DBMS Project: Operation Leg Day

(A Gym Membership Database System)

Miriam Ekiye, Cody Noack, Jesse Schultz, Yalcin Taf, Zachary Nabors, Johnathan Molina, David Holder, Ethan Bowles, Ederick Mendoza

1. **Introduction**

Our project is to create a program that allows its users full control over their account. This is to ensure that the member can make any changes they want such as; what type of membership they have, what features they want access to, and if they no longer desire to attend this gym. This is also to remove the hassle that most gyms have where in order to make any changes to your account, you must show up in person to the physical location (not everyone can be there at any time).

1. **System Study**

2.1 Proposed program

The program is simple in design and easy to apply to any computer, meaning that it is very flexible and will work in all new/recently built computers. This program has the following benefits:

* Changing membership tier
* Changing utilities/activities access
* Adding new members
* Creating new utilities/activities
* Removing members
* Members can make edits on their own(convenient accessibility)
* Accessible on any devices that can connect to the internet
* Changes are immediate(for utilities/activities)

2.2 Introduction about the front end (Java)

**In use with eclipse and additional Jar plugins like WindowBuilder / Swing Design, Java has been an efficient language to use. A few other external Java JARs were imported for use of making a database connection and developing a method to hash passwords created in the Front-end of the application.**

* The PostgreSQL JDBC Driver 42.2.8 had been used to be imported in the GUI application for setting up a database connection, creating SQL syntax to be passed as a query though these recorded statements into a database and record results from the database.
* Mindrot’s, jBCrypt-0.4, a Java implementation of the hashing algorithm was compiled and imported for part of the project library to ensure every password created will be hashed and saved in the database.
* For development tools, WindowBuilder was utilized in the Eclipse IDE for a powerful, easy to use GUI designer to create buttons, panels, labels, as well as fields onto the application frame for login use once compiled.

2.3 Introduction about the back end (PostgreSQL)

PostgreSQL is a powerful, open source object-relational database system that is simple to use and setup, and is completely reliable in its ability to manage, store, and edit data in various servers and tables. With how simple it is to separate different servers, establish logins, and set permissions to specific accounts, managing this database is extremely easy.

In short:

* Simple, easy to use database system
* Able to create separate servers for various purposes
* Simple system of establishing permissions and creating employee/admin accounts to manage database
* Easy to read and manage

1. **System Specification**

3.1 Hardware Requirements

* Minimum 32-bit processor
* 1 GB storage minimum
* 256 MB of free disk space for job result directory and log directory (total 508 MB)

3.2 Software Requirements

* Windows 7 or Newer
* Java JDK 11.0.5
* Eclipse IDE 2019-09

Plug-ins / Packages Used: [PostgreSQL JDBC Driver](https://jdbc.postgresql.org/download.html)

[Bcrypt Java Package](https://www.mindrot.org/projects/jBCrypt/)  
[Eclipse Swing add-on](https://www.eclipse.org/windowbuilder/)

* PostgreSQL Version 11.6
* AIX, FreeBSD, Linux, Solaris, Windows compatible

1. **Database Design**
   1. Conceptual Design

List of Entities

* Client
* Login
* Tier Options
* Relationship

List of attributes

Client:

* E-mail
* Phone number
* Employee
  + (1 for yes 0 for no)
* First Name
* Last Name
* Tier
  + Three levels of membership

Login

* Username
* Password

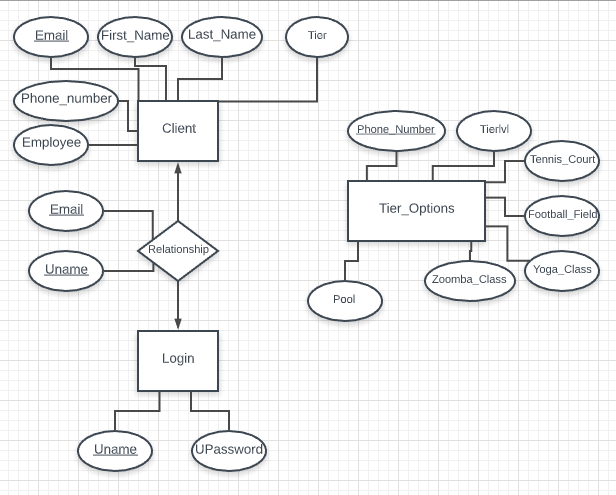
Tier Options

* Phone number
* Tier Level
* Tennis Court
* Football Field
* Yoga Class
* Zoomba Class
* Pool

Relationship

* E-mail
* Username

**E-R Model**



* 1. **Logical design**

1. Client: member’s records

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| S.no | Field name | Data type | Description | Constraints |
| 1 | Email | Varchar | Contact for member | Primary key |
| 2 | Phone\_number | Varchar | Contact for member |  |
| 3 | Employee | Varchar | Users with permissions to insert into database |  |
| 4 | First\_Name | Varchar | First name of member |  |
| 5 | Last\_Name | Varchar | Last name of member |  |
| 6 | Tier | Varchar | Determines available options for member |  |

Schema definition:

CREATE TABLE client (first\_name VARCHAR(255), last\_name VARCHAR(255), email VARCHAR(255) PRIMARY KEY, tier VARCHAR(255), phone\_number VARCHAR(255), employee VARCHAR(255));

1. Login: Login credentials for users

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| S.no | Field name | Data type | Description | Constraints |
| 1 | Uname | Varchar | Name for the client’s account | Primary key |
| 2 | UPassword | varchar | Security word for account |  |

Schema definition:

CREATE TABLE login (uname VARCHAR(255) PRIMARY KEY,

upassword VARCHAR(255));

1. Tier Options: sets value of membership to members

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| S.no | Field name | Data type | Description | Constraints |
| 1 | Phone\_number | Varchar | Contact for member | Primary key |
| 2 | Tier\_level | Varchar | Determines available options for member |  |
| 3 | Tennis\_Court | Varchar | Activity |  |
| 4 | Football\_Field | Varchar | Activity |  |
| 5 | Yoga\_Class | Varchar | Activity |  |
| 6 | Zoomba\_Class | Varchar | Activity |  |
| 7 | Pool | Varchar | Activity |  |

Schema definition:

CREATE TABLE tier\_options (phone\_number VARCHAR(255) PRIMARY KEY, tierlvl VARCHAR(255), pool VARCHAR(255), tennis\_court VARCHAR(255), football\_field VARCHAR(255), zoomba\_class VARCHAR(255), yoga\_class VARCHAR(255));

1. Relationship:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| S.no | Field name | Data type | Description | Constraints |
| 1 | Email | Varchar | Contact for member | Foreign key |
| 2 | Uname | Varchar | Name for the client’s account | Foreign key |

Schema definition:

CREATE TABLE relationship (uname VARCHAR(255) REFERENCES login(uname), email VARCHAR(255) REFERENCES client(email));

1. Implementation

This is where Users would log in

A screenshot of a cell phone

Description automatically generated

This panel gives user view of their info along with buttons

A screenshot of a cell phone

Description automatically generated

Edit information panel along with delete function

A screenshot of a cell phone

Description automatically generated

Edit tier options panel, selection is required

A screenshot of a cell phone

Description automatically generated

Employees creating a New User, Passwords must match!  
All fields must be filled!

