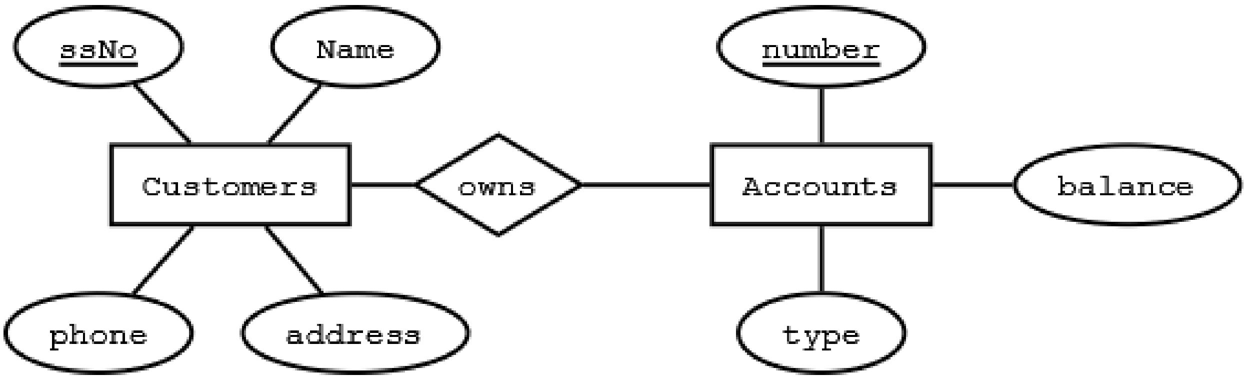
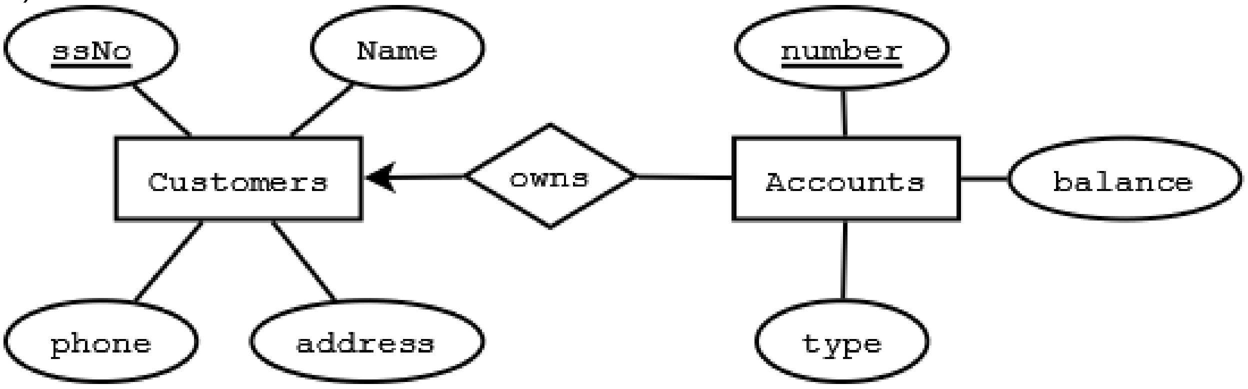
Solutions
Chapter 4

4.1.1

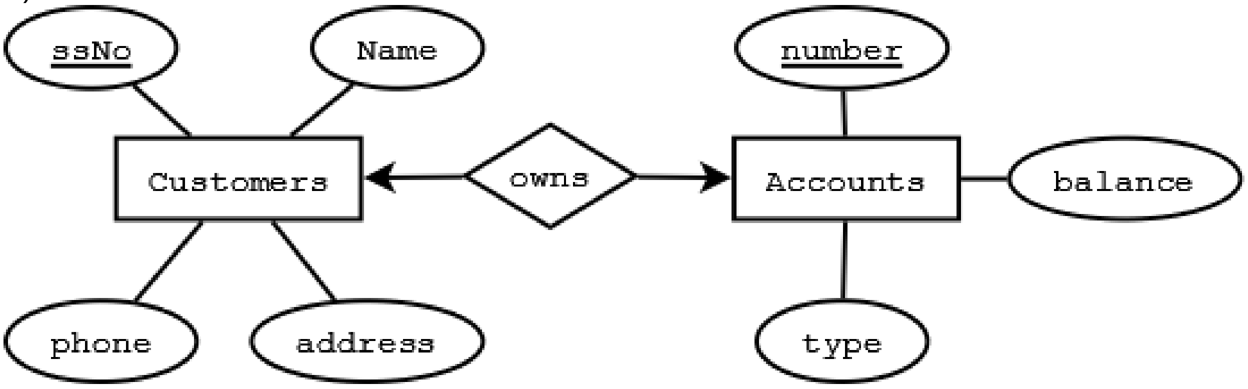


4.1.2

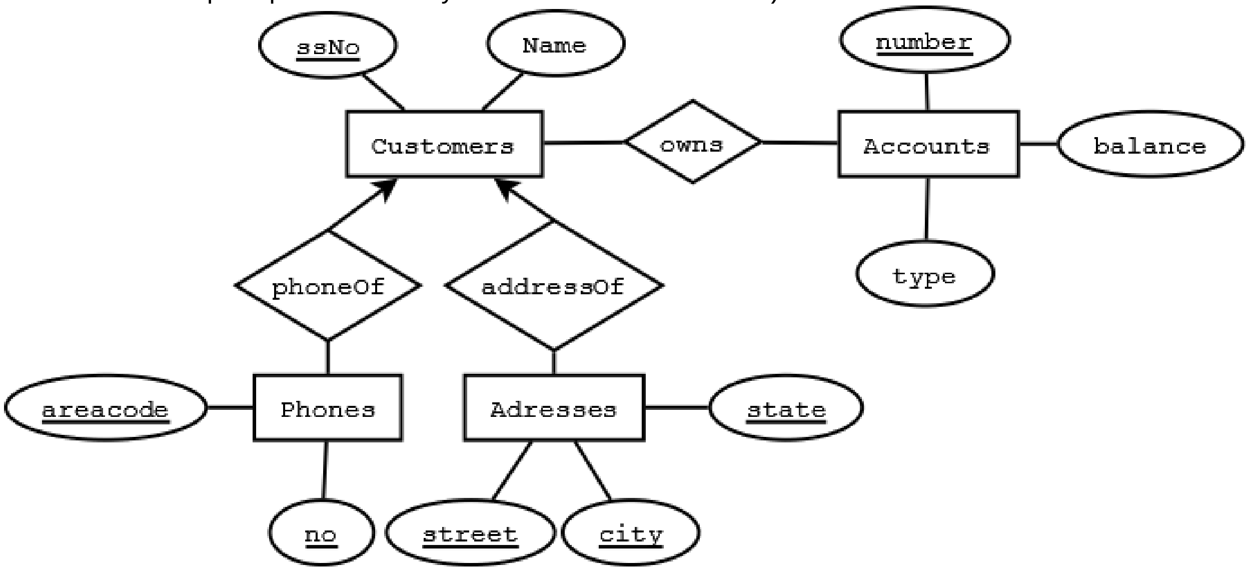
a)



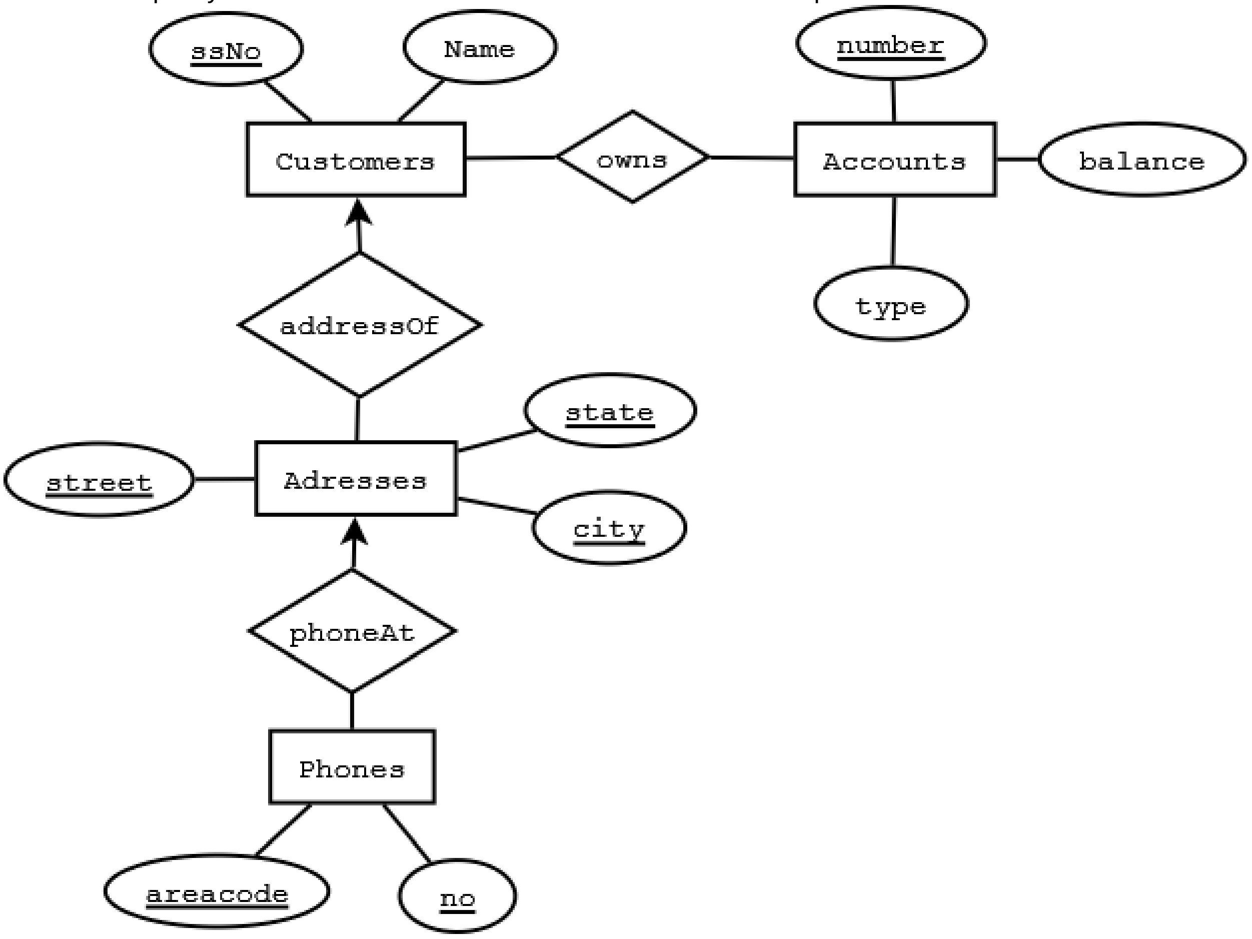
b)



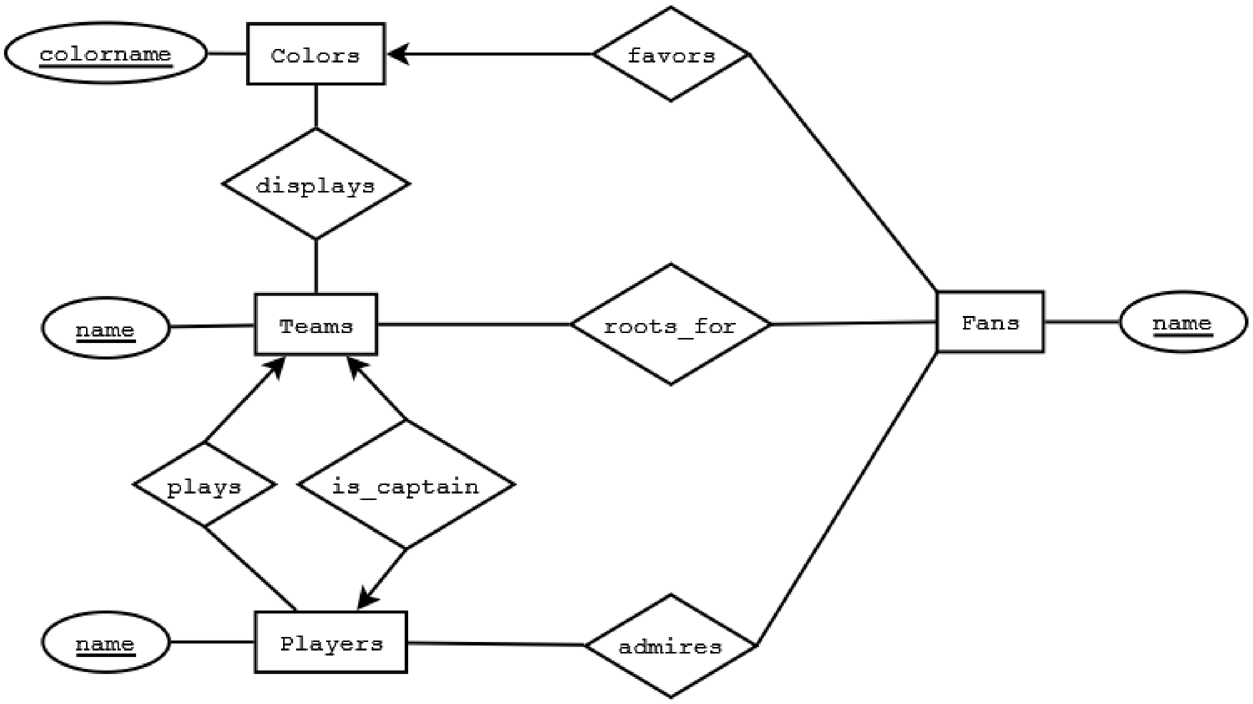
c)
In c we assume that a phone and address can only belong to a single customer (1-m relationship represented by arrow into customer).



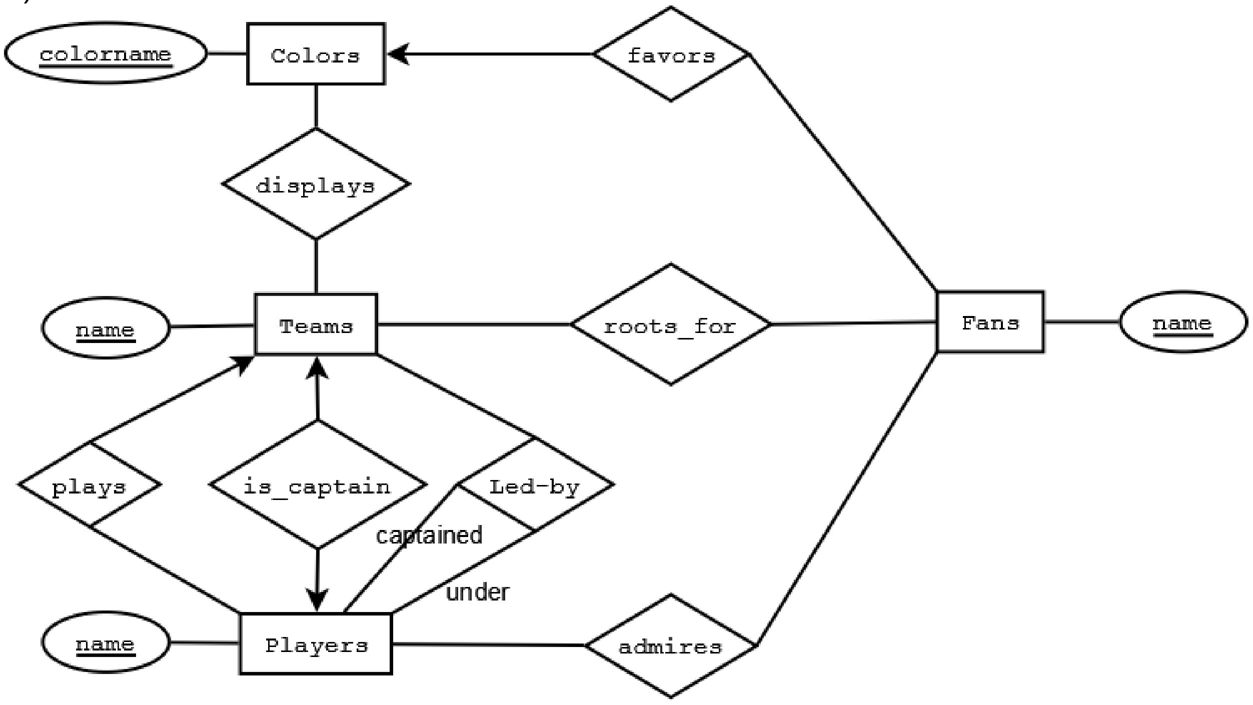
d)
In d we assume that an address can only belong to one customer and a phone can exist at only one address.
If the multiplicity of above relationships were m-to-n, the entity set becomes weak and the key ssNo of customers will be needed as part of the composite key of the entity set.
In c&d, we convert attributes phones and addresses to entity sets. Since entity sets often become relations in relational design,
we must consider more efficient alternatives.
Instead of querying multiple tables where key values are duplicated, we can also modify attributes:
(i) Phones attribute can be converted into HomePhone, OfficePhone and CellPhone.
(ii) A multivalued attribute such as alias can be kept as an attribute where a single column can be used in relational design i.e. concatenate all values. SQL allows a query "like '%Junius%'" to search the multiple values in a column alias.



