CS 1555/2055 – Database Management Systems (Spring 2020)
Dept. of Computer Science, University of Pittsburgh

Project: Olympic Games
Final Exam

Release Date: Mar. 23, 2020

Phase 1 Due: 8:00 PM, Mar. 30, 2020 | Phase 2 Due: 8:00 PM, Apr. 13, 2020
Phase 3 Due: 8:00 PM, Apr. 22, 2020 | Project Demos: Apr 23-24 2020

Purpose of the project

The primary goal of this project is to implement a single Java application program that will manages **Olympic games**, and the core of such a system is a database system. The secondary goal is to learn how to develop a relatively large and real database application, also, use the powerful features of SQL/PL which include functions, procedures, and triggers.

You must implement your application program using Java, Oracle, and JDBC. The assignment focuses on the database component and not on the user interface. Hence, NO HTML or other graphical user interface is required for this project and *carry no bonus points*.

Phase 1: The Olympics database schema and example data
Due: 8:00 PM, Mar. 30, 2020

Your Olympic games database includes the information for the Olympic games such as teams, team members, awards etc. Your database will have the relations below.

You are required to define all of the structural and semantic integrity constraints and their modes of evaluation. For both structural and semantic integrity constraints, you must state your assumptions as comments in your database creation script. Semantic integrity constraints involving multiple relations should be specified using *triggers*. **Do not change table and attribute names, otherwise your program will fail the test cases.**

- **USER_ACCOUNT** (user_id, username, passkey, role_id, last_login)
  Stores the credentials for each user registered in the system.
  Datatype
  user_id: integer
  username: varchar2(20)
  passkey: varchar2(20)
  role_id: integer
  last_login: date

- **USER_ROLE** (role_id, role_name)
  Stores the roles as an enumeration. The roles are Organizer, Coach and Guest (The password for the guest user is *GUEST*).
  Datatype
  role_id: integer
  role_name: varchar2(20)

- **OLYMPICS** (olympic_id, olympic_num, host_city, opening_date, closing_date, official_website)
  Stores each Olympics Game along with its information, where olympic_num is the unique number for the Olympic games (e.g., V, VI, VII, etc).
  Datatype

  olympic_id: integer

  olympic_num: varchar2(30)

  host_city: varchar2(30)

  opening_date: date

  closing_date: date

  official_website: varchar2(50)

- **SPORT** (sport_id, sport_name, description, dob, team_size)
  Stores the information for each sport, where dob records the year it became an Olympic sport. The team size could be 1 for an atomic sport or more than 1 for team sports (e.g., 5 for basketball, 11 for soccer and 4 for 4x4 relay).
  Datatype

  sport_id: integer

  sport_name: varchar2(30)

  description: varchar2(80)

  dob: date

  team_size: integer

- **PARTICIPANT** (participant_id, fname, lname, nationality, birth_place, dob)
  Stores participants in the olympics (i.e., athletes and coaches) along with their information.
  Datatype

  participant_id: integer

  fname: varchar2(30)

  lname: varchar2(30)

  nationality: varchar2(20)

  birth_place: varchar2(40)

  dob: date

- **COUNTRY** (country_id, country, country_code)
  Stores the countries as an enumeration.
  Datatype

  country_id: integer

  country: varchar2(20)

  country_code: varchar2(3)

- **TEAM** (team_id, olympic_id, team_name, country_id, sport_id, coach_id)
  Stores the information of each team with the Olympics game during which they were contributing. Also, the sport and the coach of the team. A team can not exist without a coach.
  Datatype
  team_id: integer
  olympic_id: integer
  team_name: varchar2(50)
  country_id: integer
  sport_id: integer
  coach_id: integer

- **TEAM_MEMBER** (team_id, participant_id)
  Stores each athlete with their team.
  Datatype
  team_id: integer
  participant_id: integer

- **MEDAL** (medal_id, medal_title, points)
  Stores the medals (gold, silver, bronze) as an enumeration.
  Datatype
  medal_id: integer
  medal_title: varchar2(6)
  points: integer

- **SCOREBOARD** (olympic_id, event_id, team_id, participant_id, position, medal_id)
  Stores the standings of the athletes in particular events along with the earned medal in case any were earned. The position is the order of finishing at the event.
  Datatype
  olympic_id: integer
  event_id: integer
  team_id: integer
  participant_id: integer
  position: integer
  medal_id: integer

- **VENUE** (venue_id, olympic_id, venue_name, capacity)
  Stores the information of venues at which the games will take place with their maximum events capacity. We assume that a venue is used in only one Olympic game.
  Datatype
  venue_id: integer
  olympic_id: integer
  venue_name: varchar2(30)
  capacity: integer

- **EVENT** (event_id, sport_id, venue_id, gender, event_time)
  Stores each event with its information, where gender declares whether it is a men's or woman's event.
  Datatype

  event_id: integer

  sport_id: integer

  venue_id: integer

  gender: char

  event_time: date

- **EVENT_PARTICIPATION** (event_id, team_id, status)
  Stores the teams competing to an event. The status is eligible (e) or not eligible (n)
  Datatype

  event_id: integer

  team_id: integer

  status: char

Initialization script and Sample Data: Every time that the program starts you need to initialize all the above tables with sample data from the last four Olympic games (Beijing, Paris, London, Rio) for 6 events (3 atomic and 3 team events) and 6 countries (which have won medals). Note that this script will help you validate the integrity constraints and triggers.

