Assignment 2
Due date: 24/12/2017

Submission is in pairs. Please contact 236363cs@gmail.com for any question you might have.

1. **Introduction**

You are about to take a lead part in the development of “Tech-book”, a (fictional) Technion's social media application. Similar to other social media applications, in Tech-book users can create friends relationship, write posts, like or unlike posts, and join groups.

Your mission is to design the database, and implement the data access layer of the system. Typically, the data access layer facilitates the interaction of other components of the system with the database by providing a simplified API that carries out a predefined desired set of operations. A function in the API may receive as input arguments *business objects*. These are regular Java classes that hold a special semantic meaning in the context of the application (typically, all other system components are familiar with them).

The ZIP file that accompanies this document contains the set of business objects to be considered in the assignment, as well as the full (unimplemented) API. Your job is to write code into these functions in order for them to fulfil their purpose as described below.

**Please notice:**

1. The database design is your responsibility. You can create and modify it as you see fit. **You will be given grade for your database design**, so bad and inefficient design will suffer from points reduction.
2. Every calculation involving the data, like filtering and sorting, must be done by querying the database. **You are prohibited from performing any calculations on the data using Java.** Additionally, when writing your queries, you should **only use the material learned in class.**
3. It is recommended to go over the relevant classes Java files and understand their usage.
4. All provided business classes are implemented with default constructor and getter\setter to each field.
2. Business Objects

In this section we describe the business objects to be considered in the assignment.

Student

- Attributes:

<table>
<thead>
<tr>
<th>Description</th>
<th>Type</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Student ID</td>
<td>Integer</td>
<td></td>
</tr>
<tr>
<td>Name</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>Faculty</td>
<td>String</td>
<td>Specifies the faculty the student is associated with.</td>
</tr>
</tbody>
</table>

- Constraints:
  1. IDs are unique across all students.
  2. IDs are positive (>0) integers
  3. Name and Faculty are not optional (not null)
  4. A student automatically joins a group with the name of their faculty
  5. You may assume that the “Faculty” string received in the input is correct.

- Notes:
  1. In the class Student you will find the static function badStudent() that returns an invalid student.

Post

- Attributes:

<table>
<thead>
<tr>
<th>Description</th>
<th>Type</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Post ID</td>
<td>Integer</td>
<td></td>
</tr>
<tr>
<td>Author</td>
<td>Integer</td>
<td>The ID of the student who wrote the post.</td>
</tr>
<tr>
<td>Text</td>
<td>String</td>
<td>The body of the post.</td>
</tr>
<tr>
<td>Likes</td>
<td>Integer</td>
<td>How many students like this post.</td>
</tr>
<tr>
<td>Date</td>
<td>LocalDateTime</td>
<td></td>
</tr>
</tbody>
</table>

- Constraints:
  1. IDs are unique across all posts.
  2. IDs are positive (>0) integers.
3. Author must be a valid student id.
4. Text and Date are not optional.
5. When you are given a Post object as input, you may ignore the “likes” field, but when you return a Post object, all fields must be correct.

Notes:

6. In the class Post you will find the static function badPost() that returns an invalid post.
7. When you are saving a post to the database please use “timestamp” type in order to save it. The class is implemented with the functions setTimestamp and getTimestamp which translate Timestamp into LocalDateTime and vise-versa.

Feed

A list (Java’s ArrayList) of posts

- Constraints:
  1. Posts are sorted by date (descending order) and then by likes (descending order)

Student ID Pair

A pair of the form (<id1>,<id2>) where id1 and id2 are IDs of students.

- Notes:
  1. For the purpose of our tests, the order of the students in the pair bears no significance.

The notion of groups

Like in other social networks, a student can join or leave a group at will. In order to make things easy, we didn’t implemented any business item for groups, but when you’ll be given a group name (i.e. a not null String) you may assume that it is a valid group name, and the group exists.