A screenshot of a cell phone

Description automatically generated

A screenshot of a cell phone

Description automatically generated

Employees search by Email

A screenshot of a cell phone

Description automatically generated

1. Additional Comments

Some features that were left out such as modification and update for employees. Make changes to data types in all tables, such as all activities/employees having Boolean, phone numbers/tier level to integer. Sufficient amounts of check-based constraints on tables as well as triggers to throw SQLExceptions back to the front-end.

1. Attach Source Code

[OperationLegDay Class](https://pastebin.com/QHaXGJVp)

[User Class](https://pastebin.com/hY9RCrZF)

1. **package** legDay;
3. **import** java.awt.CardLayout;
4. **import** java.awt.Dimension;
5. **import** java.awt.EventQueue;
6. **import** java.awt.FlowLayout;
7. **import** java.awt.Font;
8. **import** java.awt.event.ActionEvent;
9. **import** java.awt.event.ActionListener;
10. **import** java.awt.event.KeyAdapter;
11. **import** java.awt.event.KeyEvent;
12. **import** java.awt.event.MouseAdapter;
13. **import** java.awt.event.MouseEvent;
14. **import** java.sql.SQLException;
15. **import** java.util.ArrayList;
17. **import** javax.swing.ImageIcon;
18. **import** javax.swing.JButton;
19. **import** javax.swing.JCheckBox;
20. **import** javax.swing.JFrame;
21. **import** javax.swing.JLabel;
22. **import** javax.swing.JOptionPane;
23. **import** javax.swing.JPanel;
24. **import** javax.swing.JPasswordField;
25. **import** javax.swing.JScrollPane;
26. **import** javax.swing.JTable;
27. **import** javax.swing.JTextField;
28. **import** javax.swing.table.DefaultTableModel;
30. **public** **class** OperationLegDay {
32. **private** JFrame frame;
33. **private** JTextField userNameText;
34. **private** JTable userInfoTable;
35. **private** JTable userInfoTable2;
36. **private** JTextField createNewUserName;
37. **private** JPasswordField createNewPassword;
38. **private** JPasswordField confirmNewPassword;
39. **private** JTextField firstNameText;
40. **private** JTextField lastNameText;
41. **private** JTextField employeeText;
42. **private** JTextField phoneNumberText;
43. **private** JTextField emailText;
44. **private** JTextField tierText;
45. **private** JPasswordField passwordField;
46. **private** JLabel loginLabel;
47. **private** JPanel userInfoPanel;
48. **private** ArrayList<String> dbinfo=**new** ArrayList<String>();
49. **private** ArrayList<String> dbinfo2=**new** ArrayList<String>();
50. **private** ArrayList<String> updateOptions = **new** ArrayList<String>();
51. **private** User user;
52. **private** ImageIcon icon;
53. **private** JPanel createNewUserPanel;
54. **private** JPanel createNewUserPanel2;
55. **private** JPanel employeePanel;
56. **private** JTable tablePopulateData;
57. **private** JButton btnLogOut;
58. **private** JTextField searchText;
59. **private** JLabel lblSearchEmail;
60. **private** JTextField editFirstName;
61. **private** JTextField editLastName;
62. **private** JTextField editEmail;
63. **private** JTextField editPhoneNumber;
64. **private** JPanel editInfoPanel;
65. **private** JButton createNew;
66. **private** JButton deleteBtn;
67. **private** JButton btnCancel;
68. **private** JButton btnCancel\_1;
69. **private** JLabel lblPasswordsDoNot;
70. **private** JPanel editOptionsPanel;
71. **private** JCheckBox chckbxPool;
72. **private** JButton btnSubmit\_1;
73. **private** JCheckBox chckbxTennis;
74. **private** JCheckBox chckbxFootballField;
75. **private** JCheckBox chckbxZoomba;
76. **private** JCheckBox chckbxYoga;
78. ***/\*\****
79. ***\* Launch the application.***
80. ***\*/***
81. **public** **static** **void** main(String[] args) {
82. EventQueue.invokeLater(**new** Runnable() {
83. **public** **void** run() {
84. **try** {
85. OperationLegDay window = **new** OperationLegDay();
86. window.frame.setVisible(**true**);
87. } **catch** (Exception e) {
88. e.printStackTrace();
89. }
90. }
91. });
92. }
94. ***/\*\****
95. ***\* Create the application.***
96. ***\*/***
97. **public** OperationLegDay() {
98. initialize();
99. }
101. ***/\*\****
102. ***\* Initialize the contents of the frame.***
103. ***\*/***
104. **private** **void** initialize() {
105. frame = **new** JFrame();
106. frame.setBounds(100, 100, 640, 472);
107. frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);
108. frame.getContentPane().setLayout(**new** CardLayout(0, 0));
109. frame.setLocationRelativeTo(**null**);
110. icon = **new** ImageIcon("src/img/Icon.png");
111. frame.setIconImage(icon.getImage());
112. frame.setTitle("Gym");
113. frame.setResizable(**false**);
115. JPanel loginPanel = **new** JPanel();
116. frame.getContentPane().add(loginPanel, "name\_178198344352392");
117. loginPanel.setLayout(**null**);
119. userNameText = **new** JTextField();
120. userNameText.setBounds(73, 107, 116, 22);
121. loginPanel.add(userNameText);
122. userNameText.setColumns(10);
124. JLabel lblUserName = **new** JLabel("User Name");
125. lblUserName.setBounds(73, 72, 102, 16);
126. loginPanel.add(lblUserName);
128. JLabel lblPassword = **new** JLabel("Password");
129. lblPassword.setBounds(73, 206, 116, 22);
130. loginPanel.add(lblPassword);
131. *//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Login button function\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**
132. JButton btnLogin = **new** JButton("Login");
133. btnLogin.addActionListener(**new** ActionListener() {
134. **public** **void** actionPerformed(ActionEvent arg0)
135. {
136. */\**
137. *\* this gathers the information typed into the text fields on the window*
138. *\*/*
139. String user2,password2;
140. **boolean** verify;
141. **boolean** isEmp = **false**;
142. user2 = userNameText.getText();
143. password2 = passwordField.getText();
144. *//reset text fields*
145. userNameText.setText("");
146. passwordField.setText("");
147. *//passes the data into the LoginVerify*
148. user = **new** User(user2,password2);
149. **try** {
150. verify = user.loginVerify();
151. **if**(verify)
152. {
153. *//alters login status label*
154. loginLabel.setText("Login: Success");
155. *//check if user is an employee*
156. isEmp = user.isEmployee();
157. System.out.println(isEmp);
159. **if**(isEmp==**true**)
160. {
162. *//switch to alternate window*
163. employeePanel.setVisible(**true**);
164. loginPanel.setVisible(**false**);
165. *//gets info from database to populate table*
167. *//Removes duplicate problem.*
168. DefaultTableModel model = (DefaultTableModel) tablePopulateData.getModel();
169. model.setRowCount(0);

