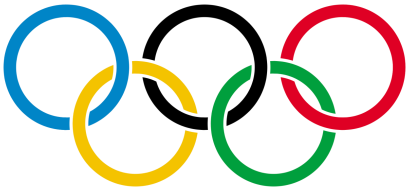
Assignment 2

SQL Programming

Due date: 5.12.19



Submission is in pairs.
Please contact Roei for any question you may have via the following email:

roei.maman20@gmail.com

1. Introduction

You are about to take a lead part in the development of the **"OlympicRecord"** database, a website that holds information about sports and the athletes participating or observing it in the upcoming 2020 Olympic Games.

In **OlympicRecord**, users with admin privileges (you) can add a sport that takes place in a city, add an athlete to a sport, confirm medals' rankings or simply make an athlete an observer of a sport.

OlympicRecord is a smart service that gives you statistics about athletes and recommends sports based on athletes' similar tastes.

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 business objects as Input arguments . These are regular Java classes that hold 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 implement these functions so that they fulfill their purpose as described below.

Please note:

1. The database design is your responsibility. You may create and modify it as you see fit. You will be graded 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 may only use the material learned in class.

3. It is recommended to go over the relevant Java files and understand their usage.

4. All provided business classes are implemented with a 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.

Athlete

Attributes:

Description	Type	Comments
Athlete ID	Integer	
Athlete name	String	
Country	String	The country the athlete represents
Active	Boolean	A Boolean attribute indicating whether the athlete is active (participant) or an observer.

Constraints:

1. IDs are unique across all athletes.
2. IDs are positive (>0) integers.
3. Name, Country and Active are not optional (not null).

Notes:

1. In the class Athlete you will find the static function badAthlete() that returns an invalid athlete.

Sport

Attributes:

Description	Type	Comments
Sport ID	Integer	
Sport Name	String	The name of the Sport
City	String	The city that the Sport takes place in
Athletes Counter	Integer	The number of Athletes participating

Constraints:

1. IDs are unique across all sports.
2. IDs are positive (>0) integers.
3. Sport name and City are not optional (not null).
4. Every new Sport's Athletes counter is initialized with 0.
5. Athletes counter can't be negative.

Notes:

1. In the class Sport you will find the static function badSport() that returns an invalid Sport celebration.

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
- ALREADY_EXISTS
- ERROR
- BAD_PARAMS

3.2 CRUD API

This part handles the CRUD - Create, Read, Update and Delete operations of the business objects in the database. Implementing this part correctly will lead to easier implementations of the more advanced APIs

ReturnValue addAthlete(Athlete athlete)

Adds an **athlete** to the database.
Input: athlete to be added.
Output: ReturnValue with the following conditions:

* OK in case of success
* BAD_PARAMS in case of illegal parameters.
* ALREADY_EXISTS if an athlete with the same ID already exists
* ERROR in case of a database error

Athlete getAthleteProfile(Integer athleteID)

Returns the athlete profile of
athleteID.

Input: athlete id .
Output: The athlete profile (an Athlete object) in case the athlete exists. BadAthlete otherwise.

ReturnValue deleteAthlete (Athlete athlete)

Deletes an athlete from the database.

Deleting an **athlete** will cause him/her to stop participating/observing in any Sport and remove all of his/her previous links (the action will not change Athletes counter of any sport).
Input: athlete to be deleted.
Output: ReturnValue with the following conditions:

* OK in case of success
* NOT_EXISTS if athlete does not exist
* ERROR in case of a database error

ReturnValue addSport (Sport sport)

Adds a **Sport** to the database.
Input: Sport to be added.
Output: ReturnValue with the following conditions:

* OK in case of success
* BAD_PARAMS in case of illegal parameters
* ALREADY_EXISTS if a sport with the same ID already exists
* ERROR in case of a database error

Note that a new Sport has 0 participants, so its Athletes counter initial value is 0.

