# CPS510 - A8

By: Isaac Martin, Bowie Chau, Hachi Ndu

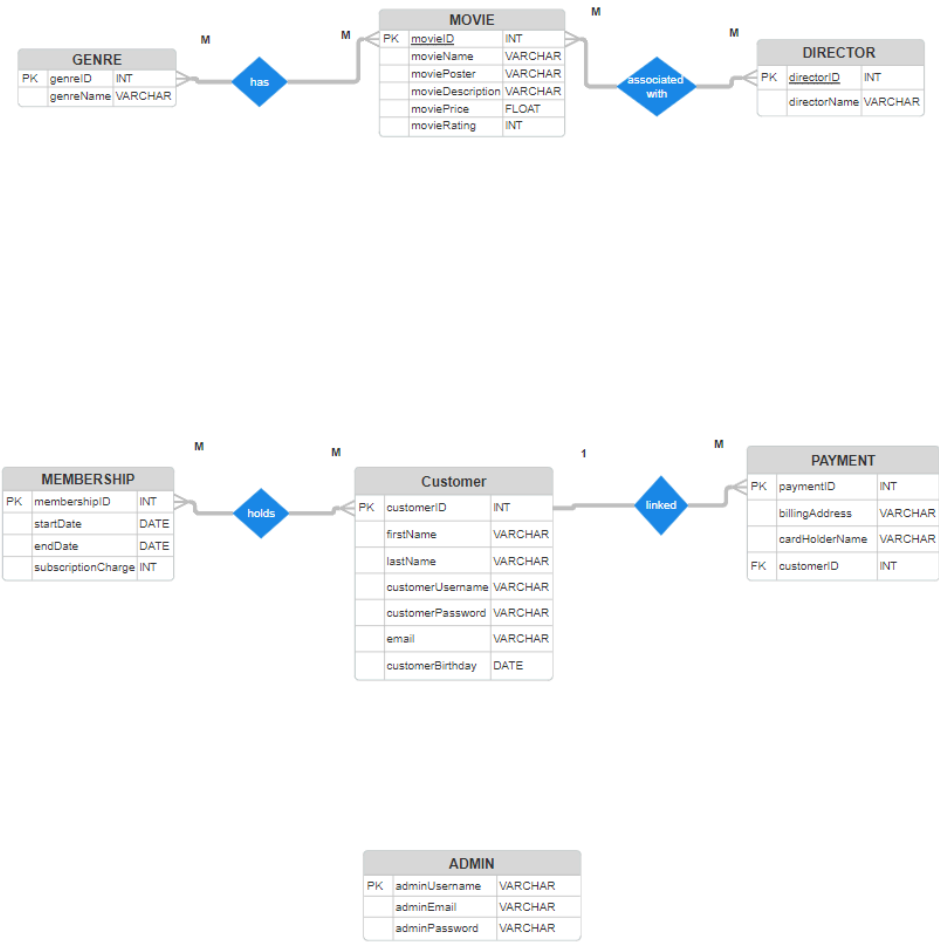
Instructor: Dr. S. B. Tajali

TA: George Lopez

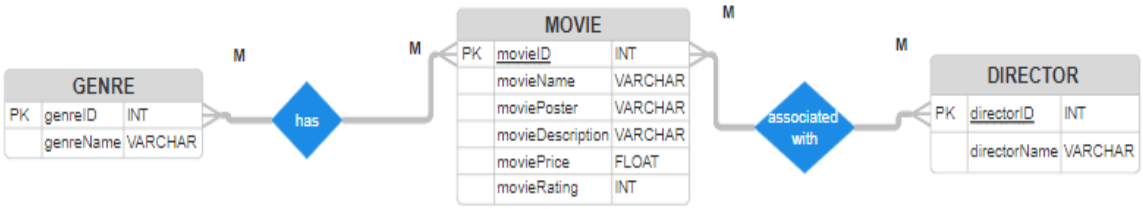
Section #: 06

Lab Number: 8

Nov 15th, 2024




After the completion of our design for our DBMS system, we arrived at the step of normalizing our design. In order to normalize our design, we first outline the functional dependencies in the system. For example, our many-to-many relationship from movie to genre and movie to director.




movieID uniquely identifies each movie and determines all attributes. Each movie can be associated with multiple directors and genres. There the functional dependencies (FD) that would express this relationship would be:

movieID → movieName, moviePoster, movieDescription, moviePrice, movieRating.