172. user.getAllInfo(tablePopulateData);
174. tablePopulateData.setPreferredScrollableViewportSize(**new** Dimension(600,300));
175. tablePopulateData.setFillsViewportHeight(**true**);
176. JScrollPane js=**new** JScrollPane(tablePopulateData);
177. js.setVisible(**true**);
178. employeePanel.add(js);
179. }
180. **else** **if**(isEmp==**false**){
182. *//alternates viewable windows*
183. userInfoPanel.setVisible(**true**);
184. loginPanel.setVisible(**false**);
186. creatingUserInfoTable();
187. }
188. }
189. **else**
190. {
191. *//if login fails, it will set loginLabel to "Login: Failed"*
192. loginLabel.setText("Login: Failed");
193. }
195. } **catch** (SQLException e1) {
196. *// TODO Auto-generated catch block*
197. e1.printStackTrace();
198. }
199. }
200. });
201. btnLogin.setBounds(386, 264, 97, 25);
202. loginPanel.add(btnLogin);
204. passwordField = **new** JPasswordField();
205. passwordField.setBounds(73, 253, 123, 22);
206. loginPanel.add(passwordField);
208. loginLabel = **new** JLabel("Login:");
209. loginLabel.setBounds(353, 72, 130, 85);
210. loginPanel.add(loginLabel);
212. userInfoPanel = **new** JPanel();
213. frame.getContentPane().add(userInfoPanel, "name\_178198350432762");
214. userInfoPanel.setLayout(**null**);
215. userInfoPanel.setVisible(**false**);
217. JLabel lblUserInfo = **new** JLabel("User Info");
218. lblUserInfo.setFont(**new** Font("Tahoma", Font.BOLD, 17));
219. lblUserInfo.setBounds(77, 89, 107, 35);
220. userInfoPanel.add(lblUserInfo);
221. *//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*log out button function\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**
222. JButton logoutBtn = **new** JButton("Log out");
223. logoutBtn.addActionListener(**new** ActionListener() {
224. **public** **void** actionPerformed(ActionEvent arg0)
225. {
226. userInfoPanel.setVisible(**false**);
227. loginPanel.setVisible(**true**);
228. userInfoPanel.remove(userInfoTable); *//this fixes the bug where information is being displayed from the first user. the table needs to be removed each time and recreated with a new user.*
229. loginLabel.setText("Logout: Success"); *// CHANGESS*
230. }
231. });
232. logoutBtn.setBounds(444, 295, 145, 50);
233. userInfoPanel.add(logoutBtn);
234. *//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*EDIT CLIENT INFO\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**
235. JButton btnEditInformation = **new** JButton("Edit Information");
236. btnEditInformation.addActionListener(**new** ActionListener() {
237. **public** **void** actionPerformed(ActionEvent e) {
239. editInfoPanel.setVisible(**true**);
240. userInfoPanel.setVisible(**false**);
241. userInfoPanel.remove(userInfoTable);
243. **try** {
244. dbinfo = user.getInfo();
245. } **catch** (SQLException e1) {
246. *// TODO Auto-generated catch block*
247. e1.printStackTrace();
248. }
250. editFirstName.setText(dbinfo.get(0));
251. editLastName.setText(dbinfo.get(1));
252. editEmail.setText(dbinfo.get(2));
253. editPhoneNumber.setText(dbinfo.get(4));
255. }
256. });
257. btnEditInformation.setBounds(231, 295, 156, 50);
258. userInfoPanel.add(btnEditInformation);
260. *////////\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*EDIT OPTIONS EDIT OPTIONS EDIT OPTIONS!!!!!!!!\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**
261. JButton btnEditOptions = **new** JButton("Edit Options");
262. btnEditOptions.addActionListener(**new** ActionListener() {
263. **public** **void** actionPerformed(ActionEvent arg0)
264. {
265. editOptionsPanel.setVisible(**true**);
266. userInfoPanel.setVisible(**false**);
268. userInfoPanel.remove(userInfoTable);
269. userInfoPanel.remove(userInfoTable2);
270. }
271. });
272. btnEditOptions.setBounds(22, 295, 150, 50);
273. userInfoPanel.add(btnEditOptions);
275. editInfoPanel = **new** JPanel();
276. frame.getContentPane().add(editInfoPanel, "name\_1703263228900954");
277. editInfoPanel.setLayout(**null**);
279. editFirstName = **new** JTextField();
280. editFirstName.setBounds(35, 66, 116, 22);
281. editInfoPanel.add(editFirstName);
282. editFirstName.setColumns(10);
284. editLastName = **new** JTextField();
285. editLastName.setBounds(35, 142, 116, 22);
286. editInfoPanel.add(editLastName);
287. editLastName.setColumns(10);
289. editEmail = **new** JTextField();
290. editEmail.setBounds(35, 222, 116, 22);
291. editInfoPanel.add(editEmail);
292. editEmail.setColumns(10);
294. editPhoneNumber = **new** JTextField();
295. editPhoneNumber.setBounds(354, 66, 116, 22);
296. editInfoPanel.add(editPhoneNumber);
297. editPhoneNumber.setColumns(10);
299. JLabel lblFirstName\_1 = **new** JLabel("First Name");
300. lblFirstName\_1.setBounds(35, 37, 98, 16);
301. editInfoPanel.add(lblFirstName\_1);
303. JLabel lblLastName\_1 = **new** JLabel("Last Name");
304. lblLastName\_1.setBounds(35, 118, 98, 16);
305. editInfoPanel.add(lblLastName\_1);
307. JLabel lblEmail\_1 = **new** JLabel("E-Mail");
308. lblEmail\_1.setBounds(35, 192, 98, 16);
309. editInfoPanel.add(lblEmail\_1);
311. JLabel lblPhoneNumber = **new** JLabel("Phone Number");
312. lblPhoneNumber.setBounds(354, 37, 116, 16);
313. editInfoPanel.add(lblPhoneNumber);
314. *//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*SUBMIT EDIT INFO\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**
315. JButton btnSubmit = **new** JButton("Submit");
316. btnSubmit.addActionListener(**new** ActionListener() {
317. **public** **void** actionPerformed(ActionEvent e) {
318. String f,l,em,p;
319. f = editFirstName.getText();
320. l = editLastName.getText();
321. em = editEmail.getText();
322. p = editPhoneNumber.getText();
323. *//reset text fields*
324. editFirstName.setText("");
325. editLastName.setText("");
326. editEmail.setText("");
327. editPhoneNumber.setText("");
329. **try** {
330. user.update(f,l,em,p);
331. } **catch** (SQLException e1) {
332. *// TODO Auto-generated catch block*
333. e1.printStackTrace();
334. }
335. *//Some method in user that will take this information and run an update query*
337. editInfoPanel.setVisible(**false**);
338. userInfoPanel.setVisible(**true**);
339. creatingUserInfoTable();
341. }
342. });
343. btnSubmit.setBounds(417, 317, 116, 57);
344. editInfoPanel.add(btnSubmit);
346. deleteBtn = **new** JButton("Delete");
347. deleteBtn.addActionListener(**new** ActionListener() {
348. **public** **void** actionPerformed(ActionEvent e) {
349. **int** input = JOptionPane.showConfirmDialog(**null**, "Do you want to proceed?", "Deleting User...",
350. JOptionPane.YES\_NO\_OPTION, JOptionPane.WARNING\_MESSAGE);
352. *// 0=yes, 1=no*
353. System.out.println(input);
354. **if**(input==0)
355. {
356. String em,p;
357. em = editEmail.getText();
358. p = editPhoneNumber.getText();
359. *//reset text fields*
360. editFirstName.setText("");
361. editLastName.setText("");
362. editEmail.setText("");
363. editPhoneNumber.setText("");
365. **try** {
366. user.removeUser(em,p);
367. } **catch** (SQLException e1) {
368. *// TODO Auto-generated catch block*
369. e1.printStackTrace();
370. }
371. *//Some method in user that will take this information and run a delete query*
373. editInfoPanel.setVisible(**false**);
374. loginPanel.setVisible(**true**);
375. }
376. }
377. });
378. deleteBtn.setBounds(88, 317, 116, 57);
379. editInfoPanel.add(deleteBtn);
381. btnCancel = **new** JButton("Cancel");
382. btnCancel.addActionListener(**new** ActionListener() {
383. **public** **void** actionPerformed(ActionEvent e) {
384. userInfoPanel.setVisible(**true**);
385. editInfoPanel.setVisible(**false**);
386. creatingUserInfoTable();
387. userInfoPanel.add(userInfoTable);
388. }
389. });
390. btnCancel.setBounds(253, 317, 116, 57);
391. editInfoPanel.add(btnCancel);
392. *//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*End button function\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**
394. createNewUserPanel = **new** JPanel();
395. frame.getContentPane().add(createNewUserPanel, "name\_178198354646768");
396. createNewUserPanel.setLayout(**null**);
397. createNewUserPanel.setVisible(**false**);
399. JLabel lblUserName\_1 = **new** JLabel("User Name");
400. lblUserName\_1.setBounds(60, 63, 103, 32);
401. createNewUserPanel.add(lblUserName\_1);
403. JLabel lblPassword\_1 = **new** JLabel("Password");
404. lblPassword\_1.setBounds(60, 185, 86, 16);
405. createNewUserPanel.add(lblPassword\_1);
407. JLabel lblConfirmPassword = **new** JLabel("Confirm Password");
408. lblConfirmPassword.setBounds(60, 274, 134, 32);
409. createNewUserPanel.add(lblConfirmPassword);
411. createNewUserName = **new** JTextField();
412. createNewUserName.setBounds(60, 103, 116, 22);
413. createNewUserPanel.add(createNewUserName);
414. createNewUserName.setColumns(10);
416. createNewPassword = **new** JPasswordField();
417. createNewPassword.setBounds(60, 214, 116, 22);
418. createNewUserPanel.add(createNewPassword);
419. createNewPassword.setColumns(10);
421. confirmNewPassword = **new** JPasswordField();
422. confirmNewPassword.setBounds(60, 319, 116, 22);
423. createNewUserPanel.add(confirmNewPassword);
424. confirmNewPassword.setColumns(10);
426. JButton submitBtn = **new** JButton("Submit");
427. submitBtn.addActionListener(**new** ActionListener() {
428. **public** **void** actionPerformed(ActionEvent e)
429. {
431. **boolean** flag = **false**;
432. String newUserName,newPassword,cNewPassword;
433. */\**
434. *\* this gathers the information typed into the text fields on the window*
435. *\*/*
436. newUserName = createNewUserName.getText();
437. newPassword = createNewPassword.getText();
438. cNewPassword = confirmNewPassword.getText();
440. *//if statement to make sure confirm new password = createNewpassword*
441. **if**(cNewPassword.equals(newPassword))
442. {
443. System.out.println("Trying to create user");
444. user = **new** User(newUserName, newPassword);
446. **try** {
447. flag = user.createNewUser();
448. } **catch** (SQLException e1) {
449. *// TODO Auto-generated catch block*
450. e1.printStackTrace();
451. }
453. **if**(flag)
454. {
455. *//reset text fields*
456. createNewUserName.setText("");          *//*
457. createNewPassword.setText("");          *// CHANGESS*
458. confirmNewPassword.setText("");         *//*
459. createNewUserPanel2.setVisible(**true**);
460. createNewUserPanel.setVisible(**false**);
461. }
463. }
464. **else**
465. {
466. lblPasswordsDoNot.setVisible(**true**);         *// CHANGESS*
467. System.out.println("passwords do not match");
468. }


