**CS 5200 Introduction to Database Management Homework #4**

For the following SQL schema:

create table Building (

id int primary key,

address varchar(5000) not null

)

create table Apartment (

id int primary key,

number varchar(31) not null,

building int not null references Building(id)

on update cascade on delete cascade

)

create table Person (

id int primary key,

name varchar(5000) not null

)

create table Owner (

person int references Person(id)

on update cascade on delete cascade,

apartment int references Apartment(id)

on update cascade on delete cascade,

primary key(person, apartment)

)

1. Install a database system and create the tables above.

2. Develop methods using JDBC (or its equivalent in another language) to perform the following tasks:

1. Given an address and a number of apartments, creates a building and the apartments in the building. Generate apartment numbers using the letter "A" followed by the apartment id.
2. Given a name, create a person.
3. Given a person name and an apartment number and building address, make that person one of the owners of the apartment.
4. Given an apartment number and building address, remove all owners of the apartment.
5. Given a building id, list all apartments in the building (both id and number), and for each apartment list the owners (both name and id)
6. import java.sql.\*;
7. import java.util.\*;
8. /\*\*
9. \* Apartment Ownership Methods.
10. \* The methods assume that all id primary keys are generated by auto incrementation.
11. \* @author Ken Baclawski
12. \*/
13. public class Asst4 {
14. /\*\*
15. \* The database connection.
16. \*/
17. private Connection connection;
18. /\*\*
19. \* Apartment information.
20. \*/
21. public class Apartment {
22. private int id;
23. private String number;
24. private List<Person> owners = new ArrayList<Person>();
25. /\*\*
26. \* Construct an empty apartment object.
27. \*/
28. public Apartment() {}
29. /\*\*
30. \* Get the identifier of the apartment.
31. \* @return The apartment id.
32. \*/
33. public int getId() {
34. return id;
35. }
36. /\*\*
37. \* Set the identifier of an apartment.
38. \* @param id The apartment identifier.
39. \*/
40. public void setId(int id) {
41. this.id = id;
42. }
43. /\*\*
44. \* Get the number of the apartment.
45. \* @return The apartment number.
46. \*/
47. public String getNumber() {
48. return number;
49. }
50. /\*\*
51. \* Set the number of an apartment.
52. \* @param number The apartment number.
53. \*/
54. public void setNumber(String number) {
55. this.number = number;
56. }
57. /\*\*
58. \* Get the owners of the apartment.
59. \* @return The list of apartment owners.
60. \*/
61. public List<Person> getOwners() {
62. return owners;
63. }
64. /\*\*
65. \* Show the apartments and their owners.
66. \* @return The apartments and their owners as a string.
67. \*/
68. public String toString() {
69. StringBuilder b = new StringBuilder("Apartment id ");
70. b.append(id).append(" number ").append(number);
71. for (Person p : owners) {
72. b.append("\n owned by ").append(p);
73. }
74. return b.toString();
75. }
76. }
77. /\*\*
78. \* Person information.
79. \*/
80. public class Person {
81. private int id;
82. private String name;
83. /\*\*
84. \* Construct an empty person object.
85. \*/
86. public Person() {}
87. /\*\*
88. \* Get the identifier of the person.
89. \* @return The person id.
90. \*/
91. public int getId() {
92. return id;
93. }
94. /\*\*
95. \* Set the identifier of an person.
96. \* @param id The person identifier.
97. \*/
98. public void setId(int id) {
99. this.id = id;
100. }
101. /\*\*
102. \* Get the name of the person.
103. \* @return The person name.
104. \*/
105. public String getName() {
106. return name;
107. }
108. /\*\*
109. \* Set the name of an person.
110. \* @param name The person name.
111. \*/
112. public void setName(String name) {
113. this.name = name;
114. }
115. /\*\*
116. \* Show the person
117. \* @return The person as a string.
118. \*/
119. public String toString() {
120. return "Person id " + id + " name " + name;
121. }
122. }
123. /\*\*
124. \* Construct an ownership object.
125. \* @param driver The driver class name.
126. \* @param url The database URL.
127. \* @param user The database user.
128. \* @param password The database password.
129. \*/
130. public Asst4(String driver, String url, String user, String password) {
131. try {
132. Class.forName(driver);
133. connection = DriverManager.getConnection(url, user, password);
134. } catch (Exception e) {
135. System.err.println("Unable to connect to the database due to " + e);
136. }
137. }
138. /\*\*
139. \* Create a new building.
140. \* @param address The building address.
141. \* @param apartmentCount The number of apartments to create.
142. \* @throws SQLException if a database error occurs.
143. \*/
144. public void createBuilding(String address, int apartmentCount) throws SQLException {