3. API

3.1 Return Type

For the return value of the API functions, we have defined the following enum type:

ReturnValue (enum)
- OK
- NOT_EXISTS
3.2 Basic API

**ReturnValue addStudent(Student student)**
Adds a student to the database. The student should join to the faculty’s group
input: student to be added
output: ReturnValue with the following conditions:
* OK in case of success
* BAD_PARAMS in case of illegal parameters
* ALREADY_EXISTS if student already exists
* ERROR in case of database error

**Student getStudentProfile(Integer studentId)**
Returns the student profile by the given id
input: student id
output: The student profile in case the student exists. BadStudent otherwise

**ReturnValue deleteStudent(Student student)**
Deletes a student from the database.
Deleting a student will cause him/her to leave their group, delete their posts and likes history, and friendships
input: student to be deleted
output: ReturnValue with the following conditions:
* OK in case of success
* NOT_EXISTS if student does not exist
* ERROR in case of database error

**ReturnValue updateStudentFaculty(Student student)**
Updates a student faculty to the new given value. The student should join the group of the new faculty, and stay in the old faculty’s group.
input: updated student
output: ReturnValue with the following conditions:
* OK in case of success
* NOT_EXISTS if student does not exist
* ALREADY_EXISTS if the student is current member of the faculty
* BAD_PARAMS in case of illegal parameters (null faculty)
* ERROR in case of database error
ReturnValue addPost(Post post, String groupName)
Adds a post to the database, and adds it to the relevant group if groupName is given (i.e., it is not null)
When a student can write a post in a group only if he/she is one of its members
input: post to be posted
output: ReturnValue with the following conditions:
* OK in case of success
* BAD_PARAMS in case of illegal parameters
* NOT_EXISTS if student is not a member in the group
* ALREADY_EXISTS if post already exists
* ERROR in case of database error

Post getPost(Integer postId)
returns the post by given id
input: post id
output: Post if the post exists. BadPost otherwise

ReturnValue deletePost(Integer postId)
Deletes a post from the database
input: post to be deleted
output: ReturnValue with the following conditions:
* OK in case of success
* NOT_EXISTS if post does not exist
* ERROR in case of database error

ReturnValue updatePost(Post post)
Updates a post’s text
input: updated post
output: ReturnValue with the following conditions:
* OK in case of success
* NOT_EXISTS if post does not exist
* BAD_PARAMS in case of illegal parameters
* ERROR in case of database error

ReturnValue likePost(Integer studentId, Integer postId)
Marks a post as liked by a student
input: student id, liked post id
output: ReturnValue with the following conditions:
* OK in case of success
* NOT_EXISTS if student or post do not exist
* ALREADY_EXISTS if the student is already likes the post
* ERROR in case of database error

Return Value unlikePost(Integer studentId, Integer postId)
Removes the like marking of a post by the student
input: student id, unliked post id
output: ReturnValue with the following conditions:
  * OK in case of success
  * NOT_EXISTS if student or post do not exist, or the student did not like the post
  * ERROR in case of database error

Return Value makeAsFriends(Integer student1, Integer student2)
Establishes a friendship relationship between two different students
input: student id 1, student id 2
output: ReturnValue with the following conditions:
  * OK in case of success
  * NOT_EXISTS if one or two of the students do not exist
  * BAD_PARAMS in case of illegal parameters (sid1==sid2)
  * ALREADY_EXISTS if the students are already friends
  * ERROR in case of database error

Return Value makeAsNotFriends (Integer student1, Integer student2)
Removes a friendship connection of two students
input: student id 1, student id 2
output: ReturnValue with the following conditions:
  * OK in case of success
  * NOT_EXISTS if one or two of the students do not exist, or they are not labeled as friends
  * ERROR in case of database error

Return Value joinGroup(Integer studentId, String groupName)
Adds a student to a group
input: id of student to be added, the group name the student is added to
output: ReturnValue with the following conditions:
  * OK in case of success
  * NOT_EXISTS if the student does not exist
  * ALREADY_EXISTS if the student are already in that group
  * ERROR in case of database error
3.3 **ReturnValue leaveGroup(Integer studentId, String groupName)**

Removes a student from a group

*input:* student id 1, student id 2

*output:* ReturnValue with the following conditions:

* OK in case of success
* NOT_EXISTS if the student is not a member of the group
* ERROR in case of database error

3.4 **Advanced API**

**Feed getStudentFeed(Integer studentId)**

Gets a list of personal posts posted by a student and his/her friends. Feed should be ordered by date and likes, both in descending order.

*input:* student id

*output:* Feed the containing the relevant posts. In case of an error, return an empty feed

**Feed getGroupFeed(String groupName)**

Gets a list of posts posted in a group. Feed should be ordered by date and likes, both in descending order.