Sport getSport (Integer SportID)

Returns the **Sport** with SportID as its id.

Input: Sport id.
Output: The Sport with SportID if exists. BadSport otherwise.

ReturnValue deleteSport (Sport sport)

Deletes a **Sport** from the database.
Deleting a Sport will delete it from everywhere as if it never existed.
Input: Sport to be deleted.
Output: ReturnValue with the following conditions:

* OK in case of success
* NOT_EXISTS if Sport does not exist
* ERROR in case of a database error

3.3 Basic API

ReturnValue athleteJoinSport(Integer sportID, Integer athleteID)

The athlete with **athleteID** is now participating/observing the sport with **sportID**.
If the athlete is not active (=observer) then he/she will pay 100$ as default, otherwise no payment is needed (0$).
Input: The ID of the sport the athlete with **athleteID** wishes to join.
Output: ReturnValue with the following conditions:

* OK in case of success
* NOT_EXISTS if sport/athlete does not exist.
* ALREADY_EXISTS if the athlete already joined the sport
* ERROR in case of a database error

- Note: an athlete can join more than one sport, make sure you change the Athletes' Counter if athlete active.
- You can assume that only active players will pay zero$.

ReturnValue athleteLeftSport(Integer sportID, Integer athleteID)

The athlete with **athleteID** is no longer participating/observing the sport with **sportID**.
Input: The ID of the sport that the athlete with **athleteID** wishes to leave.
Output: ReturnValue with the following conditions:

* OK in case of success
* NOT_EXISTS if sport/athlete does not exist or athlete doesn't participate/observe this sport.
* ERROR in case of a database error

- Note: make sure you change the Athletes' Counter if active.

ReturnValue confirmStandings(Integer sportID, Integer athleteID, Integer place)

Confirms that the athlete with id **athleteID** won a place (1^{st}, 2^{nd} or 3^{rd}) number **place** in the sport with id **sportID.** If an athlete does not participate in this sport, do nothing.
Input: **AthleteID** of the athlete who won **place** in the sport with id **sportID**.
Output: ReturnValue with the following conditions:

* OK in case of success
* NOT_EXISTS if this athlete does not participate in the sport or if the sport or athlete do not exist.
* BAD_PARAMS if **place** not between 1 and 3 (inclusive).
* ERROR in case of a database error.

ReturnValue athleteDisqualified(Integer sportID, Integer athleteID)

Disqualifying athlete with id **athleteID** and nullifying their medal from the sport with id **sportID.** The athlete will still participate/observe the sport if he/she did before.
Input: **AthleteID** of the athlete who is disqualified from the sport with id **sportID**.
Output: ReturnValue with the following conditions:

* OK in case of success
* NOT_EXISTS if the relation does not exist or sport/athlete do not exist.
* ERROR in case of a database error.

ReturnValue makeFriends(**Integer athleteID1, Integer athleteID2)**

Makes the athlete with id **athleteID**1 and the athlete with id **athleteID2** friends (symmetrical).
Input: **athleteID1** and **athleteID2** of two athletes.
Output: ReturnValue with the following conditions:

* OK in case of success
* BAD_PARAMS if both ids are the same athlete
* NOT_EXISTS if one of the athletes does not exist.
* ALREADY_EXISTS if the athletes are already friends.
* ERROR in case of database error.

ReturnValue removeFriendship(**Integer athleteID1, Integer athleteID2)**

Athlete with id **athleteID**1 and the athlete with id **athleteID2** are no longer friends (symmetrical).
Input: **athleteID1** and **athleteID2** of two athletes.
Output: ReturnValue with the following conditions:

* OK in case of success
* NOT_EXISTS if the friendship does not exist or either athlete does not exist.
* ERROR in case of database error.

