CS 1555 / 2055 – Database Management Systems (Fall 2024)
Computer Science, University of Pittsburgh

# Project Phase #4 – JDBC

Release: Friday, Apr 12, 2024 Due: 12:00PM (Noon), Monday, Apr 22, 2024

---

## Goal

The primary goal of this project is to realize a database system that supports the operations of OlympicDB, a database system for managing the teams, team members, awards, etc. of the Olympic Games. This project has you follow the typical development cycle of a database application (i.e., conceptual design, logical design, and implementation of an application).

The secondary goal is to learn how to work as a member of a team which designs and develops a relatively large, real database application.

## Description

The OlympicDB application will be developed in four phases: (1) The OlympicDB database design; (2) The OlympicDB data manipulation; (3) The OlympicDB analytical queries; and (4) The OlympicDB Java App.

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

It is expected that all members of a team will be involved in all aspects of the project development and contribute equally. Division of labor to data engineering (db component) and software engineering (java component) is not acceptable since each member of the team will be evaluated on all components.

## OlympicDB JDBC Application

The objective of this phase is to familiarize yourself with JDBC by implementing the OlympicDB application in Java using JDBC to connect to the backend database server (DBMS). The application should implement a command-line interface (non-graphical) that will allow users to connect to the database and perform predefined functions. A good design may be a simple menu that provides options for the operations from Phases 2 & 3, as listed below, and exiting the application. You are expected to use Java interfacing to connect to the local PostgreSQL server using JDBC.

For all operations, you are expected to check for and properly handle any errors reported by the DBMS (PostgreSQL), and provide appropriate success or failure feedback to the user. For example, if a requested operation *doOp* is provided with a primary key that is not in OlympicDB, a message "No <entity> Found to *doOp*" should be displayed to the

user. Further be sure that your application carefully checks the input data from the user and avoids SQL injection.

Attention must be paid when handling concurrent transactions from clients, that is, assume that multiple requests for changes of OlympicDB can be made on behalf of multiple different users concurrently. For example, it could happen when user A is trying to add a new participant B, while another user C is trying to add a new participant D, at the same time (i.e., concurrently). Specifically, when establishing a connection to OlympicDB, you need to utilize the concurrency control mechanisms supported by PostgreSQL (e.g., isolation level, locking models) to prevent nonrepeatable reads and to make sure that inconsistent states will not occur.

Your application should invoke the following operations for managing OlympicDB (you wrote these in Phase 2 & 3).

1. **connect**
   This operation should connect to your PostgreSQL database by prompting for a username and password to establish the connection. Recall that the default user is *postgres*.

2. **createAccount**
   This operation prompts for a username, passkey, role, date and time (as a single timestamp), then adds the new account to the system by inserting a new entry into the *account* relation. The *last_login* should be set with the creation date and time and account_ids should be auto-generated.

3. **removeAccount**
   This operation prompts for an account_id, then removes the account by deleting the account from the *account* relation.

4. **addParticipant**
   This operation prompts for an account_id, first name, middle initial / name, last name, country of birth, date of birth (dob), and gender, then adds the new participant to the system by inserting a new entry into the *participant* relation. participant_ids should be auto-generated.

5. **removeParticipant**
   This task should first ask the user if they would like to remove *all* or *selected* participants from OlympicDB.

   If the user's response is *all*, then the user is prompted for a confirmation. If the confirmation is *Yes*, then the operation proceeds with deleting all the participants and any entries in the *team_members* relation. If the confirmation is *No*, the operation produces a message "No Participantes were Removed" and the user should be returned to the main menu of the application.

   If the user's response is *selected*, it displays one participant at a time and asks the user to enter the displayed *participant_id* as a confirmation that they would like to delete the sensor. If the user enters the displayed *participant_id*, the currently selected participant is removed along with all of its corresponding entries in the *team_members* relation.

If the user enters 0, the currently selected participant should not be removed and the next participant should be displayed. However, if the current participant is the last participant, the user should be returned to the main menu. Lastly, if -1 is entered, the user is returned to the main menu of the application without removing the participant. If an invalid *participant_id* (other than 0 or -1) is entered, the operation should ask the user to enter one of the displayed *particpant_id*s, 0, or -1.

6. **addTeamMember**
   This operation prompts for a participant_id and team_id, then adds the participant to that team.

7. **removeTeamMember**
   This operation should first prompt for a team_id and display a formatted list (ordered in ascending order by the participant_id) of all the participants who are members of the team. Then the operation allows the user to either remove a participant from the team by entering the appropriate participant_id or to exit browsing and return to the main menu by entering -1 as a participant_id. If the user selects a valid participant_id (i.e., one that exists within OlympicDB, is a member of the specified team, and is not -1), the user should be prompted for confirmation ( *Yes* or *No*) to remove the participant from the team.

   In the event that there are no participants on the specified team, a message "No participants are currently members of the Team" should be displayed to the user.

8. **registerTeam**
   This operation prompts for an olympiad_id, sport_id, coach_id, country_code, and participating genders of the team, then adds the new team to the system by inserting a new entry into the *team* relation. team_ids should be auto-generated.

9. **addEvent**
   This operation prompts for a venue_id, olympiad_id, sport_id, participating genders, and date, then adds the new event to the system by inserting a new entry into the *event* relation.

10. **addTeamToEvent**
    This operation prompts for an event_id and team_id, then adds a (placeholder) entry to the *placement* relation if the team is eligible (i.e., not disqualified). Since the event has not yet occurred, the placeholder should set both the position and medal to NULL as the default value, which will be updated by the addEventOutcome operation.