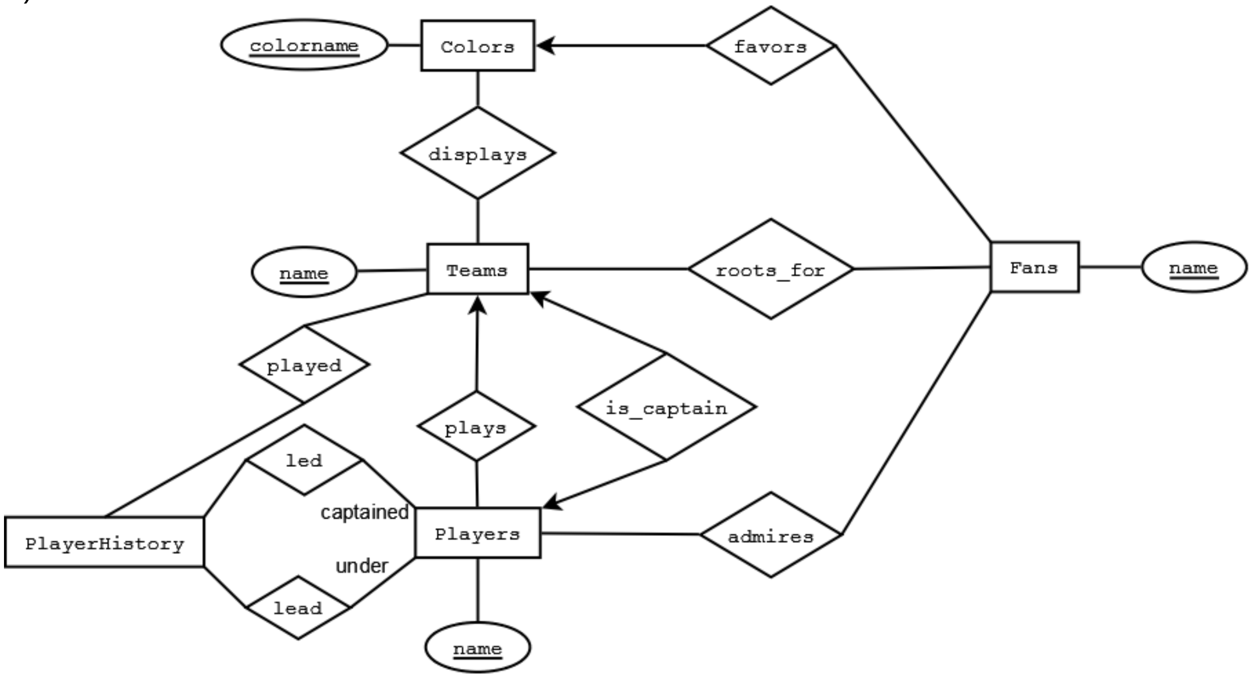
4.1.3


colorname
Colors
favors
displays
name
Teams
roots_for
Fans
name
plays
is_captain
name
Players
admires

4.1.4
a)

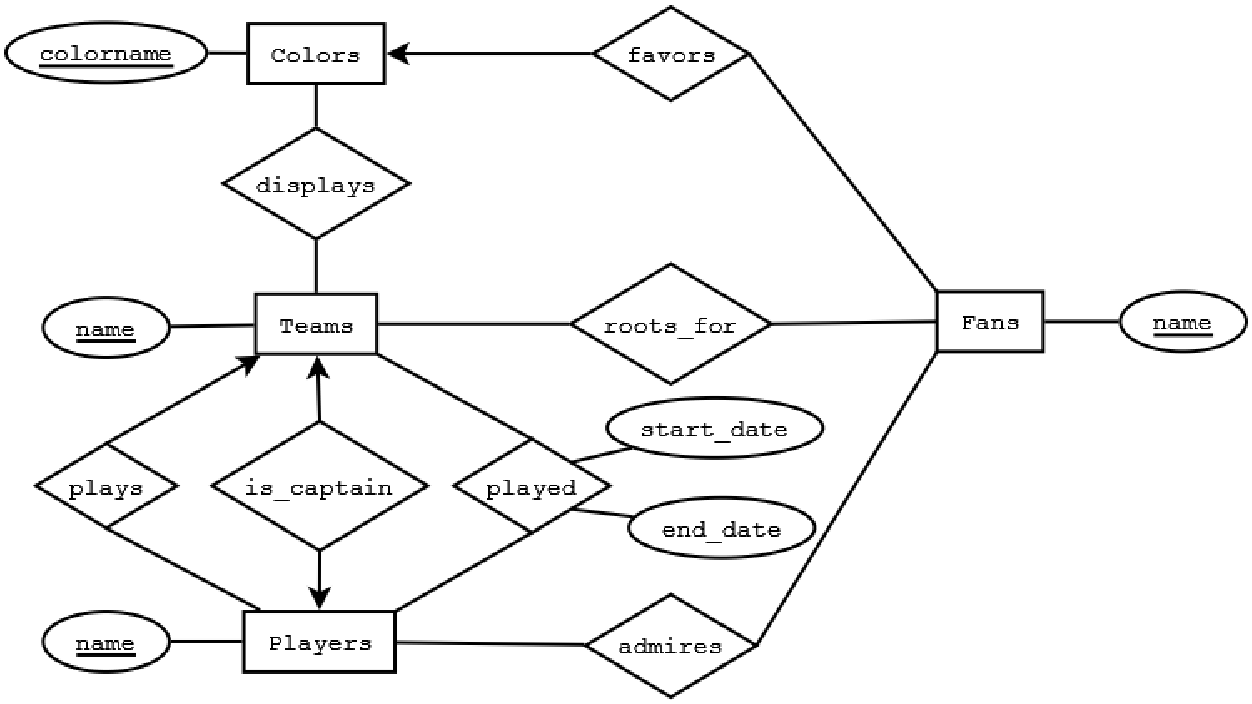

colorname
Colors
favors
displays
name
Teams
roots_for
Fans
name
plays
is_captain
Led-by
captained
under
name
Players
admires

b)

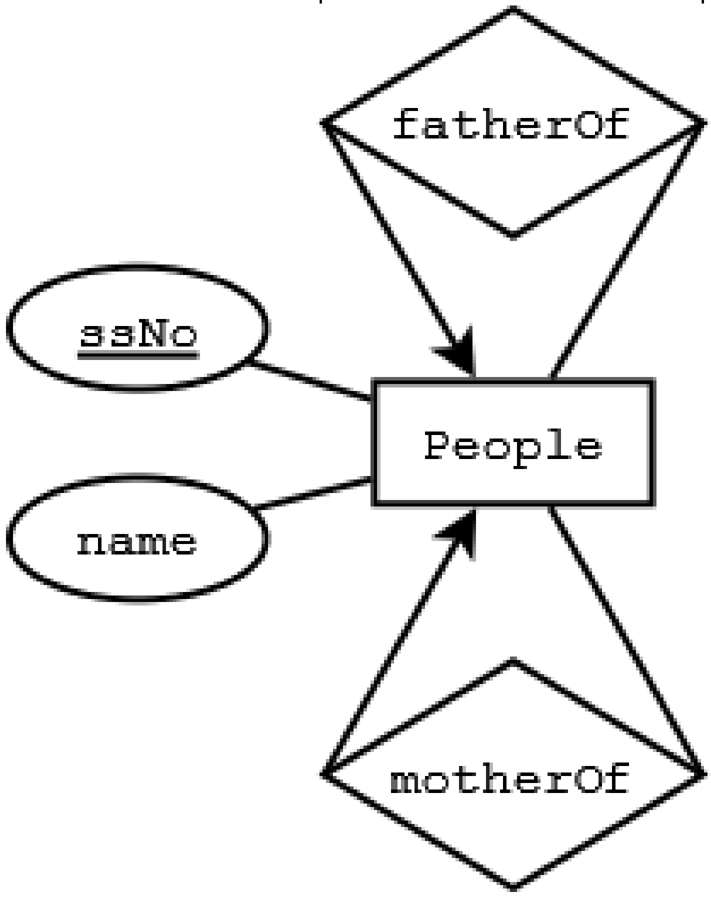


c)
The relationship "played" between Teams and Players is similar to relationship "plays" between Teams and Players.

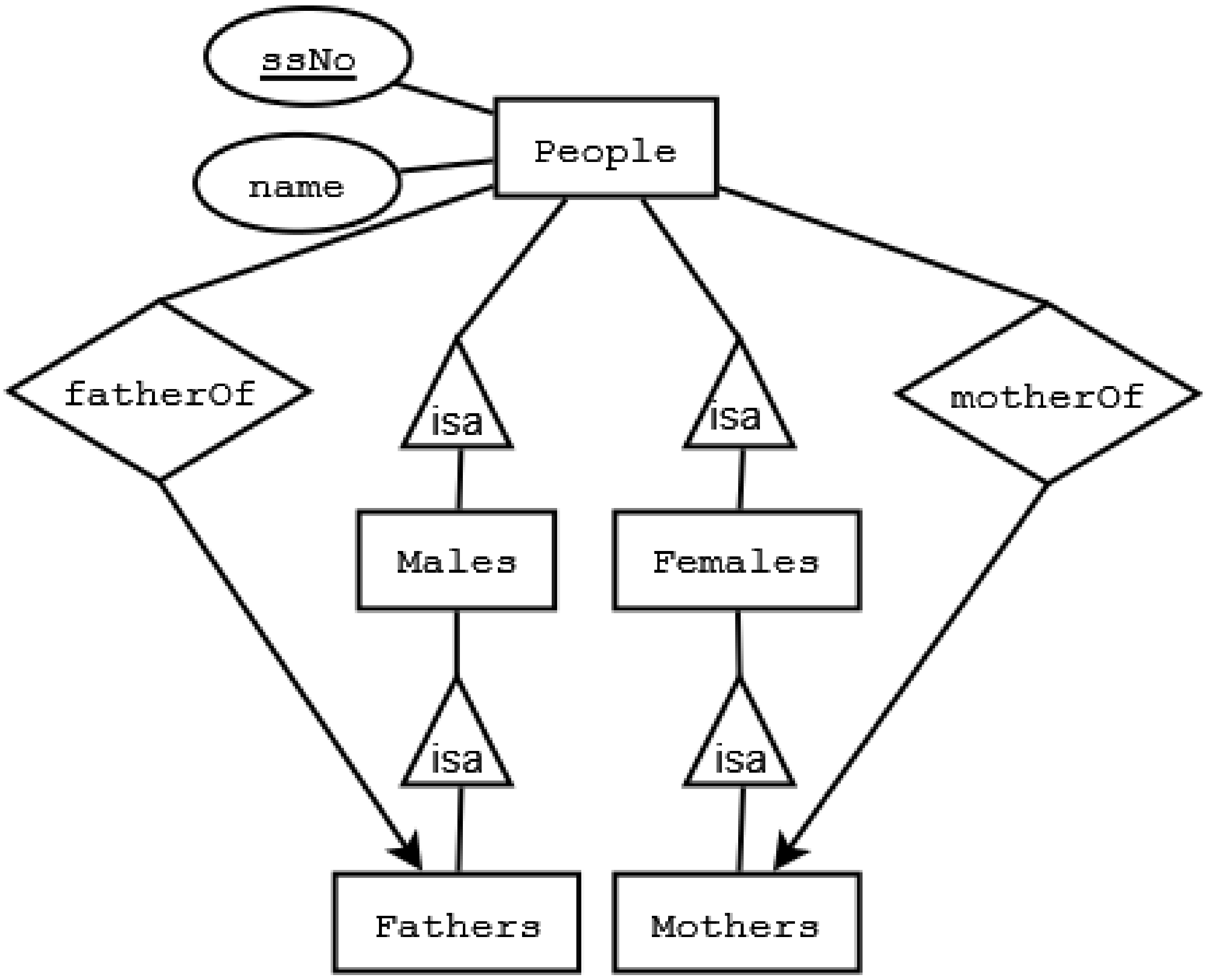
4.1.5



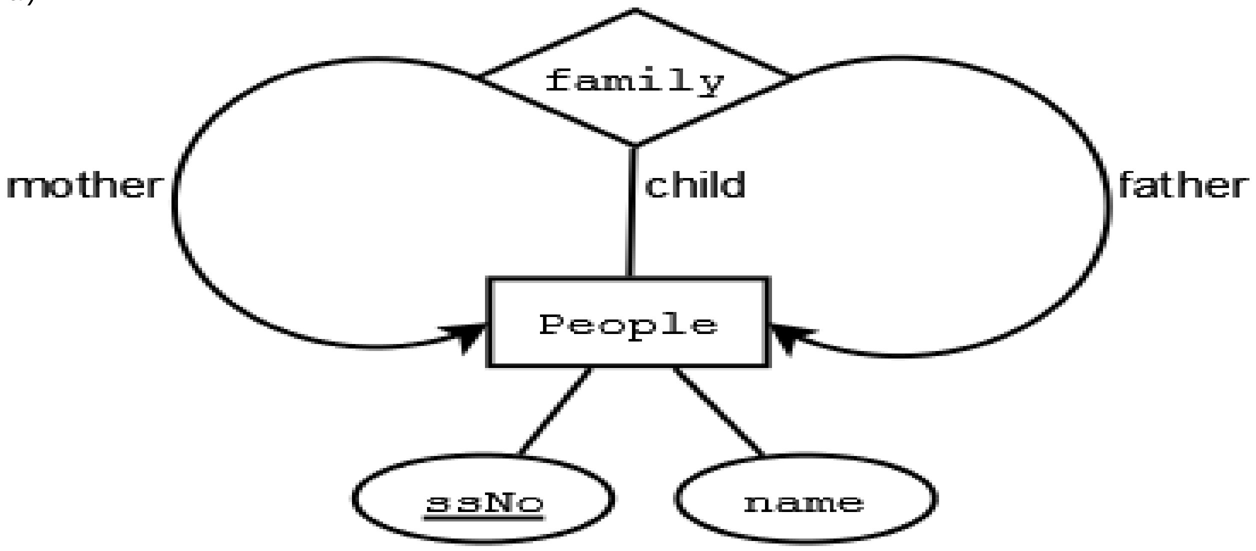
4.1.6 The information about children can be ascertained from motherOf and fatherOf relationships. Attribute ssNo is required since names are not unique.



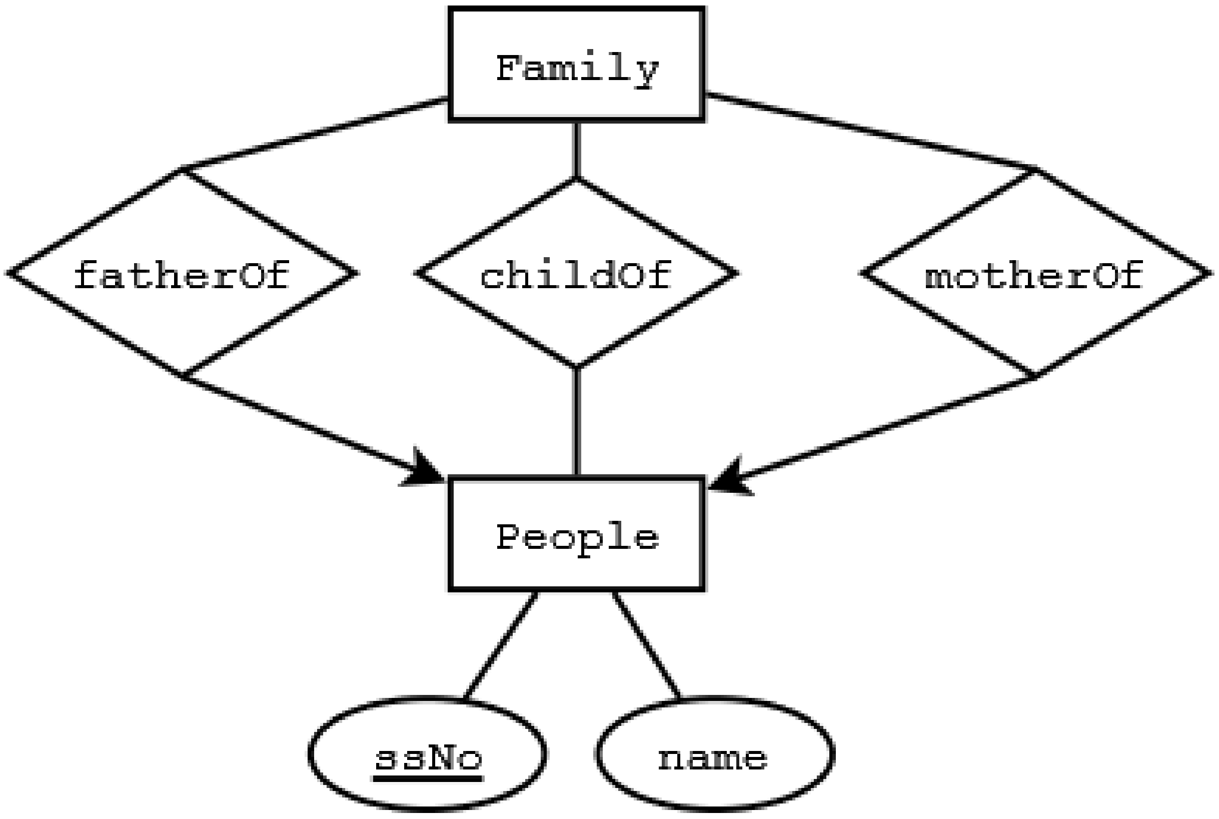
4.1.7


ssNo
name
People
fatherOf
isa
isa
motherOf
Males
Females
isa
isa
Fathers
Mothers

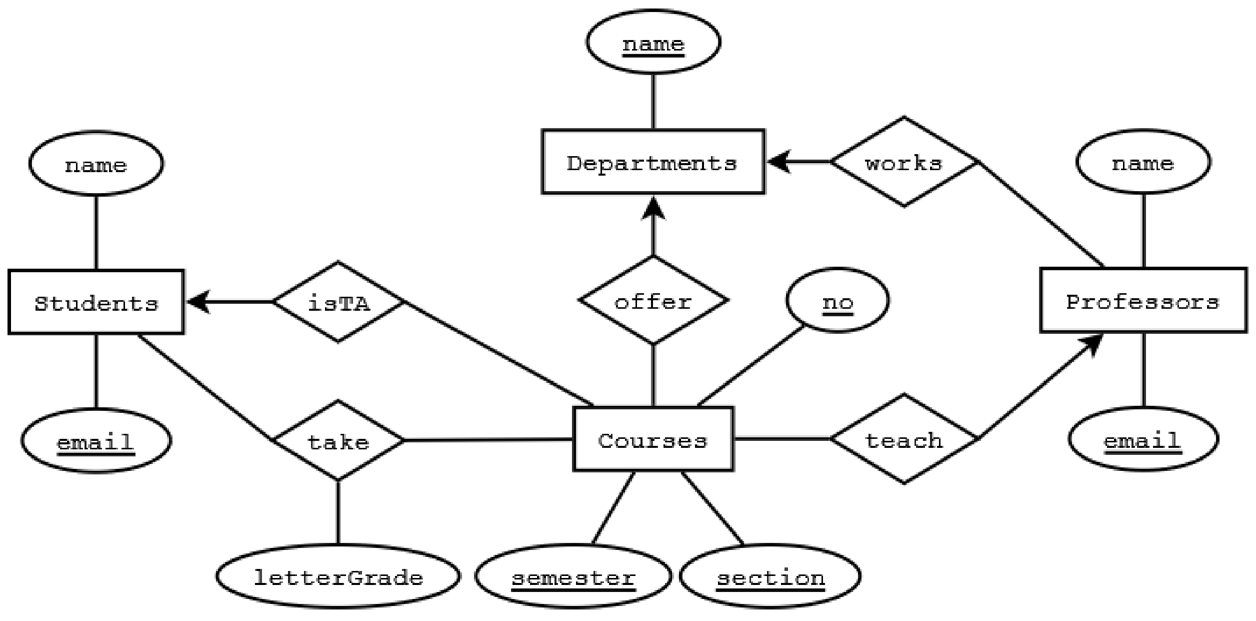
4.1.8
a)



