Week 1

Exercises for first two exercise sessions

2.1-2.3 (page 24-25)
3.1a+b (page 32-33).

1.1-1.3 (page 134-135)
3.1.b (page 147)
2.1 (page 141-142)
6.1 (page 167)

Databases in general

Relation: Two dimensional table, rows represent an element, column a property

Attribute: Colum's of a relation, describes meaning of entries.

Schema: name of relation + set of attributes. NAME(attr, attr…)

Database schema: set of relation schemas.

Tuple: Rows of a relation, has 1 component for each attribute

Component: mapping from value to attribute

Domain: the datatype of an attribute

Relation instance: the given data in the relation, changes dynamically.

ER diagram

Entity set:

- Represented by rectangle
- Entity: abstract object of sorts
- Set of these form entity set
- Reassembles a class in OOP (without methods)

Attributes:

- Represented by ellipse
- Connected to entity set
- Describes the entity

Relationships:

- Represented by diamonds
- How Entity sets are related.

Subclasses:

- Represented by triangle
- Thriller ISA Movie, meaning same attributes + possible more

Multiplicity of Relationship (binary)

- Many – Many: many A has many B
- Many – One: many A has one B
- One – One: one A has one B

Referential integrity:

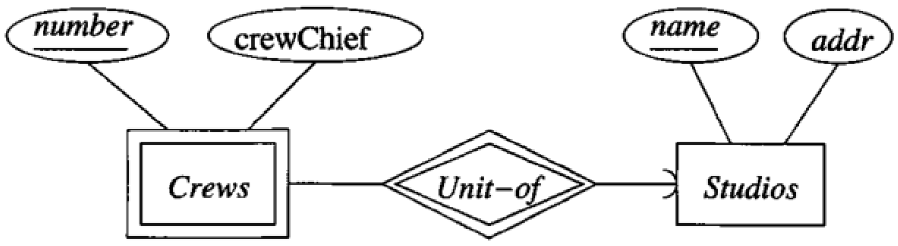
- Represented by rounded arrow.
- Indicates that the relationship must exist.

Degree constraint:

- Bounding number of relations.

Weak entity Set:

- Entity whose key is composed of attributes, some/all belonging to other entity set
- Could be due to hierarchical reasons. (not ISA)
- Eg:



  - Crew identified by crew number and studio.
    - Studios are using same numbering of crews.
- Requirements:
  - Key consist of zero or more of own + attributes from entity sets reached by many-one relationships (supporting relationships)
  - Referential integrity to supporting entity
  - Supporting attributes must be keys in supporting relation
  - Supporting entity could be weak itself
- Weak entity set represented by double rectangle
- Supporting relationship represented by double diamond

Design principles

Faithfulness:

- Should reflect reality
- Attributes should describe

Avoid redundancy:

- Same attribute on multiple entities
- Uses more space and update anomaly

Simplicity counts:

- Do not introduce more elements than necessary

Choosing right relationships:

- Otherwise lead to redundancy, update anomalies, deletion anomalies

Choosing right element

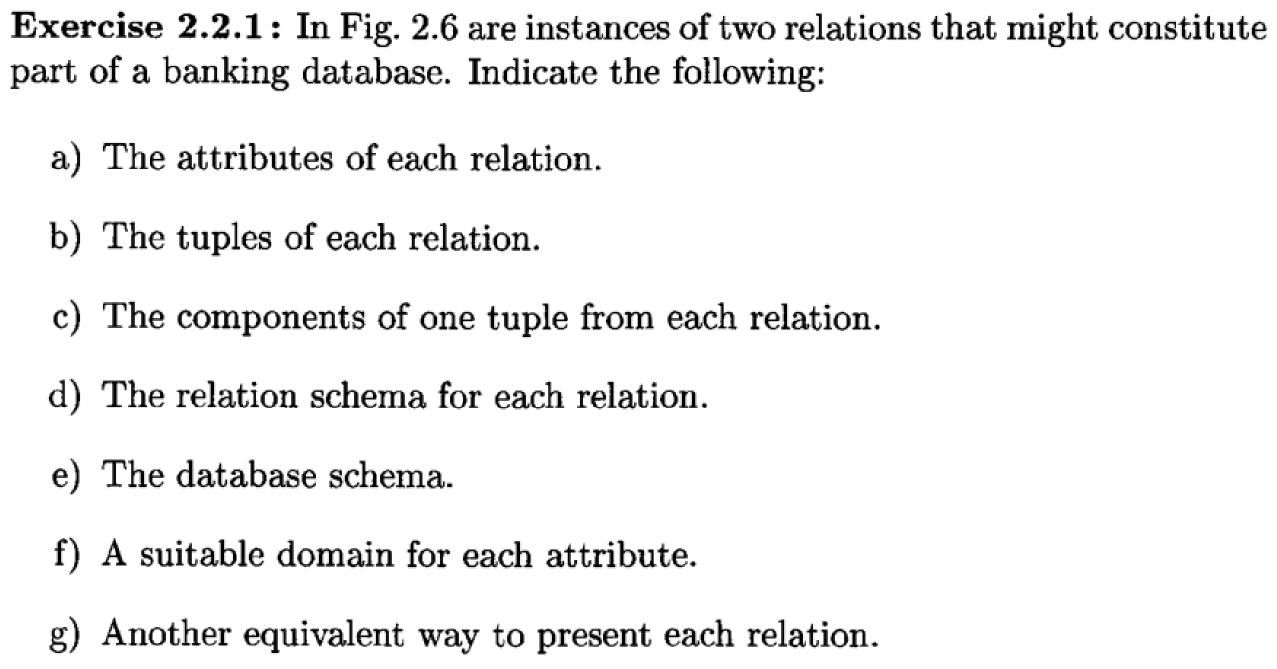
- Choose attribute or entity set/relationship combination