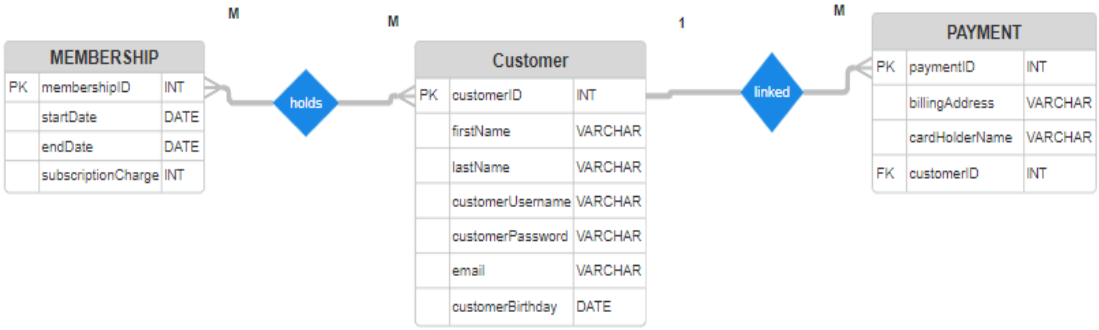
The director table FD is:

directorID → directorName

and is a Many-to-Many relationship with Movie because the directorID uniquely identifies each director and determines directorName. Each director can be linked to multiple movies. The Genres table FD is:

genreID → genreName

and is a Many-to-Many relationship with Movie the genreID uniquely identifies each genre, determining genreName. Each genre can be associated with multiple movies.




The customer's table FD is:

customerID → firstName, lastName, customerUsername, customerPassword, email, customerBirthday

customerUsername → customerID, firstName, lastName, customerPassword, email, customerBirthday

email → customerID, firstName, lastName, customerUsername, customerPassword, customerBirthday.

The relationship is a One-to-Many relationship with Payment thus each customer can have multiple payment months. There is also a Many-to-Many relationship with Membership. The

customerID, CustomerUsername, and email each uniquely identify a customer and determine all attributes. A customer can have multiple payments and memberships.

The FD to payment table is:

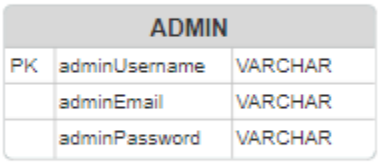
paymentID → billingAddress, cardHolderName,

customerID because payment ID uniquely identifies each payment, determining billingAddress, cardHolderName, and the linked customerID.

The FD to membership is:

membershipID → startDate, endDate, subscriptionCharge,

because membershipID uniquely identifies each membership, determining attributes like startDate, endDate, and subscriptionCharge. A membership can be associated with multiple customers.



| ADMIN | | |
|---|---|---|
| PK | adminUsername | VARCHAR |
| | adminEmail | VARCHAR |
| | adminPassword | VARCHAR |