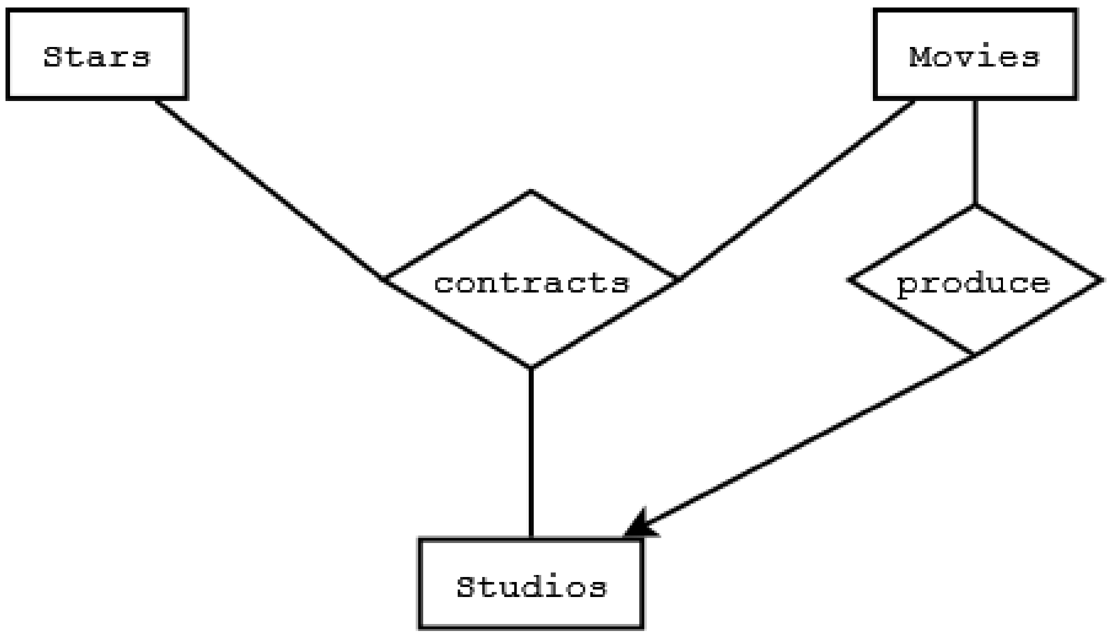
(b)

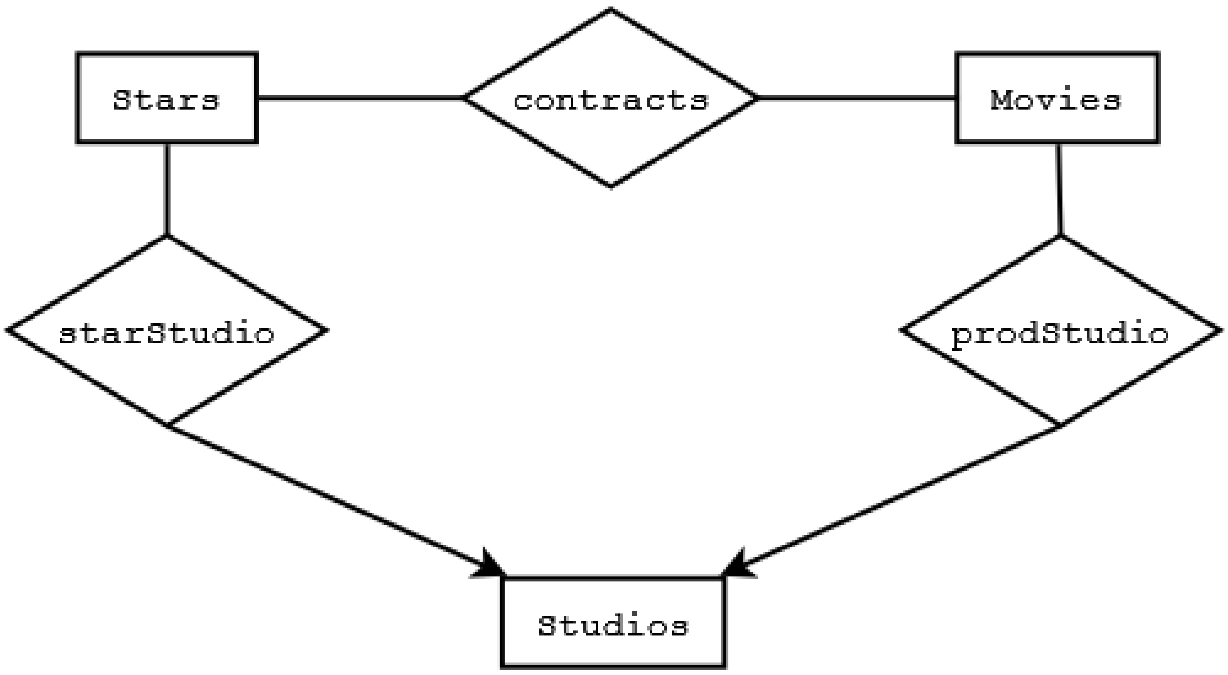


4.1.9
Assumptions
A Professor only works in at most one department.
A course has at most one TA.
A course is only taught by one professor and offered by one department.
Students and professors have been assigned unique email ids.
A course is uniquely identified by the course no, section no, and semester (e.g. cs157-3 spring 09).



4.1.10
Given that for each movie, a unique studio exists that produces the movie. Each star is contracted to at most one studio.
But stars could be unemployed at a given time. Thus the four-way relationship in fig 4.6 can be easily into converted equivalent relationships.

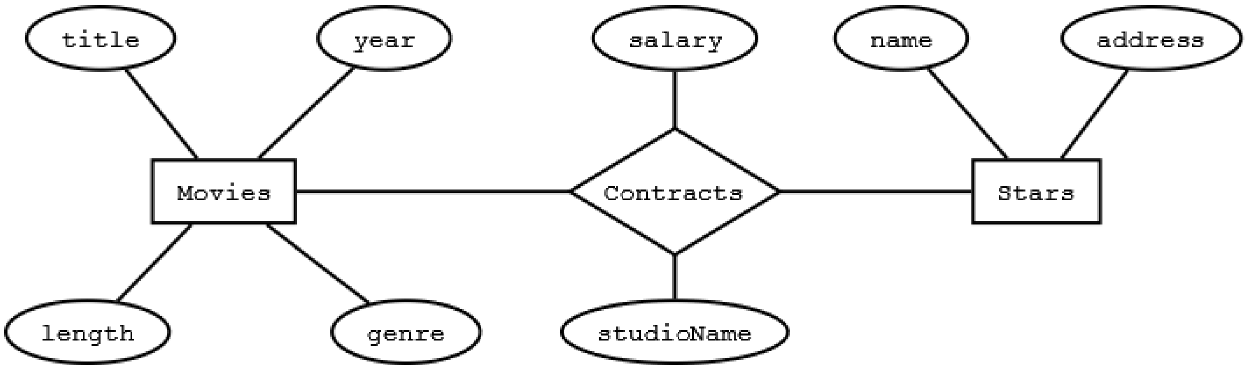




4.2.1
Redundancy: The owner address is repeated in AccSets and Addresses entity sets.
Simplicity: AccSets does not serve any useful purpose and the design can be more simply represented by creating many-to-many relationship between Customers and Accounts.

Right kind of element: The entity set Addresses has a single attribute address.
A customer cannot have more than one address.
Hence address should be an attribute of entity set Customers.
Faithfulness: Customers cannot be uniquely identified by their names. In real world Customers would have a unique attribute such as ssNo or customerNo

4.2.2
Studios and Presidents can be combined into one entity set Studios with Presidents becoming an attribute of Studios under following circumstances:
1. The Presidents entity set only contains a simple attribute viz. presidentName.
Additional attributes specific to Presidents might justify making Presidents into an entity set.

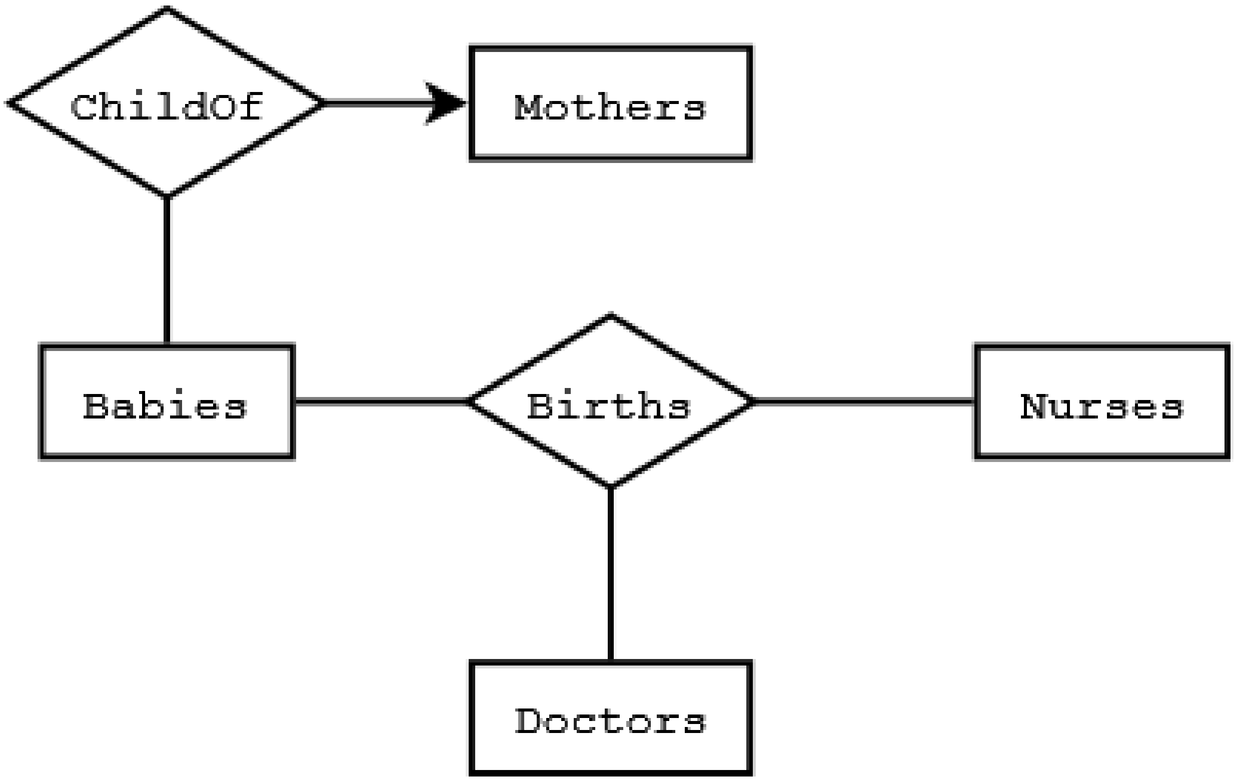
4.2.3


4.2.4 The entity sets should have single attribute.
a) Stars: starName
b) Movies: movieName
c) Studios: studioName. However there exists a many-to-many relationship between Studios and Contracts. Hence, in addition, we need more information about studios involved. If a contract always involves two studios, two attributes such as producingStudio and starStudio can replace the
Studios entity set. If a contact can be associated with at most five studios, it may be possible to replace the Studios entity set by five attributes viz. studio1, studio2, studio3, studio4, and studio5. Alternately, a composite attribute containing concatenation of all studio names in a contact can be considered. A separator character such as "$" can be used. SQL allows searching of such an attribute using query like '%keyword%'

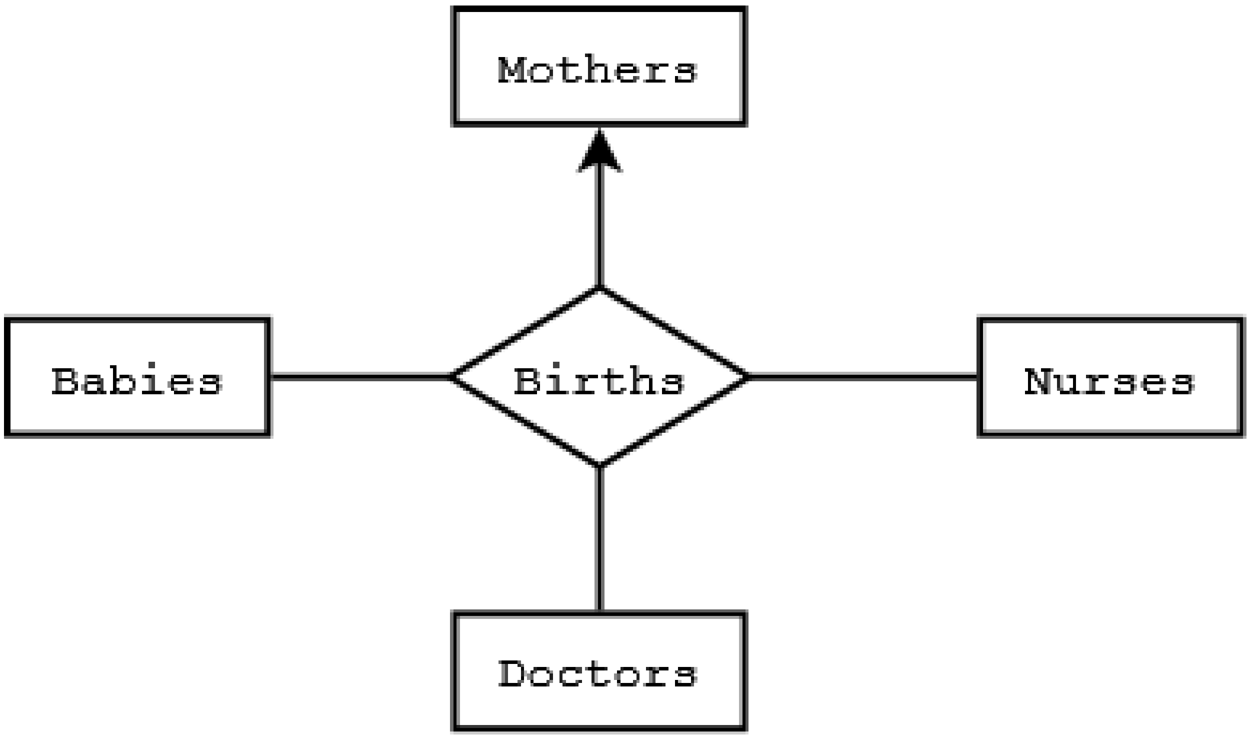
4.2.5

From Augmentation rule of Functional Dependency,
given
B -> M   (B=Baby, M=Mother)

then
BND -> M (N=Nurse, D=Doctor)
Hence we can just put an arrow entering mother.
a) Put an arrow entering entity set Mothers for the simplest solution (As in fig. 4.4, where a multi-way relationship was allowed, even though Movies alone could identify the Studio). However, we can display more accurate information with below figure.

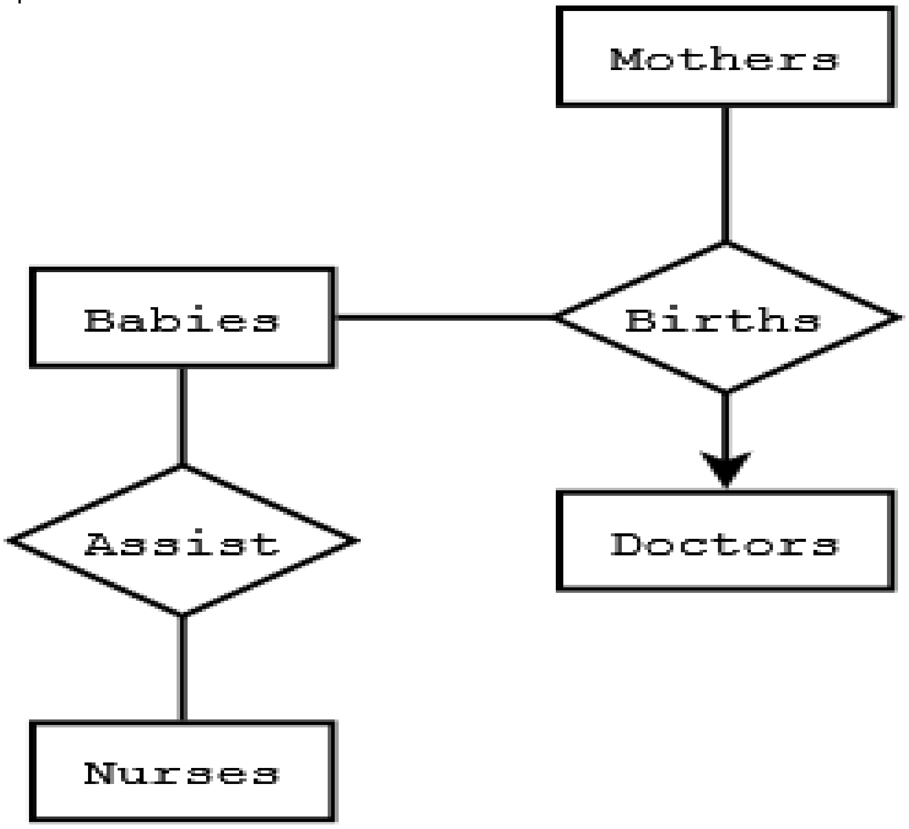


b)