Converting E/R diagrams to Relation Designs

- Entity set to relation with same attributes/keys
- Relationship to relation with keys for connected entity sets
    - Adding attributes on relation
- Week entity sets:
    - Weak entity set should include parts of key in supporting entities
    - Any relationship connected to the weak relationship should include parts of key in supporting entities
    - Supporting relationship do not need to be converted
        - General: the attributes of many-one relationship R's relation will either be attributes of the relation for W, or (in the case of attributes on R) can be added to the schema for W's relation.
- Subclasses
    - E/R Style
        - For each entity, create relation which includes key attributes from root and all of its own attributes
    - Object oriented approach
        - For each possible subtree create relation which includes all attributes in the subtree
    - Null approach
        - Create one relation containing all attributes
        - Have null values for attributes it does not have

Chapter 2

2.1



Exercise 2.2.1: In Fig. 2.6 are instances of two relations that might constitute part of a banking database. Indicate the following:

a) The attributes of each relation.

b) The tuples of each relation.

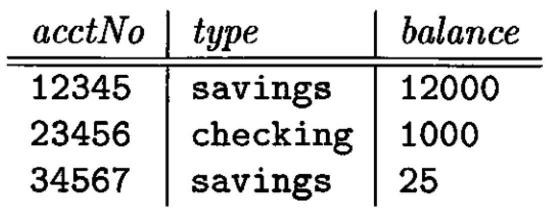
c) The components of one tuple from each relation.

d) The relation schema for each relation.

e) The database schema.

f) A suitable domain for each attribute.

g) Another equivalent way to present each relation.



acctNo	*type*	*balance*
12345	savings	12000
23456	checking	1000
34567	savings	25

The relation Accounts

Attributes: acctNo, type, balance

Tuples: (12345, savings, 12000), (23456, checkings, 1000), (34567, savings, 25)

Component: $12345 \rightarrow acctNo, checkings \rightarrow type, 1000 \rightarrow balance$

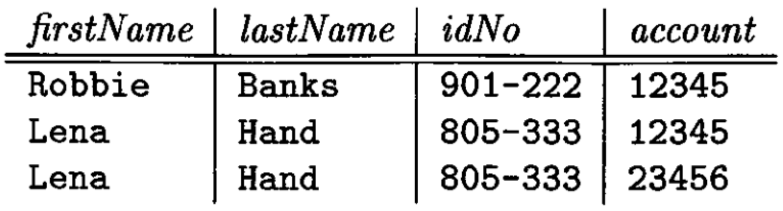
(”A tuple has one component for each attribute of the relation”)

Schema: Accounts(acctNo, type, balance)

Domains: Accounts(acctNo:integer, type:string, balance:integer)

Equivalent representation: Accounts(type, balance, acctNo)

We can swap the order of the attributes…



firstName	*lastName*	*idNo*	*account*
Robbie	Banks	901-222	12345
Lena	Hand	805-333	12345
Lena	Hand	805-333	23456

The relation **Customers**

Attributes: firstName, lastName, idNo, account

Tuples: $(Robbie, Banks, 901 - 222, 12345)$, $(Lena, Hand, 805 - 333, 12345)$, $(Lena, Hand, 805 - 333, 23456)$

Component: $Robbie \rightarrow firstName$, $Banks \rightarrow lastName$, $901 - 222 \rightarrow idNo$, $12345 \rightarrow account$

(”A tuple has one component for each attribute of the relation”)