The admin table FD is:

adminUsername → adminEmail, adminPassword

adminEmail → adminUsername, adminPassword

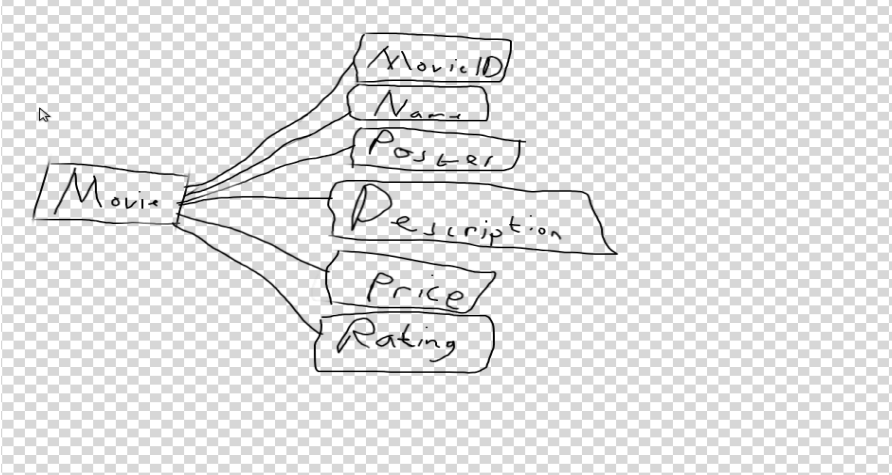
The relationship is independent from other tables meaning there is no direct relationship with other tables. The adminUsername and adminEmail uniquely identify each admin, determining the other attributes.

To ensure each table is in the Third Normal Form (3NF), we want to verify that:

1. All the tables are in the Second Normal Form(2NF): This means they have no partial dependencies (a non-key attribute depending only on part of a composite key).
2. No transitive dependencies: In 3NF, non-key attributes must not depend on other non-key attributes

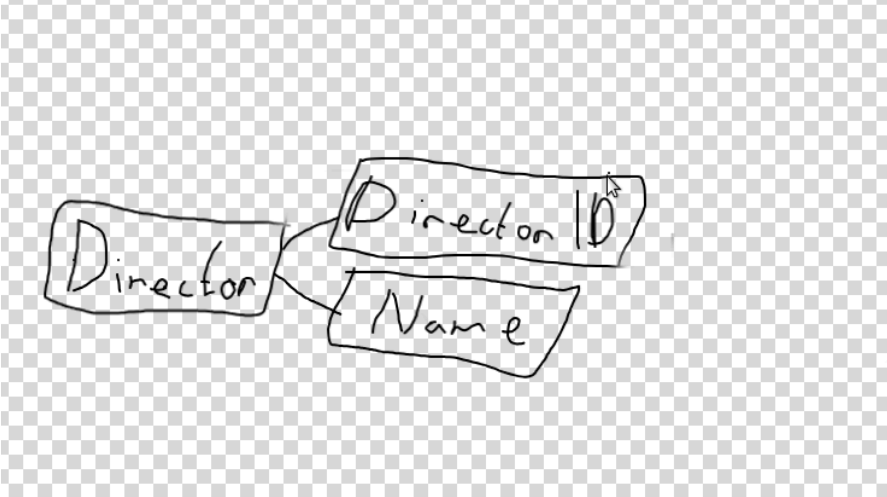
1. Movie Table

- Functional Dependencies (FDs): movieID → movieName, moviePoster, movieDescription, moviePrice, movieRating
- Verification: Since movieID is the primary key and determines all other attributes, this table has no partial or transitive dependencies, meaning it should already be in 3NF.



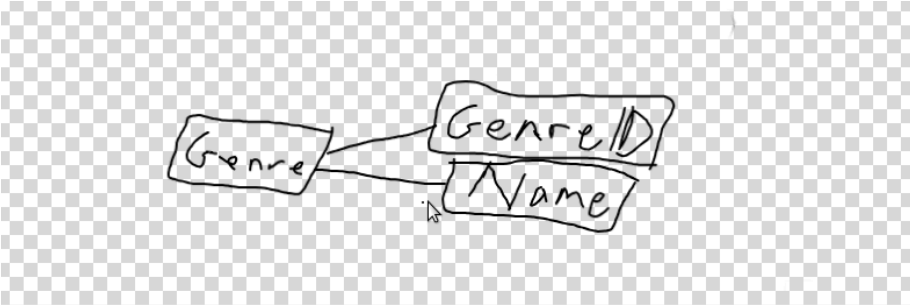

2. Director Table

- FDs: directorID → directorName
- Verification: directorID uniquely identifies each director without any non-key attributes depending on other non-key attributes. Therefore it is in 3NF.