472. }
473. });
474. submitBtn.setBounds(395, 318, 97, 25);
475. createNewUserPanel.add(submitBtn);
477. btnCancel\_1 = **new** JButton("Cancel");
478. btnCancel\_1.addActionListener(**new** ActionListener() {
479. **public** **void** actionPerformed(ActionEvent e) {
480. createNewUserName.setText("");
481. createNewPassword.setText("");
482. confirmNewPassword.setText("");
483. lblPasswordsDoNot.setVisible(**false**);
484. createNewUserPanel.setVisible(**false**);
485. employeePanel.setVisible(**true**);
486. employeePanel.add(tablePopulateData);
487. DefaultTableModel model = (DefaultTableModel) tablePopulateData.getModel();
488. model.setRowCount(0);
489. tablePopulateData.setPreferredScrollableViewportSize(**new** Dimension(600,300));
490. tablePopulateData.setFillsViewportHeight(**true**);
491. JScrollPane js=**new** JScrollPane(tablePopulateData);
492. js.setVisible(**true**);
493. employeePanel.add(js);
495. tablePopulateData.setModel(**new** DefaultTableModel(
496. **new** Object[][] {
497. {**null**, **null**, **null**, **null**, **null**, **null**},
498. },
499. **new** String[] {
500. "First Name", "Last Name", "E-Mail", "Tier", "Phone Number","Employee"
501. }));
502. }
503. });
504. btnCancel\_1.setBounds(243, 318, 97, 25);
505. createNewUserPanel.add(btnCancel\_1);
507. lblPasswordsDoNot = **new** JLabel("Passwords Do Not Match");
508. lblPasswordsDoNot.setBounds(297, 185, 176, 16);
509. lblPasswordsDoNot.setVisible(**false**);
510. createNewUserPanel.add(lblPasswordsDoNot);
512. createNewUserPanel2 = **new** JPanel();
513. frame.getContentPane().add(createNewUserPanel2, "name\_178198359276470");
514. createNewUserPanel2.setLayout(**null**);
515. createNewUserPanel2.setVisible(**false**);
517. JLabel lblFirstName = **new** JLabel("First Name");
518. lblFirstName.setBounds(66, 80, 103, 27);
519. createNewUserPanel2.add(lblFirstName);
521. JLabel lblLastName = **new** JLabel("Last Name");
522. lblLastName.setBounds(66, 155, 86, 27);
523. createNewUserPanel2.add(lblLastName);
525. JLabel lblEmployee = **new** JLabel("Employee");
526. lblEmployee.setBounds(66, 238, 56, 16);
527. createNewUserPanel2.add(lblEmployee);
529. JLabel lblPhone = **new** JLabel("Phone Number");
530. lblPhone.setBounds(298, 82, 103, 22);
531. createNewUserPanel2.add(lblPhone);
533. JLabel lblEmail = **new** JLabel("E-Mail");
534. lblEmail.setBounds(298, 160, 56, 16);
535. createNewUserPanel2.add(lblEmail);
537. JLabel lblTier = **new** JLabel("Tier");
538. lblTier.setBounds(298, 238, 56, 16);
539. createNewUserPanel2.add(lblTier);
541. firstNameText = **new** JTextField();
542. firstNameText.setBounds(66, 109, 116, 22);
543. createNewUserPanel2.add(firstNameText);
544. firstNameText.setColumns(10);
546. lastNameText = **new** JTextField();
547. lastNameText.setBounds(66, 195, 116, 22);
548. createNewUserPanel2.add(lastNameText);
549. lastNameText.setColumns(10);
551. employeeText = **new** JTextField();
552. employeeText.setBounds(66, 274, 116, 22);
553. createNewUserPanel2.add(employeeText);
554. employeeText.setColumns(10);
556. phoneNumberText = **new** JTextField();
557. phoneNumberText.setBounds(298, 109, 116, 22);
558. createNewUserPanel2.add(phoneNumberText);
559. phoneNumberText.setColumns(10);
561. emailText = **new** JTextField();
562. emailText.setBounds(298, 189, 116, 22);
563. createNewUserPanel2.add(emailText);
564. emailText.setColumns(10);
566. tierText = **new** JTextField();
567. tierText.setBounds(298, 274, 116, 22);
568. createNewUserPanel2.add(tierText);
569. tierText.setColumns(10);
571. JButton submitBtn2 = **new** JButton("Submit");
572. submitBtn2.addActionListener(**new** ActionListener() {
573. **public** **void** actionPerformed(ActionEvent e)
574. {
575. String fName,lName,phoneNumber,email,tier,employee;
576. */\**
577. *\* this gathers the information typed into the text fields on the window*
578. *\*/*
579. fName = firstNameText.getText();
580. lName = lastNameText.getText();
581. employee = employeeText.getText();
582. phoneNumber =  phoneNumberText.getText();
583. email =  emailText.getText();
584. tier = tierText.getText();

