Project 3: ChessDB Application

CMPE 321, Introduction to Database Systems, Spring 2025

Due: 20 May 2025, 23:59 o'clock

In Project 1, you designed *ChessDB*, a relational database system to manage chess tournaments in compliance with FIDE standards. The design captured complex relationships between players, coaches, arbiters, matches, and tournaments while enforcing various integrity constraints.

In this project, you implement **ChessDB** as a working database application. The system will support multiple user roles, including players, coaches, and arbiters, and enable real-world operations such as match scheduling, player selection, and rule-compliant rating submissions. Additional constraints that could not be enforced through basic DDL will be handled using triggers, procedures, and other mechanisms, ensuring both functional and data integrity in the application.

1 Project Description

This project involves the implementation of the database system *ChessDB*, whose conceptual and logical design was developed in Project 1. ChessDB models a tournament ecosystem governed by FIDE, containing users (database managers, players, coaches, arbiters), teams, sponsors, matches, tournaments, halls, and tables.

You are expected to develop a database-backed application that supports multiple user roles and performs real-world operations while enforcing domain-specific constraints. The application must be connected to a MySQL database using SQL queries written manually. You must not use any Object-Relational Mapping (ORM) tool such as SQLAlchemy, Django ORM, or Hibernate. The goal of this project is to write and understand SQL yourself, including table creation, data manipulation, and trigger logic.

Supported User Roles

- **Database Managers:** These are the administrators of the ChessDB application you are going to create. They only have a username and password (which is provided to you in the initial dataset). Their abilities are limited and predefined: they can update hall names and create users such as players, coaches, and arbiters. No new database managers can be added. You are not expected to create a signup page for these users — authentication is based on fixed credentials.

  - *Attributes:* `username, password`

- **Players:** Players are registered users with ELO ratings, titles, and personal information such as nationality and date of birth. They can participate in matches representing their teams. Each player can be registered to multiple teams but must follow tournament-specific rules such as time slot restrictions and valid team participation. Players will be able to access their match history and statistics through the interface.

  - *Attributes:* `username, password, name, surname, nationality, date_of_birth, fide_id, elo_rating, title_id`

- **Coaches:** Coaches manage teams and are responsible for match-related operations such as scheduling and selecting players. A coach can only be assigned to one team at a time and cannot have overlapping contracts. Coaches can create new matches for their team and assign players who are officially part of it. They also select an available arbiter from the certified list during match creation.

  - *Attributes:* `username, password, name, surname, nationality`
    Related tables: `CoachCertifications (certification)`

- **Arbiters:** Arbiters are the match officials. Each match must have an assigned arbiter who is responsible for rating the match after it has taken place. Arbiters can only rate matches assigned to them and only once per match. They cannot change their rating afterward, and must avoid time conflicts when being assigned.

  - *Attributes:* `username, password, name, surname, nationality, experience_level`
    Related table: `ArbiterCertifications (certification)`

Key Operations

Your application should allow users to perform the following operations depending on their roles. The interface does not need to be fancy — the focus is on correct logic, data consistency, and enforcing the rules using appropriate SQL constructs.

1. **Login and role-based access:** All users should log in using their credentials. Based on their role (player, coach, arbiter, or database manager), they should see different functionality.

2. **Creating a match (Coach):** Coaches can schedule new matches by selecting the date, time slot, hall, table, their team, the opposing team, and an available certified arbiter.

   Note: Matches must not overlap in time or place. Each match lasts 2 consecutive time slots. Constraints such as arbiter/player availability and hall/table usage should be considered. You are free to enforce these using triggers, procedures, or query validation.

3. **Assigning players to a match (Coach):** Coaches can select one player from their own team to represent them in a scheduled match. Similarly, the opposing

team must assign their own player.

Note: Only players from the coach's current team can be selected. A match must consist of one player from each team.

4. **Deleting a match (Coach):** A coach can delete any match they created. All related data — such as player assignments, match rating, and arbiter assignment — should be removed as well.
5. **Renaming a hall (Database Manager):** Database managers can update the name of any hall. The change should be reflected consistently across all related records.
6. **Viewing available halls (Coach):** Coaches can view a list of all halls, including their names, countries, and total table capacity.
7. **Submitting a rating (Arbiter):** Arbiters can rate a match they are assigned to, but only after the scheduled date has passed and only if the match has not yet been rated.

   Note: Each match can only be rated once. Ratings cannot be changed once submitted. These restrictions can be enforced via triggers or application-level validation.
8. **Viewing rating statistics (Arbiter):** Arbiters can view how many matches they have rated and the average rating they have given across all matches.
9. **Viewing co-player statistics (Player):** Players can view a list of all players they have played against at least once. Additionally, they can view the ELO rating of the player they've played with the most.

   Note: If multiple players are tied in most games played together, show the average ELO rating of those players.

Important Notice

Your database will be initialized with a dataset that may include intentionally invalid records, such as matches involving players outside of contract, or players who are no longer on a team. These inconsistencies are provided on purpose and should remain in the database as they are.

Your goal is not to clean or delete these records, but rather to ensure that no new invalid records can be inserted into the system.

You are encouraged to use `CHECK` constraints, triggers, or stored procedures wherever schema-level enforcement is insufficient. You should ensure that invalid data cannot be added, even if some preloaded records are intentionally incorrect.

For example:

- A new match should not be scheduled if there is a hall, table, or time conflict.
- A coach should not be allowed to manage multiple teams at overlapping times.

- An arbiter should not be assigned to two matches that occur at the same time.
- A match cannot be rated by someone other than the assigned arbiter.

These constraints will be tested during the demo by attempting to insert invalid data. If your system does not prevent these actions, you will lose points accordingly.