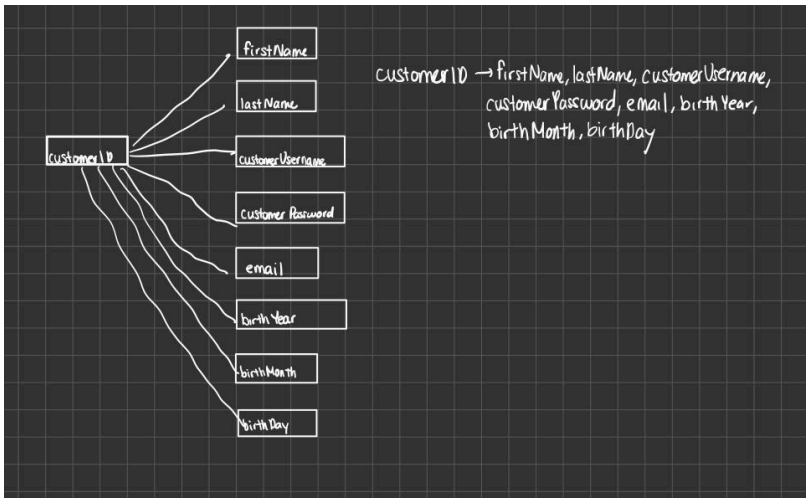


3. Genre Table

- FDs: genreID → genreName
- Verification: genreID uniquely identifies each genre, and there are no transitive dependencies. Therefore, it is in 3NF.

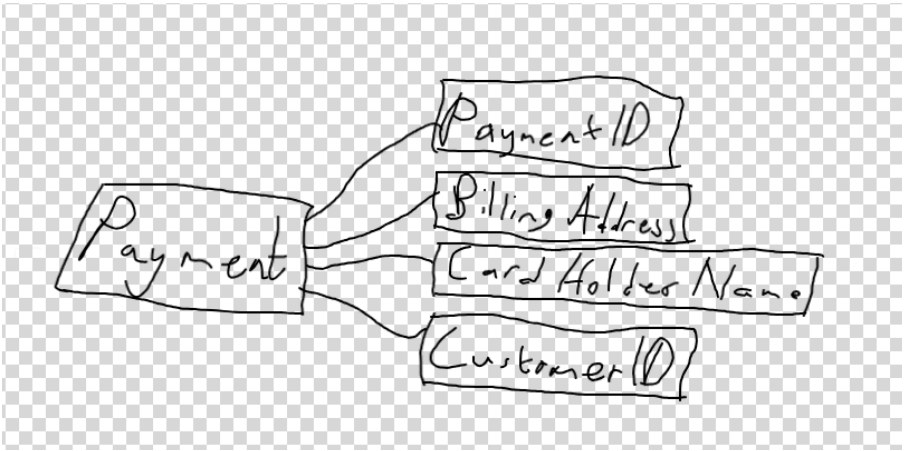



4. Customer Table




To turn Customer into 3NF, we changed the Birthday attribute. Initially we had just customerBirthday however we split it into a separate field for customerYear, customerMonth, customerDay making it more divisible for 1NF. Thus the Customer table is now 3NF.