587. *//inserts user info into the database*
588. **try** {
589. user.createUserInfo(fName,lName,email,tier,phoneNumber,employee);
590. } **catch** (SQLException e1) {
591. *// TODO Auto-generated catch block*
592. e1.printStackTrace();
593. }
595. *//reset text fields*
596. firstNameText.setText("");
597. lastNameText.setText("");
598. employeeText.setText("");
599. phoneNumberText.setText("");
600. emailText.setText("");
601. tierText.setText("");
603. loginPanel.setVisible(**true**);
604. createNewUserPanel2.setVisible(**false**);
606. tablePopulateData.setModel(**new** DefaultTableModel(
607. **new** Object[][] {
608. {**null**, **null**, **null**, **null**, **null**, **null**},
609. },
610. **new** String[] {
611. "First Name", "Last Name", "E-Mail", "Tier", "Phone Number","Employee"
612. }));
613. }
614. });
615. submitBtn2.setBounds(435, 341, 97, 25);
616. createNewUserPanel2.add(submitBtn2);
618. employeePanel = **new** JPanel();
619. frame.getContentPane().add(employeePanel, "name\_1648590061736022");
620. employeePanel.setVisible(**false**);
621. employeePanel.setLayout(**new** FlowLayout(FlowLayout.CENTER, 5, 5));

624. *//Searching Area - Label*
625. lblSearchEmail = **new** JLabel("Search By E-Mail");
626. employeePanel.add(lblSearchEmail);
628. *//Search TextField, clicking on it deletes the text.*
629. searchText = **new** JTextField();
630. searchText.setText("Enter e-mail");
631. searchText.addMouseListener(**new** MouseAdapter() {
632. @Override
633. **public** **void** mouseClicked(MouseEvent arg0) {
634. searchText.setText("");
635. }
636. });
638. *//Every key type send query to database to find matched cases.*
639. searchText.addKeyListener(**new** KeyAdapter() {
640. @Override
641. **public** **void** keyReleased(KeyEvent arg0) {
642. **try** {
643. DefaultTableModel model = (DefaultTableModel) tablePopulateData.getModel();
644. model.setRowCount(0);
645. user.getSearchedInfo(tablePopulateData, searchText.getText());
646. } **catch** (SQLException e1) {
647. e1.printStackTrace();
648. }
649. }
650. });
652. employeePanel.add(searchText);
653. searchText.setColumns(10);
654. tablePopulateData = **new** JTable();
655. employeePanel.add(tablePopulateData);
657. tablePopulateData.setModel(**new** DefaultTableModel(
658. **new** Object[][] {
659. {**null**, **null**, **null**, **null**, **null**, **null**},
660. },
661. **new** String[] {
662. "First Name", "Last Name", "E-Mail", "Tier", "Phone Number","Employee"
663. }));