145. PreparedStatement createBuilding = connection.prepareStatement("insert into Building(address) values(?);");
146. createBuilding.setString(1, address);
147. createBuilding.executeUpdate();
148. ResultSet rs = createBuilding.getGeneratedKeys();
149. rs.first();
150. int buildingId = rs.getInt(1);
151. PreparedStatement createApartment = connection.prepareStatement("insert into Apartment(number,building) values('A',?);");
152. PreparedStatement updateApartment = connection.prepareStatement("update Apartment set number=? where id=?;");
153. for (int i = 0; i < apartmentCount; ++i) {
154. createApartment.setInt(1, buildingId);
155. createApartment.executeUpdate();
156. rs = createApartment.getGeneratedKeys();
157. rs.first();
158. int apartmentId = rs.getInt(1);
159. System.out.println("Updating apartment number to " + apartmentId);
160. updateApartment.setString(1, "A" + apartmentId);
161. updateApartment.setInt(2, apartmentId);
162. updateApartment.executeUpdate();
163. }
164. }
165. /\*\*
166. \* Create a new person.
167. \* @param name The name of the person.
168. \* @throws SQLException if a database error occurs.
169. \*/
170. public void createPerson(String name) throws SQLException {
171. PreparedStatement createPerson = connection.prepareStatement("insert into Person(name) values(?);");
172. createPerson.setString(1, name);
173. createPerson.executeUpdate();
174. }
175. /\*\*
176. \* Acquire a new apartment.
177. \* @param person The name of the person who will be an owner of the apartment.
178. \* @param address The address of the building.
179. \* @param apartment The apartment number.
180. \* @throws SQLException if a database error occurs.
181. \*/
182. public void acquireApartment(String person, String address, String apartment) throws SQLException {
183. PreparedStatement createOwnership = connection.prepareStatement
184. ("insert ignore into Owner(person,apartment) select p.id, a.id from Building b, Apartment a, Person p " +
185. "where b.address=? and b.id=a.building and a.number=? and p.name=?");
186. createOwnership.setString(1, address);
187. createOwnership.setString(2, apartment);
188. createOwnership.setString(3, person);
189. int updateCount = createOwnership.executeUpdate();
190. System.out.println("Number of ownership records inserted is " + updateCount);
191. }
192. /\*\*
193. \* Dispose ownership in an apartment by all owners.
194. \* @param address The address of the building.
195. \* @param apartment The apartment number.
196. \* @throws SQLException if a database error occurs.
197. \*/
198. public void disposeApartment(String address, String apartment) throws SQLException {
199. PreparedStatement deleteOwnership = connection.prepareStatement
200. ("delete from Owner where exists(select \* from Building b, Apartment a where b.address=? and b.id=a.building and a.number=? and apartment=a.id)");
201. deleteOwnership.setString(1, address);
202. deleteOwnership.setString(2, apartment);
203. deleteOwnership.executeUpdate();
204. }
205. /\*\*
206. \* List apartments and their owners.
207. \* @param buildingId The building identifier.
208. \* @return A list of all apartments in the building together with their owners.
209. \* @throws SQLException if a database error occurs.
210. \*/
211. public List<Apartment> listApartments(int buildingId) throws SQLException {
212. PreparedStatement findApartments = connection.prepareStatement
213. ("select a.id, a.number from Apartment a where a.building=?");
214. findApartments.setInt(1, buildingId);
215. PreparedStatement findOwners = connection.prepareStatement
216. ("select p.id, p.name from Owner o, Person p where o.apartment=? and o.person=p.id");
217. ResultSet apartments = findApartments.executeQuery();
218. List<Apartment> apartmentList = new ArrayList<Apartment>();
219. while (apartments.next()) {
220. Apartment apartment = new Apartment();
221. apartment.setId(apartments.getInt(1));
222. apartment.setNumber(apartments.getString(2));
223. List<Person> ownerList = apartment.getOwners();
224. findOwners.setInt(1, apartment.getId());
225. ResultSet owners = findOwners.executeQuery();
226. while (owners.next()) {
227. Person person = new Person();
228. person.setId(owners.getInt(1));
229. person.setName(owners.getString(2));
230. ownerList.add(person);
231. }
232. apartmentList.add(apartment);
233. }
234. return apartmentList;
235. }
236. /\*\*
237. \* Test program for the methods.
238. \*/
239. public static void main(String... args) throws Exception {
240. Asst4 asst4 = new Asst4("com.mysql.jdbc.Driver", "jdbc:mysql://localhost/a", "root", "abcd");
241. asst4.createBuilding("100 Main Street", 2);
242. asst4.createPerson("Fred");
243. asst4.createPerson("Fred");
244. asst4.acquireApartment("Fred", "100 Main Street", "A1");
245. for (Apartment a : asst4.listApartments(1)) {
246. System.out.println(a);
247. }
248. asst4.disposeApartment("100 Main Street", "A1");
249. for (Apartment a : asst4.listApartments(1)) {
250. System.out.println(a);
251. }
252. }
253. }