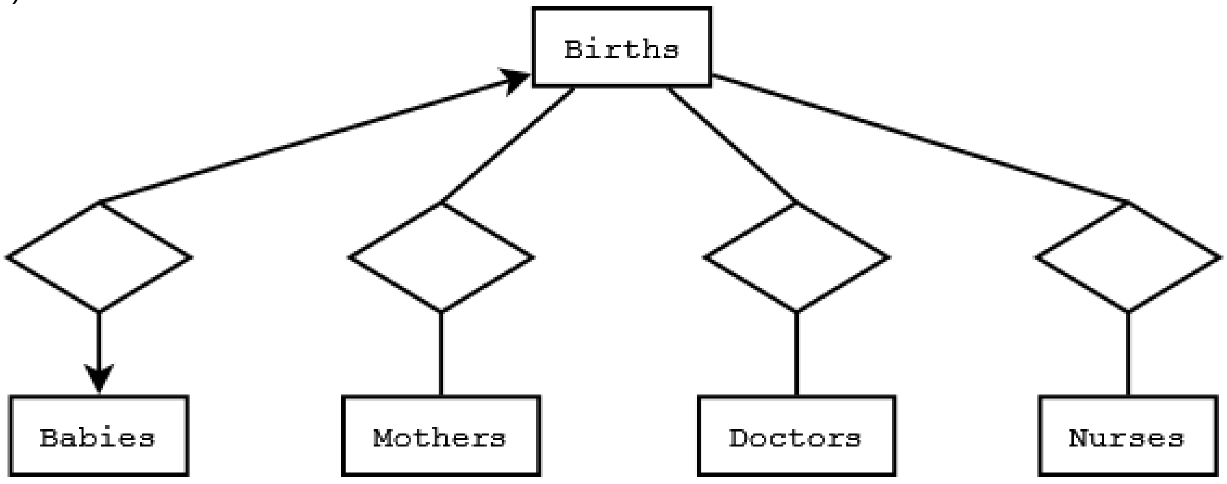


c)
Again from Augmentation rule of Functional Dependency,

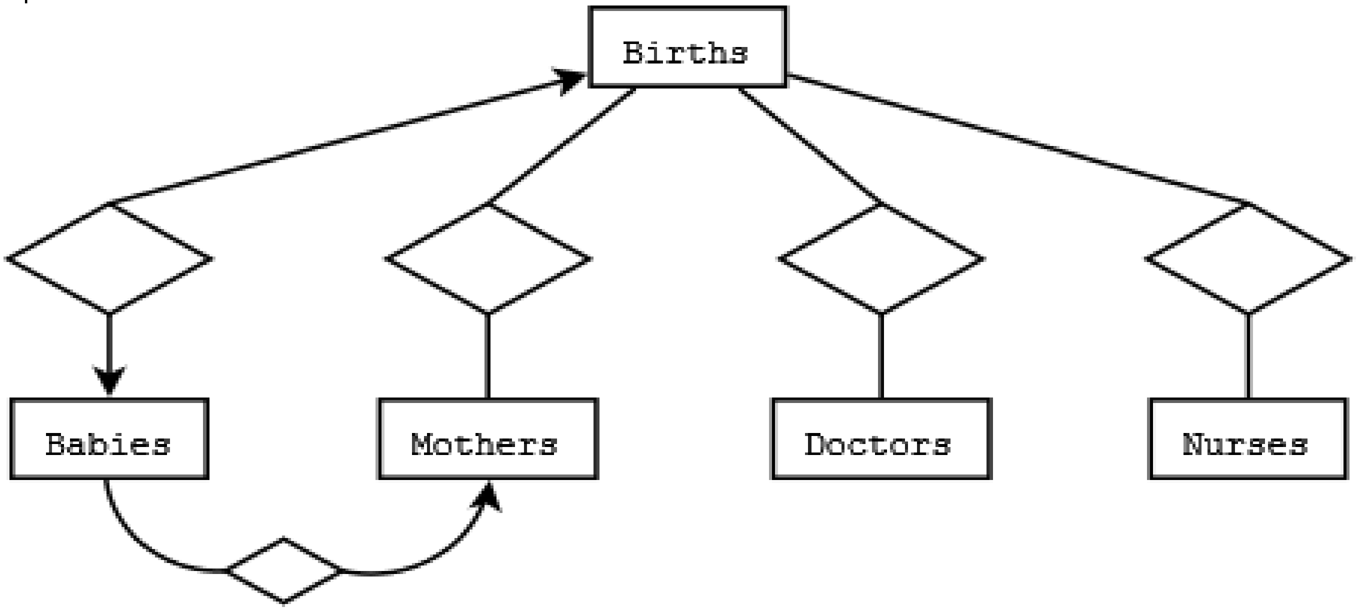
given
BM -> D
then
BMN -> D
Thus we can just add an arrow entering Doctors to fig 4.15. Below figure represents more accurate information however.



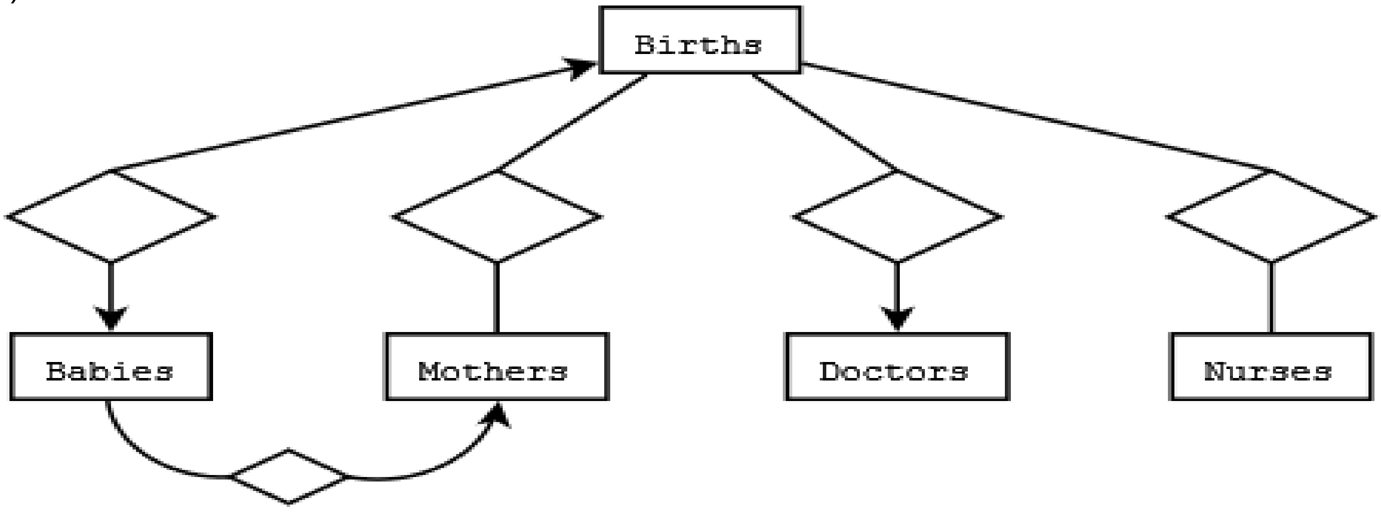
4.2.6
a)



b) Transitivity and Augmentation rules of Functional Dependency allow arrow entering Mothers from Births. However, a new relationship in below figure represents more accurate information.

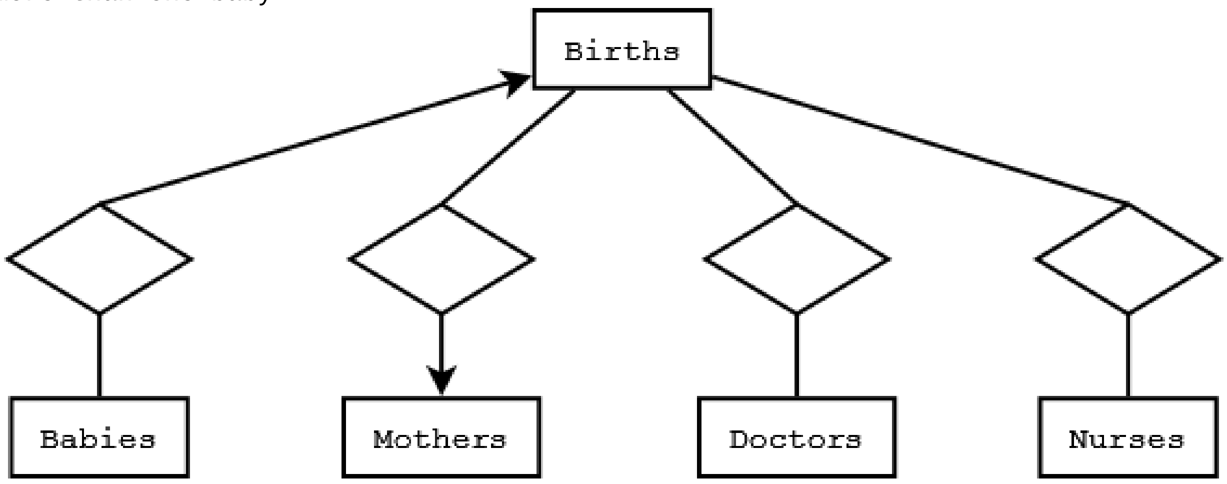


c)

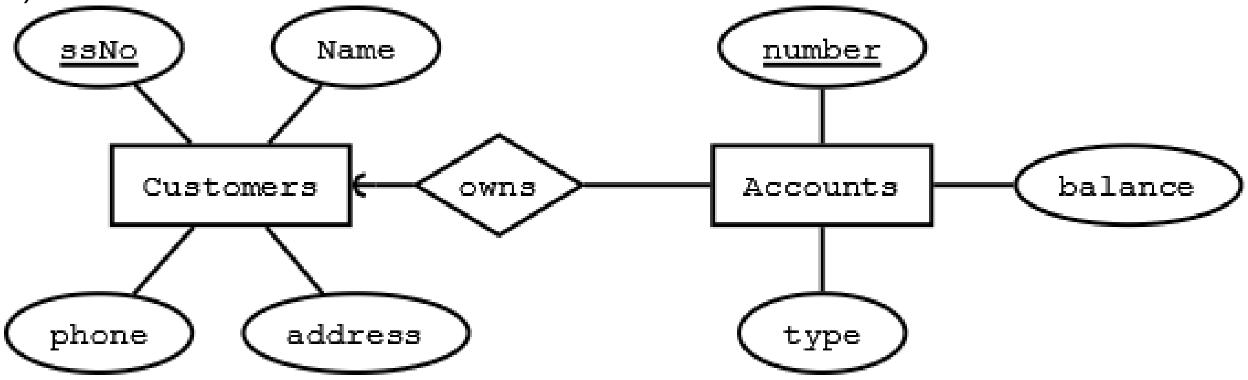


Design flaws in abc above 1. As suggested above, using Transitivity and Augmentation rules of Functional Dependency, much simpler design is possible.

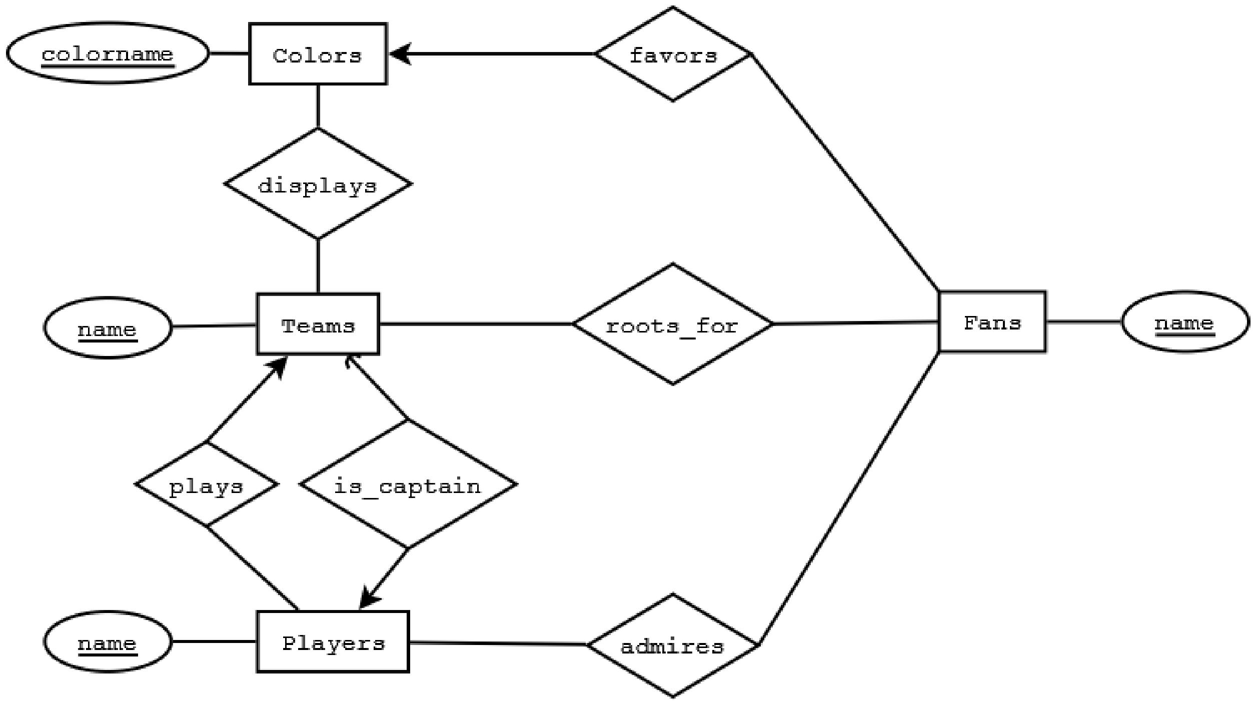
4.2.7
In below figure there exists a many-to-one relationship between Babies and Births and another many-to-one relationship between Births and Mothers. From transitivity of relationships, there is a many-to-one relationship between Babies and Mothers. Hence a baby has a unique mother while a birth can allow more than one baby.



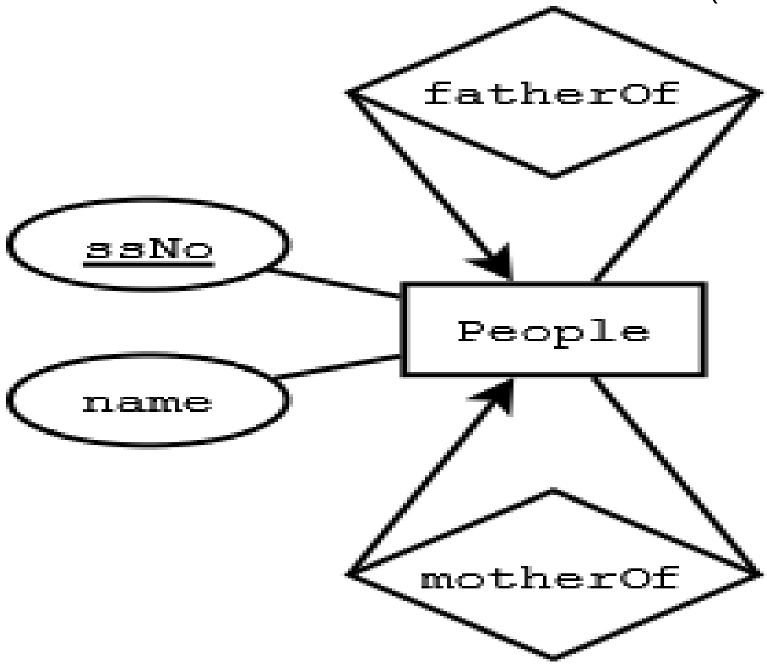
4.3.1
a)



b)
A captain cannot exist without a team. However a player can (free agent). A recently formed (or defunct) team can exist without players or colors.



c)
Children can exist without mother and father (unknown).



4.3.2
a)
The keys of both E1 and E2 are required for uniquely identifying tuples in R

b)
The key of E1

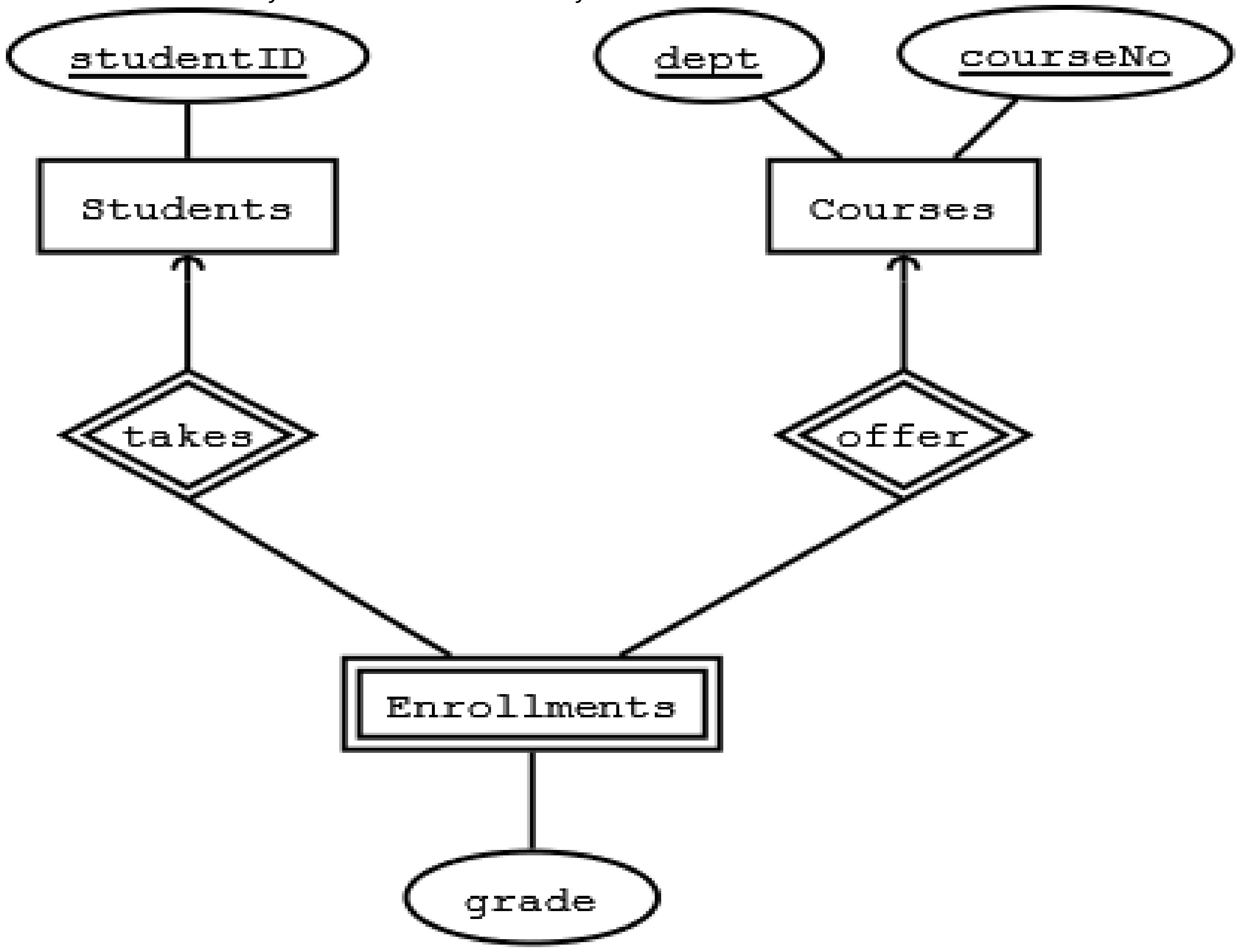
c)
The key of E2

d)
The key of either E1 or E2

4.3.3
Special Case: All entity sets have arrows going into them i.e. all relationships are 1-to-1
Any Ki

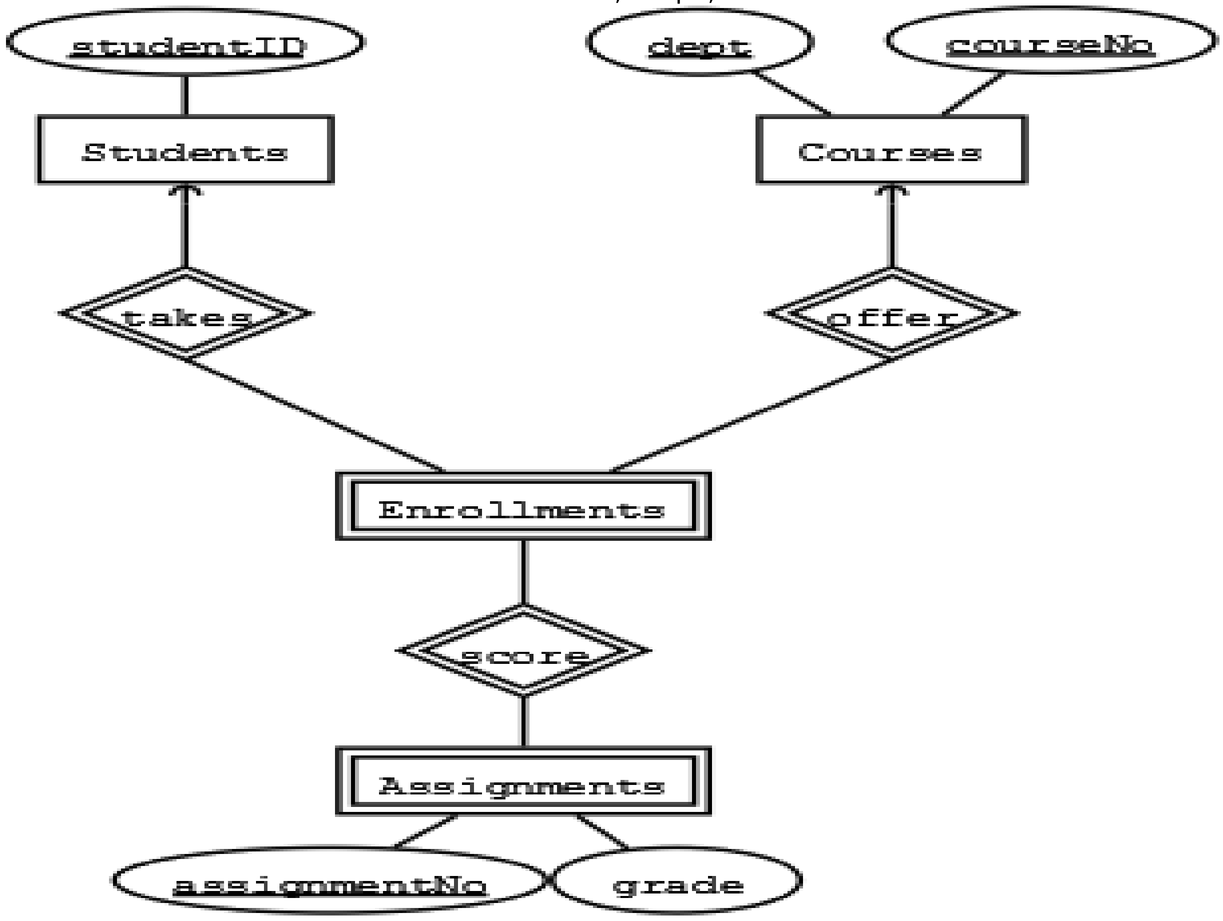
Otherwise: Combination of all Ki's where there does not exist an arrow going from R to Ei.