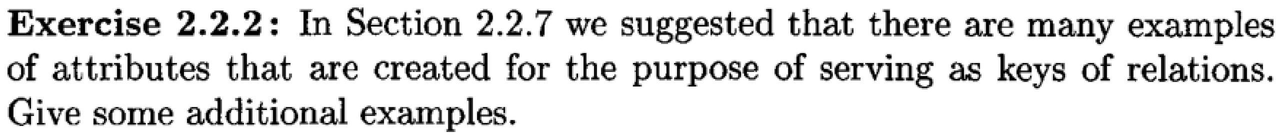
Schema: Customers(firstName, lastName, idNo, account)

Domains: Customers(firstName: string, lastName:string, idNo:string, account:integer)

Equivalent representation: Customers(lastName, idNo, account, firstName)

We can swap the order of the attributes…

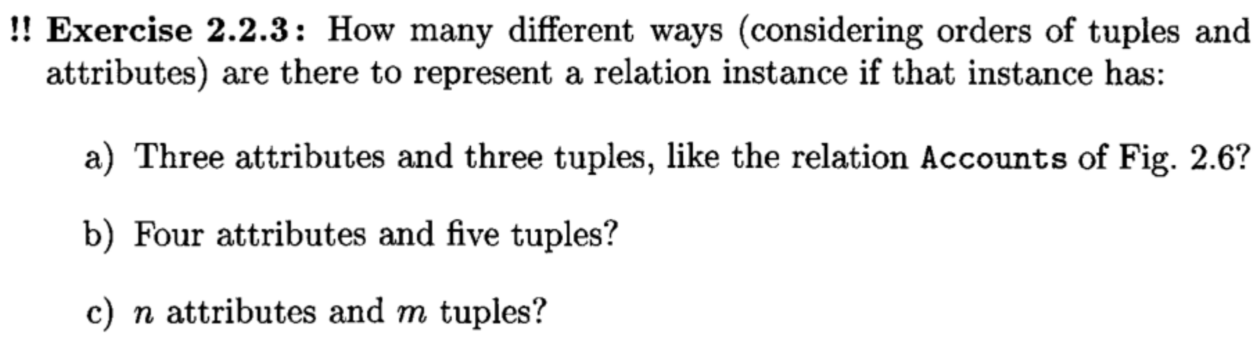
2.2



Exercise 2.2.2: In Section 2.2.7 we suggested that there are many examples of attributes that are created for the purpose of serving as keys of relations. Give some additional examples.

CPR, student id, product number, EAN, ISBN

2.3



!! Exercise 2.2.3: How many different ways (considering orders of tuples and attributes) are there to represent a relation instance if that instance has:

a) Three attributes and three tuples, like the relation Accounts of Fig. 2.6?

b) Four attributes and five tuples?

c) n attributes and m tuples?

a.

$$3! \cdot 3! = 36$$

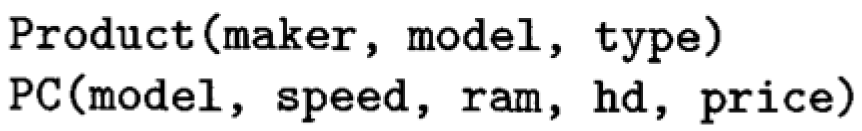
b.

$$4! \cdot 5! = \cdots$$

c.

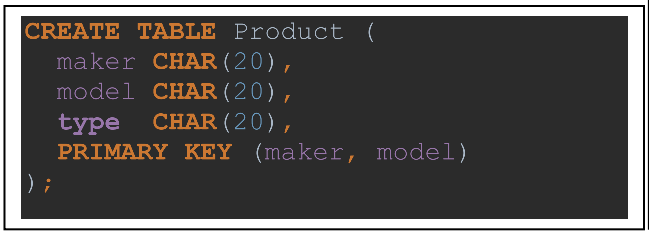
$$n! \cdot m!$$

3.1 a,b

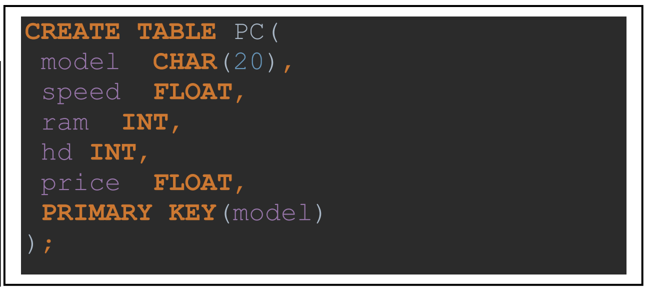


```
Product(maker, model, type)
PC(model, speed, ram, hd, price)
```

Creating the tables:



