CIS560 – Database System Concepts **Name:_____**

Assignment 3 – due Friday, September 20 (hard copy due in class)

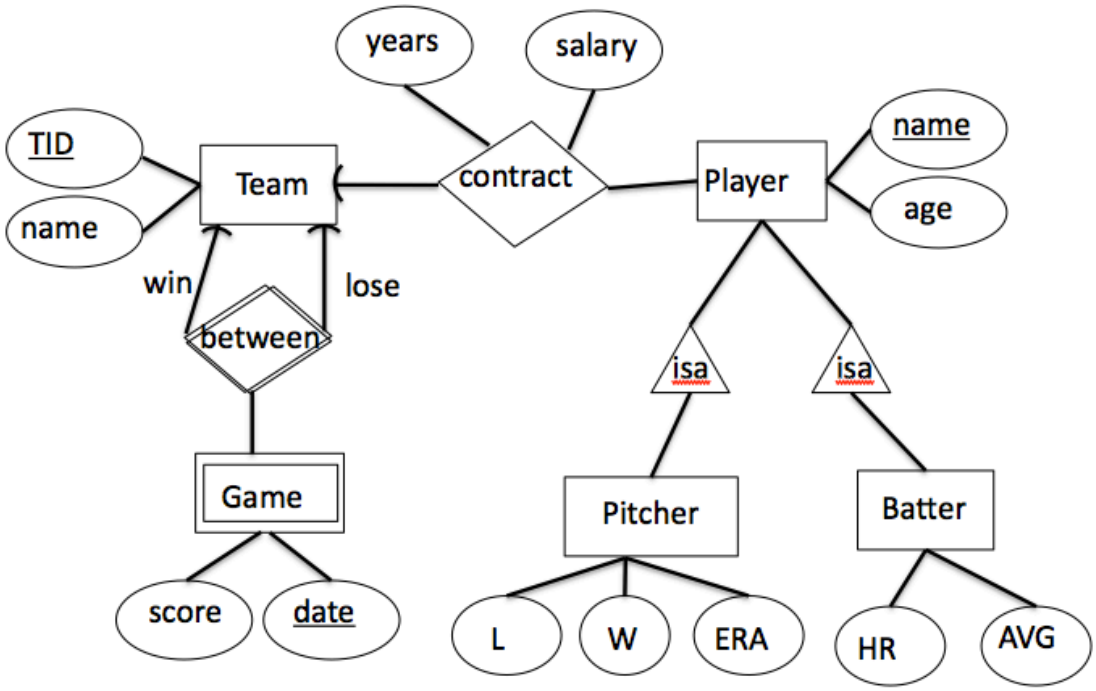
Collaboration policy*: This is an individual assignment. You are allowed to discuss the assignment with your colleagues, but your submission should reflect your own work. Sharing or copying code is not permitted. We reserve the right to divide the grade between any students who violate this policy. In addition, you must identify all those that you collaborate with on your assignment cover sheet. Consult your instructor if you need further clarification.*

1. a) Design an E/R diagram for a baseball database application described as follows:

   - Teams have a TID and a name
   - Players have a name and an age
   - Pitchers are a type of Players. Each pitcher has attributes W (win games), L (loss games), and ERA (earned run average)
   - Batters are a type of Players. Each batter has attributes AVG (batting average) and HR (home runs)
   - Each player plays for exactly one team; a team can have many players
   - Each player has exactly a contract relationship with his team, and a contract contains years and salary.
   - Games have a date (on which the game was played) and a score (e.g., "3:2")
   - A game has exactly a winning team and has exactly a losing team; A game can be uniquely identified by its winning team, losing team, and the date.

   If you feel that you must make any additional assumptions, please state them clearly in your solution. Remember to indicate the key for each entity, as well as the multiplicity of each relationship using the appropriate notation. (Same for the next exercise.)

   There could be several slightly different solutions for this problem, depending on everybody's assumptions. A sample solution is provided below.



Note that we used referential integrity as opposed to many-one to capture “exactly one” – however, we didn’t subtract points if you used a many-one relationship instead.

We could have included another “plays for” relationship between Team and Player – however, the corresponding relation would be a subset of the contract relation, therefore we omitted it.

We assumed that Player.name is unique and can be used as a key. Alternatively, we could have introduced one more unique Player.id attribute. Similarly, we assumed that TID is unique and can be used as a key for Team.

b) Translate the E/R diagram to a relational schema. Try to minimize the number of relations your solution has, and merge relations where appropriate. Don't forget to specify the keys. In translating a subclass hierarchy, use the E/R style translation.

```
Team(TID,name)
Game(WinTID, LoseTID, date, score)
     WinTID and LoseTID foreign keys referencing Team.TID

Player(name, age, TID, years, salary)
     TID foreign key referencing Team.TID
Pitcher(name, W, L, ERA)
     Name is also a foreign key referencing Player.name
Batter(name, HR, AVG)
     Name is also a foreign key referencing Player.name
```

Note: the above translation to relational schema ensures that the referential integrity (or many-one) constraints hold.

