={picid.substring(4)}

SELECT full\_size FROM pacs\_images WHERE image\_id={picid}

The variable picid is a string formed with the image ID, and preceded by either “thumb” or “norm”, which would dictate which type of image to get from the database.

This class was adapted from the GetOnePic class from the CMPUT391 example servlets.

Data analysis module

Completed using OLAP.java and Refresh.java. This module generates a data analysis report that shows the total number of images associated with a combination of the variables of patient name, test type and test date. The user can choose which of the variables to use to generate the report through checkboxes, and can choose a time hierarchy if test date is selected. If there is no hierarchy selected and test date is checked, a default hierarchy of “weekly” is used. The OLAP report is generated with the following SQL statement:

SELECT DISTINCT [patient\_id,] [test\_type,] [trunc(test\_date, d/mm/y)], COUNT(image\_id) FROM olap\_view GROUP BY CUBE ([patient\_id,] [test\_type,] [trunc(test\_date, d/mm/y)])

“olap\_view” is a view generated from the Refresh class. It is a view that joins all the necessary tables so that the OLAP generation will be quicker. The SQL statement to make this view is the following:

CREATE OR REPLACE VIEW olap\_view AS

SELECT r.patient\_id, r.test\_type, r.test\_date, pi.image\_id

FROM radiology\_record r FULL JOIN pacs\_images pi ON r.record\_id = pi.record\_id

Refreshing the table will output a success or error message, and allow the user to return to the data analysis module. The user can choose to update the OLAP view by clicking the link to refresh the table.

The results of this module is output into an HTML table which lists the patient ID, patient name, test type, and/or the beginning of the time hierarchy, and the number of images associated with all combinations. If the variable from SQL is null, then “ANY” is output into the cell.

To get the name associated with a patient ID, a second cursor is used. This is the SQL statement to get the names in the OLAP class:

SELECT first\_name, last\_name FROM persons WHERE person\_id = {person id}

A second cursor was required to get the names from the database, as extraneous results would be produced if the first\_name and last\_name columns were in the SQL statement with CUBE.

The user can also roll up or drill down in this module. Rolling up simply decreases the specificity of the time hierarchy (week -> month, month -> year) and drilling down increases the specificity of the time hierarchy (month -> week, year -> month) for the specified month/year.