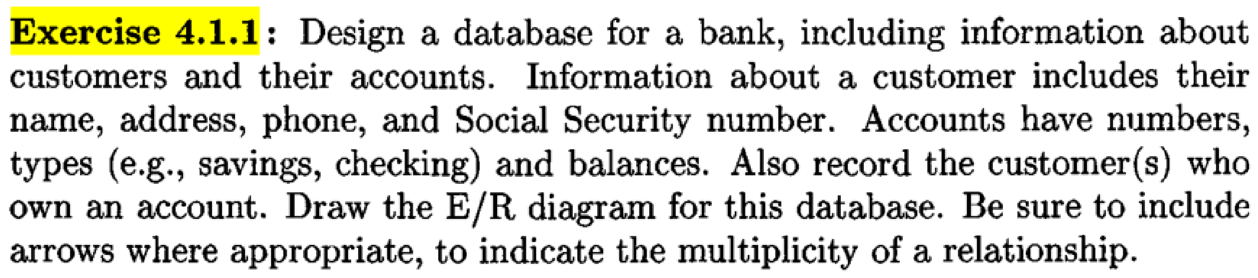
```
CREATE TABLE Product (
  maker CHAR(20),
  model CHAR(20),
  type  CHAR(20),
  PRIMARY KEY (maker, model)
);
```



```
CREATE TABLE PC(
 model  CHAR(20),
 speed  FLOAT,
 ram  INT,
 hd INT,
 price  FLOAT,
 PRIMARY KEY(model)
);
```

Chapter 4

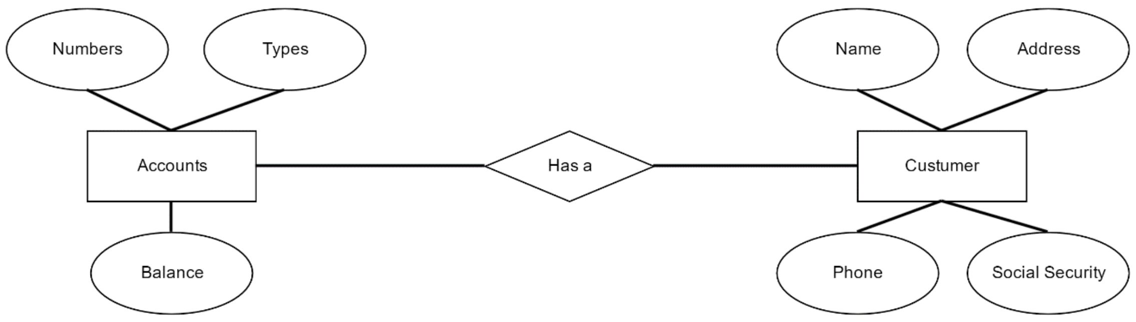
1.1



Exercise 4.1.1: Design a database for a bank, including information about customers and their accounts. Information about a customer includes their name, address, phone, and Social Security number. Accounts have numbers, types (e.g., savings, checking) and balances. Also record the customer(s) who own an account. Draw the E/R diagram for this database. Be sure to include arrows where appropriate, to indicate the multiplicity of a relationship.

Costumers: Name, Address, phone, Social Security

Accounts: Numbers, types, balances



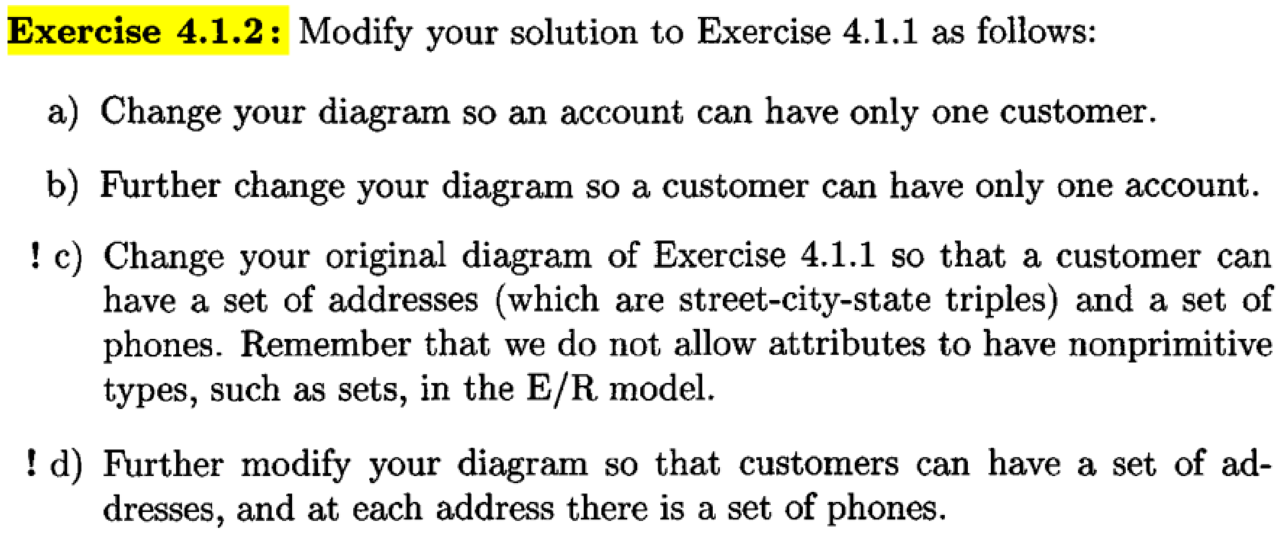
Maybe better name for relationship.

