# Specification of the data base

Neila Krika

February 2025

## 1 Introduction

### 1.1 Purpose

The data base of the game is designed to manage the core components of the "6 takes" card game, including player data and statistics, game sessions, lobby management and assets storage. It ensures data integrity, supports multiplayer gameplay, facilitates real-time interactions, and provides scalability.
The database serves as a critical backbone for the game's backend infrastructure.

### 1.2 Scope

The database is intended to handle all aspects of the game including:

- **Player managment:** Secure registration, authentification and profile maintenance.
- **Game Session Tracking:** Real-time recording of game progress, results, and player statistics.
- **Lobby Managment:** Dynamic creation, administration, and tracking of game lobbies.
- **Session Managmenet :** Handle active sessions and authentication tokens.
- **Assets Management :** Organized storage and retrieval of game assets such as cards, icons, and sounds.
- **Security and Performance:** Implementation of robust security measures, efficient indexing and support for scalability to handle increasing user loads.

### 1.3 Audience

This document is intended for **database administrators**, **developers** for application integration and backend development, **System Administrators** for deployment, monitoring, and security of the database system and finally the **Stakeholders** for providing insight into how game data is managed and structured.

## 1.4 Requirements Analysis

### 1.4.1 Functional Requirements

- **Data Management:** The system must support full CRUD (Create, Read, Update, Delete) operations on primary entities.
- **User Authentication and Authorization:** The database should integrate with an authentication system to ensure that only authorized users can access or modify data.
- **Data Relationships:** The system must effectively manage relationships between entities ensuring referential integrity across the database.
- **Reporting and Querying:** The database should facilitate complex queries and generate reports on sales, inventory levels, and customer interactions. This includes support for both ad-hoc querying and predefined reports.

### 1.4.2 Non-Functional Requirements

The database must meet several key non-functional requirements to support the dynamic needs of the game:

- **Performance :**
  - Fast query responses for real-time game state updates.
  - Optimized indexing to reduce latency during high-trafic periods.
- **Scalability :**
  - Capability to scale horizontally or vertically as user load increases.
  - Efficient resource management to support a growing number of concurrent players.
- **Security :**
  - Encrypted storage for sensitive data such as passwords and session tokens.
  - Regular security audits and vulnerability assessments.
  - Robust backup procedures must be in place.
- **Reliability and Availability:**
  - High availability with minimal downtime, achieved through replication and backup strategies.
- **Maintainability:**
  - Comprehensive documentation and audit logs for ongoing maintenance.

# 2 Table Specification

## 2.1 PLayers

- **Description:** Stores player profiles and login credentials.
- **Pimary Key:** id (INT, AUTO_INCREMENT)
- **Fields:**
  - **username** : VARCHAR(255), NOT NULL, UNIQUE – used for registration and login.
  - **email**: VARCHAR(255), NOT NULL, UNIQUE – used for registration and password recovery.
  - **password**: VARCHAR(255), NOT NULL – stores the hashed password.
  - **created_at:** TIMESTAMP, DEFAULT CURRENT-TIMESTAMP – records the account creation date.
  - **first_login:** BOOLEAN – flag to indicate the player's first login (e.g., to display game rules).
  - **icon** A reference to the player's icon from the icons table.
  - **Player statistics** INTEGER - The player's score total number of played and won game for statistics and leaderboard
- **Indexes :**
  - Index on username for efficient login lookups.
  - Index on email for recovery processes.

## 2.2 Lobbies

- **Description :** Manages game lobbies.
- **Primary Key :** id (INT, AUTO-INCREMENT)
- **Fields:**
  - **id_creator**: INT – identifies the player who created the lobby.
  - **name**: VARCHAR(255)- lobby name.
  - **state**: VARCHAR(255), DEFAULT 'PRIVATE' – indicates lobby visibility (private or public).
- **Relationships :**
  - Foreign Key: creator_id references players(id).
- **Constraints :**
  - CHECK Constraint: CHECK (state IN ('PUBLIC', 'PRIVATE')) ensures only valid lobby states.

- **Indexes :**
  - Index on creator_id to facilitate queries by lobby creator.
  - Index on state to quickly filter public versus private lobbies.

