Kaylee Moore and Jack Lindner  
**Game Rental Database**  
CS 461, Database Systems Final Project  
Professor Sutton

This project was designed to help students learn how to create a database management system from scratch. Given a project PDF, we set out to create an in-depth database setup to demonstrate our skills in database management systems (DBMS).

Our goal with this project was to provide a system that manages the inventory and rental of video games in a rental store network of franchises. In this franchise system, there are many stores, each in a different region carrying different stock and having different employees and customers.

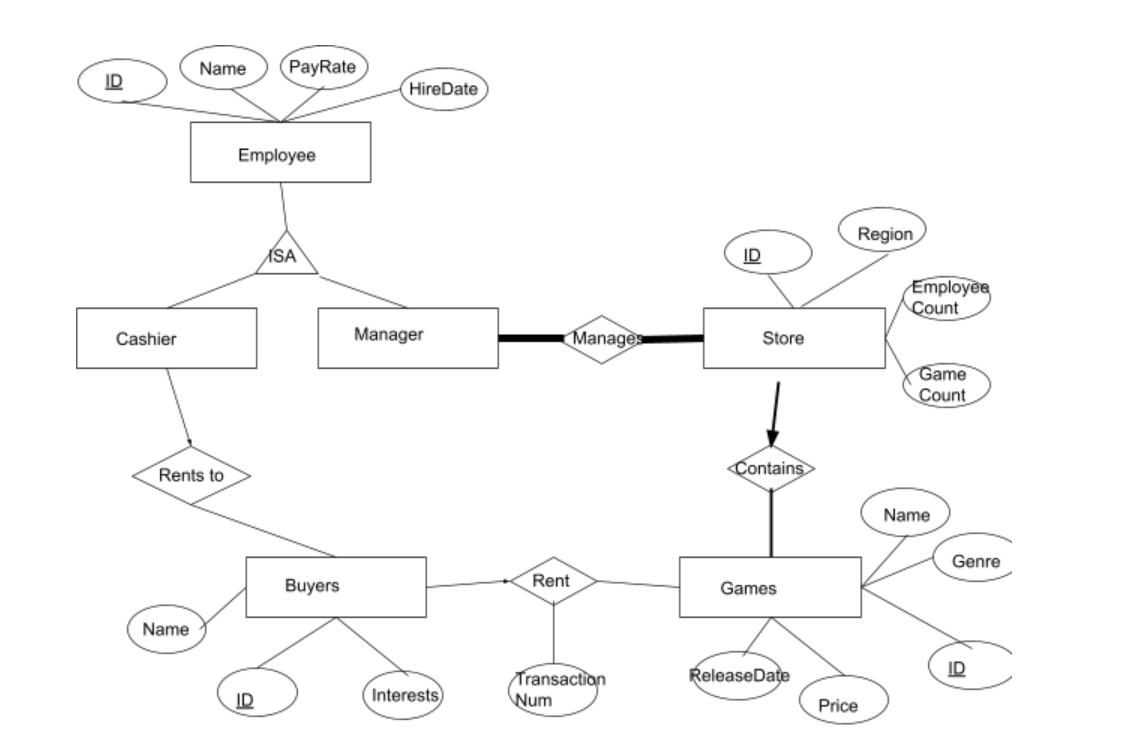
One of the main technical problems we faced when designing this database system was regarding the editing of the database itself. Adding objects to the tables in the database required a bit of creative work to form the custom SQL statements embedded in the code.

Some of the main requirements of this database were: buyers register with their name and interests, buyers can view game information, buyers can search for games at either their region’s store or different store, cashiers and managers can check out buyers, buyer can see cashiers name but no other information, everyone can see if game is rented out currently, cashiers and managers can view all store information, buyer can only view store region and game count, managers can edit inventory manually, and cashiers can but all game information but not edit the game information.

In order to mitigate redundancies, we decided to break up all the items as we built the schema. We did this by choosing not to make a single entry for each game and have a bunch of attributes, making everything self-contained instead. This eliminates overlap and redundancies between the tables which leaves less room for errors and unexpected outputs.