Phase Deliverables: Phase 1 should produce the following files:

- **init.sql** the script to initialize the tables.
- **schema.sql** the script to create the tables.
- **trigger.sql** the script to create other database objects, e.g., trigger, functions, procedures, etc.

Note that after the first phase submission, you should continue working on your project without waiting for our feedback. Furthermore, you should feel free to correct and enhance your SQL part with new views, functions, procedures etc.

Phase 2: A JDBC application to manage the Olympic games
Due: 8:00 PM, Apr. 13, 2020

You are expected to do your project in Java interfacing Oracle server using JDBC. You can develop your project on your local environment, but you have to make sure that it runs on **class3.cs.pitt.edu** in order to be graded and demonstrated if selected.

For all tasks, you are expected to check for any errors reported by the DBMS (Oracle), and provide appropriate success or failure feedback to user. Further, be sure that your application carefully checks the input data from the user and avoids SQL injection.

Attention must be paid in defining transactions appropriately. Specifically, you need to design the **SQL transactions** appropriately and when necessary, use the concurrency control mechanism supported by Oracle to make sure that inconsistent states will not occur. You should assume that multiple requests for changes of Olympics can be made on behalf of multiple different users concurrently. For example, it could happen when Coach A is trying to add a new Athlete B while another Coach C is trying to add a new Athlete D at the same time (i.e., concurrently).

One objective of this project is to familiarize yourself with all the features of SQL/PL, such as functions, procedures, and triggers. Recall that triggers and stored procedures can be used to make your code more efficient besides enforcing integrity constraints.

You should have at least 3 types of triggers:

1. ASSIGN_MEDAL: This trigger is responsible to assign the appropriate medal based on the position when new records are inserted or updated in the SCOREBOARD.

2. ATHLETE_DISMISSAL: This trigger is responsible for deleting all the data of an athlete who was dismissed because of a violation. If the athlete is a member of a team sport, then the team is also dismissed by setting the status not eligible (n) in participating in any event. If the athlete participates in an atomic sport, then the corresponding teams are removed from the events.

3. ENFORCE_CAPACITY: This trigger should check the maximum possible venue capacity before the event is associated with it. In case the capacity is exceeded, an exception should be thrown.

Your application should implement the following functions for managing the Olympic games.

Organizer

1. **createUser**
   Given a username, passkey, role_id, add a new user to the system. The "last_login" should be set with the creation date and time. Only organizers can add any kind of users to the system. User IDs should be auto-generated (look at the note below).

2. **dropUser**
   This function should remove the user from the system.

3. **createEvent**
   Given a sport ID, a venue ID, date/time and whether it is a men's or women's event, add a new event to the system. Only organizers can create events, and the system should use **the trigger ENFORCE_CAPACITY** to assure that the capacity of the associated venue is not exceeded and there are not conflicting events at the same time.

4. **addEventOutcome**
   Given an Olympic game, team, event, participant and position, add the outcome of the result to the scoreboard. The medal should be automatically assigned to the participant based on the position using **the trigger ASSIGN_MEDAL**.

Coach

5. **createTeam**
   Given an Olympic game (City, Year), sport, country, and the name of the team, add a new team to system. Team IDs should be auto-generated, and only coaches can create teams and their name is added as the team coach (team member).

6. **registerTeam**
   Given a team_id and an event_id, the team is register to an existing event.

7. **addParticipant**
   Given the first name, last name, nationality, birth_place, do, create participant.

8. **addTeamMember**
   Given a team ID and a participant, add the member to the team. Only the coach of the team.

9. **dropTeamMember**
   This function should remove the athlete from the system (i.e., deleting all of their information from the system). When a user is removed, the system should **use the trigger ATHLETE_DISMISAL** to remove the user from the teams they are a member of. Attention should be paid to handling integrity constraints.

All

10. **login**
    Given username and password, login in the system when an appropriate match is found with the appropriate role.

11. **displaySport**
    Given a sport name, it displays the Olympic year it was added, events of that sport, gender, the medals winners and their countries. (athletes who got medals should be displayed first according to medals i.e., gold, silver and bronze and sorted on the Olympic year).

12. **displayEvent**
    Given an Olympic game (City, Year) and an event_id, display the Olympic game, event name, participant and the position along with the earned medal.

13. **countryRanking**
    Given an olympic_id, display all the participating countries (country abbreviation), the first year the country participated in the Olympics (first registered in the DB) along with the number of gold, silver and bronze medals and their ranking sorted in descending order. The rank is computed based on the points associated with each medal.

14. **topkAthletes**
    Given an olympic_id and a number k, display the top-k athletes based on their rank along with the number of gold, silver and bronze medals in a descending order of their rank. The rank is computed based on the points associated with each medal.