Keys would be number for accounts og social security for customer

Convert into relational database:

- Accounts(Numbers, Types, Balance)
- Customer(Name, Address, Phone, Social Security)
- Account_Customer(number, social security)

1.2



Exercise 4.1.2: Modify your solution to Exercise 4.1.1 as follows:

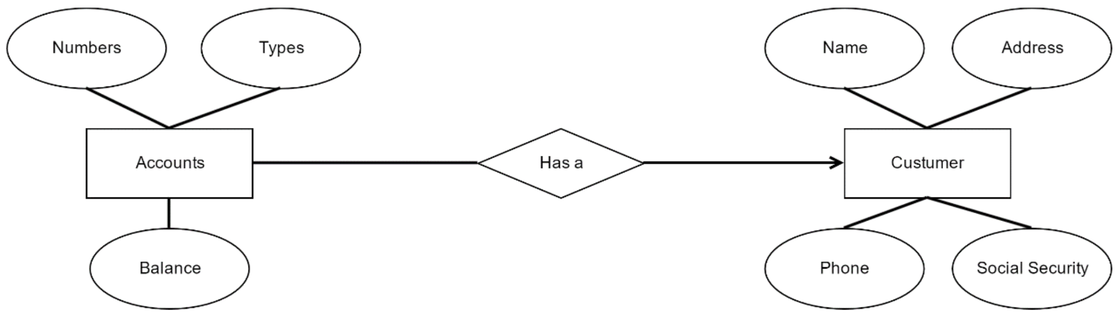
a) Change your diagram so an account can have only one customer.

b) Further change your diagram so a customer can have only one account.

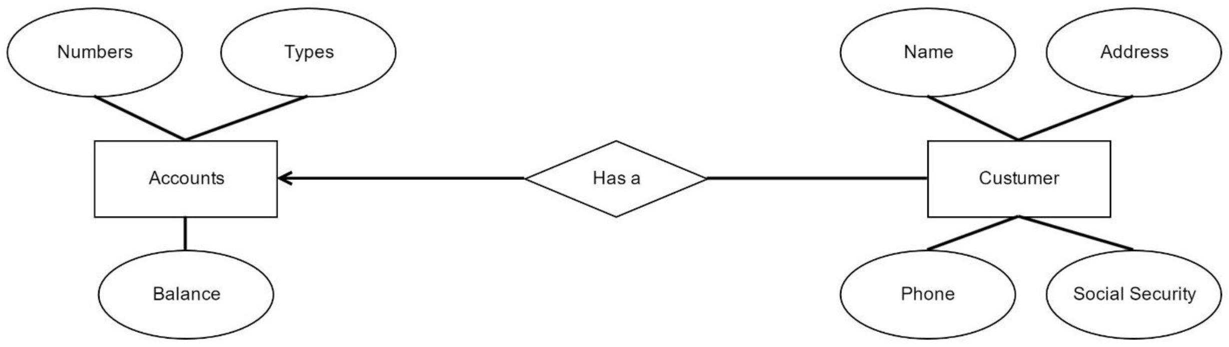
! c) Change your original diagram of Exercise 4.1.1 so that a customer can have a set of addresses (which are street-city-state triples) and a set of phones. Remember that we do not allow attributes to have nonprimitive types, such as sets, in the E/R model.

! d) Further modify your diagram so that customers can have a set of addresses, and at each address there is a set of phones.

a.

