UML Diagram:

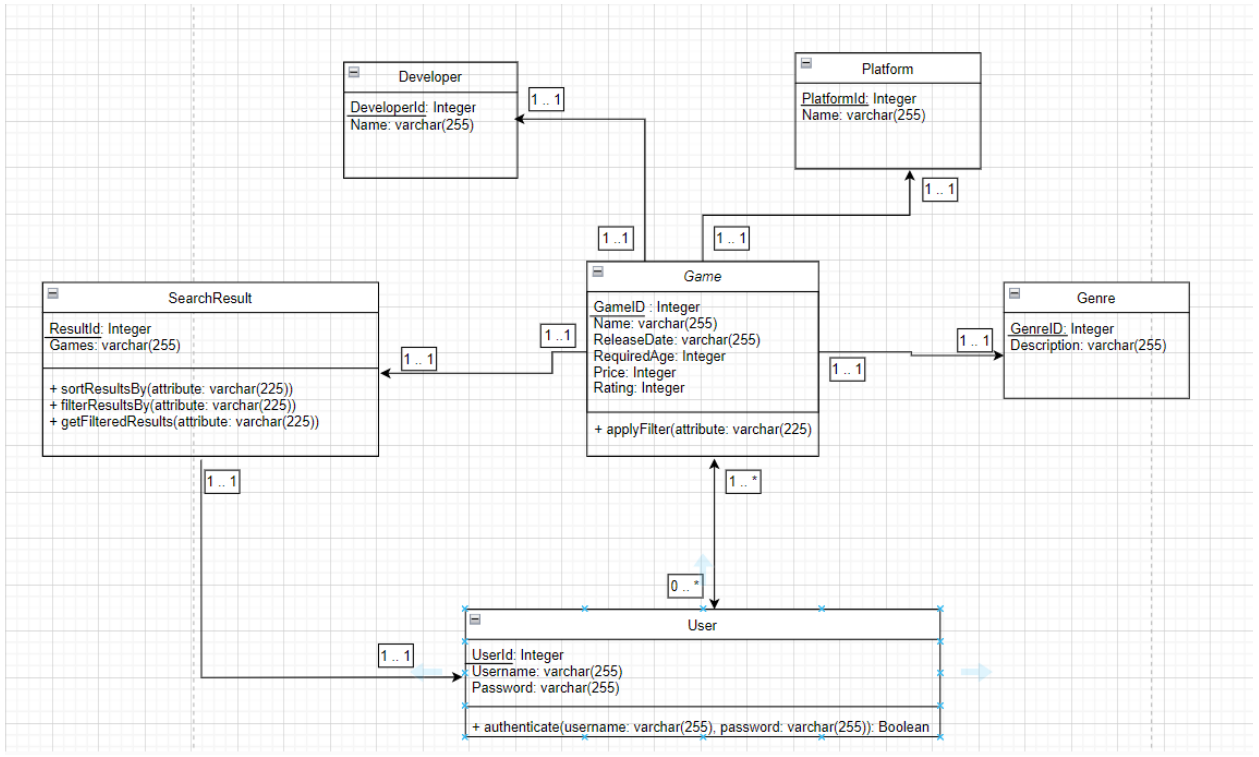

Table: Game

GameID (PK)
GameName
ReleaseDate
Price

Table: Genre
GenreID (PK)
Genre
GameID (FK to Game)

Table: Platform
PlatformID (PK)
Platform
GameID (FK to Game)

Table: User
UserId (PK)
UserName

Password

Table: Developer
DeveloperId
Name

Assumptions:

<u>Entities:</u>

User Entity:
Attributes: UserID (PK), Username, Email, Password, ProfilePicture, Preferences
Assumptions:

- Each user can have 1 profile
- Each profile corresponds to 1 user

Game Entity:
Attributes: GameID (PK), GenreID(FK), GameName, ReleaseDate, Price, Description
Assumptions:

- Each game can have 1+ associated genres
- Each genre corresponds to 1+ games
- Each game can be compatible with multiple gaming platforms
- Each gaming platform can support multiple games.