5. Payment Table



- FDs: paymentID → billingAddress, cardHolderName, customerID
- To turn payment into 3NF, we changed the Address attribute. Initially this contained the street number and street making it divisible (breaching 1NF). So we split up the address into street number and street name
- paymentID → billingStreetNumber, billingStreetName, cardHolderName, customerID

# A8

Bernstein's Algorithm Steps:

1. Determine all functional dependencies
    a. Find and remove redundancies
    b. Find and remove partial dependencies
2. Find keys
3. Create tables

STEP 1 (find all functional dependencies):
Movie Table FDs:

- movieID → {movieName, moviePoster, movieDescription, moviePrice, movieRating, movieDirector, movieGenre}

Admin Table FDs:

- adminUsername → {adminEmail, adminPassword}
- adminEmail → {adminUsername, adminPassword} (ASSUMPTION: each email is unique)

Customer Table FDs:

- customerID → {firstName, lastName, customerUsername, customerPassword, email, customerBirthday}
- customerUsername → {customerID, firstName, lastName, customerPassword, email, customerBirthday}
- email → {customerID, firstName, lastName, customerUsername, customerPassword, customerBirthday} (ASSUMPTION: each email is unique)

Payment Table FDs:

- paymentID → {billingAddress, cardHolderName, customerID}
- customerID → paymentID (ASSUMPTION: each customer can make only one payment, or payment records per customer are unique)

Membership Table FDs:

- membershipID → {startDate, endDate, subscriptionCharge, customerID}
- customerID → membershipID (ASSUMPTION: one active membership per customer)

STEP a&b (Decompose by finding and removing redundancies and partial dependencies)
Decomposing the **Movie Table**

- Functional Dependencies:
    - movieID → movieName, moviePoster, movieDescription, moviePrice, movieRating, movieDirector, movieGenre
- Decomposition:
    - Since movieDirector and movieGenre can introduce redundancy (multiple movies can have the same director or genre), create separate tables:
        - Movie: movieID, movieName, moviePoster, movieDescription, moviePrice, movieRating
        - Director: directorID, directorName
        - Genre: genreID, genreName

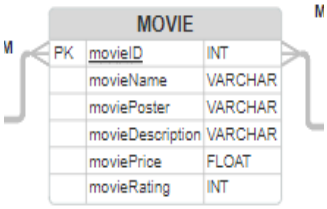
Decomposing the **Membership Table**

- Functional Dependencies:
    - membershipID → {startDate, endDate, subscriptionCharge}
    - customerID → membershipID (ASSUMPTION: each customer can have only one active membership)
- Decomposition:
    - Since customerID determines membershipID, we need to separate customerID from the main membership attributes.

STEP 2&3 (Find Keys and Create Tables):

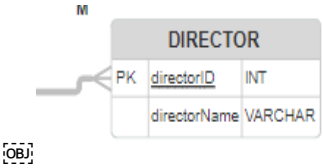
- Candidate Key:
    - movieID

movieID → movieName, moviePoster, movieDescription, moviePrice, movieRating.



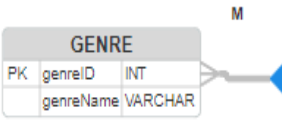

- Candidate Key:
    - directorID

directorID → directorName




- Candidate Key:
    - genreID

genreID → genreName



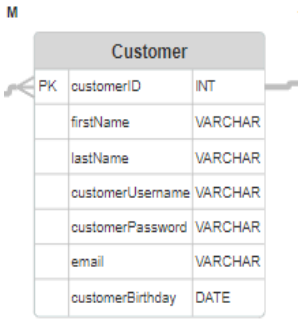

Candidate Keys:

- customerID
- customerUsername
- email

customerID → firstName, lastName, customerUsername, customerPassword, email, customerBirthday

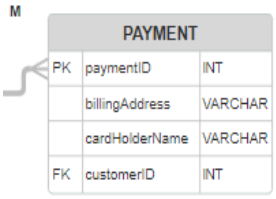
customerUsername → customerID, firstName, lastName, customerPassword, email, customerBirthday

email → customerID, firstName, lastName, customerUsername, customerPassword, customerBirthday.




- Candidate Key:
  - paymentID

OBJ paymentID → billingAddress, cardHolderName,




- Candidate Key:
  - membershipID

membershipID → startDate, endDate, subscriptionCharge,