4.4.1
No, grade is not part of the key for enrollments. The keys of Students and Courses become keys of the weak entity set Enrollments.

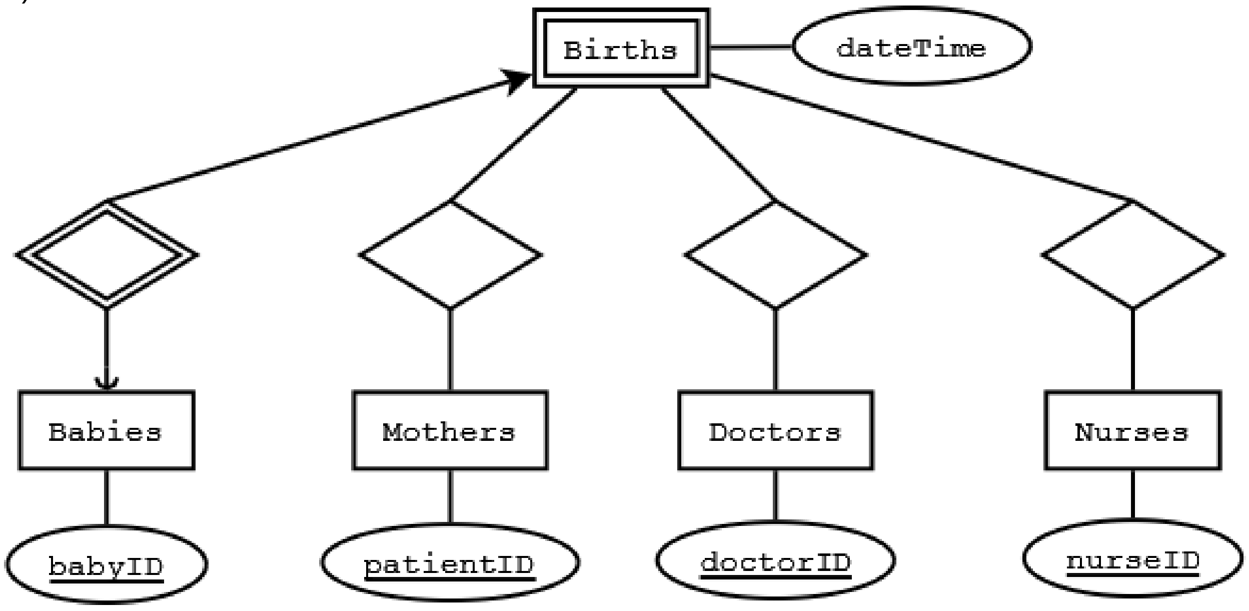


4.4.2

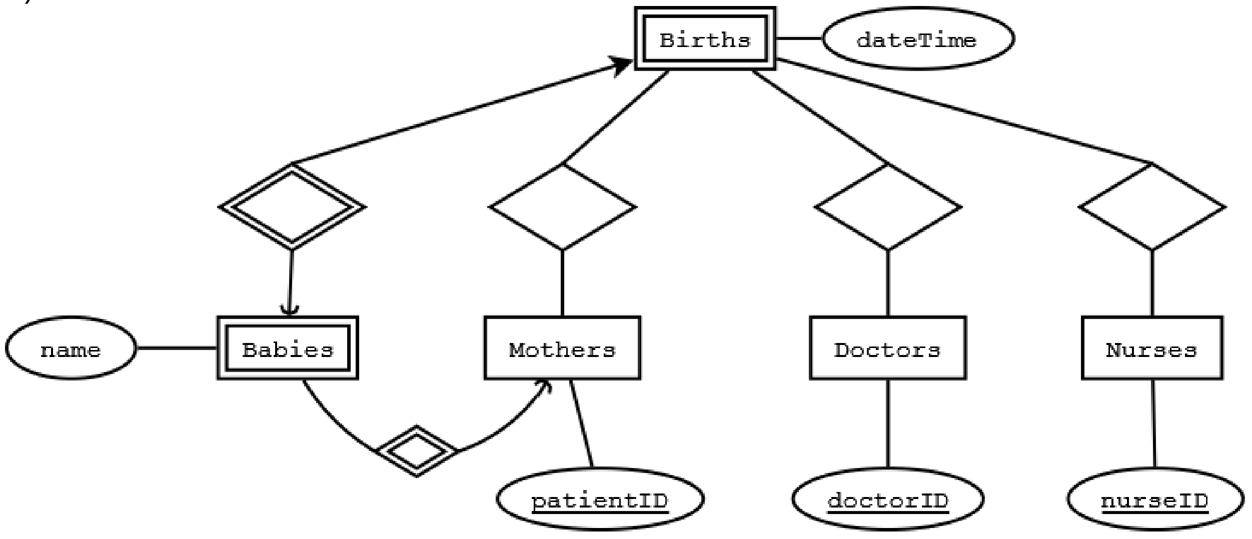
It is possible to make assignment number a weak key of Enrollments but this is not good design (redundancy since multiple assignments correspond to a course). A new entity set Assignment is created and it is also a weak entity set. Hence the key attributes of Assignment will come from the strong entity sets to which Enrollments is connected i.e. studentID, dept, and CourseNo.



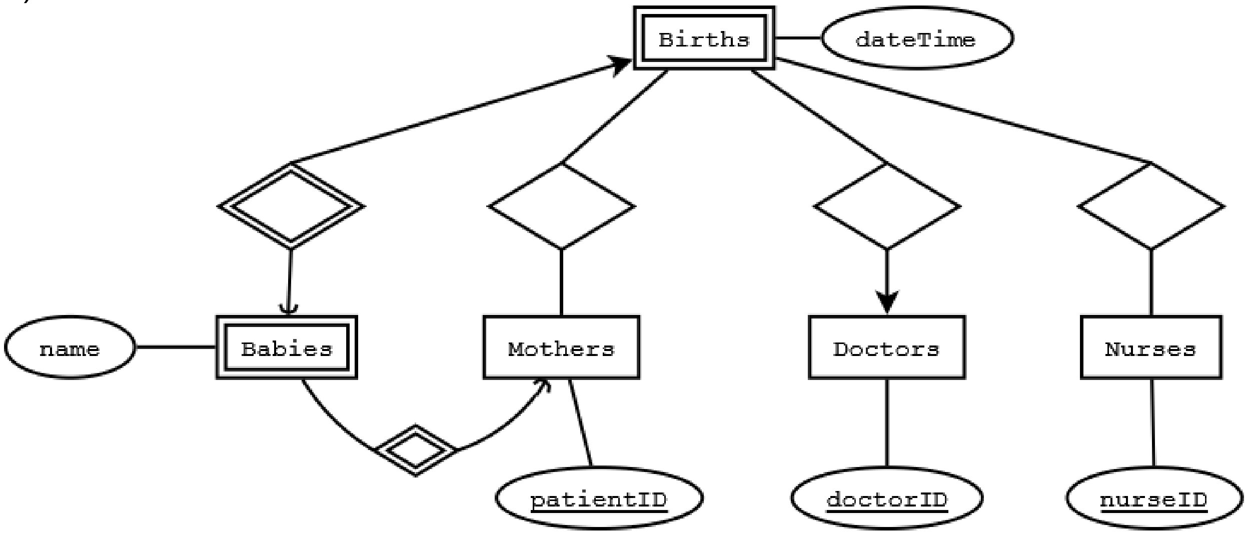
4.4.3
a)


Births
dateTime
Babies
Mothers
Doctors
Nurses
babyID
patientID
doctorID
nurseID

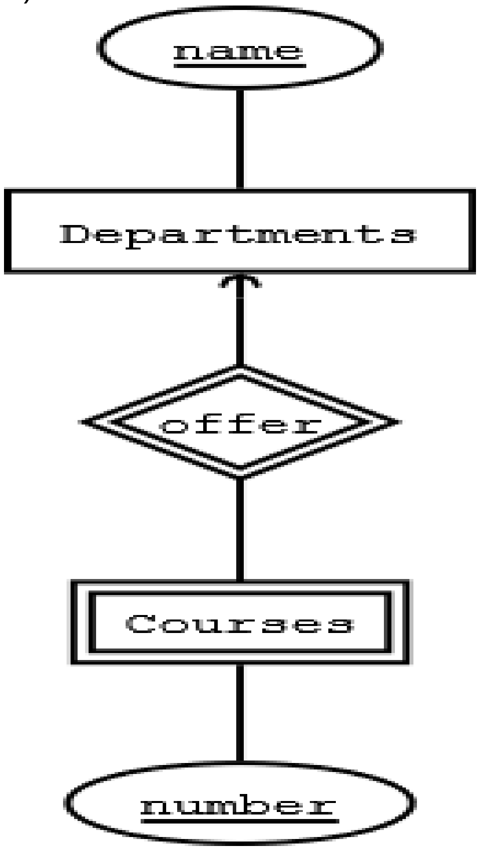
b)


Births
dateTime
name
Babies
Mothers
Doctors
Nurses
patientID
doctorID
nurseID

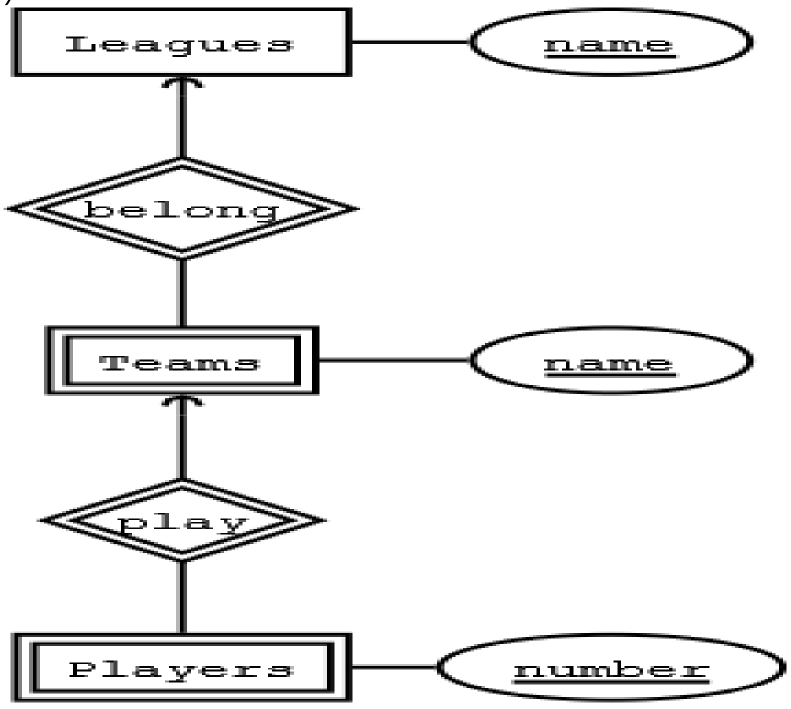
c)


Births
dateTime
name
Babies
Mothers
Doctors
Nurses
patientID
doctorID
nurseID

4.4.4
a)


name
Departments
offer
Courses
number

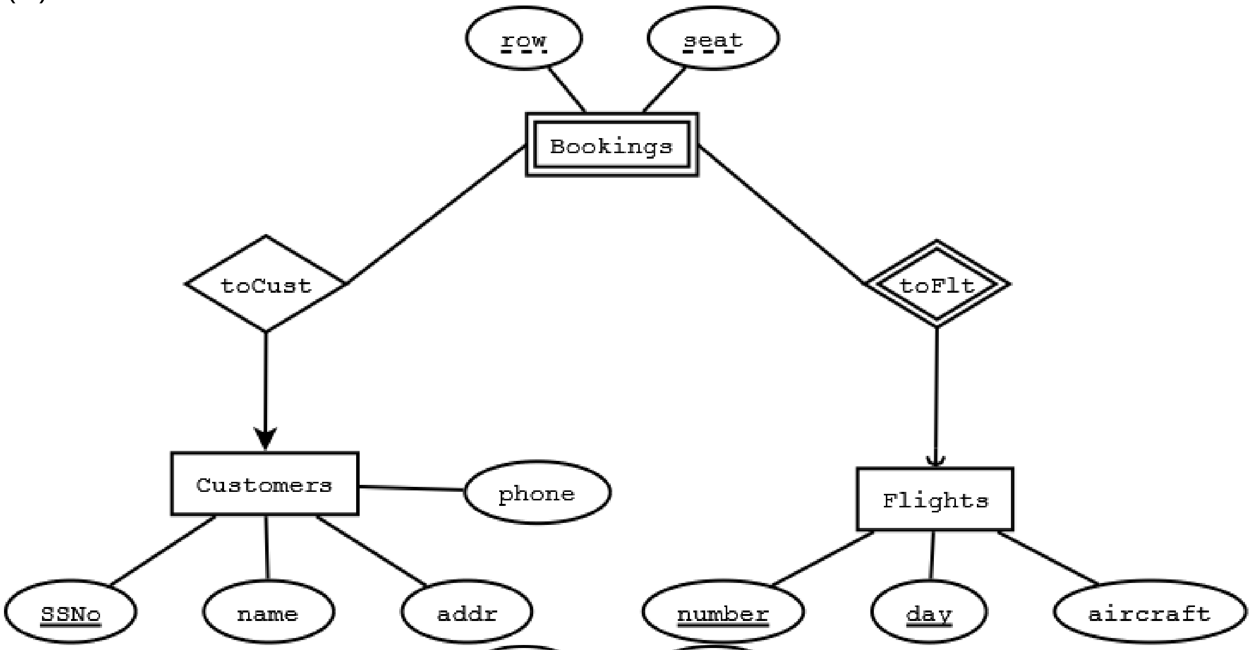
b)

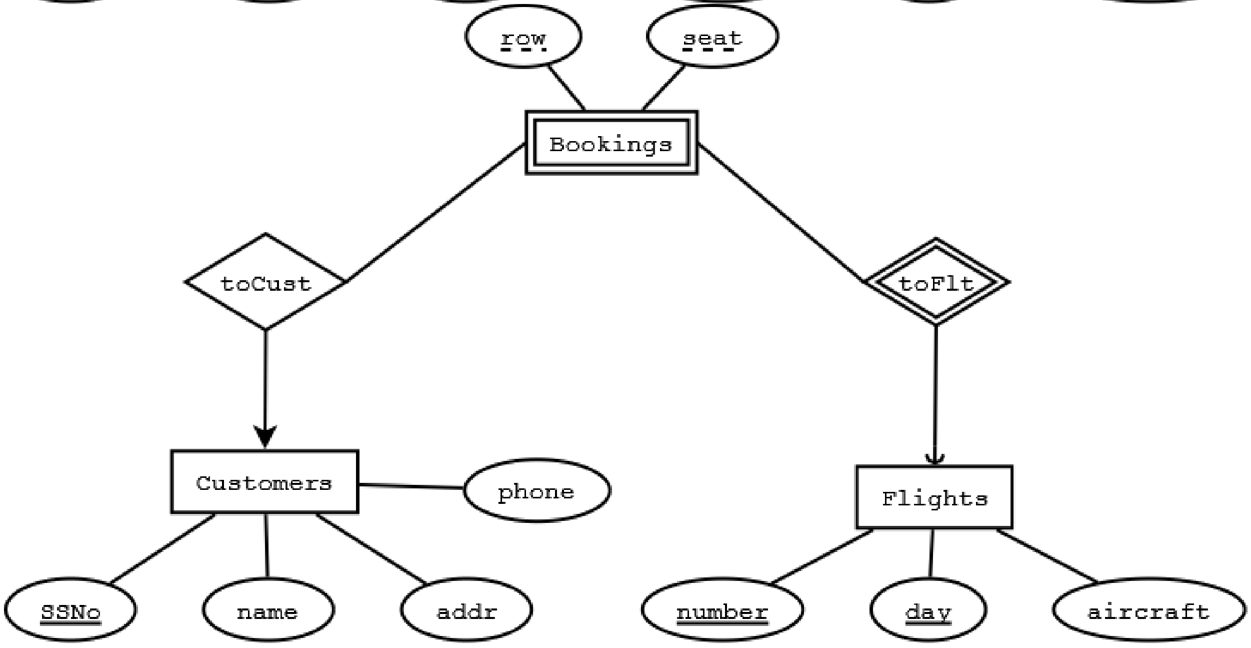


4.5.1
Customers(SSNo,name,addr,phone)
Flights(number,day,aircraft)
Bookings(custSSNo,flightNo,flightDay,row,seat)

Relations for toCust and toFlt relationships are not required since the weak entity set Bookings already contains the keys of Customers and Flights.