ReturnValue changePayment(**Integer athleteID, Integer sportID, Integer payment)**

Athlete with id **athleteID** now pays **payment**$ for observing **sportID**.
Input: The amount the athlete with id **athleteID** will now pay to observe the sport with id **sportID**.
Output: ReturnValue with the following conditions:

* OK in case of success
* NOT_EXISTS if the athlete does not observe **sportID** or either athlete or sport does not exist.
* BAD_PARAMS if payment < 0, you can assume that 0 will not be sent.
* ERROR in case of database error.

Boolean isAthletePopular (Integer athleteID)

Returns whether all of the athlete's with id **athleteID** friends participate/observe only sports that this athlete participates/observes.

Input: The id of an athlete.
Output: the Boolean result of the above condition. On any error (such as athlete does not exist) return false.

Integer getTotalNumberOfMedalsFromCountry(String country)
Returns the total number of medals that athletes from **country** won.

Input: country
Output:

* The total number of medals athletes from a given **country** won.
* 0 in any other case (for example, if there are no athletes from this country).

Integer getIncomeFromSport(Integer sportID)
Returns the amount of money payed by observers of the sport with id **sportID**.
Input: The id of a sport.
Output:

* The amount of money payed by observers of the sport with id **sportID**.
* 0 in any other case (for example, if there is no sport with sportID).

String getBestCountry()

Returns the name of the country with the most medals won by its athletes. In case of a tie, returns the first by lexicographical order (between those with the most medals).

Input: None
Output:

* String of the country with the most medals.
* The empty string "" if all countries won 0 medals or if there are no countries in the database.
* null in any other case.

String getMostPopularCity()

Returns the city that has the highest average of athletes participating in sports.
Average is defined as the number of athletes participating in a sport at the city divided by the number of sports taking place in the city. In case of equality, return the last by lexicographical order (between those which are most popular).

Input: None
Output:

* String of the city that has the highest average of athletes participating in sports that take place in that city.
* The empty string "" if there are no cities.
* null in any other case.

3.4 Advanced API

Note: In any of the following functions, if you are required to return a list of size X but there are less than X results, return a shorter list which contains the relevant results.

ArrayList<Integer> getAthleteMedals(Integer athleteId)

Returns a list with 3 items containing the number medals won by the athlete with id **athleteID**, where the number of gold medals (first place) won at zero index, silver medals (second place) won at the first index and the number of bronze medals (third place) won at the second index.

Input: The id of an athlete
Output:

*ArrayList with the athlete's id medals
*0 in all first 3 indexes in any other case.

ArrayList<Integer> getMostRatedAthletes()

Returns a list of the top 10 athletes' ids according to the following rating calculation (in descending order):

rating = sum of the medals won such that:

Gold (first place) = 3 points

Silver (second place) = 2 points

Bronze (third place) = 1 point

For example, for an athlete who won first place at sportID and

won third place at sportID2 - rating is: $(3 + 1)$.

In case of equality order by id in ascending order.

Input: None
Output:

*ArrayList with the athletes' ids that satisfy the conditions above (if there are less than 10 athletes , return an Arraylist with the <10 athletes).
*Empty ArrayList in any other case.

ArrayList<Integer> getCloseAthletes(Integer athleteID)

Returns a list of the 10 "close athletes" to the athlete with id **athleteID**.
Close athletes are defined as athletes who participate/observe at least (>=) 50% of the sports the athlete with id **athleteID** does. Note that one cannot be a close athlete of himself.
The list should be ordered by ids in ascending order.

Input: The id of an athlete
Output:

*ArrayList with the athletes' ids that meet the conditions described above.
*Empty ArrayList in any other case.

Note: athletes can be close in an empty way (athlete in question participate/observe in none).

ArrayList<Integer> getSportsRecommendation (Integer athleteID)

Returns the 3 most participated/observed sports by close athletes to the athlete with id **athleteID** (as defined in previous functions), that this athlete does not already participate/observe.

In case of equality order by id in ascending order. In case of 0 close athletes return an empty list.