To store personalized lobby parameters like changing the timer value, number of players and bots in a game and other parameters, a lobby_parameters table is used, with a 1-1 relationship FK-lobby_parameters(id_lobby) references lobbies(id) .

## 2.3 Games

- **Description :** Tracks game sessions.
- **Primary Key :** id (INT, AUTO-INCREMENT)
- **Fields:**
  - **id_lobby**: INT – identifies the lobby associated with the game.
  - **id_winner**: INT – references the player ID who won the game.
  - **status**: VARCHAR(255), DEFAULT 'STARTED' – represents the game state (allowed values: 'STARTED', 'PAUSED', 'FINISHED').
  - **started_at**: TIMESTAMP – timestamp for when the game starts.
  - **finished_at**: TIMESTAMP – timestamp for .
  - **global_score**: JSON – stores the final scores of the players in a game.
  - **current_status :** JSON - current status of game boards, players cards and current scores
  - **current_round**: INT - number of current round (essential for if the player set the end of game by number of rounds in lobby parameters).
- **Relationships :**
  - Foreign Key: id_winner references players(id).
  - Foreign Key: id_lobby references lobbies(id).
- **Constraints :**
  - CHECK Constraint: CHECK (status IN ('STARTED', 'PAUSED', 'FINISHED')) ensures valid game statuses.

## 2.4 Sessions

- **Description :** Manages user sessions for authentication and tracking game activity.
- **Primary Key :** id (INT, AUTO_INCREMENT)
- **Fields:**
  - **id_player**: INT – references the associated player.

- **token**: VARCHAR(255) – session token for user authentication.
- **created_at**: TIMESTAMP – timestamp when the session was created.
- **expire_at**: TIMESTAMP – timestamp when the session expires.

- **Relationships :**
  - Foreign Key: id_player references players(id).

### 2.5 game_players

- **Description :** Intermediate table mapping players to game sessions and tracking individual scores.
- **Primary Key :** (id_game, id_player) composite key.
- **Fields:**
  - **id_game**: INT – references the game session.
  - **id_player**: INT – references the participating player.
  - **score**: INT – stores the player's score, updated after each round.
- **Relationships :**
  - Foreign Key: id_game references games(id).
  - Foreign Key: id_player references players(id).

### 2.6 pasword_reset

- **Description:** Manages password reset and authentification in case of forgetting the password by storing the reset token and it's expiration date.
- **Fields :**
  - **id_player:** INT- ref to the player's ID
  - **reset_token** VARCHAR(255)- unique code for the resetting.
  - **expires_at :** TIMESTAMP- expiration date of the token.
  - **used :** TINYINT(1) - default 0 (false).
- **Realtionships :**
  - Foreign Key: id_player references players(id).

### 2.7 Assets tables

These tables manage game-related assets that are **cards, icons** and **sounds**.

- **Description :** Stores details about each card, icon and sound.
- **Primary Key:** id (INT, AUTO-INCREMENT)
- **Fields :**
  - **id**: Unique asset ID
  - **file_path**: VARCHAR(255) – path or URL to the card's image asset.
  - **alt**: VARCHAR - brief description of the card

  Additional fields for cards table
  - **points**: INT - numerical value or point cost associated with the card.
  - **card-number**: INT – number of the card (from 1 to 104)

## 3 View:

The **leaderboard** view is designed to display player rankings based on their total scores. It consolidates key player statistics, such as scores and wins, and presents them in descending order to highlight the top performers.

```
1
2  CREATE VIEW leaderboard AS
3  SELECT p.username, ps.score, ps.total_played, ps.total_won
4  FROM player_stats ps
5  JOIN players p ON ps.id_player = p.id
6  ORDER BY ps.score DESC
7  LIMIT 10;
8
9  --Query :
10 SELECT * FROM leaderboard;
```

This view serves as the primary source for the leaderboard display, ensuring that rankings are always up-to-date without the need for additional data storage or manual updates. By centralizing the ranking logic within the view, the system maintains consistency and simplifies the retrieval of leaderboard data.