4.5.2
(a)





(b)
Schema is changed. Since toCust is no longer an identifying relationship, SSNo is no longer a part of Bookings relation.
Bookings(flightNo,flightDay,row,seat)
ToCust(custSSNO,flightNo,flightDay,row,seat)

The above relations are merged into
Bookings(flightNo,flightDay,row,seat,custSSNo)
However custSSNo is no longer a key of Bookings relation. It becomes a foreign key instead.

```
4.5.3
    Ships(name, yearLaunched)
    SisterOf(name, sisterName)
```

```
4.5.4
(a)
Stars(name,addr)
Studios(name,addr)
Movies(title,year,length,genre)
Contracts(starName,movieTitle,movieYear,studioName,salary)
```

Depending on other relationships not shown in ER diagram, studioName may not be required as a key of Contracts (or not even required as an attribute of Contracts).

```
(b)
Students(studentID)
Courses(dept,courseNo)
Enrollments(studentID,dept,courseNo,grade)

(c)
Departments(name)
Courses(deptName,number)

(d)
Leagues(name)
Teams(leagueName,teamName)
Players(leagueName,teamName,playerName)
```

4.6.1
The weak relation Courses has the key from Depts along with number. Hence there is no relation for GivenBy relationship.

```
(a)

    Depts(name, chair)
    Courses(number, deptName, room)
    LabCourses(number, deptName, allocation)
```

(b) LabCourses has all the attributes of Courses.

```
    Depts(name, chair)
    Courses(number, deptName, room)
    LabCourses(number, deptName, room, allocation)
```

(c) Courses and LabCourses are combined into one relation.

```
    Depts(name, chair)
    Courses(number, deptName, room, allocation)
```

4.6.2
(a)

```
Person(name,address)
ChildOf(personName,personAddress,childName,childAddress)
Child(name,address,fatherName,fatherAddress,motherName,motherAddresss)
Father(name,address,wifeName,wifeAddresss)
Mother(name,address)
```

Since FatherOf and MotherOf are many-one relationships from Child, there is no need for a separate relation for them. Similarly the one-one relationship Married can be included in Father (or Mother). ChildOf is a many-many relationship and needs a separate relation.

However the ChildOf relation is not required since the relationship can be deduced from FatherOf and MotherOf relationships contained in Child relation.

(b)
A person cannot be both Mother and Father.

```
Person(name,address)
PersonChild(name,address)
PersonChildFather(name,address)
PersonChildMother(name,address)
PersonFather(name,address)
PersonMother(name,address)

ChildOf(personName,personAddress,childName,childAddress)
FatherOf(childName,childAddress,fatherName,fatherAddress)
MotherOf(childName,childAddress,motherName,motherAddress)
Married(husbandName,husbandAddress,wifeName,wifeAddress)
```

The many-many ChildOf relationship again requires a relation.

An entity belongs to one and only one class when using object-oriented approach. Hence, the many-one relations MotherOf and FatherOf could be added as attributes to PersonChild,PersonChildFather, and PersonChildMother relations.
Similarly the Married relation can be added as attributes to PersonChildMother and PersonMother (or the corresponding father relations).

(c) For the Person relation at least one of husband and wife attributes will be null.
Person(personName,personAddress,fatherName,fatherAddress,motherName,motherAddress,wifeName,wifeAddresss,husbandName,husbandAddress)
ChildOf(personName,personAddress,childName,childAddress)

4.6.3
(a)

People(name,fatherName,motherName)
Males(name)
Females(name)
Fathers(name)
Mothers(name)
ChildOf(personName,childName)

(b)
People(name)
PeopleMale(name)
PeopleMaleFathers(name)
PeopleFemale(name)
PeopleFemaleMothers(name)

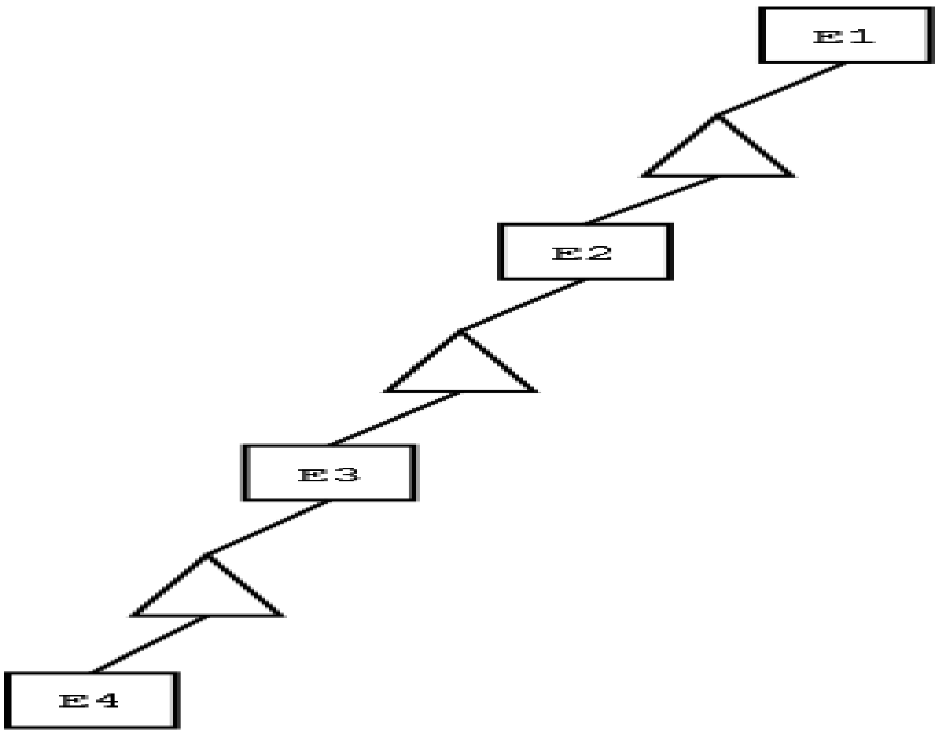
ChildOf(personName,childName)
FatherOf(childName,fatherName)
MotherOf(childName,motherName)

People cannot belong to both male and female branch of the ER diagram.
Moreover since an entity belongs to one and only one class when using object-oriented approach, no entity belongs to People relation.
Again we could replace MotherOf and FatherOf relations by adding as attributes to PeopleMale,PeopleMaleFathers,PeopleFemale, and PeopleFemaleMothers relations.

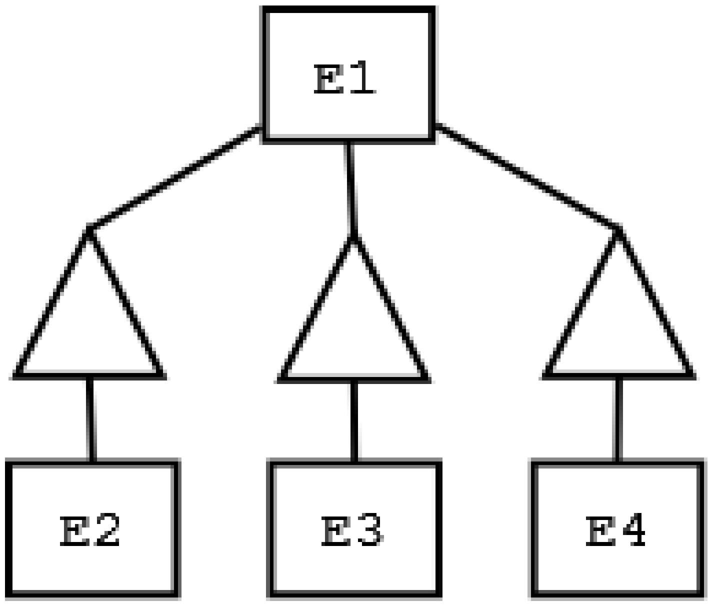
(c)
People(name,fatherName,motherName)
ChildOf(personName,childName)

4.6.4
(a)
Each entity set results in one relation. Thus both the minimum and maximum number of relations is e.
The root relation has a attributes including k keys. Thus the minimum number of attributes is a. All other relations include the k keys from root along with their a attributes. Thus the maximum number of attributes is a+k.

(b)
The relation for root will have a attributes. The relation representing the whole tree will have e*a attributes.
The number of relations will depend on the shape of the tree. A tree of e entities where only one child exists(say left child only) would have the minimum number of relations. Thus below figure will only contain 4 subtrees that contain root E1,E1E2,E1E2E3, and E1E2E3E4. With e entity sets, minimum e relations are possible.



The maximum number of subtrees result when all the entities(except root) are at depth 1. Thus below figure will contain 8 subtrees that contain root E1,E1E2,E1E3,E1E4,E1E2E3,E1E3E4,E1E2E4,and E1E2E3E4. With e entity sets, maximum 2^(e-1) relations are possible.

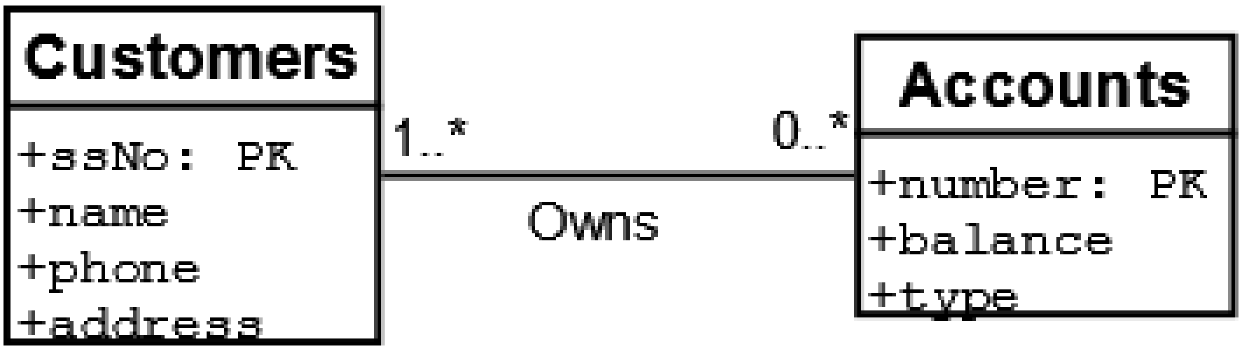


(c)
The nulls method always results in one relation and contains attributes from all e entities i.e. e*a attributes.

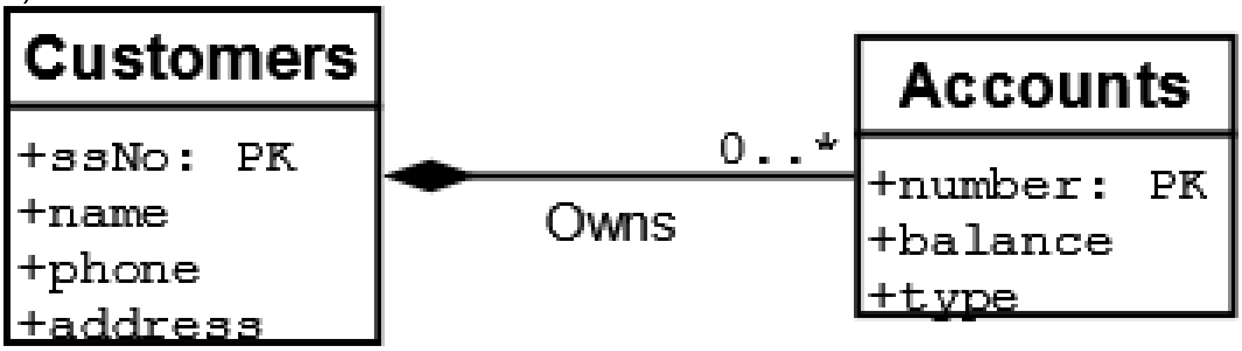
Summarizing for a,b, and c above;

	#Components		#Relations	
	Min	Max	Min	Max
Method				
straight-E/R	a	a	e	e
object-oriented	a	e*a	e	2^(e-1)
nulls	e*a	e*a	1	1

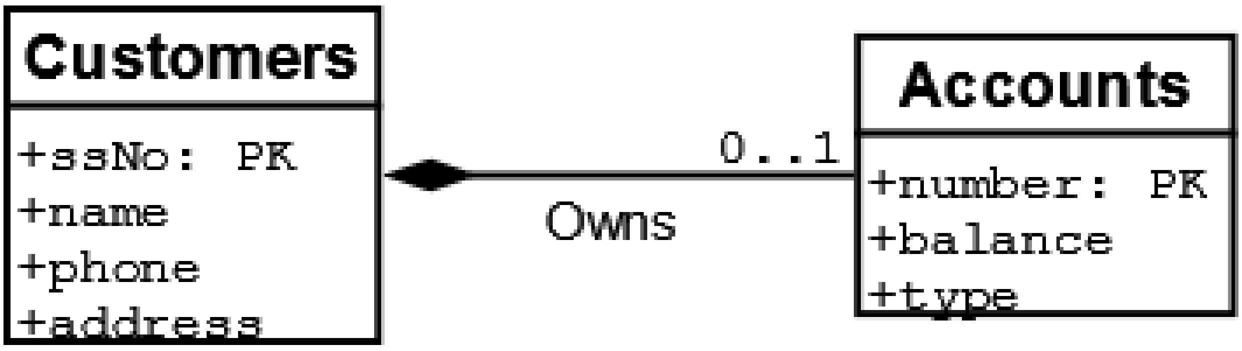
4.7.1


Customers
+ssNo: PK
+name
+phone
+address
1..*
Owns
0..*
Accounts
+number: PK
+balance
+type

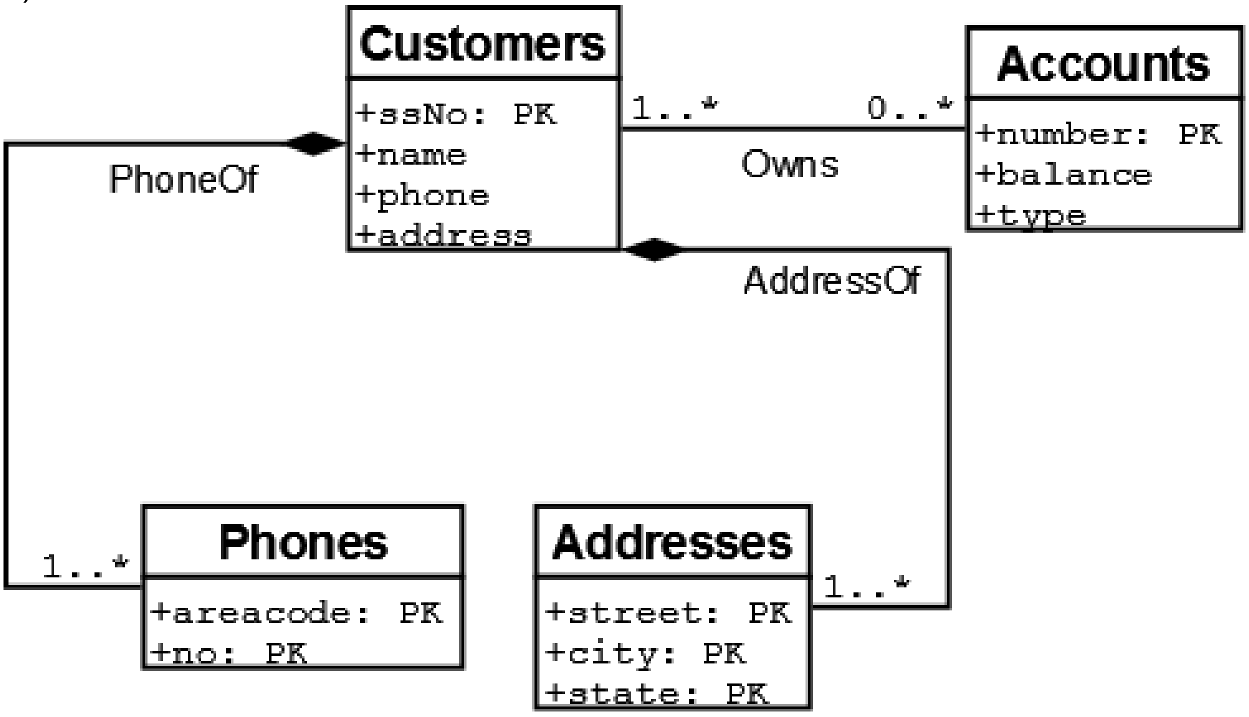
4.7.2
a)


Customers
+ssNo: PK
+name
+phone
+address
Owns
0..*
Accounts
+number: PK
+balance
+type

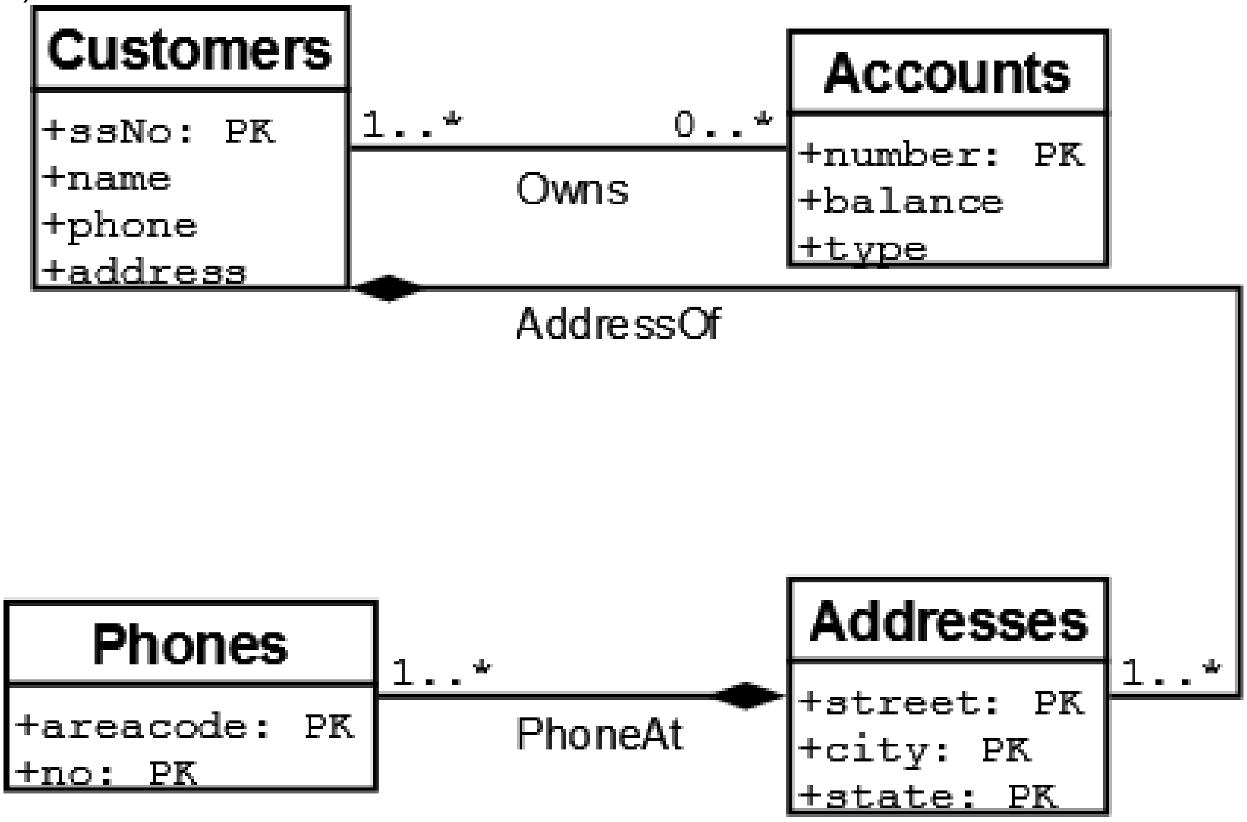
b)