15. **connectedAthletes** Given an athlete, a olympic_id and a number n, find all the athletes who are connected to this athlete based on the participation in the last $n + 1$ games. That is, it displays pairs of athletes that are n hops apart. For example if n is 1 and we have three athletes A, B, and C ($A \neq B \neq C$), then A and C are connected (1 hop apart), if A competes with B in the current Olympic games (olympic_id) and C competed with B in the immediate previous Olympic (olympic_id).

16. **logout**
    The function should return the user to the top level of the UI after marking the time of the user's logout in the user's "last_login" field of the *USER_ACCOUNT* relation.

17. **exit**
    This option should cleanly shut down and exit the program.

Note: The ID field in tables USER_ACCOUNT, OLYMPICS, SPORT, TEAM, VENUE, and EVENT should be generated by SEQUENCE, and the documentation can be found at:
`https://docs.oracle.com/en/database/oracle/oracle-database/index.html`

For this project, you need to design an easy-to-use interface. A simple text menu with different options is sufficient enough for this project. You may design nice graphic interfaces, but it is not required and it carries no bonus points. A good design may be a two level menu such that the higher level menu provides options for createUser, login, and exit. If a user successfully logged in, then the lower level menu is shown to provide options to do the remaining functions.

Furthermore, since this is a database course, if a function or a task can be implemented using the database approach, you should use that approach. You might lose points if you use the Java approach (i.e., implement the DB operations and logic in Java).

Phase Deliverables: Phase 2 should produce, in addition to Phase 1, the following files:

- **Olympic.java** the file containing your main class and all the functions.
- **Readme.txt** the file with instructions of how to compile and run your system.

Phase 3: Bringing it all together
Due: 8:00 PM, Apr. 22 2020

The primary task for this phase is to create a Java test driver program to demonstrate the correctness of your Olympics back-end. The driver program needs to call all of the above functions and display the content of the affected rows of the affected tables after each call. If a function does not modify the database, then showing the output/display of the function is enough. It may prove quite handy to write this driver as you develop the functions as a way to test them. You may use a program to dynamically generate a large number of function calls within your driver.

Note that the main program in Phase 2 (i.e., Olympics.java) should contain all functions and UI. The main program in Phase 3 (i.e., Driver.java) is only used to test the functions without user manual inputs. If your original functions in "Olympics.java" involve user manual inputs, then you may create helper functions in "Olympics.java" that take in valid input arguments and interact with the DBMS directly without involving user manual inputs, so that your original functions in "Olympics.java" can call those helper functions and the "Driver.java" can also call those helper functions for testing purposes.

Now this may not seem like a lot for a third phase (especially since it is stated that you may want to do this work alongside Phase 2), the reasoning for this is to allow you to focus on incorporating feedback from the TA regarding Phases 1 and 2 into your project as part of Phase 3. This will be the primary focus of Phase 3.

Phase Deliverables: Phase 3 should produce, in addition to Phase 2, the following files:

- **Driver.java** the driver file to show correctness of your functions.
- **Readme.txt** the file with instructions of how to compile and run your driver program.

Project Submission

The project will be collected by the TA via the private GitHub repository that is shared with only the instructor (GitHub username: Costantinos), TA (GitHub username: ralseghayer) only. To turn in your code, you must do three things by each deadline:

1. Make a commit to your project repository that represents what should be graded as your submission for that phase. The message for this commit should be "Phase X submission" where X is 1,2 or 3.

2. Push that commit to the GitHub repository that you have shared with the instructor and TA.

3. Send an email to cs1555-staff@cs.pitt.edu with the title "[CS1555] Project Phase X submission" that includes your pitt_id, a link to your GitHub repository, and Git commit ID for the commit that should be graded. Commit ID can be found on GitHub or as part of output of "gitlog" command.

You can submit multiple times for each phase. The feedback/grading for each phase will be based on the last commit before the corresponding deadline. *NO late submission is allowed.*

Grading

The project will be graded on correctness (including transaction usage), robustness (error-checking, i.e., it produces user-friendly and meaningful error messages) and readability. You will not be graded on efficient code with respect to speed although bad programming will certainly lead to incorrect and slow programs. For Phases 1 and 2, feedback will be given on how to improve your program based on the project requirement. Your project's grade is based on the final program (i.e., Phase 3 submission) and the project demo if selected. Programs that fail to compile or run or connect to the database server earn zero and *no partial points.*

Final Submission for grading should be the following files:

- **Olympic.java** the file containing your main class and all the functions.
- **Readme.txt** the file with instructions of how to compile and run your system.
- **Driver.java** the driver file to show correctness of your functions.
- **Readme.txt** the file with instructions of how to compile and run your driver program.

Academic Honesty

The work in this assignment is to be done *independently* by each student. Discussions with other students on the project should be limited to understanding the statement of the problem. Cheating in any way, including giving your work to someone else will result in an F for the course and a report to the appropriate University authority.

Enjoy your class project!