667. btnLogOut = **new** JButton("Log Out");
668. btnLogOut.addActionListener(**new** ActionListener() {
669. **public** **void** actionPerformed(ActionEvent arg0) {
670. employeePanel.setVisible(**false**);
671. employeePanel.remove(tablePopulateData);
672. loginPanel.setVisible(**true**);
673. loginLabel.setText("Logout: Success");
675. tablePopulateData.setModel(**new** DefaultTableModel(
676. **new** Object[][] {
677. {**null**, **null**, **null**, **null**, **null**, **null**},
678. },
679. **new** String[] {
680. "First Name", "Last Name", "E-Mail", "Tier", "Phone Number","Employee"
681. }));
682. }
683. });
685. createNew = **new** JButton("Create New User");
686. createNew.addActionListener(**new** ActionListener() {
687. **public** **void** actionPerformed(ActionEvent arg0)
688. {
689. employeePanel.setVisible(**false**);
690. createNewUserPanel.setVisible(**true**);
691. }
692. });
693. employeePanel.add(createNew);
694. employeePanel.add(btnLogOut);
696. editOptionsPanel = **new** JPanel();
697. frame.getContentPane().add(editOptionsPanel, "name\_1211094242985247");
698. editOptionsPanel.setLayout(**null**);
700. chckbxPool = **new** JCheckBox("Pool");
701. chckbxPool.setBounds(63, 145, 113, 25);
702. chckbxPool.setSelected(**false**);
703. editOptionsPanel.add(chckbxPool);
705. chckbxTennis = **new** JCheckBox("Tennis");
706. chckbxTennis.setBounds(182, 145, 76, 25);
707. editOptionsPanel.add(chckbxTennis);

710. chckbxFootballField = **new** JCheckBox("Football Field");
711. chckbxFootballField.setBounds(332, 145, 113, 25);
712. editOptionsPanel.add(chckbxFootballField);
714. chckbxZoomba = **new** JCheckBox("Zoomba");
715. chckbxZoomba.setBounds(63, 214, 113, 25);
716. editOptionsPanel.add(chckbxZoomba);
718. chckbxYoga = **new** JCheckBox("Yoga");
719. chckbxYoga.setBounds(182, 214, 113, 25);
720. editOptionsPanel.add(chckbxYoga);
721. editOptionsPanel.setVisible(**false**);
722. *////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////*
723. btnSubmit\_1 = **new** JButton("Submit");
724. btnSubmit\_1.addActionListener(**new** ActionListener()
725. {
726. **public** **void** actionPerformed(ActionEvent arg0) {
727. **int** count = 0;
728. **for**(**int** i =0;i<=4;i++)
729. {
730. updateOptions.add("0");
731. }
732. **int** userTier = Integer.parseInt(user.getTier());


736. **if**(chckbxPool.isSelected())
737. {
739. updateOptions.set(0, "1");
741. count++;
742. chckbxPool.setSelected(**false**);

745. }
746. **if**(chckbxTennis.isSelected())
747. {
749. updateOptions.set(1, "1");
750. count++;
751. chckbxTennis.setSelected(**false**);
752. }
754. **if**(chckbxFootballField.isSelected())
755. {
757. updateOptions.set(2, "1");
758. count++;
759. chckbxFootballField.setSelected(**false**);
760. }
761. **if**(chckbxZoomba.isSelected())
762. {
763. updateOptions.set(3, "1");
764. count++;
765. chckbxZoomba.setSelected(**false**);
767. }
768. **if**(chckbxYoga.isSelected())
769. {
770. updateOptions.set(4, "1");
771. count++;
772. chckbxYoga.setSelected(**false**);
773. }
775. **if**(count==userTier)
776. {
777. **try** {
778. user.updateTierOptions(updateOptions);
779. **for**(**int** i =0;i<=4;i++)
780. {
781. updateOptions.set(i, "0");
782. }
783. editOptionsPanel.setVisible(**false**);
784. userInfoPanel.setVisible(**true**);
785. creatingUserInfoTable();
787. } **catch** (SQLException e) {
788. *// TODO Auto-generated catch block*
789. e.printStackTrace();
790. }
791. }
793. **if**(count>userTier || count<userTier)
794. {
795. **if**(count==0)
796. {
797. JOptionPane.showMessageDialog(**null**, "Please make "+userTier+" selection");
798. }
799. **else** {
800. JOptionPane.showMessageDialog(**null**, "You must select " + userTier + " different options. Please try again");
801. **for**(**int** i =0;i<=4;i++)
802. {
803. updateOptions.set(i, "0");
804. }
805. }