Customers
+ssNo: PK
+name
+phone
+address
Owns
0..1
Accounts
+number: PK
+balance
+type

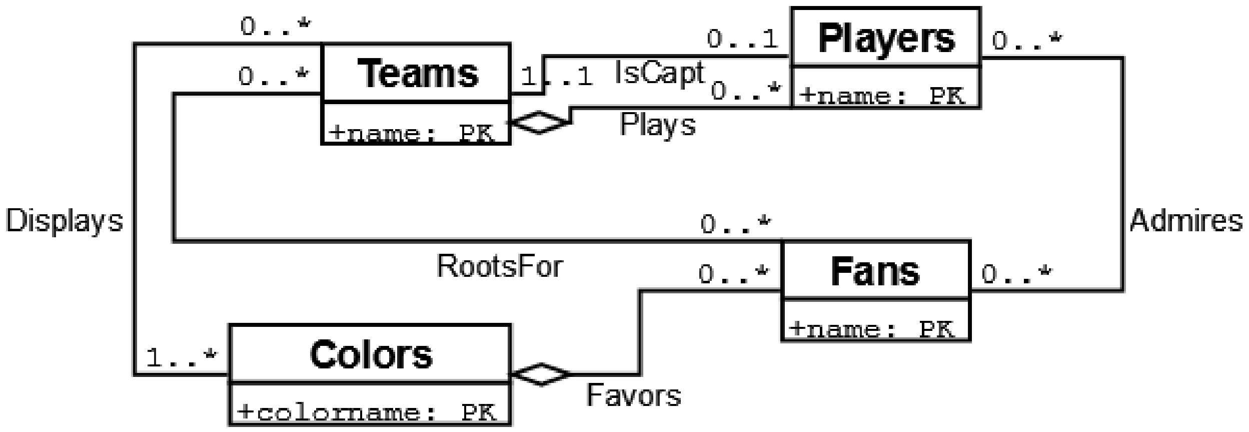
c)


Customers
+ssNo: PK
+name
+phone
+address
1..*
0..*
Owns
Accounts
+number: PK
+balance
+type
PhoneOf
AddressOf
1..*
Phones
+areacode: PK
+no: PK
Addresses
+street: PK
+city: PK
+state: PK
1..*

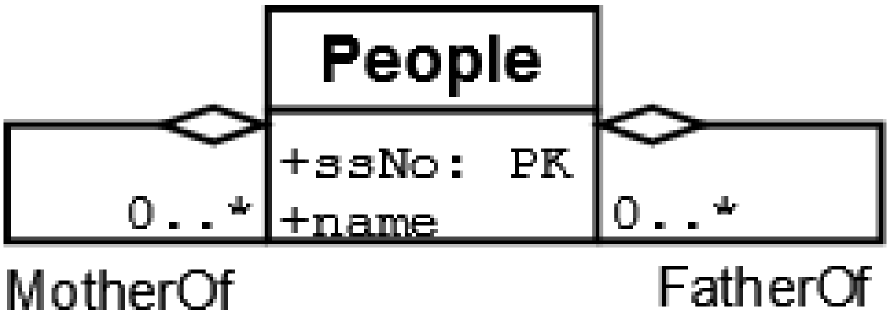
d)


Customers
+ssNo: PK
+name
+phone
+address
1..*
0..*
Owns
Accounts
+number: PK
+balance
+type
AddressOf
Phones
+areacode: PK
+no: PK
1..*
PhoneAt
Addresses
+street: PK
+city: PK
+state: PK
1..*

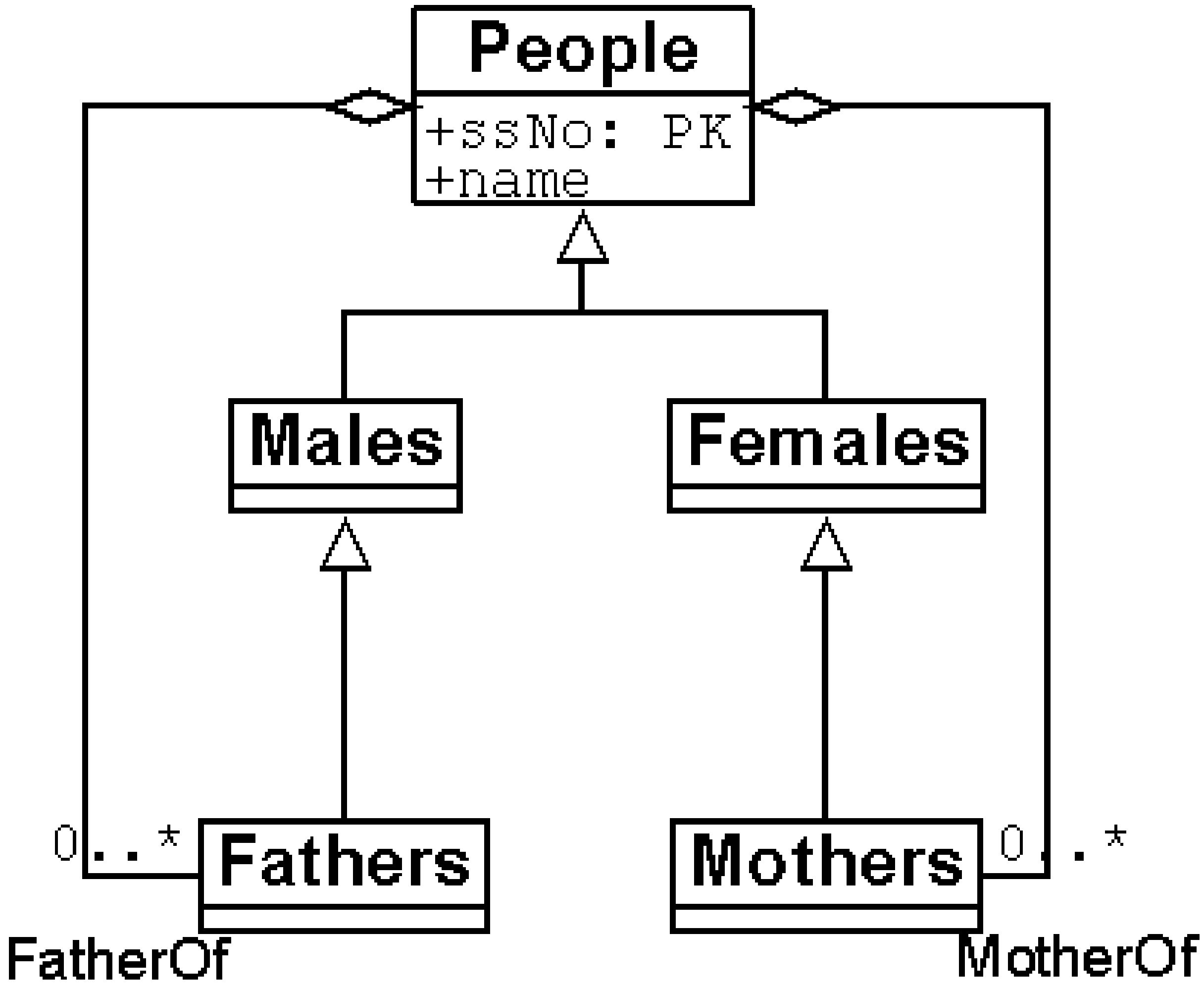
4.7.3


0..*
0..*
Teams
+name: PK
1..1
IsCapt
0..1
0..*
Plays
Players
+name: PK
0..*
Displays
Admires
0..*
RootsFor
0..*
Fans
+name: PK
0..*
1..*
Colors
+colorname: PK
Favors

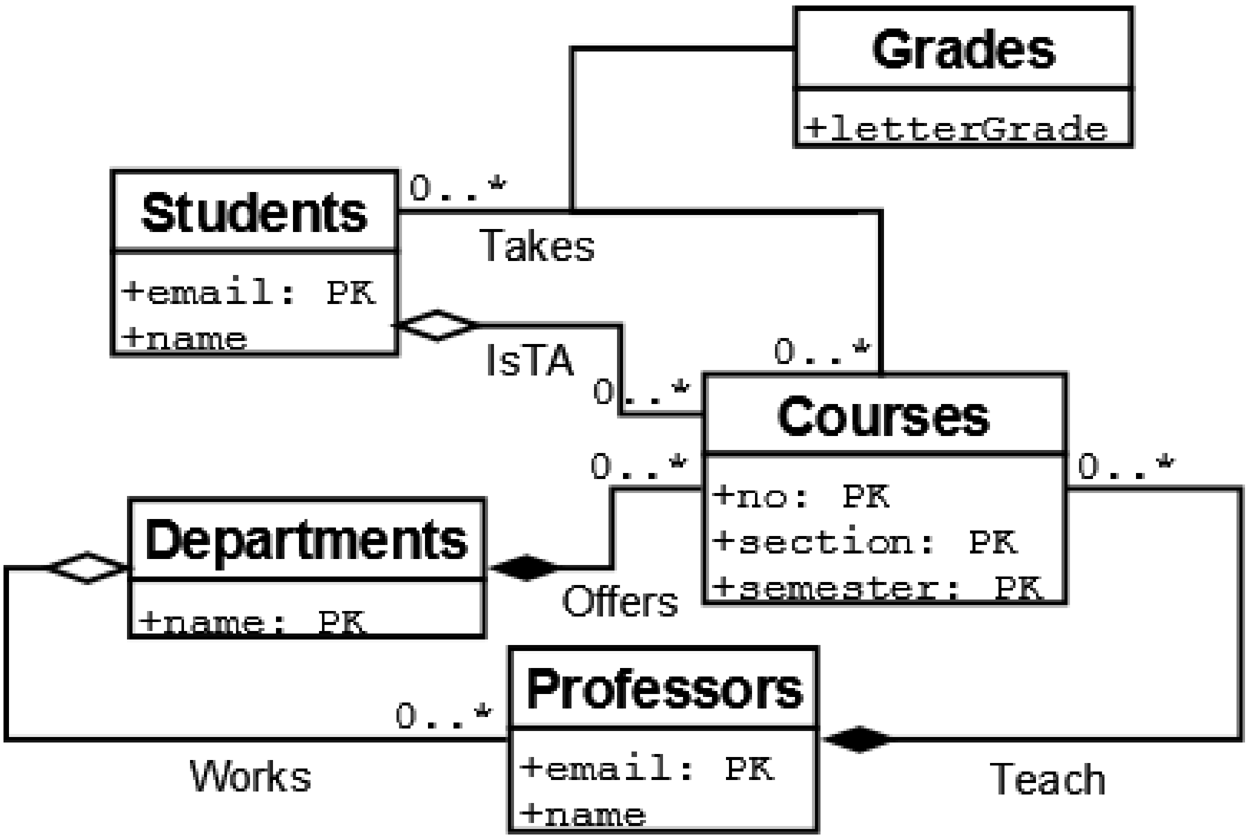
4.7.4



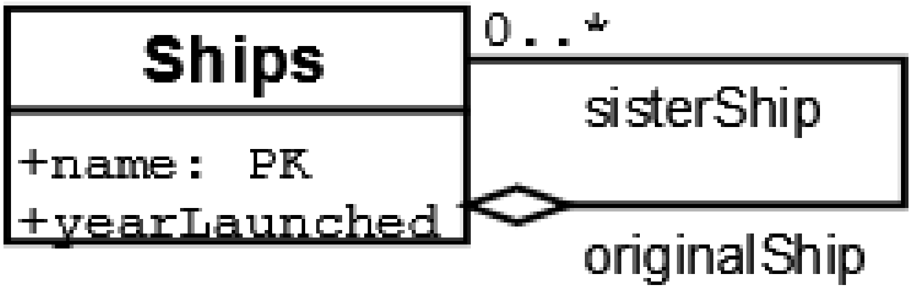
4.7.5
Males and Females subclasses are complete. Mothers and Fathers are partial. All subclasses are disjoint.



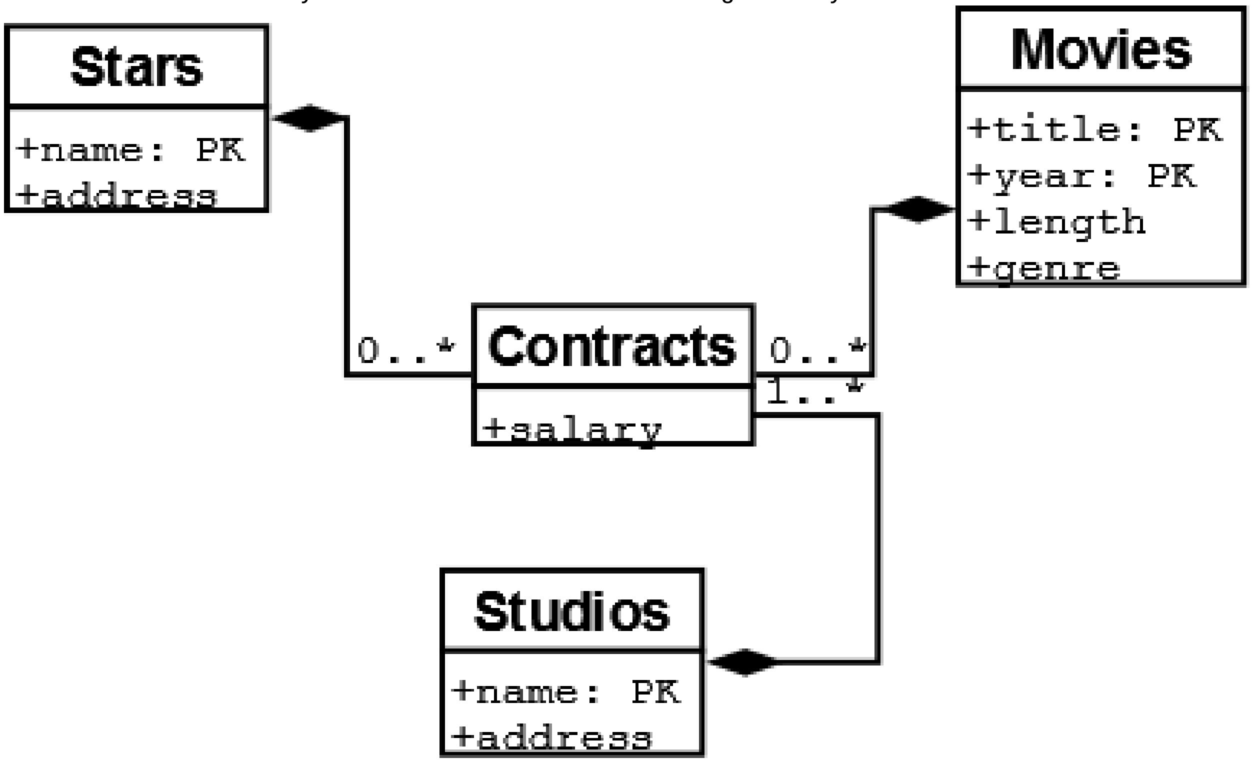
4.7.6


Grades
+letterGrade
Students
+email: PK
+name
0..*
Takes
IsTA
0..*
0..*
Courses
+no: PK
+section: PK
+semester: PK
0..*
0..*
Departments
+name: PK
Offers
Professors
+email: PK
+name
0..*
Works
Teach

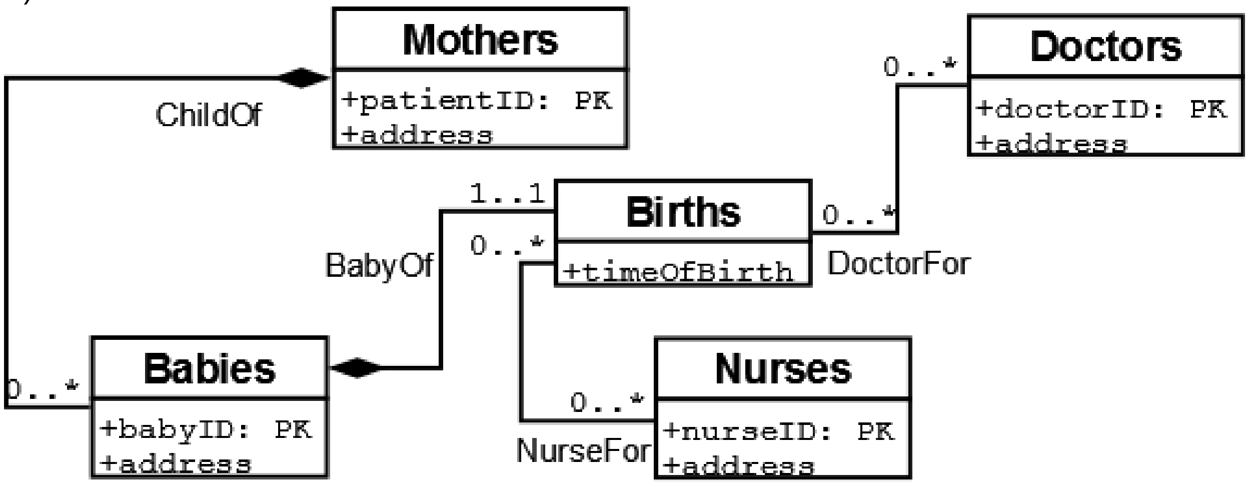
4.7.7


Ships
+name: PK
+yearLaunched
0..*
sisterShip
originalShip

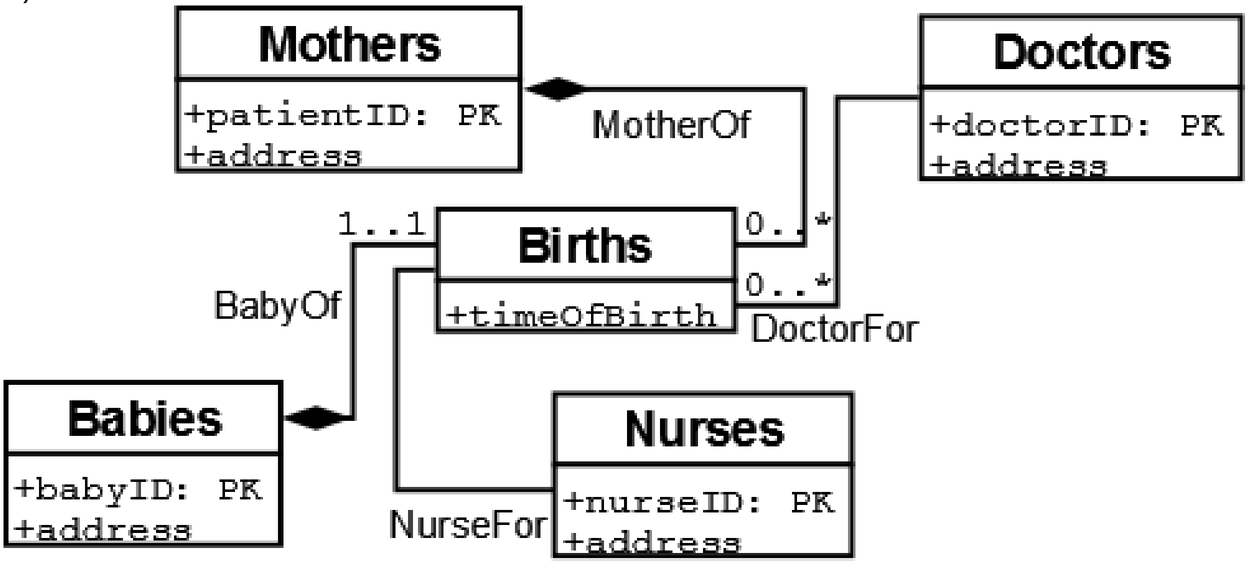
4.7.8
We convert the ternary relationship Contracts into three binary relationships between a new entity set Contracts and existing entity sets.