11. **addEventOutcome**

    This operation repeatedly prompts the user for a tuple (event_id, team_id, position) for adding a team's event outcome. That is, the user should be repeatedly prompted for each of the attributes within the tuple until the user provides an event_id of -1. When the user provides an event_id of -1, the operation should return the user to the

main menu of the application. If the user provides a valid tuple (event_id, team_id, position) and the event_id is not -1, the operation updates team's position and medal in the *placement table* for that event. Note that the medal should be updated if the team's position matches a medal_id in the *medal* relation.

In the event that there are no events in OlympicDB, a message "No Outcomes were added since there are no Events" should be displayed to the user.

12. **disqualifyTeam**
    This operation prompts for a team_id, then marks the team as disqualified by setting the team's *eligible* status to *false*. After setting the team's status to disqualified, the operation should also set the team's position to -1 for all events where the team had a placement.

13. **listVenuesInOlympiad**
    This operation prompts for an olympiad_id, then displays all the details of the venues for that olympiad in a nicely formatted way.

14. **listEventsOfOlympiad**
    This operation prompts for an olympiad_id, then displays all the details of the events for that olympiad in a nicely formatted way.

15. **listTeamsInEvent**
    This operation prompts for an event_id, then displays the teams in that event in a nicely formatted way.

16. **showPlacementsInEvent**
    This operation prompts for an event_id, then displays the team placements in that event in a nicely formatted way.

17. **listParticipantsOnTeam**
    This operation prompts for a team_id, then displays all the information about the members of the team and its coach in a nicely formatted way.

18. **listCountryPlacementsInOlympiad**
    This operation prompts for a country_code and olympiad_id, then displays all the placements of that country in all events of that Olympiad in a nicely formatted way.

19. **listAthletePlacement**
    This operation prompts for a participant_id, then displays all the placements of that participant throughout all Olympiads in a nicely formatted way.

20. **countryRankings**
    This operation should display a ranked list of all the countries (country_code, country_name, participationRank, olympiadCount) in OlympicDB based on the number of Olympiads that the country has participated in. The country rankings should be displayed in a nicely formatted way.

In the event that there are no countries in the system, a message "No Countries to Rank" should be displayed to the user.

21. **mostSuccessfulParticipantsInOlympiad**
    This operation prompts for an *olympiad_num* and integer $k$, then displays the top-$k$ athletes based on the points associated with each medal that the athlete earned. The top $k$ athletes should be displayed in a nicely formatted way.

22. **topSports**
    This operation prompts for an integer $x$ and $k$, then displays the top-$k$ sports with respect to the number of teams who participated in an event for the sport in the past $x$ years. The top-$k$ sports should be displayed in a nicely formatted way.

23. **connectedCoaches**
    This operation prompts for two coach_ids, $c_1$ and $c_2$, finds a path, if one exists, between $c_1$ and $c_2$ with at most 2 hops between them. This operation should display a string representing the path that connects $c_1$ to $c_2$ with all intermediate hops. For instance, '$c_1 \rightarrow c_3 \rightarrow c_2$' where $c_3$ is a coach that connects $c_1$ to $c_2$. In the event that no path is found between $c_1$ and $c_2$ in two hops, a message "No path was found" should be displayed to the user.

24. **exit**
    This operation should cleanly shut down and exit the program.

## Project Submission:

Phase 4 should contain the Java code for your OlympicDB application and the SQL components for the SQL DDL, triggers, operations, and analytical queries. Specifically,

- **schema.sql** – the script to create the database schema.
- **trigger.sql** – the script containing definitions of the triggers
- **operations.sql** – the script containing the implementation for the data manipulation operations
- **analytical.sql** – the script containing the implementation for the analytical queries
- **OlympicDB.java** – the file containing your main class for running the OlympicDB application
- **README** – A README file that elaborates (fully) on how to use your *OlympicDB* client application and any errors or missing operations within your application

The project will be collected by submitting your GitHub repository to Gradescope. Given that Phase 4 builds on top of the OlympicsDB system from Phases 2 & 3, you should utilize the same GitHub repository from the previous two phases.

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 group's submission for that phase.
2. Push that commit to the GitHub repository that you have shared with the TAs
3. Submit your GitHub repository (including all necessary SQL files) to Gradescope under the Project Phase 4 assignment link

   To submit to Gradescope, you will need to:

   - Select the appropriate assignment submission link (as you've previously done with homework assignments).
   - On the "Submit Programming Assignment" window that appears, choose "GitHub." If this is your first time submitting to Gradescope via GitHub, you will be prompted to link your GitHub account and authenticate.
   - Select your team's GitHub repository in the dropdown (searching will filter the repositories listed).
   - Select the branch of your GitHub repository with the code your team wishes to submit (typically just the main branch). Then click the green upload button in the bottom left of the window.
   - After uploading your team's submission, you will be taken to the submission results page. The next step is to add each team member to the assignment to allow for a single linked submission. Note: **There should only be one submission per team, i.e., every team member does not need to submit**.
   - To link team members, click "Add Group Member" in the top right corner of this page.
   - On the new window, add all of your corresponding team members to the group, then hit the green "Save" button in the bottom left of this window.
   - If done correctly, every team member should receive a confirmation email from Gradescope.

## Academic Honesty

The work in this assignment is to be done *independently* by each team. Discussions with other students or teams 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.