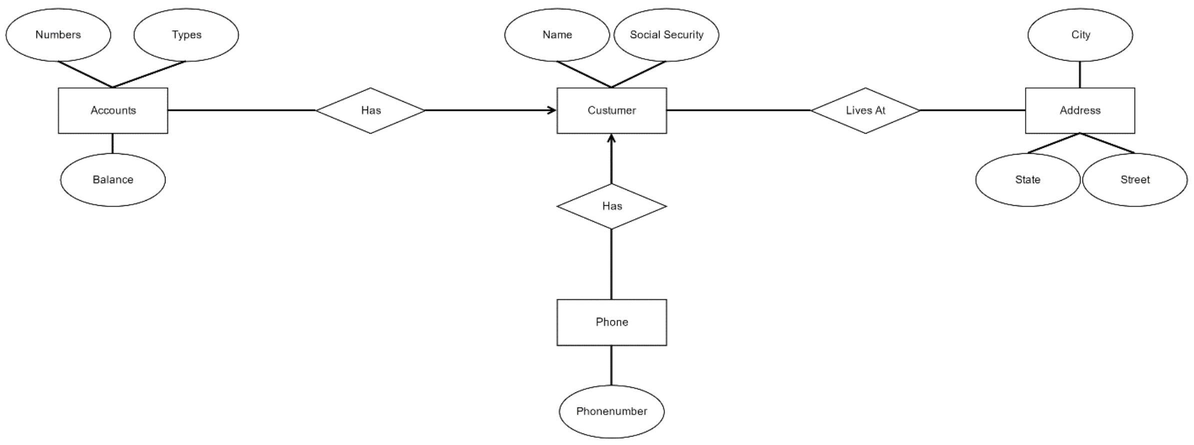


b.

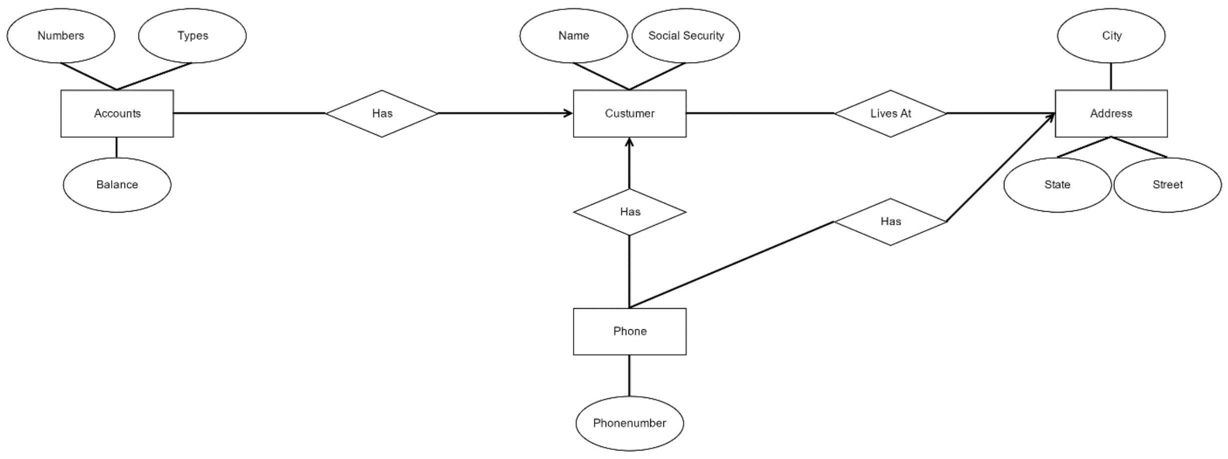


Obs. Should be arrows in both directions due to continue modifying diagram.

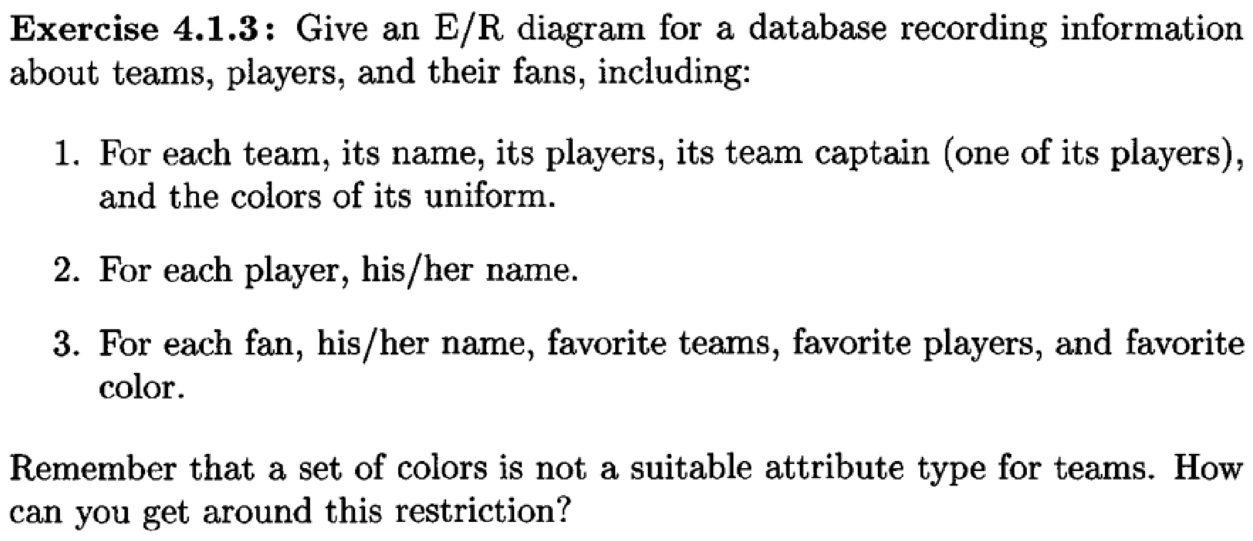
c. Introducing address and phone:



d.