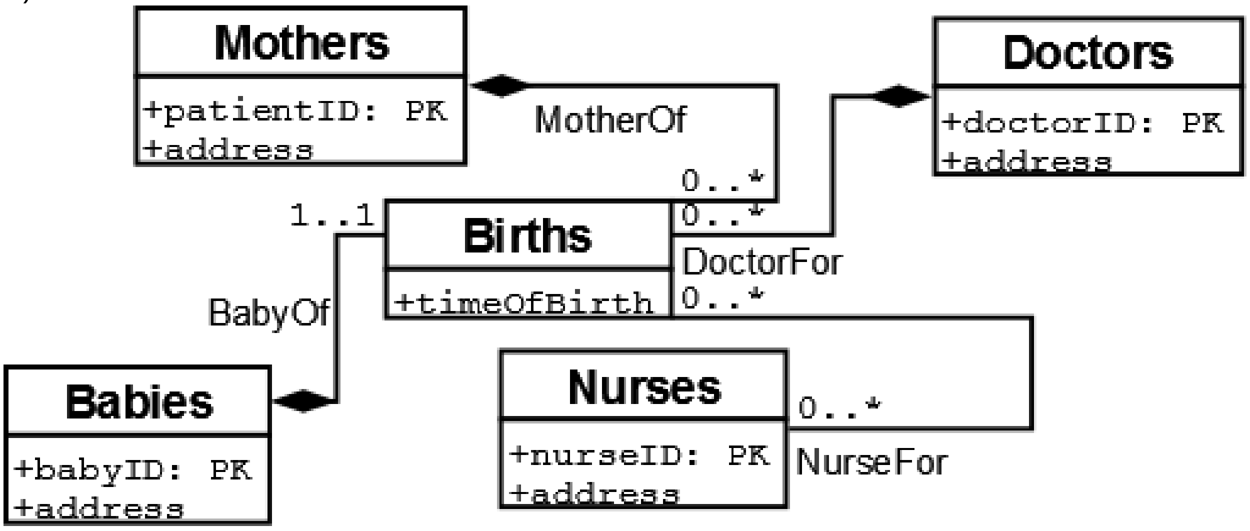
4.7.9
a)



b)



c)



4.7.10
A self-association ParentOf for entity set people has multiplicity 0..2 at parent role end.
In a Library database, if a patron can loan at most 12 books, them multiplicity is 0..12.
For a FullTimeStudents entity set, a relationship of multiplicity 5..* must exist with Courses (A student must take at least
5 courses to be classified FullTime.

```
4.8.1
Customers(SSNo,name,addr,phone)
Flights(number,day,aircraft)
Bookings(row,seat,custSSNo,FlightNumber,FlightDay)

Customers("SSNo",name,addr,phone)
Flights("number","day",aircraft)
Bookings(row,seat,"custSSNo","FlightNumber","FlightDay")

4.8.2
a)
Movies(title,year,length,genre)
Studios(name,address)
Presidents(cert#,name,address)
Owns(movieTitle,movieYear,studioName)
Runs(studioName,presCert#)

Movies("title","year",length,genre)
Studios("name",address)
Presidents("cert#",name,address)
Owns("movieTitle","movieYear",studioName)
Runs("studioName",presCert#)

b)
Since the subclasses are disjoint, Object Oriented Approach is used.
The hierarchy is not complete. Hence four relations are required
Movies(title,year,length,genre)
MurderMysteries(title,year,length,genre,weapon)
Cartoons(title,year,length,genre)
Cartoon-MurderMysteries(title,year,length,genre,weapon)

Movies("title","year",length,genre)
MurderMysteries("title","year",length,genre,weapon)
Cartoons("title","year",length,genre)
Cartoon-MurderMysteries("title","year",length,genre,weapon)


c)
Customers(ssNo,name,phone,address)
Accounts(number,balance,type)
Owns(custSSNo,accountNumber)

Customers("ssNo",name,phone,address)
Accounts("number",balance,type)
Owns("custSSNo","accountNumber")
```

d)
Teams(name,captainName)
Players(name,teamName)
Fans(name,favoriteColor)
Colors(colorname)
For Displays association,
TeamColors(teamName,colorname)
RootsFor(fanName,teamName)
Admires(fanName,playerName)

Teams("name",captainName)
Players("name",teamName)
Fans("name",favoriteColor)
Colors("colorname")
For Displays association,
TeamColors("teamName","colorname")
RootsFor("fanName","teamName")
Admires("fanName","playerName")

e)
People(ssNo,name,fatherSSNo,motherSSNo)

People("ssNo",name,fatherssNo,motherssNo)

f)
Students(email,name)
Courses(no,section,semester,professorEmail)
Departments(name)
Professors(email,name,worksDeptName)
Takes(letterGrade,studentEmail,courseNo,courseSection,courseSemester)

Students("email",name)
Courses("no","section","semester",professorEmail)
Departments("name")
Professors("email",name,worksDeptName)
Takes(letterGrade,"studentEmail","courseNo","courseSection","courseSemester")

4.8.3
a)
Each and every object is a member of exactly one subclass at leaf level. We have nine classes at the leaf of hierarchy. Hence we need nine relations.

b)
All objects only belong to one subclass and its ancestors. Hence, we need not consider every possible subtree but rather the total number of nodes in tree. Hence we need thirteen relations.

c)
We need all possible subtrees. Hence 218 relations are required.

4.9.1

```
class Customer (key (ssNo)){
      attribute integer ssNo;
      attribute string name;
      attribute string addr;
      attribute string phone;
      relationship Set<Account> ownsAccts
            inverse Account::ownedBy;
};

class Account (key (number)){
      attribute integer number;
      attribute string type;
      attribute real balance;
      relationship Set<Customer> ownedBy
            inverse Customer::ownsAccts;
};
```

4.9.2

a)
Modify class Account to contain relationship Customer ownedBy (no Set)

b)
Also remove set in relationship ownsAccts of class Customer.

c)
ODL allows a collection of primitive types as well as structures. To class Customer add following attributes in place of simple attributes addr and phone:

```
Set<string phone>
Set<Struct addr{string street,string city,string state}>
```

d)
ODL allows structures and collections recursively.

```
Set<Struct addr{string street,string city,string state},Set<string phone>>
```

4.9.3
Collections are allowed in ODL. Hence, Colors Set can become an attribute of Teams.

```
class Colors(key(colorname)){
            attribute string colorname;
                    relationship Set<Fans> FavoredBy
                      inverse Fans::Favors;
                    relationship set<Teams> DisplayedBy
                        inverse Teams::Displays;
                };

class Teams(key(name)){
          attribute string name;
                    relationship set<Colors> Displays
                        inverse Colors::DisplayedBy;
                    relationship set<Players> PlayedBy
                        inverse Players::Plays;
                    relationship PLayers CaptainedBy
                        inverse Platyers::Captains;

                    relationship set<Fans> RootedBy
                        inverse Fans::Roots;

                  };

class Players(key(name)){
                    attribute string name;
                    relationship Set<Teams> Plays
                        inverse Teams::PlayedBy;
                    relationship Teams Captains
                        inverse Teams::CaptainedBy;
                    relationship Set<Fans> AdmiredBy
                        inverse Fans::Admires;

                  };

class Fans(key(name)){
                    attribute string name;
                    relationship Colors Favors
                        inverse Colors::FavoredBy;
                    relationship Set<Teams> RootedBy
                        inverse Teams::Roots;
                    relationship Set<Players> Admires
                        inverse Players::AdmiredBy;

                  };
```

4.9.4

```
class Person {
      attribute string name;
      relationship Person motherOf
            inverse Person::childrenOfFemale;
      relationship Person fatherOf
            inverse Person::childrenOfMale;
      relationship Set<Person> children
            inverse Person::parentsOf;
      relationship Set<Person> childrenOfFemale
            inverse Person::motherOf;
      relationship Set<Person> childrenOfMale
            inverse Person::fatherOf;
      relationship Set<Person> parentsOf
            inverse Person::children;
};
```

4.9.5
The struct education{string degree,string school,string date} cannot have duplication.
Hence use of Sets does not make any different as compared to bags, lists, or arrays.
Lists will allow faster access/queries due to the already sorted nature.

4.9.6
a)

```
class Departments(key (name)) {
attribute string name;
relationship Courses offers
  inverse Courses::offeredBy;
};

class Courses(key (number,offeredBy)) {
attribute string number;
relationship Departments offeredBy
  inverse Departments::offers;
};
```

b)

```
class Leagues (key (name)) {
attribute name;
relationship Teams contains
  inverse Teams::belongs;
};

class Teams(key (name,belongs)) {
attribute name,
relationship Leagues belongs
  inverse Leagues::contains;
relationship Players play
  inverse Players::plays;
};

class Players (key(number,plays)) {
attribute number,
relationship Teams plays
  inverse Teams::play;
};
```

4.9.7

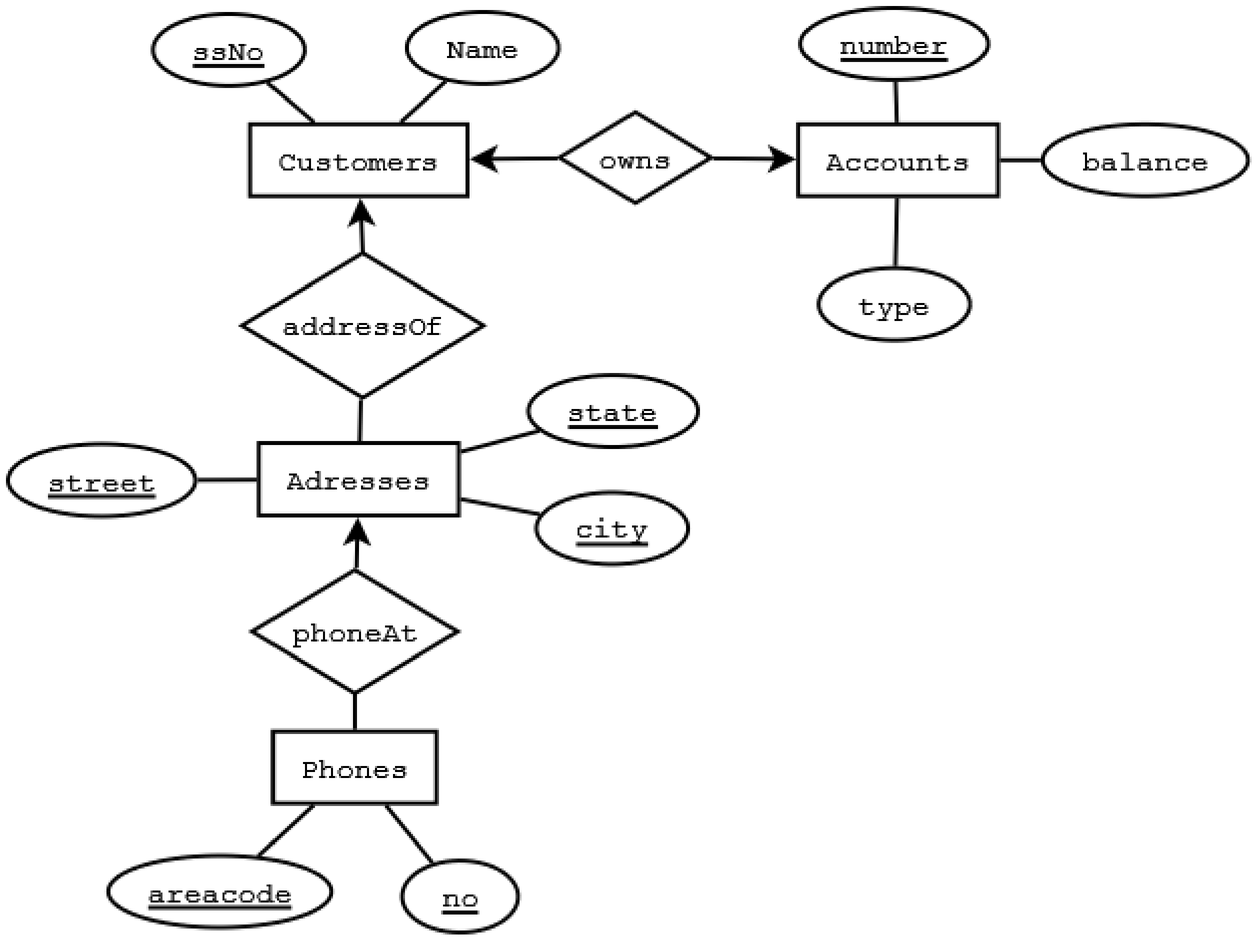
```
class Students (key email) {
  attribute string email;
  attribute string name;
  relationship Courses isTA
    inverse Courses::TA;
  relationship Courses Takes
    inverse Courses::TakenBy;
};

class Professors (key email) {
  attribute string email;
  attribute string name;
  relationship Departments WorksFor
    inverse Department::Works;
  relationship Courses  Teaches
    inverse Courses::TaughtBy;
};

class Courses (key (no,semester,section)) {
  attribute string no;
  attribute string semester;
  attribute string section;
  relationship Students TA
    inverse Students::isTA;
  relationship Students  TakenBy
    inverse Students::Takes;
  relationship Professors   TaughtBy
    inverse Professors::Teaches;
  relationship Departments   OfferedBy
    inverse Departments::Offer;
};

class Departments (key name) {
  attribute name;
  relationship Courses  Offer
    inverse Courses::OfferedBy;
  relationship Professors Works
    inverse Professors::WorksFor;
};
```

4.9.8

A relationship is its own inverse when for every attribute pair in the relationship, the inverse pair also exists. A relation with such a relationship is called symmetric in set theory. e.g. A relationship called SiblingOf in Person relation is its own inverse.

4.10.1
a)
Customers(ssNo,name,addr,phone)
Account(number,type,balance)
Owns(ssNo,accountNumber)

b)
Accounts(number,balance,type,owningCustomerssNo)
Customers(ssNo,name)
Addresses(ownerssNo,street,state,city)
Phones(ownerssNo,street,state,city,phonearea,phoneno)
We can remove Addresses relation since its attributes are a subset of relation Phones.



c)

```
Fans(name,colors)
RootedBy(fan_name,teamname)
Admires(fan_name,playername)
Players(name,teamname,is_captain)
Teams(name)--remove subset of teamcolor
Teamcolors(name,colorname)
Colors(colorname)
```

d)

```
class Person {
      attribute string name;
      relationship Person motherOf
            inverse Person::childrenOfFemale;
      relationship Person fatherOf
            inverse Person::childrenOfMale;
      relationship Set<Person> children
            inverse Person::parentsOf;
      relationship Set<Person> childrenOfFemale
            inverse Person::motherOf;
      relationship Set<Person> childrenOfMale
            inverse Person::fatherOf;
      relationship Set<Person> parentsOf
            inverse Person::children;
};
```

Person(name,mothername,fathername)
The children relationship is many-many but the information can be deduced from Person relation. Hence below relation is redundant.
Parent-Child(parent, child)

4.10.2
First consider each struct as if it were an atomic value i.e. key and value association pairs can be treated as two attributes. After applying normalization, the attributes can be replaced by the fields of the structs.

4.10.3
(a)

```
   Struct Card { string rank, string suit };
```

(b)

```
   class Hand {
          attribute Set theHand;
        };
```

(c)

```
   Hands(handId, rank, suit)
```

Each tuple corresponds to one card of a hand. HandId is required key to identify a hand.

(d) Hand contains an array of 5 elements

```
class PokerHand{attribute Array Hand(Card card1,Card card2,Card card3,Card
card4,Card card5)}
PokerHandS(handId,rank1,suit1,,rank2,suit2,rank3,suit3,rank4,suit4,rank5,suit5)
```

(e)

```
    class Deal {
       attribute Set <Struct PlayerHand { string Player, Hand theHand }
           > theDeal;
    }
```

(f) PokerDeal consist of a player and array of five card deal.

```
class PokerDeal{string Player,attribute Array Hand(Card card1,Card card2,Card
card3,Card card4,Card card5)}
```

(g) Above can similarly be represented by key player and a value consisting of five element array.

(h)
dealID is a key for Deals. Thus the relations for classes Deals and Hands are:

```
    Deals(dealID, player, handID)
    Hands(handID, rank, suit)
```

A simpler relation Deals below can also represents the classes:

```
    Deals(dealID, player, rank, suit)
```

(i)
The relation Deals(dealID,card) cannot identify the hand to which a card belongs. Also two attributes are required for a card;its rank and suit.

```
    Deals(dealID, handID, rank, suit)
```

4.10.4
(a)

```
    C(a, f, g)
```

(b)

```
    C(a, f, g, count)
```

(c)

```
    C(a, f, g, position)
```

(d)

```
    C(a, f, g, i, j)
```