808. }
809. }
810. });
811. btnSubmit\_1.setBounds(410, 345, 97, 25);
812. editOptionsPanel.add(btnSubmit\_1);
813. editOptionsPanel.setVisible(**false**);
814. }
816. **public** **void** getSearchedTable()
817. {
819. **try** {
820. *// grabs the information from user.getInfo() and equals it to the array dbinfo*
821. dbinfo = user.getInfo();
823. } **catch** (SQLException e2) {
824. e2.printStackTrace();
825. }
827. *//creates table for user information*
828. userInfoTable = **new** JTable();
829. *//sets parameters for table and out puts the information from dbinfo onto the table .*
830. userInfoTable.setModel(**new** DefaultTableModel(
831. **new** Object[][] {
832. { "First Name", "Last Name", "E-Mail", "Tier", "Phone Number"},
833. {dbinfo.get(0), dbinfo.get(1), dbinfo.get(2), dbinfo.get(3), dbinfo.get(4)},
834. },
835. **new** String[] {
836. "First Name", "Last Name", "E-Mail", "Tier", "Phone Number"
837. }
838. ));
839. userInfoTable.getColumnModel().getColumn(0).setPreferredWidth(157);
840. userInfoTable.getColumnModel().getColumn(1).setPreferredWidth(157);
841. userInfoTable.getColumnModel().getColumn(2).setPreferredWidth(300);
842. userInfoTable.getColumnModel().getColumn(3).setPreferredWidth(40);
843. userInfoTable.getColumnModel().getColumn(4).setPreferredWidth(200);
845. userInfoTable.setBounds(77, 137, 515, 35);
846. *//actually adds the the table to the userInfoPanel*
847. userInfoPanel.add(userInfoTable);
848. }
850. **public** **void** creatingUserInfoTable()
851. {
853. **try** {
854. *// grabs the information from user.getInfo() and equals it to the array dbinfo*
855. dbinfo = user.getInfo();
857. } **catch** (SQLException e2) {
858. e2.printStackTrace();
859. }
861. *//creates table for user information*
862. userInfoTable = **new** JTable();
863. *//sets parameters for table and out puts the information from dbinfo onto the table .*
864. userInfoTable.setModel(**new** DefaultTableModel(
865. **new** Object[][] {
866. { "First Name", "Last Name", "E-Mail", "Tier", "Phone Number"},
867. {dbinfo.get(0), dbinfo.get(1), dbinfo.get(2), dbinfo.get(3), dbinfo.get(4)},
868. },
869. **new** String[] {
870. "First Name", "Last Name", "E-Mail", "Tier", "Phone Number"
871. }
872. ));
873. userInfoTable.getColumnModel().getColumn(0).setPreferredWidth(157);
874. userInfoTable.getColumnModel().getColumn(1).setPreferredWidth(157);
875. userInfoTable.getColumnModel().getColumn(2).setPreferredWidth(300);
876. userInfoTable.getColumnModel().getColumn(3).setPreferredWidth(40);
877. userInfoTable.getColumnModel().getColumn(4).setPreferredWidth(200);
879. userInfoTable.setBounds(77, 137, 515, 35);
880. *//actually adds the the table to the userInfoPanel*
881. userInfoPanel.add(userInfoTable);
883. **try** {
884. dbinfo2 = user.getInfo2();
885. } **catch** (SQLException e) {
886. *// TODO Auto-generated catch block*
887. e.printStackTrace();
888. }
889. userInfoTable2 = **new** JTable();
890. userInfoTable2.setModel(**new** DefaultTableModel(
891. **new** Object[][] {
892. {"Pool", "Tennis Court", "Football Field", "Zoomba Class", "Yoga Class"},
893. {dbinfo2.get(0), dbinfo2.get(1), dbinfo2.get(2), dbinfo2.get(3), dbinfo2.get(4)},
894. },
895. **new** String[] {
896. "Pool", "Tennis Court", "Football Field", "Zoomba Class", "Yoga Class"
897. }
898. ));
899. userInfoTable2.setBounds(26, 230, 563, 35);
900. userInfoPanel.add(userInfoTable2);
901. }
902. }
903. **package** legDay;
905. **import** java.sql.Connection;
906. **import** java.sql.DriverManager;
907. **import** java.sql.ResultSet;
908. **import** java.sql.SQLException;
909. **import** java.sql.Statement;
910. **import** java.util.ArrayList;
911. **import** javax.swing.JTable;
912. **import** javax.swing.table.DefaultTableModel;
913. **import** org.mindrot.jbcrypt.BCrypt;


917. **public** **class** User
918. {
919. **private** String user;*// uname*
920. **private** String password; *// upassword*
921. **private** String eMail;
922. **private** Connection c;
923. **private** String phone;
925. **public** User(String user, String password)
926. {
927. **this**.user = user;
928. **this**.password = password;
929. }
931. **private** String uTier;
933. **public** String getTier()
934. {
935. **return** uTier;
936. }
938. **public** **void** connect() **throws** SQLException
939. {
940. *//String dburl = "jdbc:postgresql://broner.servebeer.com:35151/gymdatabase";*
941. *//String dbuser = "projectlogin";*
942. *//String dbpassword = "projectlogin";*
944. String dburl = "jdbc:postgresql://localhost:5432/gymdatabase";
945. String dbuser = "postgres";
946. String dbpassword = "SQL";
947. c = DriverManager.getConnection(dburl, dbuser, dbpassword);
948. **if** (c != **null**)
949. {
950. System.out.println("Connected to Database");
951. }
952. }
954. **public** **boolean** isEmployee() **throws** SQLException
955. {
956. connect();
957. String query = "SELECT employee FROM client WHERE Email = '" + eMail + "'";
958. Statement st = c.createStatement();
959. ResultSet rs = st.executeQuery(query);
961. **while**(rs.next())
962. {
963. String check = "0";
964. String isEmp = rs.getString(1);
966. **if**(isEmp.equals(check))
967. {
968. *//user is not employee*
969. **return** **false**;
970. }
971. }
972. *//user is employee*
973. c.close();
974. **return** **true**;
975. }
977. *//Checks for the user name in the database and compares the data input to the data stored to determine if the login is successful or not*
978. **public** **boolean** loginVerify() **throws** SQLException
979. {
980. connect();
981. **boolean** login = **false**;
982. String query = "SELECT \* FROM login WHERE uname = '" + user + "'";
983. Statement st = c.createStatement();
984. ResultSet rs = st.executeQuery(query);
986. **while**(rs.next())
987. {
988. String testuser = rs.getString(1);
989. String testpassword = rs.getString(2);              *// testpassword IS the saved hash in the database*
990. System.out.println(testpassword);
992. **if**(testuser.equals(user) && (BCrypt.checkpw(password, testpassword))){      *// password is the plaintext pulled from JTextfield*
993. System.out.println("Login Successful");                                 *// used being compared in BCrypt.checkpw Method*
994. login = **true**;
995. }
996. }
997. String query2 = "SELECT email FROM relationship WHERE uname = '" + user + "'";
998. ResultSet rs2 = st.executeQuery(query2);
1000. **while**(rs2.next())
1001. {
1002. **this**.eMail = rs2.getString(1);
1003. System.out.println(eMail);
1004. }
1005. rs2.close();
1007. rs.close();
1008. c.close();
1009. **return** login;
1010. }
1012. **public**  **void** getAllInfo(JTable tablePopulateData) **throws** SQLException
1013. {
1014. connect();
1015. JTable tpd = tablePopulateData;
1016. String query = "TABLE client";
1018. Statement st = c.createStatement();
1020. ResultSet rs = st.executeQuery(query);
1021. **while**(rs.next())
1022. {
1023. String a = rs.getString(1);
1024. String b = rs.getString(2);
1025. String c = rs.getString(3);
1026. String d = rs.getString(4);
1027. String e = rs.getString(5);
1028. String f = rs.getString(6);
1030. DefaultTableModel dftable = (DefaultTableModel) tpd.getModel();
1031. Object[] obj =  {a,b,c,d,e,f};
1033. dftable.addRow(obj);
1034. }
1035. rs.close();
1036. c.close();
1037. }
1039. *//search query*
1040. **public**  **void** getSearchedInfo(JTable tablePopulateData, String searchMail) **throws** SQLException
1041. {
1042. connect();
1043. JTable tpd = tablePopulateData;
1044. String query = "SELECT \* FROM client WHERE Email LIKE '%"+searchMail+"%'";
1045. System.out.println(query);
1047. Statement st = c.createStatement();
1049. ResultSet rs = st.executeQuery(query);
1051. **while**(rs.next())
1052. {
1053. String a = rs.getString(1);
1054. String b = rs.getString(2);
1055. String c = rs.getString(3);
1056. String d = rs.getString(4);
1057. String e = rs.getString(5);
1058. String f = rs.getString(6);
1060. DefaultTableModel dftable = (DefaultTableModel) tpd.getModel();
1061. Object[] obj =  {a,b,c,d,e,f};
1063. dftable.addRow(obj);
1064. }
1065. rs.close();
1066. c.close();
1067. }
1069. **public** ArrayList<String> getInfo() **throws** SQLException
1070. {
1072. System.out.println("Get Info Start");
1073. connect();
1074. ArrayList<String> uinfo = **new** ArrayList<String>();
1075. String query = "SELECT \* FROM client WHERE Email ='" + eMail + "'";
1076. System.out.println("Get Info: " + eMail);
1078. Statement st = c.createStatement();
1080. ResultSet rs = st.executeQuery(query);
1082. **while**(rs.next())
1083. {
1084. **for**(**int** i=1;i<=6;i++)
1085. {
1086. uinfo.add(rs.getString(i));
1087. }
1088. }
1089. **this**.phone = uinfo.get(4);
1090. **this**.uTier = uinfo.get(3);
1091. rs.close();
1092. c.close();
1093. **return** uinfo;
1094. }
1096. **public** ArrayList<String> getInfo2() **throws** SQLException
1097. {
1099. System.out.println("Get Info Start");
1100. connect();
1101. ArrayList<String> uinfo = **new** ArrayList<String>();
1102. String query = "SELECT \* FROM tier\_options WHERE phone\_number = (SELECT phone\_number "
1103. + "FROM client WHERE email ='" + eMail + "')";
1104. System.out.println("Get Info: " + eMail);
1106. Statement st = c.createStatement();
1108. ResultSet rs = st.executeQuery(query);
1110. **while**(rs.next())
1111. {
1112. **for**(**int** i=3;i<=7;i++)
1113. {
1114. uinfo.add(rs.getString(i));
1115. }
1116. }
1117. rs.close();
1118. c.close();
1119. **return** uinfo;
1120. }
1122. **public** **boolean** userExists() **throws** SQLException
1123. {
1124. String cUser=**null**;
1125. connect();
1127. String query = "SELECT uname FROM login WHERE uname='" + user + "'";
1128. Statement st = c.createStatement();
1129. ResultSet rs = st.executeQuery(query);
1130. c.close();
1132. **while**(rs.next())
1133. {
1134. cUser=rs.getString(1);
1135. }
1136. rs.close();
1137. **if**(user.equals(cUser))
1138. {
1139. *//user does exist*
1140. **return** **true**;
1141. }
1142. *//user does not exist*
1143. **return** **false**;
1144. }

