RDBS

Ahmet Oğuz Ergin

# Introduction

This document provides an example of a basic relational database structure and its relationships, demonstrated through a code-stream project <https://github.com/ahmettoguz/code-stream-docker-config>. The goal is to explore and revisit key concepts that are often forgotten over time but are essential during the initial phases of a project.

# Project Description

The code-stream project aims to deliver redemption codes to players for various games. For the purpose of this guide, the documentation covers one-to-one, one-to-many, and many-to-many relationships.

# ERD

The following is the Entity Relationship Diagram (ERD) created with Chen Notation in pgAdmin. While the diagram primarily displays 1-N relationships, all relationship types are covered and explained in detail in the subsequent sections.

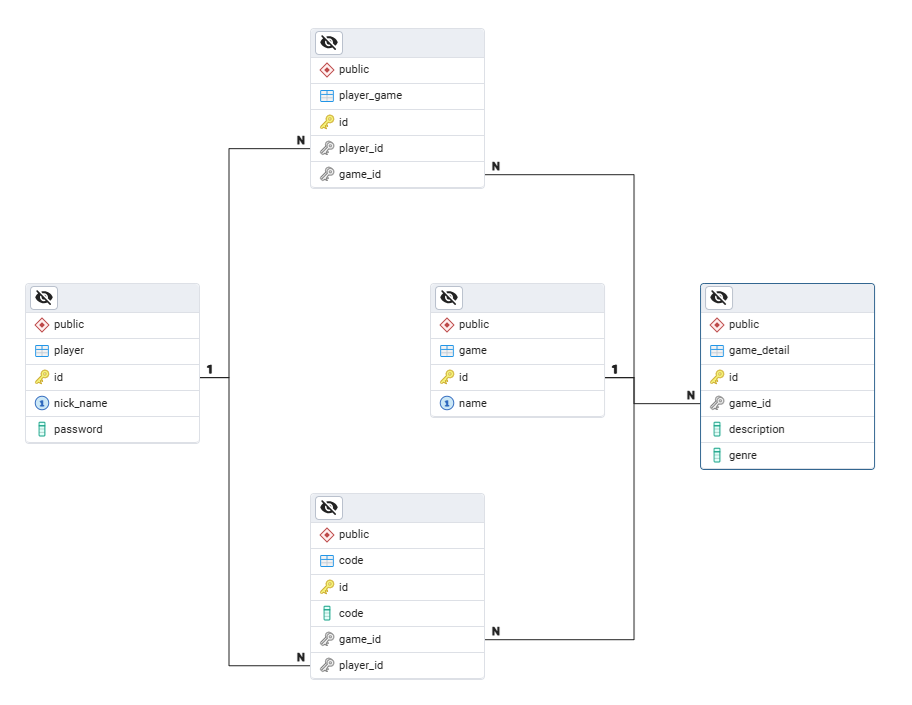


Figure 1: Entity Relationship Diagram of code-stream Project

# Relationship Between Objects

To better understand the structure of the database, the relationships between objects are explained as follows.

## Game and Game Detail Relation (One to One)

* A game can have only one game detail.
* A game detail can only belong to one game.

## Game and Code Relation (One to Many)

* A game can have multiple redemption codes.
* A redemption code can only belong to one game.

## Player and Code Relation (One to Many)

* A player can own multiple redemption codes.
* A redemption code can only belong to one player.

## Player and Game Relation (Many to Many)

* A player can be registered for multiple games.
* A game can have multiple players.

# Tables

Each table and field description is explained below, along with example data.

## player

The simple player table is shown in the following figure.

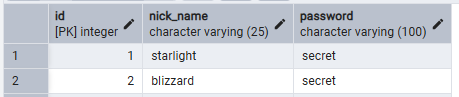


Figure 2: player Table With Data

## game

The simple player table is shown in the following figure.

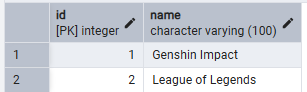


Figure 3: game Table With Data

## player\_game

Since there is a many-to-many relationship between players and games, an additional table is required to store their relationship. There are two options for defining the primary key:

* Composite Key (player\_id and game\_id): This approach works but has a drawback. If "ON DELETE SET NULL" is assigned, and you want to use it (for example, if Player 1 is removed from the player table but you don't want to lose related data), setting the player\_id to NULL would not be possible because the primary key cannot contain NULL values.
* Using an id field as the primary key This approach is simpler, easier to use, and more straightforward to understand.

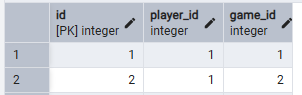


Figure 4: player\_game Table With Data

## game\_detail

The game detail table stores additional data for each game. There is a one-to-one relationship between the game and its details, with the game\_id field acting as a foreign key referencing the id field in the game table. There are two options for defining the primary key:

* Using the game\_id field as both the primary and foreign key: This approach works, but as mentioned earlier, if you want to set the game\_id to NULL upon the deletion of a game, it would not be possible because the primary key cannot contain NULL values.
* Using a id field as the primary key: This is a simpler and more intuitive approach. In this case, the game\_id field is UNIQUE, as there is a one-to-one relationship between the game and its details. The only distinction between a one-to-many and a one-to-one relationship is that the foreign key in a one-to-one relationship must be UNIQUE. It's that simple.

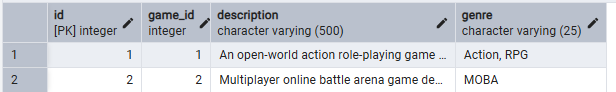


Figure 5: game\_detail Table With Data

## code

This table has two relationships, both of which are one-to-many.

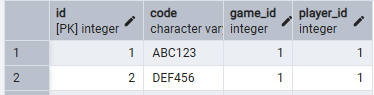


Figure 6: code Table With Data