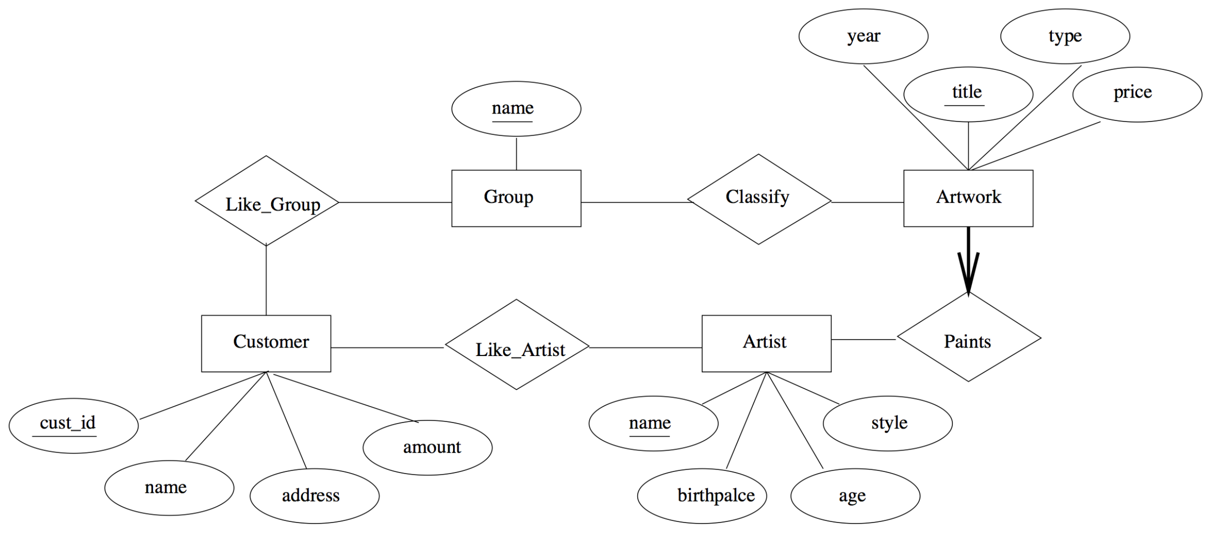
CS 430 - Database Systems
Homework Assignment 5

(Due April 8, Tuesday)

Recall the relational schema you've designed for an art gallery in assignment 2.

A database company called ArtBase builds a product for art galleries. The core of this product is a database with a schema that captures all the information that galleries need to maintain. Galleries keep information about artists, their names (which are unique), birthplaces, age and style of art. For each piece of artwork, the artist, the year it was made, its unique title, its type of art (e.g. painting, lithograph, sculpture, photograph), and its price must be stored. Pieces of artwork are also classified into groups of various kinds, for example, portraits, still lifes, works by Picaso, or works of the 19th century; a given piece may belong to more than one group. Each group is identified by a name (like those just given) that describes the group. Finally, galleries keep information about customers. For each customer, galleries keep that person's unique name, address, total amount of dollars spent in the gallery, and the artists and groups of art that the customer tends to like.

The ER diagram for this application is as follows.



A relational schema corresponding to the above ER diagram is given below.

```
artist(a_name:string, birthplace:string, age:int, style:string)
artwork(title:string, year:int, type:string, price:real, a_name:string)
customer(cust_id:string, c_name:string, address:string, amount:real)
```

```
a_group(g_name:string)
classify(title:string, g_name:string)
like_group(cust_id:string, g_name:string)
like_artist(cust_id:string, a_name:string)
```

1. **Implement this schema in the PostgreSQL instance running in the CS department.** Consider the following points while working on this section.