Input: The id of an athlete
Output: ArrayList with the relevant sports ids.

4. Database

6.1 Basic Database functions

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

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

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

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

6.2 Connecting to the Database using JDBC

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

To connect to that server, we have implemented 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 a 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=123456

Make sure that port (default: 5432), database name (default: cs236363), username (default: java), and password (default: 123456) 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 your 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.

6.3 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_VIOLIATION (23514);

To check the returned error code, the following code should be used inside the catch block: (here we check whether the error code *CHECK_VIOLIATION* has been returned)

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

Notice you can print more details about your errors using:

```
catch (SQLException e) {
        e.printStackTrace();
}
```

Tips

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

2. Use the enum type PostgreSQLErrorCodes. It is highly recommended to use the exceptions mechanism to validate Input, rather than use Java's "if else".

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

4. Before you start programming, think which Views you should define to avoid code duplication and make your queries readable and maintainable.

   (Think which sub-queries appear in multiple queries).

5. 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.

6. Remember - you are also graded on your database design (tables, views).

7. Please review and run example.java for additional information (ArrayList, String) and implementation methods.

8. AGAIN, USE VIEWS!

Submission

Please submit the following:

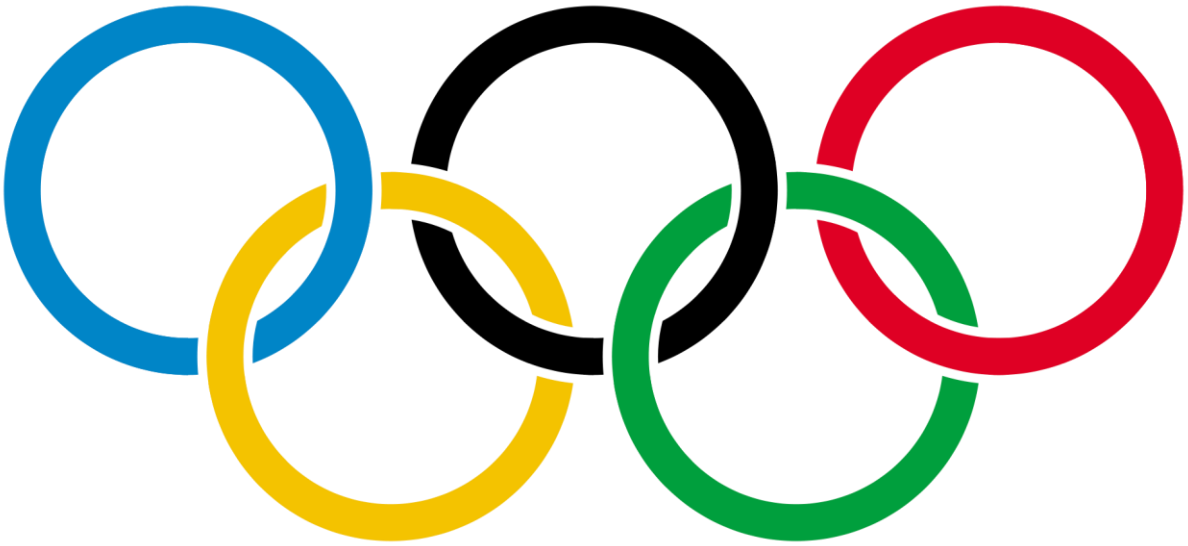
A zip file named <id1>_<id2>.zip (for example 123456789_987654321.zip) that contains the following files:

1. The file Solution.java where all 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. Is it **NOT** required to draw a formal ERD but it is indeed important to explain every design decision and it is highly recommended to include a draw of the design (again, it is **NOT** required to draw a formal ERD).

3. The file <id1>_<id2>.txt with nothing inside.

Note that you can use the unit tests framework (Junit) as explained in detail in the PDF about installing IDE, but no unit test should be submitted.



Good Luck!