2 Schema Refinement and Normalization

You are expected to analyze and refine the database schema you designed in Project 1. This step should ensure that your design adheres to normalization principles and that the schema effectively captures functional dependencies (FDs).

- First, list all non-trivial functional dependencies in your schema.
- Then, for each relation, determine whether it satisfies Boyce-Codd Normal Form (BCNF).
- If a relation is not in BCNF, check whether it is in Third Normal Form (3NF).
- If a relation does not satisfy BCNF, either:
  - Decompose it into BCNF-compliant relations, or
  - Clearly justify why decomposition was not performed.
- If you decompose a relation, indicate whether your decomposition is:
  - Lossless-join
  - Dependency preserving

Even if your original schema is already in BCNF, you must still explain how the requirements of BCNF are satisfied in terms of the functional dependencies for each relation.

In addition to normalization, you should revisit your schema to capture constraints that were not enforced in Project 1. Use SQL features such as `CHECK`, triggers, and stored procedures to enforce those rules.

Please explain which additional constraints you were able to enforce and how.

3 Requirements

Your user interface must allow the following operations:

1. All users should be able to log in with their credentials (username and password).
2. Database managers should be able to:
   - Add new users (players, coaches, arbiters)
   - Rename any existing hall
3. Coaches should be able to:
   - Create new matches by providing date, time slot, hall, table, opponent team, and selecting an available arbiter.
   - Assign one player from their team to represent the team in the match.
   - Delete a match they created. All related records (such as player assignment, arbiter, rating) must be removed.
   - View a list of all halls, including names and countries.
4. Arbiters should be able to:
   - View all matches assigned to them
   - Submit a rating for a match if it has not been rated yet and if the date of the match has passed
   - View their average match rating and the total number of matches they have rated
5. Players should be able to:
   - View the names of all players they have played against
   - View the ELO rating of the player they played with the most
   - If there's a tie, view the average ELO rating of those players
6. Database managers should be able to:
   - Add new users (players, coaches, arbiters)
   - Rename any existing hall

   Password Policy: When creating a new user, the system must validate the password according to the following rules:
   - Minimum length: 8 characters
   - Must include at least one uppercase letter `[A-Z]`
   - Must include at least one lowercase letter `[a-z]`
   - Must include at least one digit `[0-9]`

- Must include at least one special character (e.g., `@, #, $, %, &`, etc.)

If the password does not satisfy these requirements, the insertion must be rejected and an appropriate error message should be displayed.

Password Encryption: All passwords must be stored in an encrypted form in the database. Plain-text passwords are strictly forbidden. You must hash passwords using a secure algorithm (e.g., `SHA-256`, `bcrypt`, or similar), and authenticate users by comparing the hash of the input password against the stored hash.

4 Notes

- You are not expected to design a fancy interface. A simple web interface with minimal or no CSS/JavaScript is acceptable.
- You may use a backend framework (such as Django, Flask, Express, etc.) but all SQL queries must be written manually by you.
- You must boot your own database server and set up your tables manually.
- **You are not allowed to use any ORM tools** (e.g., SQLAlchemy, Hibernate, Django ORM).
- Do not use your database connector to dynamically build queries. Queries should be written as SQL strings and passed to the connector for execution.
- The application does not need to be deployed. Running locally is sufficient.
- Only MySQL is allowed.
- All date values must follow the format: `DD-MM-YYYY`. Make sure your SQL queries and input forms conform to this format.
- There will be a demo session where you will be asked to show specific operations working correctly from your interface. Demo details will be announced later.
- Your grade will be based 70% on the demo and 30% on Part 2 of this documentation.

5 Submission

This project must be completed and submitted in teams of two students. Each team must submit a single ZIP file that includes all deliverables.

Submission Format:

- Each team must submit a ZIP file named as: `GroupXX.zip`, where `XX` is your assigned group number.
- The following items must be placed **directly inside the ZIP file** (not in subfolders unless specified):
  - `code/` directory: All your implementation files including SQL scripts, trigger definitions, procedures, and your interface/backend code.
  - `part3.pdf`: Your written report for the Schema Refinement and Normalization section.
  - `Student1ID_Contribution.pdf`: Individual Contribution Report for Student 1
  - `Student2ID_Contribution.pdf`: Individual Contribution Report for Student 2

Final Folder Structure (Inside the ZIP):

```
GroupXX.zip
    code/
    part3.pdf
    2023456_Contribution.pdf
    2023789_Contribution.pdf
```

Individual Contribution Report Guidelines

Each student must write their own **Individual Contribution Report** in PDF format and include it in the ZIP file. The report should be named as:

`StudentID_Contribution.pdf`
(e.g., `2023789_Contribution.pdf`)

What to include:

- **Personal Info:** Name, Student ID, Group Number
- **Contributions:** Tasks you worked on (e.g., trigger logic, match constraints, UI forms, login system)
- **Teamwork:** How the team collaborated and divided responsibilities
- **Self-Reflection:** Your learning experience and skills improved

Formatting:

- **Length:** 1 page (max 2 pages)
- **Font & Size:** Times New Roman, 12pt, 1.5 spacing
- **Format:** PDF

Note: If one team member fails to submit their contribution report, they will not be credited for the project submission.

6 Late Submission Policy

We will accept late submissions, however;

- One day late (even one minute will be considered a day late) would mean -20 points penalty.
- Two days late (even one minute and a day will be considered two days late) would mean -50 points penalty.
- Moodle will close after two days. No other submission method will be accepted.

7 Academic Honesty

Please read carefully the academic honesty part of the syllabus as we give utmost importance to academic honesty.