Genre Entity:
Attributes: GenreID (PK), GenreName
Assumptions:

- Each genre can be associated with multiple games.
- Each game can belong to multiple genres

Platform Entity:
Attributes: PlatformID (PK), PlatformName
Assumptions:

- Each gaming platform can support multiple games
- Each game can be compatible with multiple gaming platforms

Interaction Entity:
Attributes: InteractionID (PK), UserID (FK), GameID (FK), InteractionType, InteractionDate
Assumptions:

- Each user can have one or more interactions with games
- Each game can have interactions with multiple users
- Each interaction corresponds to one user and one game
- Each interaction has attributes such as InteractionType and InteractionDate

Relationships:

User-Game Relationship:
Relationship Type: Many-to-Many
Assumptions:y

- Users can interact with multiple games
- Games can have interactions with multiple users.
- Additional attributes capture the nature of these interactions

Game-Genre Relationship:
Relationship Type: Many-to-Many
Assumptions:

- Games can belong to multiple genres (e.g. A game can be both Action & RPG)
- Genres can be associated with multiple games

Game-Platform Relationship:
Relationship Type: Many-to-Many
Assumptions:

- Games can be compatible with multiple gaming platforms (e.g. Windows, Linux, and Mac)
- Gaming platforms can support multiple games

Normalization:

BCNF Normalization:
The schema above has the following functional dependencies for each relation:

User:
UserID -> Username, Password

Game:
GameID -> Name, ReleaseDate, RequiredAge, Price, Rating

Genre:
GenreId -> Description

Platform:
PlatformId -> Name

Developer:

DeveloperId -> Name

SearchResult:

ResultId -> Games

Our relations were defined in straightforward manner such that the functional dependencies can be proven to have a superkey for their respective relations. Computing the attribute closures of each follows:

{UserId}+ = Username, Password
User attributes are: Username, Password
UserId is a superkey of User

{GameId}+ = Name, ReleaseDate, RequiredAge, Price, Rating
Game attributes are: Name, ReleaseDate, RequiredAge, Price, Rating
GameId is a superkey of Game

{DeveloperId}+ = Name
Developer attributes are: Name
DeveloperId is a superkey of Developer

{PlatformId}+ = Name
Platform attributes are: Name
PlatformId is a superkey of Platform

{ResultId}+ = Username, Password
SearchResult attributes are: Username, Password
ResultId is a superkey of SearchResult

The set of relations fits BCNF form since for each relation the non-trivial functional dependencies all have a superkey on their left-hand side.

We chose to use the BCNF form to normalize our database because it was effective in minimizing information loss and avoiding redundancy, which made sense given the nature of our database where we wanted to ensure accuracy. It is also efficient in that it gives us lossless join, although in comparison to 3NF, it does not give us dependency preservation - which matters less in the context of our project where we are giving recommendations.

Relational Schema:
Table-Name(Column1:Domain [PK], Column2:Domain [FK to table.column], Column3:Domain,...)

User (

```
   UserID:INT[PK]
   Password:VARCHAR(255)
   Username:VARCHAR(255)
)

Game (
   GameID:INT[PK]
   Name:VARCHAR(255)
   ReleaseDate:VARCHAR(255)
   RequiredAge:INT
   GenreID:INT[FK to Genre.GenreId]
   PlatformID:INT[FK to Platform.PlatformId]
   DeveloperId:INT[FK to Developer.DevelopId]
   Rating:INT
   Price:Int
)

Genre(
   GenreId(PK)
   Description:VARCHAR(255)
)

Platform(
   PlatformId(PK)
   Description:VARCHAR(255)
)

Developer(
   DeveloperId(PK)
   Description:VARCHAR(255)
)

SearchResult(
   ResultId:INT(PK)
   UserId:INT(FK to User.UserId)
   Games:VARCHAR(255)
)
```

Fixing suggestions from previous stage:
A suggestion from the previous stage was to expand on functionality other than search.

A major feature our project intends to have is to also store a history for each user in order for users to be able to reference previous purchases and queries. The table will be smaller in size compared to other tables such as the massive Game table. This will help the user understand their recent history only since that would be of more importance to them than queries made months ago.

This history table will be produced from the output of a filter procedure we will have whenever a user wants to query our game database. When a user has expressed interest in a game by either filtering by categories such as genre or through a “shopping cart/starred” feature, the games can effectively be stored in history so users can view what they have been interested in the past.

In order for a user to access their history database it would be as simple as getting all records from this table. If the user only wants the most recent history we can then limit the amount of records returned.

The columns of this table will be a list of games and the UserId which will relate to a specific user. The list of games will represent the history for the user. The UserId will ensure that a specific history can be linked to a user. Therefore, our history table will have a one-to-one relationship with the user table. This way any user will have access to their own history.