   - It's important that both primary key constraints as well as foreign key constraints are enforced.
   - You can assume that all String type attribute values will contain less than 100 characters.
   - For non integer numeric fields(`price` and `amount`) use the PostgreSQL data type `'double precision'` to inter-operate with Java `float` values. More on PostgreSQL data type is available in `http://www.postgresql.org/docs/9.1/static/datatype.html`

2. **Write a client side implementation to access this database with JDBC.**
   The guidelines for this implementation is given below. *It's important that these guidelines are strictly followed.*

   - There will be an archive called `assignment_5.tar` that you should download which contains the necessary files. It'll extract into a directory named `assignment_5`. It contains the following directories.
     - src - the source directory.
     - lib - the directory for third party libraries.
     - Makefile - Make file for compiling the source.
     - scripts - the directory to include the database scripts you will to writing as part of this assignment.

     It's important that you do not change this directory structure. Grading scripts are written assuming this directory structure.
   - There will be a set of functionality that you need to implement as described later in this document. Each of this functionality requires you to implement a Java method at the client side. Some of these tasks require you to write some procedures at the database as well. There is a class structure provided to be used with this assignment. It contains a stub class and a set of supporting classes.
     - **Stub Class**
       This class resides in the file `src/cs430/a5/Stub.java` This class contains empty methods for each of the tasks that needs to be implemented. You should implement the body of the each method as per the instructions provided later in this document. This class has a default constructor which should not be removed due to any reason. Also the signatures of the given methods should not be modified. The grading scripts assumes that the default constructor and the method signatures remain intact.
     - **Supporting Classes**
       Entity sets and relationship sets are modeled as a set of classes. An object of one of these classes represents a record in the corresponding table. These supporting classes are introduced to reduce the complexity of having to pass around several

primitive data types. These classes are available inside `src/cs430/a5/entity` directory. There will be seven classes corresponding to the seven tables in the relational schema. These classes contain attributes representing the fields in the corresponding table. For instance, the class `cs430.a5.entity.Artist` contains four attributes. `name`(String), `birthPlace`(String), `age`(int) and `style`(String) modeling the four fields `a_name`, `birthplace`, `age` and `style` in the artist table respectively.
  Do not change these supporting classes.
  - **You can implement your own classes while implementing these tasks. But do not change the method signatures of the `Stub` class or the supporting classes.**

- **Compiling the source**
  A make file is provided for compiling the code. The command `make all` will compile the code. You can use this file as it is. If you introduce packages beyond the depth of `cs430.a5.**`, then you will have to modify the line #9 of the file. This file is self explanatory. If you need any help modifying this file, please talk to the TA.
- **Use of third party libraries**
  It's required to use the PostgreSQL JDBC driver as a runtime library. You may need to download it and copy it to the `lib` directory. The make file provided for the building the project will include the content inside the `lib` directory to the classpath.

Following functionality needs to be implemented.

i). **Add an artist to the database.**
This task should be implemented inside the body of the
`public void addArtist(Artist artist) throws SQLException`
method of the `Stub` class. And object of type `Artist` will be passed as an argument. You may use the `getter` methods of this object to access it attribute values.

ii). **Add a customer to the database.**
This task should be implemented inside the body of the
`public void addCustomer(Customer customer) throws SQLException`
method of the `Stub` class.

iii). **Add an artwork to the database.**
This task should be implemented inside the body of the
`public void addArtwork(Artwork artwork, String group) throws SQLException`
method of the `Stub` class. The requirement is different than the previous two tasks. For this task you should implement a **'stored procedure'** at the database. This stored procedure should execute the following three tasks.

- Add the new artwork to the table `artwork`.
- Check if there is a group in the `a_group` table with the given group name. If not add a record with the given group name.
- Finally add an entry to the table `classify` with the `title` of the artwork and the `group name`.

At the client side, inside the `addArtwork` method, you should invoke this stored procedure.

When the stored procedure is added to the database, it will persist. But for grading purposes, save this stored procedure into a file called `q3.sql` and include it inside the `scripts` directory.

iv). **Add records to `like_group` table.**
This task should be implemented inside the body of the
`public void addLikeGroup(String customerId, String likeGroup) throws SQLException`
method of the `Stub` class.

As part of this task, you need to implement a **'trigger'** at the database. This trigger should be set to the `INSERT` operation of the `like_group` table. The corresponding procedure should be invoked after inserting a record to the table. This procedure should carry out the following tasks.

