CS437 Introduction to Database System
Project Report

Team member:
Tao, Chen (tc667)
Yang, Song (ys585)
Chenao, Wu (cw796)
Jingcan, Yang (jy549)
Git : https://github.com/sjzyjc/database_project

1.Introduction

This project creates a movie booking system that provides movies classified in a variety of genres and allows the user to make a booking. We consider this project to be very useful because booking systems exist everywhere on the internet, and are widely applied in many industries. A movie booking system is a very typical one of these, and the one we created could even be modified just in the database level to adapt other usages.

For our teammates, this project is particularly interesting because it not only allows us to create something particularly useful, as the aforementioned, but also satisfies our personal interests in films. Most movie booking websites only provide movies that are currently on, but do not archive classical films. Therefore, we would like to classify classical movies into our system.

2.Database design

2.1. Choice of entity sets

In our ER diagram, there are eight entity sets in total: *Room, Seat, Film, Actor, Genre, Schedule, Booking,* and *Customer.*

The database should store the basic information of the rooms and the seats as well as films and the actors. Thus, entity *Room*, *Seat*, *Film*, and *Actor* are needed. As one film may have many genres, *Genre* becomes an independent entity instead of an attribute of *Film*. Since one film can be shown on different dates, another entity *Schedule* is created, storing the show time separately.

The database should also keep the information of all bookings and customers that makes a booking, i.e. entity *Customer* and *Booking*.

2.2 Weak entity sets

There are two weak entities in our design: *Seat* and *Schedule*. For the entity *Seat*, the attributes 'row number' and 'seat number' can be different in one room, but could be the same in different rooms. To identify each tuple of the *Seat*, the entity *Room* provides its key 'room number' as part of the key of *Seat*.

For the entity *Schedule*, the attributes 'date' and 'start time' can be different for a particular movie, but different movies can have the same date and start time. So the entity *Film* provides its key "title" and "year" as part of the key of *Schedule*.

2.3. Cardinality of relationships

2.3.1 Binary relationships

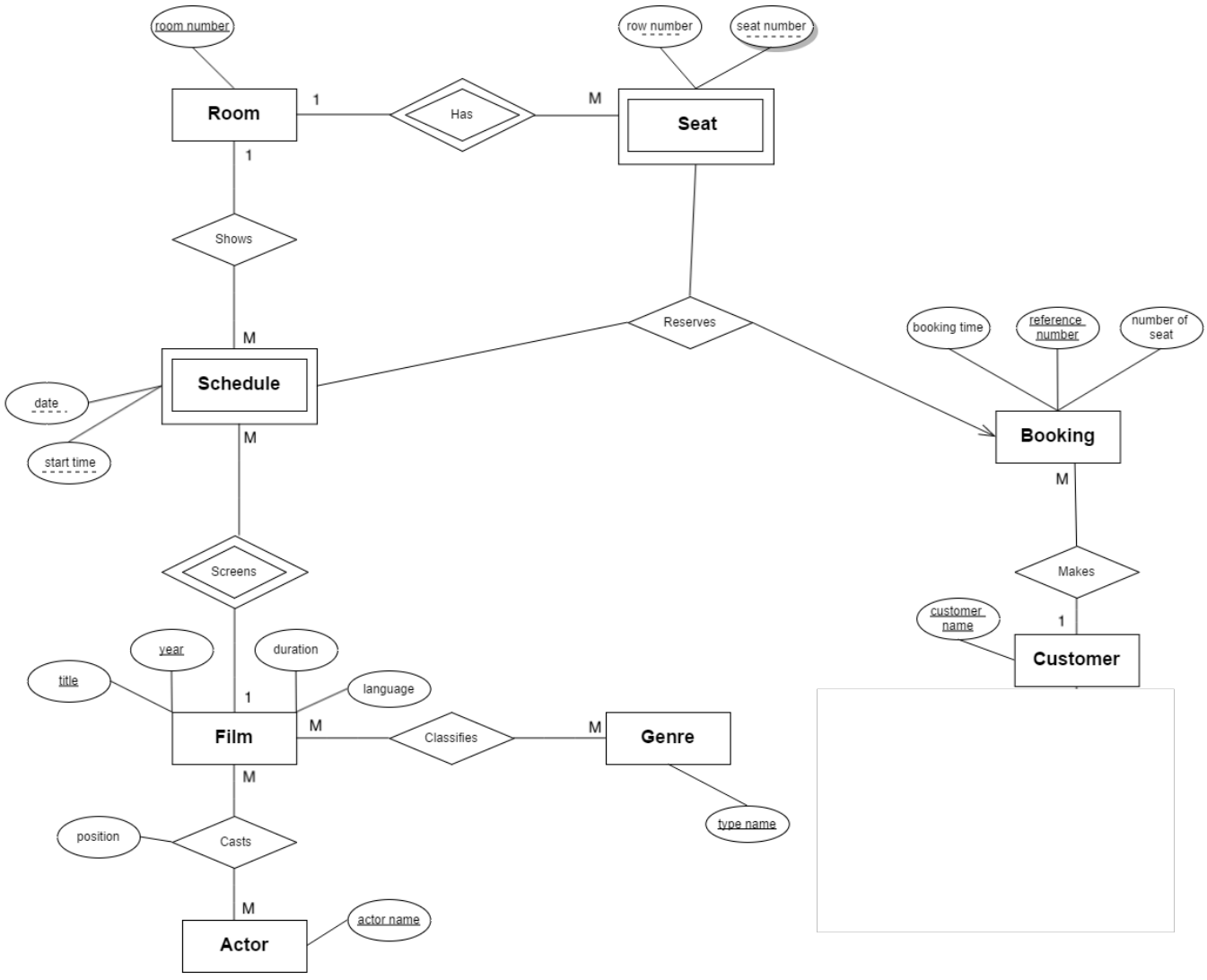
'*Room-Seat*' is a one-to-many relationship with referential integrity; because a seat must belong to one room and one room has many seats. '*Room-Schedule*' is a many-to-many relationship, because many rooms can hold a show at the same time, and one room can arrange shows of different date and time. *'Film-Actor'* is a many-to-many relationship, because a film is acted by many actors/actress, and an actor/actress can cast in many films. '*Film-Genre*' is a many-to-many relationship, because a film can be in several genres, and a genre can refer to many films. '*Schedule-Film*' is a one-to-many relationship with referential integrity, since the key of *Schedule* is borrowed from the *Film* entity. '*Customer-Booking*' is a one-to-many relationship