## 4 Event :

This event is scheduled to clean the 'password_reset' table by deleting the rows that contain a used token every 10 minutes.

```sql
CREATE EVENT delete_used_rows
ON SCHEDULE EVERY 10 MINUTE
DO
BEGIN
    DELETE FROM password_reset WHERE used = 1;
END;
```

# 5 Future Considerations

- **Scalability**: As the number of users grows, implementing caching strategies (e.g., Redis) or transitioning to distributed databases could be considered to enhance performance.
- **Security Enhancements:** To ensure long-term data protection, continuous security improvements are essential. Regular security audits—such as maintaining an audit log to track critical actions like profile deletions can help detect and prevent unauthorized changes.

```sql
CREATE TABLE audit_log (
    id INT AUTO_INCREMENT PRIMARY KEY,
    action_type VARCHAR(50),
    affected_table VARCHAR(50),
    action_time TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    performed_by VARCHAR(50)
);
```

- **Backup and Recovery:** Implementing automated backup strategies using tools like mysqldump ensures that data can be restored quickly in case of failure or data loss.

```
    mysqldump -u your_username -p 6_takes_db > db/backup/6_takes_backup
        .sql
```

  Periodic testing of backup files and storing them in secure, off-site locations (e.g., cloud storage) can enhance disaster recovery readiness.

- **Performance Optimization:** As the game evolves, continuously monitoring query performance and indexing strategies will be essential. Using tools like EXPLAIN to analyze slow queries and optimizing joins or frequently accessed tables can maintain fast response times.
- **Feature Expansion:** Future iterations of the game might require the database to support new features like seasonal tournaments, team-based modes, or social interactions. Designing the schema with flexibility in mind will ease future integrations and updates.

This roadmap ensures the database remains robust, secure, and adaptable as the project grows and user expectations evolve.

# 6 Appendix

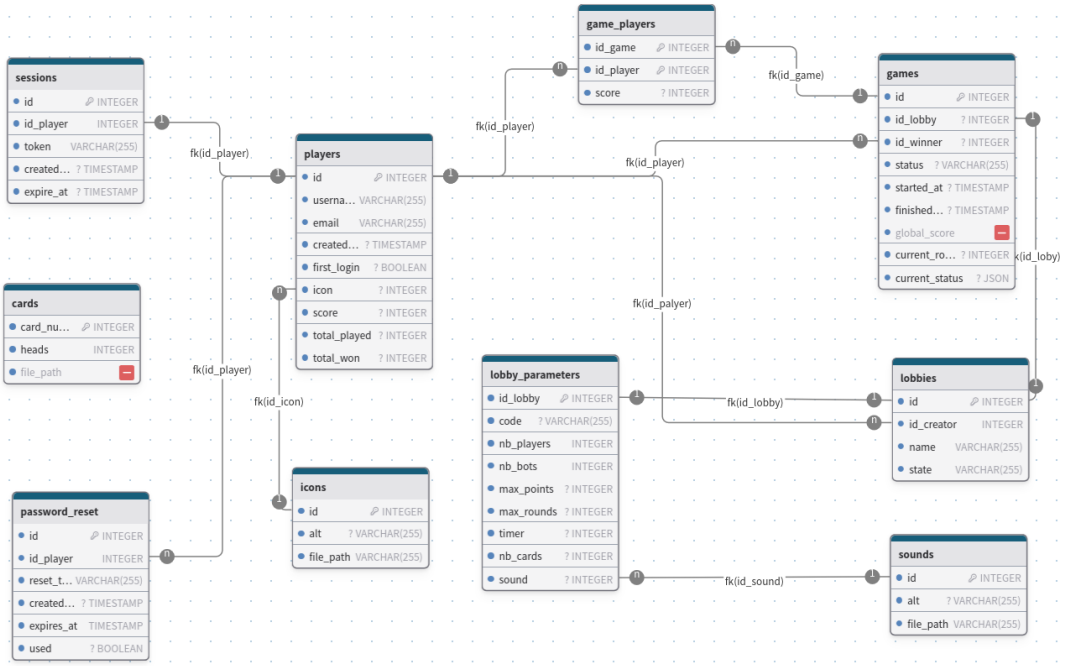


Figure 1: DB ER diagram