- It queries the `classify` table for the records with the same group name and extracts out the artworks for those records.
- Then it queries for the artist names of these artworks.
- Finally it should add records to the `like_artist` table combining the artist's name and customer id if that combination is not already recorded.

At the client side, implement a regular insert operation. At the database, it'll automatically invoke the trigger which results in an invocation of the above procedure.

Similar to stored procedures, triggers and associated functions will be persisted in the database after they are registered for the first time. For grading purposes, copy the code of the trigger and the associated function into a file named `q4.sql` and include inside the `scripts` directory.

v). **Implementing update functionality for the `style` field of `artist`**
This functionality should be implemented inside the method
`public void updateArtistStyle(String artistName, String newStyle)`
`throws SQLException`
of the `Stub` class.

Upon invoking this method, it should update the *style* field of the artist record identified by the given artist's name to the new style value.

vi). **Reading from the database.**
There is a set of read operations that should be implemented for certain tables. They should return an array of the corresponding object types for the given table.

- Get the list of artists.
(Method: `public Artist[] getArtists() throws SQLException`)
- Get the list of artworks.
(Method: `public Artwork[] getArtworks() throws SQLException`)
- Get the list of groups.
(Method: `public Group[] getGroups() throws SQLException`)
- Get the list of `classify` entries.
(Method: `public Classify[] getClassifyEntries() throws SQLException`)
- Get the list of `like_group` entries.
(Method: `public LikeGroup[] getLikeGroupEntries() throws SQLException`)
- Get the list of `like_artist` entries.
(Method: `public LikeArtist[] getLikeArtistEntries() throws SQLException`)

Please pay attention to the following points when implementing the above functionality. There will be points allocated for them.

- Use parameterized queries. Check the slides on JDBC for more information.
- Efficiently manage connections. It's not required to use any connection pooling library. It's recommended to use a single connection throughout the program and close it at the end. The test script will be invoking `close()` of the `Stub` class at the end of the program. Link your connection termination logic with this method. A database server can only handle a finite number of connections at a given time. So it's really crucial that the connections are properly closed.
- The code you write to deal with the database will throw `SQL Exceptions`. The signatures of the Stub defines them to be thrown out. So it's not required to handle them inside your code. Throwing them out of the method will be helpful to isolate issues while grading your program.
- If you have any initialization code, place them inside the `init()` of the `Stub` class. It will be invoked immediately after a `Stub` object is constructed.

Submission Instructions

- Your final deliverable should include the following.
  - Completed source code.
  - A working make file. The provided make file works out of the box. But if you have deep package structure, you may need to update it.
  - Any third party library used should be placed inside the `lib` directory.
  - Code written at the database(part iii and iv) as separate files inside the `scripts` directory.
  - If you feel that some additional information should be included for making grading easier, feel free to include them in a README file.
- Make sure that the code runs on the department Linux machines.
- Do not make changes to the tables in your database until the grading is completed from the due date. These code will be tested against your database instances in the department PostgreSQL server.
- Rename the out most directory(`assignment_5`) to `firstname_lastname`. Then create a tar archive by issuing the command; `tar -cvf firname_lastname.tar firstname_lastname`. If the student name is John Doe, then the directory name will be john_doe and the tar file will be john_doe.tar.
- Upload the final tar file to RamCT by April 8, midnight.
- Failing to adhere to conventions and guidelines may result in penalties or delays in grades.

Resources

Following resources will be useful while working on this assignment.

- PostgreSQL Data Types - `http://www.postgresql.org/docs/9.1/static/datatype.html`

- PostgreSQL Stored Procedure tutorial - `http://www.postgresqltutorial.com/introduction-to-postgresql-stored-procedures/`
- pgSQL Procedural Language - `http://www.postgresql.org/docs/9.1/static/plpgsql.html`
- PostgreSQL Triggers - `http://www.postgresql.org/docs/9.1/static/sql-createtrigger.html`
- Prepared Statements - `http://docs.oracle.com/javase/tutorial/jdbc/basics/prepared.html`