Completionist Tracker – Database Design

Dorian Kavadlo
July 2025

1. Database Technology Choice & Justification

For this project, I decided to use a relational database, specifically PostgreSQL, to store the app's data.

The Completionist Tracker deals with structured information where different parts of the data are related. Users can track their own games, each game has a checklist, and users can also import shared community checklists. Because of all these connections, a relational database made the most sense. PostgreSQL handles these kinds of relationships very well using foreign keys and joins. It also keeps the data consistent and organized through normalization.

Since users are constantly updating their progress, a relational model also helps keep things accurate and easy to manage.

I'll set up a PostgreSQL user account that the backend will use to access the database. This account will have permission to read and write to all the tables. It will be stored securely in environment variables so it's not exposed on the frontend.

2. Tables and Structures

Table: User

Purpose:
This table stores the users who sign up and manage their own games and checklists.

Fields:

- user_id: INTEGER, PRIMARY KEY, AUTO_INCREMENT
- username: VARCHAR
- email: VARCHAR
- password_hash: VARCHAR

Interaction:
Users make an account with their email and password. All of their data (like games and checklist items) connects back to their user_id.

Table: Game

Purpose:
This table holds all the games that a user is tracking in their personal library.

Fields:

- game_id: INTEGER, PRIMARY KEY, AUTO_INCREMENT
- user_id: INTEGER, FOREIGN KEY → User(id)
- title: VARCHAR
- platform: VARCHAR
- genre: VARCHAR

Interaction:
Users can add games manually after logging in. Each game is linked to the user who added it and contains its own list of goals or tasks.

Table: ChecklistItem

Purpose:
Checklist items are the specific goals the user wants to complete for each game.

Fields:

- checklist_item_id: INTEGER, PRIMARY KEY, AUTO_INCREMENT
- game_id: INTEGER, FOREIGN KEY → Game(id)
- description: TEXT
- completed: BOOLEAN
- order: INTEGER

Interaction:
Each game has its own checklist page where users can see and check off tasks. The app updates their progress automatically by calculating how many tasks are marked complete.

Example:
For the game *Hollow Knight*, a checklist item might be:

- description: "Rescue all Grubs"
- completed: false

- order: 3

Table: CommunityChecklist

Purpose:
This table stores templates created by other users that can be shared with the community and imported.

Fields:

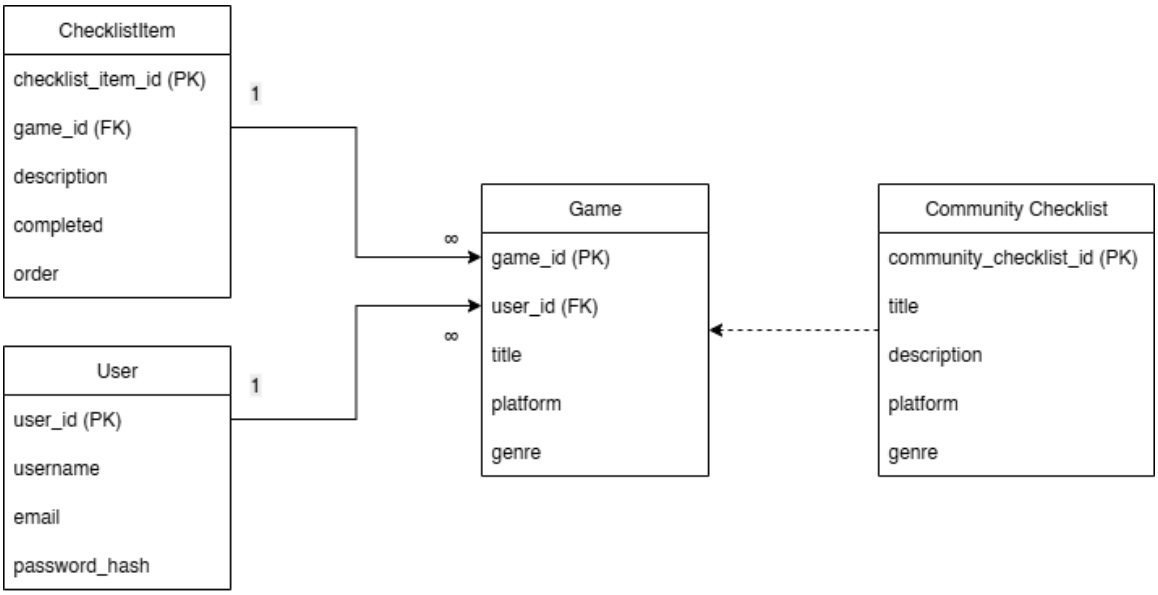
- community_checklist_id: INTEGER, PRIMARY KEY, AUTO_INCREMENT
- title: VARCHAR
- description: TEXT
- platform: VARCHAR
- genre: VARCHAR

Interaction:
Users can browse these templates and choose to import one into their own library. Once it's imported, they can edit it and track progress like any other checklist.

Example:
A community list might be called *Metroid Prime – Minimal Items Run* and include a description like "Finish the game with no energy tanks."

3. Relationships Between Tables

- One User can have many Game entries (**one-to-many**)
- One Game can have many ChecklistItems (**one-to-many**)
- CommunityChecklist is a standalone template, but users can import it, which creates a new Game and its ChecklistItems in their own account (**one-to-many** at the time of import)


ChecklistItem
checklist_item_id (PK)
game_id (FK)
description
completed
order
1
∞
Game
game_id (PK)
user_id (FK)
title
platform
genre
Community Checklist
community_checklist_id (PK)
title
description
platform
genre
User
user_id (PK)
username
email
password_hash
1
∞