1147. **public** **void** updateTierOptions(ArrayList<String> update) **throws** SQLException
1148. {
1149. connect();
1151. */\* takes user input from GUI for respective variables and inputs the value into the SQL query to update the information*
1152. *in the database. Information that is not changed in the GUI is left as the existing value.*
1153. *\*/*
1154. String query = "UPDATE tier\_options set pool = '" + update.get(0) + "' where phone\_number = '" + phone + "';"
1155. + "UPDATE tier\_options set tennis\_court = '" + update.get(1) + "' where phone\_number = '" + phone + "';"
1156. + "UPDATE tier\_options set football\_field = '" + update.get(2) + "' where phone\_number = '" + phone + "';"
1157. + "Update tier\_options set zoomba\_class = '" + update.get(3) + "' where phone\_number = '" + phone + "';"
1158. + "UPDATE tier\_options set yoga\_class = '" + update.get(4) + "' where phone\_number = '" + phone + "';";

1161. Statement st = c.createStatement();
1162. st.executeUpdate(query);
1164. c.close();
1166. }
1168. *//populates user information.*
1169. **public** **void** createUserInfo(String fName, String lName, String email,String tier, String phoneNumber,String employee) **throws** SQLException
1170. {
1171. String firstName=fName,lastName=lName,eMail2=email,phoneNumber2=phoneNumber,tier2=tier,employee2=employee;
1172. connect();
1173. String query = "INSERT INTO client VALUES ('"+firstName+"','"+lastName+"','"+eMail2+"','"+tier2+"','"+phoneNumber2+"','"+employee2+"');"
1174. + "INSERT INTO relationship VALUES ('"+user+"','"+eMail2+"');"
1175. + "INSERT INTO tier\_options VALUES ('"+phoneNumber2+"','"+tier2+"','0','0','0','0','0')";
1176. Statement st= c.createStatement();
1177. st.executeUpdate(query);
1179. c.close();
1180. }

1183. **public** **boolean** createNewUser() **throws** SQLException
1184. {
1186. **boolean** flag = **false**;
1188. flag = userExists();
1190. **if**(flag==**false**)
1191. {
1192. connect();
1193. String hashed = BCrypt.hashpw(password, BCrypt.gensalt());                       *// BCrypt.hashpw hashes password pulled from JTextfield*
1194. String query = "INSERT INTO login VALUES ('" + user + "', '" + hashed + "')";    *// BCrypt.gensalt uses default value 10 for parameter*
1195. Statement st = c.createStatement();                                              *// String strong\_salt = BCrypt.gensalt(log\_rounds)*
1196. st.executeUpdate(query);                                                         *// String stronger\_salt = BCrypt.gensalt(12)*
1197. c.close();                                                                       *// Workload increases exponentially, 2\*\*log\_rounds , ranges from 4-31*
1198. **return** **true**;
1199. }
1200. System.out.println("User Name already exists");
1201. **return** **false**;
1203. }
1205. **public** **void** update(String fname, String lname, String eMail2, String phone) **throws** SQLException
1206. {
1207. connect();
1209. String query = "UPDATE client SET first\_name = '" + fname + "' WHERE email = '" + eMail + "';"
1210. + "UPDATE client SET last\_name = '" + lname + "' WHERE email = '" + eMail + "';"
1211. + "UPDATE client SET email = '" + eMail2 + "' WHERE email = '" + eMail + "';"
1212. + "UPDATE client SET phone\_number = '" + phone + "' WHERE email = '" + eMail + "';";
1213. Statement st = c.createStatement();
1214. st.executeUpdate(query);
1215. **this**.eMail = eMail2;
1216. c.close();
1217. }
1219. **public** **void** removeUser(String email, String phone) **throws** SQLException
1220. {
1221. connect();
1222. String query = "DELETE FROM login WHERE uname = '" + user + "';"
1223. + "DELETE FROM client WHERE email = '" + email + "';"
1224. + "DELETE FROM relationship WHERE uname = '" + user + "';"
1225. + "DELETE FROM tier\_options WHERE phone\_number = '" + phone + "';";
1226. Statement st = c.createStatement();
1227. st.executeUpdate(query);
1228. c.close();
1229. }
1230. }