with referential integrity, because a customer can book many times, but one booking must be made by one customer.

2.3.2 Ternary relationships

To store the detailed information of customer's reservation, a ternary relationship is used between entity *Schedule*, *Booking* and *Seat* with a cardinality of 1:1:M. For each pair of *Booking* and *Seat*, there must be one instance of *Schedule*. For each pair of *Seat* and *Schedule*, there is at most one instance of *Booking*. For each pair of *Booking* and *Schedule*, there may be at least one seats booked.

2.4 ER Diagram



2.5 Normalization Analysis

1. Room (room_number)
    a. Keys: room_number
    b. Primary Key: room_number
    c. The relation is in 3NF.
2. Seat (room_number, row_number, seat_number)
    a. Keys: room_number, row_number, seat_number
    b. Primary Key: room_number, row_number, seat_number
    c. The relation is in 3NF.
3.Show (room_number, date, start_time, title, year)
    a. Keys: room_number, date, start_time
    b. Primary Key: room_number, date, start_time
    c. FDs: room_number, date, start_time → title, year;
    d. The relation is in 3NF.
4.Film (title, year, duration, language)
    a. Keys: title, year
    b. Primary Key: title, year
    c. FDs: title, year → duration, language
    d. The relation is in 3NF.
5.Actor (actor_name)
    a. Keys: actor_name
    b. Primary Key: actor_name
    c. The relation is in 3NF.
6.Genre (type_name)
    a. Keys: type_name
    b. Primary key: type_name
    c. The relation is in 3NF.
7.Booking (reference_number, booking_time, customer_name)
    a. Key: reference_number
    b. Primary Key: reference_number
    c. FDs: reference_number → booking_time, customer_name
    d. The relation is in 3NF.
8. Classifies (title, year, type_name)
    a. Keys: title, year, type_name
    b. Primary Key: title, year, type_name
    c. The relation is in 3NF.
9. Casts (title, year, actor_name, position)
    a. Keys: title, year, actor_name
    b. Primary Key: title, year, actor_name
    c. FDs: title, year, actor_name → position
    d. The relation is in 3NF.
10. Reserves (reference_number, room_number_1, row_number, seat _number, room_number_2, date, start_time)
    a. Keys (four candidate keys):
        · reference_number, room_number_1, row_number, seat_number;
        · room_number_1, row_number, seat_number, date, start_time;
        · reference_number, room_number_2, row_number, seat_number;
        · room_number_2, date, start_time, row_number, seat_number;
    b. Primary Key: reference_number, room_number_1, row_number, seat_number

c. FDs:
   - reference_number, room_number_1, row_number, seat_number → room_number_2, date, start_time
   - room_number_1, row_number, seat_number, date, start_time → reference_number, room_number_2
   - reference_number, room_number_2, row_number, seat_number → room_number_1, date, start_time
   - room_number_2, date, start_time, row_number, seat_number → room_number_1, reference_number
   - room_number_1→room_number_2
   - room_number_2→room_number_1
   - reference_number → room_number_2, date, start_time
   - reference_number→room_number_1

d. The relation is in 3NF

3. Implementation

3.1 Backend Implementation

We are using alaSQL to simulate a relational database and jquery to load and modify data from alaSQL database. AlaSQL is a lightweight client-side in-memory SQL database designed to work in browser and Node.js. It can support most of the SQL queries including table creation, insertion and modifications. Therefore, it is a perfect choice to develop an web application to illustrate our database structure.

3.2 Technical Challenges

One technical challenge we met was displaying the available seats for each show because this is an indirect information that we need to retrieve it by combining multiple tables i.e. reserves, seats and shows.

Another technical challenge was alaSQL does not support composite foreign keys, therefore we need to create a unique id for all seats and shows for the reserves table.

4. Future Improvements

Firstly, we could develop a full backend which will store data in disk instead of in brower's memory. We could migrate the project on Node.js - Express + MSSQL framework in the future. Then the system could work in distributed mode and more than 100 people could visit the website at the same time.

Secondly, people can register an account in this system. Then users can see what films will be on recently and save the films they like into wishing list. Also, we could add admin mode to modify tables other than reservations such as films and shows in the future. To support this, we may need a seperate table to store login credentials as well as developing a separate login page for administrators.

Thirdly, once we persist our data in real database like MSSQL database, we could add more features on the system such as writing reviews of movies and giving some rewards to the users who share the platform to their friends. In this way, the platform could be promoted and attract more users to use it. Maybe in the future, the platform could make a profit.