*input:* group

*output:* Feed the containing the relevant posts. In case of an error, return an empty feed

**ArrayList<Student> getPeopleYouMayKnowList(Integer studentId)**

Gets a list of students that the given student may know.

Denote the given student by s. The returned list should consist of every student x in the database that holds the following:

- s ≠ x.
- s and x are not friends.
- There exists a student y such that y ̸= s, y ̸= x, s and y are friends, and y and x are friends.
- There exists a group such that both s and x are members of.

*input:* student

*output:* an ArrayList containing the students. In case of an error, return an empty ArrayList

**ArrayList<StudentIdPair> getRemotelyConnectedPairs()**

Returns a list of student id pairs (s1, s2) such that the degrees of separation (definition follows) between s1 and s2 is at least 5.

To define the notion of degrees of separation let us consider a graph, called the friendship graph, where its nodes are the students in the database, and there is an edge between two students iff they are friends. The degrees of separation between students s1 and s2 is defined as the length of the shortest path connecting s1 and s2 in the undirected friendship graph.

*input:* none
output: an ArrayList containing the student pairs. In case of an error, return an empty ArrayList

4. Database

4.1 Basic Database functions

In addition to the above, you should also implement the following functions:

void createTables()
Creates the tables and views for the solution.

void clearTables()
Clears the tables for the solution (leaves tables in place but without any data).

void dropTables()
Drops the tables and views from DB.

4.2 Connecting to the Database using JDBC

Each of you should download, install and run a local PostgresQL server from https://www.postgresql.org. You may find this guide helpful.

To connect to that server, we have implement for you the DBConnector class that creates a Connection instance that you should work with in order to interact with the database.

For establishing successfully as connection with the database, you should provide a proper configuration file to be located under the folder \src\main\resources of the project. A default configuration file has already been provided to you under the name Config.properties. Its content is the following:

database=jdbc:postgresql://localhost:5432/cs236363
user=java
password=12345678

Make sure that port (default: 5432), database name (default: cd236363), username (default: java), and password (default: 12345678) are those you specified when setting up the database.

In order to get the Connection instance, you should invoke the static function
DBConnector.getConnection(). To submit a query to database, do the following:

1. Prepare your query by invoking connection.prepareStatement(<your query>). This function returns a PreparedStatement instance.

2. Invoke the function execute() or executeQuery() from the PreparedStatement instance.

The DBConnector class also implements the following functions which you may find helpful:

1. printTableSchemas() – prints the schemas of the tables in the database.

2. printSchema(ResultSet) - prints the schema of the given ResultSet.

3. printResults(ResultSet) - prints the underlying data of the given ResultSet.

### 4.3 SQL Exceptions

When preparing or executing a query, an SQL Exception might be thrown. It is thus needed to use the try/catch mechanism in order to handle the exception.

For your convenience, the PostgreSQLErrorCodes enum type has been provided to you. It captures the error codes that can be returned by the database due to error or inappropriate use. The codes are listed here:

```
INTEGRITY_CONSTRAINT_VIOLATION (23000),
RESTRICT_VIOLATION (23001),
NOT_NULL_VIOLATION (23502),
FOREIGN_KEY_VIOLATION(23503),
UNIQUE_VIOLATION(23505),
CHECK_VIOLATION (23514);
```

To check the returned error code, the following code should be used inside the catch block:

```
if(Integer.valueOf(e.getSQLState()) == PostgreSQLErrorCodes.CHECK_VIOLATION.getValue())
{
    //do something
}
```

### Tips


1. Create an auxiliary function that converts a record of ResultSet to an instance of the corresponding business object.

2. Use the enum type PostgreSQLErrorCodes.

3. Devise a comfortable database design for you to work with.

4. Use the constraints mechanisms taught in class in order to maintain a consistent database. Use the enum type PostgreSQLErrorCodes in case of violation of the given constraints.

Submission

Please submit the following:

1. The file solution.java where all of your code should be written in.

2. The file <id1>_<id2>.pdf in which you explain in detail your database design and the implantation of the API.

3. The file submitters.txt that consists of the following two lines:
   <id1><email1>
   <id2><email2>

Note the unit test should not be submitted.

Good Luck!