1.3

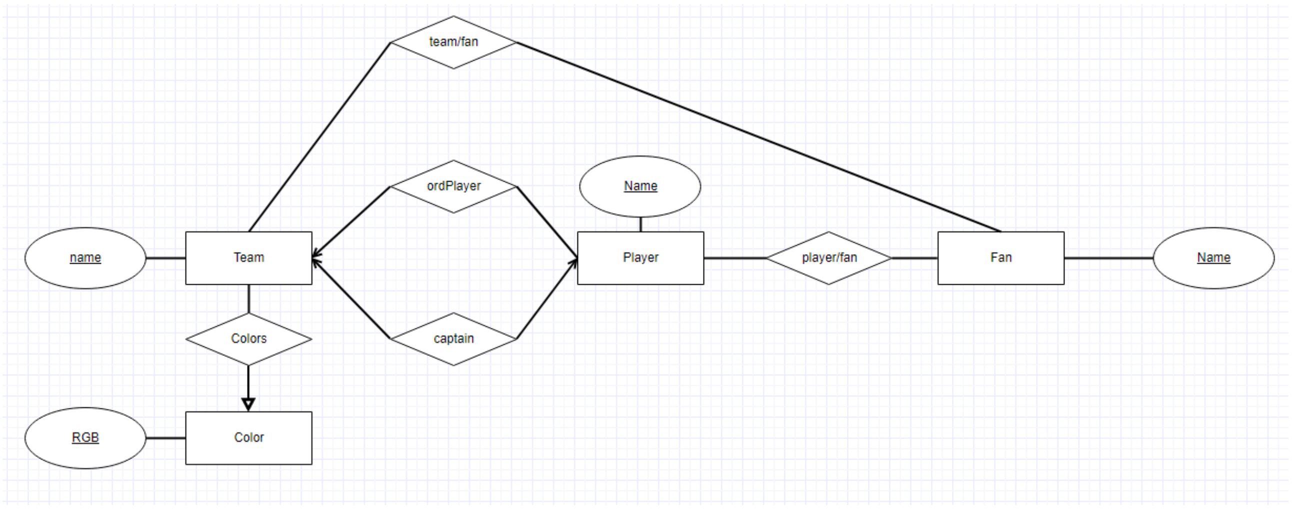

Exercise 4.1.3: Give an E/R diagram for a database recording information about teams, players, and their fans, including:

1. For each team, its name, its players, its team captain (one of its players), and the colors of its uniform.

2. For each player, his/her name.

3. For each fan, his/her name, favorite teams, favorite players, and favorite color.

Remember that a set of colors is not a suitable attribute type for teams. How can you get around this restriction?

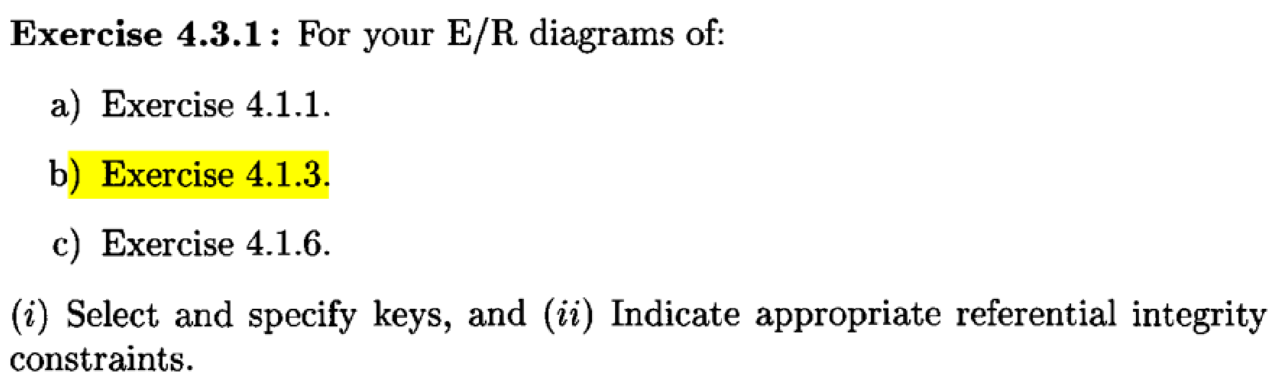


OBS.: Missing relationship between fan and color. Could combine the Player and Fan to one entity. Integrity constrains are shown with the open arrow, many-one with closed unfilled (not convention of book or Kudahl, sorry…)

Convert into relational database:

- Team(name)
- Color(RGB)
- Team_Color(name, RGB)
- Player(name)
- Fan(name)
- Player_fan(player_name, fan_name)
- Team_fan(team_name, fan_name)

3.1 b



Exercise 4.3.1: For your E/R diagrams of:

a) Exercise 4.1.1.

b) Exercise 4.1.3.

c) Exercise 4.1.6.