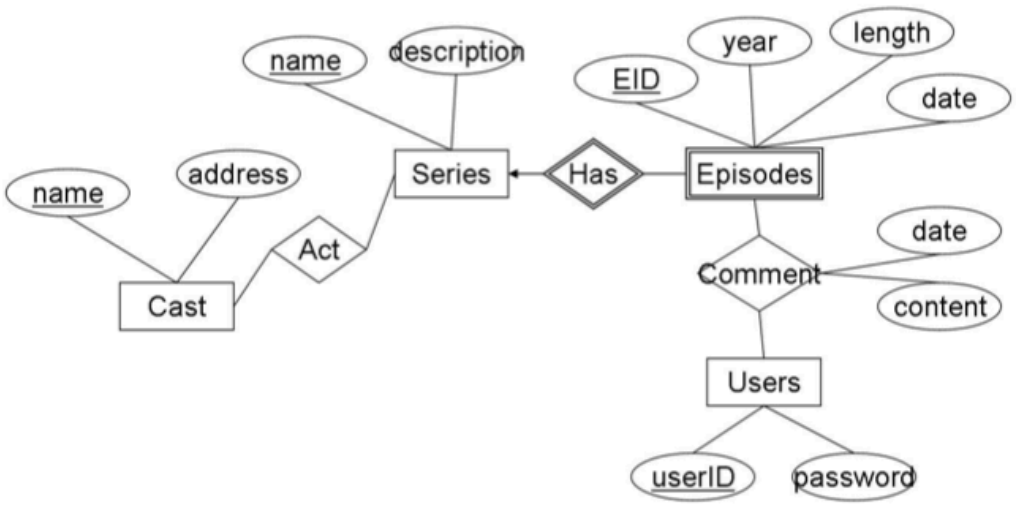
2. Your next task is to design a database for an online video service that offers hit TV series.

   a) As a first step, design an E/R diagram for your database. The following is the description of the application:

   - Each series has a name and a description.
   - We would like to record information on the regular cast of the series. In other words, for each main actor/actress in the series, we want to record his/her name and address. You may assume that the cast doesn't change.
   - Each series has many episodes. Each episode has an episode number, the year and date it was first aired, and the length of the episode in minutes. The episode number uniquely identifies an episode with respect to the series, but two different series can have the same episode number.
   - Registered viewers can comment on any episodes. For each comment, we want to record its post date and content.
   - For each registered viewer, we want to record his/her userID and password.

   Clearly specify any additional assumptions you might make.

There could be several slightly different solutions for this problem. A sample solution is provided below.



Assumptions:

- The name of each series is unique, so name is a key for Series.
- The name of actors/actresses is unique, so name is a key for Cast.
- userID is unique, so userID is a key for Users.
- Many actors/actresses can star in one series. One actor/actress can star in multiple different series. Hence, Act is many-many.
- Many users can comment on one episode. One user can comment on different episodes. Hence, Comment is many-many.

Possible variations: Comment could be modeled as an entity instead of a relation. Or we could have one entity for actors and one for actresses and these entities are ISA- related to Cast.

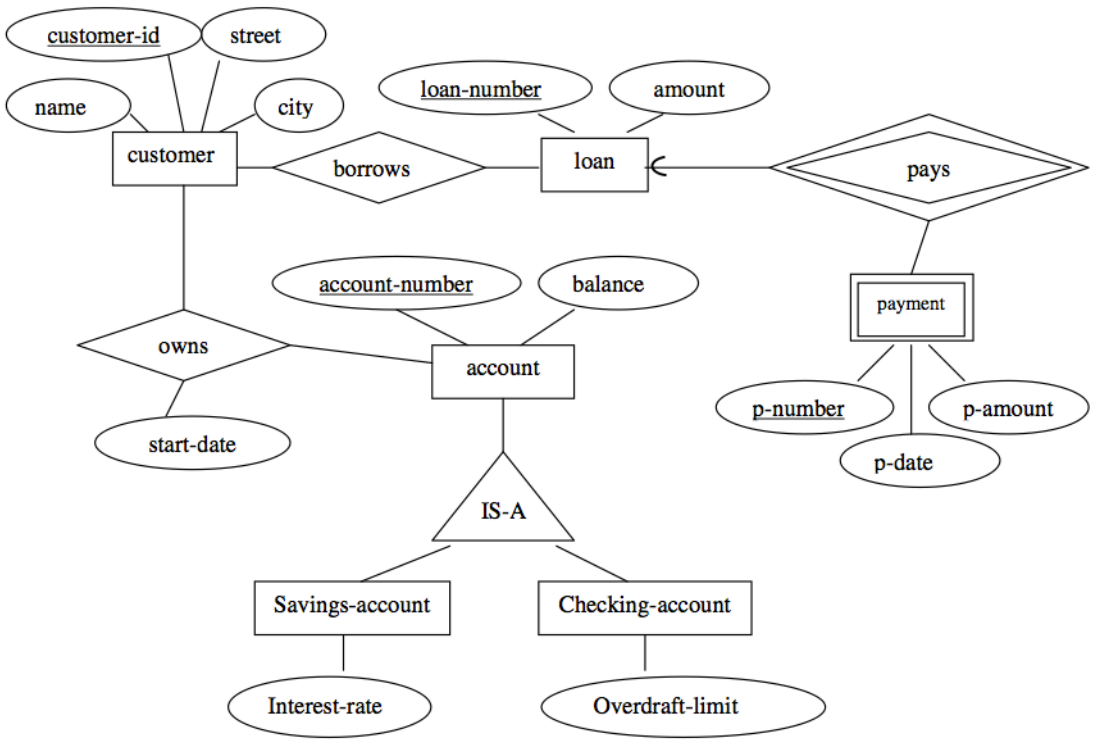
b) Next, translate your diagram to a relational schema. Try to minimize the number of relations your solution has, and merge relations where appropriate. Don't forget to specify the keys.

```
Series(name,description)
Cast(name,address)
Act(SeriesName,CastName)
   SeriesName is a foreign key referencing name in Series
   CastName is a foreign key referencing name in Cast

Episodes(EID,name,year,length,date)
   Name is a foreign key referencing Series
Users(userID,password)
```

```
Comment(name,EID,userID,date,content)
   Name is a foreign key referencing Episodes
   EID is a foreign key referencing Episodes
UserID is a foreign key referencing
```