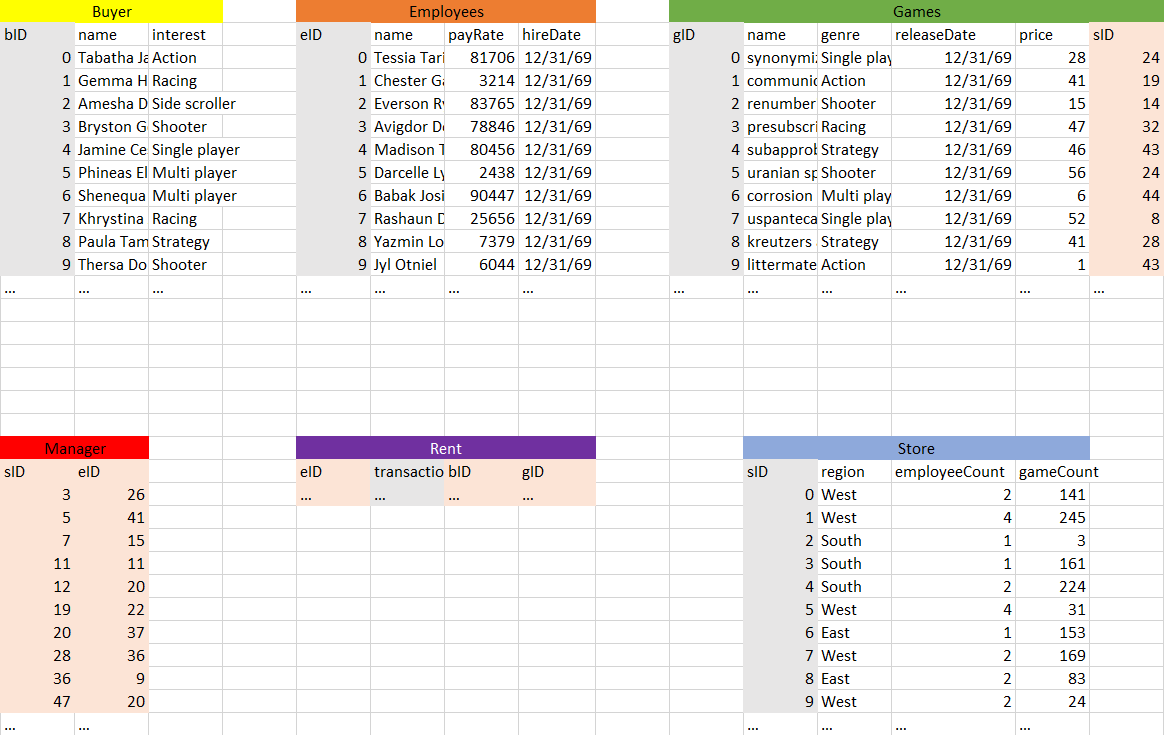
When designing the GUI, we went for a very simple and easy-to-follow design. With five buttons along the top and a dropdown selection, you can specify which table you would like to view and then act upon that data. Some actions available currently are add, delete, and modify entries, and move between pages using “previous” or “next” buttons.

ER Diagram:



We decided to break up the Entity-Relationship diagram in the fashion above so that we would reduce functional dependencies and possible weak points in our database.

BCNF of tables:



As discussed earlier in the report, we designed our database with redundancies and dependencies in mind. In the image above, gray columns are primary keys and the beige columns are foreign keys. bID from Buyer, eID from Employees, gID from Games, transactionNum from Rent, and sID from Store are all unique keys that are never duplicated. This improves our database setup because we will not need to create a different unique key for each table. Since we built the database with redundancies and dependencies in mind, BCNF did not have an effect on our database.

To generate the random data in the database, we used two sets of Internet-provided lists and one list generated by Jack Lindner. The first list, genres.txt, was created by Jack and has a bunch of random game genres that he could think of off the top of his head. The two others, names.txt and words\_alpha.txt, are both pulled of the Internet. We then used those three lists and the Java random number generator to randomly pick lines from each of these files to generate different things like renter names, game names, and game genres. To generate the pay rate of employees, we used the Math.random function to generate a number from 0 to 100,000. To generate the date for employee hiring and game release dates, we generated a random integer for each part of the date: 1 through 12 for month, 1 through 28 for day, and 1 through 2019 for the year and then assembled this into a string.

The user-interface for this database is laid out in a straightforward way. The 'New Transaction' button will allow you to input the information for a new transaction. The given dropdown prompts will show the available options for keys, as all information needs to be compatible with other tables. The 'Delete' button will display a popup box depending on the currently displayed table asking for the primary key of the item you are looking to delete. Entering the correct information will delete that entry in the table. The 'Update' button will display a series of popups, which will gather the information for a new entry to the currently displayed table. The 'Next' and 'Previous' buttons will cycle between the available tables. The dropdown box will allow you to pick a table directly, without needing to cycle through. The search textbox will filter the current displayed table, and only show the entries that match the entered information. You will need to press 'Enter' after your search query. Below the buttons, the currently displayed table will show all entries for the database table in question.

Project source code:

DatabaseUI.java

GenerateRandom.java

Discussion: When creating our database and GUI, we ran into numerous issues. Unlike other students in CS 461, we built our GUI from scratch using Java’s AWT framework. This made our application less pretty but taught me a great deal about designing GUIs for applications and connecting to databases. This also means we had to build a lot of the database interactions from scratch. We got a lot of practice in embedded SQL and table management throughout this project. The most interesting problem we had was the slight separation of the viewed tables and the database itself – the front-end tables had to be edited separately from the back-end database, which caused a lot of problems for a while. However, now all functions are synced and we got a lot more familiar with the ins and outs of database management.