(*i*) Select and specify keys, and (*ii*) Indicate appropriate referential integrity constraints.

Keys are specified in ER diagram above…

Maybe name of player + fan not unique add social security number.

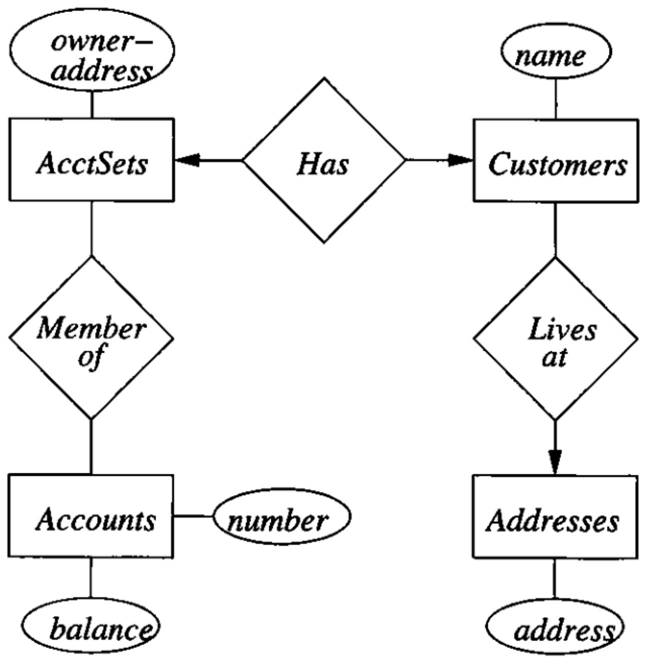
Team, maybe key name + sport.

Referential constraints:

- Team must have 1 captain
- Player must belong to 1 team

2.1

Design of bank system with customers and accounts.



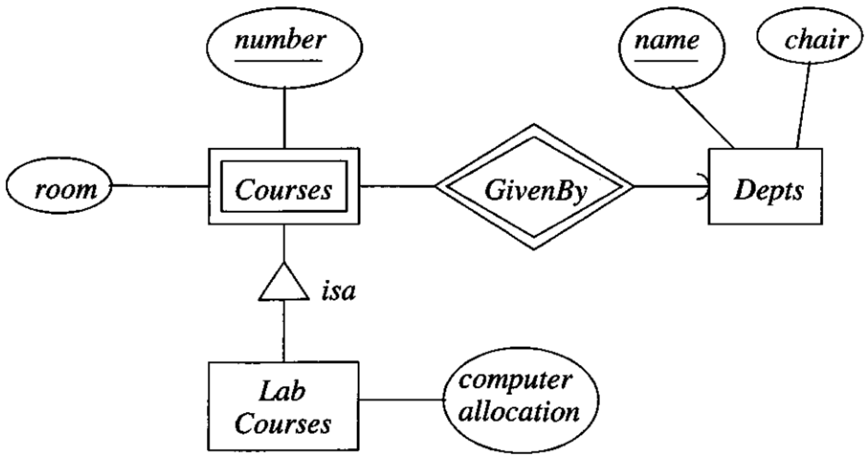
Design rule violations:

- Simplicity:

  - Could just have been made as a many-many relationships between accounts and customers.
- Redundancy:
  - OwnerAddress is redundant.
- Choosing right element
  - Address should just have been an attribute of customer
- Missing info on customer to get key

6.1

Convert following into relation database schema



a. Straight E/R method:

Courses(<u>number</u>, <u>name</u>, room)

Lab Cources(<u>number</u>, <u>name</u>, computer allocation)

Depts(<u>name</u>, chair)

A lab course will be in both lab cources and in the root.

b. Object-oriented method:

Courses(<u>number</u>, <u>name</u>, room)

CourcesLabCources(<u>number</u>, <u>name</u>, room, computer allocation)

Depts(name, chair)

A lab course will only be in coursesLabCourses, but not in the root.

c. Nulls method:

Courses(<u>number</u>, <u>name</u>, room, computer allocation)

Depts(<u>name</u>, chair)