3. Consider the following E/R diagram from a bank accounts application domain.



a) Describe in plain English the application modeled by the diagram.

The entities modeled in this application domain are: customers, loans, accounts and payments.

Customers are described by a customer_id (which is unique for a customer), name, street and city.

Loans are described by a unique loan_number and an amount.

A customer can have multiple loans and a loan could belong to more than one customer.

Loans are paid in successive payments. A payment has a number p_number, a date and an amount. The number p_number is unique for a loan, but different loans might have the same number for a payment. A payments has exactly one loan associated with it.

Accounts are generally described by a unique account number and a balance. Accounts could be Savings_accounts, in which case an interest_rate is also specified, or Checking_accounts, in which case an overdraft_limit is specified.

A customer can have multiple accounts and an account can be owned by multiple customers. The ownership relationship between customers and accounts has a date associated with it.

b) Translate the E/R diagram to a relational schema. Try to minimize the number of relations your solution has, and merge relations where appropriate. Don't forget to specify the keys. In translating a subclass hierarchy, use the E/R style translation.

```
customer(customer-id, name, street, city)
loan(loan-number, amount)
payment(p-number,loan-number, p-date, p-amount)
borrows(customer-id, loan-number)
account(account-number, balance)
owns(customer-id,account-number, start-date)
Savings-account(account-number, Interest-rate)
Checking-account(account-number, Overdraft-limit)
```

c) Write SQL "CREATE TABLE" statements that create the tables (relations) in your schema. Specify the necessary key, foreign key, NOT NULL and uniqueness constraints (justify your choices).

Given the bank domain, some attributes need to be specified, so they can't be NULL – for example, a customer can't open an account without providing name, street and city information; a loan should have a specified amount; a payment should have a date and an amount, etc.

```
CREATE TABLE Customer(
        customer_id INT PRIMARY KEY,
        name VARCHAR(30) NOT NULL,
        street VARCHAR(20) NOT NULL,
        city VARCHAR(30) NOT NULL)

CREATE TABLE Loan(
        loan_number INT PRIMARY KEY,
        amount DOUBLE NOT NULL)

CREATE TABLE Payment(
        loan_number INT REFERENCES Loan,
        p_number INT,
        p_date DATE NOT NULL,
        p_amount DOUBLE NOT NULL,
        PRIMARY KEY (loan_number, p_number))
```

```
CREATE TABLE Borrows(
        customer_id INT REFERENCES Customer,
        loan_number INT REFERENCES Loan,
        PRIMARY KEY (customer_id, loan_number))

CREATE TABLE Account(
        account_number INT PRIMARY KEY,
        balance DOUBLE NOT NULL)

CREATE TABLE Owns(
        customer_id INT REFERENCES Customer,
        account_number INT REFERENCE Account,
        start_date DATE NOT NULL,
        PRIMARY KEY (customer_is, account_number))

CREATE TABLE Savings_account(
        account_number INT PRIMARY KEY,
        interest_rate FLOAT NOT NULL,
        FOREIGN KEY (account_number) REFERENCE Account)

CREATE TABLE Checking_account(
        account_number INT PRIMARY KEY,
        overdraft_limit FLOAT NOT NULL,
        FOREIGN KEY (account_number) REFERENCE Account)
```